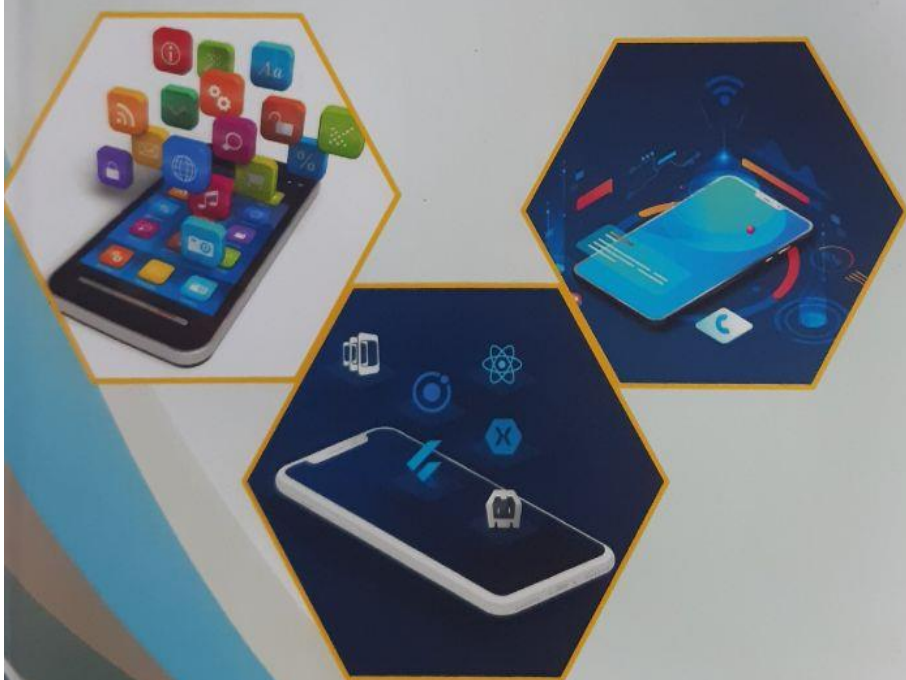


Doshanova M.Yu.

MOBIL ILOVALARINI ISHLAB CHIQISH



O'ZBEKISTON RESPUBLIKASI
RAQAMLI TEXNOLOGIYALAR VAZIRLIGI

MUHAMMAD AL-XORAZMIY NOMIDAGI TOSHKENT
AXBOROT TEXNOLOGIYALARI UNIVERSITETI

Doshanova M.Yu.

MOBIL ILOVALARINI ISHLAB CHIQISH

60610600–Dasturiy injiniring, 60611300–Axborot-kommunikatsiya
texnologiyalari sohasida kasb ta'limi, 60610500–Kompyuter injiniringi
("Kompyuter injiniringi", "AT-Servis", "Multimedia texnologiyalari")

*Muhammad Al-Xorazmiy nomidagi Toshkent axborot texnologiyalari
universiteti tomonidan o'quv qo'llanma sifatida tavsiya etilgan*

Toshkent
"METODIST NASHRIYOTI"
2024

UDK: 004.77(075.8)
BBK: 32.973ya7
D 75

Doshanova M.Yu.
Mobil ilovalarni ishlab chiqish. O'quv qo'llanma. – Toshkent:
"METODIST NASHRIYOTI", 2024. – 316 b.

O'quv qo'llanma Android va iOS operatsion tizimlarida ishlaydigan mobil qurilmalar uchun mobil ilovalarni ishlab chiqishga bag'ishlangan. Shuningdek, Android platformasi haqida asosiy ma'lumotlar keltirilgan. Android va iOS ilovalarini ishlab chiqish uchun zarur bo'lgan dasturiy vositalar tasvirlangan, ilovalarning asosiy komponentlari, asosiy vidjetlardan foydalanish, Java, XML va Swift dasturlash tillaridan foydalangan holda mobil ilovalarni ishlab chiqish, maxsus ilovalar yaratish, tasvirlarni yuklash uchun ilovalar yaratish, menyular, pastki menyular va kontent provayderlaridan foydalanish, xabar almashish va jo'natish uchun ilovalar, ma'lumotlar bazalari bilan ishlash, serverga ulanish, Google Maps xizmatlaridan foydalanish, foydalanuvchining joylashgan joyini aniqlash va ilovani internetda nashr qilish haqida ma'lumotlar keltirilgan.

O'quv qo'llanma oliy ta'lim muassasalarining 60610600–Dasturiy injiniring, 60611300–Axborot-kommunikatsiya texnologiyalari sohasida kasb ta'limi va 60610500–Kompyuter injiniringi ("Kompyuter injiniringi", "AT-Servis", "Multimedia texnologiyalari") ta'lim yo'nalishlari talabalari hamda mobil qurilmalar uchun dasturlash bilan shug'ullanuvchilar uchun mo'ljallangan.

Taqrizchilar:

D.T.Muxamediyeva- "Toshkent irrigatsiya va qishloq xo'jaligini mexanizatsiyalash muxandislari instituti" Milliy tadqiqot universiteti professori, t.f.d.

H.B.Xan - Muhammad al-Xorazmiy nomidagi TATU "Axborot texnologiyalarining dasturiy ta'minoti" kafedrasida katta o'qituvchisi.

Muhammad al-Xorazmiy nomidagi TATU Kengashining 2022-yil 22-dekabrda 5(727)-sonli qarori bilan nashr etishga tavsiya etildi. Ro'yxatga olish raqami №727-0058.

ISBN 978-9910-03-150-2

© Doshanova M.Yu., 2024.
© "METODIST NASHRIYOTI", 2024.

KIRISH

Har bir inson hayotning har bir sohasida maksimal qulaylikka erishishga bo'lgan katta intilishi xalqaro internetga ham ta'sir ko'rsatadi. Doim onlayn bo'lishni istagan foydalanuvchi kommunikator sifatida telefondan foydalanadi. Bu mobil internetning paydo bo'lishiga olib keldi. Udan uzoqda bo'lganingizda yoki sayohat va ish safarlarida noutbuk o'rniga planshet yoki shunga o'xshash mobil qurilma yordamida internetga ulanishingiz mumkin.

Faqatgina ushbu soha mutaxassislari tomonidan amalga oshiriladigan mobil ilovalarni ishlab chiqish aniq maqsad uchun mo'ljallangan. Hozirgi dasturlar sizga hamma joyda tarmoqqa ulanish imkonini beradi, boshqalari yo'nalishni ko'rsatadi, ba'zilari esa do'kon yoki kerakli mahsulotni topishda yordam beradi, uyda ovqatga buyurtma berishni amalga oshirish mumkin.

Darslik AndroidStudio, XCode integratsiyalashgan ishlab chiqish muhitlarida Java va Swift dasturlash tillaridan foydalangan holda Android va iOS OT uchun mobil ilovalarni ishlab chiqish asoslariga bag'ishlangan. Ular keng imkoniyatlarga ega va ularni o'rganish juda oson. Ushbu qo'llanma yordamida mobil ilovalar ishlab chiqishni o'rganish ancha oson va qiziqarli bo'ladi. Masalan, bir vaqtning o'zida bir nechta dasturlash tillarida (Java, Xml, Sql, Swift) mobil ilovalarni ishlab chiqishi mumkin.

Ushbu darslikning birinchi bobida asosiy e'tibor mobil ilovalarni ishlab chiqish asoslari va mobil ilovaning hayot sikliga, shuningdek, mobil ilovalarni ishlab chiqish va mobil ilovalarni ishlab chiqish muhiti tushunchalariga qaratilgan. Android va iOS operatsion tizimlari uchun mobil ilovalari Java va Dart dasturlash tillari ko'rib chiqiladi. Shuningdek, mobil ilovalarning hayot sikli, hayot siklini boshqarish, jarayonlar va ilovalar oqimini batafsil tavsiflaydi.

Ikkinchi bob mobil operatsion tizimlar turlariga, mobil operatsion tizimlar platformasi va arxitekturasi, mobil operatsion tizimlar platformasiga mos keladigan tillarga bag'ishlangan. Android platformasi va uning asoslari, Android ilovalarining asosiy turlari, mobil ilovalar arxitekturasi va ularning asosiy komponentlari, mobil ilovalarni ishlab chiqish vositalarini o'rnatish va sozlash, mobil ilovalarni ishlab chiqish uchun platformalararo tillar batafsil yoritilgan.

Darslikning uchinchi bobi Android operatsion tizimi uchun Java da mobil ilovalarni ishlab chiqishga qaratilgan. Java dasturlash tilining asosiy konstruksiyalari, ma'lumotlar turlari, maxsus sinflar va funksiyalar, sinflar

va obyektlar, konstruktorlar, initsializatorlar, shuningdek, Java tilida mobil ilovalarni ishlab chiqish usullari ko'rib chiqiladi.

Ma'lumotlar bazasi zamonaviy ilovalarda qanday rol o'ynashini hamma biladi. Yaxshi o'ylangan va to'g'ri tashkil qilingan ma'lumotlar bazasini yaratish muammosini hal qilish dinamik obyektlarni loyihalashda zarur hisoblanadi.

Darslikning to'rtinchi bobida ma'lumotlar bazalari va ma'lumotlar bazasini yaratish, so'rovlar yaratish, kontent-provayderlar va ulardan foydalanish, xabar almashish, mobil ilovalarda tarmoq uchun dasturlash, foydalanuvchining joylashgan joyini aniqlash, server bilan ishlash, geolokatsiyani aniqlash elementini yaratish bo'yicha materiallar berilgan.

Mobil qurilmalar sensorlar yordamida amalga oshiriladigan sensorli boshqaruv, imo-ishoralarni va boshqalarni aniqlashga yordam beradi.

Beshinchi bobda Androidning sensorli imkoniyatlari, smartfonlarning o'ziga xos xususiyatlari, sensorli boshqaruv, imo-ishoralarni aniqlash, imo-ishoralar to'plamini yaratish, sensorlarning turlari, sensorlar bilan ishlash, sensorlar va ulardan mobil telefonda foydalanish tamoyillarini aniqlash keltirilgan.

Oxirgi bob iOS uchun Swift tilida ilovalarni ishlab chiqishga bag'ishlangan. XCode dasturiy muhitida arxitektura va dizayn tamoyillari bo'yicha materiallar berilgan. Swift tilida ishlashning asosiy tamoyillari, o'zgaruvchilar va o'zgarmaslar, ma'lumotlar turlari, shartli tuzilmalar, operatorlar, sikllar, funksiyalar, sinflar va obyektlar, statik xususiyatlar va usullar, tuzilmalar, merosxo'rlik, polimorfizm, to'plamlar, massivlar, lug'atlar hisobga olinadi. Bir nechta darslarni o'tgandan so'ng, o'rganilgan funksiyalar bitta to'liq ilovaga birlashtiriladi.

1. MOBIL ILOVALARNI YARATISH ASOSLARI VA HAYOT SIKLI

1.1. Mobil ilovalarni yaratish uchun dasturlash muhitlari

1.1.1. Mobil ilovalarni ishlab chiqish dasturlari

Bugungi kunda mobil ilovalar yaratish uchun ko'plab dasturiy vositalar mavjud. Ular bir-biridan interfeysi, ishlatiladigan dasturlash tili, o'rnatilgan kutubxonalar, emulyatorlar va boshqalari bilan farqlanadi.

Eng keng tarqalgan mobil ilovalarni ishlab chiqish vositalari quyidagilardir: JDK, NetBeans IDE, Eclipse IDE, AndroidStudio, Intel XDK, IntelliJ IDEA, JDeveloper, BlueJ, Geany, Marmalade SDK va boshqalar.

Java Development Kit (JDK) Java dasturlashda ishlatiladigan uchta asosiy texnologiyalardan biridir. Ularga JVM (Java Virtual Machine) va JRE (Java Runtime Environment) ham kiradi. JVM Java dasturini bajarish uchun mo'ljallangan. JRE Java kodini ishga tushirish uchun asboblarga to'plami bo'lib, JVM ni yaratadi va ishga tushiradi. JRE Java dasturlarini osongina ishga tushirish uchun mustaqil komponent sifatida yoki JDK ning bir qismi sifatida ishlatilishi mumkin. JDK ishlab chiquvchilarga JVM va JRE orqali ishlaydigan dasturlarni yaratishga imkon beradi. JDK dasturiy ta'minotni ishlab chiqish uchun asboblarga to'plamidir. JDK JRE tomonidan talab qilinadi, chunki dasturlarni ishga tushirish ularning rivojlanishining ajralmas qismi hisoblanadi. Har bir JDK Java kompilyatoriga ega.

Java kompilyatori oddiy matn bo'lgan *.java fayllarini olib, ularni *.class bajariladigan fayllarga aylantira oladigan dasturdir.

NetBeans IDE - bu dasturiy ta'minot ishlab chiquvchilari uchun ochiq kodli bo'lib, Java, C/C++ va boshqa dinamik tillarda ish stoli, korporativ, mobil va veb-ilovalarni yaratish uchun to'liq vositani taqdim etadi. NetBeans IDE Windows, Linux, Solaris va Mac kabi turli platformalarda ishlashi mumkin.

NetBeans IDE ni o'rnatish va ishlatish oson bo'lib, qo'shimcha konfiguratsiya talab qilinmaydi. NetBeans IDE plaginlar va NetBeans platformasi asosidagi ilovalarni ishlab chiqish uchun kerak bo'lgan hamma narsani o'z ichiga oladi. NetBeans platformasi Swing ilovalarini ishlab chiqish uchun mo'ljallangan platformadir. Ilovalar boshqa modullarni dinamik ravishda yuklashi mumkin. Har qanday dastur foydalanuvchilarga dasturlar va modullar uchun yangi versiyalarni ishlaydigan ilovaga yuklab olish imkonini beradi.

Eclipse - Java dasturlash tili va C/C++, Python, PERL, Ruby va boshqalar kabi boshqa dasturlash tillaridan foydalangan holda ilovalarni ishlab chiqish uchun (Integrated Development Environment - IDE) mo'ljallangan. Eclipse platformasi Eclipse IDE uchun asos bo'lib xizmat qiladi, plaginlar bilan kengaytirilishi uchun mo'ljallangan. Eclipse dan mijoz ilovalari, IDE lar va boshqa vositalarni ishlab chiqish uchun foydalanish mumkin.

Eclipse mavjud bo'lgan har qanday dasturlash tili uchun IDE sifatida ishlatilishi mumkin. Java Development Tools (JDT) loyihasi Eclipse dan Java IDE sifatida foydalanish imkonini beruvchi plaginni taqdim etadi, PyDev - Eclipse dan Python IDE sifatida foydalanishga imkon beruvchi plagin, C/C++ Development Tools (CDT) - bu ruxsat beruvchi plagin. C/C++ yordamida ilovalarni ishlab chiqish uchun Eclipse dan foydalanish mumkin, Eclipse Scala plagini Eclipse ga Scala ilovalarini ishlab chiqish uchun IDE dan foydalanish imkonini beradi.

JDeveloper integratsiyalashgan dasturiy ta'minot ishlab chiqish muhitidir. Java, JavaScript, PHP, SQL, PL/SQL dasturlash tillarida hamda HTML, XML matnli belgilash tillarida ishlash imkoniyatini beradi. JDeveloper dizayn, kodlash, optimallashtirish, profil yaratish va joylashtirishdan tortib, dasturiy ta'minotni ishlab chiqishning butun hayotiy tsiklini qamrab oladi.

BlueJ Java dasturiy ta'minotini ishlab chiqish muhiti bo'lib, asosan ta'lim maqsadlarida foydalanish uchun mo'ljallangan, lekin kichik dasturlarni ishlab chiqish uchun ham mos keladi. BlueJ obyektga yo'naltirilgan dasturlashni o'rgatish uchun ishlab chiqilgan va uning dizayni boshqa rivojlanish muhitlaridan farq qiladi. Asosiy ekranda grafik tarzda ishlab chiqilayotgan (UML-ga o'xshash diagrammada) ilovaning sinf tuzilmasi ko'rsatilgan va obyektlarni interaktiv tarzda yaratish va sinab ko'rish mumkin. Ushbu interaktivlik aniq, sodda foydalanuvchi interfeysi bilan birgalikda siz loyihalashtirgan obyektlar bilan tajriba o'tkazishni osonlashtiradi.

Geany - bu GTK2 kutubxonasi yordamida yozilgan bepul dasturiy ta'minot ishlab chiqish muhitidir. Geany kompilyatorni o'z ichiga olmaydi. GNU Compiler bajariladigan kodni yaratish uchun ishlatiladi. Quyidagi operatsion tizimlar uchun har hil versiyalari mavjud: BSD, Linux, Mac OS X, Solaris va Windows.

IntelliJ IDEA - JetBrains tomonidan ishlab chiqilgan Java, JavaScript, Python dasturlash tillari uchun integratsiyalashgan dasturiy ta'minot ishlab chiqish muhitidir. Atrof-muhitni loyihalash dasturchilarning ish

unumdorligiga qaratilgan bo'lib, ularning diqqatini funksional vazifalarga qaratishga imkon beradi.

1.1.2. Android IDE

Android IDE kodni to'ldirish, real vaqtda xatolarni tekshirish va ilovangizni bir marta bosish bilan ishga tushirish, Android ilovalarini to'g'ridan-to'g'ri Android OS qurilmalarida ishlab chiqish mumkin:

- kodni bevosita smartfoningizda ko'rishingiz va tahrirlashingiz mumkin;

- Java/XML va C/C++ yordamida ishlab chiqishni qo'llab-quvvatlaydi;

- Eclipse loyihalariga to'liq mos keladi;

- ilovalarni professional ishlab chiqishni qo'llab-quvvatlaydi.

AndroidStudio - IntelliJ IDEA asosidagi Android operatsion tizimi uchun ishlab chiqish muhitidir. Dasturiy ta'minot 2013 yilda chiqarilgan va hali ham yangi versiyalari ishlab chiqilmoqda. AndroidStudio har bir yangi versiyasida ishlab chiquvchi funktsionallikni oshiradi, jarayonlarni optimallashtiradi va hokazo. AndroidStudio turli xil konfiguratsiyalarda allaqachon yozilgan yordamchi dasturlar va ilovalarning to'g'ri ishlashini tekshiradigan emulyator bilan birga keladi. Android Studio xususiyatlari:

- turli o'lchamli qurilmalarda bir vaqtning o'zida uning harakatini ko'rsatib, real vaqt rejimida ilovani tahrirlash imkonini beradi;

- turli ekran o'lchamlariga bir zumda o'tishi mumkin;

- tematik bo'limlar bilan optimallashtirish bo'yicha maslahatlar berish bo'limlari;

- beta-testlar bilan o'zaro ta'sir qilish vositalari;

- ilovalarni ishlab chiqish jarayonini tezlashtirish, uni samaraliroq qilish imkonini beradi.

Android SDK da ilova 4 ta komponentdan iborat:

1. *Activity* - grafik interfeysning asosiy komponenti (oyna yoki ekran shakliga o'xshash).

2. *Kontent provayderlar* ilovaning taqsimlangan ma'lumotlar to'plamini boshqaradi. Masalan, foydalanuvchi bilan bog'lanish ma'lumotlarini boshqaradigan Android kontent provayderi:

- ma'lumotlar fayl tizimida, SQLite ma'lumotlar bazasida, tarmoqda saqlanishi mumkin;

- boshqa ilovalarga, agar ular tegishli huquqlarga ega bo'lsa, so'rovlar yuborish yoki ma'lumotlarni o'zgartirish imkonini beradi.

3. *Intent*-lar - ilovalarning bir-biri bilan va operatsion tizim bilan ma'lumot almashishiga imkon beruvchi tizim xabarlarini:

- telefon qo'ng'irog'ini qabul qilish;
- sms-xabarlar kelishi;
- yangi *Activity* ni boshlanishi;

4. *GUI*ga ega bo'lmagan va fonda ishlaydigan ilovalar. Masalan:

- elektron pochta tekshirish;
- geoma'lumot olish.

1.1.3. Intel XDK

Intel XDK HTML5 ishlab chiqish muhiti bo'lib, HTML5 yordamida veb uchun ilovalarni yaratish va tarqatishda yordam beradi. Bu iOS, Android va Windows qurilmalari uchun HTML5 ilovalarini tezda yaratish imkonini beradi. Intel XDK xususiyatlari:

- platformalararo ilovalarni ishlab chiqishni osonlashtiradi;
- dasturiy ta'minotni yaratish, shuningdek, qurilma emulyatorini o'z ichiga oladi;
- Android, Apple iOS, Microsoft Windows 8 uchun ishlab chiqishni qo'llab-quvvatlaydi;
- HTML5 va JavaScript ishlab chiqish tillarini qo'llab-quvvatlaydi.

Intel XDK har qanday platformada ishlab chiqish imkonini beradi, chunki kompilyatsiya online bulutda amalga oshiriladi. Intel XDK mobil platformalar bilan cheklanmaydi.

Shuningdek Intel XDK, barcha HTML5 mobil platformalarida juda kam bo'lgan grafik muharriri o'z ichiga oladi. HTML komponentlaridan foydalanish WYSIWYG muharriridan foydalanish imkonini beradi. Intel XDK shuningdek, Bootstrap va jQuery Mobile kabi platformalarni qo'llab-quvvatlaydi. Intel XDK foydalanuvchi interfeysi komponentlari tezda dastur interfeysini yaratishga imkon beradi.

1.1.4. Marmelad SDK

Marmalade SDK - platformalararo ilovalarni ishlab chiqish uchun vositalar to'plami hisoblanadi. Mobil qurilmalar uchun ilovalarni ishlab chiqish, sinovdan o'tkazish va joylashtirish uchun zarur bo'lgan kutubxonalar, pluginlar, asboblari va shablonlar to'plamidan iborat. Mobil qurilmalar uchun o'yinlar va ilovalar yaratish uchun mo'ljallangan.

Marmalade SDK da emulyator mavjud va uni ma'lum bir qurilma modeli va ekran o'lchamlarini tanlash orqali dasturni sinab ko'rish mumkin.

Kamera, mikrofon, akselerometr, GPS moduli bilan o'zaro aloqani qo'llab-quvvatlaydi. Marmalade SDK, shuningdek, mobil ilovalar yaratish uchun C/C++ kodidan foydalanishga imkon beradi. Ishlab chiquvchilar bir xil kod bazasidan iloji boricha ko'proq platformalarda foydalanishlari mumkin, bu esa kattaroq loyihalar va ko'proq imkoniyatlarni beradi. Marmelad SDK sizga mavjud ishlarni, texnologiyani yoki uchinchi tomon vositalarini almashish, birlashtirish va qayta ishlatish imkonini beradi, shuningdek, u ochiq, moslashuvchan, katta va kichik loyihalarga teng darajada mos keladigan yuqori samarali o'yinni ishlab chiqish tizimini ta'minlaydi.

1.2. Mobil ilovalarni yaratish uchun dasturlash tillari

1.2.1. Android operatsion tizimi uchun Java tili

Java - bu Sun Microsystems tomonidan ishlab chiqilgan obyektga yo'naltirilgan dasturlash tili. Java ilovalari odatda maxsus bayt-kodga tarjima qilinadi, shuning uchun ular kompyuter arxitekturasidan qat'iy nazar har qanday Java virtual mashinasida ishlashi mumkin. Rasmiy chiqish sanasi - 1995 yil 23 may. Java dasturlari bayt kodga tarjima qilinadi va Java Virtual Machine (JVM) tomonidan bajariladi, bu dastur bayt kodini qayta ishlovchi va ko'rsatmalarni tarjimon sifatida apparatga uzatadi.

Dasturlarni bajarishning bunday usulining afzalligi bayt-kodning operatsion tizim va apparat ta'minotidan to'liq mustaqilligi bo'lib, u Java ilovalarini mos keladigan virtual mashina mavjud bo'lgan har qanday qurilmada ishga tushirish imkonini beradi. Java texnologiyasining yana bir muhim xususiyati moslashuvchan xavfsizlik tizimi bo'lib, unda dasturning bajarilishi virtual mashina tomonidan to'liq nazorat qilinadi. Dasturning belgilangan ruxsatlaridan oshib ketadigan har qanday operatsiya (masalan, ma'lumotlarga ruxsatsiz kirishga urinish yoki boshqa kompyuterga ulanish) darhol to'xtatilishiga olib keladi. Ko'pincha, virtual mashina konsepsiyasining kamchiliklari ishlashning pasayishini o'z ichiga oladi. Bir qator yangilanishlar Java dasturlari tezligini biroz oshirdi:

- sinf versiyalarini mashina kodida saqlash imkoniyatiga ega bo'lgan holda dasturning ishlashi paytida (JIT texnologiyasi) bayt kodini bevosita mashina kodiga o'tkazish texnologiyasini qo'llash;
- standart kutubxonalarda platformaga xos koddan (nativ kod) keng foydalanish;
- tezashtirilgan bayt-kodni qayta ishlashni ta'minlovchi apparat (masalan, ba'zi ARM protsessorlari tomonidan qo'llab-quvvatlanadigan Jazelle texnologiyasi).

Java da bir nechta asosiy texnologiyalar oilalari mavjud:

- Java SE - Java Standard Edition, Java tilining asosiy kompilyatorlarini, API, Java Runtime Environmentni o'z ichiga oladi; birinchi navbatda ishchi stoli tizimlari uchun maxsus ilovalar yaratish uchun javob beradi;

- Java EE - Java Enterprise Edition, korporativ darajadagi dasturiy ta'minotni yaratish uchun texnik xususiyatlar to'plamidir;

- Java ME - Java Micro Edition, mobil telefonlar, PDAlar, o'rnatilgan tizimlar kabi ishlov berish quvvati cheklangan qurilmalarda foydalanish uchun mo'ljallangan;

- JavaFX - Rich Client Platformasi sifatida Java evolyutsiyasining navbatdagi bosqichi bo'lgan texnologiya; korporativ ilovalar va korxonalar uchun grafik interfeyslarni yaratish uchun mo'ljallangan;

- Java kartasi - texnologiya smart-kartalarda va juda cheklangan xotira va ishlov berish imkoniyatlariga ega boshqa qurilmalarda ishlaydigan ilovalar uchun xavfsiz muhitni ta'minlaydi.

Microsoft JVM (MSJVM) o'zining dasturini ishlab chiqdi va Windows 98 dan boshlab turli xil operatsion tizimlarga kiritgan.

Microsoft JVM Sun Java dan sezilarli farqlarga ega edi, bu ko'p jihatdan turli platformalar o'rtasida dasturlarni ko'chirishning asosiy konsepsiyasini buzdi:

- masofaviy chaqiruv API (RMI) ni qo'llab-quvvatlamaslik;

- JNI texnologiyasini qo'llab-quvvatlamaslik;

- Faqat Windows platformasida ishlaydigan Java va DCOM integratsiya vositalari kabi nostandart kengaytmalarning mavjudligi.

Java tili Android operatsion tizimi uchun mobil ilovalarni yaratishda foydalaniladi. Bunday holda, dasturlar Dalvik virtual mashinasi tomonidan foydalanish uchun nostandart baytkodlarga kompilyatsiya qilinadi. Ushbu kompilyatsiya uchun qo'shimcha vosita, ya'ni Google tomonidan ishlab chiqilgan Software Development Kit ishlatiladi. Ilovalarni ishlab chiqish Android Studio, NetBeans, Eclipse da Android Development Tools (ADT) plagini yoki IntelliJ IDEA yordamida amalga oshirilishi mumkin. JDK versiyasi 8.0 yoki undan yuqori bo'lishi kerak.

2014-yil 8-dekabrda Android Studio Google tomonidan Android OS uchun rasmiy ishlab chiqish muhiti sifatida tan olingan. Ba'zi platformalar Java uchun apparat bajarilishini qo'llab-quvvatlaydi. Masalan, dasturiy ta'minot JVM o'miga apparatda Java kodini ishlatadigan mikrokontrollerlar

va Jazelle opsiyasi orqali Java bayt-kodini bajarishni qo'llab-quvvatlaydigan ARM ga asoslangan protsessorlar. Asosiy xususiyatlar:

- xotirani avtomatik boshqarish;

- istisno vaziyatlarni hal qilish uchun ilg'or imkoniyatlar;

- kirish-chiqish filtrlash vositalarining boy to'plami;

- standart to'plamlar: massiv, ro'yxat, stek va boshqalar;

- Tarmoq ilovalarini yaratish uchun oddiy vositalarning mavjudligi (jumladan, RMI protokolidan foydalanadiganlar);

- HTTP so'rovlarini amalga oshirish va javoblarni qayta ishlash imkonini beruvchi sinflarning mavjudligi;

- ko'p oqimli ilovalar yaratish uchun o'rnatilgan til vositalari;

- ma'lumotlar bazalariga yagona kirish;

- individual SQL so'rovlari darajasida - JDBC, SQLJ asosida;

- ma'lumotlar bazasida saqlash imkoniyatiga ega bo'lgan obyektlar tushunchasi darajasida - Java Data Objects va Java Persistence API asosida;

- umumlashtirishni qo'llab-quvvatlash (1.5 versiyasidan boshlab);

- dasturlarning parallel bajarilishi

1.2.2. Android operatsion tizimi uchun Dart tili

Dart - Google tomonidan yaratilgan dasturlash tili hisoblanadi. Dart JavaScript ni almashtirish uchun muqobil til sifatida ishlatiladi. Tilni ishlab chiquvchilardan biri Mark Miller JavaScript ni tuzatib bo'lmaydigan asosiy kamchiliklarga ega ekanligini, shuning uchun Dart yaratilganligini ta'kidlagan. 2011 yil 10 oktyabrda Google'da Dart tilining rasmiy taqdimoti bo'lib o'tdi.

Dart - bu Google kompaniyasining umumiy maqsadli dasturlash tili bo'lib, u asosan veb-ilovalar (mijoz tomoni va server tomoni) va mobil ilovalarni ishlab chiqish uchun mo'ljallangan. Bu shuni anglatadiki, bir xil Dart dasturi turli platformalar - Windows, Android, iOS uchun kompilyatsiya qilinishi mumkin. Dart - obyektga yo'naltirilgan dasturlash tili hisoblanadi. Dart dasturida ishlatiladigan barcha qiymatlar obyektlarni ifodalaydi. Dart o'zining rivojlanishida Smalltalk, Java, JavaScript kabi oldingi tillardan olingan. Uning sintaksisi boshqa tillarnikiga o'xshaydi.

Dart jadal rivojlanmoqda va joriy versiyasi 2.12. Dart bilan ishlash uchun Dart SDK ni o'rnatish lozim. Buning uchun <https://dart.dev/tools/sdk/archive> dan SDK zip arxivini yuklab olish va uni qattiq diskga o'rnatish lozim. Yuklab olish sahifasida Windows, Linux, MacOS uchun paketlar mavjud. Turli ishlab chiquvchilar tuzilmalari ham

mavjud. Har qanday Dart ilovasi asosiy funksiyaga ega bo'lishi kerak. Bu funksiya void tipiga ega va hech qanday parametr qabul qilmaydi, shuning uchun funksiya nomidan keyin bo'sh qavslar mavjud.

Funksiya tanasi figurali qavslarga joylashtiriladi. Dasturni har safar ishga tushirganingizda dart.exe dasturiga to'liq yo'lni kiritmaslik uchun yordamchi dasturga yo'lni muhit o'zgaruvchilariga qo'shish mumkin.

Dasturni ishga tushirish uchun dart.exe dasturidan foydalanish mumkin, lekin dart.exe ni istalgan vaqtda ishga tushirish va xuddi shu operatsion tizimga ega boshqa kompyuterga o'tkazish uchun bajariladigan fayl yaratish mumkin. Buning uchun SDK boshqa yordamchi dasturga ega - *dart2native.exe*, bu dasturning mahalliy bajariladigan faylini kompilyatsiya qilish imkonini beradi. U kompilyatsiya qilinadigan manba faylni parametr sifatida oladi.

1.3. Mobil ilovalarning hayotiy sikli

1.3.1. Ilovaning hayotiy sikli

Activity – foydalanuvchining grafik interfeysi joylashgan oyna. *Activity* oynasi odatda qurilmaning butun ekranini egallaydi, lekin bunda yarim shaffof yoki suzuvchi dialog oynalarni yaratish mumkin. Mobil ilovalar odatda ko'p oynali bo'lib, ular har bir oyna uchun bir nechta *Activity*-ni o'z ichiga oladi. *Activity*-lardan biri "asosiy" sifatida belgilanadi va foydalanuvchi dasturni birinchi marta ishga tushirgandagi ko'rinish hisoblanadi (1.1-rasm).

Har bir ilova ekrani *Activity* sinfining vorisi hisoblanadi. *Activity* yaratish uchun *Activity* sinfini to'g'ridan-to'g'ri yoki uning avlodlaridan birortasi orqali meros qilib oladigan sinf yaratish kerak. *Activity* sinfidan ning hayotiy siklini boshqarish uchun tizim tomonidan chaqirilgan barcha usullarni amalga oshirish lozim. Bunday usullardan ettitasi mavjud:

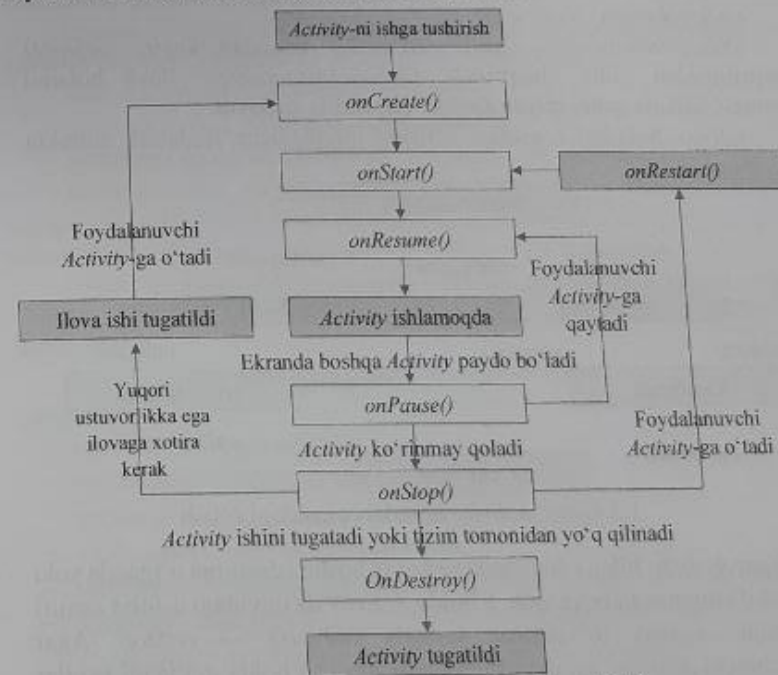
onCreate() – bu *Activity* yaratilganda tizim tomonidan chaqiriladigan usul. Usulni amalga oshirishda *Activity*-ning asosiy komponentlarini ishga tushirish kerak va ko'p hollarda tegishli *XML* faylini (*layoutfile*) kiritish uchun *setContentView()* usulini chaqirish kerak. *onCreate()* usulidan keyin *onStart()* usuli har doim chaqiriladi.

onRestart() – bu to'xtatilgan *Activity* ni ishga tushirish kerak bo'lganda tizim tomonidan chaqiriladigan usul. Ushbu usuldan keyin *onStart()* usuli har doim chaqiriladi.

onStart() – bu *Activity* foydalanuvchiga ko'rinishidan oldin tizim tomonidan chaqiriladigan usul. Ushbu usuldan so'ng *onResume()* chaqiriladi.

onResume() – bu *Activity* foydalanuvchi bilan ishlashni boshlashdan oldin tizim tomonidan chaqiriladigan usul. Ushbu usuldan keyin har doim *onPause()* chaqiriladi.

onPause() – bu *Activity* faollikni yo'qotganda tizim tomonidan chaqiriladigan usul. Ushbu usulda joriy seansdan tashqari saqlanishi kerak bo'lgan barcha o'zgarishlarni amalga oshirish kerak. Ushbu usuldan so'ng, agar *Activity* oldingi o'ringa qaytsa, *onResume()* chaqiriladi, agar *Activity* foydalanuvchidan yashirilsa, *onStop()* chaqiriladi.



1.1-rasm. *Activity* ning hayotiy sikli

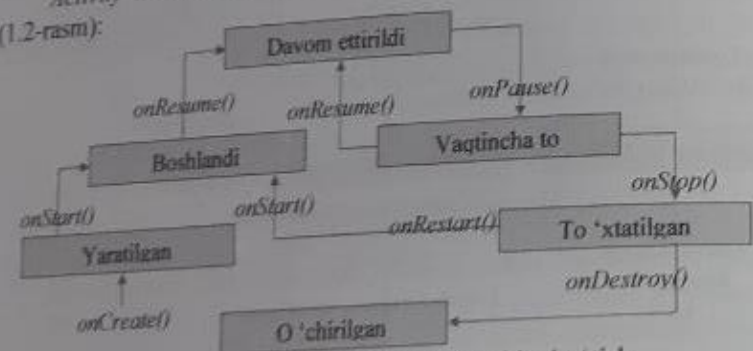
onStop() – bu *Activity* foydalanuvchi uchun ko'rinmas holga kelganda tizim tomonidan chaqiriladigan usul. Ushbu usuldan so'ng, agar *Activity* foydalanuvchi o'zaro ta'siriga qaytsa, *onRestart()* yoki faollik yo'q qilinsa, *onDestroy()* chaqiriladi.

onDestroy() – bu *Activity* yo‘q qilinishidan oldin tizim tomonidan chaqiriladigan usul. Ushbu usul *Activity* tugaganda yoki tizim resurslarini chiqarish uchun *Activity*-ni yo‘q qilganda chaqiriladi. Bu ikki ssenariyni *isFinishing()* usuli yordamida farqlash mumkin.

onRestoreInstanceState usuli
onStart() usuli tugallanganidan so‘ng, parametr sifatida uzatilgan *Bundle* obyektidan saqlangan holatni tiklaydigan *onRestoreInstanceState* usuli chaqiriladi. Ammo bu usul faqat *Bundle* null bo‘lmaganda va avval saqlangan holatni o‘z ichiga olgan holda chaqiriladi. Ilova birinchi marta ishga tushirilganda, *Bundle* obyektini null bo‘ladi, shuning uchun *onRestoreInstanceState* usuli chaqirilmaydi.

onSaveInstanceState usuli
onSaveInstanceState usuli *onPause()* usulidan keyin, *onStop()* chaqirilishidan oldin chaqiriladi. *onSaveInstanceState* ilova holatini parametr sifatida qabul qilgan *Bundle* obyektida saqlaydi.

Activity holatlari orasidagi o‘tishni quyidagicha ifodalash mumkin (1.2-rasm):



1.2-rasm. *Activity* holatlari orasidagi o‘tish

Agar *Activity* bilan ishlaganda va keyin boshqa dasturga o‘tganda yoki Bosh sahifa tugmasini bosganda, u holda *Activity* da quyidagi usullar zanjiri chaqiriladi: *Activity* to‘xtatilgan holatda *onPause* → *onStop*. Agar foydalanuvchi *Activity* ga qaytishga qaror qilsa, u holda quyidagi usullar zanjiri chaqiriladi: *onRestart* → *onStart* → *onResume*.

Yana bir holat, agar foydalanuvchi *Orqaga* tugmachasini bossa, u holda quyidagi *onPause* → *onStop* → *onDestroy* zanjiri chaqiriladi. Natijada, *Activity* yo‘q qilinadi. Agar to‘satdan vazifalar menejeri orqali yoki ilovani qayta ochish orqali *Activity* ga qaytish lozim bo‘lsa, u holda

Activity *onCreate* → *onStart* → *onResume* usullari orqali qayta yaratiladi. Quyida ba‘zi hollar uchun misollar keltirilgan:

- Ilovani ishga tushirish: *onCreate()* → *onStart()* → *onResume()*;
- Ilovadan chiqish uchun orqaga tugmasi bosilganda: *onPause()* → *onStop()* → *onDestroy()*;
- Bosh sahifa tugmasi bosilganda: *onPause()* → *onStop()*;
- Ilova yaqinda ochilgan ilovalar ro‘yxatidan yoki belgi orqali ishga tushirilganda “Home” tugmasini bosgandan so‘ng: *onRestart()* → *onStart()* → *onResume()*;
- Xabarlar maydonidan boshqa ilova ishga tushirilganda yoki Sozlamalar ilovasi ochilganda: *onPause()* → *onStop()*;
- Boshqa ilovadagi yoki Sozlamalardagi *Orqaga* tugmasi bosilsa, ilova yana ko‘ringanda: *onRestart()* → *onStart()* → *onResume()*;
- Muloqot oynasi ochilganda: *onPause()*;
- Muloqot oynasi yopilganda: *onResume()*;
- Telefonga qo‘ng‘iroq kelganda: *onPause()* → *onResume()*;
- Foydalanuvchi qo‘ng‘iroqqa javob berganda: *onPause()*; suhbat tugaganda: *onResume()*; telefon ekrani o‘chganda: *onPause()* → *onStop()*; ekran yana yoqilganda: *onRestart()* → *onStart()* → *onResume()*.

Mobil qurilmani aylantirganda, *Activity* turli holatlar zanjiri orqali o‘tadi: *onPause()* → *onStop()* → *onDestroy()* → *onCreate()* → *onStart()* → *onResume()*.

Mobil qurilmadan qo‘ng‘iroq qilinganda usullar quyidagi tartibda ishlaydi:

- *onCreate()* – *onStart()* dan keyin;
- *onRestart()* – *onStart()* dan keyin;
- *onStart()* – *onResume()* yoki *onStop()* dan keyin;
- *onResume()* – *onPause()* dan keyin;
- *onPause()* – *onResume()* yoki *onStop()* dan keyin;
- *onStop()* – *onRestart()* yoki *onDestroy()* dan keyin;
- *onDestroy()* dan keyin – hech qaysi usul chaqarilmaydi.

Activity da ishlatiladigan servislar ham o‘z hayot sikliga ega. Servislarning hayot siklida quyidagi usullar ishlatiladi:

- *onCreate()* usuli. *startService()* usuli bilan yaratilganda bir marta chaqiriladi. *startService()* usulini qayta ishlatganda chaqirilmaydi.
- *onStartCommand()* usuli. Har safar *startService()* usuli yordamida yuborilgan buyruqni olganida chaqiriladi. Usul ma‘lum bir rejimda xizmatni boshlaydigan bayroqni qaytarishi mumkin.

- `onBind()` usuli. `BindService()` usuli yordamida mijozga xizmat (`Activity`) bog'langanda (xizmat mijoz murojaat qilganda ishlaydi) chaqiriladi.

- `onRebind()` usuli. Servis mijozga qaytarilganda chaqiriladi.

- `onUnbind()` usuli. Servis mijozdan uzilganda chaqiriladi.

- `onDestroy()` usuli. Servis yo'q qilganda chaqiriladi.

`onStartCommand()` usuli hayotiy davri, agar u kutilmaganda to'xtatilgan bo'lsa, masalan, xotira etishmasligi tufayli tizim tomonidan xizmatning harakatini tavsiflovchi bayroqni qaytarishi mumkin:

- `START_STICKY` bayro'gi. Servis tizim tomonidan qayta ishga tushiriladi va ishlashda davom etadi.

- `START_REDELIVER_INTENT` bayro'gi. Servis tizim tomonidan qayta ishga tushiriladi va xizmatni ishga tushirishda `startService()` usuliga o'tkazilgan `Intent` ni qayta oladi.

- `START_NOT_STICKY` bayro'gi. Servis to'xtatilgan holatda qoladi va tizim tomonidan qayta ishga tushmaydi.

- `STOP_FOREGROUND_REMOVE` bayro'gi. Oldingi rejimni bekor qilish va servis bog'langan bildirishnomani o'chirish uchun ishlatiladi.

1.3.2. Hayotiy siklni boshqarish

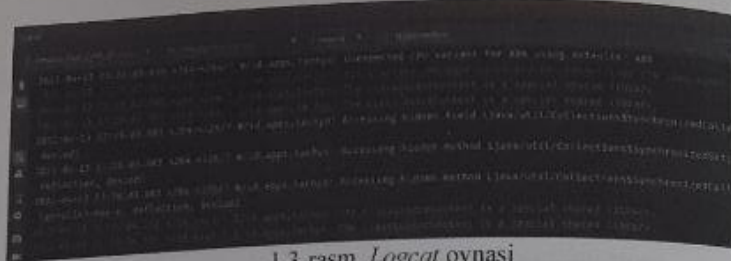
Tegishli usullarni bekor qilish orqali ushbu hayot sikli hodisalarini boshqarish mumkin. Buning uchun yangi proyektni yaratib, unda `MainActivity` sinfi ochiladi va uni quyidagicha o'zgartirish mumkin:

```
package com.example.viewapplication;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.util.Log;

public class MainActivity extends AppCompatActivity {
    private final static String TAG = "MainActivity";
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Log.d(TAG, "onCreate");
    }
    @Override
    protected void onDestroy() {
        super.onDestroy();
        Log.d(TAG, "onDestroy");
    }
    @Override
    protected void onStop() {
```

```
        super.onStop();
        Log.d(TAG, "onStop");
    }
    @Override
    protected void onStart() {
        super.onStart();
        Log.d(TAG, "onStart");
    }
    @Override
    protected void onPause() {
        super.onPause();
        Log.d(TAG, "onPause");
    }
    @Override
    protected void onResume() {
        super.onResume();
        Log.d(TAG, "onResume");
    }
    @Override
    protected void onRestart() {
        super.onRestart();
        Log.d(TAG, "onRestart");
    }
    @Override
    protected void onSaveInstanceState(Bundle outState) {
        super.onSaveInstanceState(outState);
        Log.d(TAG, "onSaveInstanceState");
    }
    @Override
    protected void onRestoreInstanceState(Bundle savedInstanceState) {
        super.onRestoreInstanceState(savedInstanceState);
        Log.d(TAG, "onRestoreInstanceState");
    }
}
```

Voqealarni qayd qilish uchun bu yerda `android.util.Log` sinfidan foydalaniladi. Misolda hayot siklining barcha asosiy usullari ko'rib chiqiladi. Barcha qayta ishlash `Log.d()` usulini chaqirish uchun qaratilgan, yuqoridagi dasturda `TAG` - tasodifiy satr qiymati va `Android Monitor` oynasida `Android Studio` pastki qismidagi `Logcat` konsolida ko'rsatiladigan ma'lumot sifatida ishlaydi. Agar ushbu konsol yashirin bo'lsa, uni `View` → `Tool Windows` → `Android Monitor` menyusini orqali ochish mumkin. Ilovani ishga tushirganda, `Logcat` oynasida disk ma'lumotlarini ko'rish mumkin, u `Activity`-ning hayot sikli usullarida aniqlangan (1.3-rasm):



1.3-rasm. Logcat oynasi

Foydalanuvchi interfeysi ekranlarini yaratishda *Activity* klassi meros qilib olinadi va foydalanuvchi bilan muloqot qilish uchun *View*-lar ishlatiladi. Har bir *Activity* mobil ilova foydalanuvchilariga ko'rsatishi mumkin bo'lgan ekrandir. Yaratilayotgan dastur qanchalik murakkab bo'lsa, shuncha ko'p ekranlar (*Activity*) talab qilinadi. Ilovani yaratishda, hech bo'lmaganda, ilovaning foydalanuvchi interfeysi uchun asos bo'lgan boshlang'ich (asosiy) ekran kerak bo'ladi. Agar kerak bo'lsa, ushbu interfeys ma'lumotlarni kiritish, uni ko'rsatish va qo'shimcha funksiyalarni ta'minlash uchun mo'ljallangan ikkilamchi *Activity*-lar bilan to'ldiriladi. Yangi *Activity*-ni ishga tushirish (yoki undan qaytish) UI (User Interfeys) ekranlari o'rtasida "ko'chib o'tishga" olib keladi. Aksariyat harakatlar to'liq ekran maydonidan foydalanish uchun mo'ljallangan, lekin yarim shaffof yoki suzuvchi dialog oynalarini ham yaratish mumkin.

Yangi *Activity* yaratish uchun *Activity* sinfi meros qilib olinadi. Sinfnı amalga oshirishda foydalanuvchi interfeysini belgilash va kerakli funksiyalarni amalga oshirish kerak. Yangi *Activity* uchun asosiy kod quyida ko'rsatilgan:

```
package com.example.myapplication;
import android.app.Activity;
import android.os.Bundle;

public class MyActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
    }
}
```

Asosiy *Activity* sinfi bo'sh ekran bo'lib, u unchalik foydali emas, shuning uchun birinchi *Views* (*View*) va *Layout* yordamida foydalanuvchi interfeysini yaratish lozim hisoblanadi.

Views ma'lumotni aks ettiruvchi va foydalanuvchi o'zaro ta'sirini ta'minlaydigan UI elementlaridir. Android ilovaning foydalanuvchi

interfeysini yaratish uchun ular ichida bir nechta *Views*-ni o'z ichiga olishi mumkin bo'lgan *View* guruhlarini deb ataladigan bir nechta *Layout* sinflarini taqdim etadi.

Activity-ga foydalanuvchi interfeysini belgilash uchun *onCreate()* qayta ishlovchisi ichida *setContentView()* usuli qo'llaniladi:

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    TextView textView = new TextView(this);
    setContentView(textView);
}
```

Ushbu misolda *Activity* uchun UI *TextView* sinfining obyektı hisoblanadi. Haqiqiy ilovalarni yaratishda ko'pincha koddan ajratilgan dastur resurslaridan foydalanadigan dizayn yondashuvi qo'llaniladi. Ushbu yondashuv dasturning ishlash shartlariga bog'liq bo'lmagan yuqori sifatli UI amalga oshirilishini ta'minlaydigan ilovalarni yaratishga imkon beradi: ilovalar foydalanuvchilarga qulay til interfeysini taklif qiladi (lokalizatsiyaga qarab), ekran o'lchamiga bog'liq va hokazo.

Eng muhimi, ilovaning yangi sharoitlarga bunday moslashuvi dastur kodida hech qanday o'zgarishlarni talab qilmaydi, faqat kerakli resurslarni (tasvirlar, satrlar va boshqalar) taqdim etish kerak. AndroidStudio-da bunga quyidagicha ko'rsatilgan standart yondashuv mavjud:

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
}
```

Ilovada *Activity*-dan foydalanish uchun uni *<application>* tuguniga *<Activity>* elementini qo'shish orqali *Manifest* faylida ro'yxatdan o'tkazish kerak, aks holda undan foydalanib bo'lmaydi. Quyida *MyActivity*-da *Activity* uchun *<Activity>* elementini yaratish ko'rsatilgan:

```
<Activity android:label="@string/app_name"
    android:name=".MyActivity">
</Activity>
```

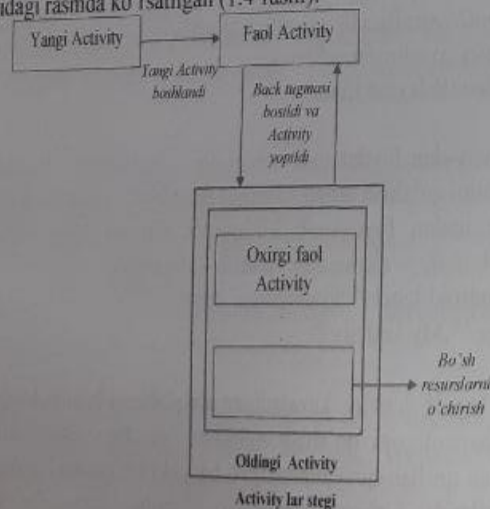
<Activity> tegida *Activity* kuzatadigan maqsadlarni belgilash uchun *<intent-filter>* elementlarini qo'shish mumkin. Har bir *<intent-filter>* *Activity* tomonidan qo'llab-quvvatlanadigan bir yoki bir nechta harakatlar va toifalarni belgilaydi. *Activity*-ga faqat asosiy dastur ishga tushgandan keyin kirish mumkin bo'ladi, agar uning asosiy amal qilish *manifest* faylida

<intent-filter> bo'lsa va misolda ko'rsatilganidek, LAUNCHER kategoriyasi bo'lsa ishlatiladi:

```
<Activity android:label="@string/app_name"
android:name=".MyActivity">
<intent-filter>
<action android:name="android.intent.action.MAIN" />
<category android:name="android.intent.category.LAUNCHER" />
</intent-filter>
</Activity>
```

Resurslarni to'g'ri boshqaradigan va foydalanuvchilarga qulay interfeysni ta'minlaydigan ilovalarni yaratish uchun Activity-ning hayot siklini yaxshi tushunish juda muhimdir. Buning sababi, Android ilovalari o'zlarining hayot siklini boshqara olmasligi, operatsion tizimning o'zi barcha jarayonlarni va natijada ular ichidagi Activity-larni boshqaradi. Bunda Activity holati operatsion tizimga ushbu ilova Activity uchun asosiy ustuvorlikni aniqlashga yordam beradi. Ilovaning ustuvorligi uning ishi tizim tomonidan to'xtatilishi ehtimoliga ta'sir qiladi.

Har bir Activity-ning holati uning hozirda bajarilayotgan Activity-lar stekidagi o'rni bilan belgilanadi. Yangi Activity ishga tushirilganda, u ko'rsatadigan ekran stekning yuqori qismiga suriladi. Agar foydalanuvchi orqaga tugmasini bossa yoki Activity boshqa yo'l bilan yopilsa, asosiy Activity stekning yuqori qismiga ko'chiriladi va faol bo'ladi. Ushbu jarayon quyidagi rasmda ko'rsatilgan (1.4-rasm):



1.4-rasm. Activity holati

Ilovaning ustuvorligiga uning eng yuqori ustuvorligi ta'sir qiladi. Operatsion tizim xotira menejeri resurslarni bo'shatish uchun qaysi dasturni yopishga qaror qilganda, u ilovaning ustuvorligini aniqlash uchun stekdagi Activity-ning o'rni haqidagi ma'lumotlarni hisobga oladi.

1.3.3. Activity holati

Activity to'rtta mumkin bo'lgan holatdan birida bo'lishi mumkin:

- *faol*. Activity oldingi planda (stekning tepasida) va foydalanuvchi bilan muloqot qilish imkoniyatiga ega. Android har qanday holatda ham uning ishlashini ta'minlashga harakat qiladi, agar kerak bo'lsa, kerakli resurslarni ta'minlash uchun stekdagi boshqa Activity-larni to'xtatadi. Agar boshqa Activity birinchi o'ringa chiqsa, ushbu Activity ishi butunlay to'xtatiladi yoki vaqtincha to'xtatiladi.

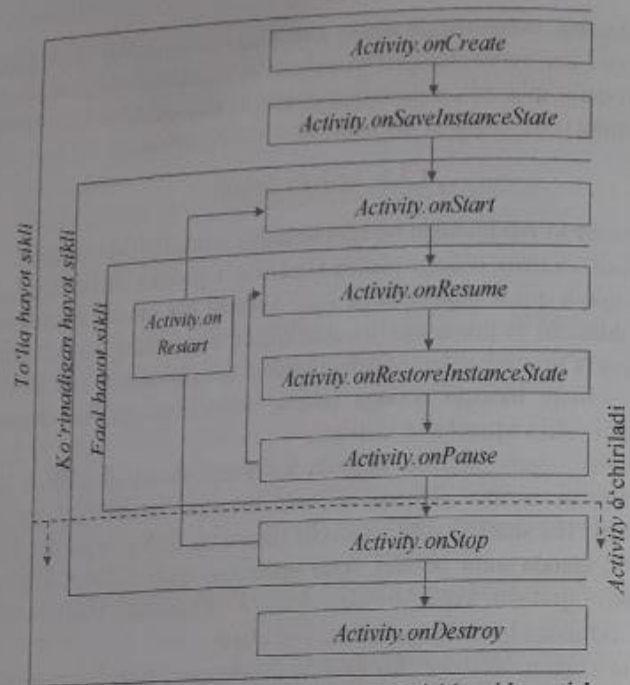
- *to'xtatilgan*. Activity ekranda ko'rinishi mumkin, lekin u bilan foydalanuvchi o'zaro aloqada bo'lishi mumkin emas; bu vaqtda u to'xtatiladi. Bu shaffof yoki suzuvchi (masalan, dialog) oynalar oldingi planda bo'lganda sodir bo'ladi. Agar operatsion tizim oldingi Activity-ga resurslarni ajratishi kerak bo'lsa, Activity to'xtatilishi mumkin. Agar Activity ekrandan butunlay yo'qolsa, u to'xtaydi.

- *to'xtagan*. Activity ko'rinmas, u xotirada yashaydi, uning holati haqida ma'lumotni saqlaydi. Agar tizim boshqa biror narsa uchun xotiraga muhtoj bo'lsa, bunday Activity muddatidan oldin tugatish uchun nomzod bo'ladi. Activity to'xtatilganda, ishlab chiquvchi ma'lumotlarni va foydalanuvchi interfeysining joriy holatini (kiritish maydonlarining holati, kursor o'rni va boshqalar) saqlashi muhimdir. Activity tugatilsa yoki yopilsa, u faol emas bo'ladi.

- *faol emas*. Activity tugallanganda va u boshlanishidan oldin Activity faolsiz holatda bo'ladi. Bunday harakatlar stekdan o'chiriladi va ulardan foydalanish mumkin bo'lishi uchun qayta boshlanishi kerak.

Ilova holatini o'zgartirish deterministik bo'lmagan jarayon bo'lib, faqat Android xotira menejeri tomonidan boshqariladi. Agar kerak bo'lsa, Android birinchi navbatda faol bo'lmagan Activity-larni o'z ichiga olgan ilovalarni yopadi, keyin to'xtatiladi va o'ta og'ir hollarda to'xtatiladi.

To'liq huquqli dastur interfeysini ta'minlash uchun uning holatidagi o'zgarishlar foydalanuvchi uchun sezilmasligi kerak. Uning holatini to'xtatilganda yoki faollikdan faolga o'zgartirganda, Activity tashqi tomondan o'zgarishsizligi kerak.



1.5-rasm. Activity holatidagi o'zgarishlarni kuzatish

Activity to'xtatilganda ishlab chiquvchi Activity holati saqlanishini ta'minlashi kerak, shunda Activity birinchi o'ringa chiqqanda uni qayta tiklash mumkin. Buning uchun Activity sinfida hodisalarni qayta ishlash vositalari mavjud bo'lib, ularning ishlab chiquvchiga Activity holatidagi o'zgarishlarni kuzatish imkonini beradi. Activity sinfi hodisalarini qayta ishlash moslamalari tegishli Activity obyektini holatidagi o'zgarishlarni uning butun hayotiy sikli davomida kuzatish imkonini beradi (1.5-rasm).

Quyida hodisani qayta ishlash usullari uchun dastur kodi misol sifatida keltirilgan:

```
package com.example.myapplication;
import android.app.Activity;
import android.os.Bundle;
public class MyActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
    }
}
```

```
@Override
public void onRestoreInstanceState(Bundle savedInstanceState) {
    super.onRestoreInstanceState(savedInstanceState);
}
@Override
public void onRestart(){ super.onRestart(); }
@Override
public void onStart(){ super.onStart(); }
@Override
public void onResume(){ super.onResume(); }
@Override
public void onSaveInstanceState(Bundle savedInstanceState) {
    super.onSaveInstanceState(savedInstanceState);
}
@Override
public void onPause(){ super.onPause(); }
@Override
public void onStop(){ super.onStop(); }
@Override
public void onDestroy(){ super.onDestroy(); }
}
```

1.3.4. Jarayonlar va oqimlar

Boshqa komponentlar ishlamay qolganda dastur komponenti ishga tushganda, Android operatsion tizimi dastur uchun yangi Linux jarayonini bitta bajarilish oqimi bilan boshlaydi. Odatiy bo'lib, bitta dasturning barcha komponentlari bir xil jarayon va oqimda ishlaydi ("asosiy tarmoq" deb ataladi). Agar dastur komponenti ushbu dastur uchun jarayon mavjud bo'lganda ishlayotgan bo'lsa (chunki ilovaning boshqa komponenti mavjud), u holda komponent ushbu jarayonda ishlaydi va bir xil bajarilish yo'lidan foydalanadi. Shu bilan birga, alohida jarayonlarda boshqa dastur komponentlarining bajarilishini tashkil qilish va har qanday jarayon uchun qo'shimcha oqim yaratish mumkin.

Jarayonlar

Jarayonlar odatiy bo'lib, bitta dasturning barcha komponentlari bir xil jarayonda ishlaydi. Biroq, ma'lum bir komponent qaysi jarayonga tegishli ekanligini nazorat qilish kerak bo'lsa, buni manifest faylida bajarish lozim.

Komponent elementining har bir turi uchun manifest yozuvi - <Activity>, <servis> va <provayder> - komponent ishlashi kerak bo'lgan

jarayonni belgilash imkonini beruvchi *android:process* atributini qo'llab-quvvatlaydi. Ushbu atributni har bir komponent o'z jarayonida ishlatishi yoki – faqat ba'zi komponentlar bir xil jarayonga ega bo'lishi uchun o'rnatishi mumkin. Ilovalar bir xil Linux foydalanuvchi identifikatorini baham ko'rishi va bir xil sertifikat bilan tizimga kirishi sharti bilan *android:process* jarayonini bir xil jarayonda turli ilovalarning komponentlarini ishga tushirish uchun sozlashi mumkin.

<*application*> elementi *android:process* atributini qo'llab-quvvatlaydi, bu barcha komponentlar uchun amal qiladigan standart qiymatni o'rnatish imkonini beradi.

Android da yetarlicha xotira bo'lmaganda va hozirda foydalanuvchiga xizmat ko'rsatayotgan boshqa jarayonlarga kerak bo'lganda jarayonni to'xtatish mumkin. Ushbu jarayonda ishlaydigan dastur komponentlari ketma-ket to'xtatiladi. Ushbu komponentlar uchun ish mavjud bo'lganda, jarayon qayta boshlanadi. Jarayonlarni tugatish to'g'risida qaror qabul qilishda Android tizimi ularning foydalanuvchi uchun nisbiy ahamiyatini hisobga oladi. Misol uchun, ekranda ko'rinmaydigan harakatlarni o'z ichiga olgan jarayonlar ko'rinadigan harakatlarni o'z ichiga olgan jarayonlarga qaraganda yopilish ehtimoli ko'proq. Shuning uchun jarayonni tugatish to'g'risidagi qaror ushbu jarayonda ishlaydigan komponentlarning holatiga bog'liq. To'xtatilishi mumkin bo'lgan jarayonlarni tanlashni tartibga soluvchi qoidalar quyida muhokama qilinadi.

Jarayonning hayot sikli

Android tizimi dastur jarayonini iloji boricha uzoqroq saqlashga harakat qiladi, lekin oxir-oqibat yangiroq yoki muhimroq jarayonlar uchun xotirani tiklash uchun eski jarayonlarni o'chirishga to'g'ri keladi. Qaysi jarayonlarni saqlab qolish va qaysilarini olib tashlashni aniqlash uchun tizim har bir jarayonni jarayonda ishlaydigan komponentlar va ushbu komponentlarning holatiga asoslangan "muhimlik ierarxiyasi" ga joylashtiradi. Avvalo, ahamiyati eng past bo'lgan jarayonlar chiqarib tashlanadi, so'ngra tizim resurslarini tiklash uchun zarur bo'lganda keyingi muhimlik darajasiga ega bo'lgan jarayonlar chiqarib tashlanadi va hokazo.

Muhimlik ierarxiyasida beshta daraja mavjud. Quyidagi ro'yxat muhimlik tartibida jarayonlarning har xil turlarini ko'rsatadi (birinchi jarayon eng muhimi va oxirigisi olib tashlanadi):

1. Oldingi jarayon

Foydalanuvchining joriy *Activity* uchun talab qilinadigan jarayoni. Agar quyidagi shartlardan biri to'g'ri bo'lsa, jarayon oldingi jarayon hisoblanadi:

- unda foydalanuvchi o'zaro aloqada bo'lgan *Activity* mavjud (*Activity*-ning *onResume()* usuli chaqiriladi);
- u foydalanuvchi o'zaro aloqada bo'lgan harakat bilan bog'liq servisni o'z ichiga oladi;
- unda "oldingi planda" ishlaydigan *startForeground()* servisi mavjud;
- unda hayot tsiklining qayta ishga tushirish usullaridan birini (*onCreate()*, *onStart()* yoki *onDestroy()*) bajaradigan servis mavjud;
- unda *onReceive()* usulini bajaradigan *BroadcastReceiver* usuli mavjud.

Odatda bir vaqtning o'zida bir nechta oldingi jarayonlar ishlaydi. Ular faqat oxirgi chora sifatida yo'q qilinadi, agar unda juda oz xotira qolsa, ular birgalikda ishlashni davom ettira olmaydilar. Odatda, bu nuqtada xotira sahilalash holatiga yetadi, shuning uchun foydalanuvchi interfeysi foydalanuvchi harakatlariga javob berishi uchun, birinchi navbatda, ba'zi jarayonlarni olib tashlashi kerak.

2. Ko'rinadigan jarayonlar

Oldingi komponentlarni o'z ichiga olmaydi, lekin ekrandagi displeyga ta'sir qilishi mumkin bo'lgan jarayonlardir. Agar quyidagi shartlardan biri bajarilsa, jarayon ko'rinadigan hisoblanadi:

- unda oldingi planda bo'lmagan, lekin foydalanuvchiga ko'rinadigan *Activity* mavjud (*onPause()* usuli chaqiriladi). Masalan, agar oldingi plandagi *Activity* uning orqasida oldingi *Activity*-ni ko'rish imkonini beruvchi dialog oynasini ishga tushirsa, bu sodir bo'lishi mumkin;
- unda ko'rinadigan yoki oldingi *Activity* bilan bog'liq servis mavjud.

Ko'rinadigan jarayon juda muhim hisoblanadi va faqat barcha oldingi jarayonlarni davom ettirish zarur bo'lganda olib tashlanishi kerak.

3. Xizmat ko'rsatish jarayoni

Xizmatni ishga tushiradigan jarayon *startService()* usuli bilan boshlangan va yuqori darajadagi ikkita toifaga kirmaydi. Uni saqlash jarayonlari foydalanuvchi ko'rgan narsaga bevosita bog'liq bo'lmasa-da, ular odatda foydalanuvchi uchun muhim amallarni bajaradi (masalan, fonda musiqa tinglash yoki tarmoqqa ma'lumotlarni yuklash), shuning uchun tizimda etarli xotira mavjud bo'lsa, ularning ishlashini ta'minlaydi. Ular barcha ko'rinadigan va oldingi jarayonlar bilan birga ishlaydi.

4. Fon jarayoni

Hozirda foydalanuvchiga ko'rinmaydigan harakatlarni o'z ichiga olgan jarayonlar (*Activity*-ning *onStop()* usuli chaqirilganda). Ushbu jarayonlar foydalanuvchi tajribasiga bevosita ta'sir qilmaydi va tizim ularni istalgan vaqtda oldingi, ko'rinadigan yoki yordamchi jarayonlar uchun

xotiradan bo'shatish uchun o'chirib tashlashi mumkin. Odatda ko'plab fon jarayonlari ishlaydi, shuning uchun ular ro'yxatda saqlanadi, shunda foydalanuvchi ko'rgan eng so'nggi *Activity*-ni o'z ichiga olgan jarayonlar oxirgi o'chiriladi. Agar *Activity*-ning hayot sikli usullari to'g'ri amalga oshirilsa va *Activity* joriy holatini saqlasa, *Activity* jarayonini o'chirish foydalanuvchi tajribasida ko'rinadigan ta'sir qilmaydi, chunki foydalanuvchi *Activity*-ga qaytganda, u barcha ko'rinadigan holat elementlarini tiklaydi.

5. Bo'sh jarayon

Bo'sh jarayon bu har qanday faol dastur komponentlarini o'z ichiga olmaydigan jarayondir. Ushbu turdagi jarayonni saqlashning yagona sababi keshlashdir, bu esa keyingi safar komponent ushbu jarayonda ishlaganda yaxshilanadi. Tizim tez-tez barcha tizim resurslarini jarayon keshi va asosiy yadro keshi o'rtasida teng taqsimlash uchun ushbu jarayonlarni olib tashlaydi.

Android tizimi jarayonni hozirgi vaqtda faol bo'lgan komponentlarning ahamiyatidan kelib chiqib, jarayonni mumkin bo'lgan eng yuqori darajada belgilaydi. Misol uchun, agar jarayon servis va ko'rinadigan *Activity*-ni o'z ichiga olsa, jarayon xizmat ko'rsatish jarayoni emas, balki ko'rinadigan hisoblanadi. Bundan tashqari, jarayonning darajasini oshirish mumkin, chunki unga bog'liq bo'lgan boshqa jarayonlar ham mavjud. Masalan, boshqa jarayonga xizmat ko'rsatadigan jarayon xizmat ko'rsatilayotgan jarayon darajasidan past darajaga ega bo'lishi mumkin emas. Masalan, agar *A* jarayonidagi kontent provayderi *B* jarayonidagi mijozga xizmat ko'rsatsa yoki *A* servis jarayoni *B* jarayonidagi komponent bilan bog'langan bo'lsa, *A* jarayoni har doim *B* jarayoni kabi muhim hisoblanadi.

Xizmatni boshqaradigan jarayon fondagi *Activity*-ga ega jarayonga qaraganda qimmatroq bo'lganligi sababli, uzoq davom etadigan *Activity*-ni boshlaydigan *Activity*, ayniqsa, *Activity*-dan uzoqroq davom etsa, shunchaki inchi zanjir yaratish o'rniga, ushbu *Activity* uchun xizmatni boshlashi mumkin. Masalan, veb-saytga rasm yuklaydigan *Activity* yuklashni amalga oshirish uchun xizmatni ishga tushirishi kerak, shuning uchun foydalanuvchi *Activity*-dan chiqqandan keyin ham yuklash fonda davom etishi mumkin. Xizmatdan foydalanish, harakatga nima bo'lishidan qat'i nazar, operatsiya hech bo'lmaganda "xizmat ko'rsatish jarayoni" ustuvorligiga ega bo'lishini kafolatlaydi. Xuddi shu sababga ko'ra, qabul qiluvchilar faqat bajarish uchun uzoq vaqt talab qiladigan operatsiyalarni emas, balki xizmatlardan foydalanishlari kerak.

Oqimlar

Ilova ishga tushirilganda, tizim dastur uchun "master" deb ataladigan ijro yo'lini yaratadi. Bu tushuncha juda muhim, chunki u hodisalarni tegishli foydalanuvchi interfeysi vidjetlariga, jumladan, grafik akslantirish hodisalariga jo'natish uchun javobgardir. Shuningdek, ilova Android UI (User Interfeys) asboblari to'plamidagi komponentlar (*android.widget* va *android.view* paketlari komponentlari) bilan o'zaro aloqada bo'ladi.

Tizim har bir komponent ekzemplari uchun alohida oqim yaratmaydi. Xuddi shu jarayonda ishlaydigan barcha komponentlar UI oqimida yaratilgan va har bir komponentning tizim xabarlarini ushbu tarmoqdan yuboriladi. Shuning uchun, tizimning qayta xabarlariga javob beradigan usullar (foydalanuvchi harakatlari haqida hisobot berish uchun *onKeyDown()* usuli yoki hayot tsiklining qayta oqimi) har doim jarayonning UI tarmog'ida ishlaydi.

Masalan, foydalanuvchi ekrandagi tugmani bosganda, ilovaning vidjetga bosish hodisasini yuboradi, bu esa o'z navbatida tugmani bosilgan holatga o'rnatadi va voqea navbatiga bekor qilish so'rovini yuboradi. UI tarmog'i so'rovni navbatdan chiqaradi va vidjetga uni qayta ko'rsatilishi kerakligi haqida xabar beradi.

Ilova foydalanuvchi harakatlariga javoban intensiv ishlarni amalga oshirganda, agar dastur to'g'ri bajarilmasa, ushbu bitta oqim modeli yomon ishlashni namoyish qilishi mumkin. Ya'ni, agar hamma narsa foydalanuvchi interfeysi oqimida sodir bo'lsa, tarmoqqa kirish yoki ma'lumotlar bazasi so'rovlari kabi uzoq davom etadigan operatsiyalarni bajarishda butun foydalanuvchi interfeysini bloklaydi. Oqim bloklanganda, hech qanday hodisani, jumladan, displeyni o'zgartirish hodisalarini qayta ishlashi mumkin emas. Foydalanuvchi nuqtai nazaridan, dastur osilgan yoki qotib qolgan ko'rinadi.

Agar foydalanuvchi interfeysi oqimi bir necha soniyadan ko'proq vaqt davomida bloklangan bo'lsa (hozirda taxminan 5 soniya), mashhur "ilova javob bermayapti" dialog oynasi ko'rsatiladi. Keyin norozi foydalanuvchi ilovadan chiqib, uni o'chirib tashlashi mumkin.

Bundan tashqari, Android foydalanuvchi interfeysining asboblari majmuasi oqim bilan xavfsiz emas hisoblanadi. Shuning uchun ishchi oqimdan foydalanuvchi interfeysida ishlamaslik kerak. Foydalanuvchi interfeysi manipulyatsiyasi foydalanuvchi interfeysi oqimidan amalga oshirilishi kerak. Shunday qilib, Androidning bir oqimli modelining faqat ikkita qoidasi mavjud:

1. Foydalanuvchi interfeysi oqimini bloklamaslik;

2. Android foydalanuvchi interfeysi asboblari to'plamiga oqim tashqarisidan murojij qilmaslik.

Nazorat savollari

1. Mobil ilovalarning operatsion tizimini yaratish uchun qanday platformalardan foydalaniladi?
2. *Logcat* oynasida qanday ma'lumotlar akslanadi?
3. *Activity* nima, uni qanday ishlashini tushuntirib bering.
4. Mobil ilovalarning hayotiy siklini tushuntirib bering.
5. *Activity*-ning hayotiy siklida qanday usullar qatnashadi?
6. Operatsion tizim mobil ilovalarini yaratish uchun qanday platformalardan foydalaniladi?
7. Mobil ilovalarda jarayonlar qanday ishlaydi?
8. Jarayonlarning qanday turlari mavjud?
9. Oqimlar qanday qo'llaniladi?
10. Ilovalarning hayotiy sikli qanday boshqariladi?

2. MOBIL QURILMALAR BILAN ISHLASH

2.1. Mobil operatsion tizim tushunchasi

2.1.1. Mobil qurilmalarda operatsion tizim tushunchasi

Zamonaviy odamning hayotini mobil qurilmalarsiz deyarli tasavvur qilib bo'lmaydi. Ularning sifati apparat xususiyatlariga ko'proq bog'liq, ammo foydalanish qulayligi mobil operatsion tizimga ko'proq bog'liq. Shuning uchun nafaqat yaxshi apparat xususiyatlarini tanlash, balki ishlash uchun qulay bo'lgan operatsion tizimni ham tanlash juda muhimdir. Mobil operatsion tizim mobil qurilmalarni boshqarish uchun mo'ljallangan tizimdir. Mobil qurilmalar uchun ko'p tarqalgan operatsion tizimlar:

- Symbian OT
- Windows Mobile OT
- Android OT
- iOS OT
- Blackberry OT

2.1.2. Symbian OT

Symbian operatsion tizimi Nokia qo'llab-quvvatlashi tufayli mobil qurilmalar uchun eng mashhur operatsion tizim bo'ldi. Tizimning kichik o'lchamga ega bo'lishi, shuningdek, grafik interfeysi va tizim yadrosi bir-biridan ajratilganligi ham muhim rol o'ynadi. Bu uni turli xil mobil qurilmalarga o'tkazishni osonlashtirdi. Har bir ishlab chiqaruvchi ushbu operatsion tizimni o'zi ishlab chiqqan apparat platformasining cheklovlariga qarab o'z distributivini yaratdi. Series 60, Series 80, Series 90, UIQ va MOAP versiyalari shunday paydo bo'ldi. Har bir versiyaning o'ziga xos xususiyatlari bor, bu har bir versiya uchun o'z ilovalarini ishlab chiqish zaruratini tug'dirdi. Bu foydalanuvchilarga noqulay edi, shuning uchun Windows Mobile, Android va iPhone OT paydo bo'lgandan so'ng, u mobil qurilmalar ishlab chiqaruvchilari orasida mashhurligini yo'qotdi. Ayni paytda mobil qurilmalarning yirik ishlab chiqaruvchilari orasida faqat Nokia o'zining smartfonlari uchun ushbu OTdan foydalanadi.

Symbian OTning afzalliklari: xotira va protsessorga past talablar, foydalanilmagan xotirani bo'shatish funksiyasi, barqarorligi, ushbu platforma uchun viruslar sonining kamligi, ko'p sonli dasturlar mavjudligi, yangi versiyalar tez chiqarildi va xatoliklar tuzatildi.

Symbian OTning kamchiliklari: eski va yangi versiyalar uchun dasturlarning mos kelmasligi, shaxsiy kompyuter bilan muloqot qilish uchun qo'shimcha dasturiy ta'minotni o'rnatish kerak.

2.1.3. Windows Mobile

Windows Mobile OT operatsion tizimlar ishlab chiqarish bo'yicha jahon yetakchisi – Microsoft tomonidan ishlab chiqilgan. Ushbu tizim Windows ish stoli versiyasi bilan bir xil dasturlash interfeysidan foydalanadi. Bu dasturlarni yozishni osonlashtiradi va foydalanuvchilar Windows ish stolidan tanish bo'lgan shaffof, ishlatish uchun qulay interfeysdan bahramand bo'lishadi.

Windows Mobile OT – bu komponentlarga asoslangan, ko'p vazifali, ko'p oqimli, ko'p platformali operatsion tizim. Buning yordamida u mobil qurilmalarda keng tarqaldi.

Windows Mobile OTning afzalliklari: ish stoli versiyasi bilan o'xshashlik, qulay sinxronizatsiya, ofis dasturlari, ko'p vazifaliligi.

Windows Mobile OTning kamchiliklari: yuqori apparat talablari, ko'p sonli viruslarning mavjudligi, tizim ishidagi xatoliklar.

2.1.4. Android operatsion tizimi

Android OT Linux operatsion tizimiga asoslangan eng yosh mobil operatsion tizimlardan biri bo'lib, Google ko'magida Open Handset Alliance (OHA) tomonidan ishlab chiqilgan. Dastur kodi jamoat mulki hisoblanadi, shuning uchun har qanday ishlab chiquvchi ushbu mobil operatsion tizimining o'z versiyasini yaratishi mumkin. Ilova ishlab chiquvchilari uchun bir nechta cheklovlar mavjud, shuning uchun Android Marketdan qulay tarzda yuklab olinadigan ko'plab bepul va pullik ilovalar mavjud.

Android OTning afzalliklari: moslashuvchanlik, ochiq kodliligi, ko'plab dasturlar mavjudligi, yuqori unumdorlik, Google xizmatlari bilan qulay o'zaro aloqa, ko'p vazifaliligi.

Android OTning kamchiliklari: ko'plab joriy versiyalar – ko'plab qurilmalar uchun yangi versiya juda kech keladi yoki umuman paydo bo'lmaydi, shuning uchun ishlab chiquvchilar eski versiyalar asosida ilovalarni ishlab chiqishlari kerak, kodning ochiqligi tufayli xakerlik hujumlariga yuqori sezgirlik, deyarli har doim yangilanishi kerak.

2.1.5. iPhone OT

iPhone OT – bu Apple kompaniyasining mobil operatsion tizimi. Bu tizim faqat Apple mahsulotlarida keng tarqalgan. iOS Apple ning iPhone, iPod, iPad va Apple TV uskunalarida o'rnatilgan operatsion sistema sanaladi.

iOS (eski nomi iPhone OT) – amerikalik muhandis Steve Jobs tomonidan ishlab chiqarilgan mobil aloqa operatsion sistemasi. iOS multitach (ko'p belgi sezuvchanlik) xususiyati va tezkorligi bilan boshqa mobil aloqa operatsion tizimlardan yaqqol ajralib turadi. iOSning birinchi versiyasi 9-yanvar 2007-yilda taqdim etildi va hozirga kelib eng so'nggi versiyasi iOS 14.5.1 (3-may 2021-yil) chiqarildi. iOSga o'yin yoki dasturlar ilovalari onlayn-do'kon App Store dasturi orqali ko'chirib olish bilan amalga oshirish mumkin.

iPhone OTning afzalliklari: foydalanish qulayligi, yuqori sifatli qo'llab-quvvatlash xizmati, ishdagi ko'plab muammolarni bartaraf etadigan muntazam yangilanishlar, AppStore do'konida juda ko'p turli xil dasturlarni sotib olish imkoniyati.

iPhone OTning kamchiliklari: norasmiy ilovalarni o'rnatish uchun jailbreak zarurati, OTning yopiq kodliligi, ko'p vazifalarning yetishmasligi.

2.1.6. Palm OT

Palm operatsion tizimi 1996 yilda taqdim etilgan. Palm OTning qiziq jihati shundaki, OTning yadrosi ko'p vazifali bo'lib, OT foydalanuvchisi uchun u bir vazifali bo'lib, fonda musiqa, MP3 va hokazolarni ijro etish imkoniyatiga ega, u bir vaqtning o'zida ekranda faqat bitta ilovani ko'rsatishi mumkin. Palm OT yadro darajasida fon vazifalarini yaratish uchun ishlab chiquvchilariga API-larni oshkor qilishni taqiqlaydi. Foydalanuvchilarning keng imkoniyatlari va qulayligi tufayli bu juda keng tarqalgan edi. Bugungi kunga qadar u deyarli qo'llanilmadi, ammo bu yil ishlab chiquvchi HP tomonidan qiziqish bildirildi. Shu tufayli bir paytlar mashhur bo'lgan operatsion tizimining qayta tiklanishiga umidlar kuchaygan.

Palm OTning afzalliklari: resurslarga talabchanligi, juda qulay foydalanuvchi interfeysi, shaxsiy kompyuter bilan qulay sinxronizatsiya, ishonchlilik.

Palm OTning kamchiliklari: to'liq huquqli ko'pmasalarning mavjud emasligi, multimedia funksiyalari ishlab chiqilmagan, tizim rivojlantirilmagan.

2.1.7. BlackBerry OT

BlackBerry operatsion tizimi faqat Research In Motion Limited (RIM) tomonidan ishlab chiqarilgan qurilmalarda ishlaydi. Korporativ foydalanuvchilarga yo'naltirilgan. U o'z nomini o'zi yaratilgan smartfonlardan oldi, chunki smartfonlarning klaviaturasi qulupnay kabi ko'rinardi. Ushbu operatsion tizimga ega smartfonlar xabarlamani ulashishning murakkabligi tufayli korporativ muhitda keng tarqaldi.

BlackBerry OTning afzalliklari: elektron pochtdan qulay foydalanish, shaxsiy kompyuter bilan oson sinxronlash, xavfsizlik sozlamalarining keng doirasi.

BlackBerry OTning kamchiliklari: faqat matnli ma'lumotlarni ko'rsatish uchun optimallashtirilgan, grafikalar bilan ishlash sifati unchalik yaxshi emas, juda qulay brauzer emas.

Ko'rib turganingizdek, qurilmaning texnik xususiyatlari mobil qurilmani tanlashda hech qanday asosiy parametr emasligi hisoblanadi.

2.2. Mobil operatsion tizimlar platformasi va arxitekturasi

2.2.1. Android platformasi va uning asoslari

Android – bu mobil qurilmalar, smartfonlar, planshetlar uchun operatsion tizim. Android hozirda mobil qurilmalar uchun eng keng tarqalgan operatsion tizim hisoblanadi.

2003 yilda Endi Rubin, Rich Miner, Nik Sears va Kris Uayt Palo Alto da Android Inc.ga asos solgan. Dastlab kompaniya mobil gadjetlarni loyihalash bilan shug'ullangan, ular geolokatsiya ma'lumotlari asosida foydalanuvchilarning ehtiyojlariga avtomatik ravishda moslashtiriladi.

2005 yil avgust oyida Android Inc. Google kompaniyasining sho'ba korxonasiga aylanadi. Endi Rubin, Rich Miner va Kris Uayt Android Inc. da Linux yadrosi asosidagi operatsion tizim ustida ishlay boshlaydi. Google minglab turli telefon modellarida foydalanish mumkin bo'lgan kuchli platformani joriy etishga qaror qildi. Shu munosabat bilan Open Handset Alliance (OHA) tashkil etildi – 80 dan ortiq kompaniyalardan iborat konsorsium o'z sa'y-harakatlarini mobil qurilmalar uchun ochiq standartlarni ishlab chiqishga yo'naltiradi. OHA tarkibiga Google, HTC, Sony, Dell, Intel, Motorola, Qualcomm, Texas Instruments, Samsung Electronics, LG Electronics, T-Mobile, Sprint Corporation, NVIDIA va boshqalar kiradi.

Androidning birinchi versiyasi 2008 yil 23 sentyabrda taqdim etilgan, versiyaga Apple Pie nomi berilgan. Bundan tashqari, har bir keyingi versiyaning nomi qandaydir shirinlikni ifodalaydi, versiyalar tartibidagi nomlarning birinchi harflari esa lotin alifbosi harflariga mos keladi. Android platformasi Linux yadrosi, o'rta dastur va o'rnatilgan mobil ilovalarga asoslangan operatsion tizimni birlashtiradi. Android mobil platformasini ishlab chiqish va rivojlantirish OHA (Open Handset Alliance) tomonidan boshqariladigan AOSP (Android Open Source Project) loyihasi doirasida amalga oshiriladi va Google butun jarayonni boshqaradi. Android fonda vazifani bajarishni qo'llab-quvvatlaydi; foydalanuvchi interfeysi elementlarining boy kutubxonasini taqdim etadi; OpenGL standartidan foydalangan holda 2D va 3D grafikalarini qo'llab-quvvatlaydi; fayl tizimiga kirishni va o'rnatilgan SQLite ma'lumotlar bazasini qo'llab-quvvatlaydi.

Arxitektura nuqtai nazaridan, Android tizimi to'liq dasturiy ta'minot to'plami bo'lib, uni quyidagi qatlamlarga bo'lish mumkin (2.1-rasm):

- bazaviy daraja (Linux yadrosi) – apparat darajasi va dasturiy ta'minot steki o'rtasidagi darajasi;

- kutubxonalar va bajarilish muhiti (Libraries & Android Runtime) ilovalar uchun eng muhim asosiy funktsionallikni ta'minlaydi, Dalvik virtual mashinasi va Android ilovalarini ishga tushirish uchun zarur bo'lgan asosiy Java kutubxonalarini o'z ichiga oladi;

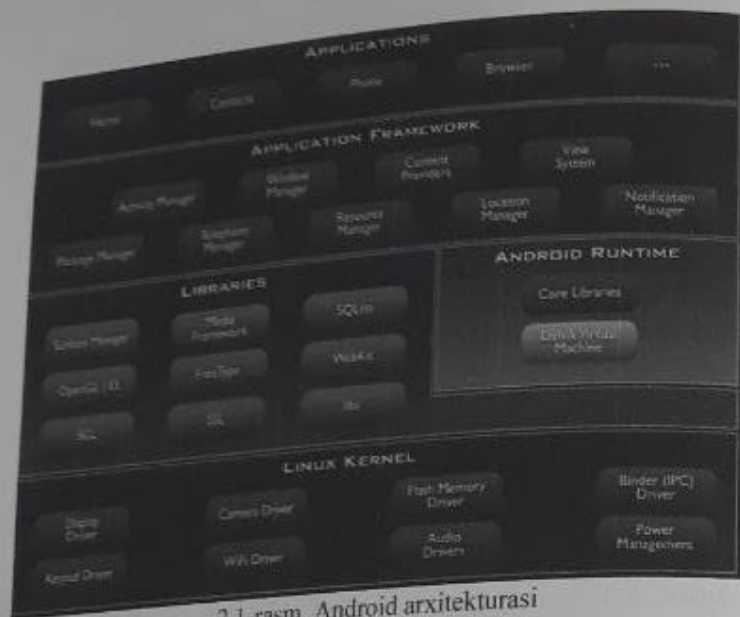
- Application Framework qatlami ishlab chiquvchilarga kutubxona darajasidagi tizim komponentlari tomonidan taqdim etilgan API-larga kirishni ta'minlaydi;

- ilovalar darajasi (Ilovalar) – oldindan o'rnatilgan asosiy ilovalar to'plami.

Komponentlar ierarxiyasi Linux 2.6 OS yadrosiga asoslangan bo'lib, apparat va dasturiy ta'minot o'rtasida oraliq daraja bo'lib xizmat qiladi, tizimning ishlashini ta'minlaydi, yadro tizimi xizmatlarini taqdim etadi: xotirani boshqarish, energiya tizimi va jarayonlar, xavfsizlik, tarmoq va drayverlar.

Yuqori daraja – kutubxonalar to'plami va bajarilish muhiti. Kutubxonalar quyidagi funksiyalarni bajaradi:

- yuqori qatlamlar uchun amalga oshirilgan algoritmlarni taqdim etish;
- fayl formatlarini qo'llab-quvvatlaydi;
- axborotni kodlash va dekodlashni amalga oshiradi (masalan, multimedia kodeklari);
- grafik tasvirni amalga oshiradi va hokazo.



2.1-rasm. Android arxitekturasi

Kutubxonalar C/C++ da amalga oshiriladi va qurilmaning o'ziga xos uskunasi uchun kompilyatsiya qilinadi, ular bilan ishlab chiqaruvchi tomonidan oldindan o'rnatilgan shaklda taqdim etiladi. Android OTda foydalanadigan kutubxonalar quyidagicha:

Surface Manager kompozit oynalar boshqaruvchisidir. Kiruvchi buyruqlari buferda to'planadi, ular ma'lum kompozitsiyani tashkil qiladi va keyin ekranda ko'rsatiladi. Bu tizimga qiziqarli uzluksiz effektlar, oyna shaffoqligi va silliq o'tishlarni yaratishga imkon beradi.

Media Framework – PacketVideo OpenCORE asosida amalga oshirilgan kutubxonalar. Audio va video kontentni yozib olish va namoyish etish, shuningdek, statik tasvirlarni ko'rsatish uchun ishlatiladi. Qo'llab-quvvatlanadigan formatlar: MPEG4, H.264, MP3, AAC, AMR, JPG va PNG.

SQLite – bu Android tomonidan asosiy ma'lumotlar bazasi mexanizmi sifatida ishlatiladigan engil va kuchli relyatsion ma'lumotlar bazasi mexanizmi.

3D kutubxonalar – apparat tezlashuvidan foydalangan holda 3D grafiklarni yuqori darajada optimallashtirilgan tarzda ko'rsatish uchun

ishlatiladi. Kutubxonalar OpenGL ES API asosida amalga oshiriladi. OpenGL ES (O'rnatilgan tizimlar uchun OpenGL) o'rnatilgan tizimlarda ishlashga moslashtirilgan OpenGL grafik dasturlash interfeysining kichik to'plamidir.

FreeType – bitmaplar bilan ishlash, shriftlarni rasterlash va ular ustida amallarni bajarish uchun kutubxona.

LibWebCore – bu WebKit brauzerining kutubxonasi bo'lib, u mashhur Google Chrome va Apple Safari brauzerlarida ham qo'llaniladi.

SGL (Skia Graphics Engine) – 2D grafikalar bilan ishlash uchun ochiq kutubxona. Bu kutubxona Google mahsuloti bo'lib, ko'pincha boshqa dasturlarda qo'llaniladi.

SSL – xuddi shu nomdagi kriptografik protokolni qo'llab-quvvatlash uchun kutubxonalar.

Libc – bu C standart kutubxonasi, ya'ni uning BSD ilovasi, Linux-ga asoslangan qurilmalarda ishlash uchun tuzilgan.

Dasturning bajarilish muhiti Java tilining asosiy kutubxonalarini uchun mavjud bo'lgan past darajadagi funktsionallikning ko'p qismini ta'minlovchi yadro kutubxonalarini va ilovalarni ishga tushirish imkonini beruvchi Dalvik virtual mashinasini o'z ichiga oladi. Har bir dastur o'ziga xos virtual mashinada ishlaydi va shu bilan ishlaydigan ilovalarni operatsion tizimdan va bir-biridan ajratib turadi. Dalvik virtual mashinasida ishlash uchun Java sinflari Android SDKga kiritilgan *dx* vositasi yordamida bajariladigan *.dex* fayllariga kompilyatsiya qilinadi. DEX (Dalvik EXecutable) – bu Dalvik virtual mashinasi uchun minimal xotiradan foydalanish uchun optimallashtirilgan bajariladigan fayl formati. Eclipse IDE va ADT (Android Development Tools) plaginidan foydalanilganda Java sinflari avtomatik ravishda *.dex* formatiga kompilyatsiya qilinadi.

Android Runtime arxitekturasi shundan iboratki, dasturlarning ishi qat'iy ravishda virtual mashina muhitida amalga oshiriladi, bu OT yadrosini uning boshqa komponentlari tomonidan mumkin bo'lgan zararlardan himoya qilishga imkon beradi. Shuning uchun, xato kodi yoki zararli dasturlar ishlaganda Android va unga asoslangan qurilmani buza olmaydi.

Arxitekturasi har qanday dasturga kirishga ruxsat berilgan boshqa ilovalarning allaqachon amalga oshirilgan imkoniyatlaridan foydalanishga imkon beradi. Uni tarkibiy qismlari o'z ichiga quyidagilarni oladi:

– ro'yxatlar, matn maydonlari, jadvallar, tugmalar yoki hatto o'rnatilgan veb-brauzer kabi vizual ilova komponentlarini yaratish uchun ishlatilishi mumkin bo'lgan kengaytiriladigan *Views* to'plami (*Viewlar*);

– kontent provayderlar (*Content Providers*), ba'zi ilovalar boshqa ilovalar uchun o'z ishlarida foydalanishlari uchun ochadigan ma'lumotlarni boshqarish;

– Resurs menejeri (*Resource Manager*), bu string ma'lumotlar, grafiklar, fayllar va boshqalar kabi funktsionalliksiz (kodni o'tkazmaydigan) resurslarga kirishni ta'minlaydi;

– bildirishnomalar menejeri (*Notification Manager*), ilovalarga holat satriida foydalanuvchi uchun o'z bildirishnomalarini ko'rsatishga imkon beradi;

– Ilovaning hayot davrlarini boshqaradigan (*Activity Manager*), harakatlar bilan ishlash tarixini saqlaydigan va harakatlar uchun navigatsiya tizimini ta'minlaydi;

– Joylashuv menejeri (*Location Manager*), bu ilovalarga vaqti-vaqti bilan qurilmaning joriy geografik joylashuvi haqidagi yangilanishlarni olish imkonini beradi.

Application Framework Android ilovalari uchun yordamchi funktsionallikni ta'minlaydi va shu bilan ilova va OT komponentlarini qayta ishlatish tamoyilini xavfsizlik siyosati doirasida amalga oshiradi.

Foydalanuvchi uchun eng yuqori va eng yaqin bu amaliy qatlamdur. Aynan shu darajada foydalanuvchi Android qurilmasi bilan o'zaro aloqada bo'ladi. Bu erda Android operatsion tizimida oldindan o'rnatilgan asosiy ilovalar to'plami keltirilgan. Masalan, brauzer, elektron pochta mijozlari, SMS ilovasi, xaritalar, kalendar, kontakt menejeri va boshqalar. Integratsiyalashgan ilovalar ro'yxati qurilma modeli va Android versiyasiga qarab farq qilishi mumkin. Ushbu daraja barcha foydalanuvchi ilovalarini ham o'z ichiga oladi. Ishlab chiquvchi odatda yangi ilovalar yaratish uchun Android arxitekturasining yuqori ikki qatlami bilan o'zaro ishlaydi. Kutubxonalar, ish vaqti tizimi va Linux yadrosi dastur tizimi orqasida berkitilgan.

2.2.2. Android ilovalarining asosiy turlari

Mobil ilovalarni ishlab chiqishni boshlaganda, qanday turdagi mobil ilovalar mavjudligi haqida tasavvurga ega bo'lish lozim. Ilovaning qaysi turga tegishli ekanligini aniqlash mumkin bo'lsa, uni ishlab chiqish jarayonida qaysi nuqtalarga asosiy e'tibor berish kerakligi aniqroq bo'ladi. Quyidagi turdagi mobil ilovalarni ajratish mumkin:

- oldingi fonda bajariladigan ilovalar;
- orqa fonda bajariladigan ilovalar;

- aralash ilovalar;
- vidjetlar.

1) Oldingi fonda bajariladigan ilovalar o'z vazifalarini faqat ekranda ko'rinadigan holatda bajaradi, aks holda ularning bajarilishi to'xtatiladi. Bunday ilovalarga, masalan, o'yinlar, matn muharrirlari, video pleerlar kiradi. Bunday ilovalarni ishlab chiqishda *Activity*-ning hayot siklini juda diqqat bilan o'rganish kerak, shunda fonga va berkitilgan ilovalarga muammosiz (uzluksiz) o'tadi, ya'ni dastur oldingi o'ringa qaytganida, uning biron bir joyda yo'qolib ketishi sezilmaydi. Ushbu silliqlikka erishish uchun fonga kirishda dastur o'z holatini saqlab qolishiga ishonch hosil qilish kerak va u birinchi o'ringa kelganda uni qayta tiklaydi. Orqa fonda bajariladigan ilovalarini ishlab chiqishda e'tibor berish kerak bo'lgan yana bir muhim jihat – bu qulay va intuitiv interfeysdir.

2) Konfiguratsiyadan so'ng orqa fon ilovalari foydalanuvchi aralashuvini talab qilmaydi, ko'pincha ular yashirin holatda ishlaydi. Bunday ilovalarga misol qilib qo'ng'iroqlarni tekshirish xizmatlari, avtomatik SMS-javob berish qismlari kiradi. Ko'pincha orqa fon ilovalari apparat, tizim yoki boshqa ilovalar tomonidan yaratilgan hodisalarni kuzatishga qaratilgan bo'lib, ular yashirin tarzda ishlaydi. Butunlay yashirin tarzda ishlovchi ilovalar uchun xizmatlarni yaratish mumkin, ammo keyin ular boshqarilmaydi. Foydalanuvchiga ruxsat berilishi kerak bo'lgan minimal harakatlar: xizmatni ishga tushirishga ruxsat berish, kerak bo'lganda uning ishini sozlash, to'xtatib turish va to'xtatish.

3) Aralash ilovalar ko'pincha fonda ishlaydi, lekin sozlashdan keyin foydalanuvchi o'zaro ta'siriga ruxsat beradi. Odatda, foydalanuvchilarning o'zaro ta'siri har qanday hodisalar haqida xabar berishgacha qisqartiriladi. Bunday ilovalarga multimedia pleerlari, matnli xabarlar (chatlar) almashish dasturlari, pochta mijozlari misol bo'la oladi. Orqa fonda ishlashni yo'qotmasdan foydalanuvchi kiritishiga javob berish qobiliyati aralash ilovalarning o'ziga xos xususiyati hisoblanadi. Bunday ilovalar odatda ko'rinadigan *Activity*-ni ham, yashirin (fon) xizmatlarini ham o'z ichiga oladi va foydalanuvchi bilan muloqot qilishda ularning joriy holatidan xabardor bo'lishi kerak. Agar ilova oldingi planda bo'lsa, foydalanuvchi interfeysini yangilash yoki ularni yangilab turish uchun foydalanuvchiga fondan bildirishnomalarni yuborish kerak bo'lishi mumkin va bunday ilovalarni ishlab chiqishda bu xususiyatlarni hisobga olish kerak.

4) Vidjetlar ish stolida grafik obyekt sifatida ko'rsatiladigan kichik ilovalardir. Misol sifatida batareya quvvati, ob-havo bashorati, sana va vaqt kabi dinamik ma'lumotlarni ko'rsatish uchun ilovalarni o'z ichiga oladi.

Albatta, murakkab ilovalar ko'rib chiqilgan har bir turdagi elementlarni o'z ichiga olishi mumkin. Ilovani ishlab chiqishni rejalashtirayotganda, undan qanday foydalanishni aniqlash kerak, shundan keyingina dizayn va ishlab chiqishning o'zi bilan davom etish lozim.

2.2.3. Ilova arxitekturasi, asosiy komponentalar

Android ilovalari arxitekturasi asosiy qurilish bloklari bo'lgan komponentlarni qayta ishlatish g'oyasiga asoslanadi. Har bir komponent alohida obyekt bo'lib, dasturning umumiy harakatini aniqlashga yordam beradi. Android tizimi shunday qurilganki, har qanday dastur boshqa ilovaning kerakli komponentini ishga tushira oladi. Misol uchun, agar ilova suratga olish uchun kameradan foydalanishi kerak bo'lsa, u ilovada kamera faolligini yaratishi shart emas. Shubhasiz, qurilmada allaqachon kameradan fotosuratlar olish uchun dastur mavjud, foydalanuvchi kamerani o'zi ishlatadigan ilovaning bir qismi deb hisoblashi uchun tegishli *Activity*-ni boshlash, suratga olish va uni ilovaga qaytarish kifoya.

Tizim komponentni ishga tushirganda, agar u hali ishlayotgan bo'lsa, komponentga ega bo'lgan dastur jarayonni boshlaydi va komponent uchun zarur bo'lgan sinflarning namunalarini yaratadi. Shuning uchun, boshqa tizimlardan farqli o'laroq, Android ilovalarida bitta kirish nuqtasi yo'q (masalan, *main()* usuli yo'q). Har bir dastur alohida jarayonda ishlayotganligi va fayllarga kirish cheklavlari tufayli dastur boshqa ilova komponentini bevosita faollashtira olmaydi. Shunday qilib, boshqa dasturning komponentini faollashtirish uchun tizimga ma'lum bir komponentni ishga tushirish niyati haqida xabar yuborish kerak, tizim uni faollashtiradi.

To'rt xil turdagi komponentlarni ajratib ko'rsatish mumkin, ularning har biri ma'lum bir maqsadga xizmat qiladi va tegishli komponent qanday yaratilishi va yo'q qilinishini belgilaydigan o'ziga xos hayot sikliga ega.

Android ilovalari tarkibi

Activity (Activities) – bu dasturning ko'rinadigan qismi (ekran, oyna, shakl), foydalanuvchi grafik interfeysini ko'rsatish uchun javobgardir. Bunda ilovada bir nechta amallar bo'lishi mumkin, masalan, elektron pochta bilan ishlash uchun mo'ljallangan ilovada bitta harakatdan yangi harflar ro'yxatini ko'rsatish, yozish uchun boshqa *Activity* va harflarni o'qish uchun boshqa *Activity*-dan foydalanish mumkin. Ilova foydalanuvchiga yagona obyekt sifatida ko'rinishiga qaramay, ilovaning barcha *Activity*-lari bir-biridan mustaqildir. Shu munosabat bilan, ushbu *Activity*-ning har qanday

turi ushbu ilova *Activity*-ga kirish huquqiga ega bo'lgan boshqa ilovadan ishga tushirilishi mumkin. Masalan, kamera ilovasi olingan suratni foydalanuvchi tomonidan belgilangan manzilga yuborish uchun yangi elektron pochta *Activity*-sini ishga tushirishi mumkin.

Servislar (Services – Xizmatlar) – fonda ishlaydigan, ko'p vaqt talab qiladigan operatsiyalarni bajaradigan yoki masofaviy jarayonlar uchun ishlaydigan komponent. Shuningdek, fonda ishlaydigan ko'rinadigan interfeysga ega bo'lmagan Android ilovasining komponenti hisoblanadi. *Servislar* foydalanuvchi interfeysini ta'minlamaydi. Masalan, foydalanuvchi boshqa ilovadan foydalanayotganda, xizmat fonda musiqa ijro etishi, foydalanuvchining *Activity* bilan o'zaro aloqasini bloklamadan tarmoqdan ma'lumotlarni yuklab olishi mumkin. Xizmat boshqa komponent tomonidan ishga tushirilishi va keyin mustaqil ishlashi mumkin yoki u ushbu komponent bilan bog'lanib qolishi va u bilan o'zaro ta'sir qilishi mumkin. *Servislar* *AndroidManifest.xml* faylida ilova tegida e'lon qilinishi kerak:

```
<application
...
<service
android:name=".ExampleService"
android:enabled="true"
android:label="StackOfSkillsService"/>
...
</application>
```

Servislar ilovaning asosiy oqimida ishlaydi, shuning uchun undagi har qanday jarayonni qayta ishlash kerak bo'lsa, unda yangi oqim yaratish kerak. *Servislar*ning bir necha turi mavjud:

– *Foreground Service*. Ushbu rejimda ishlaydigan servis yuqori ustuvorlikka ega va xotira yetishmovchiligida tizim tomonidan yo'q qilinmaydi. Bunday servis holat satrida ko'rsatiladigan bildirishnoma bilan bog'langan hamda *startForeground()* usuli yordamida ishga tushiriladi.

– *Background Service*. Standart rejimda, fonda, ilovaning asosiy oqimida ishlaydi. U *startService()* usuli yordamida ishga tushiriladi.

– *Bound Service*. Ushbu rejimda ishga tushirilgan servis mijozga, masalan, *Activity*-ga bog'lanadi va mijoz-server tamoyili asosida ishlaydi. *Servislar* bir nechta mijozlarga ulanishi mumkin va agar u bilan bog'langan barcha mijozlar o'chirilgan yoki to'xtatilgan bo'lsa, tizim xizmatni o'chirib qo'yadi. *bindService()* usuli yordamida ishga tushiriladi. *unbindService()* usuli servisni mijozdan ajratish uchun ishlatiladi. *IBinder* interfeysidan

foydalanib, to'g'ridan-to'g'ri servisga murojaat qilish mumkin va usullarni chaqirish mumkin.

Intent Service – bu *Service* sinfining merosxo'ri. *Service* dan farqli o'laroq, u alohida oqimlarda asinxron ishlaydi. *Service startService()* usuli bilan ishga tushiriladi, unda ish uchun zarur bo'lgan ma'lumotlar bilan *Intent* uzatiladi. *onHandleIntent()* usulida kiruvchi *Intent* ni boshqaradi. Ma'lumotlar bo'yicha barcha kerakli operatsiyalar bajarilgandan so'ng, servis o'z ishini to'xtatadi.

Kontent provayderlari (Content providers). Kontent provayderlar ilovaning taqsimlangan ma'lumotlar to'plamini boshqaradi. Ma'lumotlar fayl tizimida, *SQLite* ma'lumotlar bazasida, tarmoqda, ilova uchun mavjud bo'lgan boshqa har qanday joyda saqlanishi mumkin. Kontent provayderlar, agar ular tegishli huquqlarga ega bo'lsa, boshqa ilovalarga so'rovlar yuborish yoki hatto ma'lumotlarni o'zgartirish imkonini beradi. Misol uchun, *Androidda* foydalanuvchining kontakt ma'lumotlarini boshqaradigan kontent provayderi mavjud. Shu munosabat bilan, tegishli huquqlarga ega bo'lgan har qanday ariza har qanday kontaktning ma'lumotlarini o'qish va yozish uchun so'rov yuborishi mumkin. Kontent provayderi tashqaridan kirish uchun mo'ljallanmagan shaxsiy ilova ma'lumotlarini o'qish va yozish uchun ham foydali bo'lishi mumkin.

Eshittirish xabarlarini qabul qiluvchilari (Broadcast Receivers). Qabul qiluvchi – bu eshittirish bildirishnomalariga javob beruvchi komponent. Ushbu bildirishnomalarning aksariyati tizim tomonidan ishlab chiqariladi, masalan, ekran o'chirilganligi yoki batareya quvvati kamligi haqidagi bildirishnoma. Ilovalar, shuningdek, ba'zi ma'lumotlar yuklab olingani va foydalanish uchun mavjud bo'lgan boshqa ilovalarga xabar yuborish kabi translyatsiyani boshlashi mumkin. Qabul qiluvchilar foydalanuvchi interfeysini ko'rsatmasa ham, ular xabar paydo bo'lganda foydalanuvchini ogohlantirish uchun holat satrida bildirishnoma yaratishi mumkin. Bunday qabul qiluvchi boshqa komponentlar uchun o'tkazgich bo'lib xizmat qiladi va kichik hajmdagi ishlarni bajarish uchun mo'ljallangan, masalan, voqeaga mos keladigan servisni boshlashi mumkin.

Eshittirish xabarlarini qabul qiluvchilari – bu qabul qilish (*Intent*) va ularga javob berish uchun foydalaniladigan komponent hisoblanadi. Xabarlar *sendBroadcast()* usuli yordamida yuboriladi. Eshittirish xabarlarini qayta ishlash uchun *IntentFilter* obyektini yaratishi va uni *Broadcast receiver* bilan bog'lash kerak:

```
IntentFilter intentFilter = new IntentFilter();
intentFilter.addAction("SOME_ACTION");
```

```
registerReceiver(receiver, intentFilter);
yoki uni manifestda <intent-filter> tegidan foydalanib ro'yxatdan o'tkazish mumkin.
```

BroadcastReceiver registerReceiver() usuli yordamida dinamik ravishda ro'yxatga olinishi yoki ilovaning *AndroidManifest* faylidagi *<receiver>* tegida statik tarzda yaratilishi mumkin. *BroadcastReceiver* ilovaning asosiy oqimida ishlaydi.

BroadcastReceiver tizimdan xabarlarini qabul qilishi mumkin, masalan, agar telefon zaryadlangan bo'lsa yoki *Android* tizimi ishga tushirilgan bo'lsa. Barcha ko'rib chiqilgan komponentlar *Android SDK* da belgilangan sinflarning avlodlaridir.

Mobil ilova yaratishda qo'llaniladigan asosiy komponentalar quyida keltirilgan.

Asosiy komponentalar

TextView elementi matnni ekranda ko'rsatish uchun mo'ljallangan. U matnni tahrirlash imkoniyatisiz ko'rsatadi.

EditText elementi *TextView* sinfining ost sinfidir. *EditText* elementi matn maydoni bo'lib, matnni kiritish va tahrirlash imkoniyati bilan ifodalanadi. *EditText*-da *TextView*-dagi kabi barcha funksiyalardan foydalanish mumkin.

Button komponentasi ko'p ishlatiladigan elementlardan biri bo'lib, ular *android.widget.Button* sinfi bilan ifodalanadi. Tugmalarning asosiy xususiyati bosish orqali foydalanuvchi bilan muloqot qilish qobiliyatidir.

Android oddiy bildirishnomalarni yaratish uchun *Toast* sinfidan foydalanadi. Aslida *Toast* qisqa vaqt davomida ko'rsatiladigan ba'zi matnli qalqib chiquvchi oynani ifodalaydi.

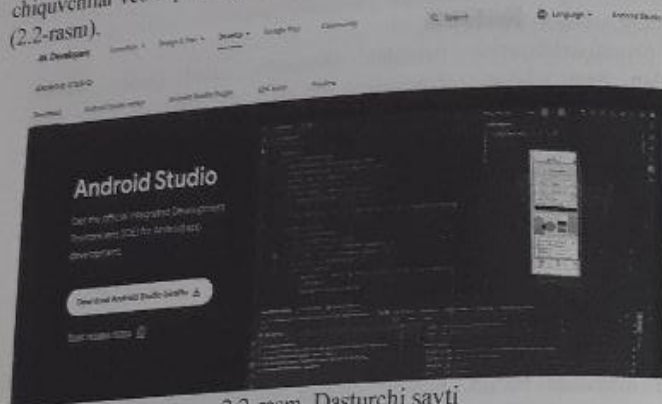
Snackbar elementi biroz *Toast*-ga o'xshaydi: u qalqib chiquvchi xabarlarini ham ko'rsatishga imkon beradi, ammo xabarlar ekran kengligiga mos ravishda cho'ziladi.

Checkbox elementi – bu belgilangan yoki belgilanmagan holatda bo'lishi mumkin bo'lgan tasdiqlash elementidir. Belgilash katakchalari bir nechta qiymatlardan bir nechtasini tanlash imkonini beradi.

2.2.4. Asboblarni o'rnatish va sozlash

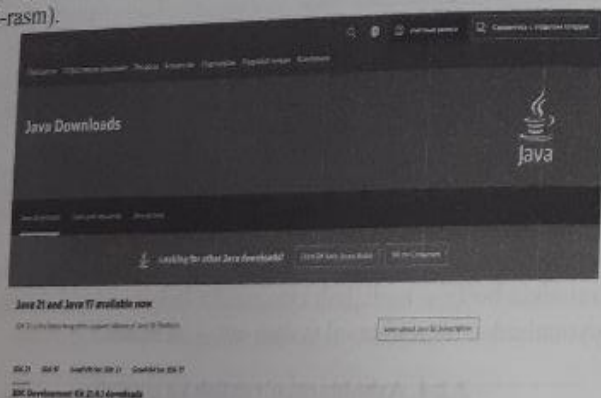
Ko'pgina *Android* ilovalari *Java* dasturlash tilida yozilgan. Eng mashhur dasturiy vosita ishlab chiqish dasturlash muhitlaridan biri bu ADT plagini va *Android SDK* o'rnatilgan *AndroidStudio* muhiti hisoblanadi. Ilgari barcha komponentlarni alohida o'rnatish kerak edi. Endi

AndroidStudio muhitining sozlangan plaginlari bilan ADT Bundle versiyasi mavjud. Unda mobil ilovalarni ishlab chiqish uchun zarur bo'lgan minimal vositalar to'plami mavjud. Shuning uchun dasturiy vositani ishlab chiqishda ADT to'plamiga kiritilmagan vositalar kerak bo'lsa, ularni ishlab chiquvchilar veb-saytidan yuklab olish va ularni muhitga qo'shish mumkin (2.2-rasm).



2.2-rasm. Dasturchi sayti

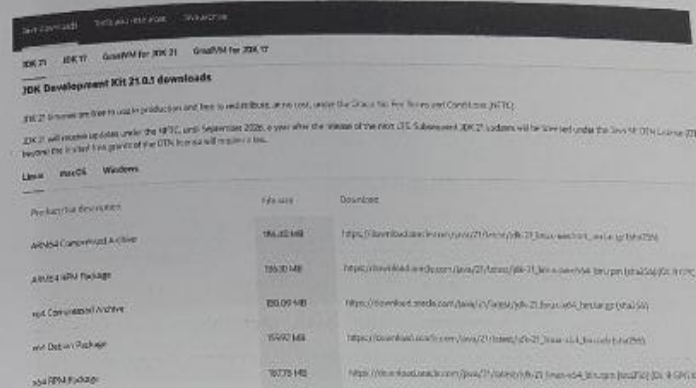
Dasturiy vositani yuklab olish uchun litsenziya shartnomasi shartlarini qabul qilish lozim va Windows versiyasini (32-bit yoki 64-bit) tanlash kerak (2.2-rasm).



2.3-rasm. Oracle veb-sayti

Yuklab olgandan so'ng, arxivni joylashtirmoqchi bo'lgan papkani ochish kerak (dasturiy vosita maxsus o'rnatishni talab qilmaydi). Paketni ochgandan so'ng, papkaga o'tish va AndroidStudioni ishga tushirish lozim.

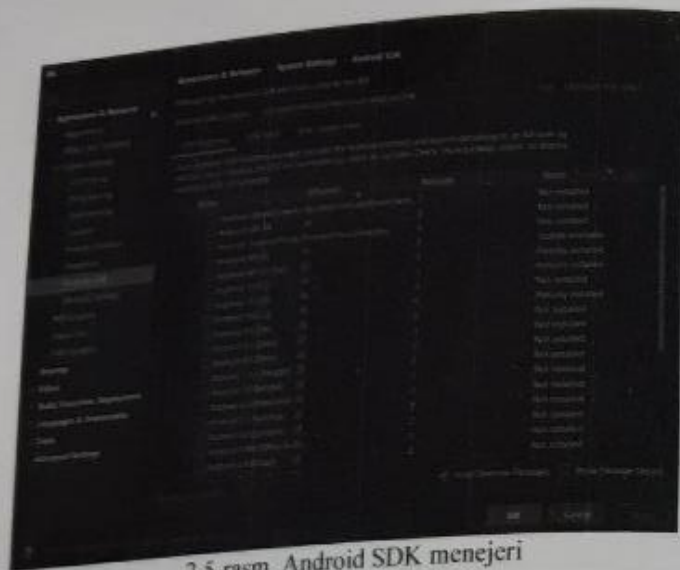
Bu erda kichik muammo yuzaga kelishi mumkin: agar kompyuterda JDK o'rnatilmagan bo'lsa, muhit ishga tushmaydi va JDK jildiga yo'lni belgilashni yoki uni o'rnatishni talab qiladi. JDK ni Oracle web-saytidan yuklab olish mumkin (2.3-rasm). JDK ni yuklab olish uchun avval litsenziya shartnomasi shartlarini qabul qilish va keyin kerakli versiyani tanlash kerak (2.4-rasm).



2.4-rasm. JDK ni yuklab olish

Yuklab olingandan so'ng, o'rnatish faylini ishga tushirish va JDK ni o'rnatish kerak. JDK o'rnatilgandan so'ng, muhit ishga tushishi kerak. Keyinchalik, ish joyini ya'ni loyihalar joylashadigan joyni tanlash (yoki yangi joy yaratish) kerak. Agar u belgilansa, bu ish maydoni sukut bo'yicha tanlanadi, aks holda bu oyna AndroidStudioni har ishga tushirganda paydo bo'ladi. Keyin ishlab chiquvchilar SDKni yanada yaxshilash uchun statistik ma'lumotlarni yuborishni taklif qiladigan oyna paydo bo'ladi. Bu oynada rozi bo'lish uchun belgilashni bosish yoki rad qilish mumkin.

Asboblar panelidagi Android SDK menejeri belgisiga e'tibor berish lozim (shuningdek, *Window* menyusida mavjud). Uning yordamida foydalanuvchi o'z muhitida yangi vositalarni qo'shishi mumkin. Bu darchada shuningdek, o'rnatilgan vositalarni ham ko'rish mumkin. Har bir vosita ro'parasida uning o'rnatilganligi yoki o'rnatilmaganligi haqidagi statusi turadi. Shuningdek, har bir vositaning API darajasi ham akslanadi (2.5-rasm).



2.5-rasm. Android SDK menejeri

Ilova yaratish bosqichlari

Ilova yaratish uchun *File* menyusidagi *New->New Project* buyruqlar ketma-ketligi bajariladi. Keyingi qadamda *New Project* darchasi ochiladi. Bu darchada foydalanuvchi uchun "Phone and Tablet", "Wear OS", "Android TV" va "Automotive" bo'limlarini taklif etiladi. Foydalanuvchi bu bo'limlardan ixtiyoriy shablonni tanlash va tayyor shablonlar bilan tanishib chiqish imkoniyatiga ega.

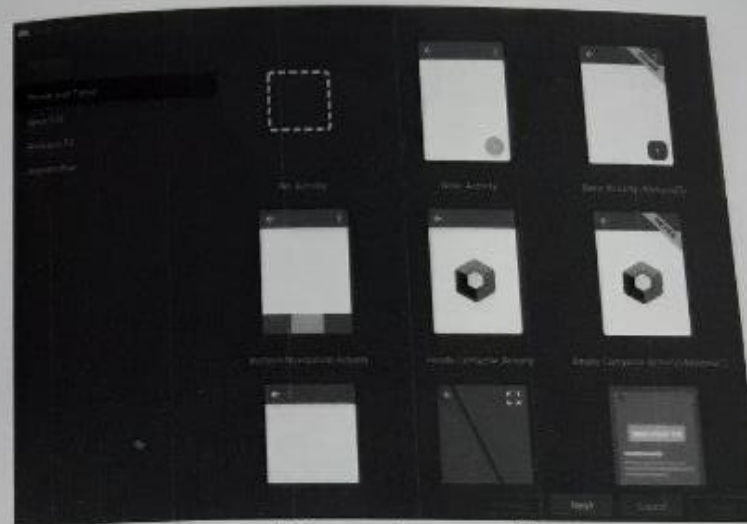
"Phone and Tablet" bo'limda "No Activity", "Empty Compose Activity", "Empty Compose Activity (Material3)", "Empty Activity", "Fullscreen Activity", "Google AdMob Ads Activity", "Google Maps Activity", "Google Pay Activity", "Login Activity", "Primary/Detail Flow", "Navigation Drawer Activity", "Responsive Activity", "Settings Activity", "Scrolling Activity", "Tabbed Activity", "Fragment+ViewModel" va "Native C++" turidagi shablonlar mavjud.

"Wear OS" bo'limda "No Activity", "Blank Activity" va "Watch Face" turidagi shablonlar mavjud.

"Android TV" bo'limda "No Activity" va "Blank Activity" turidagi shablonlar mavjud.

"Automotive" bo'limda "No Activity", "Media Service" va "Messaging Service" turidagi shablonlar mavjud.

Keltirilgan shablonlardan foydalanganda faqat shu ko'rinishdagi interfeyslarni yaratish imkonini beradi (2.6-rasm).



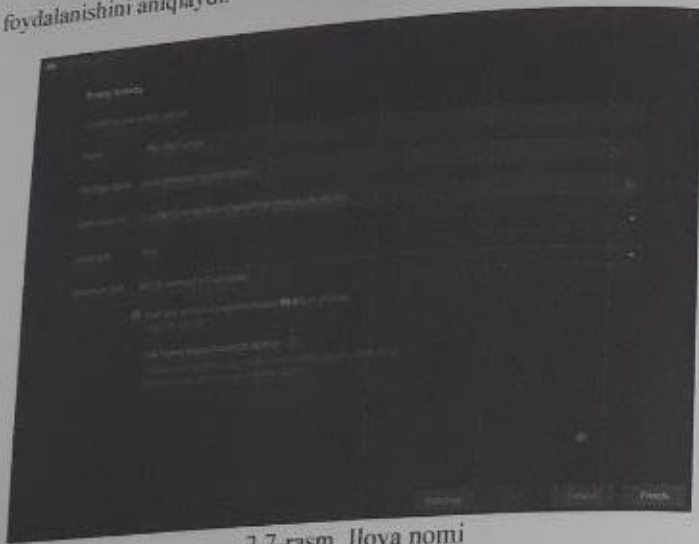
2.6-rasm. Ilova yaratish

Keyingi qadamda ilova nomini, loyihani saqlaydigan joyni va paket nomini, SDKni minimal versiyasi va dasturlash tilini tanlash lozim. Bunda loyiha nomini berilgan nom bilan qoldirmaslik yaxshiroqdir, chunki bunday nomdagi paketni *Google Play*-ga joylashtirish mumkin emas. Albatta, u yerda ta'lim dasturlari yuklanmaydi, ammo kelajakda buni yodda tutish kerak. Dasturlash tili sifatida Java va Kotlin tillari taklif etiladi. SDKni minimal versiyasi sifatida Android 5.0 versiyasi taklif etiladi (2.7-rasm).

Minimal talab qilinadigan SDK – ilova qo'llab-quvvatlaydigan Androidning minimal versiyasi hisoblanadi. Ko'pincha, iloji boricha ko'proq qurilmalarni qo'llab-quvvatlash uchun Android 5.0 versiyasi sukut bo'yicha belgilanadi. Agar ilovaning ma'lum funksiyasi faqat Androidning yangi versiyalarida ishlayotgan bo'lsa va bu ilovaning asosiy funksiyalari to'plami uchun muhim bo'lmasa, uni qo'llab-quvvatlaydigan versiyalarda opsiya sifatida belgilash mumkin.

Target SDK – ilova yoziladigan Android versiyasi; ilovani sinab ko'rilgan Androidning maksimal versiyasini moslik rejimlari uchun belgilaydi.

Compile With – ilova Android funksiyalarining qaysi versiyasidan foydalanishini aniqlaydi.



2.7-rasm. Ilova nomi

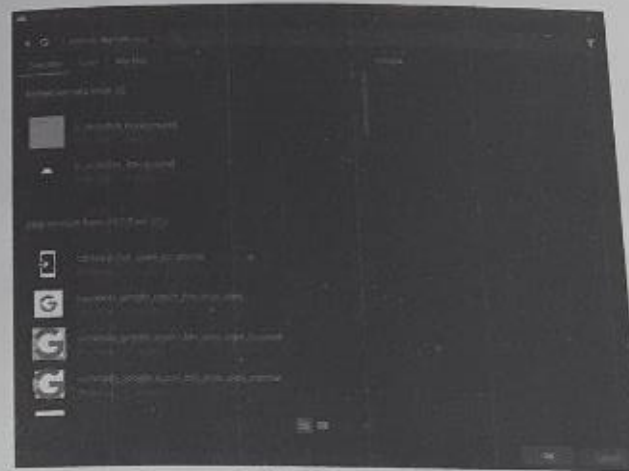
Keyingi oynani o'zgartirishlarsiz o'tkazib yuborish mumkin. Bu yerda: Shaxsiy ishga tushirish belgisini yaratish – ilova belgisini yaratish mumkin. *Activity* yaratish – faollik, mobil ilova oynasi, formasini yaratish mumkin. Ushbu loyiha kutubxona sifatida belgilanadi, ya'ni loyiha *paket* sifatida yaratiladi.

Create Project in Workspace – *Workspace* papkasida loyiha yaratish. Barcha loyihalar ushbu papkada saqlanadi.

Keyingi qadam piktogramma yaratish hisoblanadi. Bunda standart konfiguratsiyani qoldirish yoki o'z konfiguratsiyasini yaratish mumkin. Bu misolda rang sxemasi, shakli o'zgartirildi va clipartdagi rasm tanlangan.

Ko'pgina Android ilovalari o'z ekraniga (shakl, oyna) ega bo'lib, ular *Activity* deb ataladi. Keyingi ikkita oyna bo'sh *Activity*-ni yaratadi. Birinchisini o'zgartirish kerak emas. Ikkinchisida *Activity* nomini o'zgartirish mumkin. *Blank Activity* – mobil telefonlar uchun mo'ljallangan shablon. To'liq ekran *Activity* – ilovani to'liq ekranga cho'zish imkonini beruvchi shablon (navigatsiya paneli va holat panelisiz). *Master/Detail Flow* – planshet kompyuterlar uchun mo'ljallangan shablon.

Shunday tartibda mobil ilova loyihasini yaratish mumkin. Albatta, bu asboblarning to'plamining to'g'ri o'rnatilishini tekshirish uchun o'rnatilgan dastur asosiy hisoblanadi, ammo unda ko'plab ilovalar yaratish mumkin. Loyiha ishga tushgandan keyin uni tarkibini chap tomonda joylashgan maydonda ko'rish mumkin. Bu maydonda loyihadagi barcha fayllar aks ettiriladi (2.8-rasm).

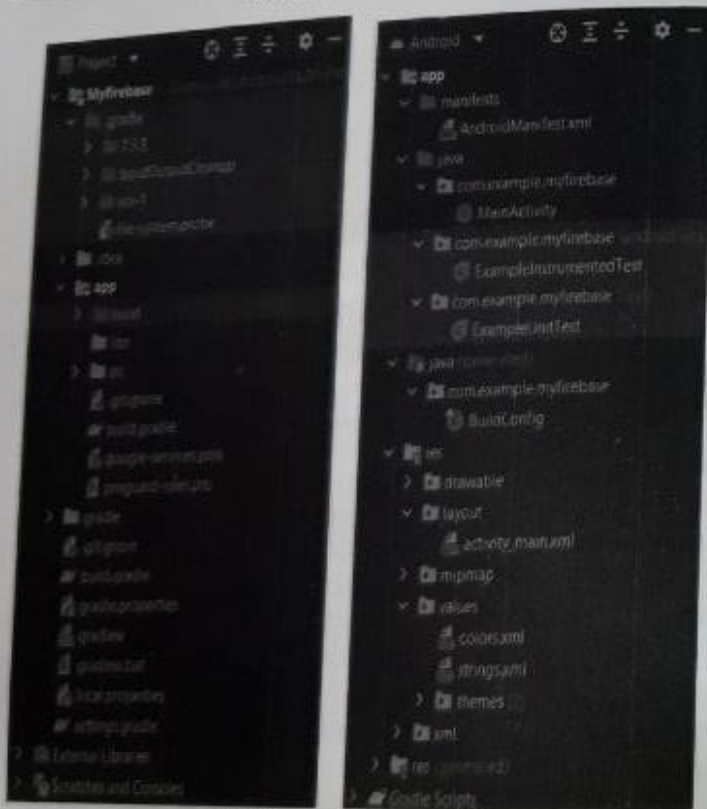


2.8-rasm. Rasm tanlash yoki yangisini yaratish oynasi

Activity fayli paketdagi *src* jildida joylashgan. U *.java kengaytmasiga ega (2.9-rasm). Loyihani ko'rishning bir nechta turi mavjud: *Project*, *Android*, *Packages*, *Project Files*, *Project Source Files*, *Project Non-Source Files*, *Open Files* va *Scratches and Consoles* (2.10-rasm).



2.9-rasm. Activity yaratish



2.10-rasm. Loyiha tarkibi

Layout pastki papkasidagi *res* papkasida *Activity*-ning qobig'i bo'lgan *xml* fayli mavjud. Aynan shu fayl qurilma ekranida ko'rinadigan bo'ladi.

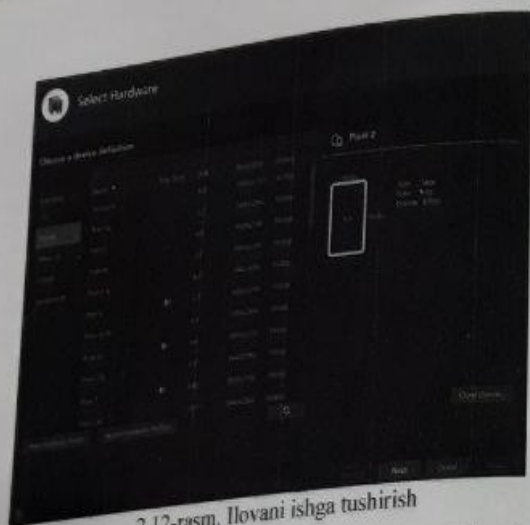
Xml fayllari bilan grafik muharrir rejimida ham ishlash va kodni bevosita tahrirlash mumkin. Sukut bo'yicha *activity_main.xml* fayli yaratiladi va unda *xml* versiyasi va kodiroyka, *ConstraintLayout* va *TextView* elementlari mavjud bo'ladi (2.11-rasm).

ConstraintLayout-da uning balandligi va eni, qo'llaniladigan kontent ma'lumotlar joylashdi. Har bir element teg ichida joylashadi. Bundan tashqari loyihada *build.gradle(app)* va *build.gradle(Modul)* fayllari mavjud bo'lib, ular o'zida loyiha konfiguratsiyasi haqidagi ma'lumotlarni saqlaydi. Bu fayllarga loyihaga yangi materiallar va komponentalar, plaginlar qo'shganda o'zgartirishlar avtomatik yoki foydalanuvchi tomonidan kiritiladi.



2.11-rasm. *Xml* fayli

Ilovani emulyatorida ishga tushirish uchun qurilma emulyatorini yaratish kerak (2.12-rasm). Buni asboblardan panelidagi smartfon tasvirlangan tugmani bosish orqali amalga oshirish mumkin. Agar tugma panelda bo'lmasa, uni *Window* menyusida topish mumkin.



2.12-rasm. Ilovani ishga tushirish

Android Virtual Device Manager darchasi ochiladi. Birinchi marta Android Virtual Device Manager darchasi ochilganda unda bitta ham virtual qurilma bo'lmaydi. Agar virtual qurilma mavjud bo'lsa u ro'yhatda akslanadi (2.18-rasm).



2.18-rasm. Android virtual qurilmalar menejeri

Virtual qurilma yaratish uchun *Create Device* tugmasini bosish lozim. Virtual qurilmani yaratish oynasi paydo bo'ladi. Bu oynada qurilmaga nom berish va kerakli xususiyatlarni tanlash kerak: Qurilma maydonida – qurilmaning modeli tanlanadi. Keyin qurilmaning ekran o'lchami tanlanadi.

Shuningdek, qo'shimcha parametrlarni o'zgartirish mumkin: SD-kartaning o'lchami, o'matilgan xotira va boshqalar (2.19-rasm).



2.19-rasm. AVD yaratish

Keyingi qadamda dasturni ishga tushirish mumkin. Buning uchun asboblardagi *Run* tugmasini (yashil doiradagi oq uchburchak) bosish lozim. Dasturni ishga tushirishdagi muammolarni konsolda kuzatish mumkin. Agar dastur ishga tushmasa, "Ishga tushirish" tugmasining o'ng tomonidagi qora uchburchakni bosib "Konfiguratsiyalarni ishga tushirish" buyrug'ini tanlab, so'ngra "Run" yorlig'ida yaratilgan qurilma tanlanadi va loyiha qayta ishga tushiriladi (2.20-rasm).

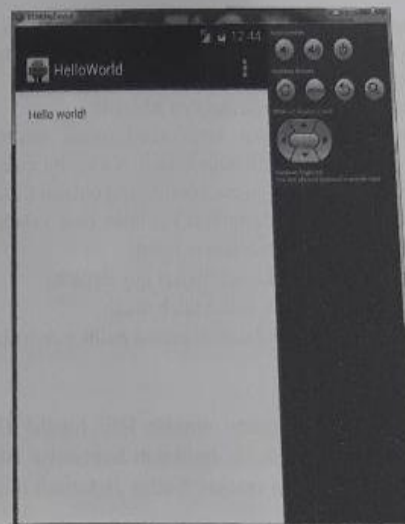


2.20-rasm. Ilovani ishga tushirish

Har bir buyruq to'g'ri bajarilgan bo'lsa, emulyator ishga tushishi kerak. Ishga tushirish vaqti kompyuterdagi RAM miqdoriga bog'liq. Dastur ishlayotgan paytda emulyatorni yopish mumkin emas, u ilovalar ishlaydigan rejimda ishlaydi (2.21-rasm).



2.21-rasm. Emulyatorni ishga tushirish



2.22-rasm. "Hello World" ilovasi va ilovalar menyusi

Agar ilova darhol ishga tushmasa, uni qurilmaning ilovalar menyusidan topish mumkin.

2.3. Mobil operatsion tizim platformasiga mos dasturlash tillari

Dasturlash tili – bu ilovalar uchun kod yoziladigan rasmiy qoidalar to'plami. Hozirgi paytda ko'plab mobil ilovalar yaratish tillari mavjud va ma'lum bir usulni tanlash operatsion tizimga, dastur turiga va unga qo'yiladigan talablarga bog'liq.

2.3.1. Android OT tillari

Java dasturlash tili

Java dasturlash tili 2022-yil iyun holatiga ko'ra eng mashhur dasturlash tili hisoblanadi. Android OT uchun Java dasturlash tili eng asosiy til hisoblanadi. Texnik qo'llab-quvvatlash va yordam ko'rsatadigan katta guruh va mutaxassis ishlab chiquvchilar jamoasi mavjud. Java dasturlash tilining afzalliklari quyidagicha:

• android uchun tabiiy kod. OT qisman Java dasturlash tilida yozilgan va Linux yadrosiga, shuningdek o'zining *Virtual Machine* virtual mashinasiga ega;

- universal – barcha platformalarda ishlaydi;
- obyektga yo'naltirilgan kod orqali mobil ilovalarni osongina masshtablash va yangilash osonroq, bu esa barcha jarayonlarni tezlashtiradi; yozish va yangilash osonroq, bu esa barcha jarayonlarni tezlashtiradi;
- sukut bo'yicha Java dasturlash tili bilan mos keladigan juda ko'p dasturlash vositalari mavjud, tezlikni oshiradi.

Java dasturlash tilining kamchiliklari quyidagicha:

- katta hajmdagi operativ xotira talab qiladi;
- tijorat maqsadlarda foydalanish uchun pullik versiyalari mavjudligi.

Kotlin dasturlash tili

Kotlin dasturlash tili o'zining mashhur IDE, IntelliJ IDEA dasturlash muhitlari bilan mashhur JetBrains tashkiloti tomonidan ishlab chiqilgan. Google-ning Android jamoasi rasman Kotlin dasturlash tilini 2021 yildan buyon qo'llab-quvvatlaydi.

Kotlin dasturlash tili Java tilidagi ba'zi muammolarni hal qilish uchun yaratilgan. Kotlin dasturlash tili sintaksisi sodda, toza va koddan kamroq foydalanishga olib keladi. Kotlin dasturlash tili murakkab sintaksisga emas, balki dolzarb muammoni hal qilishga ko'proq e'tibor qaratishga yordam beradi. Bundan tashqari, bitta loyihada Kotlin va Java tillaridan birgalikda foydalanish mumkin va bu uni kuchli qiladi. Kotlin dasturlash tilining afzalliklari:

- Java dasturlash tiliga qaraganda kamroq kod bilan ishlash imkonini beradi. Matn qanchalik kichik bo'lsa, unda kamroq xatolar mavjud bo'ladi.
- Kotlin dasturlash tili Java dasturlash tili bilan almashtirilishi mumkin, shuning uchun interfeysning turli qismlari turli tillarda yozilishi mumkin va yuqori samarali dasturlarni yaratishga yordam beradi.
- xavfsizlik, barcha sintaktik xatolar va obyektlarga noto'g'ri kirish bilan bog'liq xatolar dastur yaratish vaqtida topilishi va tuzatilishi mumkin. Bu sinov va testlashni osonlashtiradi.

• Kotlin dasturlari Java shablonlari va kutubxonalaridan foydalanadi. Kotlin dasturlash tilining kamchiliklari quyidagicha:

- dasturni yaratish tezligi ko'pincha tezdin juda sekingacha o'zgaradi.
- hali ishlab chiquvchilar orasida u qadar keng tarqalmagan, shuning uchun mutaxassislarni topish va nostandart xatolarni hal qilish bilan bog'liq muammolar bo'lishi mumkin.

2.3.2. iOS OT tillari

Swift dasturlash tili

iOS uchun mobil ilova ishlab chiqishda Swift dasturlash tilidan foydalanish mumkin. Swift dasturlash tili 2014-yilda taqdim etilgan va 2015-yilda ochiq manba deb e'lon qilingan. Swift dasturlash tili mobil ilova ishlab chiquvchilarni tezda hayratda qoldirdi. Swift dasturlash tili ayniqsa, yangi boshlovchi iOS dasturchilari orasida juda mashhur.

Apple Swift dasturlash tiliga bir qancha ajoyib funksiyalarni qo'shdi, masalan, soddalashtirilgan sintaksis, dasturchi xatolarini osongina aniqlash qobiliyati va hokazo. Apple kompaniyasining Swift dasturlash tilini targ'ib qilish bo'yicha ulkan sa'y-harakatlari shuni ko'rsatadiki, u ushbu yangi tilni mobil ilova ekotizimining asosiy dasturlash tiliga aylanishini xohlaydi. Swift dasturlash tilining afzalliklari quyidagicha:

- yuqori tezlik – C++ dasturlash tili darajasiga etadi.
- oson o'qilishi. Mantiqan, u ingliz tiliga o'xshaydi, shuningdek, oddiy sintaksis va kodga ega.
- Objective-C dasturlash tili bilan solishtirganda xavfsizlik yaxshilandi.

• koddagi xatolarni tuzatishning soddalashtirilgan usuli mavjudligi.

• yangilangan versiyaga avtomatik ravishda bog'langan va ilovaga birlashtirilgan kutubxonalar tufayli barqaror hisoblanadi.

- xavfsiz xotira boshqaruvini ta'minlaydi.

Swift dasturlash tili kamchiliklari quyidagicha:

- u rivojlanadi va o'zgaradi, shuning uchun ish sekinlashishi mumkin – versiyalar haqida ma'lumotni o'rganish va qo'llash kerak.
- Objective-C dasturlash tili fayllari bilan sinxronizatsiya ko'prigi loyihani qurishni sekinlashtiradi.

Objective-C dasturlash tili

Objective-C dasturlash tili iOS uchun asl dasturlash tili edi. Swift tili iOS-ning kelajakdagi rivojlanishi bo'lib, ko'plab ilg'or loyihalar hali ham Objective-C-ga tayanadi. Objective-C-dan Swift-ga o'tish biroz sekin bo'lishi kutilmoqda va ba'zi loyihalar uchun ikkalasi ham kerak bo'lishi mumkin.

Objective-C dasturlash tili afzalliklari quyidagicha:

- ishni soddalashtiradigan ko'plab hujjatlar mavjud.
- Swift tili bilan mos keladi.

Objective-C dasturlash tili kamchiliklari quyidagicha:

- Swift tili bilan solishtirganda yomon ishlashi.
- murakkab sintaksisiga egaligi.

Rust dasturlash tili

Rust dasturlash tili 2006 yilda C++ dasturlash tili tezligini Haskell dasturlash tilining mustahkamligi bilan birlashtirmoqchi bo'lgan dasturchi Graydon Hor tomonidan yaratilgan. Rust dasturlash tili kross-platforma ilovalarini ishlab chiquvchilar orasida eng mashhurlaridan biri hisoblanadi.

Rust dasturlash tili afzalliklari quyidagicha:

- segmentlash xatolari va ma'lumotlar sizib chiqishiga yo'l qo'ymaydigan xotira bilan xavfsiz sinxronizatsiya.
- kompilyatsiya vaqtidagi xatolar darhol ko'rinadi va tuzatishlar bo'yicha takliflar beriladi.
- tezligi bo'yicha C++ bilan solishtirish mumkin.
- kutubxonalar yordamida tarmoq ilovalarini tashkil qilish uchun ishonchli API.
- tizim bir vaqtning o'zida bir nechta hisob-kitoblarni amalga oshiradi va ularning bir-biri bilan o'zaro ta'sirini ta'minlaydi.

Rust dasturlash tili kamchiliklari quyidagicha:

- nisbatan yangi va tez rivojlanmoqda, shuning uchun tegishli adabiyotlar va mutaxassislar tanlovi yo'q.
- katta hajmdagi ma'lumotlarni mustaqil ravishda to'ldirishni talab qiluvchi va ishlab chiqish jarayonini sekinlashtiradigan qat'iy kompilyator.

2.3.3. Platformalararo tillari

JavaScript dasturlash tili

JavaScript-ning uzoq tarixi World Wide Webning boshlanishiga borib taqaladi. Web-ishlab chiquvchilarga o'z web-saytlarining foydalanuvchi interfeysini yaxshilashdan tortib to'liq web-ilovalarni yaratishgacha bo'lgan hamma narsani qilish imkonini beruvchi juda mashhur front-end va back-end tili.

Bugungi kunda Ionic 2 va React Native kabi mobil ishlab chiqish platformalari uchun maxsus ishlab chiqilgan bir nechta JavaScript versiyalari mavjud. Ushbu versiyalar va kutubxonalar bilan platformalararo mobil ilovalarni ishlab chiqish juda oson. Bu shuni anglatadiki, dasturning faqat bitta versiyasini ishlatish kerak va u iOS yoki Androidda ishlaydi.

TypeScript dasturlash tili

TypeScript JavaScript dasturlash tilining yuqori to'plamidir va ixtiyoriy statik yozishni qo'shish orqali yaxshi xavfsizlikni ta'minlaydi. Shuningdek, u keng ko'lamli ilovalarni ishlab chiqish uchun yaxshi yordam beradi. Microsoft tomonidan ishlab chiqilgan va qo'llab-quvvatlanadigan TypeScript ishlab chiquvchilarga NativeScript kabi versiyalar yordamida platformalararo mobil ilovalarni yozish imkonini beradi.

C# dasturlash tili

C# – bu Windows Mobile OT uchun mobil ilova ishlab chiqish tili hisoblanadi. Bu til C++ va Java dasturlash tiliga juda o'xshaydi. Microsoft o'zining arxitekturasini soddalashtirish va C++-ga o'xshash dizaynni saqlab qolish uchun ba'zi Java dasturlash tili xususiyatlarini qabul qildi. Shuningdek, u doimo do'stona va foydali bo'lgan katta va faol ishlab chiquvchilar hamjamiyatiga ega.

C# dasturlash tili afzalliklari quyidagicha:

- Windows qo'llab-quvvatlashga alohida e'tibor beradi, yangilanishlarni muntazam ravishda chiqaradi va xatolarni aniqlaydi, shuning uchun C# bilan qulay va tez ishlashi mumkin.
- ayrim tashkilotlar va individual ishlab chiquvchilar vositalardan bepul foydalanishlari mumkin.
- C# dasturlash tilida ishlash bilan bog'liq deyarli barcha savollarga javoblarni Internetda yoki professional jamoalarda topish mumkin.
- C# dasturlash tili bilan ishlash uchun asboblari va asboblarning katta to'plami faqat bitta tildan foydalanish imkonini beradi.
- foydalanilmayotgan obyektlardan xotirani avtomatik tozalash rejimi mavjud.
- dasturiy mahsulot yangi versiyaga yangilanganda ham to'g'ri ishlaydi.

C# dasturlash tili kamchiliklari:

- deyarli barcha operatsion tizimlarda ishlaydi, lekin baribir ustuvorlik Windows platformasiga asoslangan.
- faqat kichik firmalar, individual dasturchilar, startaplar va talabalar uchun bepul. Katta kompaniya uchun litsenziyalangan versiyani sotib olish katta xarajatni talab qiladi.

C dasturlash tili
TIOBE indeksidagi ikkinchi eng mashhur dasturlash tili bo'lib, Java dasturlash tili kabi uning hamjamiyati xatosiz kod yozish bo'yicha qimmatli maslahatlarni taklif qiladigan tajribali dasturchilar bilan to'la.

Bell Laboratoriyasida ishlayotgan Dennis Ritchi tomonidan yaratilgan C dasturlash tili - past darajadagi kompyuter operatsiyalarini bevosita boshqarish imkonini beruvchi keng qo'llaniladigan va kuchli til. Agar Android NDK (Native Development Kit) dan foydalaniladigan bo'lsa, C dasturlash tilini yaxshi bilish kerak bo'ladi.

C dasturlash tili afzalligi Java dasturlash tili bilan birgalikda kodni kamaytirish imkonini beradi, bu esa dasturning ishlashini tezlashtiradi. C dasturlash tili kamchiligi o'rganish qiyin va ilovani to'liq yaratishga imkon bermaydi, faqat kutubxonalarni ilovaga bog'laydi.

C++ dasturlash tili
C++ dasturlash tili bu C tilining kengaytmasi bo'lib, yuqori darajadagi funksiyalarga ega va obyektga yo'naltirilgan dasturlashni qo'llab-quvvatlaydi. C++ dasturlash tili ham Android NDK dasturchilarining sevimli tilidir. C++ tilidan Windows Mobile uchun mobil ilovalar ishlab chiqishda foydalanish mumkin. C++ dasturlash tili dasturiy ta'minotni ishlab chiqishda Java dasturlash tili bilan birga ishlaydi.

C++ dasturlash tili afzalliklari:

- Obyektga yo'naltirilgan dasturlashni, protsessual dasturlashni va umumiy dasturlarni qo'llab-quvvatlaydi.

• Objective-C bilan ishlashda unumdorlikni oshiradi.

C++ dasturlash tili kamchiliklari:

- to'liq ishlab chiqish uchun mo'ljallanmagan;
- o'rganish qiyin.

Python dasturlash tili

Python dasturlash tili - bu o'rganish va o'qish oson bo'lgan til. Til yaratuvchilari sintaksisni iloji boricha sodda va tushunarli qilish uchun qo'shimcha harakatlar qildilar. Bu boshlang'ich ishlab chiquvchilarga birinchi kundan boshlab yuqori mahsuldorlikni saqlab qolishga yordam beradi. Python tilida dastur kodini yozish o'zaro platformali mobil ilovalarni ishlab chiqish uchun Kivy kabi freymworkdan foydalanish mumkin.

Python dasturlash tili afzalliklari:

- ham lokal, ham web-ilovalar uchun mos;
- lokal interfeyslarni yaratish imkonini beradi;

- oson o'qiladigan sintaksis;
 - o'rganish oson.
- Python dasturlash tili kamchiliklari:
- rasmiy Android tili emas;
 - Kivy uni qo'llab-quvvatlamaydi;

Ruby dasturlash tili

Ruby dasturlash tili - bu Ada, C++, Perl, Python va Lisp ta'sirida yaratilgan obyektga yo'naltirilgan skript tili. RubyMotion - bu Ruby dasturlash tilida lokal va o'zaro platformali mobil ilovalarni ishlab chiqish uchun platforma.

Nazorat savollari:

1. Mobil operatsion tizim nima uchun ishlatiladi?
2. Symbian OS-ning afzalliklari va kamchiliklarini aytib bering.
3. Windows Mobile OT-ning afzalliklari va kamchiliklarini aytib bering.
4. Palm OS-ning afzalliklari va kamchiliklarini aytib bering.
5. iPhone OS-ning afzalliklari va kamchiliklarini aytib bering.
6. Android platforma qurilmasi nimadan iborat?
7. Android SDK nima?
8. Android uchun asosiy ishlab chiqish vositalarini tavsiflab bering.
9. Android emulyatorlarining afzalliklari va kamchiliklarini sanab o'ting.
10. Hozirda platformaning qaysi versiyasi eng ommabop?
11. Ilova turlarini tavsiflab bering.
12. Ilova xususiyatlarini qanday o'zgartirish mumkin?
13. Android ilovalari arxitekturasi qanday?
14. Fon ilovalariga tavsif bering.
15. Aralash ilovalar qanday ishlaydi?


```

Box(int a, int b) {
    width = a;
    height = b;
    depth = 10;
}
// quti hajmini hisoblash
int getVolume() {
    return width * height * depth;
}

```

Agar konstruktor aniq belgilanmagan bo'lsa ham, Java Virtual Mashina albatta uni yaratadi (bo'sh konstruktorni). Java (shuningdek, C++) statik kalit so'z yordamida ko'rsatilgan statik usullardan (inglizcha statik usul – dasturlash nazariyasida ular sinf usullari deb ham ataladi) foydalanadi. Statik maydonlar (sinf o'zgaruvchilari) C++ tilidagi kabi ma'noga ega; har bir bunday maydon sinfning mulki hisoblanadi, shuning uchun statik maydonlarga kirish uchun tegishli sinf misollarini yaratish shart emas. Misol uchun, *Math* sinfida amalga oshirilgan matematik funksiyalar bu sinfning statik usullaridir. Shuning uchun, quyidagicha yozish mumkin:

```

double x = Math.sin();
o'miga
Math m = new Math();
juft x = m.sin();

```

Statik usullar obyektlardan (sinf misollaridan) mustaqil ravishda mavjud bo'lganligi sababli ular berilgan sinfning oddiy (statik bo'lmagan) maydonlari va usullariga kirish imkoniga ega emas. Xususan, statik usulni amalga oshirishda ushbu identifikatordan foydalanmaslik kerak. Yakuniy kalit so'z maydon, usul yoki sinfni tavsiflashda turli xil ma'nolarga ega.

1. Sinfning *final* maydonini tavsiflash vaqtida yoki sinf konstruktorida initsializatsiya qilinadi. Keyinchalik uning qiymatini o'zgartirib bo'lmaydi. Agar statik sinf maydoni yoki o'zgaruvchisi doimiy ifoda bilan ishga tushirilsa, u kompilyator tomonidan nomlangan o'zgarmas sifatida ko'rib chiqiladi; bu holda ularning qiymati *switch* operatorlarida (*int* tipidagi o'zgarmaslar uchun), shuningdek *if* operatori bilan foydalanilganda shartli kompilyatsiya uchun (mantiqiy tipdagi o'zgarmaslar uchun) ishlatilishi mumkin.

2. Mahalliy (lokal) o'zgaruvchilarning qiymatlarini, shuningdek *final* kalit so'z bilan belgilangan usul parametrlarini tayinlashdan keyin o'zgartirib bo'lmaydi. Biroq, ularning qiymatlari anonim sinflar ichida ishlatilishi mumkin.

3. *Final* so'zi bilan belgilangan sinf usulini meros orqali qayta e'lon qilib bo'lmaydi.

4. *Final* sinfning merosxo'rlari bo'lishi mumkin emas. Java-da aniq *static*, *final* yoki *private* deb e'lon qilinmagan usullar C++ terminologiyasida virtual hisoblanadi: asosiy va voris sinflarda boshqacha belgilangan usulni chaqirish uchun har doim bajarilish vaqtini tekshirish amalga oshiriladi.

Java-da mavhum usul (*abstract* modifikator) – bu parametrlar va qaytish tipi ko'rsatilgan, ammo tanasi ko'rsatilmagan usul hisoblanadi. Abstrakt usul voris sinflarda aniqlanadi. C++ tilidagi mavhum usulning analogi sof virtual funksiyadir (*pure virtual function*). Sinf mavhum usullarni tasvirlay olishi uchun sinfning o'zi ham abstrakt deb e'lon qilinishi kerak. Abstrakt sinf obyektlarini yaratib bo'lmaydi.

Java-da abstraksiyaning eng yuqori darajasi interfeysdir (*interface*). Barcha interfeys usullari mavhum sanaladi; mavhum *abstract* ifodalovchisi talab qilinmaydi. Java tilidagi interfeys sinf hisoblanmaydi, garchi u aslida butunlay mavhum sinf bo'lsa ham. Sinf boshqa sinfni meros qilib olishi yoki kengaytirishi (*extends*) yoki interfeysni amalga oshirishi (*implements*) mumkin. Bundan tashqari, interfeys boshqa interfeysni meros qilib olishi yoki kengaytirishi mumkin. Java-da sinf bir nechta sinfdan meros bo'lolmaydi, lekin u bir nechta interfeyslarni amalga oshirishi mumkin. Interfeyslarni bir nechta meros qilib olish taqiqlanmaydi, ya'ni bitta interfeys bir nechtadan meros bo'lishi mumkin.

Interfeyslardan usul parametrlari turlari sifatida foydalanish mumkin. Interfeyslarni ekzemplarlarini yaratib bo'lmaydi. Java-da amalga oshirish usullari mavjud bo'lmagan, ammo JVM tomonidan maxsus tarzda ishlov beriladigan interfeyslar mavjud:

- *java.lang.Cloneable*;
- *java.io.Serializable*;
- *java.util.RandomAccess*;
- *java.rmi.Remote*.

3.2. Sinflar va obyektlar

3.2.1. Sinf tavsifi

Java obyektga yo'naltirilgan tildir, shuning uchun "sinf" va "obyekt" kabi tushunchalar unda asosiy rol o'ynaydi. Har qanday Java dasturini o'zaro ta'sir qiluvchi obyektlar to'plami sifatida qarash mumkin.

Obyektning shabloni yoki tavsifi sinfdir va obyekt shu sinfning namunasini ifodalaydi. Misol uchun, har bir inson haqida qandaydir

tasavvurga ega bo'lish mumkin - ikki qo'li, ikki oyog'i, boshi, tanasi va boshqalar.

Sinf *class* kalit so'zi yordamida aniqlanadi:

```
class Graj { }
```

Misolda *sinf Graj* deb ataladi. Sinf nomidan keyin figurali qavslar qo'yiladi, ular orasiga *sinf* tanasi - ya'ni uning maydonlari va usullari joylashtiriladi.

Har qanday obyekt ikkita asosiy xususiyatga ega bo'lishi mumkin: holat - obyekt saqlaydigan ba'zi ma'lumotlar va xatti-harakatlar - obyekt bajarishi mumkin bo'lgan harakatlar.

Obyekt holatini sinfda saqlash uchun maydonlar yoki *sinf* o'zgaruvchilari ishlatiladi. Usullar sinfdagi obyektning harakatini aniqlash uchun ishlatiladi. Misol uchun, shaxsni ifodalovchi *Graj* sinfi quyidagi tarzda ega bo'lishi mumkin:

```
class Graj {
    String name; // ismi
    int yosh; // yoshi
    void displayInfo() {
        System.out.printf("Ismi: %s \tYoshi: %d\n", name, yosh);
    }
}
```

Graj sinfi ikkita maydonni belgilaydi: *name* odamning ismini, *yosh* esa yoshini bildiradi, shuningdek *displayInfo()* usuli ham aniqlangan, u hech narsani qaytarmaydi va ma'lumotlarni konsolga chop etadi.

Endi ushbu sinfdan foydalangan holda quyidagi dasturni tuzib olish mumkin:

```
public class Dastur {
    public static void main(String[] args) {
        Graj Karl;
    }
}

class Graj {
    String name; // ismi
    int yosh; // yoshi
    void displayInfo() {
        System.out.printf("Name: %s \tYoshi: %d\n", name, yosh);
    }
}
```

Qoida tariqasida sinflar turli fayllarda aniqlanadi. Misolda, soddalik uchun bitta faylda ikkita *sinf* aniqlangan. Shuni ta'kidlash kerakki, bu holda faqat bitta *sinf* umumiy modifikatorga ega bo'lishi mumkin (misolda, bu *Dastur* sinfi) va kod faylining o'zi ushbu *sinf* nomi bilan nomlanishi kerak, ya'ni misolda fayl *Dastur.java* deb nomlanishi kerak.

Sinf yangi tipni ifodalaydi, shuning uchun ushbu tipni ifodalovchi o'zgaruvchilarni aniqlash mumkin. Shunday qilib, bu erda asosiy usulda *Graj* sinfini ifodalovchi *karl* o'zgaruvchisi aniqlanadi. Ammo bu o'zgaruvchi hech qanday obyektga ishora qilmaydi va sukut bo'yicha u *null* qiymatiga ega. Umuman olganda, uni hali ishlatish mumkin emas, shuning uchun birinchi navbatda *Graj* sinfining obyektini yaratish kerak.

3.2.2. Konstruktorlar

Oddiy usullardan tashqari, sinflar konstruktorlar deb ataladigan maxsus usullarni belgilashi mumkin. Konstruktorlar berilgan sinfning yangi obyektini yaratilganda chaqiriladi. Konstruktorlar obyektini ishga tushirishni amalga oshiradilar. Agar sinfda konstruktor aniqlanmagan bo'lsa, u holda ushbu *sinf* uchun avtomatik ravishda parametrsiz konstruktor yaratiladi. Yuqorida tavsiflangan *Graj* sinfida konstruktorlar mavjud emas. Shuning uchun, u uchun avtomatik ravishda standart konstruktor yaratiladi va undan *Graj* obyektini yaratish uchun foydalanish mumkin. Xususan, bitta obyekt yaratish quyidagicha amalga oshiriladi:

```
public class Dastur {
    public static void main(String[] args) {
        Graj karl = new Graj(); // obyekt yaratish
        karl.displayInfo();
        // ism va yoshni o'zgartirish
        karl.name = "Said";
        karl.yosh = 20;
        karl.displayInfo();
    }
}

class Graj {
    String name; // ismi
    int yosh; // yoshi
    void displayInfo() {
        System.out.printf("Name: %s \tYoshi: %d\n", name, yosh);
    }
}
```

new Graj() ifodasi *Graj* obyektini yaratish uchun ishlatiladi. *New* operatori xotirani *Graj* obyektini yaratish uchun ajratadi. Keyin standart konstruktor chaqiriladi, u hech qanday parametrlarni olmaydi. Natijada, ushbu ifoda bajarilgandan so'ng, xotirada *Graj* obyektining barcha ma'lumotlari saqlanadigan bo'lim ajratiladi. *Karl* o'zgaruvchisi esa yaratilgan yangi obyektga havola oladi. Agar konstruktor obyekt o'zgaruvchilari qiymatlarini ishga tushirmasa, ular standart qiymatlarini oladilar.

O'zgaruvchining raqamli turlari uchun bu 0 raqami, *string* tipi va sinflar uchun esa bu *null* (ya'ni, aslida qiymatning yo'qligi) qiymatidir. Obyektni yaratgandan so'ng, *Graj* obyekt o'zgaruvchilariga *karl* o'zgaruvchisi orqali kirish va ularning qiymatlarini belgilash yoki olish mumkin, masalan,

```
karl.name = "Said";
Natijada, konsolda quyidagilarni ko'rish mumkin:
Name: null   Yoshi: 0
Name: Karl  Yoshi: 20
Agar obyekt yaratishda ba'zi bir mantiqni bajarish kerak bo'lsa,
masalan, sinf maydonlari qandaydir o'ziga xos qiymatlarni qabul qilsa, u
holda sinfda konstruktorlarni e'lon qilish mumkin. Misol uchun:
public class Dastur {
    public static void main(String[] args) {
        Graj sanjar = new Graj(); // parametrsiz birinchi konstruktorni
                                // chaqirish
        sanjar.displayInfo();
        Graj karl = new Graj("Said"); // bitta parametr bilan ikkinchi
                                    // konstruktorni chaqirish
        karl.displayInfo();
        Graj sam = new Graj("Suxrob", 22);
        // ikkita parametr bilan uchinchi
        // konstruktorni chaqirish
        sam.displayInfo(); } }
class Graj {
    String name; // ismi
    int yosh; // yoshi
    Graj() {
        name = "Undefined";
        yosh = 18; }
    Graj(String n) {
        name = n;
        yosh = 18; }
    Graj(String n, int a) {
        name = n;
        yosh = a; }
    void displayInfo(){
        System.out.printf("Ismi: %s \tYosh: %d\n", name, yosh); }
}
```

```
Natija:
Ismi: Undefined   Yosh: 18
Ismi: Said        Yosh: 20
Ismi: Suxrob     Yosh: 22
```

3.2.3. *This* kalit so'z

This kalit so'zi sinfning joriy nusxasiga havolani ifodalaydi. Ushbu kalit so'z orqali obyektning o'zgaruvchilari, usullariga kirish, shuningdek uning konstruktorlarini chaqirish mumkin. Misol uchun:

```
public class Dastur {
    public static void main(String[] args) {
        Graj undef = new Graj();
        undef.displayInfo();
        Graj karl = new Graj("Karl");
        karl.displayInfo();
        Graj sam = new Graj("Sam", 25);
        sam.displayInfo(); } }
class Graj {
    String name; // ismi
    int yosh; // yoshi
    Graj() { this("Undefined", 18); }
    Graj (String name) { this (name, 18); }
    Graj(String name, int yosh) {
        this.name = name;
        this.yosh = yosh; }
    void displayInfo(){
        System.out.printf("Ismi: %s \tYoshi: %d\n", name, yosh); }
}
```

Uchinchi konstruktorda parametrlar sinf maydonlari bilan bir xil nomlanadi. Maydonlar va parametrlarni chegaralash uchun *this* kalit so'zi ishlatiladi:

```
this.name = name;
Shunday qilib, misolda name parametrining qiymati name maydoniga tayinlanganligini ko'rish mumkin.
```

Bundan tashqari, bir xil amallarni bajaradigan uchta konstruktor mavjud: *name* va *yosh* maydonlarini o'rnatadi. Takrorlashni oldini olish uchun undan sinf konstruktorlaridan birini chaqirish va uning parametrlari uchun kerakli qiymatlarni berish uchun foydalanish mumkin:

```
Graj (String name) {
```

```
this(name, 18); }
Natijada, dasturning natijasi avvalgi misoldagi kabi bo'ladi.
```

3.2.4. Initsializatorlar

Konstruktorga qo'shimcha ravishda, obyektни dastlabki ishga tushirish obyekt initsializatori yordamida amalga oshirilishi mumkin. Initsializator har qanday konstruktordan oldin bajariladi. Ya'ni, barcha konstruktorga uchun umumiy bo'lgan kodni initsializatorga joylashtirish mumkin:

```
public class Dastur {
    public static void main(String[] args) {
        Graj undef = new Graj();
        undef.displayInfo();
        Graj karl = new Graj("Karl");
        karl.displayInfo(); }
class Graj {
    String name; // ismi
    int yosh; // yoshi
    /*initsializator blokining boshlanishi*/
    {
        name = "Undefined";
        yosh = 18; }
    /*initsializator blokining oxiri*/
    Graj(){}
    Graj(String name){
        this.name = name; }
    Graj(String name, int yosh){
        this.name = name;
        this.yosh = yosh; }
    void displayInfo(){
        System.out.printf("Ismi: %s \tYosh: %d\n", name, yosh); } }
```

Konsol chiqishi:

Ismi: Undefined Yosh: 18

Ismi: Karl Yosh: 18

3.3. Java dasturlash tilida mobil ilovalarni ishlab chiqish

Kiritishni boshqarish elementlari mobil ilovaning foydalanuvchi interfeysidagi interaktiv komponentlar majmuasi hisoblanadi. Android tugmalar, matn maydonlari, qidirish satrlari, belgilash katakchalari,

kattalashtirish tugmalari, almashtirish tugmalari va boshqalar kabi foydalanuvchi interfeysida foydalanish mumkin bo'lgan keng ko'lami boshqaruv elementlarini taqdim etadi. Kiritishni boshqarish elementlariga quyidagilar kiradi:

TextView

TextView elementi matni ekranda oddiy qilib ko'rsatish uchun mo'ljallangan. U shunchaki matni tahrirlash imkoniyatisiz ko'rsatadi. Uning asosiy atributlaridan ba'zilari quyidagilar hisoblanadi:

- *android:text*: element matnini o'rnatadi;
- *android:textSize*: matn balandligini o'rnatadi, *sp* balandlik birligi sifatida ishlatiladi;
- *android:background*: elementning fon rangini o'n oltilik sanoq tizimidagi rang sifatida yoki rang resursi sifatida o'rnatadi;
- *android:textColor*: matn rangini o'rnatadi;
- *android:textAllCaps*: agar *true* bo'lsa, barcha belgilarni matn bosh harflari bilan yozadi;
- *android:textDirection*: matn yo'nalishini o'rnatadi. Standart yo'nalish chapdan o'ngga, lekin *rtl* qiymatidan foydalanib yo'nalishni o'ngdan chapga o'rnatish mumkin;
- *android:textAlignment*: matnni tekislashni o'rnatadi. U quyidagi qiymatlarni olishi mumkin:
 - *center*: markazni tekislash;
 - *textStart*: chap tomondan tekislash;
 - *textEnd*: o'ng tomondan tekislash;
 - *viewStart*: chapdan o'ngga matn yo'nalishi chapga tekislangan, o'ngdan chapga yo'nalish o'ngga tekislangan bo'lsa;
 - *viewEnd*: matnni chapdan o'ngga tekislash o'ngga, o'ngdan chapga tekislash chapga tekislangan bo'lsa;
- *android:fontFamily*: shrift turini o'rnatadi. U quyidagi qiymatlarni olishi mumkin: *monospace*, *serif*, *serif-monospace*, *sans-serif*, *sans-serif-condensed*, *sans-serif-smallcaps*, *sans-serif-light*, *casual*, *cursive*.
- *android:autoLink* bir nechta qiymatlarni qabul qilishi mumkin:
 - *none*: barcha havolalarni o'chiradi;
 - *web*: barcha web havolalarni o'z ichiga oladi;
 - *email*: elektron pochta manzillariga havolalarni o'z ichiga oladi;
 - *phone*: telefon raqamlariga havolalarni o'z ichiga oladi;
 - *map*: xarita havolalarini o'z ichiga oladi;
 - *all*: yuqoridagi barcha havolalarni o'z ichiga oladi.

Ya'ni, `android:autoLink="web"` ni o'rnatishda, agar matnda url manzili eslatib o'tilgan bo'lsa, u holda ushbu manzil ta'kidlanadi va uni bosganingizda web-brauzerga o'tish amalga oshiriladi, ushbu manzildagi sahifa ochiladi. Oldinga siljish bilan shartlarni birlashtirish mumkin, bu holatda quyidagicha beriladi:

`android:autoLink="web,email"`
TextView komponentasining dasturdagi ko'rinishi quyidagicha:

```
<TextView
    android:layout_height="wrap_content"
    android:layout_width="0dp"
    android:layout_margin="10dp"
    android:text="Hello World"
    android:textAlignment="textEnd"
    android:textSize="26sp"
    android:background="#e8eaf6"
    android:textColor="#5c6bc0"/>
```

Quyidagi misolda *TextView* elementining ishlatilishi ko'rsatilgan. Misolda *TextView* obyektini yaratilgan. *TextView* obyektini orqali `setBackground-color()`, `setTextColor()`, `setAllCaps()`, `setTextAlignment()`, `setText()`, `setTypeface()`, `setTextSize()` usullari chaqirilgan va bu usullar yordamida *TextView* obyektiga qiymatlar o'rnatilgan.

```
package com.example.layoutapp;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.ViewGroup;
import android.widget.LinearLayout;
import android.widget.TextView;
import android.graphics.Typeface;
public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        LinearLayout linearLayout = new LinearLayout(this);
        TextView textView1 = new TextView(this);
        // fon rangini o'rnatish
        textView1.setBackgroundColor(0xffe8eaf6);
        // matn rangini o'rnatish
        textView1.setTextColor(0xff5c6bc0);
        // barcha harflarni bosh harflar bilan yozadi
```

```
textView1.setAllCaps(true);
// matnni markazga tekislaydi
textView1.setTextAlignment(TextView.TEXT_ALIGNMENT_
CENTER);
// matnni o'rnatadi
textView1.setText("Android Nougat 7");
// shriftni o'rnatadi
textView1.setTypeface(Typeface.create("casual",
Typeface.NORMAL));
// matn balandligini o'rnatadi
textView1.setTextSize(26);
LinearLayout.LayoutParams layoutParams = new
LinearLayout.LayoutParams
(ViewGroup.LayoutParams.MATCH_PARENT,
ViewGroup.LayoutParams.WRAP_CONTENT);
// chetki chegaralarni o'rnatadi
layoutParams.setMargins(20,20,20,20);
// o'lchamlarni o'rnatadi
textView1.setLayoutParams(layoutParams);
linearLayout.addView(textView1);
setContentView(linearLayout); }
}
```

EditText

EditText elementi *TextView* sinfining ost sinfidir. *EditText* elementi matn maydoni bo'lib, matnni kiritish va tahrirlash imkoniyati bilan ifodalanadi. *EditText*-da *TextView*-dagi kabi barcha funksiyalardan foydalanish mumkin.

TextView-da ko'rib chiqilmagan atributlardan *android: hint* atributini alohida ko'rsatish kerak. *EditText* elementi bo'sh bo'lsa, maslahat sifatida ko'rsatiladigan matnni o'rnatish imkonini beradi. Bundan tashqari, klaviaturadan biror bir matnni kiritish uchun sozlash imkonini beruvchi *android:inputType* atributidan foydalanish mumkin. Xususan, uning qiymatlariga quyidagilar kiradi:

text: bir qatorli matn kiritish uchun oddiy maydon;
textMultiLine: ko'p qatorli matn maydoni;
textEmailAddress: @ belgisiga ega bo'lgan oddiy maydon elektron pochtaga yo'naltirilgan;

textUri: belgisi mavjud bo'lgan hamda Internet manzillarini kiritish uchun yo'naltirilgan oddiy maydon;

textPassword: parol kiritish uchun maydon;

textCapWords: yozayotganda, kiritilgan so'zning birinchi belgisi katta harfni, qolganlari kichik harfni ifodalaydi;

number: raqamli maydon;

phone: oddiy telefon uslubidagi maydon;

date: sanani kiritish maydoni;

time: vaqtni kiritish uchun maydon;

datetime: sana va vaqtni kiritish uchun maydon.

EditText komponentasining dasturdagi ko'rinishi quyidagicha:

```
<EditText
    android:id="@+id/name"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:hint="Podskazka kiriting"/>
```

EditText elementi qo'llanilgan holda yaratilgan mobil ilovaga misol quyida keltirilgan:

```
package com.example.layoutapp;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.text.Editable;
import android.text.TextWatcher;
import android.widget.EditText;
import android.widget.TextView;
public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        EditText editText = (EditText) findViewById(R.id.editText);
        editText.addTextChangedListener(new TextWatcher() {
            public void afterTextChanged(Editable s) {}
            public void beforeTextChanged(CharSequence s, int start, int
count, int after) {}
            public void onTextChanged(CharSequence s, int start,
                int before, int count) {
                TextView textView =
                findViewById(R.id.textView);
                textView = (TextView)
```

```
textView.setText(s); } });
```

```
} }
```

Bu yerda *addTextChangedListener()* usuli matn kiritish tinglovchisini, *TextWatcher* obyektini *EditText* elementiga qo'shadi. Uni ishlatish uchun uchta usulni amalga oshirish kerak, lekin faqat matn o'zgaranda chaqiriladigan *onTextChanged()* usulini amalga oshirish kerak. Kiritilgan matn *CharSequence* parametri sifatida ushbu usulga uzatiladi. Usulning o'zida ushbu matn *TextView* elementiga o'tkazilgan.

Button

Button komponentasi ko'p ishlatiladigan elementlardan biri bo'lib, ular *android.widget.Button* sinfi bilan ifodalanadi. Tugmalarning asosiy xususiyati bosish orqali foydalanuvchi bilan muloqot qilish qobiliyatidir. Tugmalarda o'rnatilishi mumkin bo'lgan ba'zi asosiy atributlar quyida keltirilgan:

text: tugmadagi matnni o'rnatadi;

textColor: tugmadagi matn rangini o'rnatadi;

font: tugmaning fon rangini o'rnatadi;

textAllCaps: agar *true* bo'lsa, matnni katta harfga o'rnatadi. Standart qiymati *true*;

onClick: tugmani bosish qayta ishlovchisini (obrabotchik) o'rnatadi.

Button komponentasining dasturdagi ko'rinishi quyidagicha:

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Kiritish"
    android:onClick="sendMessage"
    app:layout_constraintTop_toBottomOf="@+id/editText"
    app:layout_constraintLeft_toLeftOf="parent" />
```

Button komponentasi qo'llanilgan holda yaratilgan mobil ilovaga misol quyida keltirilgan:

```
package com.example.layoutapp;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.EditText;
import android.widget.TextView;
public class MainActivity extends AppCompatActivity {
    @Override
```

```

protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.Activity_main); }
public void sendMessage(View view) {
    TextView textView = (TextView)
    findViewById(R.id.textView);
    EditText editText = (EditText) findViewById(R.id.editText);
    textView.setText("Xush kelibsiz, " + editText.getText()); }
}

```

Klik bilan ishlash usulini yaratishda quyidagilarni hisobga olish lozim:

- usul umumiy modifikator bilan e'lon qilinishi kerak;
- void-ni qaytarishi kerak.

Parametr sifatida View obyektini olish lozim. Ushbu View obyektini bosilgan tugmani ifodalaydi.

```

package com.example.layoutapp;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.LinearLayout;
import android.widget.TextView;
public class MainActivity extends AppCompatActivity {
    EditText editText; TextView textView;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        LinearLayout linearLayout = new LinearLayout(this);
        linearLayout.setOrientation(LinearLayout.VERTICAL);
        textView = new TextView(this);
        textView.setLayoutParams(new
        LinearLayout.LayoutParams(
            LinearLayout.LayoutParams.MATCH_PARENT,
            LinearLayout.LayoutParams.WRAP_CONTENT
        ));
        linearLayout.addView(textView);
        editText = new EditText(this);
        editText.setHint("Ismni kiriting");
    }
}

```

```

editText.setLayoutParams(new LinearLayout.LayoutParams(
    LinearLayout.LayoutParams.MATCH_PARENT,
    LinearLayout.LayoutParams.WRAP_CONTENT ));
linearLayout.addView(editText);
Button button = new Button(this);
button.setText("CLICK");
button.setLayoutParams(new LinearLayout.LayoutParams(
    LinearLayout.LayoutParams.WRAP_CONTENT,
    LinearLayout.LayoutParams.WRAP_CONTENT ));
linearLayout.addView(button);
button.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {
        textView.setText("Xush kelibsiz, " + editText.getText());
    }
});
setContentView(linearLayout);
}
}

```

Checkbox

Checkbox elementi – bu belgilangan yoki belgilanmagan holatda bo'lishi mumkin bo'lgan tasdiqlash elementidir. Belgilash katakchalari bir nechta qiymatlardan bir nechtasini tanlash imkonini beradi.

android:onClick atributi, oddiy tugmalar holatida bo'lgani kabi, belgilash katakchasi uchun bosish moslamasini o'rnatishga imkon beradi.

CheckBox komponentasining dasturdagi ko'rinishi quyidagicha:

```

<CheckBox android:id="@+id/enabled"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Belgilash"
    android:textSize="26sp"
    android:onClick="onCheckboxClicked"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/selection"/>

```

CheckBox komponentasi qo'llanilgan holda yaratilgan mobil ilovaga misol quyida keltirilgan:

```

package com.example.layoutapp;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.CheckBox;

```

```

import android.widget.TextView;
public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
    public void onCheckboxClicked(View view) {
        CheckBox language = (CheckBox) view;
        boolean checked = language.isChecked();
        TextView selection = (TextView)
        findViewById(R.id.selection);
        switch(view.getId()) {
            case R.id.java:
                if (checked) {
                    selection.setText("Java");
                }
                break;
            case R.id.javascript:
                if (checked) {
                    selection.setText("JavaScript");
                }
                break;
        }
    }
}

```

Bosilgan belgilash katakchasi *onCheckboxClicked* bosish qayta ishlovchisiga parametr sifatida o'tkaziladi. *Checkbox* har bosilganda qayta ishlovchi ishga tushadi. Ya'ni, katakchani belgilaganda ham, belgini olib tashlaganda ham *isChecked()* usulidan foydalanib, belgilash katakchasi tanlangan yoki yo'qligini bilib olish mumkin – bu holda usul *true* ni qaytaradi. *onCheckboxClicked* usulining dasturdagi ko'rinishi quyidagicha:

```

public void onCheckboxClicked(View view) {
}

```

Toast suzuvchi oynalari

Android oddiy bildirishnomalarni yaratish uchun *Toast* sinfidan foydalanadi. Aslida *Toast* qisqa vaqt davomida ko'rsatiladigan ba'zi matnli qalqib chiquvchi oynani ifodalaydi. *Toast* obyektini *xml* belgilash kodida, masalan, *activity_main.xml* faylida yaratib bo'lmaydi. *Toast* faqat java kodida ishlatilishi mumkin. Qalqib chiquvchi oynani yaratish uchun *Toast.makeText()* usuli qo'llaniladi, unga uchta parametr uzatiladi: joriy kontekst (joriy *Activity* obyekt), ko'rsatilgan matn va oynani ko'rsatish vaqti. Oynani ko'rsatish vaqti sifatida butun son qiymatini – millisekundlar

sonini yoki o'rnatilgan *Toast.LENGTH_LONG* (3500 millisekund) va *Toast.LENGTH_SHORT* (2000 millisekund) o'zgarmlaridan foydalanish mumkin. Oynaning o'zini ko'rsatish uchun *show()* usuli chaqiriladi. *Toasts* lar oyna interfeysining pastki qismida, markazda ko'rsatiladi. Lekin *setGravity()* va *setMargin()* usullari yordamida oynada akslanish o'rini sozlash mumkin. *setGravity()* usulining birinchi parametri *Toast* konteynerning qaysi qismida joylashtirilishi kerakligini belgilaydi, ikkinchi va uchinchi parametrlar mos ravishda gorizont va vertikal ravishda ushbu pozitsiyadan chekinishlarni o'rnatadi. *setMargin()* usuli ikkita parametрни oladi: konteynerning chap chetidan chekka konteyner kengligining ulushi sifatida va yuqori chetidan konteyner uzunligining foizi sifatida.

Toast-ning dasturdagi ko'rinishi quyidagicha:

```

public void onClick(View v) {
    Toast toast = Toast.makeText(this, "Next
    clicked!", Toast.LENGTH_LONG);
    toast.show();
}

```

Snackbar komponentasi

Snackbar elementi biroz *Toast*-ga o'xshaydi: u qalqib chiquvchi xabarlarni ham ko'rsatishga imkon beradi, ammo xabarlar ekran kengligiga mos ravishda cho'ziladi. *Snackbar* elementi *make()* usuli yordamida yaratiladi, unga 3 ta parametr uzatiladi: qalqib chiquvchi xabari birlashtirilgan *View* obyekt, xabarning o'zi qator sifatida va xabar qancha vaqt ko'rsatilishini belgilaydigan parametr. Oxirgi parametr raqamli qiymatni olishi mumkin - millisekundlar soni yoki 3 ta o'zgarmlardan birini: *Snackbar.LENGTH_INDEFINITE* (cheklanmagan vaqt uchun ko'rsatish), *Snackbar.LENGTH_LONG* (uzoq akslanish) yoki *Snackbar.LENGTH_SHORT* (qisqa akslanish). *Snackbar* elementi yaratilgandan so'ng *show()* usuli yordamida ko'rsatiladi. *Snackbar* komponentasining dasturdagi ko'rinishi quyidagicha:

```

public void onClick(View view){
    Snackbar.make(view, "Hello Android",
    Snackbar.LENGTH_LONG).show();
}

```

ToggleButton komponentasi

ToggleButton komponentasi xuddi *CheckBox* elementi kabi, ikkita holatda bo'lishi mumkin: belgilangan va belgilanmagan, har bir holat uchun

o'z matnini alohida o'rnatish mumkin. *ToggleButton* komponentasining atributlari quyidagicha:

- *android:textOn*: tugma matnini belgilangan holatda o'rnatadi;
- *android:textOff*: tugma matnini belgilanmagan holatda o'rnatadi.

Xuddi boshqa tugmalar kabi elementni bosishni *onClick* hodisasi yordamida boshqarish mumkin. Masalan, *ToggleButton* komponentasining dasturdagi ko'rinishi quyidagicha:

```
<ToggleButton
    android:id="@+id/toggle"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textOn="Yoqilgan"
    android:textOff="O'chirilgan"
    android:onClick="onToggleClicked"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
```

ToggleButton komponentasining qo'llanilgan holda yaratilgan mobil ilovaga misol quyida keltirilgan:

```
package com.example.viewapp;
import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Toast;
import android.widget.ToggleButton;
public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
    public void onToggleClicked(View view) {
        // tugma yoqilganligi yoki yoqilmaganligini tekshirish
        boolean on = ((ToggleButton) view).isChecked();
        if (on) {
            // agar yoqilgan bo'lsa, bajariladigan harakatlar
            Toast.makeText(this, "Chiroq yoqilgan",
                Toast.LENGTH_LONG).show();
        } else {
```

```
// agar o'chirilgan bo'lsa, bajariladigan harakatlar
    Toast.makeText(this, "Chiroq o'chirilgan",
        Toast.LENGTH_LONG).show(); } }
```

RadioButton komponentasi

RadioButton komponentasi radiotugmalar guruhini taqdim etadi, u *RadioButton* sinfi bilan ifodalanadi. Biroq, *CheckBox*-dan farqli, radio tugmalar guruhidan faqat bitta radiotugmani tanlash mumkin.

Tanlash uchun radiotugmalar ro'yxatini yaratish uchun avvalo barcha radiotugmalarni o'z ichiga olgan *RadioGroup* obyektini yaratish kerak. *RadioButton* komponentasining dasturdagi ko'rinishi quyidagicha:

```
<RadioGroup
    android:id="@+id/radios"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/selection">
    <RadioButton android:id="@+id/java"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Java"
        android:onClick="onRadioButtonClicked"/>
</RadioGroup>
```

DatePicker komponentasi

DatePicker komponentasi sana tanlash vositasini ifodalaydi. Uning atributlari orasida quyidagilar mavjud:

- *android:calendarTextColor*: kalendar matni rangini o'rnatadi;
- *android:calendarViewShown*: kalendar ko'rinishi ko'rsatilishini

belgilaydi;

- *android:datePickerMode*: sana tanlash rejimini o'rnatadi;
- *android:dayOfWeekBackground*: hafta kunining fon rangini tanlash panelini o'rnatadi;

- *android:endYear*: ko'rsatilgan oxirgi yilni belgilaydi;

- *android:firstDayOfWeek*: haftaning birinchi kunini belgilaydi;

- *android:headerBackground*: tanlangan sana satri uchun fon rangini

o'rnatadi;

- `android:maxDate`: mm/dd/yyyy formatida ko'rsatiladigan maksimal sanani o'rnatadi;
- `android:minDate`: mm/dd/yyyy formatida ko'rsatiladigan minimal sanani o'rnatadi;
- `android:spinnersShown`: spinner vidjetda ko'rsatilishini belgilaydi;
- `android:startYear`: ko'rsatiladigan boshlang'ich yilni belgilaydi;
- `android:yearListSelectorColor`: yil tanlash maydoni uchun rangni o'rnatadi.

`DatePicker` komponentasi usullari quyidagilarni o'z ichiga oladi:

- `int getDayOfMonth()`: tanlangan kunning sonini qaytaradi;
- `int getMonth()`: tanlangan oyning sonini qaytaradi (0 dan 11 gacha);
- `int getYear()`: tanlangan yil raqamini qaytaradi;
- `void init(int year, int monthOfYear, int dayOfMonth, DatePicker.OnDateChangedListener onDateChangedListener)`: boshlanish sanasini belgilaydi. Oxirgi parametr tanlangan sanani o'zgartirish tinglovchisini o'rnatadi;

- `void setOnDateChangedListener(DatePicker.OnDateChangedListener onDateChangedListener)`: tanlangan sanani o'zgartirish tinglovchisini o'rnatadi;
- `void setFirstDayOfWeek(int firstDayOfWeek)`: haftaning birinchi kunini belgilaydi;
- `void updateDate(int year, int month, int dayOfMonth)`: tanlangan sanani dasturiy yangilaydi.

`DatePicker` komponentasining dasturdagi ko'rinishi quyidagicha:

```
<DatePicker android:id="@+id/datePicker"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/dateTextView" />
```

`DatePicker` komponentasini qo'llanilgan holda yaratilgan mobil ilovaga misol quyida keltirilgan:

```
package com.example.viewapp;
import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.DatePicker;
import android.widget.TextView;
```

```
public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        TextView dateTextView = findViewById(R.id.dateTextView);
        DatePicker datePicker = this.findViewById(R.id.datePicker);
        // Oy noldan boshlangan. Ko'rsatish uchun 1 ni qo'shish lozim
        datePicker.init(2023, 02, 01, new
        DatePicker.OnDateChangedListener() {
            @Override
            public void onChanged(DatePicker view, int year, int
            monthOfYear, int dayOfMonth) {
                // Oylarni ortga hisoblash noldan boshlanadi.
                // Ko'rsatish uchun 1 ni qo'shish lozim
                dateTextView.setText("Sana: " + view.getDayOfMonth() +
                "/" + (view.getMonth() + 1) + "/" + view.getYear());
                // noodatiy yozuv
                // dateTextView.setText("Sana: " + dayOfMonth + "/" +
                (monthOfYear + 1) + "/" + year); } }); }
    }
```

Yuqoridagi mobil ilova dasturida `datePicker.init()` usulidan foydalangan holda boshlang'ich sanani 2023-yil 1-mart qilib belgilangan, chunki oylarni hisoblash noldan boshlanadi. Bundan tashqari, oxirgi parametr, `DatePicker.OnDateChangedListener` obyektini sana tanlagichni qayta ishlashni o'rnatadi. Foydalanuvchi har safar sanani tanlaganida `DatePicker.OnDateChangedListener` obyektining `onDateChanged()` usuli ishga tushadi. Ushbu usul to'rtta parametrni oladi - `view` (`DatePicker` elementi), `year` (tanlangan yil), `monthOfYear` (tanlangan oy), `dayOfMonth` (tanlangan kun). Keyin tanlangan kun, oy va yilni olish mumkin. Bundan tashqari, `onDateChanged()` usulining parametrlaridan ham, `DatePicker` ning o'z usullaridan ham foydalanish mumkin. Tanlashdan oldingi dastlabki holat 2023-yil 1-mart sanasi belgilangan.

`DatePicker` sukut bo'yicha kalendar rejimida ko'rsatiladi, ammo `android:datePickerMode` atributidan foydalangan holda boshqa rejim - `spinner`-ni qo'shish mumkin:

```
<DatePicker android:id="@+id/datePicker"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />
```

```

android:datePickerMode="spinner"
app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintTop_toBottomOf="@+id/dateTextView"

```

▷ Bunday holda, *spinner* kalendarining chap tomonida ko'rsatiladi. Agar kalendarini umuman ko'rsatish lozim bo'lmasa, u holda *android:calendarViewShown="false"* atributini o'rnatish mumkin.

TimePicker komponentasi

TimePicker komponentasi vaqtni 24 soatlik yoki 12 soatlik formatda ko'rsatishi mumkin bo'lgan vaqt tanlash vidjetini taqdim etadi. *TimePicker* atributlari orasida *timePickerMode*-ni ajratib ko'rsatish kerak, bu displey rejimiga ruxsat beradi va ikkita qiymatdan birini qabul qilishi mumkin: *clock* (soat sifatida ko'rsatish) va *spinner* (*spinner* sifatida ko'rsatish). *TimePicker* usullari quyidagilarni o'z ichiga oladi:

- *int getHour()*: soatni qaytaradi (24 soatlik formatda);
- *int getMinute()*: daqiqalarni qaytaradi;
- *boolean is24HourView()*: 24 soatlik format ishlatilsa, *true*-ni qaytaradi;

- *void setHour(int hour)*: *TimePicker* uchun soatni o'rnatadi;
- *void setis24HourView(boolean is24HourView)*: 24 soatlik formatni o'rnatadi;

- *void setMinute(int minute)*: daqiqalarni o'rnatadi;

void setOnTimeChangeListener(TimePicker.OnTimeChangeListener onTimeChangeListener): *TimePicker*-da vaqtni o'zgartirish tinglovchisini *TimePicker.OnTimeChangeListener* obyekti sifatida o'rnatadi.

TimePicker komponentasining dasturdagi ko'rinishi quyidagicha:

```

<TimePicker android:id="@+id/timePicker"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintTop_toBottomOf="parent">

```

TimePicker komponentasini qo'llanilgan holda yaratilgan mobil ilovaga misol quyida keltirilgan:

```

package com.example.viewapp;
import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.TextView;

```

```

import android.widget.TimePicker;
public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        TextView timeTextView = findViewById(R.id.timeTextView);
        TimePicker timePicker = findViewById(R.id.timePicker);
        timePicker.setOnTimeChangeListener(new
        TimePicker.OnTimeChangeListener() {
            @Override
            public void onTimeChanged(TimePicker view, int
            hourOfDay, int minute) {
                timeTextView.setText("Vaqt: " + hourOfDay + ":" + minute);
                // yoki quyidagicha
                // timeTextView.setText("Vaqt: " + view.getHour() + ":" +
                view.getMinute()); } }); }
}

```

Nazorat savollari:

1. Mobil ilovalarning OT ni yaratish uchun qanday platformalaridan foydalaniladi?
2. Abstrakt sinflar va ulardan foydalanish
3. Anonim sinflar va ulardan foydalanish
4. Shablonlar va konteynerlardan foydalanish
5. Mavhum sinflar nima uchun kerak?

4. MOBIL ILOVALARDA MA'LUMOT BAZASI BILAN ISHLASH, GEOLOKATSIYA BILAN ISHLASH

4.1. Ma'lumotlar bazasi bilan ishlash

4.1.1. Androidda ma'lumotlar bazasiga kirish

Android ma'lumotlar bazalari bilan ishlash uchun taniqli SQLite kutubxonasidan foydalanadi. SQLite o'zini ko'plab iste'molchi elektron qurilmalari va dasturlarida, jumladan MP3 pleerlar, iPhone, iPod Touch, Mozilla Firefox va boshqalarda qo'llaniladigan juda ishonchli ma'lumotlar bazasi tizimi ekanligini isbotladi. SQLite yordamida murakkab dastur ma'lumotlarini saqlash va boshqarish ilovasi uchun mustaqil relyatsion ma'lumotlar bazalarini yaratish mumkin. Android ma'lumotlar bazalarini ma'lumotlar bazalarini yaratish mumkin. /data/data/<to'plam nomi>/ma'lumotlar bazalari katalogida saqlaydi. Odatiy bo'lib, barcha ma'lumotlar bazalari shaxsiy hisoblanadi va ularga faqat ularni yaratgan ilovalar kirishi mumkin. PHP + MySQL to'plami bilan tajribaga ega bo'lganlar ko'plab ma'lumotlarni topadilar va uning qanday ishlashini tezda aniqlaydilar.

Android-da `SQLiteOpenHelper` sinfi mavjud. Mobil ilovalarda hamma sinflar `Activity`-dan meros bo'lib keladi (`Activity`-ni kengaytiradi). Ma'lumotlar bazasi bilan ishlashda ham xuddi shunday, ya'ni hamma sinflar `SQLiteOpenHelper` sinfidan meros bo'lib keladi.

4.1.2. Ma'lumotlar bazasini yaratish

Android eng keng tarqalgan ma'lumotlar bazasini boshqarish tizimlaridan biri - SQLite bilan ishlash imkoniyatiga ega. Buning uchun `android.database.sqlite` paketi SQLite ma'lumotlar bazalari bilan ishlash imkonini beruvchi sinflar to'plamini belgilaydi va har bir mobil ilova o'z ma'lumotlar bazasini yaratishi mumkin. Android-da SQLite-dan foydalanish uchun SQL ifodasi yordamida ma'lumotlar bazasini yaratish kerak. Shundan so'ng, ma'lumotlar bazasi mobil ilova katalogida quyidagi manzil bo'yicha saqlanadi:

`DATA/ma'lumotlar/[ilova_nomi]/ma'lumotlar bazalari/[Ma'lumotlar bazasi fayl nomi]`

Odatiy bo'lib, Android operatsion tizimi standart dasturlarda - kontaktlar ro'yxatida, kameradagi fotosuratlarini, musiqa albomlarini saqlash uchun ishlatiladigan bir qator o'rnatilgan ma'lumotlar bazasi SQLite-ni o'z ichiga oladi.

Ma'lumotlar bazalari bilan ishlashning asosiy funkcionalligi `android.database` paketi tomonidan taqdim etiladi. SQLite bilan bevosita ishlash funkcionallari `android.database.sqlite` paketida joylashgan. SQLite-dagi ma'lumotlar bazasi `android.database.sqlite.SQLiteDatabase` sinfi bilan ifodalanadi. U ma'lumotlar bazasiga so'rovlarni bajarish va u bilan turli xil amallarni bajarish imkonini beradi.

`android.database.sqlite.SQLiteCursor` sinfi so'rovlarni taqdim etadi va unga ushbu so'rovlarga mos keladigan qatorlar to'plamini qaytarish imkonini beradi.

`android.database.sqlite.SQLiteQueryBuilder` sinfi SQL so'rovlarni yaratishga imkon beradi.

SQL iboralarning o'zi `android.database.sqlite.SQLiteStatement` sinfi bilan ifodalanadi, bu foydalanuvchiga ifodalarga dinamik ma'lumotlarni kiritish imkonini beradi.

`android.database.sqlite.SQLiteOpenHelper` sinfi barcha jadvallar mavjud bo'lmasa, ular bilan ma'lumotlar bazasini yaratishga imkon beradi.

SQLite quyidagi ma'lumotlar tipi tizimidan foydalanadi:

- `INTEGER`: Java tilidagi `int` tipiga o'xshash butun sonni ifodalaydi;

- `REAL`: Java-dagi `float` va `double`-ga o'xshash suzuvchi nuqta sonini ifodalaydi;

- `TEXT`: Java-da `String` va `char`-ga o'xshash belgilar to'plamini ifodalaydi;

- `BLOB`: Java-dagi `int` tipiga o'xshash tasvir kabi ikkilik ma'lumotlar majmuasini ifodalaydi.

Saqlanayotgan ma'lumotlar Java-da tegishli turlarni ifodalashi kerak.

Ma'lumotlar bazasini yaratish va ochish

Android-dagi `Activity` kodidan yangi ma'lumotlar bazasini yaratish yoki ochish uchun `openOrCreateDatabase()` usulini chaqirish mumkin. Ushbu usul uchta parametрни qabul qilishi mumkin:

- ma'lumotlar bazasi nomini;
- ish rejimini belgilovchi raqamli qiymat (odatda `MODE_PRIVATE` doimiy shaklida);

• ma'lumotlar bazasi bilan ishlash uchun kursor yaratishni ifodalovchi `SQLiteDatabase.CursorFactory` obyektini ko'rinishidagi ixtiyoriy parameter.

Masalan, `app.db` ma'lumotlar bazasini yaratish quyidagicha amalga oshiriladi:

```

        SQLiteDatabase db =
        getBaseContext().openOrCreateDatabase("app.db", MODE_PRIVATE,
        null);

```

Ma'lumotlar bazasi so'rovini bajarish uchun *SQLiteDatabase* sinifining *execSQL()* usulidan foydalanish mumkin. Ushbu usulga SQL ifodasi uzatiladi. Masalan, ma'lumotlar bazasida foydalanuvchilar jadvalini yaratish quyidagicha amalga oshiriladi:

```

        SQLiteDatabase db =
        getBaseContext().openOrCreateDatabase("app.db", MODE_PRIVATE,
        null);

```

```

        db.execSQL("CREATE TABLE IF NOT EXISTS users (name
        TEXT, age INTEGER)");

```

Agar yuqoridagi ifodani bajarishdan tashqari ma'lumotlar bazasidan ba'zi ma'lumotlarni olish kerak bo'lsa, u holda *rawQuery()* usuli qo'llaniladi. Ushbu usul parametr sifatida SQL ifodasini, shuningdek SQL ifodasi uchun qiymatlar to'plamini oladi. Masalan, ma'lumotlar bazasidan barcha obyektlarni olish quyidagicha amalga oshiriladi:

```

        SQLiteDatabase db =
        getBaseContext().openOrCreateDatabase("app.db", MODE_PRIVATE,
        null);

```

```

        db.execSQL("CREATE TABLE IF NOT EXISTS users (name TEXT,
        age INTEGER)");

```

```

        Cursor query = db.rawQuery("SELECT * FROM users;", null);
        if(query.moveToFirst()){

```

```

            String name = query.getString(0);
            int age = query.getInt(1);

```

```

        }

```

db.rawQuery() usuli qabul qilingan ma'lumotlarni olish mumkin bo'lgan *Cursor* obyektini qaytaradi.

Ma'lumotlar bazasida obyektlar mavjud bo'lmashligi vaziyati yuzaga kelishi mumkin va bunday holatlar uchun *query.moveToFirst()* usuli yordamida ma'lumotlar bazasidan olingan birinchi obyektga o'tishga harakat qilinadi. Agar bu usul *false* qiymat qaytarsa, so'rovga ma'lumotlar bazasidan hech qanday ma'lumot olmagan hisoblanadi.

Ma'lumotlar bazasi bilan ishlash uchun yangi loyiha quyidagi tarzda yaratiladi. *activity_main.xml* faylida oddiy grafik interfeys yaratib olinadi:

```

<?xml version="1.0" encoding="utf-8"?>

```

```

<androidx.constraintlayout.widget.ConstraintLayout

```

```

    xmlns:android="http://schemas.android.com/apk/res/android"

```

```

    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="16dp" >

```

```

<Button

```

```

    android:id="@+id/button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Click"
    android:onClick="onClick"
    app:layout_constraintBottom_toTopOf="@id/textView"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintTop_toTopOf="parent" />

```

```

<TextView

```

```

    android:id="@+id/textView"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textSize="22sp"
    app:layout_constraintTop_toBottomOf="@id/button"
    app:layout_constraintLeft_toLeftOf="parent" />

```

```

</androidx.constraintlayout.widget.ConstraintLayout>

```

Oddiy grafik interfeys yaratib olingandan keyin *MainActivity.java* sinfida ma'lumotlar bazasi bilan o'zaro aloqa aniqlab olinadi:

```

package com.example.sqliteapp;
import androidx.appcompat.app.AppCompatActivity;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.os.Bundle;
import android.view.View;
import android.widget.TextView;
public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
    public void onClick(View view){

```

```

        SQLiteDatabase db =
        getBaseContext().openOrCreateDatabase("app.db", MODE_PRIVATE,
        null);
        db.execSQL("CREATE TABLE IF NOT EXISTS users (name
        TEXT, age INTEGER, UNIQUE(name))");
        db.execSQL("INSERT OR IGNORE INTO users VALUES
        ('Tom Smith', 23), ('John Dow', 31);");
        Cursor query = db.rawQuery("SELECT * FROM users;", null);
        TextView textView = findViewById(R.id.textView);
        textView.setText("");
        while(query.moveToNext()){
            String name = query.getString(0);
            int age = query.getInt(1);
            textView.append("Name: " + name + " Age: " + age + "\n");
        }
        query.close();
        db.close();
    }

```

Bu yerdagi tugmani bosganda avval *app.db* ma'lumotlar bazasida *users* nomli yangi jadval yaratiladi va keyin *INSERT* SQL operatori yordamida ma'lumotlar bazasiga ikkita obyektini qo'shadi.

Keyin, *SELECT* iborasidan foydalanib, barcha qo'shilgan foydalanuvchilarni ma'lumotlar bazasidan *Cursor* shaklida olinishi mumkin.

query.moveToNext() usulini chaqirish orqali barcha obyektlarni *while* tsikli yordamida bittama – bitta ketma-ket o'tib chiqiladi.

Kursordan ma'lumotlarni olish uchun *query.getString(0)* va *query.getInt(1)* usullari qo'llaniladi. Qavslar ichida usullarga ma'lumotlarni oladigan ustun raqami beriladi. Masalan, yuqorida avval qator sifatida foydalanuvchi nomini, keyin raqam sifatida yoshni qo'shish mumkin. Bu shuni anglatadiki, nolinch ustun *getString()* usuli yordamida olingan satr qiymati bo'ladi va keyingi, birinchi ustunda *getInt()* usuli qo'llaniladigan raqamli qiymat bo'ladi.

Kursor va ma'lumotlar bazasi bilan ishlashni tugatgandan so'ng, barcha bog'langan obyektlar yopiladi:

```

        query.close();
        db.close();

```

Agar kursor yopilmasa, xotirani sarf qilish muammosiga duch kelish mumkin va dastur ishga tushganda, tugmani bosgandan so'ng, qo'shilgan ma'lumotlar matn maydonida ko'rsatiladi (4.1-rasm).

click

Name: Tom Smith Age: 23
Name: John Dow Age: 31

4.1-rasm. Tugmani bosgandan so'nggi natija

4.1.3. So'rovlarni yaratish

Ma'lumotlar bazasi bilan ishlash va so'rovlarni yaratish uchun ba'zi usullar qo'llaniladi. *onCreate()* usulini ilovada quyidagicha qo'llash mumkin:

```

@Override
protected void onCreate (Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.Activity_main);
    CatsDataBase sqh = yangi CatsDataBase(bu);
    SQLiteDatabase sqdb = sqh.getWritableDatabase();
    // ma'lumotlar bazasi ulanishlarini yopish
    sqdb.close();
    sqh.close();
}

```

Loyihani ishga tushirganda hech narsa sodir bo'lmaydi, lekin aslida *cat_database.db* fayli */data/data/ru.alexanderklimov.ru.databasedemo/databases* katalogida paydo bo'ladi. Agar emulyatordagi *File Explorer* yorlig'ida *DDMS* bo'limini ochganda va fayl tuzilishini qaraganda, buni ko'rish mumkin.

Ma'lumotlar bazasidan ma'lumotlarni qo'shish va olishning turli usullari mavjud.

Ma'lumotlarni kiritishning birinchi usuli *ContentValues* sinfidan foydalanishdir. Matn maydonidagi ma'lumotlar ma'lumotlar bazasiga tushadigan tugmachaga misol:

```
case R.id.buttonCVInsert:
    ContentValues cv = new ContentValues();
    cv.put(CatsDataBase.CATNAME,txtData.getText().toString());
    sqdb.insert(CatsDataBase.TABLE_NAME,CatsDataBase.CATNAM
E, cv);
    txtData.setText("");
    break;
```

Bu usul juda qulay, kam kod talab qiladi va o'qish oson. Sinfning nusxasini yaratib va keyin kerakli ma'lumotlarni kerakli ustunga yozish uchun *put()* usulidan foydalanish mumkin. Shundan so'ng, tayyorlangan ma'lumotlarni jadvalga joylashtiradigan *insert()* usuli chaqiriladi.

Agar bir nechta ustunlar bo'lsa, *put()* usulini bir necha marta chaqirish mumkin:

```
values.put(CatsDataBase.CATNAME,txtData.getText().toString());
values.put(CatsDataBase.EMAIL,txtEmail.getText().toString());
values.put(CatsDataBase.PHONE,txtPhone.getText().toString());
insert() usuli uchta argumentga ega. Birinchisi, yozuvlar tuziladigan jadval nomini belgilaydi. Uchinchi avval yaratilgan ContentValues obyektini belgilaydi. Ikkinchi argument ustunni ko'rsatish uchun ishlatiladi. SQL bo'sh yozuvni kiritishga ruxsat bermaydi va agar bo'sh ContentValue ishlatilsa, xatolikka yo'l qo'ymaslik uchun ikkinchi argumentga null o'rnatish lozim.
```

Ikkinchi usul an'anaviy SQL-ning *INSERT INTO...* so'rovidan foydalanadi. Oddiy so'rovni satr sifatida shakllantirib, keyin uni *execSQL()* usuliga o'tkaziladi. Ushbu usulning asosiy noqulayligi qo'shtirmoqlarda chalkashmaslik hisoblanadi. Agar biror narsa kiritilmagan bo'lsa, xabar jurnallariga (*log*) qarash lozim.

```
case R.id.buttonSQLQuery:
    String insertQuery = "INSERT INTO" +
    CatsDataBase.TABLE_NAME +
    + "(" + CatsDataBase.CATNAME + ") VALUES (" +
    txtData.getText().toString() + ")";
    sqdb.execSQL(insertQuery);
    txtData.setText("");
```

break;

Bundan tashqari, ma'lumotlarni ikki usulda o'qish mumkin. Ikkala holatda ham natija *Cursor* obyektini sifatida qaytariladi. Uni ekranda aylanib yuradigan sichqoncha kursori bilan aralastirmaslik lozim.

query() usuli juda ko'p parametrlarga ega. Birinchi parametrda jadval nomini, ikkinchi parametrda ustun nomlari qatorini belgilash lozim, keyin qo'shimcha shartlar qo'yiladi. Sukut bo'yicha hamma joyda *null* qo'yish mumkin. Bundan tashqari, *while* tsikli orqali ma'lumotlarni chiqarib, jurnallarga (*log*) joylashtirish mumkin. Loyihani qayta ishga tushirish va ma'lumotlar bazasiga qanday ma'lumotlarni kiritganligini tekshirish mumkin:

```
case R.id.buttonQuery:
    Cursor cursor = sqdb.query(CatsDataBase.TABLE_NAME,
new String[] {
```

```
CatsDataBase._ID, CatsDataBase.CATNAME },
null, // WHERE bandi uchun ustunlar
null, // WHERE bandi uchun qiymatlar
null, // qatorlarni guruhlamaslik uchun
null, // qator guruhlari bo'yicha filtrlamaslik uchun
null // Saralash tartibi
);
```

```
while (cursor.moveToNext()) {
    int id = cursor.getInt(cursor.getColumnIndex(CatsDataBase._ID));
    String name = cursor.getString(cursor
.getColumnIndex(CatsDataBase.CATNAME));
    Log.i("LOG_TAG", "ROW " + id + " HAS NAME " +
name);
```

```
}
cursor.close();
break;
```

Ikkinchi usul to'liqmas (*raw*) SQL so'rovidan foydalanadi. Birinchidan, so'rovlar qatori shakllantiriladi va *rawQuery()* usuliga beriladi.

```
case R.id.buttonRawQuery:
    String query = "SELECT " + CatsDataBase._ID + "," +
    + CatsDataBase.CATNAME + " FROM " +
CatsDataBase.TABLE_NAME;
    Cursor cursor2 = sqdb.rawQuery(query, null);
    while (cursor2.moveToNext()) {
        int id = cursor2.getInt(cursor2
```

```

        .getColumnIndex(CatsDataBase._ID));
String name = cursor2.getString(cursor2
        .getColumnIndex(CatsDataBase.CATNAME));
Log.i("LOG_TAG", "ROW " + id + " HAS NAME " +
name);
    }
    cursor2.close();
    break;

```

4.2. Kontent provayderlar va ulardan foydalanish

Kontent provayderi ma'lumotlar omboriga kirishni boshqaradi. Android ilovasida provayderni amalga oshirish uchun dastur manifestiga muvofiq sinflar to'plami yaratilishi kerak. Ushbu sinflardan biri kontent provayderi va boshqa ilovalar o'rtasida interfeysni ta'minlovchi *ContentProvider* sinfidan meros bo'lishi kerak. Ushbu komponentning asosiy maqsadi boshqa ilovalarga ma'lumotlarga kirishni ta'minlashdan iborat, lekin hech narsa ilovaning foydalanuvchiga kontent provayderi tomonidan boshqariladigan ma'lumotlarni so'rash va o'zgartirish imkonini beruvchi *Activity*-ga ega bo'lishiga to'sqinlik qilmaydi. Mobil ilovalarda kontent-provayderlar quyidagi hollarda kerak bo'ladi:

- ilova murakkab ma'lumotlar yoki fayllarni boshqa ilovalarga taqdim etadi;
- ilova foydalanuvchilarga murakkab ma'lumotlarni boshqa ilovalarga nusxalash imkonini beradi;
- ilova qidiruv platformasi (framework) yordamida maxsus qidiruv imkoniyatlarini taqdim etadi.

Agar ilova kontent provayderidan foydalanishni talab qilsa, ushbu komponentni yaratish uchun bir necha bosqichlarni bajarish kerak.

Kontent provayderlari bilan ishlaydigan ma'lumotlar ikki shaklda tashkil etilishi mumkin:

- Ma'lumotlar fotosuratlar, audio yoki video kabi fayl bilan ifodalanadi. Bunday holda, ma'lumotlarni ilovaning shaxsiy xotirasida saqlash kerak. Boshqa dasturning so'roviga javoban provayder faylga havolani qaytarishi mumkin.

- Ma'lumotlar qandaydir struktura bilan ifodalanadi, masalan, jadval, massiv. Bunday holda, ma'lumotlarni jadval shaklida saqlash kerak. Jadval qatori ba'zi bir obyektini, masalan, xodim yoki mahsulotni ifodalaydi. Va ustun – bu korxonaning ba'zi mulki, masalan, xodimning nomi yoki

mahsulot narxini ifodalaydi. Android-da bunday ma'lumotlarni saqlashning keng tarqalgan usuli SQLite ma'lumotlar bazasida, ammo har qanday doimiy saqlash usulidan foydalanish mumkin.

ContentProvider sinfidan to'g'ridan-to'g'ri yoki uning avlodlaridan birortasi orqali meros bo'ladigan sinf yaratish uchun kerakli usullarni bekor qilish kerak.

Yaratilgan kontent provayderi boshqa ilovalardan so'rovlarni qayta ishlash orqali tuzilgan ma'lumotlarga kirishni boshqaradi. Barcha so'rovlar oxir-oqibat *ContentResolver* obyektini chaqiradi, bu esa o'z navbatida kirish uchun *ContentProvider* obyektidagi tegishli usulni chaqiradi.

query() – provayderdan ma'lumotlarni oladigan, jadval, qatorlar va ustunlarni argument sifatida qabul qiluvchi, shuningdek, natijani sortirovka qilish tartibi, *Cursor* tipidagi obyektini qaytaradigan usul.

insert() – yangi qator qo'shadigan, jadvalni argument sifatida qabul qiladigan va qator elementlarining qiymatlarini qo'shilgan qatorning *URI*-ni qaytaradigan usul.

update() – mavjud satrlarni yangilaydigan, jadvalni, yangilanadigan qatorlarni va qator elementlarining yangi qiymatlarini argument sifatida qabul qiladigan, yangilangan qatorlar sonini qaytaradigan usul.

delete() – satrlarni o'chiradigan, o'chirish uchun jadval va qatorlarni argument sifatida oladigan, o'chirilgan qatorlar sonini qaytaruvchi usul.

getType() – bu *URI* ga mos keladigan ma'lumotlar turini tavsiflovchi *MIME* formatidagi *String* tipini qaytaruvchi usul.

onCreate() – provayder yaratilgandan so'ng darhol tizim tomonidan chaqiriladigan usul provayderni ishga tushirishni o'z ichiga oladi. Shuni ta'kidlash kerakki, *ContentResolver* obyektini unga kirishga harakat qilmaguncha provayder yaratilmaydi.

OnCreate()-dan tashqari yuqoridagi barcha usullar mijoz ilovasi tomonidan chaqiriladi. Bu usullarning barchasi xuddi shu nomdagi *ContentResolver* sinfining usullari bilan bir xil ko'rinishga ega.

Provayderning avtorizatsiya qatorining ta'rifi, uning satrlari va ustun nomlari uchun *URI* agar provayder *Intent*-larni boshqarishi kerak bo'lsa, unda *intent* harakatlari, tashqi ma'lumotlar va bayroqlar aniqlanishi kerak. Bundan tashqari, ilovalar provayder ma'lumotlariga kirish uchun kerak bo'lgan ruxsatlarni aniqlash kerak. Bu qiymatlarning barchasi alohida sinfda doimiylik sifatida belgilanishi kerak, keyinchalik bu sinf boshqa ishlab chiquvchilarga berilishi mumkin.

Quyidagi ilova foydalanuvchi haqidagi turli ma'lumotlarni, ba'zi tegishli ma'lumotlarni fayllar yoki sozlamalarda saqlashi mumkin. Biroq,

Android OT kirish va foydalanish mumkin bo'lgan foydalanuvchi bilan bog'liq bir qator muhim ma'lumotlarni saqlaydi. Bular kontaktlar ro'yxati va saqlangan tasvirlar va video materiallarning fayllari va qo'ng'iroqlar haqidagi ba'zi belgilar va boshqalar, ya'ni ba'zi kontentlar. Android OT da ushbu kontentga kirish uchun kontent provayderlari aniqlanadi (*content provider*).

Androidda *android.content* paketida belgilangan quyidagi o'rnatilgan provayderlar mavjud:

- *AlarmClock*: budilnikni boshqarish;
- *Brauser*: brauzer va zakladkalar tarixi;
- *CalendarContract*: kalendar va hodisa ma'lumotlari;
- *CallLog*: qo'ng'iroqlar haqida ma'lumot;
- *ContactsContract*: kontaktlar;
- *MediaStore*: media fayllar;
- *SearchRecentSuggestions*: qidiruv takliflari;
- *Settings*: tizim sozlamalari;
- *UserDictionary*: Tez terish uchun ishlatiladigan so'zlar lug'ati;
- *VoicemailContract*: ovoqli pochta yozuvlari.

Android kontaktlar o'rnatilgan API-ga ega bo'lib, u kontaktlar ro'yxatini o'qish va tahrirlash imkonini beradi. Hamma kontaktlar SQLite ma'lumotlar bazasida saqlanadi, lekin ular bitta jadvalni ifodalamaydi. Kontaktlar uchun bitta - ko'p munosabat bilan bog'langan uchta jadval mavjud: odamlar haqidagi ma'lumotlarni saqlash uchun jadval, ularning telefonlari uchun jadval va elektron pochta manzillari uchun jadval. Ammo Android API tufayli jadvallar orasidagi munosabatlardan mavhumlash mumkin. Shunday qilib, kontaktlarga kirish uchun ilova manifest faylida tegishli ruxsatlarni o'rnatish kerak:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="
"http://schemas.android.com/apk/res/android"
package="com.example.eugene.sqliteapp">
<uses-permission android:name="
"android.permission.READ_CONTACTS" android:maxSdkVersion="21"
/>
<application
android:allowBackup="true"
android:icon="@mipmap/ic_launcher"
android:label="@string/app_name"
android:theme="@style/AppTheme">
```

```
<Activity
android:name=".MainActivity"
android:label="@string/app_name">
<intent-filter>
<action android:name="android.intent.action.MAIN" />
<category android:name="
"android.intent.category.LAUNCHER" />
</intent-filter>
</Activity> </application> </manifest>
```

Kontaktlar ro'yxatini ko'rsatish uchun quyidagi interfeys elementlarini kiritish kerak:

```
<LinearLayout xmlns:android="
"http://schemas.android.com/apk/res/android"
android:orientation="vertical"
android:layout_width="match_parent"
android:layout_height="match_parent">
<TextView
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:text="@string/contact_list">
</TextView>
<ListView
android:id="@+id/contactList"
android:layout_width="match_parent"
android:layout_height="wrap_content">
</ListView>
</LinearLayout>
```

Kontaktlar ro'yxatini ko'rsatish uchun *ListView* elementidan foydalaniladi va *Activity* sinfida kontaktlar olinadi:

```
package com.example.sqliteapp;
import android.support.v7.app.ActionBarActivity ;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.provider.ContactsContract;
import android.database.Cursor;
import android.widget.ListView;
import android.widget.ArrayAdapter;
import java.util.ArrayList;
```



```

public class MainActivity extends ActionBarActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        ListView contactsList = (ListView)
            findViewById(R.id.contacts_list);
        ArrayList<String> contacts = new ArrayList<String>();
        Cursor contactsCursor = getContentResolver()
            .query(ContactsContract.Contacts.CONTENT_URI, null,
            null, null, null);
        while (contactsCursor.moveToNext()) {
            String contact = contactsCursor.getString(
                contactsCursor.getColumnIndex(ContactsContract.Contacts.DISPLAY_NAME_PRIMARY));
            contacts.add(contact); }
        ArrayAdapter<String> adapter = new
        ArrayAdapter<String>(this,
            android.R.layout.simple_list_item_1, contacts);
        contactsList.setAdapter(adapter); }
}

```

Barcha kontaktlar va tegishli funksiyalar maxsus SQLite ma'lumotlar bazalarida saqlanadi. Lekin ular bilan bevosita ishlash shart emas. Buning uchun *Cursor* sinfining obyektidan foydalanish mumkin. Uni olish uchun birinchi navbatda *getContentResolver()* usuli chaqiriladi, u *ContentResolver* obyektini qaytaradi. Keyin *query()* usuli chaqiriladi. Ushbu usul bir qator parametrlardan o'tadi, ulardan birinchisi *URI* – olishni istagan resursni ifodalaydi. Kontaktlar ma'lumotlar bazasiga *ContactsContract.Contacts.CONTENT_URI* doimiysi yordamida kirishi mumkin.

contactsCursor.moveToNext() usuli kontakt yozuvlari bo'ylab ketma-ket o'tishga imkon beradi, *contactsCursor.getString()* chaqiruvi orqali bir vaqtning o'zida bitta kontaktni o'qiydi.

Qabul qilingan barcha kontaktlar ro'yxatga qo'shiladi, keyinchalik ular *ListView*-da ko'rsatiladi.

Kontaktlarni qo'shish – bu kontaktlar ro'yxatini o'zgartirish, ya'ni uni yozib olish so'rovidir. Shuning uchun manifest faylida tegishli ruxsatni o'rnatish kerak. Oldingi loyihadagi manifest fayliga *manifest* ildiz tegidan keyin quyidagi qatorni qo'shish lozim.

```

<uses-permission android:name=
"android.permission.WRITE_CONTACTS" android:maxSdkVersion=
"21" />

```

Kontaktlar ro'yxatiga o'zgartirish kiritish uchun *android.permission.WRITE_CONTACTS* ruxsatini o'rnatish kerak. Kontakt qo'shish uchun yangi *Activity* qo'shish lozim va uni *AddContactActivity* deb nomlash kerak. Matn maydonini va unga kontakt qo'shish tugmachasini joylashtirish lozim:

```

<LinearLayout xmlns:android=
"http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="@dimen/Activity_horizontal_margin"
    android:orientation="vertical">
    <TextView android:text="Yangi kontakt:"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
    <EditText
        android:id="@+id/contactText"
        android:layout_width="match_parent"
        android:layout_height="45dip" />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Qo'shish"
        android:onClick="addContact" />
</LinearLayout>

```

Activity kodida kontakt qo'shilgan holda *addContact* ishlov beruvchisi quyidagicha yoziladi:

```

package com.example.sqliteapp;
import android.provider.ContactsContract;
import android.support.v7.app.ActionBarActivity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.content.ContentValues;
import android.content.ContentUris;
import android.provider.ContactsContract.RawContacts;

```

```

import android.provider.ContactsContract.Data;
import
android.provider.ContactsContract.CommonDataKinds.StructuredName;
import android.widget.EditText;
import android.widget.Toast;
import android.net.Uri;
public class AddContactActivity extends ActionBarActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.Activity_add_contact);
    }
    public void addContact(View view) {
        ContentValues contactValues = new ContentValues();
        EditText contactText = (EditText)
findViewById(R.id.contactText);
        String newContact = contactText.getText().toString();
        contactValues.put(RawContacts.ACCOUNT_NAME,
newContact);
        contactValues.put(RawContacts.ACCOUNT_TYPE,
newContact);
        Uri newUri =
getContentResolver().insert(RawContacts.CONTENT_URI,
contactValues);
        long rawContactsId = ContentUris.parseId(newUri);
        contactValues.clear();
        contactValues.put(Data.RAW_CONTACT_ID, rawContactsId);
        contactValues.put(Data.MIMETYPE,
StructuredName.CONTENT_ITEM_TYPE);
        contactValues.put(StructuredName.DISPLAY_NAME,
newContact);
        getContentResolver().insert(Data.CONTENT_URI,
contactValues);
        Toast.makeText(this, newContact + " kontaktlar royxatiga
qoshilgan", Toast.LENGTH_LONG).show();
    }
    Barcha qo'shish kodi addContact tugmachasini bosish moslamasida
joylashgan. Android-da kontaktlar uchta jadvalga ajratilgan: contacts, raw
contacts va data. Oxirgi ikkita jadvalga yangi kontakt qo'shish uchun
sozlamalar tufayli kontaktlar jadvaliga qo'shilmaydi, ammo bu shart emas.

```

Kontakt ma'lumotlari kalitlar va ularning qiymatlaridan, ya'ni lug'at obyektidan iborat *ContentValues* obyektini ifodalaydi. Yaratilganidan keyin unga bir nechta elementlar qo'shiladi:

```

contactValues.put(RawContacts.ACCOUNT_NAME, newContact);
contactValues.put(RawContacts.ACCOUNT_TYPE, newContact);

```

Kontaktning nomi va turi quyidagicha o'rnatiladi. *RawContacts.ACCOUNT_NAME* va *RawContacts.ACCOUNT_TYPE* qiymatlari kalit sifatida, matn maydonidagi matn esa ularning qiymati sifatida o'rnatiladi. Keyinchalik, ushbu obyekt *insert()* usuli yordamida *RawContacts* jadvaliga qo'shiladi:

```

Uri newUri =
getContentResolver().insert(RawContacts.CONTENT_URI,
contactValues);

```

insert() usuli *URI*-ni qaytaradi – jadvaldagi qo'shilgan obyektga havola, undan identifikatori olish mumkin. Keyin, tozalashdan so'ng, yana ma'lumotlar bilan to'ldirib, Ma'lumotlar jadvaliga qo'shiladigan obyektни tayyorlash mumkin:

```

contactValues.put(Data.RAW_CONTACT_ID, rawContactsId);
contactValues.put(Data.MIMETYPE,

```

```
StructuredName.CONTENT_ITEM_TYPE);
```

```
contactValues.put(StructuredName.DISPLAY_NAME, newContact);
```

hamda yana qo'shimcha *insert()* usuli orqali qo'shish mumkin:

```
getContentResolver().insert(Data.CONTENT_URI, contactValues);
```

Ushbu *Activity*-ga asosiy *Activity*-dan kirish uchun asosiy menyuda mos keladigan menyu bandini yaratish mumkin, qaysi birini tanlash orqali u yangi kontakt qo'shish uchun sahifaga yo'naltiriladi. Masalan, asosiy *Activity* menyusi resurs faylidagi menyu bandi:

```

<menu xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:app="http://schemas.android.com/apk/res-auto"
xmlns:tools="http://schemas.android.com/tools" tools:context=

```

```
“.MainActivity”>
```

```
<item
```

```
android:id= “@+id/action_settings”
```

```
android:title= “Qoshish”
```

```
android:orderInCategory= “100”
```

```
app:showAsAction= “never” />
```

```
</item>
```

```
</menu>
```

Asosiy *Activity* da shu bo'limni qayta ishlash uchun quyidagicha kod yoziladi:

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    getMenuInflater().inflate(R.menu.menu_main, menu);
    return true;}

```

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    int id = item.getItemId();
    if (id == R.id.action_settings) {
        Intent intent = new Intent(this, AddContactActivity.class);
        startActivity(intent);
        return true; }
    return super.onOptionsItemSelected(item);}

```

Har qanday kontent provayderining ma'lumotlariga kirish uchun mijoz sifatida provayder bilan bog'lanish uchun dastur kontekstining *getContentResolver* usuli yordamida olinishi mumkin bo'lgan *ContentResolver* sinfining obyektini ishlatiladi:

```
ContentResolver cr = getApplicationContext().getContentResolver();
ContentResolver obyektini mijoz so'rovlarini yuborish orqali kontent
```

provayderi obyektini bilan o'zaro ta'sir qiladi. Kontent provayderi so'rovlarni qayta ishlaydi va qayta ishlash natijalarini qaytaradi.

Kontent provayderlari o'z ma'lumotlarini iste'molchilarga relyatsion ma'lumotlar bazasi jadvallariga o'xshash bir yoki bir nechta jadvallar shaklida taqdim etadilar. Har bir satr tegishli nomlangan maydonlarda ko'rsatilgan xususiyatlarga ega alohida "obyekt" dir. Odatda, har bir satr o'ziga xos butun son indeksiga va "_id" nomiga ega bo'lib, kerakli obyektini yagona aniqlash uchun ishlatiladi. Kontent provayderlari odatda ma'lumotlar bilan ishlash uchun kamida ikkita *URI*-ni taqdim etadilar: biri bir vaqtning o'zida barcha ma'lumotlarni talab qiladigan so'rovlar uchun, ikkinchisi esa ma'lum bir "string" ga kirish uchun. Ikkinchi holda, *URI* oxiriga / qo'shiladi (bu "_id" indeksiga mos keladi). Ma'lumotlarni qidirish so'rovlari *ContentResolver* obyektining so'rov usulidan foydalangan holda ma'lumotlar bazasi so'rovlariga o'xshaydi. Javob, shuningdek, olingan ma'lumotlar to'plamiga (tanlangan jadval qatorlari) "maqsadli" kursor shaklida keladi:

```
ContentResolver cr = getContentResolver();
Cursor c = cr.query(ContactsContract.Contacts.CONTENT_URI,
null, null, null, null);
String where = KEY_COL3 + "=" + requiredValue;
String order = KEY_COL5;
```

```
Cursor someRows = cr
    .query(MyProvider.CONTENT_URI, null, where, null, order);
Uri myRowUri = cr.insert(SampleProvider.CONTENT_URI,
newRow);
```

```
ContentValues[] valueArray = new ContentValues[5];
int count = cr.bulkInsert(MyProvider.CONTENT_URI, valueArray);
Bitta elementni kiritishda kiritish usuli kiritilgan elementning URI-ni qaytaradi, ko'proq elementlarni kiritish esa kiritilgan elementlar sonini qaytaradi. Quyida kiritilgan elementni o'chirishga misol keltirilgan:
```

```
ContentResolver cr = getContentResolver();
cr.delete(myRowUri, null, null);
String where = "_id < 5";
cr.delete(MyProvider.CONTENT_URI, where, null);
```

Quyida kiritilgan elementni o'zgartirishga misol keltirilgan:

```
ContentValues newValues = new ContentValues();
newValues.put(COLUMN_NAME, newValue);
String where = "_id < 5";
getContentResolver().update(MyProvider.CONTENT_URI,
newValues, where, null);
```

4.3. Mobil ilovalarda tarmoqli dasturlash

Android ilovasi, boshqa mobil ilovalar kabi, odatda kichik hajmga ega, lekin juda funkSIONALdir. Kichkina qurilmada bunday turli xil funksiyalarga erishishning usullaridan biri dastur ma'lumotlarini turli manbalardan olishdir. Masalan, T-mobile G1 ancha rivojlangan xaritalash funksiyalariga ega Xaritalar dasturi bilan birga keladi.

Biroq, biz bilamizki, ushbu dastur Google Maps va boshqa xizmatlar bilan integratsiyalashgan, buning natijasida bunday mukammallikka erishiladi. Siz yaratgan dasturlar boshqa dasturlardan olingan ma'lumotlardan ham keng foydalanishi mumkin. Odatda, HTTP bir nechta dasturlarni birlashtirish uchun ishlatiladi. Masalan, Internetda Android dasturlaridan birini samaraliroq ishga tushirish uchun kerakli xizmatlarni taqdim etadigan Java servletini mavjud bo'lishi mumkin. Ushbu servletni androidda qanday qo'llash mumkin? Shunisi qiziqki, Android SDK J2EE bilan ishlash uchun universal vosita bo'lgan Apache uchun *HttpClient* mijozini o'z ichiga oladi. Android SDK Android talablari uchun o'zgartirilgan *HttpClient* versiyasini o'z ichiga oladi, ammo amaliy dasturlash interfeyslari (API) mos keladigan J2EE API-lariga juda o'xshash

bo'ladi.

Apache *HttpClient* mijozni murakkab va ko'p qirrali hisoblanadi. Ushbu mijoz HTTP protokolini to'liq qo'llab-quvvatlashda, HTTP GET va HTTP POST usullaridan foydalanish mumkin. Quyida *HttpClient* uchun umumiy foydalanish bosqichlariga namuna keltirilgan.

1. *HttpClient* yaratish (yoki mavjud havolani olish).
2. *PostMethod* yoki *GetMethod* kabi yangi HTTP usulini ishga tushirish.

3. HTTP parametrlarining nomlari va qiymatlarini o'rnatish.

4. *HttpClient* yordamida HTTP ni chaqirish.

5. HTTP javobini qayta ishlash.

HttpClient kodi yordamida HTTP GET so'rovini qanday amalga oshirish quyidagi misolda ko'rsatilgan. *HttpClient* kodi Internetga kirishga urinayotganligi sababli, *HttpClient* yordamida HTTP chaqiruvlarini ishlatganda, manifest fayliga *android.permission.INTERNET* kirish huquqini qo'shish kerak. HTTP GET so'rovini olish uchun *HttpClient*-dan foydalanish mumkin:

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.URI;
import org.apache.http.HttpResponse;
import org.apache.http.client.HttpClient;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.impl.client.DefaultHttpClient;
public class TestHttpGet {
public void executeHttpGet() throws Exception {
    BufferedReader in = null; try {
        HttpClient client = new DefaultHttpClient();
        HttpGet request = new HttpGet();
        request.setURI(new URI("http://code.google.com/android/"));
        HttpResponse response = client.execute(request);
        in = new BufferedReader (new
InputStreamReader(response.getEntity().getContent()));
        StringBuffer sb = new StringBuffer("");
        String line =
        String NL = System.getProperty("line.separator");
        while ((line = in.readLine()) != null) { sb.appendline + NL); }
        in.close(); String page = sb.toString();
```

```
System.out.println(page);
} finally { if (in != null) { try { in.close();
} catch (IOException e) { e.printStackTrace(); }
} } }
```

HttpClient *HttpGet*, *HttpPost* va boshqalar kabi HTTP so'rovlarining har xil turlarini tavsiflash uchun abstraksiyalarni taqdim etadi. HTTP so'rovining o'zini bajarish uchun *client.execute()* usuli chaqiriladi. So'rovni bajargandan so'ng, kod olingan javobni to'liq o'qiydi va uni satr obyektiga joylashtiradi. *BufferedReader* usuli *finally* kod blokida yopilganligini ko'rish mumkin. Shu bilan birga, asosiy HTTP ulanishi tugatiladi.

Yuqoridagi sinf *android.app.Activity*-ga qo'shimcha emas. Boshqacha qilib aytadigan bo'lsak, *HttpClient*-dan foydalanish uchun ma'lum bir *Activity* kontekstida bo'lishi shart emas, chunki *HttpClient* Android dasturiy ta'minot paketining bir qismidir va bu mijoz ham Android komponenti kontekstida, ham quyidagi tarzda ishlatilishi mumkin. Kod HTTP so'rovini serverga hech qanday HTTP parametrlarini uzatmasdan amalga oshiradi. "Ism/qiymat" parametrlari URL manziliga "ism/qiymat" juftlarini qo'shish orqali so'rov bilan birga uzatilishi mumkin. HTTP GET so'roviga parametrlarni qo'shish quyidagicha amalga oshiriladi:

```
HttpGet method = new
HttpGet("http://somehost/WS2/Upload.aspx?one=valueGoesHere");
client.execute(method);
```

HTTP GETni amalga oshirishda so'rovning parametrlari (ismlari va qiymatlari) URL manziliga uzatiladi. Parametrlarni shu tarzda o'tkazishda bir qator cheklovlarga duch kelish mumkin. Gap shundaki, giperhavola uzunligi 2048 belgidan oshmasligi kerak. HTTP GET o'rniga HTTP POST dan foydalanish mumkin. POST usuli yanada moslashuvchan - u ishlatilganda parametrlar so'rov tanasida uzatiladi.

HTTP POST chaqiruvini amalga oshirish HTTP GET chaqiruviga juda o'xshaydi. *HttpClient* bilan HTTP POST so'rovini yaratish quyidagi tarsda amalga oshiriladi:

```
import java.util.ArrayList;
import java.util.List;
import org.apache.http.HttpResponse;
import org.apache.http.NameValuePair;
import org.apache.http.client.HttpClient;
import org.apache.http.client.entity.UrlEncodedFormEntity;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.impl.client.DefaultHttpClient;
```

```

import org.apache.http.message.BasicNameValuePair;
public class TestHttpPost {
public String executeHttpPost() throws Exception {
BufferedReader in = null; try {
HttpClient client = new DefaultHttpClient();
HttpPost request = new HttpPost(
"http://somewebsite/WS2/Upload.aspx");
List<NameValuePair> postParameters = new
ArrayList<NameValuePair>();
BasicNameValuePair("one", "valueGoesHere");
UrlEncodedFormEntity formEntity = new UrlEncodedFormEntity(
postParameters);
request.setEntity(formEntity);
HttpResponse response = client.execute(request);
in = new BufferedReader(new
InputStreamReader(response.getEntity().getContent()));
StringBuilder sb = new StringBuilder("");
String line =String NL - System.getProperty("line.separator");
while ((line = in.readLine()) != null) { sb.appendline + NL);
in.close();
String result = sb.toString(); return result;
} finally { if (in != null) { try { in.close();
} catch (IOException e) { e.printStackTrace();
} } }
}
}

```

HTTP POST qo'ng'iroqlarini amalga oshirishda "ism/qiyamat" shakli parametrlari odatda HTTP so'rovda uzatilgan URLning bir qismi sifatida kodlanadi. Ushbu operatsiyani bajarish uchun *NameValuePair* obyekt namunalari ro'yxatini yaratish va keyin bu ro'yxatni *UrlEncodedFormEntity* obyektiga qo'shish kerak.

NameValuePair obyekti "ism/qiyamat" kombinatsiyasini o'z ichiga oladi va *UrlEncodedFormEntity* sinfi keyinchalik HTTP chaqiruvlarida (odatda POST chaqiruvlari) foydalanish uchun *NameValuePair* obyektlari ro'yxatini kodlashi mumkin. *UrlEncodedFormEntity* yaratilgandan so'ng, *UrlEncodedFormEntity* ni *HttpPost* obyekt turi sifatida o'rnatish va keyin so'rovni yuborish mumkin. Keyinchalik, alohida "ism/qiyamat" juftliklari parametrlari bilan to'ldirilgan *NameValuePair* obyektlari ro'yxati yaratiladi. Parametr nomi bitta bo'ladi va qiymat *valueGoesHere* bo'ladi. Keyin *UrlEncodedFormEntity* nusxasi yaratiladi. Uning konstruktoriga

NameValuePair objects ro'yxati beriladi. Nihoyat, *setEntity* usulini chaqiriladi va POST so'rovi *HttpClient* misoli yordamida amalga oshiriladi.

HTTP POST oddiy "ism/qiyamat" parametrlarini hamda fayllar kabi murakkab parametrlarni uzatish imkonini beradi. HTTP POST ko'p qismli POST deb ataladigan boshqa so'rov formati bilan ham ishlashi mumkin. Ushbu turdan foydalanib, so'rov bilan nafaqat "ism/qiyamat" juftlarini, balki har qanday fayllarni ham yuborish mumkin. Afsuski, Android-ga kiritilgan *HttpClient* versiyasida ko'p qismli POST to'g'ridan-to'g'ri qo'llab-quvvatlanmaydi.

4.4. Server bilan ishlash

Ko'pgina mobil ilovalar mijoz-server arxitekturasidan foydalanadi. Umumiy sxema quyidagicha:

- server – masofaviy kompyuterda ishlaydigan va mijoz ilovalari bilan "muloqot" funksiyasini amalga oshiruvchi dastur (so'rovlarni tinglaydi, o'tkazilgan parametr va qiymatlarni taniydi, ularga to'g'ri javob beradi);
- mijoz – mobil qurilmada serverga tushunarli bo'lgan so'rovni shakllantirishi va olingan javobni o'qishi mumkin bo'lgan dastur;
- o'zaro aloqa interfeysi – har ikki tomon tomonidan so'rovlar/javoblarni yuborish/qabul qilish formati va usuli.

Server va u bilan ishlaydigan Android mijozini amalga oshiruvchi mobil ilovani ko'rib chiqamiz. Misol tariqasida, biz mobil Internet messenjeridan (Viber, ICQ) foydalanamiz va dastur "Internet chat" deb nomlanadi. A qurilmasiga o'rnatilgan mijoz B qurilmasiga o'rnatilgan mijozga xabar yuboradi va aksincha. Server A va B ... C, E ... va boshqalar o'rtasidagi aloqa rolini o'ynaydi. Shuningdek, u mijoz qurilmalaridan birida o'chirilgan taqdirda ularni qayta tiklash uchun xabarlarining "yig'guvchisi" rolini o'ynaydi.

Xabarlarini saqlash uchun serverda ham, mijoz qurilmalarida ham SQL ma'lumotlar bazasidan foydalanamiz. Bundan tashqari, Internet-chat qurilma ishga tushirilganda ishga tushishi va fonda ishlashi mumkin bo'ladi. O'zaro aloqa HTTP so'rovlari va JSON javoblari orqali amalga oshiriladi.

"Server" ni amalga oshirish uchun SQL va PHP bilan ishlashga imkon beruvchi har qanday xostingda ro'yxatdan o'tishimiz kerak.

Bo'sh SQL ma'lumotlar bazasini va unda jadval yaratamiz.

```
CREATE TABLE `chat` (
```

```
  `_id` int(11) NOT NULL AUTO_INCREMENT,
```

```
  `author` text CHARACTER SET utf8 COLLATE utf8_unicode_ci
NOT NULL,
```

```

`client` text CHARACTER SET utf8 COLLATE utf8_unicode_ci
NOT NULL,
`data` bigint(20) NOT NULL,
`text` text CHARACTER SET utf8 COLLATE utf8_unicode_ci NOT
NULL,
PRIMARY KEY (`id`))

```

Ma'lumotlar bazasi va undagi jadvalning tuzilishi quyidagicha:

1. author – xabar muallifi;
2. client – xabarni oluvchi;
3. data – serverda xabar qabul qilingan vaqt va sana;
4. text – matnli xabar.

Keyingi ikkita faylda "server" ni ro'yxatdan o'tkazishda foydalanuvchi tomonidan qabul qilingan ma'lumotlar bazasiga kirish uchun ma'lumotlarni o'z ichiga olgan o'zgaruvchilarni o'zgartirishimiz kerak.

```

$mysql_host = "localhost"; // sql server
$mysql_user = "129340eb_chat"; // foydalanuvchi
$mysql_password = "123456789"; // parol
$mysql_database = "129340eb_chat"; // chat ma'lumotlar bazasi nomi
Chat.php fayli API bo'lib, u serverga tushunarli bo'lgan so'rovlar
tuzilishini amalga oshiradi.

```

```

<?php
$mysql_host = "localhost"; // sql server
$mysql_user = "129340eb_chat"; // foydalanuvchi
$mysql_password = "123456789"; // parol
$mysql_database = "129340eb_chat"; // chat ma'lumotlar bazasi nomi
Biz ...chat.php?action=select so'rov qatorida o'tkazilgan
parametrlarni tekshiramiz. Action o'zgaruvchisi quyidagicha bo'lishi
mumkin:

```

- select - JSON-da chat jadvalining mazmunini shakllantiradi va uni qaytarib yuboradi;

- insert - suhbat jadvaliga yangi qator qo'shadi, bizga 4 ta parametr kerak: muallif/oluvchi/yaratish vaqti/xabar. Biz yaratish vaqtini parametrlarda o'tkazmaymiz, biz uni serverdagi joriy vaqtdan olamiz;

- delete - chat jadvalidagi barcha yozuvlarni o'chiradi.

// uzatilgan action ni oladi

```
if (isset($_GET["action"])) { $action = $_GET['action']; }
```

// agar action=insert bo'lsa, unda author|client|text ni ham olamiz

```
if (isset($_GET["author"])) { $author = $_GET['author']; }
```

```
if (isset($_GET["client"])) { $client = $_GET['client']; }
```

```

if (isset($_GET["text"])) { $text = $_GET['text']; }
if (isset($_GET["data"])) { $data = $_GET['data']; }
// SQL serveriga ulanish
mysql_connect($mysql_host, $mysql_user, $mysql_password);
// serverdagi ma'lumotlar bazasiga ulanish
mysql_select_db($mysql_database);
mysql_set_charset('utf8'); // kodlash
// agar mavjud bo'lsa, so'rovni qayta ishlash
if($action == select){ // agar harakat SELECT bo'lsa
if($data == null){
// chat jadvalidagi hamma ma'lumotlarni tanlaydi va
// ularni JSONga qaytaradi
$sql=mysql_query("SELECT * FROM chat");
}else{
// ma'lum vaqtdan keyin chat jadvalidagi barcha ma'lumotlarni
// tanlanadi va uni JSONga qaytaradi
$sql=mysql_query("SELECT * FROM chat WHERE data > $data"); }
while($e=mysql_fetch_assoc($sql))
$output[]=$e;
print(json_encode($output)); }
if($action == insert && $author != null && $client != null && $text
!= null){
// agar INSERT amali kerak bo'lsa
// vaqt = server vaqtiga, mijoz vaqti emas
$current_time = round(microtime(1) * 1000);
// skriptga ma'lumotlarni uzatish misoli:
// o'tkazilgan parametrlar bilan qatorni kiritish
mysql_query("INSERT INTO `chat`(`author`,`client`,`data`,`text`)
VALUES ('$author','$client','$current_time','$text');");
if($action == delete){
// yozuvlar jadvalini to'liq tiklash
mysql_query("TRUNCATE TABLE `chat`"); }
mysql_close();
?>

```

API so'rovi tuzilishi quyidagi tartibda amalga oshiriladi:

• majburiy action atributi – tanlashga teng bo'lishi mumkin (server o'z ma'lumotlar bazasidan yozuvlar ro'yxati bilan javob beradi), kiritish (server o'z ma'lumotlar bazasiga yangi yozuv qo'shadi), o'chirish (server o'z ma'lumotlar bazasini tozalaydi);

- agar `action=insert` bo'lsa, qo'shimcha parametrlarni o'tkazishimiz kerak bo'ladi: `author` (xabarni yozgan), `client` (xabar kimga qaratilgan), `text` (xabar);

- `action=select` qo'shimcha ma'lumotlar parametrini o'z ichiga olishi mumkin, bu holda server javobi ma'lumotlar bazasidan barcha xabarlarni emas, faqat yaratilish vaqti yuboritganidan kechroq bo'lgan xabarlarni o'z ichiga oladi.

Masalan:

- `chat.php?action=delete` – serverdagi barcha yozuvlarni o'chirib tashlaydi;

- `chat.php?action=insert&author=Jon&client=Smith&text=Salom` – serverga yangi yozuv qo'shadi: muallif Jon, qabul qiluvchi Smit, kontent Salom;

- `chat.php?action=select&data=151351333` - o'tgan vaqtdan keyin olingan barcha yozuvlarni uzoq formatda qaytaradi.

`FoneService.java` fonda ishlaydi, u alohida trekda har 15 soniyada serverga so'rov yuboradi. Agar server javobida yangi xabarlar bo'lsa, `FoneService.java` ularni mahalliy ma'lumotlar bazasiga yozadi va xabarlar bilan `ListView`-ni yangilash uchun `ChatActivity.java`-ga xabar yuboradi. `ChatActivity.java` (agar u hozir ochiq bo'lsa) xabarni oladi va mahalliy ma'lumotlar bazasidan `ListView` tarkibini yangilaydi.

`ChatActivity.java`-dan yangi xabar yuborish `FoneService.java`-ni chetlab o'tib, darhol serverga yuboriladi. Shu bilan birga, xabarimiz mahalliy ma'lumotlar bazasiga yozilmagan! U yerda server javobi ko'rinishida qaytarib olingandan keyingina paydo bo'ladi. Ushbu dasturni har qanday Internet chatining ishlashidagi muhim hodisa bilan bog'liq holda ishlatganda – xabarlarni vaqt bo'yicha majburiy guruhlash amalga oshirilgan. Vaqt bo'yicha guruhlashdan foydalanmasangiz, xabarlar ketma-ketligi buziladi. Mijoz ilovalarini jismonan millisekundlarda sinxronlash mumkin emasligini va hatto turli vaqt zonalarida ishlashi mumkinligini hisobga olsak, server vaqtdan foydalanish eng mantiqiy bo'ladi.

Yangi xabar yaratishda serverga so'rov yuboramiz: xabar muallifining ismi, xabarni qabul qiluvchining ismi, xabar matni. Ushbu yozuvni server javobi sifatida qaytarib olib, yuborgan ma'lumotni olamiz, shuningdek to'rtinchi parametr: server xabarni qabul qilgan vaqt.

`MainActivity.java`-da aniqlik uchun mahalliy ma'lumotlar bazasidan xabarlarni o'chirish imkoniyatini qo'shilgan – bu dasturni yangidan o'rnatishga teng (bu holda `FoneService` tanlangan barcha xabarlarni qabul qilish uchun serverga so'rov yuboradi). Bundan tashqari, serverda

joylashgan ma'lumotlar bazasidan barcha xabarlarni o'chirish uchun so'rov yuborish mumkin. Android platformasi Linux yadrosiga asoslangan va tarmoq vositalarining boy to'plamini o'z ichiga oladi.

2-jadvalda Android SDK-ga kiritilgan tarmoq bilan bog'liq ba'zi paketlar ro'yxati keltirilgan.

2-jadval.

Paket tavsifi

Paket nomi	Tavsifi
1	2
java.net	Tarmoq funksiyalari, jumladan, oqim va datagramma socketlari, IP protokoli va umumiy HTTP moslamalari bilan bog'liq sinflarni o'z ichiga oladi. Bu ko'p maqsadli tarmoq resursidir. Ushbu tanish paket yordamida tajribali Java dasturchilari darhol ilovalar yaratishni boshlashlari mumkin.
java.io	Ushbu paket juda muhim, garchi tarmoqlar bilan bevosita bog'liq bo'lmasa ham. Uning sinflari boshqa Java paketlarida joylashgan socketlar va ulanishlar tomonidan qo'llaniladi. Ular mahalliy fayllar bilan almashish uchun ham ishlatiladi (bu ko'pincha tarmoq bilan o'zaro aloqada bo'lganda sodir bo'ladi).
java.nio	Ma'lum turdagi ma'lumotlarni buferlovchi sinflarni o'z ichiga oladi. Java-dan foydalangan holda ikkita so'nggi nuqta o'rtasida tarmoq aloqasini o'rnatish uchun qulay.
org.apache.*	HTTP aloqalari uchun aniq nazorat va funktsionallikni ta'minlovchi paketlar to'plami. Bu Apache, mashhur ochiq manbali web-server.
android.net	Asosiy java.net.* sinflariga qo'shimcha ravishda qo'shimcha tarmoq kirish socketlarini o'z ichiga oladi. Ushbu paket an'anaviy web-funktsionallikdan tashqariga chiqadigan Android ilovalarini ishlab chiqishda tez-tez ishlatiladigan URI sinfini o'z ichiga oladi.
android.net.http	SSL sertifikatlari bilan ishlash uchun sinfarni o'z ichiga oladi.
android.net.wifi	Android platformasida WiFi (802.11 Wireless Ethernet) ning barcha jihatlarini amalga oshirish uchun darslarni o'z ichiga oladi. Hamma qurilmalar ham Wi-Fi

	imkoniyatlariga ega emas, ayniqsa Android Motorola va LG kabi uyali telefon ishlab chiqaruvchilarining telefon segmentiga kirib kelayotgani uchun.
android.telephony.gsm	SMS (matnli) xabarlarni boshqarish va yuborish uchun zarur bo'lgan sinflarni o'z ichiga oladi. Vaqt o'tishi bilan, CDMA kabi GSM bo'lmagan tarmoqlarda shunga o'xshash funksiyalarni ta'minlaydigan qo'shimcha paketlar paydo bo'ladi, masalan, android.telephony.cdma.

Yuqoridagi ro'yxat to'liq emas, lekin bu platforma nimaga qodirligi haqida umumiy fikr beradi. Android-ni internetga ulash qanchalik oson ekanligini ko'rsatish uchun bu yerda web-sahifadan matn chiqarish misoli keltirilgan. Avval foydalanuvchi interfeysi jihatlarini ko'rib chiqamiz, keyin esa tarmoq bilan bog'liq kodni ko'rib chiqamiz. Bu yerda uchta foydalanuvchi interfeysi elementi mavjud:

- *EditText* web-sahifani belgilash imkonini beradi (rasmda ko'rsatilgan).

- Tugma dasturga web-sahifadan matnni chiqarishni buyurish uchun ishlatiladi.

- qabul qilingan ma'lumotlar *TextView*da ko'rsatiladi.

Quyidagi ilova kodida foydalanuvchi interfeysining to'liq tartibi bo'lgan *main.xml* fayl ko'rsatilgan.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent" >
    <EditText
        android:layout_height="wrap_content"
        android:id="@+id/address"
        android:layout_width="fill_parent"
        android:text="http://google.com">
    </EditText>
    <Button
        android:id="@+id/ButtonGo"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content">
```

```
        android:text="go!" >
    </Button>
    <TextView
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:background="#ffffff"
        android:textColor="#000000"
        android:id="@+id/pagetext" />
</LinearLayout>
Ushbu misol uchun GetWebPage.java kodi quyida ko'rsatilgan.
package com.msi.getwebpage;
import android.app.Activity;
import android.os.Bundle;
// used for interacting with user interface
import android.widget.Button;
import android.widget.TextView;
import android.widget.EditText;
import android.view.View;
// used for passing data
import android.os.Handler;
import android.os.Message;
// used for connectivity
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.net.URL;
import java.net.URLConnection;
public class GetWebPage extends Activity {
    /** Activity birinchi marta yaratilganda chaqiriladi */
    Handler h;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        final EditText eText = (EditText) findViewById(R.id.address);
        final TextView tView = (TextView) findViewById(R.id.pagetext);
        this.h = new Handler() {
            @Override
            public void handleMessage(Message msg) {
                // bu yerda kiruvchi xabarlarni qayta ishlash keltirilgan
```



```

switch (msg.what) {
    case 0:
        tView.append((String) msg.obj);
        break;
    }
super.handleMessage(msg);
}
final Button button = (Button) findViewById(R.id.ButtonGo);
button.setOnClickListener(new Button.OnClickListener() {
    public void onClick(View v) {
        try {
            tView.setText("");
            // Bosish orqali amalni bajarish
            URL url = new URL(eText.getText().toString());
            URLConnection conn = url.openConnection();
            // Javobni olish
            BufferedReader rd = new BufferedReader(new
InputStreamReader(conn.getInputStream()));
            String line = "";
            while ((line = rd.readLine()) != null) {
                Message lmsg;
                lmsg = new Message();
                lmsg.obj = line;
                lmsg.what = 0;
                GetWebPage.this.h.sendMessage(lmsg);
            }
        } catch (Exception e) {
            // ...
        }
    }
});
}
}

```

Ushbu kodni bir nechta umumiy sohalarga bo'lish mumkin. Bir nechta muhim (majburiy) import bayonotlari ilovada ishlatiladigan foydalanuvchi interfeysi, o'tish va tarmoq funksiyalari sinflariga tegishli havolalarni taqdim etadi. Tarmoqlar bilan ishlashga oid barcha kodlar *OnClickListener* sinfining *onClick* usulida joylashgan. U belgilangan tugma bosilganda olinadi.

URL va *URLConnection* sinflari foydalanuvchi tanlagan web-saytga haqiqiy ulanishni ta'minlaydi. *BufferedReader* misoli web-sayt ulanishidan ma'lumotlarni o'qiydi. Har bir satr o'qilishi bilan uning matni *TextView*-ga qo'shiladi. Ushbu ma'lumotlar oddiygina *TextView*-ga tayinlanmagan. Xabar obyektini yaratish va ushbu obyektini ishlov beruvchi misoliga o'tkazish mexanizmini ko'rib chiqamiz. Bu, ayniqsa, bir vaqtning o'zida bir nechta ish zarrachalari ishlashi mumkin bo'lgan real dunyo ilovalarida foydalanuvchi interfeysini yangilashning ma'qul yo'lidir.

Ushbu misolda Android ilovasi *Apache* yoki *Internet Information Server* (Microsoft® serveridagi IIS) kabi HTTP web-serverlari bilan bog'lanadi. Agar dastur HTTP o'rniga TCP socketiga to'g'ridan-to'g'ri kirishi kerak bo'lsa, u boshqacha tarzda tuzilishi kerak edi. Quyida uzoq server bilan ishlashning boshqa vositalarini ko'rsatadigan kod parchasi keltirilgan. Ushbu kod alohida oqim sifatida amalga oshiriladi.

```

// Vaqt mijoz
public class Requester extends Thread {
    Socket requestSocket;
    String message;
    StringBuilder returnStringBuffer = new StringBuilder();
    Message lmsg;
    int ch;
    @Override
    public void run() {
        try {
            his.requestSocket = new Socket("remote.servername.com", 13);
            InputStreamReader isr = new
InputStreamReader(this.requestSocket);
            getInputStream(), "ISO-8859-1");
            while ((this.ch = isr.read()) != -1) {
                this.returnStringBuffer.append((char) this.ch);
            }
            this.message = this.returnStringBuffer.toString();
            this.lmsg = new Message();
            this.lmsg.obj = this.message;
            this.lmsg.what = 0;
            h.sendMessage(this.lmsg);
            this.requestSocket.close();
        } catch (Exception ee) {
            Log.d("sample application", "failed to read data" +
ee.getMessage());
        }
    }
}
}
}

```

Oldingi misolda bo'lgani kabi, yuqoridagi kod ma'lumotlarni jo'natuvchiga UI yangilanishi va keyingi ishlov berish bilan qaytarish uchun xabar va ishlov beruvchi yondashuvdan foydalanadi. Yuqoridagi koddan farqli o'laroq, bu misol HTTP serveri bilan bog'lanmaydi, shuning uchun *URLConnection* sinfi ishlatilmaydi. Buning o'rniga, quyi darajadagi *Socket* sinfi 13-portda masofaviy serverga oqim socket ulanishini ochadi. Port 13 oddiy "vaqt serveri" ilovasidir.

Vaqt serveri kirituvchi socketga ulanishni qabul qiladi va sana va vaqtni matn formatida chaqiruvchi socketga qaytaradi. Ma'lumot yuborilgandan so'ng, server socketni yopadi. Ushbu misol, shuningdek, *InputStreamReader()* usuli va maxsus kodlashdan foydalanishni ko'rsatadi.

Android yordamida hal qilinishi mumkin bo'lgan yana bir vazifa - matnli xabarlar yuborish. Quyidagi misolda matnli xabar yuborish ko'rsatilgan.

```
void sendMessage(String recipient,String myMessage) {
    SmsManager sm = SmsManager.getDefault();
    sm.sendTextMessage("destination number", null, "hello there", null,
    null);
}
```

Matnli xabar yuborish jarayoni juda oddiy. Birinchidan, *getDefault()* statik usuli yordamida *SmsManager*ga havola olamiz. Keyin *sendTextMessage* usulidan foydalanamiz va uning argumentlar quyidagilar:

- qabul qiluvchining mobil telefon raqami, hudud kodini o'z ichiga oladi;

- xizmat ko'rsatish markazining telefon raqami, *null* qiymati xabarlar qayta ishlash uchun standart xizmat ko'rsatish markazidan foydalanishga roziligingizni bildiradi. Eng ilg'or ilovalardan tashqari barchasida ushbu parametr uchun *null*-dan foydalaniladi;

- sizning xabaringiz, xabar uzunligi 160 baytdan kam bo'lishi kerak, aks holda ma'lumotlar bir nechta xabarlarga bo'linadi;

- *Intent*-ni yuborish, xabar yuborilgandan keyin yoki xatolik yuz berganda boshlangan ixtiyoriy harakat (*intent*). Agar bunday bildirishnoma talab etilmasa, ushbu parametr *null* deb o'tkazilishi mumkin;

- *Intent*-ni etkazib berish, yetkazib berishni tasdiqlaganidan keyin boshlangan ixtiyoriy harakat (*intent*). Agar bunday bildirishnoma talab etilmasa, ushbu parametr *null* deb o'tkazilishi mumkin.

4.5. Foydalanuvchining joylashuvini aniqlash

Android qurilmalari hozirgi joylashuvimiz haqida ma'lumot berishi mumkin. Bu juda qulay, masalan, xaritadan foydalanish, hududingizga tegishli ma'lumotlarni olish (ob-havo prognozi) va hokazo.

Buning uchun provayder ishlatiladi va ma'lumotlarni provayder orqali olish mumkin. Hozirgi vaqtda ikkita provayder mavjud: *GPS* va *Network*.

GPS - bu GPS sun'iy yo'ldoshlaridan olingan ma'lumotlar.

Network - bu uyali yoki WiFi orqali olinishi mumkin bo'lgan koordinatalar. Ushbu provayder ishlashi uchun Internet kerak.

Koordinatalarni so'raydigan va ko'rsatadigan oddiy ilovani yozaylik. Yangi loyiha yaratamiz va *strings.xml* ga quyidagi qatorlarni qo'shamiz:

```
<string name="provider_gps">GPS</string>
<string name="provider_network">Network</string>
```

```
<string name="location_settings">Location settings</string>
```

Shuningdek *main.xml* da mobil ilovaning akslanishi uchun quyidagi tarzda elementlarni joylashingiz:

```
<?xml
    version="1.0"
    encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="5dp">
<TextView
    android:id="@+id/tvTitleGPS"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/provider_gps"
    android:textSize="30sp">
</TextView>
<TextView
    android:id="@+id/tvEnabledGPS"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textSize="24sp">
</TextView>
<TextView
    android:id="@+id/tvStatusGPS"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textSize="24sp">
</TextView>
<TextView
    android:id="@+id/tvLocationGPS"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textSize="24sp">
```

```

</TextView>
<TextView
  android:id="@+id/tvTitleNet"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:layout_marginTop="10dp"
  android:text="@string/provider_network"
  android:textSize="30sp">
</TextView>
<TextView
  android:id="@+id/tvEnabledNet"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:textSize="24sp">
</TextView>
<TextView
  android:id="@+id/tvStatusNet"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:textSize="24sp">
</TextView>
<TextView
  android:id="@+id/tvLocationNet"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:textSize="24sp">
</TextView>
<Button
  android:id="@+id/btnLocationSettings"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:layout_marginTop="10dp"
  android:onClick="onClickLocationSettings"
  android:text="@string/location_settings">
  </Button></LinearLayout>

```

MainActivity.java fayli quyidagi ko'rinishga ega:

```

package com.develop.location;
import java.util.Date;
import android.app.Activity;

```

```

import android.content.Intent;
import android.location.Location;
import android.location.LocationListener;
import android.location.LocationManager;
import android.os.Bundle;
import android.view.View;
import android.widget.TextView;
public class MainActivity extends Activity {
  TextView tvEnabledGPS;
  TextView tvStatusGPS;
  TextView tvLocationGPS;
  TextView tvEnabledNet;
  TextView tvStatusNet;
  TextView tvLocationNet;
  private LocationManager locationManager;
  StringBuilder sbGPS = new StringBuilder();
  StringBuilder sbNet = new StringBuilder();
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    tvEnabledGPS = (TextView) findViewById(R.id.tvEnabledGPS);
    tvStatusGPS = (TextView) findViewById(R.id.tvStatusGPS);
    tvLocationGPS = (TextView)
  findViewById(R.id.tvLocationGPS);
    tvEnabledNet = (TextView) findViewById(R.id.tvEnabledNet);
    tvStatusNet = (TextView) findViewById(R.id.tvStatusNet);
    tvLocationNet = (TextView) findViewById(R.id.tvLocationNet);
    locationManager = (LocationManager)
  getSystemService(LOCATION_SERVICE); }
  @Override
  protected void onResume() {
    super.onResume();
    locationManager.requestLocationUpdates(LocationManager.GPS_
  PROVIDER, 1000 * 10, 10, locationListener);
    locationManager.requestLocationUpdates(
    LocationManager.NETWORK_PROVIDER, 1000 * 10, 10,
    locationListener);
    checkEnabled(); }

```

```

@Override
protected void onPause() {
    super.onPause();
    locationManager.removeUpdates(locationListener); }
private LocationListener locationListener = new LocationListener()
{
    @Override
    public void onLocationChanged(Location location) {
        showLocation(location); }
    @Override
    public void onProviderDisabled(String provider) {
        checkEnabled(); }
    @Override
    public void onProviderEnabled(String provider) {
        checkEnabled();
        showLocation(locationManager.getLastKnownLocation(provider
)); }
    @Override
    public void onStatusChanged(String provider, int status, Bundle
extras) {
        if (provider.equals(LocationManager.GPS_PROVIDER)) {
            tvStatusGPS.setText("Status: " + String.valueOf(status));
        } else if
(provider.equals(LocationManager.NETWORK_PROVIDER)) {
            tvStatusNet.setText("Status: " + String.valueOf(status)); } } };
private void showLocation(Location location) {
    if (location == null)
        return;
    if
(location.getProvider().equals(LocationManager.GPS_PROVIDER))
{
        tvLocationGPS.setText(formatLocation(location));
    } else if (location.getProvider().equals(
LocationManager.NETWORK_PROVIDER)) {
        tvLocationNet.setText(formatLocation(location)); } }
private String formatLocation(Location location) {
    if (location == null)
        return "";
    return String.format(

```

```

"Coordinates: lat = %1$.4f, lon = %2$.4f, time = %3$.4f %3$.4f",
location.getLatitude(), location.getLongitude(), new Date(
location.getTime())); }
private void checkEnabled() {
    tvEnabledGPS.setText("Enabled: "
+ locationManager
.isProviderEnabled(LocationManager.GPS_PROVIDER));
    tvEnabledNet.setText("Enabled: "
+ locationManager
.isProviderEnabled(LocationManager.NETWORK_PROVID
ER)); }
public void onClickLocationSettings(View view) {
    startActivity(new Intent(
android.provider.Settings.ACTION_LOCATION_SOURCE_SE
TTINGS));};
}

```

onCreate() usulida *TextView* komponentlarini aniqlaymiz va *LocationManager*-ni olamiz, u orqali ishlaymiz. *OnResume*-da *requestLocationUpdates()* usulidan foydalanamiz va uning kirishiga quyidagilarni uzatamiz:

- provayder turi: *GPS_PROVIDER* yoki *NETWORK_PROVIDER*;
 - ma'lumotlarni qabul qilish orasidagi minimal vaqt (millisekundlarda). Kodda 10 soniya keltirilgan. Koordinatani kechiktirmasdan olish uchun 0 soniya kiritish mumkin. Ammo bu faqat minimal vaqt. Ma'lumotni qabul qilishning haqiqiy kutilishi uzoqroq bo'lishi mumkin.
 - minimal masofa (metrda). Bular agar joylashuvingiz belgilangan metrlar soniga o'zgargan bo'lsa, unda yangi koordinatalarni olasiz.
 - tinglovchi, *locationListener* obyekt.
- Shuningdek, provayderlarni kiritish haqidagi ma'lumotlarni yangilash mumkin.

OnPause()-da *removeUpdates()* usulini o'chirib qo'yish lozim. *locationListener* - *LocationListener* interfeysini quyidagi usullar bilan amalga oshiradi:

onLocationChanged - *Location* obyekt uchun yangi joylashuv ma'lumotlari. Bu yerda *showLocation()* usuli chaqiriladi, u joylashuv ma'lumotlarini ekranda aks ettiradi.

onProviderDisabled - ko'rsatilgan provayderni o'chirib qo'yish. Ushbu usulda ekrandagi provayderlarning joriy holatini yangilaydigan *checkEnabled()* usulini chaqiramiz.

onProviderEnabled - ko'rsatilgan provayderni yoqish. Bu erda *checkEnabled()* chaqiriladi. Keyinchalik, *getLastKnownLocation()* usuli (u *null*-ni qaytarishi mumkin) yoqilgan provayderdan oxirgi mavjud joyni so'raydi va uni ko'rsatadi. Agar ilgari biron bir joylashuvga asoslangan ilovadan foydalangan bo'lsangiz, bu juda dolzarb bo'lishi mumkin.

onStatusChanged - ko'rsatilgan provayderning holatini o'zgartirish. Holat maydonida *OUT_OF_SERVICE* (ma'lumotlar uzoq vaqt davomida mavjud bo'lmaydi), *TEMPORARILY_UNAVAILABLE* (ma'lumotlar vaqtincha mavjud emas), *AVAILABLE* (ma'lumotlar mavjud) qiymatlarini o'z ichiga olishi mumkin. Ushbu usulda shunchaki ekranda yangi holatni ko'rsatamiz.

Provayderlar tizim sozlamalarida yoqilgan va o'chirilgan bo'ladi. Shunday qilib, provayder undan koordinatalarni olish uchun mavjudmi yoki yo'qmi oddiygina aniqlanadi. Standart usullar orqali provayderlarni dasturiy yoqish/o'chirish mavjud emas. *ShowLocation* usuli kirish sifatida *Location*-ni oladi, *getProvider()* usuli yordamida uning provayderini aniqlaydi va tegishli matn maydonida koordinatalarni aks ettiradi. *FormatLocation()* usuli *Location*-ni kiritish sifatida qabul qiladi, undan ma'lumotlarni o'qiydi va undan qatoni formatlaydi: *getLatitude* - kenglik, *getLongitude* - uzunlik, *getTime* - aniqlash vaqti. *checkEnabled()* usuli provayderlar *isProviderEnabled()* usuli yordamida yoqilgan yoki o'chirilganligini aniqlaydi va bu ma'lumotni ekranda ko'rsatadi.

Location sozlamalari tugmasi bosilganda *onClickLocationSettings()* usuli ishga tushadi va foydalanuvchi provayderni yoqishi yoki o'chirib qo'yishi uchun sozlamalarni ochadi. Buni amalga oshirish uchun *action* bilan *Intent*-dan foydalaniladi *action* = *ACTION_LOCATION_SOURCE_SETTINGS*.

Ilova manifestida koordinatalarni aniqlash uchun ruxsatni ko'rsatish kerak - *ACCESS_FINE_LOCATION*, bu sizga tarmoqdan ham, GPSdan ham foydalanish imkonini beradi. *ACCESS_COARSE_LOCATION* ruxsati ham mavjud bo'lib, u faqat tarmoq provayderiga kirish imkonini beradi. Biz dasturni saqlaymiz va ishga tushiramiz. Agar mobil qurilmada GPS o'chirilgan bo'lsa, WiFi o'chirilgan va mobil Internet o'chirilgan bo'lsa, ilovani ishga tushirganingizda quyidagi ko'rinish paydo bo'ladi (4.4-rasm):

Location

GPS

Enabled: false

Network

Enabled: true

Location settings

4.4-rasm. Ilovani ishga tushirish

Bu yerda GPS o'chirilgan, Network yoqilgan. Ammo Internet yoqilmaganligi sababli, Network ma'lumotlarni ko'rsatmaydi. Mobil Internet yoki Wi-Fi yoqilgan bo'lishi kerak. WiFi yoqilgandan so'ng, ko'rinish o'zgaradi va 15-20 soniyadan so'ng Network joriy joylashuv haqidagi ma'lumotlarni ko'rsatadi. Bular kenglik, uzunlik va vaqt (4.5-rasm).

Location

GPS

Enabled: false

Network

Enabled: true

Coordinates: lat = 55.7541, lon = 37.6209, time = 2013-12-14 20:23:46

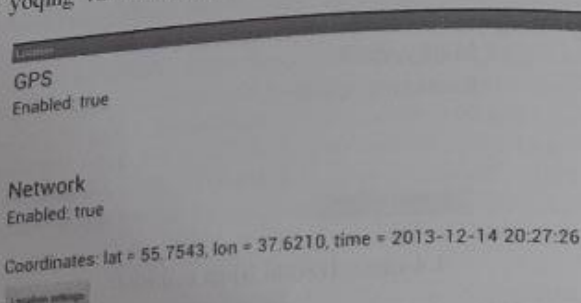
Location settings

4.5-rasm. Joriy joylashuv haqidagi ma'lumotlarni ko'rsatish

Kodda ma'lumotni yangilashning minimal tezligi 10 soniya. Ammo tarmoq provayderi ko'proq narsaga ega bo'lishi mumkin.

Endi GPS-ni yoqaylik. Buni amalga oshirish uchun "Joylashuv sozlamalari" tugmasini maxsus qo'yamiz, foydalanuvchi sozlamalarga o'tish uchun uni bosishi kerak bo'ladi.

Ilova GPS o'chirilganligini va Network yoqilganligini ko'rsatadi. Albatta, GPS-ni tizimning tezkor sozlamalari (yuqori o'ng) orqali yoqish va o'chirish mumkin. Lekin hamma foydalanuvchilar bu haqda bilishmaydi. GPS-ni yoqing va ilovaga qaytish uchun Orqaga tugmasini bosing (4.6-rasm).



Location
GPS
Enabled: true

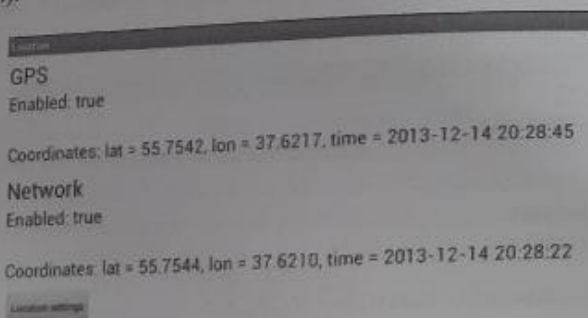
Network
Enabled: true

Coordinates: lat = 55.7543, lon = 37.6210, time = 2013-12-14 20:27:26

Location settings

4.6-rasm. GPS ni qo'shish

Endi GPS yoqilganligini ko'rsatadi, biz koordinatalarni kutmoqdamiz (4.7-rasm).



Location
GPS
Enabled: true

Coordinates: lat = 55.7542, lon = 37.6217, time = 2013-12-14 20:28:45

Network
Enabled: true

Coordinates: lat = 55.7544, lon = 37.6210, time = 2013-12-14 20:28:22

Location settings

4.7-rasm. Kiritilgan GPS

Bir muncha vaqt o'tgach, GPS 2-holatni yoqdi (*AVAILABLE*) (4.8-rasm).

Network ning holati ko'rsatilmaydi. Agar GPS signali yaxshi bo'lsa, har 10 soniyada siz joylashuvingiz haqida ma'lumot olasiz. Agar signal yomon bo'lsa, ma'lumotlar kamroq kelishi mumkin va holat ba'zan 1 ga o'zgaradi (*TEMPORARILY_UNAVAILABLE*). Uchinchi provayder turi – *PASSIVE_PROVIDER*. O'z-o'zidan, bu provayder hech qanday ma'lumotni

qaytarmaydi. Ammo tizimdagi boshqa birov oddiy provayderlar orqali manzilni aniqlashga urinayotganda joylashuv ma'lumotlarini olishingiz mumkin bo'ladi. Tizim foydalanuvchi uchun natijalarni takrorlaydi.



Location

GPS

Enabled: true

Status: 2

Coordinates: lat = 55.7544, lon = 37.6207, time = 2013-12-14 20:31:15

Network

Enabled: true

Coordinates: lat = 55.7544, lon = 37.6209, time = 2013-12-14 20:31:02

Location settings

4.8-rasm. Status o'zgarishi

getAllProviders() usuli barcha mavjud provayderlar ro'yxatini qaytaradi. *getProviders(boolean enabledOnly)* usuli barcha yoki faqat yoqilganlarini qaytaradi. *Location* obyektini koordinatalar, vaqt va provayderga qo'shimcha ravishda bo'sh bo'lishi mumkin bo'lgan yana bir nechta atributlarga ega:

- *getAccuracy()* – metrlarda ko'rsatish aniqligi;
- *getAltitude()* – balandlik metrlarda;
- *getSpeed()* – m/s da harakat tezligi;
- *getBearing()* – joriy harakat yo'li shimolga yo'ldan og'ish burchagi.

Location AVD emulyatori orqali ham tekshirilishi mumkin. Buni amalga oshirish uchun DDMS-ni ochish (Window > Open Perspective > DDMS) va Emulyatorni boshqarish yorlig'ini tanlash lozim. Pastki qismida koordinatalarni kiritish uchun maydonlar va yuborish tugmasi bo'lgan "Manual" yorlig'i bo'ladi.

Nazorat savollari:

1. Mobil ilovalardagi SQLite ma'lumotlar bazalarini tavsiflab bering.
2. Ma'lumotlar bazasi qanday yaratiladi?
3. Ma'lumotlar bazasi bilan qanday ishlash mumkin?
4. Qanday qilib so'rovlarni yaratishim mumkin?
5. Ma'lumotlar bazasi bilan ishlashda qanday usullardan foydalaniladi?
6. Kontent provayder tushunchasini tavsiflang.
7. Kontent provayderning yaratilishini tavsiflang.

8. Yaratilgan kontent provayderlaridan qanday foydalaniladi?
9. Kontent provayderlar bilan ishlashda qanday usullardan foydalaniladi?
10. Xabar xizmatlari qanday ishlaydi?
11. Xabarlar qanday qabul qilinadi va uzatiladi?
12. Foydalanuvchi interfeysi orqali xabarlar qanday almashinadi?
13. Server bilan ishlashni tavsiflang.
14. HTTP GET orqali serverga ulanish qanday amalga oshiriladi?
15. HTTP POST orqali serverga ulanish qanday amalga oshiriladi?

5. MOBIL DATCHIKLAR BILAN ISHLASH

5.1. Androidning sensor imkoniyatlari

Sensorli boshqaruvlar mobil ilova bilan ishlash uchun tegish imo-ishoralaridan foydalanishni o'z ichiga oladi. Quyida Android tizimi tomonidan qo'llab-quvvatlanadigan imo-ishoralar to'plami keltirilgan:

- *Bosish*: tanlangan element uchun standart amalni ishga tushirish;
- *Uzoq bosish*: elementni tanlash. Ushbu imo-ishorani kontekst menyusiga qo'ng'iroq qilish uchun ishlatmaslik kerak;
- *Surish yoki sudrab olish*: Tarkibni aylantirish yoki bir xil darajadagi interfeys elementlari orasida harakatlanish;
- *Uzoq bosishdan keyin siljish*: Ma'lumotlarni qayta guruhlash yoki ularni konteynerga ko'chirish;
- *Ikki marta bosish*: Kattalashtirish, matnni ajratib ko'rsatish;
- *Ikki marta bosish bilan sudrab olib tashlash*: o'lchamini o'zgartirish, imo-ishora markaziga nisbatan kengaytirish yoki qisqartirish;
- *Barmoqlarning qisqarishi (chimchilab yopish)*: tarkibni kamaytirish, qulash;
- *Ko'paytirish barmoqlari (chimchilab ochiladi)*: tarkibni oshirish, kengaytirish.

Ilovani sensorli imo-ishoralar yordamida boshqarish qobiliyati, agar ilova ma'lum bir imo-ishora ekrandagi tegishlar to'plami ostida yashiringanligini va tegishli harakatni bajarsa hisobga olinadi deyish mumkin. Imo-ishorani aniqlash jarayoni odatda ikki bosqichdan iborat: ma'lumotlarni yig'ish va imo-ishorani aniqlash. Keling, ushbu bosqichlarni batafsil ko'rib chiqaylik. Sensorli ekran bilan ishlashda foydalanuvchi bajarishi mumkin bo'lgan asosiy harakatlar: barmoq bilan ekranga tegish, barmoqni ekran bo'ylab harakatlantirish va qo'yib yuborish. Ushbu harakatlar Android tizimi tomonidan tegish hodisalari (tegish hodisalari sifatida tan olinadi. Har safar sensorli hodisa sodir bo'lganda, *onTouchEvent()* usuli chaqiriladi. Agar ushbu usul *Activity* sinfida yoki biron bir komponentda amalga oshirilsa, hodisani boshqarish mumkin bo'ladi, aks holda hodisa e'tiborga olinmaydi.

Imo-ishora ekranga birinchi marta tegish bilan boshlanadi, tizim foydalanuvchi barmoqlarining holatini kuzatayotganda davom etadi va yakuniy hodisa bilan yakunlanadi: barmoqlar ekranga tegmaydi. *onTouchEvent()* usuliga o'tkazilgan *MotionEvent* obyekt har bir o'zaro ta'sirning tafsilotlarini beradi. Sensorli hodisalarni aniqlaydigan *MotionEvent* sinfining asosiy o'zgarma-lari quyida keltirilgan:

- *MotionEvent.ACTION_DOWN* – barmoq bilan ekranga tegish har qanday tegish hodisasi yoki imo-ishoraning boshlang'ich nuqtasidir;
- *MotionEvent.ACTION_MOVE* – barmoqni ekran bo'ylab harakatlantirish;

- *MotionEvent.ACTION_UP* – barmoqni ekrandan ko'tarish.

Ilova imo-ishorani aniqlash uchun taqdim etilgan ma'lumotlardan foydalanishi mumkin. Imo-ishoralarni aniqlash uchun o'zingizning hodisangizni boshqarishingiz mumkin, shuning uchun ilovada o'zboshimchalik bilan ishoralar bilan ishlashingiz mumkin. Agar ilova standart imo-ishoralardan foydalanishi kerak bo'lsa, *GestureDetector* sinfidan foydalanishingiz mumkin. Bu sinf individual tegish hodisalari bilan ishlamasdan standart imo-ishoralarni tanib olish imkonini beradi.

Android standart imo-ishoralarni tanib olish uchun *GestureDetector* sinfini taqdim etadi. U qo'llab-quvvatlaydigan ba'zi imo-ishoralar qatoriga quyidagilar kiradi: *onDown()*, *onLongPress()*, *onFling()* va boshqalar. *GestureDetector* sinfidan *onTouchEvent()* usuli bilan birgalikda foydalanish mumkin.

Android 1.6 versiyasidan boshlab *android.gesture* paketida joylashgan imo-ishoralarni saqlash, yuklash, yaratish va tanib olish imkonini beruvchi ishoralar API-sini taqdim etadi. Xuddi shu 1.6 versiyasidan boshlab *Android Virtual Device (AVD)* imo-ishoralarni yaratishga imkon beruvchi *Gesture Builder* deb nomlangan oldindan o'rnatilgan dasturni o'z ichiga oladi. Yaratilgandan so'ng, imo-ishoralar virtual qurilmaning SD-kartasida saqlanadi va ularni ikkilik manba sifatida ilovaga qo'shish mumkin.

Imo-ishoralarni tanib olish uchun *Activity* XML fayliga *GestureOverlayView* komponentini qo'shish kerak. Ushbu komponent oddiy GUI elementi sifatida qo'shilishi va *RelativeLayout* kabi tartibga kiritilishi mumkin. Boshqa tomondan, u boshqa komponentlar ustidagi shaffof qatlam sifatida ishlatilishi mumkin, bu holda u *Activity* XML faylida ildiz elementi sifatida yozilishi kerak. Yuqoridagilarning barchasiga qo'shimcha ravishda, ilovada maxsus imo-ishoralardan foydalanish uchun *OnGesturePerformedListener* interfeysi va uning *onGesturePerformed()* usulini qo'llash kerak.

Standart imo-ishoralar bilan ishlash uchun Android *GestureDetector* sinfini taqdim etadi. Bu sinf ikkita ichki o'rnatilgan tinglovchi interfeysini o'z ichiga oladi: *OnGestureListener* va *OnDoubleTapListener*, bu interfeyslar standart imo-ishoralarni tinglovchi usullarni belgilaydi. *GestureDetector* shuningdek, *SimpleOnGestureListener* o'rnatilgan sinfni ham o'z ichiga oladi, u barcha interfeys usullaridan tegishli bo'lganda *false-*

ni qaytaradigan bo'sh ilovalarni o'z ichiga oladi: *OnGestureListener* va *OnDoubleTapListener*. Biz qo'llab-quvvatlanadigan barcha imo-ishoralarning tan olinishini namoyish etadigan dastur ishlab chiqamiz. Ilovada tan olingan imo-ishora haqidagi ma'lumotlarni ko'rsatish uchun bitta *Activity*, bitta ma'lumot maydoni mavjud bo'ladi. Ilova quyidagicha ishlaydi: foydalanuvchi qo'llab-quvvatlanadigan tegish imo-ishoralardan birini amalga oshiradi, ma'lumot maydonida tan olingan imo-ishora haqidagi ma'lumotlar ko'rsatiladi.

1. Oddiy dastur yaratamiz va ma'lumotni ko'rsatish uchun formaga *TextView* qo'shamiz.

2. Ilova mantig'ini ishlab chiqamiz. *Activity*-ga mos keladigan *java* sinfiga quyidagi qo'shimchalarni kiritamiz.

o *Activity* sinfi interfeyslarni amalga oshirishi kerak: *GestureDetector.OnGestureListener* va *GestureDetector.OnDoubleTapListener*, buning uchun sinf deklaratsiyasiga quyidagi tuzilmani qo'shamiz:

```
implements GestureDetector.OnGestureListener,  
GestureDetector.OnDoubleTapListener;
```

o Bizga *GestureDetectorCompat* sinfining nusxasi kerak bo'ladi, shuning uchun quyidagi o'zgaruvchini *Activity* sinfining maydoni sifatida e'lon qilamiz:

```
GestureDetectorCompat mDetector;
```

Activity sinfining *onCreate()* usulida *GestureDetectorCompat* sinfining nixsasini yaratib va uni *mDetector* o'zgaruvchisiga tayinlaymiz:

```
mDetector = new GestureDetectorCompat(this,this);
```

konstruktoring parametrlaridan biri *GestureDetector.OnGestureListener* interfeysini amalga oshiradigan sinfdir, bizning holatlarimizda *this* so'zi ishlatiladi, ya'ni parametr *Activity* sinfining o'zi. Ushbu interfeys ma'lum bir tegish hodisasi sodir bo'lganda foydalanuvchilarni xabardor qiladi.

Activity sinfining *onCreate()* usulida quyidagi qator:

```
mDetector.setOnDoubleTapListener(this);
```

GestureDetector.OnDoubleTapListener interfeysini amalga oshiradigan sinf bo'lishi kerak bo'lgan ikki marta bosish hodisalari uchun tinglovchini o'rnatadi. Bizning holatda, *this* so'zi ishlatiladi, ya'ni tinglovchi yana *Activity* sinfining o'zi bo'ladi.

GestureDetector obyektigizga hodisalarni qabul qilishiga ruxsat berish uchun *Activity* yoki GUI elementida *onTouchEvent()* usulini bekor qilish kerak va barcha aniqlangan hodisalarni detektor misolidan o'tkazish lozim.


```

public boolean onTouchEvent(MotionEvent event){
    this.mDetector.onTouchEvent(event);
    // Superklassni amalga oshirish uchun chaqirish
    return super.onTouchEvent(event);
}

```

Tayyorgarlikdan so'ng, tegish hodisalarini tinglash uchun javobgar bo'lgan interfeyslarda e'lon qilingan barcha usullarni amalga oshirish vaqti keldi.

GestureDetector.OnGestureListener interfeysi usullari:

onDown() – tegish ko'rinishini kuzatib boradi, ya'ni barmoq ekranga bosiladi;

onFling() – surish ishorasining ko'rinishini kuzatib boradi;

onLongPress() – barmoqni ekranga uzoq vaqt bosib ushlab turishni kuzatib boradi;

onScroll() – aylantirish imo-ishorasining ko'rinishini kuzatib boradi (surish);

onShowPress() – sensorli hodisa sodir bo'lganligini va qisqa vaqt ichida boshqa hodisalar sodir bo'lmaganligini kuzatib boradi;

onSingleTapUp() – bir marta bosish imo-ishorasining ko'rinishini kuzatib boradi (klik).

GestureDetector.OnDoubleTapListener interfeysi usullari:

onDoubleTap() – ikki marta bosish ishorasi ("ikki marta bosish") sodir bo'lishini kuzatib boradi;

onDoubleTapEvent() – ikki marta bosish imo-ishorasi paytida sodir bo'ladigan hodisani tinglaydi, jumladan tegish, harakatlantirish, barmoqni ko'tarish.

onSingleTapConfirmed() – bir marta bosish imo-ishorasining ko'rinishini kuzatib boradi (klik).

Ilova kodida unda barcha qo'llab-quvvatlanadigan imo-ishoralar tan olinadi, paydo bo'lgan va tan olingan imo-ishora haqidagi ma'lumotlar ma'lumot maydonida ko'rsatiladi (*TextView*). Amaliyot sifatida, ushbu ilovani qayta ishlab chiqarish va tizim ma'lum bir imo-ishorani qanday tan olishini tekshirish taklif etiladi. Asosiy imo-ishoralar qanday bajarilishini tushunish uchun juda foydali. Dasturda dasturchining tanlovi bo'yicha faqat ba'zi qo'llab-quvvatlanadigan imo-ishoralarni tanib olishni namoyish qilamiz. Biz surish ishorasining tan olinishini ko'rib chiqamiz (*fling*). Ilovada tanib olingan imo-ishora haqidagi ma'lumotlarni ko'rsatish uchun bitta *Activity*, bitta ma'lumot maydoni kiritiladi. Ilova quyidagicha ishlaydi: foydalanuvchi qo'llab-quvvatlanadigan tegish imo-ishoralardan birini

amalga oshiradi, ma'lumot maydonida tan olingan imo-ishora haqidagi ma'lumotlar ko'rsatiladi.

1. Oddiy dastur yaratamiz va ma'lumotni ko'rsatish uchun formaga *TextView* qo'shamiz.

2. Ilova mantig'ini o'ylab topish. *Activity*-ga mos keladigan *java* sinfiga quyidagi qo'shimchalarni kiritamiz.

Bizga *GestureDetectorCompat* sinfining namunasi kerak bo'ladi, shuning uchun quyidagi o'zgaruvchini *Activity* sinfining maydoni sifatida e'lon qilamiz:

```

GestureDetectorCompat mDetector;
Activity sinfining onCreate() usulida GestureDetectorCompat
sinfining namunasi yaratib va uni mDetector o'zgaruvchisiga tayinlanadi:
mDetector=new GestureDetectorCompat(this, new
MyGestListener());

```

konstruktorda sensorli hodisalarini kuzatish uchun mas'ul bo'lgan argument *GestureDetector.SimpleOnGestureListener* sinfidan meros bo'lgan ichki sinf *MyGestListener()* sinfining nusxasidir.

GestureDetector.SimpleOnGestureListener sinfi
GestureDetector.OnGestureListener va

GestureDetector.OnDoubleTapListener sinf interfeyslarini amalga oshiradi, bu sinfdagi interfeyslarda e'lon qilingan barcha usullar bo'sh amalga oshirishga ega va qiymatni qaytaradiganlar noto'g'ri qaytaradi. Shuning uchun, ba'zi bir voqea yoki hodisalarning ba'zi bir kichik to'plamini tanib olish uchun voris sinfida mos keladigan usullarni amalga oshirishni yozish kifoya qiladi.

Quyidagi kod ishorasini taniydigan dasturni ko'rsatadi, ya'ni *onFling()* usuli amalga oshiriladi, paydo bo'lgan va tan olingan imo-ishora haqidagi ma'lumotlar ma'lumot maydonida (*TextView*) ko'rsatiladi. Tinglovchi *MyGestListener()* sinfining namunasi bo'lib, u *GestureDetector.SimpleOnGestureListener* sinfining merosxo'ri hisoblanadi.

```

package com.example.lagestall;
import android.os.Bundle;
import android.app.Activity;
import android.support.v4.view.GestureDetectorCompat;
import android.view.*;
import android.widget.*;
public class MainActivity extends Activity
    implements GestureDetector.OnGestureListener,

```

```

GestureDetector.OnDoubleTapListener {
    TextView tvOutput;
    GestureDetectorCompat mDetector;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.Activity_main);
        tvOutput = (TextView)findViewById(R.id.textView1);
        mDetector = new GestureDetectorCompat(this, this);
        mDetector.setOnDoubleTapListener(this);
        public boolean onTouchEvent(MotionEvent event) {
            this.mDetector.onTouchEvent(event);
            // Superklassni amalga oshirish uchun chaqirish
            return super.onTouchEvent(event);
        }
        @Override
        public boolean onKeyDown(MotionEvent event) {
            tvOutput.setText("onDown: " + event.toString());
            return false;
        }
        @Override
        public boolean onFling(MotionEvent event1, MotionEvent event2,
            float velocityX, float velocityY) {
            tvOutput.setText("onFling: " + event1.toString()+event2.toString());
            return true;
        }
        @Override
        public void onLongPress(MotionEvent event) {
            tvOutput.setText("onLongPress: " + event.toString());
        }
        @Override
        public boolean onScroll(MotionEvent e1, MotionEvent e2, float
            distanceX, float distanceY) {
            tvOutput.setText("onScroll: " + e1.toString()+e2.toString());
            return true;
        }
        @Override
        public void onShowPress(MotionEvent event) {
            tvOutput.setText("onShowPress: " + event.toString());
        }
        @Override
        public boolean onSingleTapUp(MotionEvent event) {
            tvOutput.setText("onSingleTapUp: " + event.toString());
        }
    }
}

```

```

return true;
}
@Override
public boolean onDoubleTap(MotionEvent event) {
    tvOutput.setText("onDoubleTap: " + event.toString());
    return true;
}
@Override
public boolean onDoubleTapEvent(MotionEvent event) {
    tvOutput.setText("onDoubleTapEvent: " + event.toString());
    return true;
}
@Override
public boolean onSingleTapConfirmed(MotionEvent event) {
    tvOutput.setText("onSingleTapConfirmed: " + event.toString());
    return true;
}
}
Interfeys ilovalari yordamida qo'llab-quvvatlanadigan imo-ishoralarni
tanib olish dasturi kodi quyida keltirilgan:
package com.example.labgestsubset;
import android.os.Bundle;
import android.app.Activity;
import android.support.v4.view.*;
import android.view.*;
import android.widget.*;
public class SubsetGestActivity extends Activity {
    private GestureDetectorCompat mDetector;
    private TextView tvOut;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.Activity_subset_gest);
        mDetector = new GestureDetectorCompat(this, new
MyGestListener());
        tvOut = (TextView)findViewById(R.id.textView1);
    }
    @Override
    public boolean onTouchEvent(MotionEvent event) {
        this.mDetector.onTouchEvent(event);
        return super.onTouchEvent(event);
    }
}
class MyGestListener extends
GestureDetector.SimpleOnGestureListener {
    @Override
    public boolean onFling(MotionEvent event1, MotionEvent event2,

```

```
float velocityX, float velocityY) {
    tvOut.setText("onFling: " + event1.toString()+event2.toString());
    return true; } }
}
```

Yuqorida *GestureDetector.SimpleOnGestureListener* sinfi yordamida imo-ishoralarni aniqlash dasturi keltirilgan.

5.2. Sensorlar turi va ular bilan ishlash

Fizik va holatlar sensorlari mobil ilovalarni yaxshilash yo'llarini taqdim etadi. Zamonaviy mobil qurilmalarda elektron kompaslar, muvozanat, yorqinlik va yaqinlik datchiklari mavjud bo'lib, ular qurilma bilan o'zaro aloqada bo'lish uchun butunlay yangi imkoniyatlarni ochib beradi, masalan, kengaytirilgan reallik va fazodagi harakatlarga asoslangan kiritish.

Android-dagi sensorlar bir nechta toifalarga bo'linadi: harakat, joylashuv va atrof-muhit.

- *Harakat sensorlari* – bu sensorlar uch o'qda tezlanish va aylanish kuchlarini o'lchaydi. Ushbu turkumga akselerometr, tortishish sensorlari, giroskoplar va aylanish vektor sensorlari kiradi.

- *Atrof-muhit sensorlari* – bu sensorlar atrof-muhit havosining harorati va bosimi, yorug'lik intensivligi va namlik kabi turli xil atrof-muhit parametrlarini o'lchaydi. Bu toifaga barometrlar, fotometrlar va termometrlar kiradi.

- *Joylashuv sensorlari* – bu sensorlar qurilmaning jismoniy holatini o'lchaydi. Ushbu turkumga orientatsiya sensorlari va magnetometrlar kiradi.

Quyida mashhur sensorlarning ba'zi bir mavjud turlari keltirilgan:

- akselerometr (*TYPE_ACCELEROMETER*)
- giroskop (*TYPE_GYROSCOPE*)
- yorug'lik sensori (*TYPE_LIGHT*)
- masofa sensori (*TYPE_PROXIMITY*)
- magnet maydon sensori (*TYPE_MAGNETIC_FIELD*)
- barometr (*TYPE_PRESSURE*)

- atrof-muhit harorati sensori (*TYPE_AMBIENT_TEMPERATURE*)

- nisbiy namlik o'lchagich (*TYPE_RELATIVE_HUMIDITY*)

Har bir mobil qurilmada o'ziga xos sensorlar to'plami bo'lishi mumkin. Ko'pchilikda akselerometr va giroskop mavjud. Datchiklar bilan ishlash, xususan, orientatsiya va harorat sensorlari uchun sinflar mavjud. Datchiklar bilan ishlashda quyidagilarni yodda tutish kerak:

- ko'rsatmalar juda notekis bo'lishi mumkin. Ilova sezgir bo'lib qolishi uchun o'qishlarning o'rtacha qiymatidan foydalanish kerak;

- ma'lumotlar notekis keladi, ya'ni har xil o'lchamda;
- foydalanuvchining kelajakdagi harakatlarini bashorat qilishga harakat qiladi. Masalan, agar qurilmaning aylanishni boshlanishi haqida ma'lumot mavjud bo'lsa, keyingi harakatni bashorat qilish va unga tayyorgarlik ko'rish mumkin.

Emulyatorlarda sensorlar bilan ishlashni sinab ko'rish deyarli mumkin emas, shuning uchun haqiqiy qurilmalardan foydalanish lozim. Emulyatorlarning so'nggi versiyalarida sensorlar imkoniyatlari ro'yxati kengaytirilgan.

Android sensorlar bilan ishlash uchun sinflar va interfeyslar to'plamini taqdim etadi. Ushbu sinflar va interfeyslar *android.hardware* paketining bir qismi bo'lib, quyidagi vazifalarni bajarishga imkon beradi:

- qurilmada qaysi sensorlar mavjudligini aniqlash;
- maksimal qiymat, ishlab chiqaruvchi, energiya talablari va ruxsatlar kabi individual sensor imkoniyatlarini aniqlash;
- datchiklardan ma'lumotlarni yig'ish va ma'lumotlar yig'ishning minimal chastotasini aniqlash;
- hodisa tinglovchilarini sensorlardan ulash va ajratish, hodisalar sensor qiymatlarini o'zgartirishdan iborat.

Android sensorlar bilan ishlash uchun quyidagi sinflar va interfeyslarni taqdim etadi:

SensorManager – bu sinf sensor bilan bog'langan xizmatni yaratish uchun ishlatilishi mumkin. Shuningdek, u sensorlarga kirish va ro'yxatga olish, hodisa tinglovchilarini sensorlardan ulash va ajratish, ma'lumot to'plash uchun turli usullarni taqdim etadi. Ushbu sinf sensorning aniqligini, ma'lumotlarni yig'ish chastotasini va sensorni sozlash uchun ishlatiladigan o'zgarmaslarni o'z ichiga oladi.

Sensor – bu sinf sensorni ishga tushirish uchun ishlatiladi, sensor xususiyatlarini aniqlash usullarini taqdim etadi.

SensorEvent – tizim sensor hodisasiga mos keladigan obyektini yaratish va quyidagi ma'lumotlarni taqdim etish uchun ushbu sinfdan foydalanadi: sensor ma'lumotlari; hodisani yaratgan sensor turi, ma'lumotlarning aniqligi va voqea sodir bo'lgan vaqt.

SensorEventListener – ushbu interfeys sensor qiymati o'zgaranda yoki sensorning aniqligi o'zgaranda bildirishnomalarni (sensor hodisalari) oladigan ikkita usulni amalga oshirish uchun ishlatilishi mumkin.

SensorManager sinfi datchiklar bilan ishlash uchun mas'ul bo'lib, u Android sensor tizimining turli jihatlarini tavsiflovchi bir nechta o'zgarmlarni o'z ichiga oladi, jumladan:

- *sensor turi* - orientatsiya, akselerometr, yorug'lik, magnit maydon, yaqinlik, harorat va boshqalar;
- *o'lchov chastotasi* - o'yinlar uchun maksimal, foydalanuvchi interfeysi uchun normal. Ilova ma'lum bir namuna tezligini so'raganda, bu faqat sensorli quyi tizim nuqtai nazaridan tavsiya etiladi. Belgilangan chastotada o'lchovlar amalga oshirilishiga kafolat yo'q;
- *aniqlik* - yuqori, past, o'rt, ishonchsiz ma'lumotlar.

Sensor turlari

• *TYPE_ACCELEROMETER* - X, Y, Z o'qlari bo'ylab fazoda tezlanishni o'lchaydi;

• *TYPE_AMBIENT_TEMPERATURE* - tselsiy bo'yicha haroratni (API 14) o'lchash uchun yangi sensor, eskirgan *TYPE_TEMPERATURE* o'miga ishlatiladi;

• *TYPE_GRAVITY* - uch o'qli koordinata tortishish sensori. Bu virtual sensor va akselerometr tomonidan qaytariladigan ko'rsatkichlar uchun past o'tkazuvchan filtrdir;

• *TYPE_GYROSCOPE* - uch o'qli koordinata giroskop, qurilmaning fazodagi joriy holatini uchta o'q bo'ylab darajalarda qaytaradi. Boshqa ma'lumotlarga ko'ra, u uch koordinata o'qi bo'ylab qurilmaning aylanish tezligini radianda soniyasiga qaytaradi;

• *TYPE_LIGHT* - yorug'lik miqdorini o'lchaydi. Atrof-muhit yorug'ligini lyuksda tasvirlaydigan yorug'lik sensori. Ushbu turdagi sensor odatda ekran yorqinligini dinamik ravishda o'zgartirish uchun ishlatiladi;

• *TYPE_LINEAR_ACCELERATION* - tortishish kuchidan qat'iy nazar tezlanish raqamlarini qaytaradigan uch o'qli koordinata chiziqli tezlanish sensori. Bu akselerometr ko'rsatkichlaridan foydalanadigan virtual sensor;

• *TYPE_MAGNETIC_FIELD* - uchta o'qli koordinata bo'ylab mikrotaslardagi magnit maydonning joriy qiymatlarini aniqlaydigan magnit maydon sensori;

• *TYPE_ORIENTATION* - orientatsiya sensori. Qurilmaning burulishlarini, egilishlarini va aylanishini o'lchaydi.;

• *TYPE_PRESSURE* - atmosfera bosimi sensori (barometr) joriy bosimni millibarda qaytaradi. Atmosfera bosimini ikki nuqtada solishtirib, dengiz sathidan balandlikni aniqlashi mumkin. Barometrlar ob-havoni bashorat qilish uchun ham ishlatilishi mumkin;

• *TYPE_PROXIMITY* - qurilma va nishon o'rtasidagi masofani santimetrdagi bildiruvchi yaqinlik sensori. Obyekt qanday tanlanganligi va qaysi masofalar qo'llab-quvvatlanishi ushbu sensorning apparat ta'minotiga bog'liq, ikkita qiymatni qaytarish mumkin - *Yaqin* va *Uzoq*. Uning odatiy ishlatilishi ekran yorqinligini avtomatik ravishda sozlash yoki ovozli buyruqni bajarish uchun qurilma va foydalanuvchi qulog'i orasidagi masofani aniqlashdir;

• *TYPE_RELATIVE_HUMIDITY* - nisbiy namlik sensori, foiz sifatida qiymat qaytaradi(API 14);

• *TYPE_ROTATION_VECTOR* - qurilmaning fazodagi o'rmni o'qqa nisbatan burchak shaklida qaytaradi. Akselerometr dan o'qishni oladigan virtual sensor va giroskop. Magnit maydon sensori o'qishlarini ham ishlatishi mumkin;

• *TYPE_GEOMAGNETIC_ROTATION_VECTOR* - *TYPE_ROTATION_VECTOR*ga muqobil sensor hisoblanadi. Kamroq anqlik, lekin kamroq batareya iste'mol qiladi. Android 4.4 (API 19) dan boshlab taqdim etilgan;

• *TYPE_POSE_6DOF* - *TYPE_ROTATION_VECTOR*ga muqobil sensor hisoblanadi. Android 7.0 (API 24) dan boshlab taqdim etilgan;

• *TYPE_SIGNIFICANT_MOTION* - Android 4.3 (API 18) dan boshlab taqdim etilgan;

• *TYPE_MOTION_DETECT* - harakat detektor. Android 7.0 (API 24) dan boshlab taqdim etilgan;

• *TYPE_STATIONARY_DETECT* - Android 7.0 (API 24) dan boshlab taqdim etilgan;

• *TYPE_STEP_COUNTER* - qadamlar sonini hisoblash sensori;

• *TYPE_STEP_DETECTOR* - qadamlar boshlanishini aniqlash;

• *TYPE_HEART_BEAT* - pulsni aniqlash sensori. Android 7.0 (API 24) dan boshlab taqdim etilgan;

• *TYPE_HEART_RATE* - yurak faolligini aniqlash sensori. Android 4.4 (API 20) dan boshlab taqdim etilgan;

• *TYPE_LOW_LATENCY_OFFBODY_DETECT* - Android 8.0 (API 26) dan boshlab taqdim etilgan.

Uskuna sensorlarga qo'shimcha ravishda, qurilmalar bir nechta apparat sensorlarining kombinatsiyasidan foydalangan holda soddalashtirilgan, aniqlangan yoki birlashtirilgan o'qishni ta'minlaydigan virtual sensorlardan foydalanadi. Ba'zi hollarda bu usul qulayroqdir.

Sensorlarga kirish uchun *getService()* usulini chaqirish kerak.
// Kotlin tilida

```

private lateinit var sensorManager: SensorManager
sensorManager = getSystemService(SENSOR_SERVICE) as
SensorManager
// Java tilida
private SensorManager sensorManager;
sensorManager = (SensorManager)
getSystemService(SENSOR_SERVICE);

```

Qurilma bir xil turdagi sensorlarning bir nechta ilovalarini o'z ichiga olishi mumkin. Standart dasturni topish uchun *SensorManager* obyektidan *getDefaultSensor()* usulini chaqirib, unga sensor turini parametr sifatida yuqorida tavsiflangan doimiylardan biri ko'rinishida o'tkazish kerak.

Quyidagi kod parchasi standart giroskopni tavsiflovchi obyektini qaytaradi. Agar berilgan tip uchun standart sensor mavjud bo'lmasa, null qaytariladi.

```

// Kotlin tilida
sensorManager.getDefaultSensor(Sensor.TYPE_GYROSCOPE)
// tekshirish variantidan foydalangan ma'qul
if (sensorManager.getDefaultSensor(Sensor.TYPE_GYROSCOPE)
!= null) {
// Muvaffaqiyatli, giroskop mavjud
} else {
// Muvaffaqiyatsiz, giroskop aniqlanmadi
}
// Java tilida
Sensor defaultGyroscope = sensorManager.getDefaultSensor
(Sensor.TYPE_GYROSCOPE);

```

SensorManager sinfida *Sensor.TYPE_ALL* doimiysi va *getName()* usuli orqali qurilmadagi mavjud sensorlar ro'yxatini olish imkonini beruvchi *getSensorList()* usuli mavjud:

```

// Kotlin tilida
package ru.sensors
import android.hardware.Sensor
import android.hardware.SensorManager
import android.os.Bundle
import androidx.appcompat.app.AppCompatActivity
class MainActivity : AppCompatActivity() {
private lateinit var sensorManager: SensorManager
override fun onCreate(savedInstanceState: Bundle?) {
super.onCreate(savedInstanceState)

```

```

setContentView(R.layout.Activity_main)
sensorManager = getSystemService(SENSOR_SERVICE) as
SensorManager
val deviceSensors: List<Sensor> =
sensorManager.getSensorList(Sensor.TYPE_ALL)
println(deviceSensors.joinToString("\n"))
// Ma'lumotlar ushbu formatda chiqariladi
// {Sensor name= "MPL Accelerometer", vendor= "InvenSense",
version=1, type=1, maxRange=39.226593, resolution=0.0011901855,
power=0.5, minDelay=5000}
}

```

```

// Java tilida
package ru.sensors;
import java.util.ArrayList;
import java.util.List;
import android.app.ListActivity;
import android.content.Context;
import android.hardware.Sensor;
import android.hardware.SensorManager;
import android.os.Bundle;
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemClickListener;
public class SensorsActivity extends ListActivity {
@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
SensorManager sensorManager = (SensorManager)
getSystemService(Context.SENSOR_SERVICE);
List<Sensor> deviceSensors =
sensorManager.getSensorList(Sensor.TYPE_ALL);
List<String> listSensorType = new ArrayList<>();
for (int i = 0; i < deviceSensors.size(); i++) {
listSensorType.add(deviceSensors.get(i).getName());
}
setListAdapter(new ArrayAdapter<>(this,
android.R.layout.simple_list_item_1, listSensorType));
getListView().setTextFilterEnabled(true);
}
}

```

Har bir qurilma o'ziga xos sensorlar to'plamiga ega bo'lganligi sababli, natijalar hamma uchun har xil bo'ladi.

Bundan tashqari, ma'lum bir turdagi mavjud sensorlar ro'yxatini olish mumkin. Quyidagi kod parchasi barcha mavjud bosim sensorlarini ifodalovchi *Sensor* obyektlarini qaytaradi:

```
// Kotlin tilida
val pressureSensors: List<Sensor> =
    sensorManager.getSensorList(Sensor.TYPE_PRESSURE)
// Java tilida
List<Sensor> pressureSensors =
    sensorManager.getSensorList(Sensor.TYPE_PRESSURE);
Sensor ishlab chiqaruvchisi va uning versiyasini tekshiradigan
murakkab shartni yaratish mumkin. Agar kerakli sensor mavjud bo'lmasa,
muqobil variantni tanlash mumkin.
// Kotlin tilida
if (sensorManager.getDefaultSensor(Sensor.TYPE_GRAVITY) !=
    null) {
    val gravitySensors: List<Sensor> =
        sensorManager.getSensorList(Sensor.TYPE_GRAVITY)
        sensor = gravitySensors.firstOrNull {
            it.vendor.contains("Qualcomm") && it.version == 1 }
            println(sensor?.vendor) }
            if (sensor == null) {
                sensor = if
(sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER) !=
    null)
{sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER)
} else {
    null } }
```

Shuningdek, *android.hardware.SensorListener* interfeysi ham ishlatiladi. Interfeys real vaqtda sensor qiymatlarini kiritish uchun ishlatiladigan sinf bilan amalga oshiriladi. Ilova bir yoki bir nechta mavjud apparat sensorlarini kuzatish uchun ushbu interfeysni amalga oshiradi.

Interfeys ikkita talab qilinadigan usulni o'z ichiga oladi:

- *Sensor* qiymati o'zgarganda *onSensorChanged(int sensor, float values[])* usuli chaqiriladi. Ushbu usul faqat ushbu ilova tomonidan boshqariladigan sensorlar uchun chaqiriladi. Usul argumentlari sensorning qiymati o'zgarishini ko'rsatadigan butun sonni va sensorning haqiqiy qiymatini ifodalovchi suzuvchi nuqta qiymatlari qatorini o'z ichiga oladi. Ba'zi sensorlar faqat bitta ma'lumot qiymatini ta'minlaydi, boshqalari esa

uchta suzuvchi nuqta qiymatini beradi. Orientatsiya sensorlari va akselerometr har biri uchta ma'lumot qiymatini beradi.

- *Sensor* ko'rsatkichlarining aniqligi o'zgarganda *onAccuracyChanged(int sensor, int accuracy)* usuli chaqiriladi.

Argumentlar ikkita butun son: biri sensorni belgilaydi, ikkinchisi esa ushbu sensor uchun yangi aniqlik qiymatiga mos keladi.

Har safar qiymatlar o'zgarganda sensor xizmati *onSensorChanged()* ni chaqiradi. Barcha sensorlar suzuvchi nuqta qiymatlari qatorini qaytaradi. Har safar qiymatlar o'zgarganda sensor xizmati *onSensorChanged()* ni chaqiradi. Barcha sensorlar suzuvchi nuqta qiymatlari qatorini qaytaradi.

Massivning o'lchami sensorning xususiyatlariga bog'liq. *TYPE_TEMPERATURE* sensori bitta qiymatni qaytaradi - harorat Selsiy bo'yicha, boshqalari bir nechta qiymatlarni qaytarishi mumkin. Masalan, faqat magnit azimut haqida ma'lumot olish uchun *TYPE_ORIENTATION* sensori tomonidan qaytarilgan birinchi raqamdan foydalanish mumkin.

Sensorning aniqlik darajasini ifodalash usullarida ishlatiladigan aniqlik parametri doimiylardan birini ishlatadi.

- *SensorManager.SENSOR_STATUS_ACCURACY_LOW*. Sensor tomonidan taqdim etilgan ma'lumotlarning aniqligi past ekanligini va kalibrash kerakligini ko'rsatadi.

- *SensorManager.SENSOR_STATUS_ACCURACY_MEDIUM*. Sensorning o'rtacha aniqlik darajasini va kalibrash natijani yaxshilashi mumkinligini ko'rsatadi.

- *SensorManager.SENSOR_STATUS_ACCURACY_HIGH*. Sensor ko'rsatkichlari imkon qadar aniq.

- *SensorManager.SENSOR_STATUS_UNRELIABLE*. Sensor tomonidan taqdim etilgan ma'lumotlar yaroqsiz. Bu shuni anglatadiki, sensorning kalibrash kerak, aks holda natijalarni o'qish mumkin emas.

Sensorlar tomonidan yaratilgan hodisalarni qabul qilish uchun *SensorEventListener* interfeysini amalga oshirishni *SensorManager* bilan ro'yxatdan o'tkaziladi. Kuzatilishi kerak bo'lgan *Sensor* obyekti va yangilanishlarni qabul qilish chastotasini belgilash kerak.

Obyektni olinganidan so'ng, yangilangan ma'lumotlarni olishni boshlash uchun *onResume()* usulida *registerListener()* usuli chaqiriladi va ma'lumotlarni qabul qilishni to'xtatish uchun *onPause()* usulida *unregisteredListener()* usuli chaqiriladi. Bunday holda, sensorlar faqat *activity* ekranida ko'rinadigan bo'lsa ishlatiladi.

Quyidagi misol *SensorEventListener*-ni standart yangilanish tezligi bilan standart yaqinlik sensori uchun qanday ro'yxatdan o'tkazishni ko'rsatadi:

```

Sensor
sensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY);
sensorManager.registerListener(mySensorEventListener,
sensor, SensorManager.SENSOR_DELAY_NORMAL);

```

SensorManager sinfi mos yangilanish chastotasini tanlash uchun quyidagi o'zgarishlarni o'z ichiga oladi (kamayish tartibida):

- *SensorManager.SENSOR_DELAY_FASTEST* – sensor ko'rsatkichlari uchun mumkin bo'lgan eng yuqori yangilanish tezligi;
- *SensorManager.SENSOR_DELAY_GAME* – o'yinlarni boshqarish uchun ishlatiladigan chastota;
- *SensorManager.SENSOR_DELAY_NORMAL* – standart yangilanish chastotasi;
- *SensorManager.SENSOR_DELAY_UI* – foydalanuvchi interfeysini yangilash chastotasi.

Tanlagan chastotaga rioya qilish shart emas. *SensorManager* natijalarni ko'rsatganidan tezroq yoki sekinroq qaytarishi mumkin (garchi bu odatda tezroq bo'lsa ham). Ilovada sensorlardan foydalanganda resurslar sarfini minimallashtirish uchun eng past chastotani tanlashga harakat qilish kerak.

5.3. Mobil ilovalarda mobil datchiklardan foydalanish

Android qurilmalar uchun standart sensor konfiguratsiyasini belgilamaydi, ya'ni qurilma ishlab chiqaruvchilari o'zlarining Android qurilmalariga istalgan sensor konfiguratsiyasini kiritishlari mumkin. Natijada, qurilmalar keng konfiguratsiyadagi bir nechta sensorlarni o'z ichiga olishi mumkin. Agar ilovada muayyan turdagi sensordan foydalanilsa, ilova muvaffaqiyatli ishlashi uchun sensor qurilmada mavjudligiga ishonch hosil qilish kerak.

Qurilmada ma'lum sensor mavjudligini tekshirishning ikki yo'li mavjud:

- ishlash vaqtida datchiklarni aniqlash va vaziyatga qarab dastur funksiyalarini yoqish yoki o'chirish;
- muayyan sensor konfiguratsiyasiga ega qurilmalarni nishonga olish uchun Google Play filtrlaridan foydalanish.

Agar ilovada ma'lum turdagi sensordan foydalansa, ish vaqtida sensorni aniqlash uchun sensor ramkasidan foydalanish mumkin, so'ngra ilovaning xususiyatlarini mos ravishda o'chirib qo'yish yoki yoqish mumkin. Masalan, navigatsiya ilovasi harorat, barometrik bosim, joylashuv va kompas sarlavhasini ko'rsatish uchun harorat sensori, bosim sensori,

GPS sensori va geomagnit maydon sensoridan foydalanishi mumkin. Agar qurilmada bosim sensori bo'lmasa, ish vaqtida bosim sensori yo'qligini aniqlash uchun sensor ramkasidan foydalanish mumkin va keyin ilovada foydalanuvchi interfeysining bosimni ko'rsatadigan qismini o'chirib qo'yish mumkin. Masalan, quyidagi kod qurilmada bosim sensori mavjudligini tekshiradi:

```

private SensorManager sensorManager;
SensorManager sensorManager;
getSystemService(Context.SENSOR_SERVICE);
if (sensorManager.getDefaultSensor(Sensor.TYPE_PRESSURE) !=
null){
} else {
}

```

Ilovani Google Play-da nashr qilish uchun *manifest* faylidagi `<uses-feature>` elementidan ilova uchun mos sensor konfiguratsiyasiga ega bo'lmagan qurilmalardan ilovani filtrlash uchun foydalanish mumkin. `<uses-feature>` elementi ma'lum sensorlar mavjudligiga qarab ilovalarni filtrlash imkonini beruvchi bir nechta apparat deskriptorlariga ega. Quyidagi sensorlarni belgilash mumkin: akselerometr, barometr, kompas (geomagnit maydon), giroskop, yorug'lik va yaqinlik. Quyida akselerometrga ega bo'lmagan ilovalarni filtrlaydigan *manifest* yozuviga misol keltirilgan:

```

<uses-feature
android:name="android.hardware.sensor.accelerometer"
android:required="true" />

```

Agar ushbu element va deskriptorni ilova *manifest*ga qo'shsa, foydalanuvchilar ilovani faqat qurilmada akselerometr bo'lsagina Google Play-da ko'radi.

Kodda faqat `android:required="true"` identifikatorini o'rnatish kerak, agar ilova to'liq ma'lum bir sensorga tayansa. Agar ilovada ba'zi funksiyalar uchun sensordan foydalansa, lekin sensorsiz ishlayotgan bo'lsa, `<uses-feature>` elementida sensorni belgilash kerak, lekin deskriptorni `android:required="false"` qilib o'rnatish kerak. Bu qurilmalarda ushbu sensor bo'lmasa ham ilovani o'rnatishiga yordam beradi.

Quyida Android SDK tarkibidagi ba'zi apparat xususiyatlari 3-jadvalda tasvirlangan.

Funksiya	Tavsifi
1	2
android.hardware.Camera	Ilovalarga suratga olish, oldindan ko'rish ekranini yozib olish yoki sozlamalarni o'zgartirish uchun videokamera bilan o'zaro ishlash imkonini beruvchi sinf
android.hardware.SensorManager	Android platformasining ichki sensorlariga kirishni ta'minlovchi sinf. Har bir Android qurilmasi <i>SensorManager</i> -dagi barcha sensorlarni qo'llab-quvvatlamaydi
android.hardware.SensorListener	Interfeys real vaqtda sensor qiymatlarini kiritish uchun ishlatiladigan sinf bilan amalga oshiriladi. Ilova bir yoki bir nechta mavjud apparat sensorlarini kuzatish uchun ushbu interfeysni amalga oshiradi
1	2
android.media.MediaRecorder	Muayyan joyda (masalan, bolalar xonasida) tovushlarni yozish uchun ishlatilishi mumkin bo'lgan ommaviy axborot vositalarini yozib olish uchun ishlatiladigan sinf. Shuningdek, binolarga kirishni nazorat qilish va xavfsizlik maqsadlarida audio qismlarni tahlil qilish mumkin. Misol uchun, kalit uchun konserjga borish o'rniga, odatdagidek kelish vaqtida eshikni o'z ovozi bilan ochish mumkin.

android.FaceDetector	Xotirada saqlangan fotosuratdan odamning yuzini tanib olish imkonini beruvchi sinf. Hech narsa shaxsni yuzdan ko'ra yaxshiroq isbotlay olmaydi. Agar undan qurilmani qullash uchun foydalansa, endi parollarni eslab qolish shart emas – mobil telefonning biometrik imkoniyatlari yetarli.
android.os.*	Operatsion muhit bilan o'zaro ishlash uchun bir nechta foydali sinflarni o'z ichiga olgan paket, jumladan quvvatni boshqarish, fayllarni qidirish, ishlov beruvchi va xabar almashish uchun sinflar. Ko'pgina boshqa portativ qurilmalar singari, Android telefonlari ham juda ko'p elektr energiyasini iste'mol qilishi mumkin. To'g'ri hodisani nazorat qilish uchun qurilmaning kerakli vaqtda "uyg'onishini" ta'minlash alohida e'tiborga loyiq bo'lgan muhim loyihalash jihati hisoblanadi.
1	2
java.util.Sana java.util.Taymer java.util.TimerTask	Haqiqiy dunyo voqealarini o'lchashda sana va vaqt ko'pincha muhim ahamiyatga ega. Masalan, <i>java.util.Date</i> sinfi voqea sodir bo'lganda yoki ma'lum bir holat sodir bo'lganda vaqt tamg'asini olish imkonini beradi. <i>java.util.Timer</i> va <i>java.util.TimerTask</i> mos ravishda davriy rejalashtirilgan harakatni yoki ma'lum bir vaqtning o'zida bir martalik harakatni bajarish uchun ishlatilishi mumkin.

Android.hardware.SensorManager Android sensor tizimining turli jihatlarini tavsiflovchi bir nechta o'zgarishlarni o'z ichiga oladi, jumladan *Sensor* ilovalar markazi va *SensorListener* interfeysidir. U ikkita talab qilinadigan usulni o'z ichiga oladi:

- sensor qiymati har doim o'zgarganda *onSensorChanged(int sensor,float values[])* usuli chaqiriladi. Ushbu usul faqat ushbu ilova tomonidan boshqariladigan sensorlar uchun chaqiriladi. Usul argumentlari sensorning qiymati o'zgarishini ko'rsatadigan butun sonni va sensorning haqiqiy qiymatini ifodalovchi suzuvchi nuqta qiymatlari qatorini o'z ichiga oladi. Ba'zi sensorlar faqat bitta ma'lumot qiymatini ta'minlaydi, boshqalari esa uchta suzuvchi nuqta qiymatini beradi. Orientatsiya sensorlari va akselerometr har biri uchta ma'lumot qiymatini beradi;

- sensor ko'rsatkichlarining aniqligi o'zgarganda *onAccuracyChanged(int sensor,int accuracy)* usuli chaqiriladi. Argumentlar ikkita butun son: biri sensorni belgilaydi, ikkinchisi esa ushbu sensor uchun yangi aniqlik qiymatiga mos keladi.

Sensor bilan ishlash uchun dastur bir yoki bir nechta sensorlar bilan bog'liq harakatlarni qabul qilish uchun ro'yxatdan o'tishi kerak. Ro'yxatdan o'tish *SensorManager* sinfining tinglovchisi *registerListener()* ro'yxatga olish usuli yordamida amalga oshiriladi.

Barcha Android qurilmalari SDK ro'yxatidagi ma'lum bir sensorni qo'llab-quvvatlamaydi. Agar ma'lum bir qurilmada ma'lum bir sensor bo'lmasa, ilova ushbu vaziyatni ehtiyotkorlik bilan hal qilishi kerak.

Sensor bilan ishlashga misol

Ushbu ilova oddiygina orientatsiya va akselerometr sensorlaridagi o'zgarishlarni kuzatib boradi. Sensor qiymatlari o'zgarganda, ular *TextView* vidjetida ekranda ko'rsatiladi.

Ilova *AndroidStudio* muhitida *Android Developer Tools* plagini bilan yaratilgan va quyida ushbu ilova uchun kod ko'rsatilgan.

IBMEyes.java

```
package com.eyes;
import android.app.Activity;
import android.os.Bundle;
import android.util.Log;
import android.widget.TextView;
import android.hardware.SensorManager;
import android.hardware.SensorListener;
public class IBMEyes extends Activity implements
```

```
SensorListener {
```

```
final String tag = "IBMEyes";
SensorManager sm = null;
TextView xViewA = null;
TextView yViewA = null;
TextView zViewA = null;
TextView xViewO = null;
TextView yViewO = null;
TextView zViewO = null;
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    sm = (SensorManager)
    getSystemService(SENSOR_SERVICE);
    setContentView(R.layout.main);
    xViewA = (TextView) findViewById(R.id.xbox);
    yViewA = (TextView) findViewById(R.id.ybox);
    zViewA = (TextView) findViewById(R.id.zbox);
    xViewO = (TextView) findViewById(R.id.xboxo);
    yViewO = (TextView) findViewById(R.id.yboxo);
    zViewO = (TextView) findViewById(R.id.zboxo);
}
public void onSensorChanged(int sensor, float[] values) {
    synchronized (this) {
        Log.d(tag, "onSensorChanged:" + sensor + ", x: " +
        values[0] + ", y: " + values[1] + ", z: " + values[2]);
        if (sensor ==
        SensorManager.SENSOR_ORIENTATION) {
            xViewO.setText("Orientation X: " + values[0]);
            yViewO.setText("Orientation Y: " + values[1]);
            zViewO.setText("Orientation Z: " + values[2]); }
        if (sensor ==
        SensorManager.SENSOR_ACCELEROMETER) {
            xViewA.setText("Accel X: " + values[0]);
            yViewA.setText("Accel Y: " + values[1]);
            zViewA.setText("Accel Z: " + values[2]); } } }
    public void onAccuracyChanged(int sensor, int accuracy) {
        Log.d(tag, "onAccuracyChanged: " + sensor + ", accuracy: " +
        accuracy); }
    @Override
```



```

mrec.start();
protected void stopRecording() {
    mrec.stop();
    mrec.release();
    processaudiofile(audiofile.getAbsolutePath()); }
protected void processaudiofile() {
    ContentValues values = new ContentValues(3);
    long current = System.currentTimeMillis();
    values.put(MediaStore.Audio.Media.TITLE, "audio" +
audiofile.getName());
    values.put(MediaStore.Audio.Media.DATE_ADDED, (int) (current
/1000));
    values.put(MediaStore.Audio.Media.MIME_TYPE, "audio/3gpp");
    values.put(MediaStore.Audio.Media.DATA,
audiofile.getAbsolutePath());
    ContentResolver contentResolver = getContentResolver();
    Uri baseUri =
MediaStore.Audio.Media.EXTERNAL_CONTENT_URI;
    Uri newUri = contentResolver.insert(baseUri, values);
    sendBroadcast(new
Intent(Intent.ACTION_MEDIA_SCANNER_SCAN_FILE, newUri));
}

```

StartRecording() usulida *MediaRecorder* nusxasi yaratiladi va ishga tushiriladi.

- Mikrofon (MIC) ma'lumot manbai sifatida tanlangan.
- Chiqarish formati 3GPP (*.3gp fayllar) ga o'rnatildi, bu mobil qurilmalar uchun mo'ljallangan media formati.
- Kodlovchi AMR_NB, 8 kHz audio formatiga o'rnatilgan. NB tor chastotali degan ma'noni anglatadi. Turli ma'lumotlar formatlari va mavjud kodlovchilar SDK hujjatlarida muhokama qilinadi.

Audio fayl ichki xotirada emas, balki alohida kartada saqlanadi. *External.getExternalStorageDirectory()* ushbu katalogda yaratilgan xotira kartasi va vaqtinchalik fayl nomlarini qaytaradi. Keyin bu fayl *setOutputFile()* usulini chaqirish orqali *MediaRecorder* misoli bilan bog'lanadi. Audio ma'lumotlar ushbu faylda saqlanadi.

Tayyorlash usulini chaqirish *MediaRecorder*-ni ishga tushirishni yakunlaydi. Yozish jarayoni boshlanishi kerak bo'lganda, *start()* usuli chaqiriladi. Xotira kartasidagi faylga yozish *stop()* usuli chaqirilguncha davom etadi. Ushbu usul *MediaRecorder* misoliga ajratilgan resurslarni

chiqaradi. Audio klip yozib olingandan so'ng, bir nechta holatlarni bajarish kerak:

- Qurilma kutubxonasiga audio yozuv qo'shish.
- Ovozni aniqlash uchun bir necha amallarni bajarish;
- Qayta ishlash uchun audio faylni tarmoq papkasiga avtomatik

yuklash.

Kod misolida, *processaudiofile()* usuli kutubxonaga audio yozuvni qo'shadi. *Intent* o'rnatilgan ilovaga yangi ma'lumotlar mavjudligi haqida xabar berish uchun ishlatiladi.

Yaratilgandan so'ng u audio yozmaydi hamda *AndroidManifest.xml* fayliga ruxsat kiritish kerak:

```

<uses-permission
android:name="android.permission.RECORD_AUDIO">
</uses-permission>

```

Nazorat savollari:

1. Mobil ilovalarda qanday imo-ishoralarni tanib olish mumkin?
2. Imo-ishorani aniqlashning qanday usullari qo'llaniladi?
3. Sensorli boshqaruv uchun qanday usullardan foydalaniladi?
4. Sensorlarni aniqlash uchun qanday usullardan foydalaniladi?
5. Datchiklarni aniqlash uchun qanday usullardan foydalaniladi?

6. IOS UCHUN SWIFT DASTURLASH TILIDA ILOVALARNI YARATISH

6.1. Swift tiliga kirish

iOS va MacOS uchun asosiy dasturlash tili Objective-C edi, ammo 2014-yil 2-iyundan boshlab yangi va qulay dasturlash tili – Swift joriy etildi. Objective-C bilan solishtirganda, Swift quyidagi xususiyatlarga ega:

- Swift obyektga yo'naltirilgan dasturlash tili;
- sodda va aniq sintaksisi;
- qattiq tiplashtirilganlik. Har bir o'zgaruvchi o'ziga xos tipga ega;
- xotirani avtomatik boshqarish.

Swift tili C va Objective-C dan foydalangan holda Cocoa API-lariga to'liq mos keladi. Swift tili rivojlanishda davom etmoqda. 2017-yil 19-sentabrda iOS va Mac OT uchun yangi funksiyalar qo'shilgan yangi 4.0 versiyasi chiqdi. Swift tili kompilyatsiya qilinadigan dasturlash tilidir, ya'ni kompilyator yordamida ishlab chiquvchi kodni boshqaruvchi dasturga kompilyatsiya qiladi, so'ngra dastur fayli AppStore-ga yuklanishi va tarqatilishi mumkin. Swift tili Mac OS 10.12 Yosemite yoki undan keyingi versiyasini talab qiladi. Mac OT-siz dasturni kompilyatsiya qilish deyarli mumkin emas. Biroq, Windows OT yoki Linux asosidagi operatsion tizimda ishlaydigan oddiy shaxsiy kompyuterda iOS va Mac OT uchun ilovalar yaratish mumkin emas. Shuningdek, Swift tilida har qanday operatsion tizimda kod yozish va uni maxsus xizmatlardan foydalangan holda pullik yoki bepul kompilyatsiya qilish mumkin.

To'g'ridan-to'g'ri ilova ishlab chiqish uchun Swift til vositalari, kod yozish uchun matn muharriri, dasturni tuzatish uchun iPhone va iPad simulyatorlari kerak bo'ladi. Mobil ilova ishlab chiqish uchun Apple Xcode dasturlash muhitini taqdim etadi.

XCodeda yozilgan dasturni testlash uchun simulyatorlar mavjud, ammo ba'zi hollarda haqiqiy smartfonlarda mobil ilovalarni testlash afzalroqdir. Shuningdek, ilovalarni yaratishdan oldin Apple veb saytida ro'yxatdan o'tish kerak (6.1-rasm). Buning uchun havola orqali shu saytga o'tish lozim: <https://developer.apple.com/register/>.

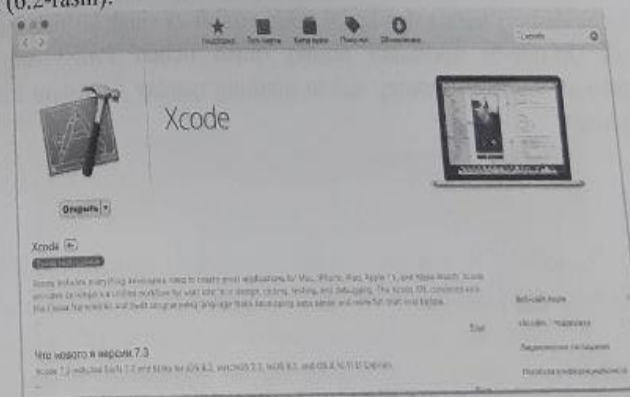
Ro'yxatdan o'tish uchun Apple ID va parol bilan saytga kirish kerak. Agar bunday identifikator bo'lmasa, Apple ID yaratish havolasini bosish orqali yangi akkaunt yaratish mumkin. Ro'yxatdan o'tgandan so'ng, sayt <https://developer.apple.com/resources/sahifasiga> yo'naltiradi (6.1-rasm).



6.1-rasm. Saytdan ro'yxatdan o'tish

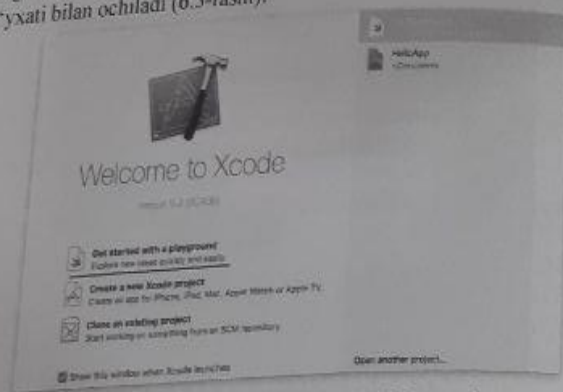
6.2. Xcode da Swift bilan ishlash

iOS tizimi mobil ilova ishlab chiqish uchun XCode deb nomlangan maxsus dasturlash muhitini talab qiladi. XCode iOS va Mac OSX uchun ilovalar yaratish uchun Swift va Objective-C tillaridan foydalanish imkonini beradi. XCode dasturlash muhiti bepul va uni App Store dan o'rnatish mumkin (6.2-rasm):



6.2-rasm. XCode ni o'rnatish

O'rnatilgandan so'ng, XCode-ni ishga tushirganda boshlang'ich ekran yangi loyiha yaratish variantlari, shuningdek, avval yaratilgan loyihalar ro'yxati bilan ochiladi (6.3-rasm):

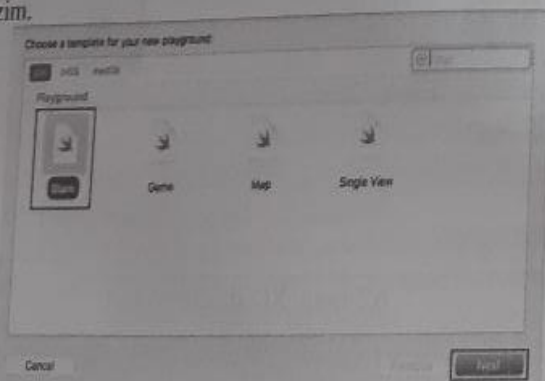


6.3-rasm. XCode-ni dastlabki oynasi

XCode dasturlash muhiti dastlabki oynada loyiha yaratishning quyidagi variantlarini tanlashni taklif etadi:

- Playground da dastur yaratish (*Get started with a playground*);
- yangi XCode loyihasini yaratish (*Create a new Xcode project*);
- mavjud loyihani clonlash (*Clone an existing project*).

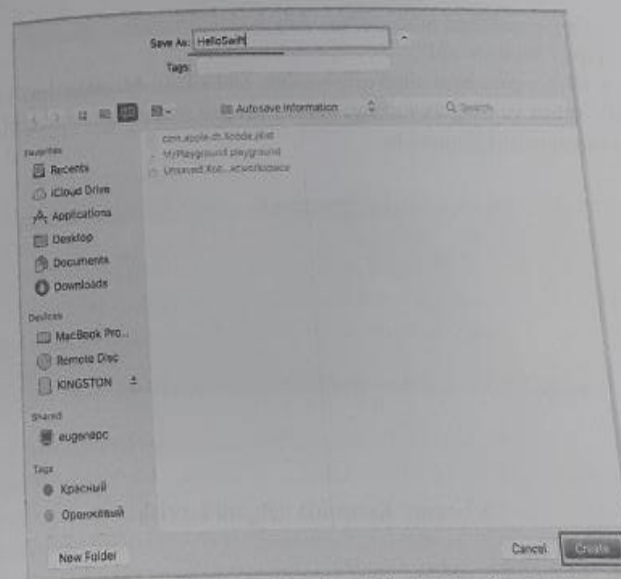
Xcode Playground deb nomlangan qulay hujjat turini o'z ichiga oladi, bu to'liq dastur yaratmasdan kodni tezda yozish va sinab ko'rish imkonini beradi. XCode-da kodlashni mashq qilish uchun Playgrounds dan foydalanish mumkin, shuning uchun ularning qanday ishlashini tushunib olish lozim.



6.4-rasm. Loyiha uchun kerakli shablonni tanlash oynasi

Yuqorida ko'rsatilgan rasmdagi darchadan birinchi bandni tanlagandan keyin Yangi Playground yaratish oynasi ochiladi. Bu oynadan iOS, tvOS va MacOS tizimlari uchun ilovalar yaratish turi tanlab olinadi. Bu oyna orqali Blank, Game, Map va SingleView ko'rinishidagi shablonlarni tanlash mumkin. Odatda iOS va boshqa tizimlar uchun ko'proq Blank shabloni tanlanadi (6.4-rasm).

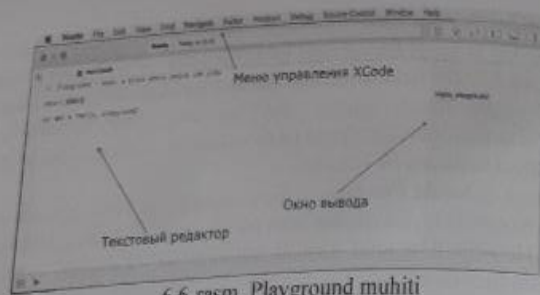
Keyingi qadamda Previous va Next tugmalaridan biri bosiladi. Next tugmasini bosganda loyiha qanday nom ostida saqlanishini belgilash kerak. Masalan, HelloSwift nomini kiritish mumkin va "Yaratish" tugmasini bosish lozim (6.5-rasm).



6.5-rasm. Loyiha uchun kerakli shablonni tanlash oynasi

Ushbu sozlamalardan so'ng Playground muhiti ochiladi. U Swift amallari, ifodalari va operatsiyalari bilan ishlash mumkin bo'lgan konsol chiqish oynasi bilan matn muharririni taqdim etadi (6.6-rasm).

XCode-ni boshqarish uchun menyu MacOS-ning yuqori panelida o'rnatiladi. Playground-ning ko'p qismini Swift tilidagi buyruqlar kiritilgan matn muharriri egallaydi va o'ng tomonda konsolning natija chiqish oynasi joylashgan bo'lib, u erda kiritilgan buyruqlar natijasini ko'rish mumkin.

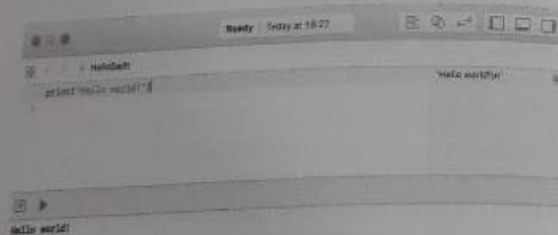


6.6-rasm. Playground muhiti

Playground matn muharririda oddiy kodni kiritamiz:

```
print("Hello world!")
```

`print()` – qatorlarni chop qilish uchun ishlatiladi. Misolda, bu “Hello world!” qatori va o’ng tomondagi konsol chiqish oynasida rasmdagi (6.7-rasm) xabarni ko’rish mumkin:



6.7-rasm. Konsolda natijani ko’rish

XCode da dastur tuzilishi

Swift dasturining tuzilishi buyruqlar to’plamidan iborat bo’lib, ularning har biri operator deb ataladi. Masalan, quyidagi ko’rsatma:

```
print("hello world!")
```

Qoida sifatida, har bir ko’rsatma bitta qatorga joylashtiriladi:

```
print("hello world!")
```

Swift tilining sintaksisi C, C++, C#, Java, C-ga o’xshash, ya’ni dasturlash tillarida bir qatorli ko’rsatmalar nuqta-vergul bilan tugamaydi. Ammo bir qatorga bir nechta ko’rsatmalar qo’yiladigan bo’lsa, ular nuqta-vergul bilan ajratilishi kerak:

`print("salom dunyo!"); print("Swiftga xush kelibsiz");`
Swift tuzilmali bloklarni ifodalash uchun figurali qavslardan foydalanadi. Misol uchun:

```
class Group { // sinf blokining boshlanishi
  func print() { // funksiya blokining boshlanishi
    print("guruh 320-18")
  } // funksiya blokining oxiri
} // sinf blokining oxiri
```

Izohlar

Swift-da yozilgan kodga sharhlarni belgilash mumkin. Ko’p qatorli sharhlar yaratish uchun `/* izoh matni */` tuzilmasidan foydalanish mumkin:

```
/*
```

Birinchi Swift dasturi

funksiya salom dunyo qatorini chop etadi

```
*/
```

```
print("salom dunyo!")
```

Bir qatorli sharhlar yaratish uchun ikki tomonlama chiziq ishlatiladi:

```
print("salom dunyo!") // funksiya salom dunyo qatorini chop etadi
```

```
print("Welcome to swift") // Swiftga xush kelibsiz qatorini chop etadi
```

Dasturni kompilyatsiya qilishda sharhlar e’tiborga olinmaydi va ishlab chiquvchiga yozilgan kodini eslatish uchun xizmat qiladi.

6.3. Swift dasturlash tilining asosiy konstruksiyalari

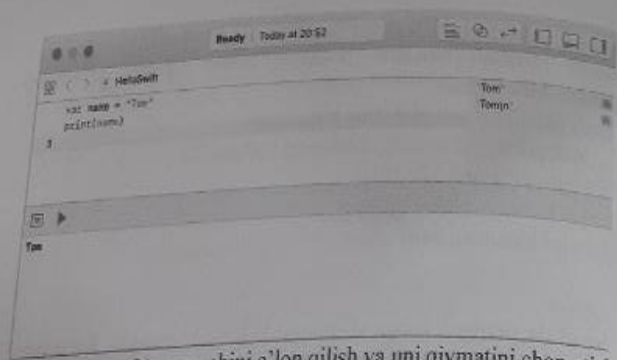
Swift dasturi ikkita xususiyatga ega: u ba’zi ma’lumotlarni saqlashi va amallarni bajarishi mumkin. Swift ma’lumotlarni saqlash uchun o’zgaruvchilardan foydalanadi. O’zgaruvchi qiymatga ega bo’lgan xotiradagi nomlangan joyni ifodalaydi. O’zgaruvchilar nom va qiymatga ega. `Var` kalit so’zi o’zgaruvchini e’lon qilish uchun ishlatiladi. Misol uchun:

```
var name = "Tom"
```

O’zgaruvchi `name` deb nomlanadi va u qiymati sifatida `Tom` qatorini saqlaydi. O’zgaruvchi aniqlangandan so’ng, uni ishlatish mumkin, masalan, qiymatini chop etish uchun uni `print()` funksiyasiga o’tkazish mumkin (6.8-rasm):

```
var name = "Tom"
```

```
print(name)
```



6.8-rasm. O'zgaruvchini e'lon qilish va uni qiymatini chop etish

O'zgaruvchilarning o'ziga xos xususiyati shundaki, dastur davomida ularning qiymatini bir necha marta o'zgartirish mumkin. Misol uchun:

```

var name = "Tom" // o'zgaruvchining qiymati - Tom
name = "Karl" // o'zgaruvchining qiymati - Karl

```

O'zgaruvchilardan tashqari, dasturda ma'lumotlarni saqlash uchun o'zgarmas (o'zgarmas)lardan foydalanish mumkin. O'zgarmaslar o'zgaruvchilarga o'xshaydi, ular ba'zi qiymatlarni ham saqlaydi, bundan tashqari ular *let* kalit so'zi yordamida aniqlanadi va ularni ishga tushirgandan so'ng ularning qiymatini o'zgartirilmaydi:

```

let name = "Tom" // bu o'zgarmas, uni ozgartirib bolmaydi

```

Agar dastur davomida o'zgaruvchining qiymati o'zgarmasa, u holda bu o'zgaruvchi o'rimga o'zgarmasdan foydalangan ma'kul.

Bir qatorda bir vaqtning o'zida bir nechta o'zgaruvchilar va o'zgarmaslarni belgilash mumkin. Bunday holda, ular vergul bilan ajratiladi:

```

var name = "Aziz", surname = "Asadov"

```

O'zgaruvchilar va o'zgarmaslar unikal nomlarga ega bo'lishi kerak. Dasturda bir xil nomli bir nechta o'zgaruvchilar va o'zgarmaslardan foydalanish mumkin emas.

Agar name o'zgaruvchisi bir nechta so'zlardan iborat bo'lsa, unda faqat birinchisi kichik harf bilan boshlanadi. Misol uchun:

```

var ozgName = "Karl"
var ozgStreatAddress = "St. Medisson avenue, 47"

```

Ma'lumotlar tiplari

Har bir o'zgaruvchi yoki o'zgarmas ma'lum bir turdagi qiymatni saqlaydi. Misol uchun, yuqorida ishlatilgan name o'zgaruvchisi satrni saqlaydi:

```

var name = "Karl"

```

Swift tilida quyidagi ma'lumotlar tiplari mavjud:

- *int8*: butun sonlarni ifodalaydi, hajmi 8 bit (-128 dan 127 gacha);
 - *uint8*: musbat butun sonlarni ifodalaydi, hajmi 8 bit (0 dan 255 gacha);
 - *int16*: butun sonlarni ifodalaydi, hajmi 16 bit (-32768 dan 32767 gacha);
 - *uint16*: musbat butun sonlarni ifodalaydi, hajmi 16 bit (0 dan 65535 gacha);
 - *int32*: butun sonlarni ifodalaydi, hajmi 32 bit (-2147483648 dan 2147483647 gacha);
 - *uint32*: musbat butun sonlarni ifodalaydi, hajmi 32 bit (0 dan 4294967295 gacha);
 - *int64*: butun sonlarni ifodalaydi, hajmi 64 bit (-9223372036854775808 dan 9223372036854775807 gacha);
 - *uint64*: musbat butun sonlarni ifodalaydi, hajmi 64 bit (0 dan 18446744073709551615 gacha);
 - *int*: 1, -30, 458 kabi butun sonlarni ifodalaydi. 32-bitli platformalarda *Int32*, 64-bitli platformalarda *Int64* ga teng;
 - *uint*: 1, 30, 458 kabi musbat butun sonlarni ifodalaydi. 32 bitli platformalarda *UInt32* va 64 bitli platformalarda *UInt64* ga teng;
 - *float*: 32-bitli suzuvchi nuqtali raqam, kasr qismida 6 tagacha raqam mavjud;
 - *double*: 64-bitli suzuvchi nuqtali raqam, kasr qismida 15 tagacha raqamni o'z ichiga oladi;
 - *float80*: 80-bitli suzuvchi nuqta raqami;
 - *bool*: mantiqiy true yoki false qiymatini ifodalaydi;
 - *string*: satrni ifodalaydi;
 - *character*: bitta belgini ifodalaydi.
- O'zgaruvchilar va o'zgarmaslar tipi aniq yoki yashirin tarzda belgilanishi mumkin. Dasturda tiplarni aniq belgilash mumkin:
- ```

var yosh: Int = 46
var name: String = "Karl"

```
- Tipga ega o'zgaruvchining ta'rifi shablona muvofiq amalga oshiriladi:
- ```

variable_name: o'zgaruvi_tipi = o'zgaruvchi_qiymat

```

Bundan tashqari, avval o'zgaruvchini belgilash va keyin unga qiymat berish mumkin:

```
var name: String  
name = "Karl"
```

Bir vaqtning o'zida bir xil tipdagi o'zgaruvchilar to'plamini ham belgilash mumkin:

```
var balandligi, vazni: Double
```

Yashirin tiplar

Agar o'zgaruvchilar va o'zgarmaslar tipi aniq ko'rsatilmasa, u saqlangan qiymat asosida tizim tomonidan avtomatik ravishda chiqariladi. Misol uchun:

```
var name = "Karl"
```

Name o'zgaruvchisining tipini aniq belgilamaydi, lekin u satrni saqlagani uchun tizim bu o'zgaruvchiga String tipidagi obyekt sifatida qaraydi. Yoki, masalan:

```
var yosh = 46
```

Shuningdek, u o'zgaruvchining tipini aniq ko'rsatmaydi, shuning uchun tizim bu o'zgaruvchiga Int obyektini, ya'ni butun son sifatida qaraydi.

Biroq, ushbu yondashuv bilan Swift har doim ham kerak bo'lishi mumkin bo'lgan tiplarni aniqlamaydi. Masalan, Swift barcha butun sonlarni Int tipidagi obyektlar sifatida, kasr sonlarni esa Double tipidagi obyektlar sifatida ko'rib chiqadi. Noto'g'ri vaziyatlarga tushmaslik uchun buni hisobga olish kerak. Misol uchun:

```
var a = 5.7 // Double tipi
```

```
var g: Float = 1,2
```

```
a = g // Xato - har xil tiplar
```

Misolda, tayinlangan qiymatga asoslanib, a o'zgaruvchisi Double tipini va g o'zgaruvchisi Float tipini ifodalaydi, shuning uchun a o'zgaruvchini g ga belgilash xatolikka olib keladi.

Raqamli ma'lumotlarni yozish formatlari

Swift tili kasr tizimi bilan ishlaydi. Biroq, u boshqa tizimlar bilan ham ishlashi mumkin:

- o'nlilik: sonlar qanday bo'lsa shunday, hech qanday old qo'shimchalarsiz ishlatiladi;

- ikkilik: sondan oldin 0b prefiksi ishlatiladi;

- sakkizlik: sondan oldin 0o prefiksi ishlatiladi;

- o'n oltilik: sondan oldin 0x prefiksi ishlatiladi.

Masalan, barcha sanoq sistemalarida 10 raqamini yozilishi:

```
let c = 10
```

```
let b = 0b1010 // 10 ikkilik tizimida
```

```
let t = 0o12 // 10 sakkizlik tizimida
```

```
let n = 0xA // 10 o'n oltilik tizimida
```

Suzuvchi nuqtali raqamlar ikki tizimda yozilishi mumkin: o'nlilik va o'n oltilik. O'nlilik sanoq sistemasida uzun sonlarni yozishni soddalashtirish uchun e (ko'rsatkich) belgisidan foydalanish mumkin. Misol uchun:

```
var k = 5.7e2 // 57
```

```
var y = 3.7e-2 // 0.037
```

O'zgaruvchan nuqtali raqamlarni o'n oltilik tizimda yozish uchun p prefiksidan foydalanish mumkin. Misol uchun:

```
var p = 0xFp2 // 15 * 2 darajasida 2 yoki 60,0
```

```
var x = 0xFp-2 // 15 * 2 darajasida -2 yoki 3,75
```

Arifmetik amallar

Swift tili arifmetik amallar to'plamiga ega. Raqamlar ustida arifmetik amallar bajariladi:

- +, ikkita raqam qo'shilishi:

```
var p = 5
```

```
var c = 6
```

```
var x = p + c // 11
```

- -, ikkita sonni ayirish:

```
var p = 7
```

```
var c = 2
```

```
var x = p - c // 5
```

- -, unar minus. -1 ga ko'paytirilgan sonni qaytaradi:

```
var p = -11
```

```
var c = -p // 11
```

```
var x = -c // -11
```

- *, ko'paytirish:

```
var p = 4
```

```
var c = 8
```

```
var x = p * c // 32
```

- /, bo'lish:

```
var p = 15
```

```
var c = 3
```

```
var x = p/c // 5
```


Bo'lishda, bo'linishda qanday ma'lumotlar ishtirok etishi va natija qanday bo'lishini ko'rib chiqish lozim. Masalan, quyidagi holatda kasr sonlarni bo'lish amalga oshiriladi:

```
let p: Double = 10
let c: Double = 4
let x: Double = p / c // 2.5
```

Double operatsiyasining natijasi Double qiymati bo'lib, u 2,5 ga teng. Ammo Int tipidagi qiymatlar olinsa, natija quyidagicha bo'ladi:

```
let p: Int = 10
let c: Int = 4
let x: Int = p / c // 2
```

Ikkala operandlar ham Int tipiga ega, shuning uchun amalning natijasi Int tipidagi qiymatdir. Bu kasr bo'lishi mumkin emas, shuning uchun kasr qismi tashlanadi va natijada 2,5 emas, balki 2 raqami chiqadi.

• %, bo'lishning qolgan qismini qaytaradi:

```
var p = 10
var c = 4
var x = p % c // 2
```

Arifmetik amallarda ular faqat bir xil turdagi o'zgaruvchilar orasida bajarilishini hisobga olish kerak. Misol uchun, quyidagi misolda xatolik yuz beradi:

```
var p: Int8 = 35
var c: Int32 = 12
var x = p + c
```

Misolda, p va c bir xil tipni ifodalashi kerak.

Shuningdek, arifmetik amallar operandlari tegishli turdagi obyektini qaytaradi. Misol uchun, quyidagi misolda xatolik yuz beradi:

```
var p: Int8 = 34
var c: Int8 = 26
var x: Int32 = p + c
```

Misolda, p va c kabi x o'zgaruvchisi Int8 tipini ifodalashi kerak.

Bir qator amallar arifmetik amallar o'zlashtirish orqali birlashtiriladi.

• +=, qo'shish orqali o'zlashtirish, joriy o'zgaruvchiga qandaydir qiymat qo'shadi:

```
var c = 5
c += 20
print(c) // 25
// ekvivalent
// c = c + 20
```

• -=, ayirish orqali o'zlashtirish, joriy o'zgaruvchidan qandaydir qiymatlarni ayiradi:

```
var p = 13
p -= 8
print(p) // 7
// ekvivalent
// p = p - 8
```

• *=, ko'paytirish orqali o'zlashtirish, joriy o'zgaruvchini qandaydir qiymatga ko'paytiradi va unga ko'paytirish natijasini beradi:

```
var p = 12
p *= 4
print(p) // 48
// ekvivalent
// p = p * 4
```

• /=, bo'lish orqali o'zlashtirish, joriy o'zgaruvchining qiymatini boshqa qiymatga bo'lib, bo'lish natijasini beradi:

```
var c = 9
c /= 3
print(c) // 3
// ekvivalent
// c = c / 3
```

• %=, qoldiqli bo'lish orqali o'zlashtirish, joriy o'zgaruvchining qiymatini boshqa qiymatga bo'lib, o'zgaruvchiga bo'lishning qoldiq qismini beradi:

```
var p = 36
p %= 6
print(p) // 6
// ekvivalent
// p = p % 6
```

Arifmetik operatsiyalarda barcha operandlar bir xil ma'lumotlar tipini ifodalashi kerak. Amal bajarilishi natijasi operandlar tipi bilan bir xil turdagi qiymatdir:

```
var k: Int8 = 5
var a: Int8 = 7
var y: Int8 = k + a
print(y) // 12
```

Biroq, operandlar har doim ham bir xil tipni ifodalamaydi. Bundan tashqari, operatsiya natijasi tayinlangan o'zgaruvchi yoki o'zgarmas tipi har

doim ham operandlar tipiga to'g'ri kelmaydi. Bunday hollarda tipdagi konversiyalarni amalga oshirish kerak.

Agar operatsiyalar operandlari har xil turdagi sonli harflarni ifodalasa, Swift avtomatik ravishda konvertatsiyani amalga oshiradi;

```
let y = 10/3.0
```

```
print (y) // 3.333333333333333
```

Misolda 10 soni *Int* tipini, 3.0 esa *Double* tipini ifodalaydi. Bunda y *Double* tipiga o'tkaziladi va bo'lish amalga oshiriladi.

Agar operandlar o'zgarmaslar yoki o'zgaruvchilarni ifodalasa yoki boshqa operatsiya yoki ifodaning natijasi bo'lsa, unda aniq turdagi konvertatsiya qilinishi kerak. Buning uchun maxsus funksiyalar - ma'lumotlar tiplarini ishga tushirish moslamalari qo'llaniladi. Ular ma'lumotlar tipi nomlari bilan bir xil: *Int8()*, *Int()*, *Float()*, *Double()* va boshqalar. *Double()* funksiyasi qiymatni *Double* tipiga o'zgartiradi va qiymatning o'zi qavs ichiga o'tkaziladi. Misol uchun:

```
let s = 4 // Int tipini ifodalaydi
```

```
let t = Double(s) // Double tipiga aylanadi
```

```
print (t) // 4.0
```

Bunday holda, *Int* tipidagi qiymatni *Double* tipidagi o'zgaruvchiga to'g'ridan-to'g'ri o'tkazish mumkin emas, konversiyadan keyin ikkala o'zgaruvchi ham 5 raqamini o'z ichiga oladi:

```
let p = 5 // Int tipini ifodalaydi
```

```
let e: Double = p // Xato - har xil tiplar
```

Bu yerda aniq turdagi konvertatsiyadan foydalanish kerak:

```
let e = Double(d)
```

Agar operandlar har xil tiplarni ifodalasa, tipdagi konvertatsiya yordam berishi mumkin. Masalan, quyidagi holatda xatolik yuz berishi mumkin:

```
let p = 10
```

```
let c = 3.0
```

```
let y = p / c // kompilyatsiya xatosi
```

Raqamlar bilan razryad amallarini bajarish

Razryad amallari yoki bit bo'yicha operatsiyalar butun sonlarning alohida bitlarida bajariladi. Har bir raqam xotirada ma'lum bir tasvirga ega. Masalan, 5 soni ikkilik tizimda 101 sifatida ifodalanadi. 7 soni 111 ga teng.

Swift tili quyidagi bit bo'yicha operatsiyalarga ega:

- **&** (mantiqiy ko'paytirish amali yoki *AND* operatsiyasi), ikkita raqamning mos keladigan ikkita raqamini taqqoslaydi va agar ikkala raqamning mos raqamlari 1 bo'lsa, 1 ni qaytaradi. Aks holda, 0 ni qaytaradi.

```
let p = 6 // 110
```

```
let q = 5 // 101
```

```
let c = p & q // 100 - 4
```

```
print (c) // 4
```

Misolda *p* 6 ga teng, bu ikkilik tizimda 110 ga, ikkilik tizimda *q* 5 yoki 101 ga teng. Natijada, operatsiyani bajarishda 4 yoki 100 raqami olinadi.

- **|** (mantiqiy qo'shish operatsiyasi yoki *OR* operatsiyasi), agar ikkala raqamning mos keladigan raqamlaridan kamida bittasi 1 bo'lsa, 1 ni qaytaradi. Aks holda, 0 ni qaytaradi.

```
let p = 6 // 110
```

```
let q = 5 // 101
```

```
let c = p | q // 111 - 7
```

```
print (c) // 7
```

- **^** (*XOR* operatsiyasi), agar ikkala raqamning mos keladigan raqamlari teng bo'lmasa, 1 ni qaytaradi.

```
let p = 6 // 110
```

```
let q = 5 // 101
```

```
let c = p ^ q // 011 - 3
```

```
print (c) // 3
```

- **~** (inversiya operatsiyasi yoki *EMAS* operatsiyasi), bitta operandni oladi va uning barcha bitlarini inversiya qiladi. Agar razryad 1 bo'lsa, u 0 ga aylanadi va aksincha, agar razryad 0 bo'lsa, u 1 ga aylanadi.

```
let p = 6 // 000000000000000110
```

```
let q = ~p // 11111111111111001
```

```
print (q) // -7
```

- **<<** (razryad bo'yicha chapga siljish), birinchi operandning bitlarini ikkinchi operand tomonidan belgilangan razryadlar soniga qarab chapga siljitadi. Ya'ni, o'ngdagi birinchi operand nollar bilan to'ldirilgan bo'lib, ularning soni ikkinchi operandga teng.

```
let p = 6 // 110
```

```
let q = 2
```

```
let c = p << q // 110 << 2 = 11000
```

```
print (c) // 24
```

- **>>** (razryad bo'yicha o'ngga siljish), birinchi operandning bitlarini ikkinchi operand tomonidan belgilangan razryadlar soni bo'yicha o'ngga siljitadi.

```
let p = 13 // 1101
```

```
let q = 2
```

```
let c = p >> q // 1101 >> 2 = 11
```

```

print (c) // 3
Shuningdek, razryadli operatsiyalarni o'zlashtirish bilan
birlashtiradigan bir qator operatsiyalar mavjud:
• &=: razryad bo'yicha VA operatsiyadan keyin o'zlashtirish.
• |=: razryad bo'yicha OR operatsiyadan keyin o'zlashtirish.
• ^=: XOR operatsiyadan keyin o'zlashtirish.
• <<=: razryad bo'yicha chapga siljitgandan keyin o'zlashtirish.
• >>=: razryad bo'yicha o'ngga siljitgandan keyin o'zlashtirish.
Operatsiyalarni qo'llash bo'yicha quyida misollar keltirilgan:
var p = 13 // 1101
p &= 5 // 1101 & 0101 = 0101
print (p) // 5
p |= 6 // 0101 | 0110 = 0111
print (p) // 7
p ^= 4 // 111^100 = 011
print (p) // 3
p <<= 3 // 11 << 3 = 11000
print (p) // 24
p >>= 1 // 11000 >> 1 = 1100
print (p) // 12

```

Belgilar va string turlari

Matn bilan ishlash uchun ikki turdagi ma'lumotlardan foydalaniladi: *Character* va *String*. Belgilar bitta belgini, *String* esa bir nechta belgilar qatorini ifodalaydi. *String* funkcionalligi jihatidan har xil bo'lgan *Character* obyektlari to'plami emas, balki alohida tipdir.

Satr va belgilarni aniqlashning eng oson yo'li *String* literallaridan foydalanishdir, ya'ni qo'sh tirnoq ichiga olingan qiymatlar:

```

var p:Character = "s"
var salom: String = "salom"

```

Satrdan farqli o'laroq, *Character* tipidagi o'zgaruvchiga bir nechta belgi kirita olmaysiz, aks holda xatolik yuzaga keladi:

```

var p:Character = "ghjk" // Xato

```

Bo'sh satr "" belgilari yoki *String()* yordamida yaratiladi. Satrlar bo'sh bo'lishi mumkin va hech narsa o'z ichiga olmaydi, lekin baribir ishga tushirilishi mumkin:

```

var str1: String = ""
var str2: String = String()

```

Satrdan boshqarish ketma-ketligi yoki eskeyp ketma-ketligi deb ataladigan maxsus belgilar bo'lishi mumkin. Ulardan asosiy lari:

- \n: yangi qatorga o'tish
- \t: tab tabulyatsiya
- \" : qo'shtirnoq
- \\: teskari chiziq

Misol uchun, qo'shtirnoq yoki teskari chiziqdan foydalanadigan ko'p qatorli matnni kiritish quyidagicha yoziladi:

```

let text = " MChJ \ \"Favorit\" \n Direktori: Davletov"
print (text);
XCode dasturlash muhiti berilgan qatorni quyidagicha izohlaydi:
MChJ \"Favorit Direktori: Davletov\"
Favorit Direktori: Davletov
Boshqa ma'lumotlar tiplarining qiymatlarini satrga aylantirish uchun o'zgartirilmoqchi bo'lgan qiymatni o'zgaruvchiga o'tkazish kerak:
var raqam: Int = 76
var str: String = String(raqam) // \"76\"

```

Satrlarni birlashtirish
 Satrlarni birlashtirish (konkatenatsiya) uchun + (qo'shish) operatoridan foydalaniladi:

```

var satr: String = "" // ""
satr += "salom" // "salom"

```

Interpolyatsiya - bu turli xil harflar va qiymatlarni boshqa ma'lumotlar tiplarini ifodalashi mumkin bo'lgan qatorga birlashtirishdir. *Interpolyatsiya* qilish uchun boshqa turdagi ma'lumotlar obyektlari oldidan teskari chiziq bilan qavslar ichiga olinadi:

```

var yosh: Int = 26
var satr: String = "Yosh: \ (yosh) " // "Yosh: 26"

```

```

var vazni: Double = 50,8

```

```

satr = "Yosh: \ (yosh) va vazni: \ (vazn) " // "Yosh: 26 va vazni: 50,8"

```

Interpolyatsiya murakkab ifodalar va operatsiyalarni ham o'z ichiga olishi mumkin, masalan:

```

let y1 = 7
let y2 = 13
let satr = "y1 + y2 = \ (y1 + y2) "
print (satr) // y1 + y2 = 20

```

Bool tipi. Shartli ifodalalar

Bool tipi mantiqiy qiymatni true (to'g'ri) yoki false (noto'g'ri) ifodalaydi. Ya'ni, Bool obyekti ikkita holatda bo'lishi mumkin:

```
var vkl: Bool = true  
vkl=false
```

Bool tipidagi obyektlar qandaydir shartni ifodalovchi shartli ifodalarnatijasi bo'lib, shartning rostligiga qarab rost yoki yolg'onni qaytaradi: true – shart rost va false – shart noto'g'ri bo'lsa.

Taqqoslash operatsiyalari

Taqqoslash operatsiyalari ikkita qiymatni taqqoslaydi va taqqoslash natijasiga qarab, Bool tipidagi obyektni qaytaradi: true yoki false.

• =, tenglik operatori. Ikki qiymatni taqqoslaydi va agar ular teng bo'lsa, true ni qaytaradi:

```
var p = 3  
var q = 3  
var c = p = q  
print(c) // true, chunki p q ga teng  
var d = 4  
c = p = d
```

```
print(c) // false, chunki p d ga teng emas
```

• !=, tengsizlik amali. Ikki qiymatni solishtiradi va agar ular teng bo'lmasa, true qiymatini qaytaradi:

```
var p = 3  
var q = 3  
var c = p != q  
print(c) // false, chunki p q ga teng  
var d = 4  
c = p != d
```

```
print(c) // true, chunki p d ga teng emas
```

• >, ikkita qiymatni taqqoslaydi va agar birinchi qiymat ikkinchisidan katta bo'lsa, true qiymatini qaytaradi:

```
var p = 8  
var q = 5  
var c = p > q  
print(c) // true, chunki p q dan katta  
var d = 4  
c = d > p  
print(c) // false, chunki d p dan kichik
```

• <, ikkita qiymatni taqqoslaydi va agar birinchi qiymat ikkinchisidan kichik bo'lsa, true qiymatini qaytaradi:

```
var p = 8  
var q = 5  
var c = p < q  
print(c) // false, chunki p q dan katta  
var d = 4  
c = d < p
```

```
print(c) // true, chunki d p dan kichik
```

• >=, ikkita qiymatni solishtiradi va agar birinchi qiymat ikkinchisidan katta yoki unga teng bo'lsa, true qiymatini qaytaradi:

```
var p = 5  
var q = 5  
var c = p >= q  
print(c) // true, chunki p q ga teng  
var d = 4  
c = d >= p
```

```
print(c) // false, chunki d p dan kichik
```

• <=, ikkita qiymatni solishtiradi va agar birinchi qiymat ikkinchisidan kichik yoki teng bo'lsa, true qiymatini qaytaradi:

```
var p = 8  
var q = 5  
var c = p <= q  
print(c) // false, chunki p q dan katta  
var w = 6  
c = w <= p  
print(c) // true, chunki w p dan kichik
```

Mantiqiy operatsiyalar

Mantiqiy operatsiyalar Bool tipidagi obyektlarda bajariladi va natijada Bool obyektni ham qaytaradi.

• !, mantiqiy "YO'Q" yoki inkor operatsiyasi. Obyektning qiymatini o'zgartiradi: agar u true ga teng bo'lsa, u holda operatsiya false qaytaradi va aksincha:

```
var val: bool = rost  
var natija = !val // false
```

• &&, mantiqiy "VA" yoki mantiqiy ko'paytirish amali. Operatsiyaning ikkala operandlari ham true bo'lsa, u true qiymatini qaytaradi:

```

let val: Bool = true
let birhil = true
let natija = val && birhil // true - ikkala operand ham true
let p: Bool = true
let q: Bool = false
let c: Bool = true

```

let w = p && b && c // false, chunki q = false
 • ||, mantiqiy "YOKI" yoki mantiqiy qo'shish amali. Agar operatsiya operandlaridan kamida bittasi true bo'lsa, u true ni qaytaradi:

```

var val: Bool = true
var birhil = false
val || birhil // true, chunki val true ga teng
var p: Bool = true
var q: Bool = false
var c: Bool = false
p || q || c // true, chunki p = true

```

Ko'pincha mantiqiy operatsiyalar bir nechta taqqoslash operatsiyalarini birlashtiradi:

```

let p = 10
let q = 12
let c = p > 8 && q < 10
let w = p > 8 || q < 10
print(c) // false
print(w) // true

```

Shartli operator

Dasturlash tillarida ko'pincha muayyan shartlarning bajarilishiga qarab ma'lum harakatlarni bajarish kerak bo'ladi. Agar ma'lum shartlar bajarilsa, bitta harakatni bajarish lozim, agar bajarilmasa, boshqa shartni bajarish lozim. Dasturlash tillarida shartli konstruksiyalar qo'llaniladi. Swift tilida quyidagi shartli konstruksiyalar mavjud.

If/else tuzilishi

If tuzilmasi ma'lum bir shartning haqiqatini tekshiradi va tekshirish natijalariga qarab, ma'lum bir kodni bajaradi:

```

if shart {
  // harakatlar to'plami
}

```

Misol uchun:

```
let sif1 = 14
```

```

let sif2 = 9
if sif1 > sif2 {
  print("sif1 sif2 dan katta")
}

```

Bu yerda, agar birinchi raqam ikkinchisidan katta bo'lsa, unda ochilish va yopish figurali qavslar orasida joylashgan if blokidagi barcha kod ishlaydi. Agar birinchi raqam ikkinchisidan kichik bo'lsa, if tuzilmasidagi amallar ishlamaydi.

If so'zidan keyin Bool tipidagi qiymat kelishi kerak. Taqqoslash operatsiyasining natijasi mantiqiy qiymatni qaytaradi, keyin misolda hech qanday xatolik yuzaga kelmaydi. Agar if dan keyin raqam yoki qator berilsa, dastur xato bilan tugaydi:

```

let sif1 = 22
let sif2 = 15
if sif1 {
  print("sif1 sif2 dan katta")
}

```

Agar shartni tekshirishda ba'zi amallarni bajarish kerak bo'lsa, else blokidan foydalanish mumkin:

```

let sif1 = 22
let sif2 = 15
if sif1 > sif2 {
  print("sif1 sif2 dan katta")
}
else {
  print("sif1 sif2 dan kichik")
}

```

Raqamlarni solishtirganda uchta holatni hisoblash mumkin: birinchi raqam ikkinchidan katta, birinchi raqam ikkinchidan kichik va raqamlar teng. Else if tuzilmasidan foydalanib, qo'shimcha shartlarni bajarish mumkin:

```

let sif1 = 14
let sif2 = 9
if sif1 > sif2 {
  print("sif1 sif2" dan katta)
}
else if (sif1 < sif2){
  print("sif1 sif2 dan kichik")
}

```

```

else {
    print("sif1 va sif2 teng")
}

```

Ternar operator oddiy if tuzilmasiga o'xshaydi va quyidagi sintaksisga

ega: [birinchi operand - shart]? [ikkinchi operand]: [uchinchi operand]
 Shartga qarab uchlik operator ikkinchi yoki uchinchi operandni qaytaradi: agar shart rost bo'lsa, ikkinchi operand qaytariladi; agar shart noto'g'ri bo'lsa, uchinchi. Misol uchun:

```

var sif1 = 23
var sif2 = 11
var son3 = sif1 > sif2? sif1 - sif2 : sif1 + sif2

```

Misolda *son3* 11 ga teng bo'ladi, chunki *sif1* *sif2* dan katta, shuning uchun ikkinchi operand bajariladi: *sif1 - sif2*.

Switch/case

Switch/case tuzilmasi *if/else* tuzilmasiga o'xshaydi, chunki u bir vaqtning o'zida bir nechta shartlarni bajarishga imkon beradi:

```

var num: Int = 31
switch num {
case 0:
    print("O'zgaruvchi 0 ga teng")
case 10:
    print("O'zgaruvchi 10 ga teng")
case 31:
    print("O'zgaruvchi 31 ga teng")
default:
    print("Raqamni tanib bo'lmadi")
}

```

Switch kalit so'zidan keyin taqqoslanadigan ifoda keladi. Bu o'zgaruvchan yoki doimiy bo'lishi mumkin. Ushbu ifodaning qiymati ketma-ketlik bilan *case* dan keyin joylashtirilgan qiymatlar bilan taqqoslanadi. Agar moslik topilsa, *case* bloki bajariladi. Agar moslik topilmasa, *default* operatori bajariladi.

Misolda, *num* o'zgaruvchisi 31 bo'lganligi sababli, quyidagi *case* bloki bajariladi:

```

case 31:
    print("O'zgaruvchi 31")

```

Case blokining oxirida bajarishni to'xtatish va *switch/case* blokidan chiqish uchun *break* operatori joylashtiriladi. *Swift* da bunday hollarda

break operatoridan foydalanish shart emas. Biroq, ba'zi bir ma'lum qiymatlarni qayta ishlash va *switch* tuzilmasidan chiqish bor. Bunday holda, *case* yoki *default* amallardan keyin *break* iborasini belgilash mumkin:

```

var num: Int = 0
switch num {
case 0:
    print("O'zgaruvchi 0")
case 10:
    tanaffus
case 31:
    print("O'zgaruvchi 31")
default:
    break
}

```

Misolda, agar *num* 10, 0 yoki 31 dan boshqa raqam bo'lsa, *switch* dan chiqadi.

Boshqa barcha qiymatlarga mos kelish uchun pastki chiziq `_` dan foydalanish mumkin:

```

let raqam = 7
switch num {
case 4:
    print("Raqam = 4")
case 3:
    print("Raqam = 3")
case _:
    print("Raqam 4 ham, 3 ham emas, lekin aniq emas")
}

```

Bundan tashqari, ifodani bitta qiymat bilan emas, balki qiymatlar guruhi bilan solishtirish mumkin:

```

var num: Int = 20
switch num {
case 0, 10: // agar raqam 0 yoki 10 bo'lsa
    print("O'zgaruvchi 0 yoki 10")
case 11.. $<20$ : // agar son 11 dan 20 gacha bo'lsa, 20 dan tashqari
    print("O'zgaruvchi 11 dan 20 gacha")
case 20...30: // agar son 20 dan 30 gacha bo'lsa
    print("O'zgaruvchi 20 dan 30 gacha")
default:
    print("Raqamni tanib bo'lmadi")
}

```

}
 case 0, 10 operatori taqqoslash uchun ikkita qiymatni, 0 va 10 ni belgilaydi va ifoda ushbu qiymatlardan biriga teng bo'lsa, ishga tushadi.
 case 11..<20 operatori 11 dan 20 gacha (20 dan tashqari) qiymatlarning butun diapazonini belgilaydi va agar ifoda ushbu diapazondagi qiymatga teng bo'lsa, ishga tushadi.
 case 20...30 operatori 20 dan 30 gacha bo'lgan qiymatlarning butun diapazonini (shu jumladan ikkala raqamni ham) belgilaydi va agar ifoda ushbu diapazondagi qiymatga teng bo'lsa, ishga tushadi.

Swift 4 da bitta diapazon chegarasini o'tkazib yuborish mumkin:

```
let i = 8
switch i {
case ...<0:
  print("i manfiy son")
case 1...:
  print("i - ijobiy raqam")
case 0:
  print("i 0 ga teng")
default:break
}
```

Oddiy turdagi ifodalarga qo'shimcha ravishda kortejlarni solishtirish mumkin:

```
let GrajInfo = ("Karl", 22)
switch GrajInfo {
case ("Anvar", 33):
  print("Ismi: Anvar, yoshi: 33")
case (_, 22):
  print("Ism: \GrajInfo.0) va yoshi: 22")
case ("Karl", _):
  print("Ismi: Karl va yoshi: \GrajInfo.1)")
case ("Karl", 1...30):
  print("Ismi: Karl va yoshi 1 dan 30 gacha")
default:
  print("Ma'lumot tanib olinmadi")
}
```

Bu yerda *GrajInfo* korteji ketma-ket holda holatlar ketma-ketligidagi uchta kortej bilan taqqoslanadi. Taqqoslashda to'liq kortejni belgilash mumkin:

```
case ("Anvar", 33):
```

```
print("Ismi: Anvar, yoshi: 33")
Yoki pastki chiziq o'rniga _ ni qo'yish orqali kortej elementlaridan birini ham o'tkazib yuborishi mumkin:
case (_, 22):
```

```
print("Ism: \GrajInfo.0) va yoshi: 22")
```

Bunday holda, kortejning ikkinchi elementi 22 ga teng bo'lsa, kortejning birinchi elementi qanday bo'lishi muhim emas.

Raqamli ma'lumotlar uchun aniq qiymatni emas, balki qiymatlar oralig'ini ham belgilash mumkin:

```
case ("Karl", 1...30):
```

```
print("Ismi: Karl va yoshi 1 dan 30 gacha")
```

Bunday holda, kortejning ikkinchi elementi 1 dan 30 gacha bo'lishi kerak.

Ishlatilgan *switch/case* tuzilmasida taqqoslangan ifoda uchta holat ketma-ketligiga mos keladi – ikkinchi, uchinchi va to'rtinchi, lekin ulardan faqat birinchisi bajariladi. Ammo keyingi *case* ketma-ketligi (yoki *default* operatori) bajarilishi uchun oldingi *case* blokining oxirida *fallthrough* operatoridan foydalanish kerak:

```
let GrajInfo = ("Karl", 22)
```

```
switch GrajInfo {
```

```
case ("Anvar", 33):
```

```
  print("Ismi: Anvar, yoshi: 33")
```

```
case (_, 22):
```

```
  print("Ism: \GrajInfo.0) va yoshi: 22")
```

```
  fallthrough
```

```
case ("Karl", _):
```

```
  print("Ismi: Karl va yoshi: \GrajInfo.1)")
```

```
default:
```

```
  print("Ma'lumot tanib olinmadi")
```

```
}
```

Qiymatni bog'lash mexanizmi o'zgaruvchilar va o'zgarmaslarni *case* bloklarida aniqlash imkonini beradi, ularning qiymatlari taqqoslangan ifoda qiymati bilan bog'lanadi:

```
let number = 5
```

```
switch number {
```

```
case 1:
```

```
  print("Raqam = 1")
```

```
case 2:
```

```
  print("Raqam = 2")
```

```
case let n:
  print ("Raqam = \n")
}
```

Misolda, agar *number* qiymati 1 va 2 ga teng bo'lmasa, u holda uning *case* blokida ishlatiladigan *n* doimiyiga o'tkaziladi.

Bunday holda, bog'lanish ko'proq o'zgaruvchilar va o'zgarmaslar bilan, yoki kortejlar bilan bajarilishi mumkin:

```
let GrajInfo = ("Karl", 22)
switch GrajInfo {
case (let name, 22):
  print("Ismi: \n(name) va yoshi: 22")
case ("Karl", let age):
  print("Ismi: Karl va yoshi: \n(age)")
case let (name, age):
  print("Ismi: \n(name) va yoshi: \n(age)")
}
```

GrajInfo-dagi ikkinchi element 22 bo'lsa, blok ishga tushadi:

```
case (let name, 22):
  print("Ismi: \n(name) va yoshi: 22")
```

Bu erda *name* o'zgaruvchisi *GrajInfo* kortejidan birinchi elementning qiymatini oladi. Ushbu tuzilmada *default* blokidan foydalanmaydi, chunki blok

```
case let (name, age):
  print("Ismi: \n(name) va yoshi: \n(age)")
```

oldingi holatlar *case* operatorlariga kirmaydigan barcha mumkin bo'lgan holatlarni samarali ravishda bekor qiladi. Bu blok *GrajInfo* kortejining elementlaridan tashkil topgan o'zgarmasni belgilaydi.

Agar *case* blokini bajarishda qo'shimcha shartlar belgilansa, buning uchun *where* operatoridan foydalanish mumkin. Misol uchun:

```
let i = 8
switch i {
case let k where k < 0:
  print("i manfiy son")
case let k where k > 0:
  print("i musbat son")
case 0:
  print ("i 0 ga teng")
default:break
}
```

Kortejlarga misol:
let GrajInfo = ("Karl", 22)

```
switch GrajInfo {
case ("Karl", _) where GrajInfo.1 > 10 && GrajInfo.1 < 15:
  print("Ismi: Karl va yoshi 10 dan 15 gacha")
case ("Karl", _) where GrajInfo.1 >= 20:
  print("Ismi: Karl va yoshi 20 dan katta")
default:
  print ("Noma'lum")
}
```

Bu erda *where* ifodasi qo'shimcha shartlarni belgilaydi. Agar ular bajarilmasa, u holda *case* bloki ham bajarilmaydi.

nil va *Optional* turlari

Optional turlar qiymatga ega bo'lishi yoki bo'lmasligi mumkin bo'lgan obyektlarni ifodalaydi. *Optional* turlar asosiy turlarga o'xshaydi. Ularning barchasi savol belgisi bilan tugaydi: *Int?*, *String?*. Savol belgisi bu *optional* tur ekanligini bildiradi.

Masalan:

```
let someString = "123"
let someRaqam = Int (someString)
```

Bu yerda *Int(someString)* initsializatori *someString* satrini raqamga aylantiradi. Misolda "123" qatori 123 raqamini o'z ichiga oladi. Biroq, *someString* o'zgaruvchisi "salom" qatorini ifodalagan bo'lsa-chi? Bunday holda, initsializator qatorni raqamga aylantira olmaydi. Shuning uchun initsializator faqat *Int* obyektini emas, balki *Int?* ni, ya'ni qiymatga ega bo'lishi yoki bo'lmasligi mumkin bo'lgan obyektini qaytaradi.

Aslida, agar obyektning qiymati bo'lmasa, unga *nil* maxsus qiymati beriladi. Ushbu qiymatni kodda aniq belgilash mumkin:

```
var raq: Int? = 12
raq = nil // endi raq o'zgaruvchisi hech qanday qiymatga ega emas
Nil qiymati faqat opsional turdagi obyektlarga qo'llanilishi mumkin.
```

Aslida *Int?* turidagi yozuv *Optional<Int>* ning qisqartmasi. Ya'ni, o'zgaruvchini quyidagicha belgilash mumkin:

```
var sifra: ixtiyoriy<Int> = 12
```

Misolda, sifra o'zgaruvchisiga 12 raqami berilgan, lekin o'zgaruvchining qiymati sifatida *Optional(12)* bo'ladi, uni quyidagicha yozish mumkin:

```
var sifra : Optional <Int>= Optional(12)
```



```
// yoki shunday
var sifra2 = Optional(12)
Shu bilan birga, Optional<Int> emas, balki Optional<String> yoki
Optional<Double> ekanligini tushunish kerak, masalan:
var sifra = Optional(12)
sifra = Optional("12") // Xato sifra Optional<Int> turini bildiradi,
// Optional<String> emas
```

Optional turdagi obyektlar bilan ishlashda ular oddiy turdagi obyektlarga ekvivalent emas. Quyidagi misol ishlamaydi:

```
var p: Int? = 12
var q: Int = 10
var c = p + q // xato - har xil turlar
```

Bu erda *p* va *q* har xil turdagi o'zgaruvchilar, lekin ikkala o'zgaruvchi ham butun sonlarni saqlaydi va *optional* turdagi obyektlar bilan to'liq ishlash uchun ulardan qiymat olish kerak. ! operatori qiymatni olish uchun ishlatiladi. *Optional* turdagi obyekt nomidan keyin undov belgisi qo'yiladi. Bu operator shuningdek *unwrap* operatori yoki *forced unwrap* operatori deb ham ataladi:

```
var p: Int? = 12
var q: Int = 10
var c = p! + q // c = 22
```

Yana bir misol:

```
var q: Int = 10
var p: Int? = Int("123")
q = p! + q
print(p!) // 123
print(q) // 133
```

Swift ushbu turlarning qiymatini olishning yana bir usulini taqdim etadi, ya'ni optional turlardan bilvosita ochildmagan *Optional* bilan foydalanish:

```
var q: Int = 10
var p: Int! = Int("123")
q = p + q
print(p) // 123
print(q) // 133
```

Bu erda *p* o'zgaruvchisi *Int?* emas, *Int!* tipidadir. Aslida, bu bir xil *Optional* emas.

Agar misoldagi *p* o'zgaruvchisi ma'lum qiymatga ega bo'lmasa, dastur yana xatoga yo'l qo'yadi. Masalan, var *p* misolida: *Int! = Int("abc")*

yoki var *p: Int? = Int("abc")*. Shuning uchun, *optional* turdagi obyektlarni ishlatishdan oldin, ularning har qanday qiymatga ega ekanligini tekshirish maqsadga muvofiqdir.

Tekshirish uchun *if* shartidan foydalanish mumkin. Uning umumiy shakli:

```
if var o'zgaruvchi | let doimiy = optional_qiymat {
    actions1
} else { actions2
}
```

Agar *optional_qiymat* nilga teng bo'lmasa, u yaratilgan o'zgaruvchiga tayinlanadi va *actions1* bajariladi. Aks holda, *actions2* amalga oshiriladi.

Misol uchun:

```
var str: String = "123"
var q: Int = 10
if var p = Int(str) {
    p += q
    print(p)
}
else {
    print(q)
}
```

Agar *Int(str)* ifodasi satrni raqamga muvaffaqiyatli o'zgartirsa, ya'ni qiymatga ega bo'lsa, u holda *p* o'zgaruvchisi yaratiladi, natijada olingan qiymat beriladi va keyin kod bajariladi:

```
p += q
print(p)
```

Agar satrdan raqamga o'tkazish muvaffaqiyatsiz bo'lsa va *Int(str)* ifodasi *nil* ga qaytsa, *else* blokidagi kod bajariladi:

```
else {
    print(q)
}
```

Ammo misolda ular *nil* ni boshqa yo'l bilan tekshirishlari mumkin:

```
var str: String = "123"
var q: Int = 10
var p: Int? = Int(str)
if p != nil {
    p += q
    print(p)
}
```

```

else {
    print (q)
}

```

Agar bir nechta o'zgaruvchilar yoki o'zgaruvchilarning qiymatlarini tekshirish kerak bo'lsa, bitta *if* ifodasida ko'rsatish mumkin:

```

let p = Int("123")
let q = Int("456")
if let aVal = p, let qVal = q {
    print (pVal)
    print (qVal)
}

```

```

else {
    print ("Xato")
}

```

Misolda, agar *p* va *q* ikkalasi *nil* bo'lsa, *if* operatori bajariladi. Aks holda *else* bloki bajariladi.

Optional obyektini konkret turdagi obyekt bilan solishtirganda, Swift konkret turdagi obyektini *optional* turga o'zgartiradi:

```

let p: Int? = 10
if p == 10 {
    print("p 10 ga teng")
}
else { print("p 10 ga teng emas")
}

```

= va != amallari shunday ishlaydi. Biroq, <, >, <=, >= operatsiyalari bilan ular boshqacha. Masalan, quyidagi kod xatoga chiqaradi:

```

let p: Int? = 10
if p > 5 {
    print("p 5 dan katta")
}

```

Va bunday operatsiyalarda *optinal* obyektga ! operatori qo'llanilishi kerak:

```

let p: Int? = 10
if p != nil && p! > 5 {
    print("p 5 dan katta")
}

```

Agar *switch* tuzilmasidagi taqqoslangan qiymat *optional* obyektini ifodalasa, u holda ? agar mavjud bo'lsa, uning qiymatini olish va solishtirish mumkin:

```

let i = Int("1")
switch i {
case 1?:
    print("i 1 ga teng")
case let n?:
    print("i ga teng \(n)")
case nil:
    print("i aniqlanmagan")
}

```

?? Operator *optional* obyektning qiymatlarini *nil*-ga tekshirish imkonini beradi. Bu operator ikkita operand oladi *p ?? 10*. Agar birinchi operand *nil*ga teng bo'lmasa, u holda birinchi operandning qiymati qaytariladi. Agar birinchi operand *nil*-ga teng bo'lsa, ikkinchi operand qaytariladi:

```

let p = Int("234")
let q = p ?? 10
print(q) // 234

```

Misolda *p* doimiysi *nil*-ga teng bo'lmagani uchun *p ?? 10* bu doimiyning qiymatini qaytaradi, ya'ni 234 raqami.

for-in sikli

for-in sikli

For-in tsikli yordamida to'plam elementlarini (massivlar, to'plamlar, lug'atlar) yoki ketma-ketliklarni takrorlash mumkin. U quyidagi shaklga ega:

```

for ketma-ketlikdagi obyekt in ketma-ketligi {
    // harakatlar
}

```

Masalan, massiv elementlarini takrorlaymiz:

```

1...5 dagi element uchun {
    print (element)
}

```

1...5 ifodasi 1 dan 5 gacha bo'lgan beshta raqamdan iborat ketma-ketlikni hosil qiladi va sikl ketma-ketlikning barcha elementlaridan o'tadi. Har bir o'tishda u bitta raqamni chiqaradi va uni element o'zgaruvchisiga o'tkazadi. Shunday qilib, sikl besh marta ishlaydi.

Where operatori yordamida elementlar ketma-ketligidan tanlash shartlarini o'rnatishi mumkin:

```

for i in 0...10 where i % 2 == 0 {

```

```
print(i) // 0, 2, 4, 6, 8, 10
```

} Bu yerda 0...10 ketma-ketligidan *while* operatoridan keyingi shartga $i \% 2 == 0$ mos keladigan elementlarga, ya'ni juft sonlar ajratib olinadi.

while sikli
while operatori ba'zi shartlarni tekshiradi va agar u rost bo'lsa, u kod blokini bajaradi. Ushbu sikl quyidagi shaklga ega:

```
while {  
    // operatorlar
```

```
}
```

Misol uchun:

```
var i = 10
```

```
while i > 0 {
```

```
    print(i)
```

```
    i--
```

```
}
```

Bundan tashqari, agar shart har doim rost bo'lsa, cheksiz siklga aylanib qoladi va dasturdan chiqish imkoniyati bo'lmaydi.

Repeat-while takrorlash sikli birinchi navbatda siklni bir marta bajaradi va agar ba'zi shartlar to'g'ri bo'lsa, siklni davom ettiradi. U quyidagi shaklga ega:

```
repeat {
```

```
    // operatorlar
```

```
} while shart
```

Masalan, oldingi *while* siklini qayta yozamiz:

```
var i = 10
```

```
repeat {
```

```
    print(i)
```

```
    i--
```

```
} while i > 0
```

Bu yerda sikl ham i o'zgaruvchining qiymati 0 ga aylanmaguncha 10 marta bajariladi. Boshqa vaziyatga misol:

```
var i = -1
```

```
repeat {
```

```
    print(i)
```

```
    i--
```

```
} while i > 0
```

i o'zgaruvchisi 0 dan kichik, lekin sikl bir marta bajariladi.

Ba'zida sikl tugashini kutmasdan sikldan chiqish vaziyati yuzaga keladi. Bunday holda *break* operatoridan foydalanish mumkin.

```
for i in 0...10 {
```

```
    if i == 5 {
```

```
        break
```

```
    }  
    print(i) // 0, 1, 2, 3, 4
```

Sikl i o'zgaruvchining qiymati 5 raqamiga teng yoki yo'qligini tekshirayotganligi sababli, takrorlash 5 raqamiga yetganda, *break* operatori ishlaydi va sikl tugaydi.

Agar tekshirish paytida sikl tugamasligi nazarda tutilsa, shunchaki keyingi elementga o'tiladi. Buning uchun *continue* operatoridan foydalanish mumkin:

```
for i in 0...10 {
```

```
    if i == 5 {
```

```
        continue
```

```
    }  
    print(i) // 0, 1, 2, 3, 4, 6, 7, 8, 9, 10
```

Bunday holda, tekshiruv shartini qoniqtirmaydigan 5 raqamga yetganda, bu raqamni shunchaki o'tkazib yuboradi va ketma-ketlikning keyingi elementiga o'tadi.

Funksiyalar

Funksiya nomi (funksiya nomi) bo'lgan va dasturning turli joylarida qayta ishlatilishi mumkin bo'lgan ko'rsatmalar to'plamini ifodalaydi. Funksiya quyidagi rasmiy ta'rifga ega:

```
func funksiya_nomi (parametrlar) -> return_type {  
    // ko'rsatmalar to'plami
```

```
}
```

Avval *func* kalit so'zi, keyin esa funksiya nomi keladi. Asosan, funksiyani nomlash uchun o'zgaruvchilarni nomlash bilan bir xil bo'lgan qoidalar qo'llaniladi. Funksiya nomi uchun *camel case* rejimidan foydalaniladi.

Funksiya nomidan keyin qavslar ichida parametrlar yoziladi. Agar parametrlar bo'lmasa, shunchaki bo'sh qavslar ketadi.

Agar funksiya har qanday qiymatni qaytarsa, u holda qavs ichidagi parametrlardan keyin strelka va qaytariladigan qiymatning turi mavjud

bo'ladi. Oxirida esa figurali qavslarda, aslida funksiyani ifodalovchi kod bloki beriladi. Eng oddiy funksiyani quyidagicha ifodalash mumkin:

```
func pechName(){
    print("Mening ismim Anvar")
}
```

Bu yerda funksiya *pechName* deb ataladi. Bu funksiya hech narsani qaytarmaydi, shuning uchun bu erda qavslardan keyin darhol operatorlar to'plami bilan figurali qavslar keladi. Bu funksiya "Mening ismim Anvar" satrini chiqaradi.

Bundan tashqari, funksiya nomi bilan uni qayta-qayta chaqirish mumkin. Funksiyani chaqirish uchun uning nomi ko'rsatiladi, shundan so'ng uning parametrlari uchun qiymatlar qavs ichida keltirilgan:

```
func pechName(){
    print("Mening ismim Anvar")
}
pechName()
pechName()
pechName()
Xususan, bu erda funksiya uch marta chaqiriladi.
Dasturda funksiya parametrlaridan quyidagicha foydalaniladi:
func pechInfo(name: String, yosh: Int){
    print("Ismi: \{name\}; yoshi: \{yosh\}")
}
```

```
pechInfo(name: "Karl", yosh: 18) // Ismi: Karl ; yoshi: 18
pechInfo(name: "Roman", yosh: 35) // Ismi: Roman ; yoshi: 35
```

Parametrlar soni ixtiyoriy bo'lishi mumkin. Yuqoridagi misolda ikkita parametrdan foydalanilgan: *name* va *yosh*. Har bir parametrdan nomi va turi aniqlangan. Masalan, birinchi parametr *name* deb ataladi va *String* tipiga ega.

Funksiyani chaqirishda parametrlarning nomi va turini hisobga olish kerak. Funksiyani chaqirayotganda, uning barcha parametrlari uchun qiymatlarni nomi bilan aniqlash kerak. Ya'ni, parametr nomi ko'rsatilgan va uning qiymati ikki nuqta bilan ajratilgan: *name: "Karl"* va o'tkazilgan qiymat turi bo'yicha parametrga mos kelishi kerak:

```
printlnInfo(name: "Karl", yosh: 18)
```

Funksiyani chaqirganda, uni chaqirishda parametr nomlarini ko'rsatish shart emas. Buning uchun parametr nomidan oldin pastki chiziq qo'yiladi. Pastki chiziq va parametr nomi o'rtasida probel bo'lishi kerak:

```
func printlnInfo(name: String, _ yosh: Int){
    print("Ismi: \{name\}; yoshi: \{yosh\}")
}
```

```
} printlnInfo("Karl", 18) // Ismi: Karl ; yoshi: 18
Bunday funksiya chaqirilganda, qiymatlar pozitsiya bo'yicha parametrlarga o'tkaziladi. Bunday holda, o'tkazilgan qiymat parametr turiga mos kelishi kerak.
Agar funksiyada ikkita parametr aniqlagan bo'lsa, uni chaqirganda, standart qiymatlariga o'rnatish mumkin:
func printlnInfo(name: String = "Karl", yosh: Int = 22){
    print("Ismi: \{name\}; yoshi: \{yosh\}")
}
```

```
printlnInfo(name: "Roman", yosh: 18) // Ismi: Roman ; yoshi: 18
printlnInfo(name: "Aziz", yosh: 22) // Ismi: Aziz ; yoshi: 22
printlnInfo() // Ismi: Karl ; yoshi: 22
Funksiyani chaqirganda, ba'zi parametrlar uchun qiymat o'tkazilmaydi, keyin bu parametr standart qiymatdan foydalanadi.
Swift-dagi funksiya ba'zi qiymat yoki natijani qaytarishi mumkin:
func printlnHello(){
    print("Salom dunyo")
}
```

Bunday funksiya hech narsani qaytarmaydi, chunki qaytarish tipi ko'rsatilmagan. Bu quyidagi funksiyaga ekvivalent bo'ladi:

```
func printlnHello() -> Void {
    print("Salom dunyo")
}
func printlnHello() -> () {
    print("Salom dunyo")
}
```

Void tipi funksiya aslida hech narsani qaytarmasligini bildiradi. Qandaydir qiymat qaytaradigan funksiyaga misol:

```
func sum(_ x: Int, _ y: Int) -> Int{
    return x + y
}
```

```
println(sum(4,5)) // 9
```

```
println(sum(5,6)) // 11
```

Sum funksiyasi ikkita sonning yig'indisini qaytaradi va qaytish tipi *Int* hisoblanadi.

Agar funksiya *void* dan boshqa qiymatni qaytarsa, u holda funksiya tanasida *return* operatoridan foydalanish kerak. Ushbu operatoridan keyin qaytariladigan qiymat keladi.

Misolda *Int* qiymati qaytarilyapti, qaytarilgandan keyin *Int* qiymati yoki *Int* obyektini qaytaruvchi ifoda berilgan. *Return* operatori chaqirilgandan so'ng, funksiya tugaydi, shuning uchun *return* operatoridan keyin biron bir ko'rsatma qo'yishning ma'nosi yo'q.

Funksiya ba'zi qiymatlarni qaytargani uchun ushbu qiymatni ba'zi bir o'zgaruvchiga yoki doimiyga belgilash mumkin va undan keyin dasturda foydalanish mumkin. Funksiyaning qaytarish tipi o'zgaruvchi yoki doimiy tipiga mos kelishi kerak:

```
func sum(_ x: Int, _ y: Int) -> Int {
  return x + y }
let a = sum(4,5)
let b = sum(10, 23)
```

Funksiya bir vaqtning o'zida bir nechta qiymatlarni murakkabroq qiymat shaklida qaytarishi mumkin, masalan, kortej shaklida bo'lgan turli qiymatlarni o'z ichiga olishi mumkin.

```
func getInfo(_ ishhaqi: Double) -> (soliq: Double, ijarahaqi: Double) {
  let soliq = ishhaqi * 0,13
  let ijarahaqi = ishhaqi * 0,05
  return (soliq, ijarahaqi)
}
```

```
var losses = getInfo(11000)
print("Daromad solig'i: \${losses.tax}")
print("Ijara haqi: \${losses.rent}")
```

Funksiyalar ixtiyoriy turdagi qiymatlarni qaytarishi mumkin, ular qiymatga ega bo'lishi mumkin yoki bo'lmasligi mumkin. Agar ba'zi sharoitlarda funksiya qiymatlarni qaytarmasa, ulardan foydalanish foydali bo'lishi mumkin.

```
func o'lchashSoliq(_ ishhaqi: Double) -> Double? {
  if (ishhaqi > 1000) { return ishhaqi * 0,13 }
  return nil
}
```

```
if let soliq = o'lchashSoliq(11000) { // 1430
  print (soliq) }
if let soliq = o'lchashSoliq(110) { // nil
  print (soliq) }
```

Agar daromad 1000 dan kam bo'lsa, daromad solig'ini hisoblash uchun *o'lchashSoliq()* funksiyasi nolga teng bo'ladi. Aks holda u daromadning 13 foizini qaytaradi. Ikkala holat *Double* tipi bilan tavsiflanadi.

Funksiya qiymat qaytaradimi yoki yo'qligini tushunish uchun *if* tuzilmasidan foydalanish mumkin:

```
if let soliq = o'lchashSoliq(110) { // nil
  print (soliq) }
Ko'pnuqta (...) operatoridan foydalanib, bir xil turdagi parametrlarning ixtiyoriy sonini o'rnatish mumkin.
```

```
func sum(_ numbers: Int...) -> Int {
  var total: Int = 0
```

```
  for number in numbers {
    total += number }
  return total }
```

```
sum(1, 2, 3, 4, 5) // 15
```

Agar parametr nomlansa, uning nomi ko'rsatiladi, shundan so'ng ikki nuqtadan keyin barcha qiymatlar vergul bilan ajratiladi:

```
func sum(numbers: Int...) -> Int {
  var total: Int = 0
```

```
  for number in numbers {
    total += number }
  return total
}
```

```
sum(numbers: 1, 2, 3, 4, 5) // 15
```

Barcha funksiya parametrlari doimiy qiymatlardir va ularni o'zgartirib bo'lmaydi. Masalan, quyidagi funksiyani belgilashda xatoga duch kelinadi:

```
func increase(_ n: Int) {
  n += 10
}
```

Odatda *n* parametr doimiy hisoblanadi va uning qiymatini o'zgartirib bo'lmaydi. Agar parametr qiymatini o'zgartirish kerak bo'lsa, u *inout* kalit so'zi bilan aniqlanishi kerak:

```
func increase(_ n: inout Int) {
  n += 10 }
```

```
var d = 20
```

```
increase(&d)
```

```
print(d) // 30
```

O'zgaruvchilar funksiyaga o'tkazilganda, ularga ampersand belgisi qo'shiladi. Ya'ni, o'zgaruvchiga uning qiymatini emas, balki havola o'tkaziladi. Shunday qilib, funksiyadan tashqarida aniqlangan o'zgaruvchining qiymatini o'zgartirish mumkin. Misol, o'zgaruvchilar qiymatlarining almashinuvi:

```
func swap(a: inout Int, b: inout Int){
    let temp = a
    a = b
    b = temp }
var num1 = 10
var num2 = 13
swap(&num1, &num2)
print(num1) // 13
print(num2) // 10
```

Funksiya tipi

Har bir funksiyaning o'ziga xos tipi mavjud bo'lib, u funksiyaning parametr turlari va qaytarish tipidan iborat. Masalan, quyidagi funksiya:

```
func schet(_ x: Int, _ y: Int) -> Int{
    return x + y
}
```

Bu funksiya $(Int, Int) \rightarrow Int$ tipiga kiradi. Yoki, masalan, quyidagi funksiya:

```
func pechName(name: String){
    print(name)}
```

U $(String) \rightarrow Void$ tipiga ega.

Funksiya turidan foydalanib, ushbu turdagi o'zgaruvchilar yoki o'zgaruvchilarni belgilash va ularni ma'lum funksiyalarga dinamik ravishda belgilash mumkin:

```
func sum(_ p: Int, _ q: Int) -> Int{
    return p + q}
func razn(_ p: Int, _ q: Int) -> Int{
    return p - q }
```

var nekfun: $(Int, Int) \rightarrow Int$

```
nekfun=sum
print(nekfun(5, 4)) // 9
nekfun=razn
print(nekfun(5, 4)) // 1
```

Ikki funksiya sum va razn aniqlanadi, ular parametrlari va qiymatlarning qaytariladigan turlariga ega, lekin muayyan harakatlarda farqlanadi: bir holatda qo'shish amalga oshiriladi, ikkinchisida esa raqamlar chiqariladi.

Nekfun o'zgaruvchisi ham aniqlanadi, uning turi - Int tipidagi ikkita parametrga va Int tipidagi qaytish qiymatiga ega funksiyaga ega. Nekfun

o'zgaruvchisi sum va razn funksiyalari bilan bir xil parametrlarga va qaytariluvchi tipiga ega. Shunday qilib, dinamik ravishda o'zgaruvchini funksiyalardan biriga o'rnatish va uni chaqirish mumkin:

```
nekfun=sum
print(nekfun(5, 4)) // 9
```

Sum funksiyasi bu erda chaqiriladi. Natijada, $4 + 5$ yig'indisi olinadi. Keyin o'zgaruvchini boshqa funksiyaga dinamik ravishda o'rnatish mumkin:

```
nekfun=razn
var natija = nekfun(5, 4) // 1
```

Funksiya turlarini parametr turlari sifatida yoki boshqa funksiyalarda qaytariluvchi tiplari sifatida ishlatish mumkin. Funksiya turlari parametr turlari sifatida:

```
func sum(_ a: Int, _ b: Int) -> Int{
    return a + b}
func subtract(_ a: Int, _ b: Int) -> Int{
    return a - b }
func getResult(_ binaryFunc: (Int, Int) -> Int, _ a: Int, _ b: Int){
    let result = binaryFunc(a, b)
    print(result)
}
```

```
getResult(sum, 13, 10) // 23
getResult(subtract, 12, 8) // 4
```

getResult funksiyasi o'zining birinchi parametri sifatida funksiyani oladi. Yuqorida tavsiflangan sum va subtract funksiyalari ushbu parametr turiga mos keladi, shuning uchun ular getResult funksiyasini chaqirganda foydalanish mumkin:

```
getResult(jadval, 13, 10)
```

Funksiya tiplari qaytariluvchi tiplar sifatida

Funksiya tiplarini qaytariluvchi tiplar sifatida qaralishi quyidagi misolda keltirilgan:

```
func add(_ x: Int, _ y: Int) -> Int {return x + y}
func subtract(_ x: Int, _ y: Int) -> Int {return x - y}
func multiply(_ x: Int, _ y: Int) -> Int {return x * y}
func select _ n: Int -> (Int, Int) -> Int{
    switch n {
        case 2: return subtract
        case 3: return multiply
```

```

default: return add }
let x = 12, y = 8
var someFunc = select(1) // add
print(someFunc(x, y)) // 20
someFunc = select(2) // subtract
print(someFunc(x, y)) // 4
someFunc = select(3) // multiply
print(someFunc(x, y)) // 96

```

Select funksiyasining qaytish tipi (*Int, Int*) -> *Int* tipi hisoblanadi. Select ikkita *Int* parametrini oladigan va *Int* qiymatini qaytaruvchi funksiyani qaytarishi kerak. Ushbu ta'rifga *sum*, *multiply* va *subtract* funksiyalari mos keladi. Shuning uchun, tanlashda ma'lum bir qiymatni emas, balki *n* parametrining qiymatiga qarab ushbu funksiyalardan birini qaytariladi.

Qaytarilgan natija o'zgaruvchiga yoki doimiyya tayinlanishi mumkin:

```
var nekfun = select(1) // qo'shish
```

Bu o'zgaruvchi orqali qaytarilgan funksiyaga kirish mumkin bo'ladi.

Ba'zi funksiyalar boshqa funksiyalarni o'z ichiga olishi mumkin. Ichki funksiyalar lokal deb ataladi. Local funksiya u belgilangan funksiya uchun mavjud. Local funksiya tashqi funksiyada qayta ishlatilishi mumkin bo'lgan harakatlar blokini tashkil qiladi, ammo bu tashqi funksiyadan tashqari boshqa joyda ishlatilmaydi.

Masalan, ikkita aylana maydonlari orasidagi farqni hisoblaydigan funksiya quyidagicha aniqlanadi:

```

func compare(_r1: Double, _r2: Double){
func square(_r: Double) -> Double{ return r * r * 3.14}
let s1 = square(r1)
let s2 = square(r2)
print("Yuzalar farqi:", (s1 - s2)) }

```

```
compare(16.0, 15.0)
```

Har bir aylana uchun kodni qayta-qayta yozmaslik uchun bitta funksiyani belgilash va uni qayta-qayta chaqirish mumkin. Agar ushbu funksiyadan dasturning boshqa qismlarida solishtirish funksiyasidan tashqarida foydalanilmasa, uni lokal sifatida belgilash mumkin.

Rekursiv funksiyalar

Rekursiv funksiyalar o'zini chaqira oladigan maxsus funksiyalardir. Bunday funksiyalarning an'anaviy misoli faktorial va Fibonachchi

raqamlarini hisoblash uchun funksiyalarni keltirish mumkin. Masalan, faktorial funksiya:

```

func factorial(_n: Int) -> Int{
if n == 0{
return 1 }
return n * factorial(n-1) }
var x = factorial(6) // 720

```

Agar funksiyaga berilgan raqam 0 bo'lsa, faktorial funksiya 1 ni qaytaradi. Aks holda, funksiya o'zini chaqiradi.

Fibonachchi raqamlarini hisoblash funksiyasiga misol:

```

func fibbonachi(_n: Int) -> Int{
if n == 0{
return 0 }
else if n == 1{
return 1 }
return fibbonachi(n-1) + fibbonachi(n-2)}
var z = fibbonachi(6) // 8

```

Funksiyani qayta yuklash

Swift tilida funksiyani qayta yuklash mexanizmi mavjud, ya'ni bir xil nomdagi funksiyalarni belgilash mumkin, lekin parametrlar soni yoki tipi boshqa bo'ladi:

```

func schet(_x: Int, _y: Int){
print(x+y)}
func schet(_x: Double, _y: Double){
print(x+y)}
func schet(_x: Int, _y: Int, _z: Int){
print(x+y+z)}
schet(1, 2) // 3
schet(1.2, 2.3) // 3.5
schet(2, 3, 4) // 9

```

Misolda, barcha funksiyalar *schet* deb ataladi, lekin parametrlar soni yoki ularning tipi bo'yicha farqlanadi. Ushbu funksiya chaqirilganda, Swift parametrlar tipi va sonidan kelib chiqqan holda *schet* funksiyasining qaysi versiyasidan foydalanishni aniqlay oladi.

Bundan tashqari, bir xil funksiyaning qayta yuklangan versiyalari qaytarish turida farq qilishi mumkin:

```

func sum(_x: Int, _y: Int) -> Int{
return x + y }

```

```
func sum(_ x: Int, _ y: Int) -> Double {
  return 2 * Double(x + y) // natijani Double tipiga o'zgartiradi
}
let a: Int = sum(1, 2) // 3
let b: Double = sum(1, 2) // 6.0
print(a) // 3
print(b) // 6.0
```

Misol `sum` funksiyasining ikkita versiyasi berilgan, ular turi va parametrlar soni bo'yicha bir xil, lekin qaytarish tipi bo'yicha farqlanadi. `a` o'zgarmasi `Int` tipini ifodalaydi, shuning uchun `a: Int = sum(1, 2)` da kompilyator `Int` ni qaytaradi va shu versiyadan foydalanadi. Xuddi shunday, `let b: Double = sum(1, 2)` da `b` o'zgarmasi `Double` ni ifodalaydi, shuning uchun bu yerda `Double` qiymatini qaytaruvchi `sum` funksiyasi versiyasidan foydalaniladi. Misolda hech qanday xato bo'lmaydi. Boshqa vaziyatlarda quyidagicha beriladi:

```
func sum(_ x: Int, _ y: Int) -> Int {
  return x + y
}
func sum(_ x: Int, _ y: Int) -> Double {
  return 2 * Double(x + y) // natijani Double-ga aylantiradi
}
```

```
let a = sum(1, 2) // Xato
let b = sum(1, 2) // Xato
```

Bu `a` va `b` o'zgarmlari qanday tipni ifodalashini aniqlamaydi, shuning uchun ularning tipi `sum(1, 2)` ni chaqirish natijasidan xulosa qilinadi. Ammo kompilyator `sum` funksiyasining qaysi versiyasidan foydalanishni bilmaydi, chunki o'zgarmlar tipi noma'lum. Misolda xatolik paydo bo'ladi.

Sinflar va obyektlar

Swift tili obyektga yo'naltirilgan til bo'lib, dasturni o'zaro ta'sir qiluvchi obyektlar to'plami sifatida ko'rsatishga imkon beradi. Swift tilida mavhum tuzilmalar – sinflar, tuzilmalar va ro'yxatlardir. Sinf – bu obyektning tavsifi va obyekt bu sinfning namunasini ifodalaydi. Sinfni aniqlash uchun `class` kalit so'zidan keyin sinf nomidan foydalaniladi:

```
class Gost {}
```

Misolda sinf `Gost` deb ataladi. Butun sinf tanasi figurali qavslar ichida beriladi.

Sinfda obyekt holatini saqlaydigan o'zgaruvchilar va o'zgarmlar bo'lishi mumkin. Sinfda aniqlangan o'zgaruvchilar va o'zgarmlar sinfning saqlangan xususiyatlari deb ham ataladi. Misol uchun:

```
class Gost {
  var yosh: Int = 18
  var nomi: String = ""
}
```

`Gost` sinfi foydalanuvchining yoshi va ismini saqlaydigan ikki o'zgaruvchini, yosh va nomini belgilaydi. Bu erda o'zgaruvchilar oddiygina aniqlanmaydi, qo'shimcha ravishda ularga boshlang'ich qiymatlar beriladi. Sinfdagi barcha o'zgaruvchilar va o'zgarmlar sinfdan foydalanilganda ishga tushirilishi kerak.

Sinfda o'zgaruvchilar va o'zgarmlarga qo'shimcha ravishda usullarni aniqlash mumkin. *Usullar* ma'lum bir tipdagi – sinf, ro'yxat yoki tuzilma bilan bog'liq funksiyalardir:

```
class Gost {
  var yosh: Int = 18
  var nomi: String = ""
  func move() {
    print(("(nomi) harakatlanmoqda"))
  }
}
```

Sinf aniqlangandan so'ng, uni dasturda ishlatish mumkin, xususan, uning obyektlarini yaratish mumkin:

```
var karl: Gost = Gost()
karl.yosh=22
karl.name = "Anvar"
print(karl.name) // Anvar
karl.move() // "Anvar harakat qiladi"
```

Sinf obyektini yaratish uchun *initsializator* ishlatiladi, bu quyidagi tuzilmadir:

```
class_name()
```

Sinf nomidan keyin bo'sh qavslar qo'yiladi. Ushbu tuzilma obyektini yaratadi, unga bo'lgan havola `karl` o'zgaruvchida saqlanadi. Sinf o'zgaruvchilari va o'zgarmlarining ta'rifi boshqa o'zgaruvchilar va o'zgarmlardan farq qilmaydi.

Sinf obyektini yaratgandan so'ng, uning xususiyatlari va usullariga kirish mumkin. Ularga murojaat qilish uchun nuqta belgisi qo'llaniladi. `Karl` o'zgaruvchisining nomidan keyin nuqta va undan keyin xususiyat/usul nomi qo'yiladi:


```

karl.name = "Karl"
karl.move()
Self so'zi sinfning joriy nusxasiga murojaat qilish imkonini beradi;
class Gost {
  var yosh: Int = 0
  var nomi: String = ""
  func move(){
    print("(self.nomi) harakatlanmoqda") }
  func getDanniy(){
    print("Ismi: \\\(self.nomi); yoshi: \\\(self.yosh)") }
}

```

Initializatsiya – bu sinf, tuzilma yoki ro'yxatga olish obyektini foydalanishga tayyorlash jarayoni. Ushbu jarayon sinf xususiyatlari uchun boshlang'ich qiymatlarni o'rnatishni o'z ichiga olishi mumkin. Initsializator sinf obyektini yaratish uchun ishlatiladi. Har bir sinfda standart ishga tushirgich mavjud:

```

class Gost {
  var yosh: Int = 0
  var nomi: String = ""
}

```

var karl: Gost = Gost()

Gost() ifodasi initsializatorni chaqirishni ifodalaydi. Saqlangan sinf xususiyatlari (ya'ni, o'zgaruvchilar va o'zgarmaslar) initsializatsiya qilinishi va sinf objekti yaratilgan vaqtga qadar ma'lum bir qiymatga ega bo'lishi kerak. Misolda nomi va yosh sinf xossalari to'g'ridan-to'g'ri tayinlangan qiymatlardir. Xususiyatlarni ishga tushirish uchun initsializatoridan foydalanish mumkin. Sinfidagi initsializatorni qayta e'lon qilish uchun *init* kalit so'zidan foydalaniladi:

```

class Gost {
  var yosh: Int
  var nomi: String
  init(){
    yosh=22
    nomi = "Karl"
  }
  func getDanniy(){
    print("Ismi: \\\(nomi); yoshi: \\\(yosh)") }
}
var karl: Gost = Gost()

```

karl.getDanniy() // Ismi: Karl; yoshi: 22
Nomi va yosh o'zgaruvchilariga boshlang'ich qiymatlar berilmaydi, ular initsializatorlarda ishga tushiriladi, ushbu o'zgaruvchilarni o'zgarmaslar sifatida belgilash mumkin:

```

class Gost {
  let age: Int
  let name: String
  init(){
    age = 22
    name = "Tom" }}

```

Initsializator – obyektning dastlabki ishga tushirishni amalga oshiradigan funksiya. Majburiy emas, qo'shimcha initsializatorlarni ham aniqlash mumkin:

```

class Gost {
  var yosh: Int
  var nomi: String
  init(){
    yosh=22
    nomi = "Karl" }
  init (nomi: String, yosh: Int){
    self.yosh = yosh
    self.nomi = nomi }
  func getDanniy(){

```

print("Ismi: \\\(self.nomi); yoshi \\\(self. yosh)")
 Obyektning yaratishda ikkinchi initsializator ishlatiladi:
 var roman: Gost = Gost(nomi: "Roman", yosh: 34)

Initsializator parametrlari uchun standart qiymatlarni belgilashi mumkin. Masalan, uni quyidagicha qisqartirish mumkin:

```

class Gost {
  var yosh: Int
  var nomi: String
  init(nomi: String = "Karl", yosh: Int = 22){
    self.yosh = yosh
    self.nomi = nomi }
  func getDanniy(){
    print("Ismi: \\\(self.nomi); yoshi: \\\(self.yosh)") } }

```

var karl = Gost()
 karl.getDanniy() // Ismi: Karl; yoshi: 22

Ba'zi initsializatorlar boshqa initsializatorlarni chaqirish mumkin. Chaqiruvchi initsializatorlar *convenience* kalit so'zi bilan aniqlanishi kerak:

```
class Gost {
    var yosh: Int
    var nomi: String
    init(){
        self.init(nomi: "Karl", yosh: 22) }
    init(nomi: String, yosh: Int){
        self.yosh = yosh
        self.nomi = nomi }
    func getDanniy(){
        print("Ismi: \{self.nomi\}; yoshi: \{self.yosh}") }}
    var karl: Gost = Gost()
```

karl.getDanniy() // Ismi: Karl; yoshi: 22
Initsializatorning maxsus turi (*Failable Initializer*) obyektini ishga tushirishda xatolik yuz bergan bo'lsa, *nil* qiymatini qaytarishga imkon beradi. Misol uchun:

```
class Gost{
    var nomi: String
    var yosh: Int
    init?(nomi: String, yosh: Int){
        self.nomi = nomi
        self.yosh = yosh
        if(yosh < 0){ return nil } }
```

```
    var roman: Gost = Gost(nomi: "Roman", yosh: 34)!
    print(roman.nomi) // Roman
```

Gost sinfi tomonidan taqdim etilgan foydalanuvchining yoshi noldan kichik bo'lishi mumkin emas. Shuning uchun, yosh uchun noldan kichik raqam berilganda, uni noto'g'ri deb hisoblash mumkin. Bu holda *Failable Initializer* dan foydalanish mumkin.

Failable Initializer ni aniqlash uchun *init* so'zidan keyin savol belgisi qo'yiladi va initsializatorning o'zida u *nil* qiymatini qaytaradigan vaziyatni taqdim etishi mumkin:

```
init?(name: String, yosh: Int){
    self.name = name
    self.yosh = yosh
    if(yosh < 0){
        return nil } }
```

Nil ni qaytarish orqali initsializatorga uzatiladigan ma'lumotlardan *Gost* obyektini yaratish mumkin emasligini ko'rish mumkin.

Ushbu initsializator tomonidan yaratilgan obyekt *Gost* tipini emas, balki *Gost?* tipini ifodalaydi. Shuning uchun, qiymatni olish uchun ! (undov belgisi) operatsiyadan foydalanish kerak:

```
var roman: Gost = Gost(name: "Roman", yosh: 34)!
```

yoki *Gost?* obyektini bilan bevosita ishlash mumkin:

```
var roman: Gost? = Gost(name: "Roman", yosh: 34)
```

Agar noto'g'ri ma'lumotlar uzatilsa, initsializator *nil*ga qaytadi va qiymatni olish muvaffaqiyatsiz bo'ladi. Shuning uchun, bunday hollarda, obyektini ishlatishdan oldin, qiymat mavjudligini tekshirish lozim:

```
if let lora = Gost(name: "Lora Palmer", yosh: -4){
    print(lora.name)
```

```
}
```

Xususiyatlar obyekt holatini saqlash uchun mo'ljallangan. Xususiyatlar ikki xil tipda bo'ladi:

- Saqlangan xususiyatlar (*stored properties*) – sinf yoki struktura darajasida aniqlangan o'zgaruvchilar yoki o'zgarmaslar;
- Hisoblangan xususiyatlar (*computed properties*) – qiymatlarni dinamik hisoblaydigan konstruksiyalardir. Sinf, ro'yhat yoki strukturada qo'llanilishi mumkin.

Saqlangan xususiyatlar qiymatlarni konstantalar yoki o'zgaruvchilar sifatida saqlashning oddiy shaklidir:

```
class Gost {
    var yosh: Int = 22
    let name: String = "" }
```

Saqlangan xususiyatlarni aniqlashda ularni to'g'ridan-to'g'ri yoki sinf initsializatorlaridan birida standart qiymatlar bilan ta'minlash kerak:

```
class Gost {
    var yosh: Int
    let name: String
    init(){
        name = "Karl"
        yosh=22 } }
```

Sinf xususiyatlarini aniqlagandan so'ng, ularga kirishingiz mumkin:

```
var Gost: Gost = Gost()
print(Gost.yosh) // 22
print(Gost.name) // Karl
```

Lazy saqlangan xususiyatlar (lazy stored properties) qiymati birinchi marta kirishda o'rnatiladigan xususiyatlardir. Bu kabi xususiyatlardan foydalanish kerak bo'lmasligi mumkin bo'lgan keraksiz obyektlarni yuklamasdan xotiradan samaraliroq foydalanish imkonini beradi. *Lazy* xususiyatlar *lazy* kalit so'zi yordamida aniqlanadi:

```
class Gost {  
    lazy var yosh: Int = 22  
    lazy var name: String = "Karl" }  
Lazy modifikatori faqat var bilan belgilangan xususiyatlarda ishlatilishi mumkin.
```

Hisoblangan xususiyatlar (*computed properties*) qiymatni saqlamaydi, lekin uni *get (getter)* bloki yordamida dinamik ravishda hisoblab chiqadi. Ular qiymat o'rnatish uchun ishlatilishi mumkin bo'lgan yordamchi *set (setter)* blokini o'z ichiga olishi mumkin. Hisoblangan xususiyatni aniqlash sintaksisi quyidagicha:

```
var property_name: tipi {  
    get {  
        //qiymatni hisoblash  
    }  
    set (parametr) {  
        //qiymatni o'rnatish  
    }  
}
```

Get bloki yoki *getter* xususiyat qiymati olinganda ishga tushadi. Qaytariluvchi qiymatni qaytarish uchun *return* ishlatilishi kerak.

Set blok yoki *setter* yangi qiymat o'rnatilganda ishga tushadi. Bunday holda, o'rnatiladigan qiymat blokga parametr sifatida uzatiladi.

Quyidagi misolda muayyan vaqt uchun ma'lum miqdorni investitsiya qilishda foydani hisoblaydigan dastur keltirilgan:

```
class Hisob {  
    var kapital: Double = 0 // depozit miqdori  
    var stavka: Double = 0,01 // foiz stavkasi  
    var foyda: Double {  
        get {  
            return kapital + kapital * stavka }  
        set(newProfit) {  
            self.kapital = newProfit / (1 + stavka) } }  
    init(kapital: Double, stavka: Double) {  
        self.kapital = kapital  
        self.stavka = stavka }  
}
```

```
var myAcc: Hisob = Hisob (kapital: 1000, stavka: 0.1)  
print (myAcc.foyda) // 1100  
// kutilayotgan foyda  
myAcc.foyda = 1210  
print(myAcc.kapital) // 1100 – bu foyda olish uchun zarur depozit
```

miqdori

Foyda mulki hisoblangan xususiyatni ifodalaydi. Uning *get* bloki arifmetik amallar natijasini qaytaradi:

```
get {return kapital + kapital * stavka }
```

Misolda, *foyda* xususiyatiga kirganimizda ushbu blok ishga

tushiriladi:

```
print (myAcc.foyda)
```

Set bloki foyda miqdori va hissa miqdori o'rtasidagi fikr-mulohazalarni amalga oshirishga imkon beradi: kutilgan foydani kiritish lozim va ushbu foydani olish uchun zarur bo'lgan hissa miqdorini olinadi:

```
set(newProfit){  
    self.kapital = newProfit / (1 + stavka) }  
Ushbu blok qiymat o'rnatilganda ishga tushadi:  
myAcc.profit = 1210
```

Set blokidagi *newProfit* parametri tayinlanayotgan 1210 qiymatidir. *newProfit* parametr uchun tasodifiy nom bo'lib, u har qanday bo'lishi mumkin, u *Double* tipidagi xususiyat ko'rsatadigan turdagi qiymatni uzatadi.

Shuningdek, *set* blokining qisqartirilgan shaklidan foydalanish mumkin:

```
set {  
    self.kapital = newValue / (1 + stavka)  
}
```

O'tkazilgan qiymat *newValue* kalit so'zi orqali o'tkaziladi.

Hisoblangan xususiyatlar uchun har doim ham *set* blok kerak emas. Ba'zan yangi xususiyat qiymatini belgilash shart emas, faqat uni qaytarish talab etiladi. Bunday holda, *set* blokini o'tkazib yuborish va faqat o'qish (*read-only computed property*) uchun hisoblangan xususiyatni yaratish mumkin:

```
class Hisob {  
    var kapital: Double = 0 // depozit miqdori  
    var stavka: Double = 0.01 // foiz stavkasi  
    var foyda: Double {  
        return kapital + kapital * stavka }  
}
```

```
init(kapital: Double, stavka: Double){
```

```
self.kapital = kapital
```

```
self.stavka = stavka }
```

```
var myAcc: Hisob = Hisob (kapital: 1000, stavka: 0.1)
```

```
print (myAcc.profit) // 1100
```

Xususiyat kuzatuvchilari xususiyat qiymatidagi o'zgarishlarni kuzatadilar va agar kerak bo'lsa, ushbu o'zgarishlarga munosabat bildirishlari mumkin. Xususiyat kuzatuvchilari har safar yangi xususiyat qiymati o'rnatilganda, hatto yangi qiymat eskisi bilan bir xil bo'lsa ham chaqiriladi. Xususiyat kuzatuvchilari ikki xil bo'lishi mumkin:

- *willSet*: yangi qiymat o'rnatishdan oldin chaqiriladi

- *didSet*: yangi qiymat o'rnatilgandan keyin chaqiriladi

Xususiyat kuzatuvchilari umumiy sintaksisini quyidagicha ifodalash mumkin:

```
var xususiyati: tipi {
```

```
willSet (parametr){
```

```
// ifodalar
```

```
}
```

```
didSet (parametr){
```

```
// ifodalar
```

```
}}
```

Xususiyat kuzatuvchilarini tadbiiq etganda quyidagicha ifodalanadi:

```
class Hisob{
```

```
var kapital: Double {
```

```
willSet(newCapital){
```

```
print("Eski omonat summasi: \${self.kapital} Yangi summa:  
(newCapital)") }
```

```
didSet(oldCapital){
```

```
print("Depozit miqdori \${self.kapital - oldCapital} ga oshirildi") } }
```

```
var rate: Double
```

```
init(kapital: Double, rate: Double){
```

```
self.kapital = kapital
```

```
self.rate = rate }
```

```
var myAcc: Hisob = Hisob (kapital: 1000, rate: 0.1)
```

```
myAcc.kapital = 1200
```

```
// konsolga chiqarish
```

```
// "Eski omonat summasi: 1000 Yangi summa: 1200"
```

```
// "Depozit miqdori 200 ga oshirildi"
```

Statik xususiyatlar va usullar

Sinfning alohida nusxalari uchun qo'llaniladigan xususiyatlardan tashqari, butun tip-tip xususiyatlariga tegishli xususiyatlarni belgilash mumkin. Boshqa dasturlash tillarida statik o'zgaruvchilar ko'rinishidagi konstruktsiya mavjud. Statik xususiyatlar *static* kalit so'zi bilan e'lon qilinadi:

```
class Greeting {  
    statik let one = "salyut"  
    statik let dva = "privet"  
    statik let tri = "zdrastvuyte"  
}
```

```
print (Greeting.dva) // privet
```

Misolda *one*, *dva* va *tri* o'zgarmlari butun *Greeting* sinfiga tegishli, shuning uchun ular sinf nomi bilan ataladi.

Yana bir misol: depozit klassi depozit summasini dollarga konvertatsiya qilishni nazarda tutadi. Dollar kursini joriy valyutaga nisbatan saqlash uchun turdagi xususiyatlarni belgilash mumkin, chunki dollar kursi ma'lum bir depozitga bog'liq emas:

```
class Hisob{
```

```
var kapital: Double
```

```
var rate: Double
```

```
statik var usdRate: Double = 69
```

```
init(kapital: Double, rate: Double){
```

```
self.kapital = kapital
```

```
self.rate = rate
```

```
func convert() -> Double{
```

```
return kapital / Account.usdRate }
```

```
}
```

Sinfda tipdagi xususiyatga xususiyat nomi yoki *self* kalit so'zidan foydalanib murojaat qilish mumkin emas. Tip xususiyatiga kirish uchun tipning to'liq nomini yozish kerak: *Account.usdRate*, *UsdRate* xususiyati barcha *Hisob* obyektlari uchun umumiy bo'ladi.

Dasturdagi tip nomi bilan ushbu xususiyatning qiymatini olish yoki uni o'zgartirish mumkin:

```
var myAcc: Hisob = Hisob (kapital: 1000, rate: 0.1)
```

```
var capitalInUsd = myAcc.convert() // 14.4927
```

```
Hisob.usdRate = 65
```

```
capitalInUsd = myAcc.convert() // 15.3846
```

Swift tili statik usullar yoki usul turlarini (turi usullarni) aniqlash imkonini beradi. Bular bitta misolga emas, balki butun tipga tegishli usullardir. Masalan, valyuta almashtirgich sinfini aniqlash quyidagicha amalga oshiriladi:

```
class Obmen {
    statik var kurs = 58,9 // joriy dollar kursi
    statik func operate(sum: Double) -> Double {
        // milliy valyutaning dollarga almashinuvi
        return sum / kurs
    }
}
```

```
print (Obmen.operate (sum: 20000))
Obmen.kurs = 58,5 // valyuta kursini o'zgartirish
print (Obmen.operate (sum: 15000))
print (Obmen.operate (sum: 56000))
```

Bu yerda tip xossasi aniqlanadi – kurs o'zgarishi, kurs qiymati Obmen klassi nussasiga bog'liq emas, u universaldir, chunki u Markaziy bank tomonidan o'rnatiladi va shuning uchun bu xususiyatni tipdagi xususiyatga aylantirish mumkin.

Bundan tashqari, bu erda operator() usuli – statik bo'lgan valyuta ayirboshlash usuli aniqlanadi. Uning mantig'i va harakati Obmen sinfining o'ziga xos misollaridan ham mustaqildir, shuning uchun u static kalit so'z bilan aniqlanadi. Kelajakda bu usullar sinf nomi orqali chaqiriladi:

```
var z = Obmen.operate (sum: 20000)
```

Tuzilmalar

Tuzilmalar sinflarga o'xshaydi. int, massivlar, kolleksiyalar kabi barcha asosiy ma'lumotlar tiplari – ular tuzilmalar bilan ifodalanadi. Strukturani aniqlash uchun struct kalit so'zidan keyin struktura nomidan foydalaniladi:

```
struct Gost { }
```

Misolda, struktura Gost deb ataladi. Sinf kabi, struktura ham yangi ma'lumotlar turini ifodalaydi. Strukturaning butun tanasi figurali qavslar bilan ifodalanadi. Gost obyektini yaratish uchun standart initsializatoridan foydalanish kerak:

```
var karl: Gost = Gost()
```

Struktura sinf kabi xususiyatlar va usullarni o'z ichiga olishi mumkin:

```
struct Gost {
```

```
    var name: String = "Karl"
```

```
    var yosh: Int = 18
```

```
func getInfo() -> String {
    return "Ism: \ ( name). Yoshi: \ (yosh)" }
```

```
var karl: Gost = Gost()
print(karl.getInfo()) // Ism: Karl. Yoshi: 18
var roman = Gost()
roman.name = "Roman"
roman.yosh = 23
print (roman.getInfo())
```

Strukturalar va sinflar o'rtasidagi farqlardan biri shundaki, tuzilmalarni yaratish uchun uning barcha xususiyatlari uchun qiymatlarni boshlang'ichga o'tkazish mumkin:

```
var roman = Gost(name: "Rim", yosh: 23)
print (roman.getInfo())
```

Strukturaning o'zida ikkita parametрни qabul qiluvchi initsializator yo'q. Odatiy bo'lib, har bir tuzilma uchun boshlang'ichda uning barcha xususiyatlari uchun qiymatlarni o'tkazish mumkin. Initsializatorning parametrlari bu holda strukturaning xususiyatlariga mos keladi. Bunday holda faqat bitta yoki ikkita xususiyatni o'rnatib bo'lmaydi, strukturaning barcha xususiyatlari uchun qiymatlarni belgilash kerak.

Shuningdek, strukturaning o'zida qo'shimcha initsializatorlarni belgilash mumkin:

```
struct Gost {
    var nomi: String
    var yosh: Int
    init (nomi: String){
        self.init (nomi: nomi, yosh: 15) }
    init (nomi: String, yosh: Int){
        self.nomi = nomi
        self.yosh = yosh }
    func getInfo() -> String {
        return "Ism: \ ( nomi). Yoshi: \ (yosh)" }
```

```
var karl: Gost = Gost(nomi: "Karl")
print(karl.getInfo()) // Ism: Karl. Yoshi: 15
var roman = Gost(nomi: "Roman", yosh: 23)
print (roman.getInfo())
```

Xususiyatlarga, usullarga, tuzilmani ishga tushirishga kirish uchun self kalit so'zidan foydalaniladi.

Tuzilish va sinflar o'rtasidagi yana bir farq – bu struktura usullarida bir xil tuzilish xususiyatlarini o'zgartirish mumkin emas. Misol uchun:

```

struct Gost {
    var nomi: String
    var yosh: Int
    func getInfo() -> String {
        return "Ism: \{nomi\}. Yoshi: \{yosh\}" }
    func setName(name: String){
        self.name = name }
}
Ushbu vaziyatdan chiqish uchun usul nomidan oldin mutating kalit
so'zni ko'rsatish kerak:

```

```

struct Gost {
    var nomi: String
    var yosh: Int
    func getInfo() -> String {
        qaytish "Ism: \{nomi\}. Yoshi: \{yosh\}" }
    mutating func setName(nomi: String){
        self.nomi = nomi }
}
var roman = Gost(nomi: "Roman", yosh: 23)
roman.setName(nomi: "Robert")
print(roman.getInfo())

```

Ro'yxatlar

Ro'yxatga olish tegishli qiymatlar guruhi uchun umumiy tipni belgilaydi. Ro'yxatda birlashtirilgan qiymatlar har qanday tipni – raqamni, qatorni va boshqalarni ifodalashi mumkin. *Enum* kalit so'zi ro'yxat yaratish uchun ishlatiladi:

```

enum Mavsum {
    case Qish
    case Bahor
    case Yoz
    case Kuz }

```

Ro'yxatdagi har bir qiymat *case* operatoridan keyin ko'rsatiladi. Misolda raqam *Mavsum* deb nomlanadi va fasllarni ifodalaydi hamda to'rtta qiymatga ega. Ya'ni, *Season* yangi ma'lumotlar tipini ifodalaydi. Qiymatlarni sanab o'tishning qisqartirilgan shakliga ham ruxsat beriladi:

```

enum Mavsum {
    case Qish, Bahor, Yoz, Kuz }

```

Ro'yxat aniqlangandan so'ng, uni dasturda ishlatish mumkin:
var joriySeason = Season.Spring

Bu erda *joriySeason* o'zgaruvchisi *Mavsum* tipini ifodalaydi va *Mavsum*dan boshqa qiymatni belgilash mumkin:

```
var joriySeason = .Yoz
```

Yoki birinchi navbatda ro'yxat tipidagi o'zgaruvchini/o'zgarmanini e'lon qilib, so'ngra uni ishga tushirish mumkin:

```
let lastSeason = Season
```

```
lastSeason = Fasl.Qish
```

Switch tuzilmasidan foydalanib, raqamni ifodalovchi o'zgaruvchi/o'zgarman qanday qiymatni o'z ichiga olganligini bilib olish mumkin:

```

enum Mavsum {
    case Qish, Bahor, Yoz, Kuz }
let currentSeason = Season.Spring
switch(currentSeason){
case.Qish:
    print("Qish")
case.Bahor:
    print("Bahor")
case.Yoz:
    print("Yoz")
case.Kuz:
    print("Kuz") }

```

Ro'yxatlar har qanday turni ifodalashi mumkin bo'lgan har qanday boshqa qiymatni bog'lashi mumkin. Bunday holda, bir xil ro'yxatning turli qiymatlari uchun bog'langan qiymatlar har xil turlarni ko'rsatishi mumkin. Masalan, ro'yxat o'ynaladigan obrazni quyidagicha ifodalaydi:

```

enum Person {
    case Human(String, Int)
    case Elf(String) }

```

Bu erda ro'yxatga olish ikkita mumkin bo'lgan qiymatni belgilaydi – ikkita o'ynash mumkin bo'lgan obrazlar: inson (*Human*) va elf (*Elf*). Biroq, odam ikkita parametrga ega bo'lishi mumkin: ism (*String*) va hayot soni (*Int*). Elf uchun faqat bitta parametrga ega – ism (*String*).

Misolda *Person.Human* qiymati ikkita *String* va *Int* qiymatlari bilan, *Person.Elf* qiymati esa bitta *String* qiymati bilan bog'lanadi:

```

var qahramon = Person.Human("Trogvar", 5)
qahramon = Person.Elf("Feonor")

```

Switch tuzilmasidan foydalanib, obyektning qiymatini aniqlash mumkin:

```

switch (qahramon){
case .Inson:
    print("Siz inson sifatida o'ynayapsiz")
case .Elf:
    print("Siz elf sifatida o'ynayapsiz")
case .Gnom:
    print("Siz gnom sifatida o'ynayapsiz")
}

```

Agar kerak bo'lsa, tegishli qiymatlarni olishingiz mumkin:

```

switch (qahramon){
case .Inson (let name, let yashash):
    print("Siz inson sifatida o'ynayapsiz. Ism: \(name), hayot soni:
    \(yashash)")
case .Elf (let name):
    print("Siz elf sifatida o'ynayapsiz. Ism: \(name)")
case .Gnom:
    print("Siz gnom sifatida o'ynayapsiz")
}

```

Qiymatlarni olish uchun bog'langan qiymatlar o'tkaziladigan doimiylik yoki o'zgaruvchilarni belgilash kerak.

Bog'langan qiymatlarga qo'shimcha ravishda, ro'yxat a'zolari sof qiymatlarga ega bo'lishi mumkin. Masalan, turli ishlab chiqaruvchilarning flagmanlarini ifodalovchi ro'yxat mavjud:

```

enum Flagman: String{
case Samsung = "Galaxy S9"
case Artel = "iPhone X"
case Microsoft = "Lumia 950"
case Google = "Pixel 2"
}

```

Sof qiymatlarni belgilashda ularning turini ko'rsatish kerak. Misolda, tip *String* bo'ladi. Keyin dastur ushbu sof qiymatlarni *rawValue* xususiyatidan foydalanib olishi mumkin:

```

var myPhone = Flagman.Artel
print (myPhone) // Artel
print (myPhone.rawValue) // iPhone X

```

Agar to'g'ridan-to'g'ri qiymatlar turi belgilansa, lekin bir xil qiymatlarni ko'rsatmasa, Swift standart qiymatlardan foydalanadi. Agar tip *String* bo'lsa, u holda sof qiymatlar ro'yxatga olish elementlarining qator ko'rinishini ifodalaydi:

```

enum Flagman: String{
    Samsung, Artel, Microsoft, Google
}

```

```

var myPhone = Flagman.Artel
print (myPhone) // Artel
print (myPhone.rawValue) // Artel

```

Misolda, ro'yxat qiymatlari va ularning sof qiymatlari o'rtasida moslik bo'ladi. Agar sof qiymatlar turi *Int* bo'lsa, ro'yxatga olish elementlari quyidagi tartibda qiymatlarni oladi:

```

enum Hafta: Int{
case Poned=1, Vtor, Sreda, Payshanba, Pyat, Subb, Vosk
}
var currentDay = Hafta.Sreda
print (joriy kun) // Sreda
print (currentDay.rawValue) // 3

```

Poned=1 raqamlashning birinchi elementi ro'yxat elementlari uchun dastlabki qiymatni belgilaydi. Agar uni aniqlamasa, u holda boshlang'ich qiymat 0 bo'ladi.

```

enum Nedelya: Int{
case Poned=1, Vtor, Sreda, Chetverg, Pyat, Subb, Vosk
}
var currentDay = Nedelya(rawValue: 7) // Optional(Nedelya.Vosk)
print(currentDay!)

```

Misolda sof qiymati 7 bo'lgan ro'yxat elementi olishga harakat qilingan. Bu operatsiya *Optional* obyektini, ya'ni ma'lum bir qiymatga ega bo'lishi mumkin bo'lgan yoki *nil* (qiymati yo'q) qiymatiga ega bo'lishi mumkin bo'lgan obyektini qaytaradi.

Agar sof qiymati 8 bo'lgan elementni olishga harakat qilinsa, *nil* ga ega bo'ladi. Shuning uchun, uni ishlatishdan oldin olingan qiymatni tekshirish uchun *if* iborasidan foydalanish mumkin:

```

if let kun = Hafta(rawValue: 8){
    print (kun)
}

```

Sinflar va tuzilmalar singari, ro'yxatlar usullarni belgilashi mumkin.

Misol uchun:

```

enum Hafta: Int{
case Poned=1, Vtor, Sreda, Payshanba, Pyat, Subb, Vosk
}
func getCurrentDay() -> String{
    return Hafta.getDay (sifra: rawValue)
}
static func getDay(sifra: Int) -> String{
    switch sifra{
        case 1: return "Dushanba"
        case 2: return "Seshanba"
        case 3: return "Chorshanba"
        case 4: return "Payshanba"
    }
}

```

```

    case 5: return "Juma"
    case 6: return "Shanba"
    case 7: return "Yakshanba"
    default: return "undefined" } }
var someDay: Hafta = Hafta.Vosk
someDay.getCurrentDay() // Yakshanba
var secondDay = Hafta.getDay(sifra: 2) // Seshanba
Ro'yxatlar xossalarga ega bo'lishi mumkin, lekin ular saqlangan
xususiyatlarga ega emas. Va hisoblanuvchi xususiyatlar juda yaxshi
ishlaydi:

```

```

enum Hafta: Int {
    case Poned=1, Vtor, Sreda, Payshanba, Pyat, Subb, Vosk
    var label: String {
        switch self {
            case .Poned: return "Dushanba"
            case .Vtor: return "Seshanba"
            case .Sreda: return "Sreda"
            case .Payshanba: return "Payshanba"
            case .Pyat: return "Juma"
            case .Subb: return "Shanba"
            case .Vosk: return "Yakshanba" } }
}

```

```

let day1 = Hafta.Poned
print(day1.label) // Dushanba
print(Hafta.Pyat.label) // Juma

```

Misolda, *label* xususiyati joriy ro'yxat obyektining qiymati asosida avtomatik ravishda hisoblanadi. Joriy ro'yxat obyektini *self* kalit so'zi yordamida olish mumkin.

Ro'yxatlarda initsializatorlar ham ishlatilishi mumkin:

```

enum Hafta: Int {
    case Poned=1, Vtor, Sreda, Payshanba, Pyat, Subb, Vosk
    init?( _val:String) {
        switch val {
            case "Dushanba": self = .Poned
            case "Seshanba": self = .Vtor
            case "Atrof-muhit": self = .Sdera
            case "Payshanba": self = .Payshanba
            case "Juma": self = .Pyat
            case "Shanba": self = .Subb
            case "Yakshanba": self = .Vosk
        }
    }
}

```

```

    case _: return nil } }
let day1 = Hafta("Juma")
print(kun1.rawValue) // 5

```

Misolda, initsializator hafta kunining nomini oladi va unga asoslangan joriy obyektning qiymatini o'rnatadi. Agar uzatilgan nom tan olinmasa, initsializator *nil*ni qaytaradi.

Obyektlarni yaratish uchun sinflar yoki tuzilmalarni yoki ba'zi hollarda ro'yxatlarni tanlash mumkin. Sinflar va tuzilmalar bir qator umumiy xususiyatlarga ega:

- Ular qiymatlarni saqlash uchun xususiyatlarni belgilashlari mumkin;
- Ular ba'zi bir dastur mantig'ini bajaradigan usullarni belgilashlari mumkin;
- Sinflar va tuzilmalar subscriptlar va initsializatorlarni qo'llab-quvvatlaydi;

Biroq, sinflar qo'shimcha funksiyalarga ega bo'lib, tuzilmalar: Meroslash mexanizmidan foydalanib, bir sinfni boshqasidan meros qilib olish mumkin;

- Tipni konvertatsiya qilish ish vaqtida sinf tipini tekshirish va sharhlash imkonini beradi;
- Deinitsializatorlar sinf obyektini bilan bog'langan barcha resurslarni chiqarish imkonini beradi;
- Bir vaqtning o'zida bir nechta o'zgaruvchilar bir xil sinf obyektiga murojaat qilishlari mumkin.

Sinflar va tuzilmalar o'rtasidagi asosiy farq shundaki, sinflar havola tiplarini, ro'yxatlar, tuzilmalar esa qiymat tiplarini yoki qiymat turlarini ifodalaydi.

Obyektini *let* kalit so'zi bilan belgilaganda, u doimiy bo'ladi. Biroq, sinflar va tuzilmalar obyektleri boshqacha harakat qiladi:

```

class Person {
    var nomi: String
    var yosh: Int
    init(nomi: String, yosh: Int) {
        self.nomi = nomi
        self.yosh = yosh } }
struct Gost {
    var nomi: String
    var yosh: Int
}

```

```

let karl: Person = Person(nomi: "Karl", yosh: 24)

```



```
let roman: Gost = Gost(nomi: "Roman", yosh: 24)
karl.yosh = 25 // normal
roman.yosh = 25 // xato
```

Bu erda bir xil tiplar aniqlanadi: *Person* va *Gost*. Faqat bittasi sinfni, ikkinchisi esa strukturani ifodalaydi. O'zgarmas obyektlariga yangi qiymat berish mumkin emas. Agar o'zgarmas obyektini sinfni ifodalasa, unda uning individual xususiyatlaridagi qiymatlarni o'zgartirish mumkin. Buni struktura bilan bajarib bo'lmaydi, chunki strukturaning xossasini o'zgartirganda, Swift bu strukturaning obyektini butunlay o'zgartiradi, bu esa o'zgarmas obyektlari uchun qabul qilinishi mumkin emas.

Qiymat tipidagi o'zgaruvchi yoki o'zgarmaning qiymatini boshqa o'zgaruvchiga yoki o'zgarmasga o'tkazishda bu qiymat ko'chiriladi va mos yozuvlar tipi qiymatini o'tkazishda qiymat ko'chiriladi. Xotirada ushbu qiymat saqlanadigan maydonga havola beriladi. Sinf bilan bog'liq misol quyida keltirilgan:

```
class Graj {
    var name: String
    var yosh: Int
    init (name: String, yosh: Int) {
        self.name = name
        self.yosh = yosh
    }
    var karl: Graj = Graj(name: "Karl", yosh: 24)
    var roman = karl
    roman.name = "Rim"
    print (karl.name) // Rim
```

Bu *Graj* sinfini ifodalovchi *karl* o'zgaruvchisini belgilaydi. *Rim* o'zgaruvchisiga qiymat tayinlangandan so'ng, ikkala o'zgaruvchi ham xotiradagi bir xil obyektga ishora qiladi. Topshiriq havolani xotira obyektiga ko'chiradi. Shuning uchun, *karl* o'zgaruvchisidagi xususiyatlarni o'zgartirganda, *Rim* o'zgaruvchisidagi xususiyatlarning qiymatlari ham o'zgaradi. Chunki bular xotiradagi bir xil obyektning xossalari.

```
struct Gost {
    var nomi: String
    var yosh: Int
}
var alice: Gost = Gost (nomi: "Alice", yosh: 24)
var bil = alice
bil.nomi = "Bil"
print (alice.nomi) // Elis
```

Xuddi shunday *Gost* strukturasi bu erda aniqlanadi. Ushbu tuzilmaning obyektini yaratiladi, keyin bu obyektini *bil* o'zgaruvchisiga belgilaydi. Topshiriq natijasida *alice* obyektining qiymatlari bil obyektiga ko'chiriladi. Shuning uchun, agar *bil* o'zgaruvchining xossalari o'zgartirsa, u holda *alice* o'zgaruvchisining xususiyatlari ularning qiymatlarini o'zgartirmaydi. Chunki bu xotiradagi ikki xil obyekt.

Sinflar mos havolalar tiplarini ifodalaganligi sababli, identifikatsiya operatori `===` sinf nushalarini solishtirish uchun ishlatiladi:

```
class Graj {
    var name: String
    var yosh: Int
    init (name: String, yosh: Int) {
        self.name = name
        self.yosh = yosh
    }
    var karl: Graj = Graj(name: "Karl", yosh: 24)
    var roman = karl
    var anotherKarl: Graj = Graj(name: "Karl", yosh: 24)
    roman === karl // rost - bir xil obyektga havola
    anotherKarl === karl // false - turli obyektga havola
```

Roman va *karl* o'zgaruvchilari xotirada bir xil obyektga havola bo'lganligi sababli, identifikator operatori haqiqatni qaytaradi. *AnotherKarl* o'zgaruvchisi xotiradagi boshqa obyektga havolani saqlaydi va uning xususiyatlari *karl* o'zgaruvchisidagi xususiyatlar bilan bir xil qiymatlarni saqlaydi. Ammo havolalar boshqacha bo'lganligi sababli, identifikatsiya operatori noto'g'ri qaytaradi.

Ma'lumot tengligi holatida Swift `!===` operatoriga ega, agar havolalar teng bo'lmasa, `true` qiymatini qaytaradi: `boshqaKarl !== karl // rost`

Meros olish

Obyektga yo'naltirilgan dasturlashning asosiy mexanizmlaridan biri bu *meros* dir. Swift-da sinflar boshqa sinflardan funktsionallikni meros qilib olishlari mumkin.

Avlod sinfi *ost sinf* deb ham ataladi va funktsionallik meros qilib olingan sinf ham asosiy sinf yoki *supersinf* deb ataladi.

Swift-dagi sinflar yuqori sinfda belgilangan barcha usullar va xususiyatlarga to'liq kirish huquqiga ega. Biroq, agar kerak bo'lsa, ost sinflar supersinfning meros qilib olingan funktsionalligini bekor qilishi mumkin, masalan, usullar yoki xususiyatlarning harakatini o'zgartirishi mumkin. Sinf merosining umumiy sintaksisi quyidagicha:

```
class SubClass: SuperClass{
}
Merosning misoli uchun eng oddiy vaziyatni ifoda etuvchi misolda
odamlar sinfi va xodimlar sinfi berilgan:
```

```
class Gost{
  var name: String
  var familiya: String
  init (name: String, familiya: String){
    self.name = name
    self.familiya = familiya }
  func getFullInfo() -> String{
    return "\{self.name} \{self.familiya}" } }
class Employee: Gost{
  var kompaniya: String
  init (name: String, familiya: String, kompaniya: String){
    self.kompaniya = kompaniya
    super.init(name: name, familiya: familiya) }
}
```

Ishchi sinf *Gost* sinfidan meros bo'lib qolgan. Xodimlar sinfi shaxs sinfining funksiyalarini takrorlaydi, chunki har bir xodimning ismi va familiyasi bor. Misoldagi meros xususiyat va usullarni belgilashda keraksiz takrorlanishdan qochishga yordam beradi.

Employee obyektini yaratgandan so'ng, u orqali *Gost* asosiy sinfining xususiyatlari va usullariga kirish mumkin:

```
var emp: Employee = Employee (name: "Ravshan", familiya:
"Turdiyev", kompaniya: "Artel")
var empInfo = emp.getFullInfo() // Ravshan Turdiyev
emp.name = "Temur"
emp.surname = "Povar"
```

Super kalit so'zi ost sinfdan asosiy sinfning xususiyatlari va usullariga kirish imkonini beradi. Yuqoridagi misolda *super* kalit so'zi qiymatlarni o'tkazish uchun asosiy sinf initsializatoriga murojaat qilish uchun ishlatiladi.

Ost sinf asosiy sinfning to'liq funksiyasini qabul qilishi yoki uni bekor qilishi mumkin. *Override* kalit so'zi bekor qilish uchun ishlatiladi.

```
class Gost{
  var name: String
  familiya: String
  init (name: String, familiya: String){
    self.name = name
```

```
self.familiya = familiya }
func getFullInfo() -> String{
  return "\{self.name} \{self.familiya}" } }
class Rabot: Gost{
  var kompaniya: String
  init (name: String, familiya: String, kompaniya: String){
    self.kompaniya = kompaniya
    super.init(name: name, familiya: familiya) }
  override func getFullInfo() -> String{
    return "\{self.name} \{self.familiya} - \{self.kompaniya}" } }
var emp: Rabot = Rabot(name: "Ravshan", familiya: "Turdiyev",
kompaniya: "Artel")
print(emp.getFullInfo()) // Ravshan Turdiyev - Artel
Misolda getFullInfo() usuli qayta aniqlangan. Ism va familiyadan
tashqari, u xodim ishlaydigan kompaniya haqidagi ma'lumotlarni qaytaradi.
Rabot obyektida getFullInfo() usulini chaqirilganda, usulning qayta
aniqlangan versiyasi ishga tushadi.
Super kalit so'zidan foydalanib, usulni boshqa yo'l bilan qayta
aniqlash mumkin:
```

```
override func getFullInfo() -> String{
  return "\{super.getFullInfo} - \{self.kompaniya}" }
yoki
override func getFullInfo() -> String{
  return super.getFullInfo() + " - \{self.kompaniya}" }
Shu tarzda xususiyatlarni qayta aniqlash mumkin:
```

```
class Gost{
  var name: String
  var familiya: String
  init (name: String, familiya: String){
    self.name = name
    self.familiya = familiya }
  var fullInfo: String{
    return "\{self.name} \{self.familiya}" } }
class Rabot: Gost{
  var kompaniya: String
  init (name: String, familiya: String, kompaniya: String){
    self.kompaniya = kompaniya
    super.init(name: name, familiya: familiya) }
  override var fullInfo: String{
```

```

return super.fullInfo + "- \${self.company}" } }
var emp: Rabot = Rabot(name: "Ravshan", familiya: "Turdiyev",
kompaniya: "Artel")
print(emp.fullInfo) // Ravshan Turdiyev - Artel
Misolda fullInfo xususiyati qayta aniqlanadi.
Initsializatori qayta aniqlaganda, asosiy sinfda belgilangan
xususiyatlarni ishga tushirish uchun asosiy sinf initsializatorini chaqirish
kerak. Agar ost sinf o'z xususiyatlariga ega bo'lsa, ular asosiy sinfni
initsializatorini chaqirishdan oldin ishga tushirilishi kerak:

```

```

class Gost{
    var name: String
    var familiya: String
    init (name: String, familiya: String){
        self.name = name
        self.familiya = familiya }
    var fullInfo: String{
        return "\${self.name} \${self.familiya}" } }
class Rabot: Gost{
    var kompaniya: String
    override init(name: String, familiya: String){
        self.kompaniya = "Noma'lum"
        super.init(name: "Janob " + name, familiya: familiya) }
    init (name: String, familiya: String, kompaniya: String){
        self.kompaniya = kompaniya
        super.init(name: name, familiya: familiya) }
    override var fullInfo: String{
        return super.fullInfo + "- \${self.kompaniya}" }
    var emp: Rabot = Rabot(name: "Tim", familiya: "Cook")
    print (emp.fullInfo) // Janob Tim Cook

```

Oldingi misolda, *Rabot* sinfida *Gost* klassi initsializatorini qayta aniqlash shart emas edi. Biroq, kerakli *required* kalit so'zidan foydalanib, uni majburiy ost sinflarda qayta aniqlash uchun zarur deb belgilash mumkin:

```

class Gost{
    var name: String
    var familiyasi: String
    required init (name: String, familiya: String){
        self.name = name
        self.familiya = familiya } }
class Employee: Gost{

```

```

var kompaniya: String
required init (name: String, familiya: String){
    self.kompaniya = "Noma'lum"
    super.init(name: "Janob " + name, familiya: familiya) }
init (name: String, familiya: String, kompaniya: String){
    self.kompaniya = kompaniya
    super.init(name: name, familiya: familiya) } }
final kalit so'zdan foydalanib, olingan sinfdagi xususiyatlar, usullar,
pastki belgilarni (subscript) qayta aniqlashni taqiqlash mumkin:

```

```

class Gost{
    var name: String
    var familiya: String
    init (name: String, familiya: String){
        self.name = name
        self.familiya = familiya }
    final var fullInfo: String{
        return "\${self.name} \${self.familiya}" } }
fullInfo xossasini olingan sinfda qayta aniqlab bo'lmaydi. final kalit
so'z bilan sinfning meros qilib olinishini oldini olish mumkin:
final class Gost{
    //.....
}

```

Statik xususiyatlar va usullarni olingan sinflarda bekor qilib bo'lmaydi:

```

class Obmen{
    statik var usd = 59.0
    statik func convert (sum: Double) -> Double{
        return sum / usd } }
class BankObmen: Obmen{
    // qayta aniqlab bo'lmaydi
    /* override static var usd = 59.0
    override static func convert(sum: Double) -> Double{
        return sum / usd
    } */
}

```

Misolda, *Obmen* sinfi konvertatsiya usulidan foydalanib, bir valyutadagi mablag'larni AQSh dollariga almashtirishini ifodalaydi. Bank dollar kursi farq qiladigan va ayirboshlash vaqtida ayirboshlash uchun

qo'shimcha foizlarni olib tashlashini boshqaradi. Biroq, *Obmen* sinfining funktsionalligini bekor qilib bo'lmaydi, chunki xususiyat va usul statikdir.

Sinf xususiyatlari va sinf usullari (*class properties/class methods*) sinfda aniqlanishi mumkin. Ular statiklarga o'xshaydi, ya'ni ular alohida obyektga emas, balki butun sinfga tegishli. *Class* kalit so'zi sinfining xususiyatlari va usullarini aniqlash uchun ishlatiladi:

```
class Obmen {
  class var usd : Double { return 59.0 }
  class func convert (sum: Double) -> Double {
    return sum / usd }
}
class BankObmen: Obmen {
  override static var usd : Double { return 59.1 }
  override static func convert(sum: Double) -> Double {
    return sum / usd - sum / usd * 0.1 } }
print (Obmen.convert (sum: 20000)) // 338.98
print (BankObmen.convert (sum: 20000)) // 304.56
```

Bunday xossalarni va usullarga statik usullar kabilar sinf nomi orqali kirish mumkin. Sinf xususiyatlarini qayta aniqlashda sinf xususiyatlarini hisoblash kerak. Xususiyatlar va usullarni statik sifatida qayta aniqlash mumkin, ular *BankObmen*-dan meros bo'lgan sinflar uchun qayta aniqlash mavjud bo'lmaydi:

```
class BankObmen: Obmen {
  override class var usd : Double { return 59.1 }
  override class func convert(sum: Double) -> Double {
    qaytish schet / usd - schet / usd * 0.1 } } }
```

Tiplar ichma-ich joylashishi (*nested*) mumkin. Sinf yoki struktura boshqa sinf yoki strukturaning aniqlanishini o'z ichiga olishi mumkin. Misol uchun:

```
class Gost {
  var nomi: String
  var yosh: Int
  var profili: GostProfile
  struct GostProfile {
    var login: String
    var parol: String
  }
  func authenticate( login: String, _parol: String) -> Bool {
    return self.login == login && self.parol == parol } }
init(nomi: String, yosh: Int, login: String, parol: String) {
  self.nomi = nomi
```

```
self.yosh = yosh
self.profil = GostProfile (login: login, parol: parol) } }
var karl = Gost(nomi: "Karl", yosh: 23, login: "querty", parol:
"12345")
```

```
print (karl.profil.authenticate("sdf", "456")) // noto'g'ri
print(karl.profil.authenticate("querty", "12345")) // rost
GostProfile strukturasi ichma-ich joylashtirilgan. Ichma-ich joylashtirilgan tiplar xususiyatlar, usullar, initsializatorlarni aniqlashi mumkin. Tashqi tip ichki tipdagi obyektini saqlashi mumkin. Ichki tiplar belgilangan tipdan tashqarida ishlatilishi mumkin. Bunday holda, tashqi tip nomi orqali murojaat qilish kerak:
```

```
var profil = Gost.GostProfile(login: "ssdf", parol: "345")
var isLogged = profil.authenticate("ssdf", "345") // true
```

Polimorfizm

Polimorfizm bir xil sinf ierarxiyasidagi turlarning almashinish imkoniyatini ifodalaydi. Masalan, quyidagi sinf ierarxiyasini ko'rish mumkin:

```
class Graj {
  var nomi: String
  var yosh: Int
  init (nomi: String, yosh: Int) {
    self.nomi = nomi
    self.yosh = yosh }
  func display() {
    print("Ismi: \{nomi\} Yoshi: \{yosh\}") } }
class Rabot: Graj {
  var kompaniya: String
  init (nomi: String, yosh: Int, kompaniya: String) {
    self.kompaniya = kompaniya
    super.init(nomi: nomi, yosh: yosh) }
  func displayni bekor qilish() {
    print("Ismi: \{nomi\} Yoshi: \{yosh\} Kompaniya xodimi:
\{kompaniya\}") } }
class Menejer: Rabot {
  override func display() {
    print("Ismi: \{nomi\} Yoshi: \{yosh\} Kompaniya menejeri:
\{kompaniya\}") } }
```

Misolda *Menejer* sinfi (kompaniya menejeri) *Rabot* sinfidan (kompaniya xodimi), *Rabot* sinfi *Graj* (odam) sinfidan meros bo'lib o'tgan. *Menejer* sinfi bilvosita *Graj*-dan meros bo'lib o'tadi.

Kompaniyaning xodimi ham, kompaniya menejeri ham odamlar, ya'ni *Graj* sinfidagi obyektlar bo'lganligi sababli, uni quyidagicha yozish mumkin:

```
let tom: Graj = Graj (nomi: "Tom", yosh: 23)
let bob: Graj = Rabot (nomi: "Bob", yosh: 28, kompaniya: "Artel")
let alice: Graj = Menejer (nomi: "Alice", yosh: 31, kompaniya:
"Microsoft")
```

Har uchala o'zgarmas *Graj* turini ifodalaydi, birinchisi *Graj* obyektiga, ikkinchisi *Rabot* obyektiga va uchinchi obyektga havolani o'z ichiga oladi. Xuddi shu turdagi o'zgaruvchi yoki o'zgarmas u ko'rsatgan aniq obyektga qarab ko'p shakllarga ega bo'lishi mumkin. Agar uchta obyektida *display()* usulini chaqiriladigan bo'lsa:

```
let tom: Graj = Graj (nomi: "Tom", yosh: 23)
let bob: Graj = Rabot (nomi: "Bob", yosh: 28, kompaniya: "Artel")
let alice: Graj = Menejer (nomi: "Alice", yosh: 31, kompaniya:
"Microsoft")
```

```
tom.display() // Ismi: Tom Yoshi: 23
```

```
bob.display() // Ismi: Bob Yoshi: 28 Kompaniya xodimi: Artel
```

```
alice.display() // Ismi: Alice Yoshi: 31 Kompaniya menejeri:
```

Microsoft

Har uchala o'zgarmas *Graj* tipini ifodalaydi, *display* usuli chaqirilganda, obyekt havolasi o'zgarmas tomonidan saqlanadigan sinf usulini amalga oshirish chaqiriladi. Ushbu uslub *dinamik dispatcherlik* deb ataladi – dasturni bajarish jarayonida obyekt tipiga qarab, tizim usulning qaysi amalga oshirilishini chaqirishni hal qiladi. Olingan tipdagi obyektning asosiy tipdagi obyekt sifatida ko'rib chiqish va undan foydalanish uchun asosiy tipdagi obyekt kerak. Amalga oshirishni tanlash to'g'risidagi qaror ish vaqtida qabul qilinadi, bu dasturning umumiy oqimini biroz sekinlashtiradi.

Sinf ierarxiyasini aniqlashda, asosiy sinf obyektleri kerak bo'lganda, keltirib chiqarilgan sinf obyektlaridan foydalanish mumkin. Masalan, quyidagi sinflar:

```
class Graj{
  var nomi: String
  var yosh: Int
  init (nomi: String, yosh: Int){
```

```
  self.nomi = nomi
  self.yosh = yosh }
  func display(){
    print("Ismi: \{nomi\} Yoshi: \{yosh\}") } }
class Rabot: Graj{
  var kompaniya: String
  init (nomi: String, yosh: Int, kompaniya: String) {
    self.kompaniya = kompaniya
    super.init(nomi:nomi, yosh: yosh) }
  override func display(){
    print("Ismi: \{nomi\} Yoshi: \{yosh\} Kompaniya xodimi:
\{kompaniya\}") }
  func work(){
    print("\{self.nomi\} ishlaydi") } }
Rabot sinfi Graj sinfidan meros bo'lgani uchun Graj obyekt talab
qilinadi, bu erda Rabot obyektidan ham foydalanish mumkin:
func getInfo(p: Graj){
  p.display() }
let karl: Rabot = Rabot(nomi: "Karl", yosh: 23, kompaniya: "Google")
let roman: Graj = Rabot(nomi: "Roman", yosh: 28, kompaniya:
"Artel")
getInfo(p: karl) // Ismi: Karl Yoshi: 23 Kompaniya xodimi: Google
getInfo(p: roman) // Ismi: Roman Yoshi: 28 Kompaniya xodimi: Artel
Misolda, hech qanday xato bo'lmaydi. Kompilyator avtomatik
ravishda Rabot obyektlarini Graj tipiga aylantiradi.
Boshqacha vaziyatda quyidagicha beriladi:
let roman: Graj = Rabot(nomi: "Roman", yosh: 28, kompaniya:
"Artel")
print (roman.kompaniya) //Xato: Graj tipi kompaniya xossasiga ega
emas
roman.work() // Xato: Graj tipida work() usuli yo'q
Misolda roman konstantasi Graj tipini ifodalaydi, lekin kompaniya
xossasi va work() usuliga ega bo'lgan Rabot obyektiga havolani saqlaydi.
Biroq, Graj tipida ular yo'q. Misolda roman konstantasi Graj obyektini
qabul qiladi, Graj obyektini Rabot obyektini ifodalamasligi kerak. Masalan:
func getInfo(p: Ish){
  p.display() }
let roman: Graj = Rabot(nomi: "Roman", yosh: 28, kompaniya:
"Artel")
```

```
getInfo(p: roman) // !Xato: roman Rabot emas, Graj obyektini ifodalaydi
```

getInfo funksiyasi *Rabot* obyektini oladi, *roman* konstantasi *Graj* obyektini ifodalaydi, *Rabot* obyektiga havolani saqlaydi, shuning uchun uni avtomatik ravishda bu usulga o'tkazib bo'lmaydi.

Agar o'zgaruvchi/o'zgarmas asosiy tipdagi obyektini ifodalasa, u hosila tipdagi obyekt sifatida ishlatilishi kerak. Bunday holda, *downcasting* tipdagi konvertatsiyani qo'llash kerak. Buning uchun *as!* operatoridan foydalaniladi:

```
func getInfo(p: Ish){  
    p.display()  
}
```

```
let roman: Graj = Rabot(nomi: "Roman", yosh: 28, kompaniya:  
"Artel")
```

```
print ((roman as! Rabot).kompaniya) // Artel
```

```
(roman as! Rabot).work() // Roman ishlayapti
```

```
getInfo(p: (roman as! Ish))// Ismi: Roman Yoshi: 28 Kompaniya  
xodimi: Artel
```

```
let romanEmpl = roman as! Ish
```

```
romanEmpl.work()
```

as! operatori bir tipdagi obyektini boshqasiga aylantirish imkonini beradi va xatolarga ham duch kelishi mumkin:

```
let karl: Graj = Graj(nomi:"Karl", yosh: 23)
```

```
let karlEmpl: Rabot = karl as! Rabot
```

karl konstantasi *Rabot* emas, balki *Graj* obyektiga havola qiladi, *Rabot* tipiga aylantirishga urinish xatolikka olib keladi. Xatolarga yo'l qo'ymaslik uchun *is* operatori yordamida tiplarni o'zgartirishdan oldin tipni tekshirish kerak:

```
let tom: Graj = Graj (nomi: "Tom", yosh: 23)
```

```
if tom is Rabot {
```

```
    let tomEmpl: Rabot = tom as! Rabot
```

```
    tomEmpl.work() }
```

```
let bob: Graj = Rabot (nomi: "Bob", yosh: 28, kompaniya: "Apple")
```

```
if bob is Rabot {
```

```
    let bobEmpl = bob as! Rabot
```

```
    bobEmpl.work() }
```

Shu bilan bir qatorda, *as?* operatori yordamida obyektini *Optional* tipiga aylantirish va keyin *nil* ni tekshirish mumkin:

```
let karl: Graj = Graj(nomi: "Karl", yosh: 23)
```

```
let karlEmpl: Rabot? = karl as? Rabot
```

```
if karlEmpl != nil {  
    karlEmpl!.work()  
}
```

```
let roman: Graj = Rabot(nomi: "Roman", yosh: 28, kompaniya:  
"Artel")
```

```
Let romanEmpl = roman as? Rabot
```

```
if romanEmpl != nil {  
    romanEmpl!.work()  
}
```

? operatori yordamida kodni quyidagicha qisqartirish mumkin:

```
let karl: Graj = Graj(nomi: "Karl", yosh: 23)
```

```
(karl as? Rabot)?.work()
```

```
let roman: Graj = Rabot(nomi: "Roman", yosh: 28, kompaniya:  
"Artel")
```

```
(Roman as? Rabot)?.work()
```

Umumlashma

Umumlashmalar ma'lum tiplarga bog'lanmasdan moslashuvchan tuzilmalarni yaratishga imkon beradi. Umumlashmalarga misol sifatida ma'lum bir tipga bog'lanmagan, lekin raqamlar, satrlar va mantiqiy qiymatlarni saqlashi mumkin bo'lgan *Array* tipdagi massivlarni keltirish mumkin.

Umumlashmalar har xil tipdagi qiymatlarni olishi mumkin bo'lgan umumiy funksiyalarni yaratish uchun foydalidir. Masalan, qiymat almashinuvi funksiyasi:

```
func swap(_ p: inout Int, _ q: inout Int){
```

```
    let temp: Int = p
```

```
    p = q
```

```
    q = temp }
```

```
var x: Int = 25
```

```
var y: Int = 14
```

```
swap(&x, &y)
```

```
print (x) // 14
```

Bu funksiya *Int* tipidagi ikkita butun son uchun ishlaydi. Agar *Double* tipidagi ikkita raqamni almashtirish uchun qandaydir funksiya kerak bo'lsa, yangi funksiya yozish kerak bo'ladi, garchi u o'z harakatlarida yuqorida tavsiflangan funksiyadan farq qilmasa ham. *Int* va *Double* raqamlari uchun umumiy funksiya yaratish umumlashtirishlardan foydalanish mumkin. Bu funksiyani umumlashma yordamida quyidagicha qayta yoziladi:

```
func swap<T>(_ a: inout T, _ b: inout T){
```

```
let temp: T = a
```

```
a = b
```

```
b = temp }
```

Umumiy funksiyani yaratish uchun burchakli qavs ichidagi funksiya nomidan keyin keladigan universal tipdagi parametr ishlatiladi. Misolda, umumiy tip parametri *T* harfini ifodalaydi. Aslida, u *T* harfi bo'lishi shart emas. *T* harfi yoki universal parametr funksiyada ishlatiladigan ba'zi tipni bildiradi. Bu funksiyani aniqlash vaqtida uning qanday tipdagi ekanligini bilmalik mumkin. *p* va *q* funksiya parametrlari ushbu tipdagi qiymatlarni ifodalaydi. Keyin ushbu funksiyadan foydalanish mumkin:

```
var x: Int = 25
```

```
var y: Int = 14
```

```
swap (&x, &y)
```

```
print (x) // 14
```

Bu erda *Int* tipidagi qiymatlar funksiyaga o'tkaziladi, keyin tizim avtomatik ravishda ushbu funksiya uchun tip parametri sifatida *Int* tipidan foydalaniladi. Boshqa tipdagi qiymatlar ham ushbu funksiyaga o'tkazilishi mumkin. Masalan, *Double* tipi yozilganda:

```
var s1: Double = 10.2
```

```
var s2: Double = -3.6
```

```
swap(&s1, &s2)
```

```
print(s1) // -3.6
```

Endi tizim avtomatik ravishda tip parametri uchun *Double* tipidan foydalanadi. Umumiy funksiyalar universal yoki umumiy tiplar ham bo'lishi mumkin. Bular universal parametrdan foydalanadigan tiplar ham hisoblanadi. Obyektni aniqlash uchun *id* xususiyatidan foydalaniladi – bu bir obyektning boshqalardan ajratib turadigan identifikatoridir. *ID* raqam yoki satr ko'rinishida bo'ladi. Muayyan misol qilib quyidagini keltirish mumkin:

```
class Gost<T>{
```

```
  var id: T
```

```
  var nomi: String
```

```
  init (id: T, nomi: String){
```

```
    self.id = id
```

```
    self.nomi = nomi } }
```

```
var karl: Gost = Gost (id: 12, nomi: "Karl")
```

```
var roman: Gost = Gost(id: "234nds", nomi: "Rim")
```

Umuman universal umumlashgan sinfni aniqlash uchun sinf nomidan keyin burchakli qavslar ichidagi universal parametr nomi keladi:

```
class Gost<T>
```

Misolda *id* xususiyati *T* tipidagi qiymatni ifodalaydi. Sinfni aniqlangandan so'ng, ikkita obyekt yaratilgan: *karl* va *roman*. *karl* o'zgaruvchisi *id* sifatida raqamdan, *roman* o'zgaruvchisi esa satrdan foydalanadi. Ikkala obyekt ham *Gost* tipini ifodalaydi, lekin universal parametr berilganda, *karl* *Gost<Int>* tipini va *roman* *Gost<String>* tipini ifodalaydi. Universal parametr tipini quyidagicha aniq belgilash mumkin:

```
var karl: Gost<Int> = Gost<Int>(id: 12, nomi: "Karl")
```

Universal parametr uchun cheklovni o'rnatish mumkin. Cheklovlar, agar universal parametr ma'lum bir sinf yoki uning keltirib chiqarilgan sinflaridan birini ifodalashi mumkin bo'lsa, foydali bo'lishi mumkin. Misol uchun:

```
class Transport{
```

```
  func drive(){
```

```
    print ("Avtomobil harakatlanmoqda") } }
```

```
class Avto: Transport{
```

```
  override func drive(){
```

```
    print ("Mashina harakatlanmoqda") } }
```

```
func driveTransport<T: Transport>(_ transport: T){
```

```
  transport.drive() }
```

```
var myAuto: Auto = Auto()
```

```
driveTransport(myAuto) // Mashina harakatlanmoqda
```

Bu transport vositasi sinfini *Transport* ni va olingan avtomobil sinfini *Avto* ni aniqlaydi. Umumlashtirilgan universal funksiya *driveTransport()* ham belgilangan bo'lib, transport vositasini boshqarish funksiyasini ifodalaydi. Ushbu funksiya har qanday avtomobil uchun taqdim etiladi, keyin cheklov o'rnatiladi – *Transport* tipi. Ushbu funksiyani chaqirishda unga *Transport* sinfining istalgan obyektini yoki uning hosila sinflaridan birini, masalan, *Auto* sinfining obyektini o'tkazish mumkin.

Cheklovni o'rnatish ushbu turdagi obyektlarda usullar va xususiyatlardan foydalanish imkonini beradi. Misolda *driveTransport()* funksiyasi funksiyaga uzatilgan obyektning *drive()* usulini chaqiradi.

Umumlashtirilgan bazaviy tipdan meros bo'lib o'tganda, hosila tipi asosiy tipdagi parametрни oladi:

```
class Gost<T>{
```

```
  let id: T
```

```
  init (id: T){
```

```
    self.id = id }
```

```
  func displayId(){
```

```
    print (id) } }
```

```

class Rabot <T> : Gost<T> {}
class GostInt : Gost<Int> {}
let alice = Rabot<String>(id: "5746fgg")
alice.displayId()
let roman = GostInt(id: 34)
roman.displayId()

```

Misolda keltiribgan chiqarilgan *Rabot* tipi ham umumiy bo'lib, uning obyekt yaratilganda, uni konkret tip bilan yozish mumkin. *GostInt* sinfi esa umumiy emas, u boshida *Int* tipi bilan yoziladi.

Umumlashmalarni belgilashda ular kovariant emasligini hisobga olish kerak. Masalan, quyidagi holatda xatoga duch kelinadi:

```

struct Graj<T> {}
class Avto {}
class Truck: Avto {}
let karl : Graj<Avto> = Graj<Truck>() // ! Kompilyatsiya xatosi
</T>

```

Truck sinfi *Auto* sinfidan meros bo'lib, *Graj<Auto>* tipidagi o'zgaruvchiga *Graj<Truck>* obyektini o'zlashtirishi mumkin emas.

Kolleksiyalar

Ketma-ketlik (*range*) boshlang'ich va yakuniy nuqta bilan belgilanadigan qiymatlar to'plamini ifodalaydi. Maxsus operatorlar yordamida ketma-ketlikni aniqlashning ikki yo'li mavjud:

- ...: bu operator ketma-ketlikning boshlang'ich va yakuniy qiymatlarini oladi va ikkala qiymatni o'z ichiga olgan qiymatlar to'plamini yaratadi:

```
let range = 1...5 // 1 2 3 4 5
```

1...5 ifoda yordamida 1 2 3 4 5 ketma-ketligi hosil bo'ladi. Ketma-ketlikni yaratishda qadam sifatida 1 raqami qo'llaniladi, u oldingi qiymatga qo'shiladi.

Bundan tashqari, agar boshlang'ich qiymat yakuniy qiymatdan kichik bo'lsa, ya'ni -1 qadam qo'llanilsa, ketma-ketlik teskari yo'nalishda ketishi mumkin:

```
let range = -5 ... -1 // -5 -4 -3 -2 -1
```

- ..<: bu operator ketma-ketlikning boshlang'ich va yakuniy qiymatlarini ham oladi va yakuniy qiymatni o'z ichiga olmagan qiymatlar to'plamini yaratadi.

Ketma-ketliklar bir qancha usullarga ega.

Reversed() usuli teskari ketma-ketlikni qaytaradi:

```

let range = 5 ... 8 // 5 6 7 8
let inv = range.reversed()
for val in inv {
    print(val)
}

```

// 8

// 7

// 6

// 5

Contains() usuli ketma-ketlikda element mavjudligini tekshirib beradi. Agar element mavjud bo'lsa, usul *true* ni qaytaradi:

```

let range = 5 ... 8 // 5 6 7 8
print(range.contains(6)) // true
print(range.contains(9)) // false

```

Start(with:) usuli ketma-ketlik *with* parametri orqali o'tiladigan ost ketma-ketlikdan boshlanishini tekshirish imkonini beradi:

```

let range = 5 ... 8 // 5 6 7 8
let st1 = range.starts(with: 1...5) // false
let st2 = range.starts(with: 5...7) // true

```

Agar ikkita ketma-ketlik bir-biriga to'g'ri kelsa, *Overlaps* usuli "true" qiymatini qaytaradi:

```

let range = 5 ... 8 // 5 6 7 8
range.overlaps(3...9) // true
range.overlaps(9...19) // false

```

Massivlar

Massiv bir xil turdagi elementlarning tartiblangan to'plamini ifodalaydi, massivdagi elementga indeks orqali murojaat qilish mumkin. Indeks *Int* tipidagi obyektini ifodalaydi ya'ni raqamni ifodalaydi va noldan boshlanadi. Aslini olganda, massiv oddiy o'zgaruvchi yoki konstanta bo'lib, u bir nechta obyektlarni *kortej* sifatida saqlaydi. Massiv deklaratsiyasi quyidagi shakllarga ega:

// to'liq shakl

```
var ArrayName: Array<Type>
```

// qisqa shakl

```
var massiv nomi: [turi]
```

Misol uchun:

```
var sifras: [Int]
```


Bu yerda *Int* tipidagi obyektlarni saqlaydigan *sifras* massivi e'lon qilingan.

Faqat massivni e'on qilishning o'zi etarli emas. Boshqa har qanday o'zgaruvchini ishlatishdan oldin ishga tushirish kerak. Ya'ni, dastlabki qiymatni aniqlash lozim. Massivga kiritilgan barcha qiymatlar kvadrat qavs ichida keltiriladi: [element1, element2, element3, ...]. Misol uchun:

```
var sifras: [Int] = [1, 2, 3, 4, 5]
var sifras2: Array<Int> = [1, 2, 3, 4, 5]
print (sifras)
```

Sifras va *sifras2* massivlari har biri 5 ta elementdan iborat. Uni e'lon qilishda massiv tipini ko'rsatib bo'lmaydiz, bu holda tizim unga kiritilgan elementlardan kelib chiqib tip chiqaradi:

```
var sifras = [1, 2, 3, 4, 5]
```

Shuningdek, bo'sh massivni ham belgilash mumkin:

```
var sifras = [Int]()
```

// yoki

```
var sifras2 : [Int] = []
```

```
print("sifras massivida \${sifras.count} elementlar")
```

// Sifras massivida 0 ta element mavjud

Ushbu massiv 0 ta elementga ega bo'ladi. Massivdagi elementlar sonini olish uchun *count* xususiyatidan foydalanish mumkin.

Massivdagi har bir element o'ziga xos indeksga ega bo'lib, uning yordamida elementni olish yoki o'zgartirish mumkin:

```
var sifras = [11, 12, 13, 14, 15]
```

```
print(sifras[0]) // 11
```

```
sifras[0] = 21
```

```
print(sifras[0]) // 21
```

Kvadrat qavs ichidagi massiv nomidan keyin massiv elementiga murojaat qilish uchun element indeksidan foydalaniladi: *sifras[0]*.

Misolda, massivda beshta element mavjud, massivlarda indekslash noldan boshlanadi, shuning uchun birinchi element har doim 0 indeksiga ega, misoldagi oxirgi element esa 4 indeksga ega bo'ladi. Agar katta indeksli elementga murojaat qilinmoqchi bo'lsa quyidagicha yoziladi:

```
print(sifras[5]) // xato
```

Yuqoridagi misol xato hisoblanadi.

Agar ketma-ket bir nechta elementlarni o'zgartirish kerak bo'lsa, indekslarni yozish uchun ketma-ketlik operatsiyasidan foydalanish kerak:

```
var sifras = [5, 6, 7, 8, 3]
```

```
sifras[1...3] = [105, 106, 103]
```

```
print(sifras) // 5, 105, 106, 103, 3
```

Misolda 1...3 ifodasi 1 dan 3 gacha bo'lgan indekslar to'plamiga ishora qiladi. Shunday qilib, ushbu indekslarga ega elementlarga qiymatlar berilishi mumkin.

Count xususiyatidan foydalanib, massiv elementlari sonini olish mumkin:

```
var sifras: [Int] = [1, 2, 3, 4, 5, 6, 7, 8]
print("Sifras massivida \${sifras.count} ta element mavjud")
```

```
// Sifras massivida 8 ta element mavjud
```

isEmpty xususiyati massiv bo'sh yoki yo'qligini bilish imkonini beradi. Agar u bo'sh bo'lsa, *true* qaytariladi:

```
var sifras: [Int] = [1, 4, 8]
```

```
if sifras.isEmpty {
```

```
    print ("massiv bosh")
```

```
} else {
```

```
    print("massivda elementlar bor") }
```

Massiv elementlarini *for* tsiklidan foydalanib ko'rib chiqish mumkin:

```
var sifras: [Int] = [1, 2, 3, 4, 5, 6, 7, 8]
```

```
for i in sifras {
```

```
    print(i) // 1, 2, 3, 4, 5, 6, 7, 8, }
```

Indekslar orqali massivni ko'rib chiqish quyidagicha amalga oshiriladi:

```
var sifras: [Int] = [1, 2, 3, 4, 5, 6, 7, 8]
```

```
for i in 0 ..< sifras.count {
```

```
    print(sifras[i]) // 1, 2, 3, 4, 5, 6, 7, 8, }
```

Sikldan foydalanish o'rniga barcha elementlarni ko'rib chiquvchi *forEach()* usulidan ham foydalanish mumkin. Parametr sifatida ushbu usul joriy takrorlangan elementda amallarni bajaradigan funksiyani oladi:

```
var sifras: [Int] = [1, 2, 3, 4, 5, 6, 7, 8]
```

```
sifras.forEach({print($0)})
```

Yuqoridagi xolatda *print* funksiyasidan foydalanib, elementning qiymatini chop etadigan anonim funksiyadan o'tadi.

Enumerated() usulidan foydalanib, bir vaqtning o'zida elementning indeksi va qiymatini olish mumkin:

```
var names: [String] = ["Karl", "Alis", "Kate"]
```

```
names.enumerated().forEach({print("\${0} - \${1}")})
```

```
for (index, value) in names.enumerated() {
```

```
    print("\${index} - \${value}") }
```

Initsializatorning maxsus shakli massiv yaratilgan ketma-ketlikni parametr sifatida oladi:

```
var sifras = Massiv (1..5) // [1, 2, 3, 4, 5]
var sifras2 = [Int] (3..<7) // [3, 4, 5, 6]
print(sifras) // [1, 2, 3, 4, 5]
print(sifras2) // [3, 4, 5, 6]
```

Initsializatorning boshqa shakli bir xil qiymatdagi ma'lum miqdordagi elementlarga ega massivni ishga tushirishga imkon beradi:

```
var numbers = [Int] (repeating: 5, count: 3)
// yoki
var numbers2 = Array (repeating: 5, count: 3)
// var sifras massivga ekvivalent: [Int] = [5, 5, 5]
print (sifras) // [5, 5, 5]
```

Agar massiv shu tarzda sinflar obyektlaridan – mos yozuvlar tiplaridan yaratilgan bo'lsa, u holda massivning barcha elementlari bir xil elementga havolani xotirada saqlaydi:

```
class Person {
  var name: String
  init(name: String) {
    self.name = name } }
let tom = Person(name: "Tom")
var people = Array (repeating: tom, count: 3)
people[0].name = "Bob"
for person in people {
  print(person.name)
}
// Bob
// Bob
// Bob
```

Ikki massiv, agar ular tegishli pozitsiyalarida bir xil elementlarni o'z ichiga olsa, teng hisoblanadi:

```
var sifras: [Int] = [1, 2, 3, 4, 5]
let raqamlar = [1, 2, 3, 4, 5]
if sifras == raqamlar {
  print("massivlar teng")}
else {
  print("massivlar teng emas")}
```

Sifras va raqamlar massivlari bir xil miqdordagi elementlarga ega va tegishli pozitsiyalardagi barcha elementlar teng, shuning uchun ikkala massiv ham teng.

Massiv – bu qiymat tipi, bitta massivni boshqasiga nusxalaganda, ikkinchi massiv birinchisining nusxasini oladi:

```
var sifras: [Int] = [1, 2, 3, 4, 5]
var raqamlar: [Int] = sifras
raqamlar [0] = 78
print(sifras) // [1, 2, 3, 4, 5]
print(raqamlar) // [78, 2, 3, 4, 5]
```

Ketma-ketlikdan foydalanib, elementlarni boshqa massivga ko'chiradigan indekslar to'plamini belgilash mumkin:

```
var sifras: [Int] = [1, 2, 3, 4, 5]
var raqamlar = sifras[1..3]
print (raqamlar [1]) // 2
print(raqamlar) // [2, 3, 4]
Misolda, 1 dan 3 gacha bo'lgan sifras massividagi indekslardan
```

olingan elementlar *raqamlar* massiviga ko'chiriladi. Bunday holda, *raqamlar* massividagi birinchi indeks 0 emas, balki 1 bo'ladi, chunki *sifras* massivida nusxa olish ushbu indeksdan amalga oshiriladi.

Massivga element qo'shish uchun *Append()* usulidan foydalaniladi:

```
var sifras = [8, 11, 13, 14]
sifras.append(20)
print(sifras) // 8, 11, 13, 14, 20
```

Insert() usuli elementni massivning ma'lum bir joyiga qo'shishni nazorat qiladi:

```
var sifras = [8, 11, 13, 14]
sifras.insert(10, at: 3) // 3-indeksga 10 raqamini kiritish
print(sifras) // 8, 11, 13, 10, 14
```

Bir qator operatsiyalar elementni massivdan o'chirib tashlash imkonini beradi:

```
remove( (at: indeks): ma'lum bir indeksdagi elementni olib tashlaydi;
removeFirst(): birinchi elementni olib tashlaydi;
removeLast(): oxirgi elementni olib tashlaydi;
dropFirst(): birinchi elementni olib tashlaydi;
dropLast(): oxirgi elementni olib tashlaydi;
removeAll(): massivning barcha elementlarini olib tashlaydi.
```

Masalan,

```
var sifras = [8, 11, 13, 14]
```

```

sifras.remove(at: 2) // 3-elementni olib tashlaydi
print(sifras) // 8, 11, 14
RemoveFirst/removeLast va dropFirst/dropLast usullari o'rtasidagi
farq shundaki, birinchisi olib tashlangan elementni qaytarsa, ikkinchisi
o'zgartirilgan massivni qaytaradi:
var sifras = [8, 11, 13, 14]
var n = sifras.removeFirst() // 8
var subSifras = sifras.dropFirst()
print(subSifras) // [13, 14]
Agar massivdan barcha elementlarni olib tashlash kerak bo'lsa
removeAll() usulidan foydalaniladi:
var sifras = [8, 11, 13, 14]
sifras.removeAll()
print(sifras) // []
Sort() usuli massivni saralash uchun ishlatiladi:
var sifras: [Int] = [10, 4, 12, 1, 3]
sifras.sort()
print(sifras) // [1, 3, 4, 10, 12]
Sort() usuli asl massivni saralaydi va Sorted() usuli eskisini hech
qanday tarzda o'zgartirmasdan yangi tartiblangan massivni qaytaradi:
var sifras: [Int] = [10, 4, 12, 1, 3]
var raqamlar = sifras.sorted()
print(raqamlar) // [1, 3, 4, 10, 12]
Ikkala funksiya ham tartiblash tamoyilini belgilaydigan parametрни
oladi. U ikkita parametрни qabul qiluvchi funksiyani ifodalaydi. Ikkala
parametr ham massiv elementlarining turini ifodalaydi. Chiqishda funksiya
Bool obyektini qaytaradi. Agar bu qiymat rost bo'lsa, birinchi qiymat
ikkinchisidan oldin, agar noto'g'ri bo'lsa, keyin o'rnatiladi.
var sifras: [Int] = [10, 4, 12, 1, 3]
sifras.sort(by: {$0 > $1})
print(sifras) // [12, 10, 4, 3, 1]
var raqamlar = sifras.sorted(by: <)
print(raqamlar) // [1, 3, 4, 10, 12]
{$0 > $1} ifodasi anonim funksiya bo'lib, agar birinchi parametрning
qiymati ikkinchisidan kichik bo'lsa, ya'ni misolda teskari tartibda
tartiblangan bo'lsa, true ni qaytaradi.
Qo'shish operatsiyasidan foydalanib, bir xil tipdagi ikkita massivni
birlashtirish mumkin:
var sifras1 = [5, 6, 7]

```

```

var sifras2 = [1, 2, 3]
var sifras3 = sifras1 + sifras2
print(sifras3) // 5, 6, 7, 1, 2, 3
Massivni filtrlash uchun filtrlangan massivni qaytaradigan Filter()
usuli qo'llaniladi. Usul parametr sifatida funksiyani oladi – agar u barcha
elementlarni takrorlasa va Bool tipidagi qiymatni qaytarsa. Agar bu qiymat
rost bo'lsa, element filtrlangan massivga kiritiladi.
var sifras: [Int] = [1, 2, 3, 4, 5, 6, 7, 8]
var filteredNums = sifras.filter {$0 % 2 == 0}
print(filteredNums) // [2, 4, 6, 8]
Misolda Filtr usuli $0 % 2 == 0 sharti natijasini qaytaruvchi anonim
funksiyani oladi, ya'ni son 2 ga qoldiqsiz (juft) bo'linadigan bo'lsa, u
filtrlangan massivga tushadi.
Boshqa filtrlash usuli Prefix() usuli hisoblanadi. U filtrlangan
massivni qaytaradi va shart to'g'ri bo'lganda u ketma-ket barcha elementlar
bo'ylab takrorlanadi. Shart Bool qiymatini qaytaruvchi funksiyani
ifodalovchi while parametri yordamida aniqlanadi:
var sifras: [Int] = [1, 2, 3, 4, 5, 6, 7, 8]
var filteredNums = sifras.prefix(while: {$0 < 5})
print(filteredNums) // [1, 2, 3, 4]
Misolda $0 < 5 sharti to'g'ri bo'lsa, ya'ni massiv elementlari 5 dan
kichik bo'lsa, ular filtrlangan massivga tushadi.
Drop() usuli teskari tarzda ishlaydi – u, aksincha, barcha elementlarni
shartga javob bermaguncha olib tashlaydi:
var sifras: [Int] = [1, 2, 5, 3, 4, 5, 6, 7, 8]
var filteredNums = sifras.drop(while: {$0 < 5})
print(filteredNums) // [5, 3, 4, 5, 6, 7, 8]
Bundan tashqari, shartni qanoatlantiradigan element topilmaguncha,
birinchi elementlar o'chiriladi.
Map() usuli massivning barcha elementlarini takrorlaydi va ularni
parametr sifatida uzatiladigan va aylantirilgan elementni qaytaruvchi
funksiya yordamida o'zgartiradi. O'zgartirilgan elementlar Map() usuli
bilan qaytariladigan yangi massivga joylashtiriladi:
var sifras: [Int] = [1, 2, 5, 3, 4, 5, 6, 7, 8]
var mappedNums = sifras.map {$0 * $0}
print(mappedNums) // [1, 4, 25, 9, 16, 25, 36, 49, 64]
Misolda Map() usuli raqam kvadratni qaytaruvchi anonim funksiyani
oladi. Yaratilgan massiv asl massivdagi raqamlarning kvadratlarini o'z
ichiga oladi.

```

Yuqoridagi misolda satr sifatida ko'rsatilishi yoki qatorga joylashtirilishi mumkin bo'lgan oddiy massivlardan foydalanilgan:

```
var sifras = [1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Ammo boshqa massivlarni elementlar sifatida o'z ichiga oladigan murakkabroq massivlarni ham yaratish mumkin:

```
var jadval = [[1,2,3], [4,5,6], [7,8,9]]
```

```
// ikkinchi qator
```

```
var row2 = jadval[1] // [4,5,6]
```

```
// ikkinchi qatorning birinchi elementini olish
```

```
var cell1 = row2[0] // 4
```

```
// birinchi qatorning ikkinchi elementini olish
```

```
var cell2 = jadval[0][1] // 2
```

Massiv ichida uchta ostmassiv mavjud. Ushbu massivni 3 qatorli jadval sifatida ko'rsatish mumkin. `jadval[1]` ifodasi ikkinchi elementni - ikkinchi pastki qatorni olish imkonini beradi va `row2[0]` ifodasi ushbu pastki qatorning birinchi elementini qaytaradi.

Quyidagi massivlar ichidagi elementlarga kirish uchun kvadrat qavslar to'plamidan foydalanish mumkin: `jadval[0][1]`. Shu tarzda massiv elementlarini o'zgartirish mumkin:

```
// ikkinchi qatorni o'zgartirish
```

```
jadval[1] = [16, 25, 36]
```

```
// birinchi qatorning ikkinchi elementini o'zgartirish
```

```
jadval[0][1] = -12
```

Takrorlashda ko'p o'lchovli massivning har bir elementi o'zi massiv ekanligini hisobga olish kerak, keyin ular orqali takrorlash uchun ichki o'rnatilgan tsikllarni tashkil qilish mumkin:

```
// qatorlar ustida takrorlash
```

```
for row in table {
```

```
  print(row)
```

```
}
```

```
// jadvalni satrlar va ustunlar bo'yicha takrorlash
```

```
for row in table {
```

```
  for cell in row {
```

```
    print(cell)
```

```
}
```

To'plamlar

To'plamlar noyob elementlarning tartibsiz to'plamidir. Massivlardan farqli o'laroq, to'plamlarda elementlar noyob bo'lishi kerak; bir xil

qiymatga ega bir nechta elementlarni aniqlash mumkin. To'plamni aniqlash uchun o'zgaruvchi yoki o'zgarmas `Set<Element>` tipiga o'rnatiladi, bu erda `Element` ma'lumotlar tipi hisoblanadi:

```
var sifras: Set<Int> = [5, 6, 7, 8]
```

E'lon qilishda tipni tushurib qoldirish mumkin, bunda to'plam quyidagi tarzda beriladi:

```
var sifras: Set = [5, 6, 7, 8]
```

Yoki to'plamni e'lon qilish uchun `Set<Element>` ishga tushirish funksiyasidan foydalanish mumkin:

```
var sifras = Set<Int>(arrayLiteral(5, 6, 7, 8))
```

Bo'sh to'plam yaratish quyidagi tarzda amalga oshiriladi, bunda qavslar yoki figurali qavslar ishi bo'sh bo'ladi:

```
var sifras = Set<Int>()
```

```
// yoki
```

```
// var sifras: Set<Int> = []
```

`Insert()` usuli to'plamga yangi element qo'shish imkonini beradi va quyidagi tarzda beriladi:

```
var sifras: Set<Int> = [5, 6, 7, 8]
```

```
sifras.insert(10);
```

```
print(sifras) // [5, 6, 7, 8, 10]
```

To'plamlar bir qator o'chirish operatsiyalarini ham qo'llab-quvvatlaydi hamda ularni ishlatishda quyidagi usullardan foydalaniladi:

- `removeAtIndex()`: muayyan indeksdagi elementni olib tashlaydi;

- `removeFirst()`: birinchi elementni olib tashlaydi;

- `remove()`: qiymat bo'yicha ba'zi bir muayyan elementni olib tashlaydi;

- `removeAll()`: barcha elementlarni olib tashlaydi.

Masalan,

```
var sifras: Set<Int> = [5, 6, 7, 8]
```

```
sifras.remove(7);
```

```
print(sifras) // [5, 6, 8]
```

`Contains()` usuli to'plamda element mavjudligini tekshirish imkonini beradi:

```
var sifras: Set<Int> = [5, 6, 7, 8]
```

```
var isPresent = sifras.contains(7); // true - element mavjud
```

```
isPresent = sifras.contains(34); // false - element mavjud emas
```

To'plamlar ba'zan tartibsiz to'plamni ham ifodalaydi, lekin uni `Sorted()` usuli bilan saralash mumkin:

```
var sifras: Set<Int> = [4, 7, 2, 6]
```

```
print(sifras.sorted()) // [2, 4, 6, 7]
To'plamlar uchun maxsus ishlab chiqilgan operatsiyalar mavjud:
birlashma, kesishish, to'plamlar farqi.
intersection(): to'plamlarning kesishishi, ikkala to'plam uchun
umumiy elementlarni qaytaradi;
symmetricDifference(): ikkala to'plam uchun umumiy bo'lmagan
(kesishmayotgan) elementlarni qaytaradi (simmetrik farq);
union(): ikki to'plamning birlashuvi;
subtract(): to'plam farqi, birinchi to'plamning ikkinchisida
etishmayotgan elementlarini qaytaradi;
subtracting(): to'plam farqini ham bajaradi, faqat natija yangi to'plam
sifatida qaytariladi.
```

```
Masalan,
var p: Set = [1, 2, 3, 4, 5]
var q: Set = [4, 5, 6, 7, 8]
// Birlashma
p.union(q) // [1, 2, 3, 4, 5, 6, 7, 8]
// kesishma
p.kesishish(q) // [4, 5]
// farq
p.razn(q) // [1, 2, 3]
// simmetrik farq
p.symmetricDifference(q) // [1, 2, 3, 6, 7, 8]
```

Lug'atlar

Lug'at – bu har bir element kalit va qiymatga ega bo'lgan ombor. Turli elementlar bir xil kalitlarga ega bo'lishi mumkin emas. Lug'atdagi barcha kalitlar noyob bo'lishi kerak. Kalit yordamida lug'atdagi elementni topish, uni o'zgartirish yoki o'chirish mumkin. Lug'atni aniqlash uchun kvadrat qavslar ichida [Key: Value] formatida bir nechta elementlarni yozish kerak. Misol uchun:

```
telefon =
["Artel": "iPhone 6S", "Microsoft": "Lumia 950", "Google": "Nexus
X5"]
```

Bu erda lug'at *telefon* lar deb ataladi va uchta elementni o'z ichiga oladi. Masalan, birinchi element: "Artel": "iPhone 6S". "Artel" kalitni, "iPhone 6S" esa qiymatni bildiradi. Kalit ham, qiymat ham *String* tipiga kiradi. Lekin bu ixtiyoriy hisoblanadi. Shuningdek, kalitning tipini va qiymatini lug'atda aniq belgilash mumkin:

```
var telefonlar: [String: String] =
["Artel": "iPhone 6S", "Microsoft": "Lumia 950", "Google": "Nexus
X5"]
To'liq tipdagi e'lon qilishdan ham foydalanish mumkin, u quidagicha
yoziladi:
var telefonlar: Dictionary<String, String> =
["Artel": "iPhone 6S", "Microsoft": "Lumia 950", "Google": "Nexus
X5"]
```

Bundan tashqari bo'sh lug'at yaratish imkoniyati ham mavjud, u quyidagi konstruksiyaga ega:

```
var telefonlar: Dictionary<String, String> = [:]
// muqobil variant
var telefonlar2: [String: String] = [:]
// initsializator yordamida
var telefonlar3 = [String: String]()
isEmpty xususiyatidan foydalanib, lug'atda elementlar mavjudligini
tekshirish mumkin, shu element bo'lmasa, true qaytaradi:
var telefonlar: Dictionary<String, String> = [:]
```

```
if phones.isEmpty {
    print("telefonlar lugati bosh")
} else {
    print("Telefonlar lugatida elementlar mavjud") }
```

Lug'atdagi elementlar sonini aniqlash uchun *count* xususiyatidan foydalanish mumkin, u quidagicha yoziladi:

```
var telefonlar: [String: String] = ["Artel": "iPhone 6S", "Microsoft":
"Lumia 950", "Google": "Nexus X5"]
print (telefonlar.count)
```

Element kalitidan foydalanib, lug'atdagi ushbu elementga murojaat qilish, uni olish yoki o'zgartirish mumkin:

```
var telefonlar: [String: String] = ["Artel": "iPhone 6S", "Microsoft":
"Lumia 950", "Google": "Nexus X5"]
```

```
// elementni kalit bo'yicha olish
print(telefonlar["Artel"]) // iPhone 6S
// elementni o'zgartirish
telefonlar["Artel"] = "iPhone 5SE"
```

Lug'atdagi elementni o'zgartirishga alternativa sifatida *UpdateValue* usuli hisoblanadi:

```
phones.updateValue("iPhone 5SE", forKey: "Artel")
print(telefonlar["Artel"]) // iPhone 5SE
```

Lug'atdan elementni olib tashlash yoki o'chirish uchun unga nil qiymatini belgilash kifoya:

```
telefonlar["Google"] = nil
```

Shu bilan bir qatorda, *forKey* parametri yordamida olib tashlanadigan elementning kalitini oladigan *RemoveValue()* usulidan foydalanish mumkin:

```
phones.removeValue(forKey: "Google")
```

RemoveValue() usuli o'chirilayotgan obyektning qiymatini qaytaradi,

agar lug'atda bunday kalitga ega ob'jekt bo'lmasa, nil qaytariladi:

```
if let releasedValue = phones.removeValue(forKey: "Google") {
```

```
    print("Obyekt \removedValue olib tashlandi.")
```

```
} else {
```

```
    print("Lug'atda olib tashlanadigan element mavjud emas")
```

```
}
```

Lug'atni takrorlash uchun standart *for-in* siklidan foydalaniladi:

```
var telefonlar: [String: String] = ["Artel": "iPhone 6S", "Microsoft":
```

```
"Lumia 950", "Google": "Nexus X5"]
```

```
for (manufacturer, model) in telefonlar {
```

```
    print("\(manufacturer): \(model)")
```

Takrorlashda har bir lug'at obyektini (*key*, *value*) kortej sifatida qaytariladi, birinchi element kalitni, ikkinchi element esa qiymatni ifodalaydi. Kalitlar va qiymatlarni alohida-alohida takrorlash mumkin.

Kalitlarni alohida-alohida takrorlash quyidagicha amalga oshiriladi:

```
for manufacturer in phones.keys {
```

```
    print(manufacturer)
```

Qiymatlarni alohida-alohida takrorlash quyidagicha amalga oshiriladi:

```
for model in phones.values {
```

```
    print(model)
```

O'rnatilgan global *zip()* funksiyasidan foydalanib, ikkita massivni

Zip2Sequence obyektiga birlashtirish mumkin, so'ngra ular *Dictionary* tipidagi initsializatorga o'tkaziladi:

```
let mamlakatlar = ["Eron", "Iroq", "Suriya", "Livan"]
```

```
let boshharflar = ["Tehron", "Bag'dod", "Damashq", "Bayrut"]
```

```
var seq = zip(mamlakatlar, boshharflar)
```

```
var dict = Dictionary(uniqueKeysWithValues: seq)
```

```
for (key, value) in dict {
```

```
    print("\(key) - \(value)")
```

Misolda *mamlakatlar* massividagi har bir element ketma-ket *boshharflar* massividagi mos keladigan element bilan taqqoslanadi. Keyin

natija *uniqueKeysWithValues* parametri orqali *Dictionary* initsializatoriga uzatiladi. Shunday qilib, lug'at shakllanadi. Dastur natijasi:

Eron - Tehron Iroq - Bagdod Suriya - Damashq Livan - Bayrut

Agar ikkala massivda takroriy qiymatlar bo'lsa, ularni birlashtirishning bu usuli muvaffaqiyatsiz bo'ladi, lug'atda barcha kalitlar yagona bo'lishi kerak. Buning uchun Lug'atni ishga tushirishning boshqa shaklidan foydalanish kerak:

```
mamlakatlar = ["Eron", "Iroq", "Suriya", "Livan"]
```

```
boshharflar = ["Tehron", "Bag'dod", "Damashq", "Bayrut", "Tehron"]
```

```
var seq = zip(mamlakatlar, poytaxtlar)
```

```
var dict = Dictionary(seq, uniquingKeysWith: return {$1})
```

```
for (key, value) in dict {
```

```
    print("\(key) - \(value)")
```

Misolda, birlashtirilgan ketma-ketliklar birinchi parametr sifatida initsializatorga o'tkaziladi. *uniquingKeysWith* ning ikkinchi parametri esa takroriy kalitga mos keladigan ikkinchi massivdan barcha qiymatlarni oladigan funksiyaga ishora qiladi. Yuqoridagi holatda bu ikkita element hisoblanadi va qandaydir natijani qaytarish kerak. Bu holatda ikkinchi parametrning qiymati qaytariladi.

Subscriptlar

Sinflar, tuzilmalar va ro'yxatlar subscriptlarni e'lon qilishi mumkin. Subscriptlar kolleksiyalar yoki ketma-ketlik elementlariga kirish uchun ishlatiladi. Shunga o'xshash tushuncha mavjud - *indeksatorlar*. Subscriptlar sinf yoki tuzilma obyektiga alohida to'plam kabi munosabatda bo'lish imkonini beradi. *Subscript* kalit so'zi subscriptni aniqlash uchun ishlatiladi:

```
subscript (parametrlar) -> obyekt_qaytish_tipi {
```

```
    get {
```

```
        // qaytariladigan qiymat
```

```
    }
```

```
    set (newValue) {
```

```
        // newValue yangi qiymatni o'rnatish
```

```
    }
```

Subscript kalit so'zidan keyin, qavslar ichida elementlarni olish uchun ishlatiladigan parametrlar yoziladi. Parametrlar obyektini olish uchun raqamli indeksni ifodalaydi. Keyinchalik, ishlatiladigan elementlarning tipi ko'rsatiladi.

Subscript ikkita blokdan iborat bo'lishi mumkin: *get* va *set*. *Get* bloki elementni qaytaradi va *set* bloki *newValue* parametri orqali o'tkaziladigan yangi qiymatni o'rnatadi.

Masalan, kutubxona sinfi misolida kutubxona ma'lum bir kitoblarni to'plamini ifodalaydi. Kutubxona sinfini kitoblarni to'plamini deb o'ylash va kitoblarni indeks bo'yicha olish uchun subscriptlardan foydalanish mumkin:

```
class Kitob { // Kitob sinfi
    var nomi: String
    init (nomi: String) {
        self.nomi = nomi
    }
}
class Kutubxona { // kutubxona sinfi
    var kitoblar: [Kitob] = [Kitob]()
    init() {
        kitoblar.append(Kitob(nomi: "Utgan kunlar"))
        kitoblar.append(Kitob(nomi: "Tong"))
        kitoblar.append(Kitob(nomi: "Bukhoro"))
    }
    subscript (indeks: Int) -> Kitob {
        get {
            return kitoblar[indeks]
        }
        set (newValue) {
            kitoblar[indeks] = newValue
        }
    }
}
var myKutubxona: Kutubxona = Kutubxona()
var firstBook: Kitob = myKutubxona[0] // 0 indeksidagi elementni olish
```

```
print(firstBook.nomi) // Utgan kunlar
myKutubxona[2] = Kitob(nomi: "Abdulloh Qahhor")
// 2-indeksga elementni o'rnatish
print(myKutubxona[2].nomi) // Abdulla Qahhor
```

Bu yerda subscript *Kitob* tipi bilan ishlash uchun mo'ljallangan. *Get* blokida *Book* obyektini *kitoblar* massividan indeks bo'yicha olinadi. *Set* blokida *kitoblar* massivida *Kitob* obyektini o'rnatiladi.

Dasturda kerakli kitobni olish uchun kutubxonaga indeks bo'yicha massiv sifatida kirish mumkin:

```
var firstBook: Kitob = myKutubxona[0].
```

Ikki xil subscript mavjud:

- O'qish va yozishni qo'llab-quvvatlaydigan subscriptlar (*get* va *set* bloklari bilan);

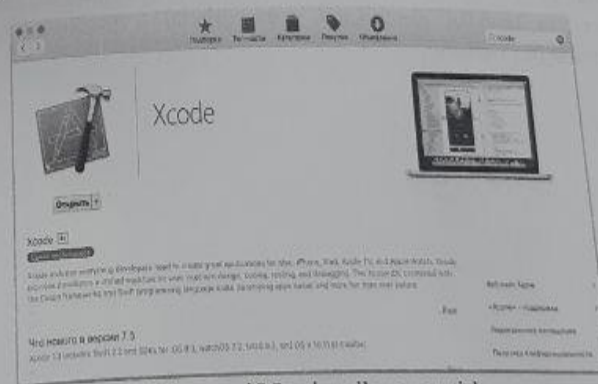
- Faqat o'qish uchun mo'ljallangan subscriptlar (faqat *get* bloki bilan)

Subscriptni faqat o'qish uchun kutubxona sinfini o'zgartirish quyidagicha amalga oshiriladi:

```
class Kutubxona {
    var kitoblar: [Kitob] = [Kitob]()
    init() {
        kitoblar.append(Kitob(nomi: "Utgan kunlar"))
        kitoblar.append(Kitob(nomi: "Tong"))
        kitoblar.append(Kitob(nomi: "Bukhoro"))
    }
    subscript (indeks: Int) -> Kitob {
        return kitoblar[indeks]
    }
}
```

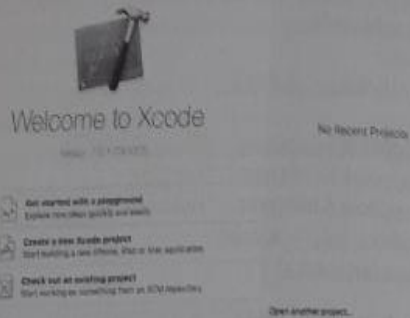
6.4. Swift dasturlash tilida mobil ilovalarni ishlab chiqish

iOS uchun mobil ilova sifatida vazni saqlash uchun tana massasi indeksini va kerakli miqdordagi kaloriyalarni hisoblash uchun oddiy ilova yaratish ketma-ketligi misoli ko'riladi (6.9-rasm). Buning uchun *Garissa Benedikt* formulalaridan va tana massasi indeksidan foydalaniladi.



6.9-rasm. iOS uchun ilova yaratish

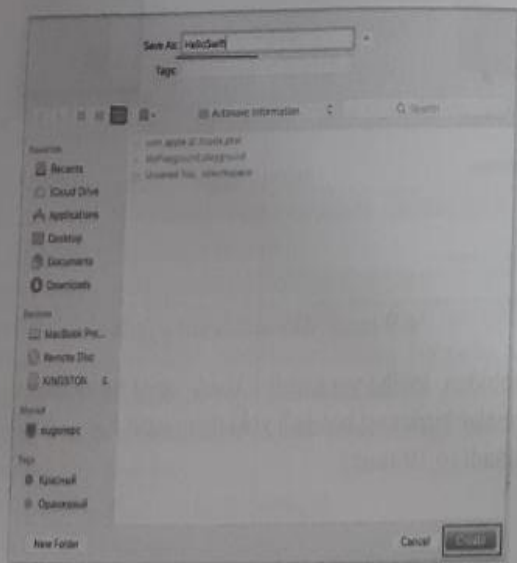
1. Birinchidan, loyiha yaratiladi. *Xcode* ishga tushiriladi va *CMD + Shift + N* tugmalar birikmasi bosiladi yoki menyudan *File -> New -> Project* buyrig'i tanlanadi (6.10-rasm).



6.10-rasm. Ilova turini tanlash

2. Ochilgan darchadan Yorliqli ilova tanlanadi va *Next* tugmasi bosiladi.

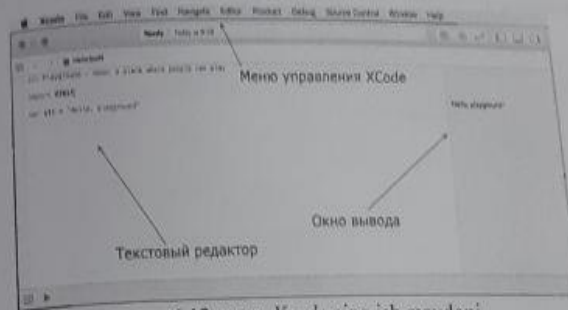
3. Ushbu bosqichda dastur nomini (Mahsulot nomi) o'ylab topish kerak va *Next* tugmasi bosiladi va saqlash uchun papka tanlanadi. Tashkilot nomi va tashkilot identifikatori parametrlarini o'zgarishsiz qoldirish mumkin (6.11-rasm).



6.11-rasm. Ilovaga kerakli ma'lumotlarni kiritish

Asosiysi, "Language" maydonidagi qiymat *Swift* bo'lishi kerak.

4. Loyihani saqlagandan so'ng, *Xcode* ning ish maydoni paydo bo'ladi (6.12-rasm).

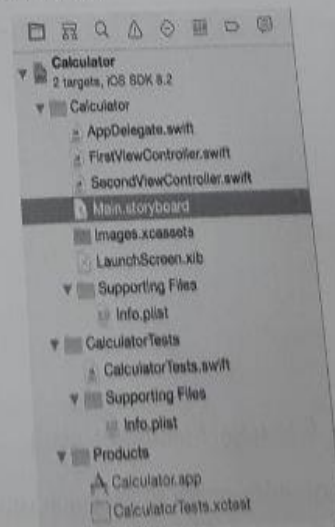


6.12-rasm. *Xcode* ning ish maydoni

Chap tomonda *Navigator* paneli, o'ng tomonda *Utilita* lar paneli joylashgan. Ushbu panellarning har birida turli funksiyalarga kirish imkonini beruvchi o'z yorliqlari mavjud.

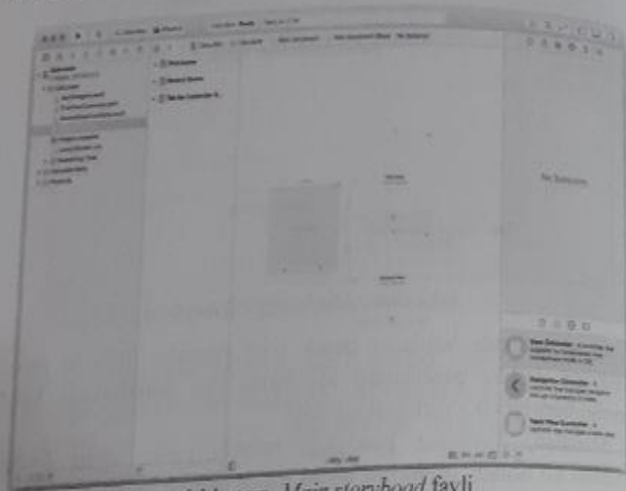
Masalan, *Navigator* panelidagi birinchi yorliq loyiha fayllari ro'yxatini ko'rsatadi, uchinchisi loyiha ichida qidirish imkonini beradi va hokazo.

5. Loyiha fayllari ro'yxatidan *Main.storyboard* nomli faylni topish va ustiga bosish lozim (6.13-rasm).



6.13-rasm. *Main.storyboard* nomli faylni topish

Main.storyboard fayli ilovada qaysi ekranlar (kontrollerlar) borligini belgilaydi, shuningdek ekranlarga elementlar qo'shish, ekranlar orasidagi munosabatlarni o'rnatish va hokazolarni belgilaydi (6.14-rasm).



6.14-rasm. Main.storyboard fayli

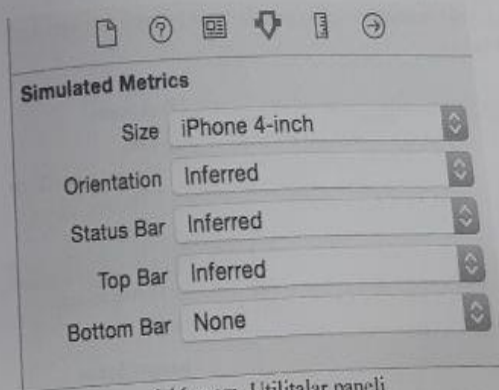
Agar biror bir kontroller tanlansa, u ko'k ramka bilan belgilanadi (6.15-rasm):



6.15-rasm. Kontroller tanlash

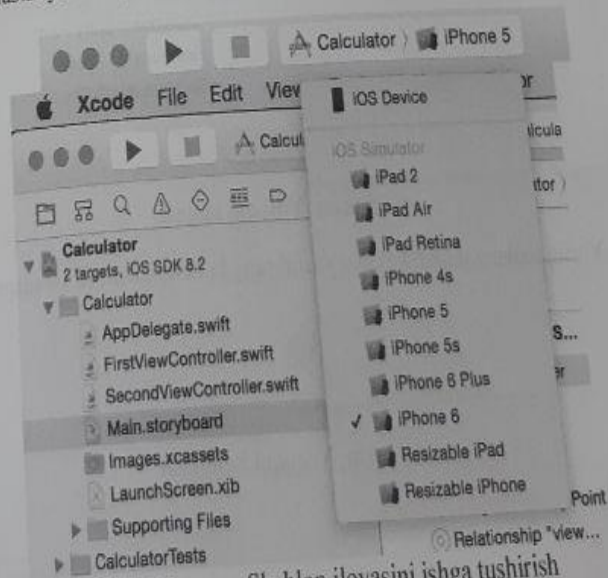
Tekshirish moslamasi tanlangandan so'ng, uning xususiyatlari Utilitalar panelida paydo bo'la boshlaydi. Masalan, o'lcham maydonida boshqa

qiymatni tanlash orqali kontroller o'lchamini o'zgartirish mumkin (6.16-rasm).



6.16-rasm. Utilitalar paneli

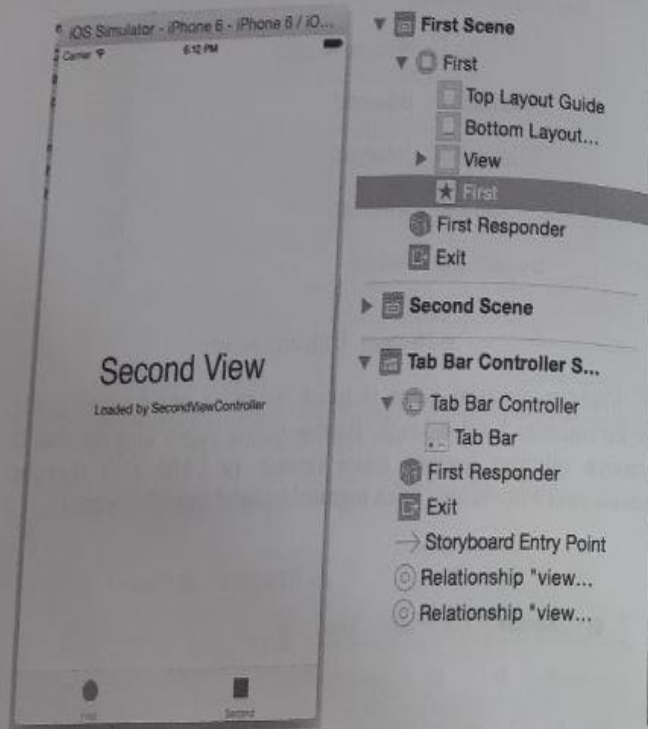
6. Shablon ilovasini ishga tushirish lozim va uning simulyatorida qanday ko'rinishini ko'rish kerak. Buning uchun yuqori chap burchakda simulyatsiya qilingan qurilma turini tanlash va `CMD + R` tugmalar birikmasini yoki `Play` belgisi bilan tugmani bosish lozim (6.17-rasm).



6.17-rasm. Shablon ilovasini ishga tushirish

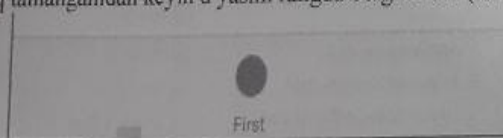
Ko'rinib turibdiki, dastur ikkita ekranga ega, ular o'rtasida ma'lumot almashish yorliqlar paneli yordamida amalga oshiriladi.

7. Yorliqlar nomini o'zgartirish kerak. Buni amalga oshirish uchun uni bosish yoki boshqaruv ierarxiyasida tanlash orqali yorliqni tanlash kerak (6.18-rasm).



6.18-rasm. Yorliqni tanlash

Yorliq tanlanganidan keyin u yashil rangda belgilanadi (6.19-rasm).



6.19-rasm. Yorliqni belgilash

8. Endi *Utilita*-lar panelida *Atribut*-lar inspektorini tanlash lozim va Sarlavha maydonidagi qiymatni birinchi yorliq uchun *BMR/BMI*, ikkinchisi uchun *Kilocalories* ga o'zgartirish lozim (6.20-rasm).



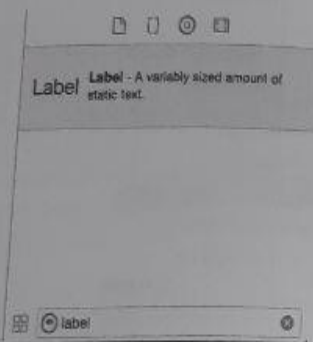
6.20-rasm. Sarlavha maydonidagi qiymatni o'zgartirish

Birinchi kontrollerda barcha turdagi yozuvlar joylashtirilgan. Ularni olib tashlash kerak. Buning uchun ularni tanlash va *Delete* tugmasini bosish lozim (6.21-rasm).



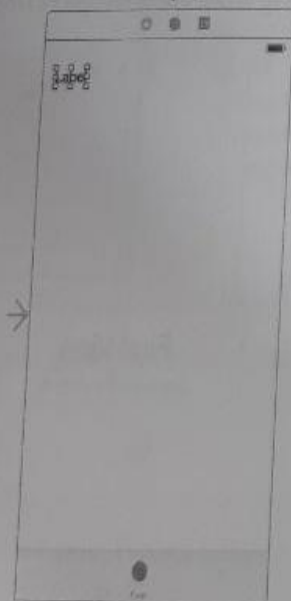
6.21-rasm. Yozuvlarni o'chirish

9. Boshqaruv elementlarini qo'shish uchun *Utilita*-lar panelining pastki qismida obyektlarni tanlash va ularni sahnaga qo'shish mumkin bo'lgan *Obyekt*-lar kutubxonasi mavjud. *Label* obyektini topish va uni *BMR/BMI* kontrolleriga tortib o'tish lozim. Buni amalga oshirishdan oldin, o'lchov standart bo'lishi uchun sahnaga ikki marta bosish lozim (6.22-rasm).



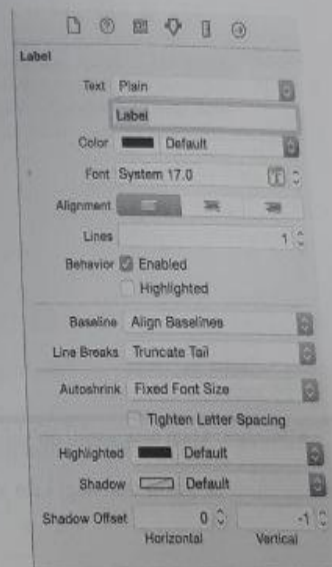
6.22-rasm. Label obyektini topish

Sahnada shunga o'xshash element paydo bo'lishi kerak (6.23-rasm).



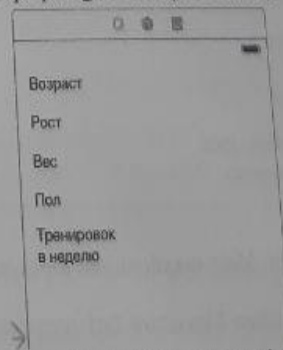
6.23-rasm. Label obyektini akslantirish

Utilita-lar panelida matnni, shrift hajmini va hokazolarni o'zgartirish mumkin (6.24-rasm).



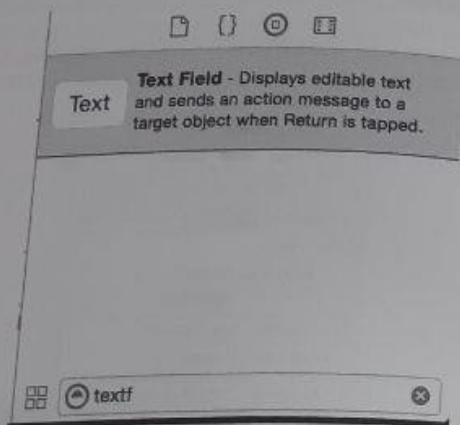
6.24-rasm. Label obyektini o'zgartirish

10. Yana bir nechta *Label*-larni tortib o'tish lozim va ularga rasmdagi kabi matnlarni berish lozim. Oxirgi yorliq uchun satrlarni 2 ga o'rnatish kerak, shunda matn boshqa qatorga o'tadi (6.25-rasm).



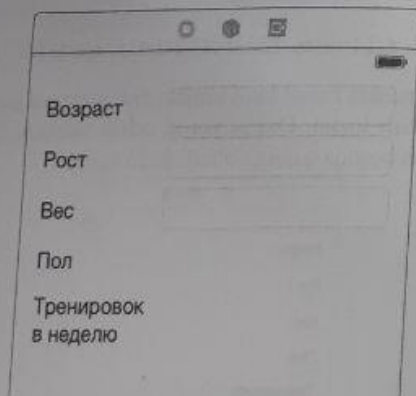
6.25-rasm. Bir nechta Label-larni tortib o'tish

11. Endi 3 ta matn maydonini qo'shish lozim (Matn maydoni) (6.26-rasm).



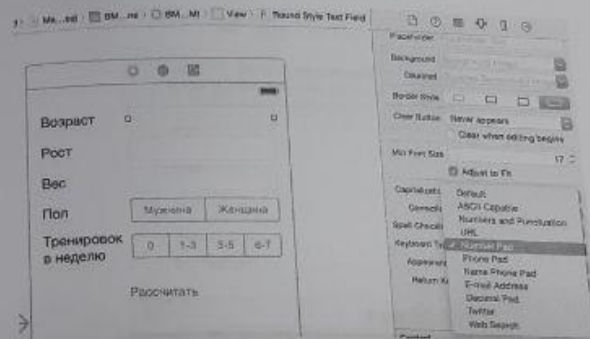
6.26-rasm. Matn maydonini qo'shish

Matn maydonini qo'shish va ularni rasmdagi kabi joylashtirish lozim (6.27-rasm).



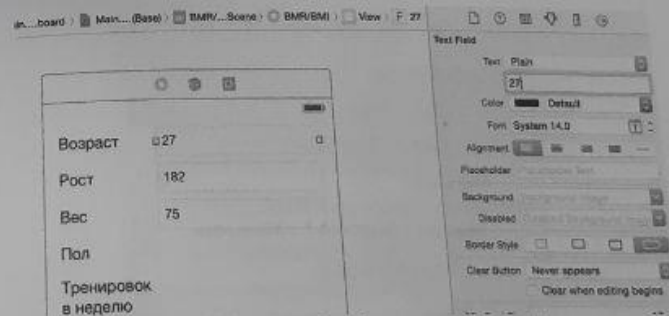
6.27-rasm. Matn maydonlarini joylashtirilganligi

Matn maydonlari uchun klaviatura turi parametrini *Number Pad*-ga o'rnatish lozim (6.28-rasm).



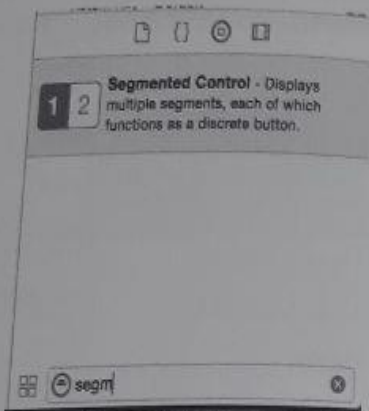
6.28-rasm. *Number Pad*-ni o'rnatish

Number Pad-ga o'rnatilgandan so'ng standart matnni o'rnatish lozim (6.29-rasm).



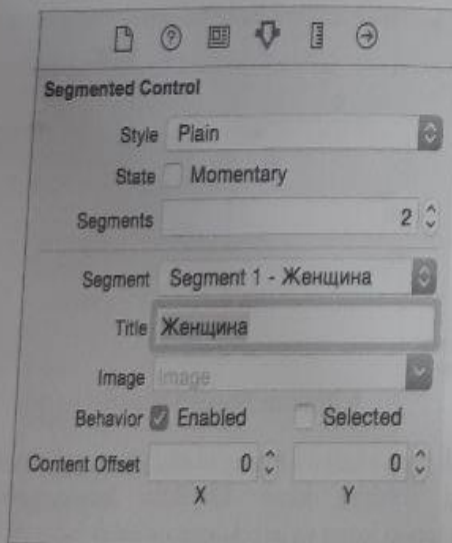
6.29-rasm. Standart matnni o'rnatish

12. Endi jins va haftalik mashg'ulotlar sonini tanlash uchun boshqaruv elementlarini qo'shish lozim. Obyektlar kutubxonasida *Segment*-li boshqaruvni topish lozim va uni ekranga qo'shish lozim (6.30-rasm).



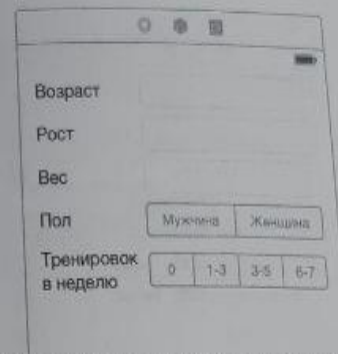
6.30-rasm. Segment-li boshqaruvni qo'shish

Segment-li boshqaruv har bir segment uchun segmentlar sonini va matni o'zgartirishga imkon beradi (6.31-rasm).



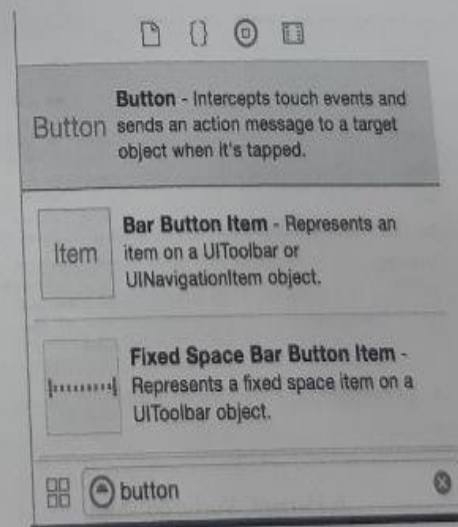
6.31-rasm. Segment-larni qiymatlarini o'zgartirish

Segment-larni qiymatlarini xuddi skrinshotdagidek o'zgartirish lozim (6.32-rasm).



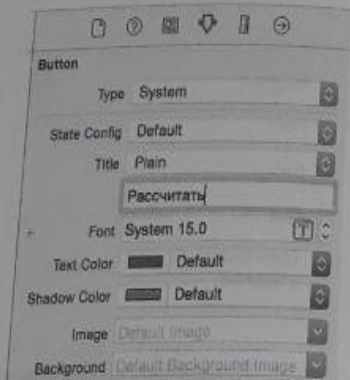
6.32-rasm. Segment-larni qiymatlarini o'zgartirish

13. Keyingi qadamda *Button* (tugma) qo'shish lozim (6.33-rasm). Bu darchada tugmaning oddiy shaklidan tashqari *Bar Button Item* va *Fixed Space Bar Button Item* shakllari ham mavjud.



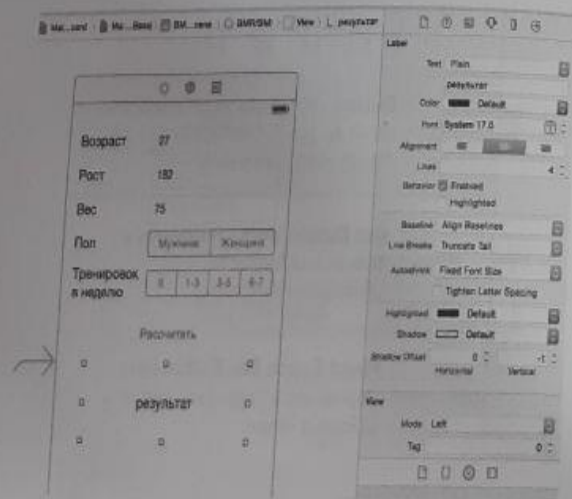
6.33-rasm. Tugma qo'shish

Tugmani ilovaga qo'shishdan oldin qo'shiladigan tugmaga nom berish lozim. Shuningdek, tugmaning boshqa parametrlarini ham o'zgartirish mumkin (6.34-rasm).



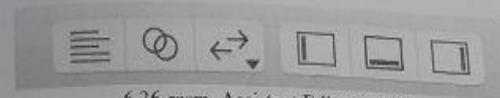
6.34-rasm. Tugma parametrlarini o'zgartirish

Shundan so'ng, 4 ga teng chiziq bilan boshqa yorliq qo'shish lozim (6.35-rasm).



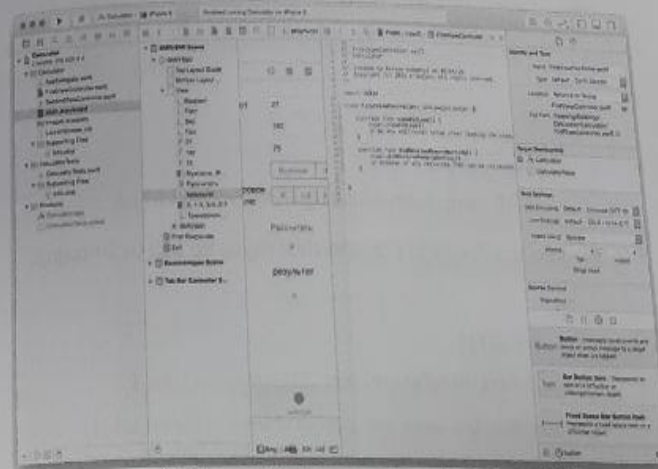
6.35-rasm. Yorliq qo'shish

14. BMI/BMR kontrollerini tanlash kerak, yuqori o'ng burchakdagi ikkita kesishgan halqani bosish kerak va Assistant Editor ochiladi. U ushbu kontroller bilan bog'langan kodni ko'rsatadi (6.36-rasm).



6.36-rasm. Assistant Editor paneli

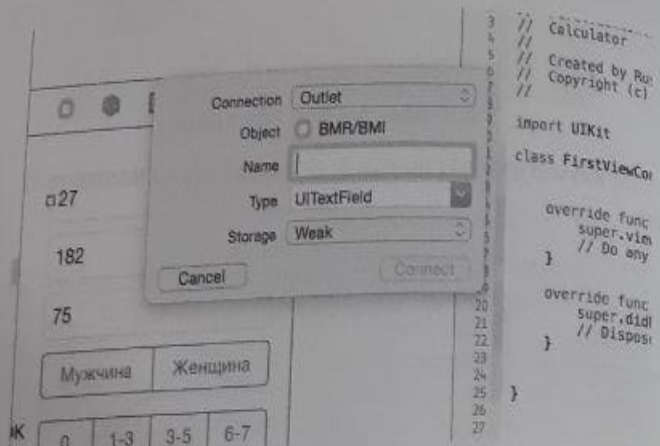
15. Keyingi qadamda Boshqaruv elementlari tortib o'tkaziladi (6.37-rasm).



6.37-rasm. Boshqaruv elementlarini qo'shish

Boshqaruv elementlarini qo'shishni amalga oshirish uchun birinchi matn maydoni tanlanadi (bu yosh bo'ladi), *Ctrl* tugmasi bosiladi, ustiga yana bir marta bosiladi va tugmani qo'yvormasdan, uni sinf ichiga tortib o'tiladi. Agar hamma narsa to'g'ri bajarilgan bo'lsa, "Insert Outlet, Action, ..." so'rovi ko'rinadi.

Keyin kursorni qo'yib yuborib va ulanishni yaratish uchun oyna ko'rinadi. Nom uchun *ageTextField*-ni kiritish va *Connect*-ni bosish lozim (6.38-rasm).



6.38-rasm. Nom uchun *ageTextField*-ni kiritish

16. Ekranda *ageTextField* o'zgaruvchisi paydo bo'ladi (6.39-rasm).

```

1 //
2
3 import UIKit
4
5 class FirstViewController: UIViewController {
6
7     @IBOutlet weak var ageTextField: UITextField!
8
9     override func viewDidLoad() {
10         super.viewDidLoad()
11         // Do any additional setup after loading the view
12     }
13
14     override func didReceiveMemoryWarning() {
15         super.didReceiveMemoryWarning()
16         // Dispose of any resources that can be recreated
17     }
18 }
19
20
21
22
23
24
25
26
27
28

```

6.39-rasm. *ageTextField* o'zgaruvchisi

Buni matn maydonining qolgan qismi, segmentlangan boshqaruv elementlari va matn natijasi bilan yorliq uchun bajarish lozim. Natijada u shunday bo'lishi kerak (6.40-rasm):

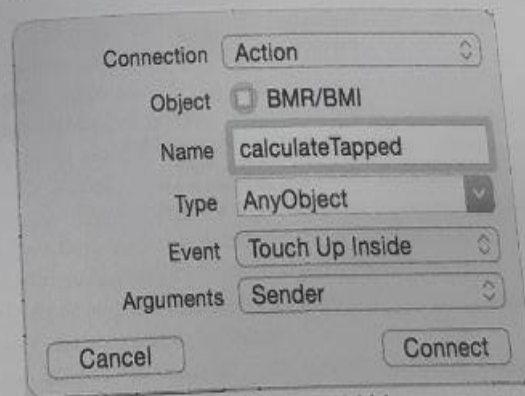
```

11 class FirstViewController: UIViewController {
12
13     @IBOutlet weak var ageTextField: UITextField!
14     @IBOutlet weak var heightTextField: UITextField!
15     @IBOutlet weak var weightTextField: UITextField!
16     @IBOutlet weak var sexSegmentedControl: UISegmentedControl!
17     @IBOutlet weak var activitySegmentedControl: UISegmentedControl!
18     @IBOutlet weak var resultsLabel: UILabel!
19
20
21
22
23
24
25
26
27
28
29

```

6.40-rasm. Dastur kodi ko'rinishi

Keyin tugmani sudrab o'tish kerak, lekin ulanish turini *Outlet* sifatida emas, balki *Action* sifatida belgilash lozim. Nom sifatida *accountTapped* ni kiritish lozim (6.41-rasm).



6.41-rasm. Nom kiritish

Nom kiritilgandan keyin kodka quyidagi ko'rinishdagi o'zgarish yuzaga keladi (6.42-rasm):

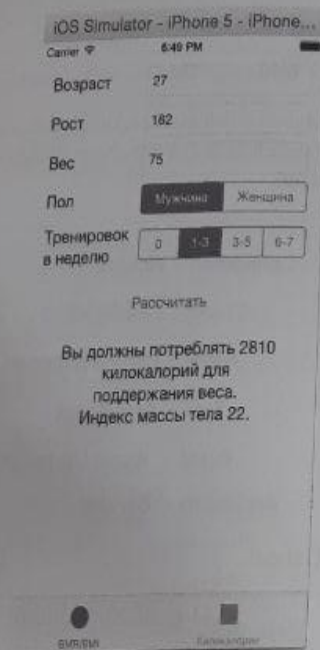
```

11 class FirstViewController: UIViewController {
12
13     @IBOutlet weak var ageTextField: UITextField!
14     @IBOutlet weak var heightTextField: UITextField!
15     @IBOutlet weak var weightTextField: UITextField!
16     @IBOutlet weak var sexSegmentedControl: UISegmentedControl!
17     @IBOutlet weak var activitySegmentedControl: UISegmentedControl!
18     @IBOutlet weak var resultsLabel: UILabel!
19
20     @IBAction func calculateTapped(sender: AnyObject) {
21
22     }
23
24     override func viewDidLoad() {
25
26
27
28
29

```

6.42-rasm. Nom kiritilgandan keyin koddagi o'zgarish

17. Endi yuqoridagi kodni *accountTapped* usuliga ko'chirish lozim. Ushbu kod hisoblashni amalga oshiradi va natijani ekranda ko'rsatadi.
18. Ilovani ishga tushirish va sinovdan o'tkazish lozim (6.43-rasm).



6.43-rasm. Ilovani ishga tushirish va sinovdan o'tkazish

Nazorat savollari:

1. Swift-da Any va AnyObject-dan qachon foydalanish kerak?
2. Swiftda lug'atlardan qanday foydalanish kerak?
3. Ilova arxitekturası: nima uchun muhim va nimalarga e'tibor berish kerak?
4. Eng oddiy ma'lumotlarni qanday saqlash va o'qish kerak?
5. Swiftda kengaytmalardan qanday foydalanish mumkin?;
6. Swift-da View Controllers o'rtasida ma'lumotlar qanday uzatiladi?
7. Swift Scope va Variable Definition Context
8. Swift-da kolleksiyalar bilan ishlash funksiyalarini qisqartirish va filtrlash

7. FLUTTERDA ANDROID OT UCHUN MOBIL ILOVA ISHLAB CHIQISH

7.1. Flutter-ga kirish

Flutter – bu Google kompaniyasining freymworki bo'lib, u bir xil koddan foydalanishi mumkin bo'lgan platformalararo ilovalarni yaratish imkonini beradi. Platformalar turlari ko'p – bular web-illovalar, Android va iOS uchun mobil ilovalar, Windows, MacOS, Linux ish stoli operatsion tizimlari uchun grafik ilovalar, shuningdek, web-illovalar hisoblanadi.

Flutter bilan ishlashning o'ziga xos xususiyati shundaki, turli platformalar uchun ilovalar bir xil kodga ega bo'lishi mumkin. Amaldagi platformalar ekvivalent bo'lmaganligi sababli, kodning ha'zi alohida qismlari ma'lum bir OT uchun sozlanishi kerak, masalan, iOS uchun, lekin shunga qaramay, kodning aksariyati bir xil bo'lishi mumkin. Bu ishlab chiquvchilarga barcha qo'llab-quvvatlanadigan platformalar uchun ilovalar yaratishda vaqt va resurslarni sezilarli darajada tejash imkonini beradi. Dasturlash tili sifatida Dart dasturlash tilidan foydalaniladi. Ilovani yaratishda Flutter Android yoki iOS yoki boshqa platformalarda ishlashi mumkin bo'lgan Dart AOT (ishga tushirishdan oldin ilova kompilyatsiyasi) yordamida Dart kodini mahalliy dastur kodiga tarjima qiladi. Biroq, dasturni ishlab chiqishda Flutter uni tezlashtirish uchun JIT (ilovaning ishlayotgan vaqtda kompilyatsiyasi) dan foydalanadi.

Flutterni ishlab chiqish uchun nima ishlatiladi? Dastur kodini yozish uchun har qanday matn muharriridan foydalanish mumkin, so'ngra dasturni kompilyatsiya qilish uchun Flutter SDK-dan buyruq qatori yordam dasturidan foydalanish mumkin. Biroq, Android Studio va IntelliJ IDEA kabi muhitlar, shuningdek, Visual Studio Code matn muharriri uchun Google ishlab chiqishni osonlashtiradigan maxsus plaginlarni chiqargan. Shuning uchun, Android Studio va IntelliJ IDEA ko'pincha Flutterda mobil ishlab chiqish uchun ishlatiladi.

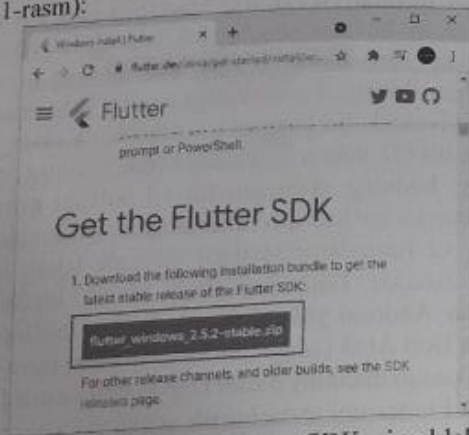
Flutter SDK-ni o'rnatish

Turli xil operatsion tizimlarga SDK-ni o'rnatish uchun <https://flutter.dev/docs/get-started/install/> sahifasidan kerakli havolalari tanlab yuklab olish mumkin.

Flutter-ni Windows-ga o'rnatish

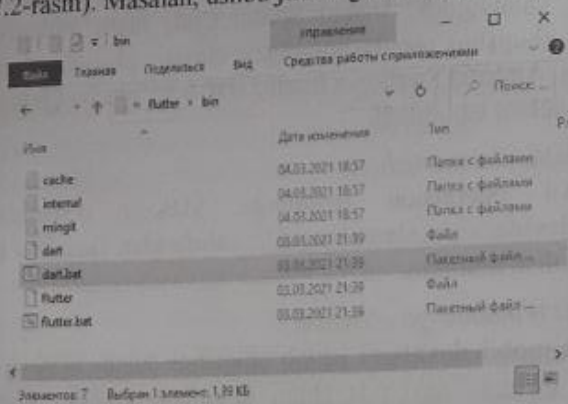
Flutterda mobil ilovalarni ishlab chiqishni uchun SDK-ni o'rnatish lozim. Lekin birinchi navbatda shuni ta'kidlash kerakki, Flutter SDK Windows tizimida ishlashi uchun tizimda *Git for Windows* o'rnatilgan

bo'lishi kerak, uni <https://git-scm.com/download/win> sahifasidan topish mumkin. Flutter Android SDK dan ham foydalanadi. Buning uchun Android Studio-ni o'rnatish lozim, u bilan birga barcha kerakli vositalar o'rnatiladi. Flutter SDK-ni to'g'ridan-to'g'ri o'rnatish uchun <https://flutter.dev/docs/get-started/install/windows> sahifasidan kerakli havola tanlab yuklab olish mumkin. Ushbu sahifada Flutter SDK-ni yuklab olish bo'limida Flutter SDK bilan zip arxiviga havolani topib va uni yuklab olish mumkin (7.1-rasm):



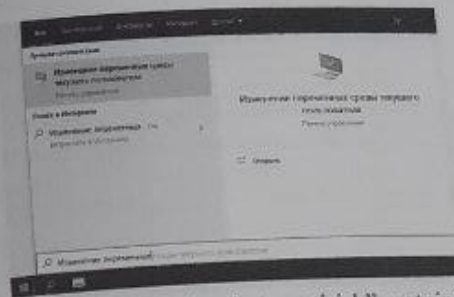
7.1-rasm. Windows uchun Flutter SDK-ni yuklab olish

Arxivni yuklab olgandan keyin, masalan, C diskida ochish mumkin. Misol uchun arxivni C:\flutter jildida ochish mumkin. Flutter\bin papkasidagi arxivda dasturni kompilyatsiya qilish vositalarini topish mumkin (7.2-rasm). Masalan, ushbu jildning to'liq yo'li C:\flutter\bin:



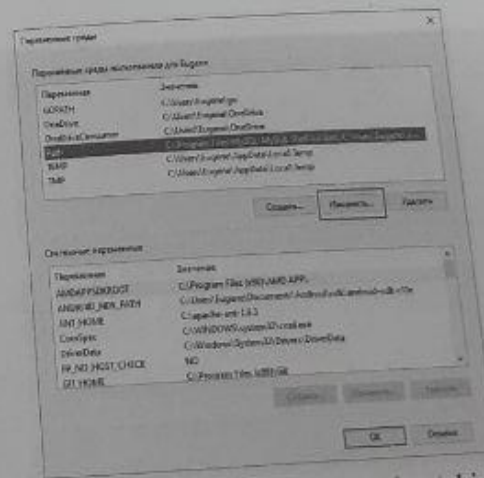
7.2-rasm. Flutter\bin papkasi ko'rinishi

Windows OTda qidiruv orqali muhit o'zgaruvchisini qo'shish uchun "Joriy foydalanuvchining muhit o'zgaruvchilarini o'zgartirish" varianti tanlanadi. Buning uchun qidiruv maydoniga "O'zgaruvchilarni o'zgartirish" matni kiritiladi (7.3-rasm):



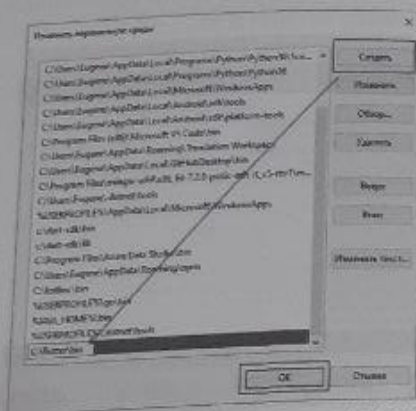
7.3-rasm. "O'zgaruvchilarni o'zgartirish" matnini kiritish

"Joriy foydalanuvchining muhit o'zgaruvchilarini o'zgartirish" bo'limini tanlash lozim. Keyin barcha muhit o'zgaruvchilarini ko'rish mumkin bo'lgan oyna ochiladi (Shuningdek, Система -> Дополнительные параметры системы -> Переменные среды orqali o'tish mumkin). Keyin Flutter SDK-dagi bin papkasiga yo'lni qo'shish orqali Path o'zgaruvchisini o'zgartirish kerak. Buning uchun Path bandini tanlash va "Изменить" tugmasini bosish lozim (7.4-rasm):



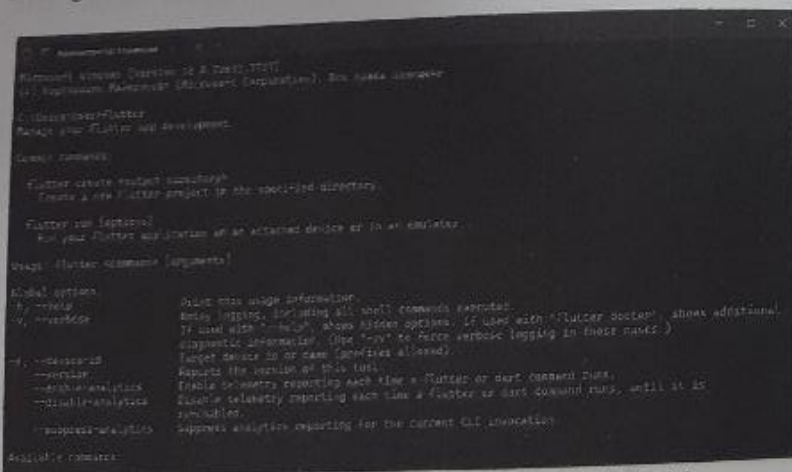
7.4-rasm. Muhit o'zgaruvchisini qo'shish oynasi

Keyin, "Создать" tugmasini bosish va paydo bo'lgan kiritish maydoniga Flutter SDK-dan bin papkasiga yo'lni kiritish kerak (7.5-rasm);



7.5-rasm. bin papkasiga yo'lni kiritish

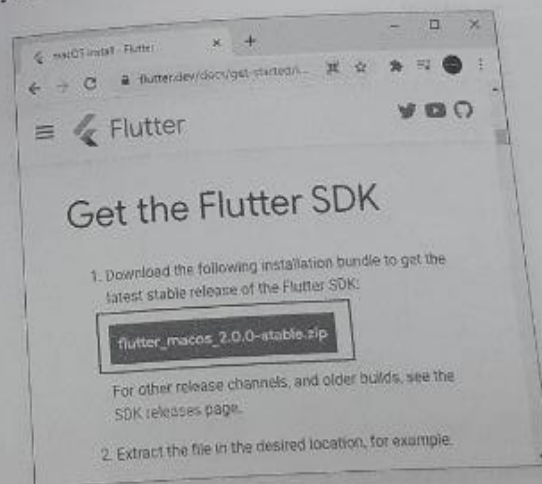
Flutter to'g'ri o'rnatilganligini tekshirish uchun buyruq qatorini ochish va flutter buyrug'ini kiritish kerak. Agar Windows bu buyruqni tanisa va ha'zi yordam ma'lumotlarini ko'rsatsa (masalan, konsolda ma'lum buyruqlardan qanday foydalanish kerak), u holda Flutter o'rnatilgan va sozlangan hisoblanadi (7.6-rasm).



7.6-rasm. Flutter to'g'ri o'rnatilganligini tekshirish

Shu bilan bir qatorda, Flutter SDK ni Git orqali o'rnatish mumkin. Buni amalga oshirish uchun SDK joylashgan papkaga buyruq satiri orqali o'tish lozim va keyin buyruq satrida quyidagi buyruqni bajarish kerak: git clone -b stable https://github.com/flutter/flutter.git Bunday holda, Flutter SDK ning so'nggi versiyasi GitHub omboridan yuklab olinadi. Ushbu buyruqni bajarish uchun yuqorida aytib o'tilganidek, Git for Windows dasturini o'rnatish kerak.

Flutter-ni MacOS-ga o'rnatish Flutter-ni o'rnatish va yangilash uchun git-dan foydalaniladi, shuning uchun git-ni o'rnatish kerak. Biroq, agar Xcode o'rnatilgan bo'lsa, git-ni o'rnatish shart emas, chunki Xcode git-ni o'z ichiga oladi. Flutter SDK-ni o'rnatish uchun <https://flutter.dev/docs/get-started/install/macos> saytiga o'tish va sahifadagi Flutter SDK havolasini topish va uni yuklab olish kerak (7.7-rasm):



7.7-rasm. MacOS uchun Flutter SDK-ni yuklab olish

Keyinchalik, Flutter SDK bilan jildni qattiq diskning biror joyiga joylashtirish kerak. Misol uchun, bu holda, Flutter SDK joriy foydalanuvchining tub papkasida (Users/Gost) joylashgan.

Terminalda Flutter SDK dan foydalanish uchun tizim o'zgaruvchilariga SDK ichidagi bin papkasiga yo'lni qo'shish kerak. Joriy terminal oynasi uchun qo'shish quyidagicha amalga oshiriladi:

```
export PATH="$PATH:[Flutter SDK jildiga yo'l]/flutter/bin"
```

Flutter buyruq'ini birinchi marta ishlatganda, Dart SDK yuklanadi.

Flutter-ni yangilash

Agar Flutter-ni yangilash kerak bo'lsa, buyruq satrida quyidagi buyruqni bajarish kerak:

```
flutter upgrade
```

Flutter faol rivojlanayotganligi sababli uni muntazam yangilab turish yaxshiroqdir. Flutter SDK ni o'rnatgandan so'ng birinchi oddiy dasturni yaratishni ko'rib chiqamiz.

Windows uchun dastur yaratish

Avvalo, Flutter loyihalari uchun lokal diskda katalog yaratiladi. Masalan, `C:\flutterse` katalogi.

CMD konsolida `cd` buyruq'i yordamida yaratilgan katalogga o'tish kerak. Keyin quyidagi buyruqni kiritish lozim (7.8-rasm):

```
flutter create myapp
```



```
Администратор: Командная строка
Microsoft Windows [Version 10.0.22631.3737]
(c) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

C:\Users\User>flutter create myapp
Creating project myapp...
Resolving dependencies in 'myapp'... (1.7s)
Downloading packages...
Got dependencies in 'myapp'.
Wrote 120 files.

All done!
You can find general documentation for Flutter at:
https://docs.flutter.dev/
Detailed API documentation is available at:
https://api.flutter.dev/
If you prefer video documentation, consider:
https://www.youtube.com/c/FlutterDev

In order to run your application, type:

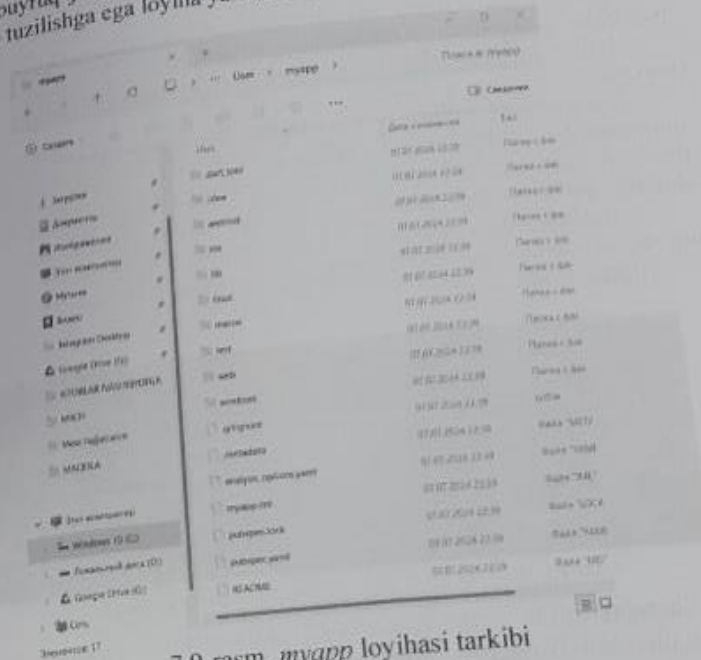
$ cd myapp
$ flutter run

Your application code is in myapp\lib\main.dart.

C:\Users\User>
```

7.8-rasm. CMD konsolida buyruqni kiritish

Bu buyruq joriy papkada `myapp` nomli loyiha yaratadi. Natijada murakkab tuzilishga ega loyiha yaratiladi (7.9-rasm).



7.9-rasm. `myapp` loyihasi tarkibi

Loyiha tuzilishining asosiy elementlari quyidagilar:

- `.dart-tool` papkasi foydalanilgan paketlar haqidagi ma'lumotlarni saqlaydigan maxsus papkadir;
- `idea` papkasi asosiy konfiguratsiyani o'z ichiga olgan Android Studio uchun maxsus papkadir;
- `android` papkasida Dart ilovasini Android bilan bog'lash imkonini beruvchi kod va qo'shimcha fayllar mavjud;
- `ios` papkasida Dart ilovasini iOS bilan bog'lash imkonini beruvchi kod va qo'shimcha fayllar mavjud;
- `build` papkasida ilovalarni yaratish jarayoni natijasida yaratilgan fayllar mavjud;
- `lib` papkasida haqiqiy Dart ilovasi fayllari mavjud. Flutterda ilova yaratishda asosiy ish aynan shu papka yordamida amalga oshiriladi;
- `test` papkasi testlar bilan fayllarni saqlash uchun mo'ljallangan;

- web papkada Flutter-da web-illovani yaratish uchun kod va qo'shimcha fayllar mavjud;

- `pubsec.yaml` fayli loyiha konfiguratsiyasini, xususan, loyiha paketini, bog'liqliklar ro'yxatini va boshqalarni saqlaydi;

Ushbu loyihada ishga tushirish mumkin bo'lgan minimal funksiyalar mavjud.

Ilovani ishga tushirish va testlash uchun emulyatorlar yoki haqiqiy qurilmalardan foydalanish mumkin. Flutter-da web-illovalarni ishga tushirish uchun brauzerlar shartli "qurilmalar" sifatida ishlatiladi, ishchi stoli ilovalarini testlash uchun - joriy kompyuter ishlatiladi.

Loyihani ishga tushirishda qurilma mavjudligini tekshirish uchun buyruq satrida quyidagi buyruqni bajarish mumkin (7.10-rasm): flutter devices

```
flutter@ms:~$ flutter devices
Connected devices
-----
Model Name: *wllatd5bc270c2c * android-arm64 * Android 8.1.0 (API 27)
Device type: *chrome *
CPU: *arm64-t802
Page layout: *edge * web-javascript * Microsoft Edge
CPU: *arm64-t802
```

7.10-rasm. Qurilma mavjudligini tekshirish buyrug'i

Shunday qilib, skrinshotda test uchun uchta "qurilma" borligini ko'rish mumkin. Birinchisi, Android ilovasini testlash uchun mo'ljallangan Nexus 5X mobil qurilmasi akslangan. Qolgan ikkita qurilma esa Chrome va Microsoft Edge web-illovalarini testlash uchun mo'ljallangan. Shuni ta'kidlash kerakki, skrinshotdan ko'rinib turibdiki, Google Chrome brauzeriga "chrome" matn belgisi, Microsoft Edge-ga esa "edge", Nexus 5X uchun esa "013a70d5bc970c2c" yorlig'i berilgan. Ushbu belgilar ma'lum bir qurilmada loyihani ishga tushirishga imkon beradi. Odatda ro'yxatdagi birinchi qurilma tanlanadi.

Android-da ilovani ishga tushirish

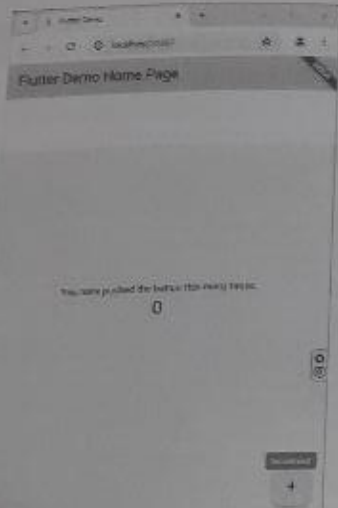
Mobil qurilmani testdan o'tkazish uchun drayverni o'rnatish kerak. Agar Windows 10 OT bo'lsa, unda, tizimning o'zi drayverni yangilash markazi orqali topishi va uni o'rnatishi mumkin. Shuningdek, Google Usb Driver paketini ham o'rnatish kerak. Ushbu paketni Android SDK Manager orqali o'rnatish mumkin. Bundan tashqari, mobil qurilmada ishlab chiqish rejimini va USB orqali nosozliklarni tuzatishni yoqish kerak.

Loyihani ishga tushirish uchun buyruq satriga o'tib `cd` buyrug'i yordamida `myapp` katalogiga o'tish lozim. Keyin loyihani ishga tushirish va quyidagi buyruqni kiritish lozim (7.11-rasm): flutter run

```
flutter@ms:~$ flutter run
Launching lib/main.dart on Nexus 5X in debug mode...
Checking the license for package Android SDK Build-Tools 29.0.2 in C:\Users\Ivan\AppData\Local\AndroidSdk\licenses
License for package Android SDK Build-Tools 29.0.2 accepted.
Preparing "Install" Android SDK Build-Tools 29.0.2 (revision: 29.0.2)
"Install" Android SDK Build-Tools 29.0.2 (revision: 29.0.2) ready.
Installing Android SDK Build-Tools 29.0.2 in C:\Users\Ivan\AppData\Local\AndroidSdk\build-tools\29.0.2
"Install" Android SDK Build-Tools 29.0.2 (revision: 29.0.2) complete.
"Install" Android SDK Build-Tools 29.0.2 (revision: 29.0.2) finished.
Running Gradle task 'assembleDebug'...
  38.9s
Installing build/app/outputs/flutter-apk/app.apk...
  738ms
A splash screen was provided to Flutter, but this is deprecated. See flutter.dev/go/android-splash-migration for migration steps.
```

7.11-rasm. Loyihani ishga tushirish

Dasturni har doim loyiha papkasidan kompilyatsiya qilish va ishga tushirish kerak. Natijada ulangan Android qurilmasiga o'rnatiladigan kompilyatsiya qilingan `apk` paketi paydo bo'ladi. Birinchi marta ilova ishga tushirilganida emulyatsiya jarayoni uzoq davom etishi mumkin. Natijada, standart tarkibga ega ilova ulangan qurilmada ishga tushadi (7.12-rasm):



7.12-rasm. Loyihani ulangan qurilmada ishga tushishi

Dasturni ishga tushirish ilovasi *lib* papkasida joylashgan *main.dart* fayli ichida aniqlanadi. Odatda bu fayl quyidagi tarkibga ega bo'ladi:

```
import 'package:flutter/material.dart';
void main() {
  runApp(const MyApp());
}
class MyApp extends StatelessWidget {
  const MyApp({Key? key}): super(key: key);
  // Ushbu vidjet ilovaning ildizidir
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Demo',
      theme: ThemeData(
        // Bu ilovaning mavzusi.
        // Ilovangizni "flutter run" bilan ishga tushirib ko'ring.
        // Ilovada ko'k asboblar paneli borligini ko'rasiz.
        // Keyin, ilovadan chiqmasdan, quyidagi asosiy Swatch-ni
        // Colors.green- ga o'zgartirib ko'ring va keyin "tez qayta yuklash" ni
        // ishga tushiring ("flutter run" ishga tushirilgan konsolda "r"
```

```
// tugmasini bosib yoki shunchaki "tez qayta yuklash" uchun
// o'zgarishlarni saqlang. Flutter IDE).
// Hisoblagich nolga qaytmaganiga e'tibor bering: ilova qayta ishga
// tushmaydi.
primarySwatch: Colors.blue, ),
home: const MyHomePage(title: 'Flutter Demo Home Page'), );
}}
```

```
class MyHomePage extends StatefulWidget {
  const MyHomePage({Key? key, required this.title}): super(key: key);
  // Bu vidjet ilovangizning bosh sahifasidir. Bu holatga tegishli, ya'ni
  // uning ko'rinishiga ta'sir qiluvchi maydonlarni o'z ichiga olgan
  // State obyektini (quyida belgilangan) bor.
  // Bu sinf davlat uchun konfiguratsiyadir. Bu holda ilova vidjeti
  // tomonidan taqdim etilgan va State qurish usulida foydalaniladigan
  // qiymatlarni (bu holda sarlavha) saqlaydi. Vidjet kichik sinfidagi
  // maydonlar har doim "yakuniy" deb belgilangan.
  final String title;
  @override
  State<MyHomePage> createState() => _MyHomePageState();
}
class _MyHomePageState extends State<MyHomePage> {
  int _counter = 0;
  void _incrementCounter() {
    setState() {
      // Bu setState usuli Flutter tizimiga ushbu holatda nimadir
      // o'zgartirishini bildiradi, bu esa displey yangilangan qiymatlarni
      // aks ettirishi uchun quyida yaratish usulini qayta ishga tushirishiga
      // olib keladi. Agar biz _counter ni setState() ga qo'ng'iroq qilmasdan
      // o'zgartirsak, unda qurish usuli qayta chaqirilmaydi va shuning
      // uchun hech narsa sodir bo'lmaydi..
      _counter++; }); }
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text(widget.title), ),
      body: Center(
        child: Column(
```

```

mainAxisAlignment: MainAxisAlignment.center,
children: <Widget>[
  const Text(
    'You have pushed the button this many times:',
  ),
  Text(
    '$_counter',
    style: Theme.of(context).textTheme.headline4, ), ], ), ),
floatingActionButton: FloatingActionButton(
  onPressed: _incrementCounter,
  tooltip: 'Increment',
  child: const Icon(Icons.add), ), );
}

```

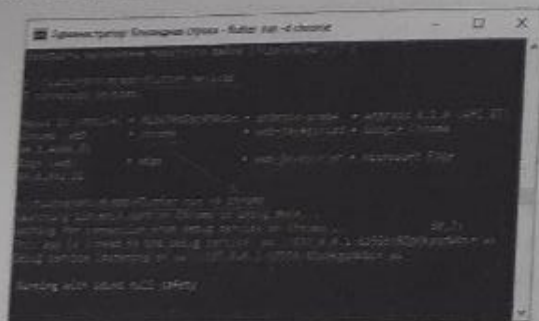
Dart dasturlash tilida bu yerda bir qancha sinflar aniqlangan bo'lib, ular amalda dasturni yaratadi.

Web-da dasturni ishga tushirish

Loyiha iOS va Web-da ishga tushirish uchun ham qo'llab-quvvatlanadi. Brauzerda web-ilovaga matn yozish uchun loyihani qanday ishga tushirishni ko'rib chiqamiz. Shunday qilib, yuqoridagi skrinshotda uchta "qurilma" borligi keltirilgan edi. Ulardan biri Google Chrome brauzeri bo'lib, buning uchun "chrome" deb nomlangan buyruqni kiritish lozim:

```
flutter run -d chrome
```

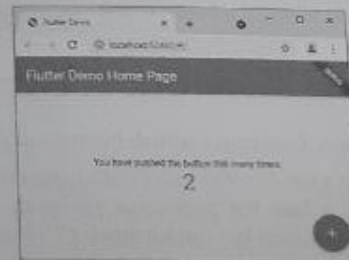
Ya'ni, loyihani ishga tushirish uchun yana bir xil *flutter run* buyrug'i kiritiladi. Faqat *-d* bayrog'idan foydalanib, loyiha ishga tushiriladigan qurilma belgilanadi. Qurilmaning matn yorlig'i ushbu bayroqqa o'tkaziladi - bu holda "chrome" (ya'ni Google Chrome brauzeri) va shunga mos ravishda web-ilova ishga tushadi.



7.13-rasm. Loyihani ishga tushirishda Google Chrome brauzerini tanlash

Shunga ko'ra, agar *Microsoft Edge*-da loyihani ishga tushirish kerak bo'lsa, tegishli matn belgisini taqdim etish kerak:
`flutter run -d edge`

Loyihani *web*-da ishga tushirganda, testlovchi web-server ishga tushiriladi, unda dastur o'rnatiladi va keyin ushbu ilovaga kiradigan web-brauzer ishga tushiriladi. Natijada, brauzerda *Android*-dagi kabi o'xshash mantiqqa ega deyarli bir xil interfeys ko'rinadi (7.14-rasm):



7.14-rasm. Web-da dasturni ishga tushirish

Ilovani qayta ishga tushirish

Agar loyiha *flutter run* buyrug'i yordamida ishga tushirilgan bo'lsa, dastur kodini o'zgartirish va dasturni qayta yuklash uchun buyruq satriga "r" belgisini kiritish mumkin (*Hot reload* deb ataladi). Ammo shuni ta'kidlash kerakki, qayta yuklash har doim ham ishlamasligi mumkin. Bunday holda, *flutter run* buyrug'ini qayta ishga tushirish mumkin va ilova shunga qarab o'zgaradi.

MacOS da birinchi ilovani yaratish

MacOS-da ilova yaratish *Windows*-dagi jarayonlardan unchalik farq qilmaydi. Avvalo, Flutter loyihalari uchun lokal diskda katalog yaratish lozim. Masalan, *Documents/flutter_src* katalogi. Terminalni ishga tushirib unda *cd* buyrug'i yordamida yaratilgan katalogga o'tiladi. Keyinchalik, quyidagi buyruq yordamida joriy terminal oynasi uchun *bin* papkasiga yo'l qo'shiladi:

```
export PATH="$PATH:[Flutter SDK joylashgan papkaga yo'l]/flutter/bin"
```

Keyin loyihani yaratish buyrug'ini kiritish lozim (7.15-rasm):

```
flutter create myapp
```

```

fluttersrc ~ -bash — 80x17
Last login: Sat Jan 30 16:54:41 on ttys000

The default interactive shell is now zsh.
To update your account to use zsh, please run `chsh -s /bin/zsh`.
For more details, please visit https://support.apple.com/kb/HT208050.
MacBook-Pro-Eugene:~ eugene$ cd /Users/eugene/Documents/fluttersrc
MacBook-Pro-Eugene:fluttersrc eugene$ export PATH="$PATH:/Users/eugene/flutter/bin"
MacBook-Pro-Eugene:fluttersrc eugene$ flutter create myapp
Creating project myapp...
myapp/ios/Runner.xcworkspace/contents.xcworkspacedata (created)
myapp/ios/Runner.xcworkspace/xcshearedata/IDEWorkspaceChecks.plist (created)
myapp/ios/Runner.xcworkspace/xcshearedata/WorkspaceSettings.xcsettings (created)
myapp/ios/Runner/Info.plist (created)
myapp/ios/Runner/Assets.xcassets/LaunchImage.imageset/LaunchImage@2x.png (created)

```

7.15-rasm. Loyihani yaratish buyrug'ini kiritish

Keyin `cd` buyrug'i yordamida `myapp` katalogiga o'tiladi. Keyinchalik, joriy terminal oynasi uchun `bin` papkasiga yo'lni qo'shiladi va loyihani ishga tushirish uchun quyidagi buyruq kiritiladi (7.16-rasm):

`flutter run`

```

myapp ~ -bash — 80x22
For more details, please visit https://support.apple.com/kb/HT208050.
MacBook-Pro-Eugene:~ eugene$ cd /Users/eugene/Documents/fluttersrc/myapp
MacBook-Pro-Eugene:myapp eugene$ export PATH="$PATH:/Users/eugene/flutter/bin"
MacBook-Pro-Eugene:myapp eugene$ flutter run
1 connected device.

 Nexus 5X (mobile) • 013a7061c970c1c • android-arm64 • Android 8.1.0 (API 27)
MacBook-Pro-Eugene:myapp eugene$ flutter run
Launching lib/main.dart on Nexus 5X in debug mode...
Checking the license for package Android SDK Platform 29 in /Users/eugene/Library/Android/sdk/licenses
License for package Android SDK Platform 29 accepted.
Preparing "Install Android SDK Platform 29 (revision: 5)".
"Install Android SDK Platform 29 (revision: 5)" ready.
Installing Android SDK Platform 29 in /Users/eugene/Library/Android/sdk/platform-29
"Install Android SDK Platform 29 (revision: 5)" complete.
"Install Android SDK Platform 29 (revision: 5)" finished.
Running Gradle task 'assembleDebug'...

```

7.16-rasm. Loyihani ishga tushirish buyrug'ini kiritish

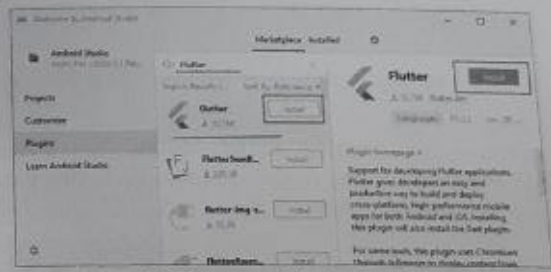
Android Studio-da dastur yaratish

Flutterni ishlab chiqish uchun ko'pincha Android Studio kabi ishlab chiqish muhiti tanlanadi. Kodni oddiy matn muharririda terish va uni konsolda kompilyatsiya qilish mumkin bo'lsa-da, ishlab chiqish muhiti mobil ilova yaratish jarayonini sezilarli darajada osonlashtiradi. Bundan tashqari, Android Studio Flutter-da nafaqat Android-ning o'zi, balki boshqa

qo'llab-quvvatlanadigan platformalar uchun ham ilovalar yaratish imkonini beradi.

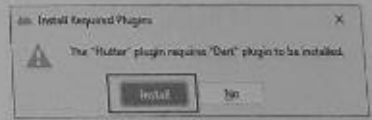
Android Studio bilan ishlash uchun, avvalo uni o'rnatish kerak. Uni <https://developer.android.com/studio> saytidan yuklab olish mumkin.

Odatda, Android Studio Flutter-ni qo'llab-quvvatlamaydi, shuning uchun tegishli plaginni o'rnatish kerak. Buni amalga oshirish uchun Android Studio-da, boshlang'ich ekranda *Plugins* bandi tanlanadi (yoki Android Studiyada, *File* menyusining *Settings* bo'limiga o'tiladi va keyin ochilgan oynada *Plugins* bandi tanlanadi). Plaginar panelida *Flutter plagini* topiladi (7.17-rasm):



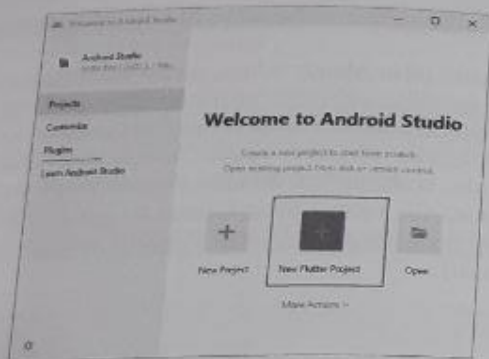
7.17-rasm. Plugins bandi

Kerakli plaginni qidirishni soddalashtirish uchun qidiruv paneliga "Flutter" so'zini kiritish mumkin va birinchi natija aynan o'rnatilishi kerak bo'lgan natija bo'ladi. Plaginni o'rnatishda Dart plaginini o'rnatishni so'ragan oyna ham paydo bo'ladi. Shuningdek, uni o'rnatish uchun *Install* tugmasini bosish lozim (7.18-rasm):



7.18-rasm. Dart plaginini o'rnatish

Plaginni o'rnatgandan so'ng, Android Studio-ni qayta ishga tushirish kerak bo'ladi. Qayta ishga tushirilgandan so'ng, Android Studio-ning boshlang'ich ekranida *New Flutter Project* tugmachasini ko'rish mumkin (7.19-rasm):



7.19-rasm. Android Studio-ning boshlang'ich ekranida *New Flutter Project* tugmachasini ko'rinishi

Flutter uchun loyiha yaratish uchun ushbu *New Flutter Project* tugmasi bosiladi. Muqobil sifatida Android Studio-da loyiha yaratish uchun menyu bandiga o'tish mumkin *File -> New -> New Flutter Project* (7.20-rasm):



7.20-rasm. Flutter uchun loyiha yaratishning muqobil varianti

Keyin yangi loyiha yaratish oynasi ochiladi. Chap tomonda Flutter elementini tanlash va markazda Flutter SDK yo'li maydonida Flutter SDK-ga yo'lni ko'rsatish lozim (7.21-rasm):



7.21-rasm. Flutter SDK ga yo'lni ko'rsatish

Keyingi oynada bir qator loyiha sozlamalarini kiritish lozim (7.22-rasm):



7.22-rasm. Loyiha sozlamalarini kiritish

Loyiha nomi maydonida loyihaga nom berish lozim. Shunday qilib, bu loyiha *hello_app* deb nomlanadi.

Project location maydonida, agar taklif qilingan standart joy qoniqarli bo'lmasa, loyiha manzilini o'zgartirish mumkin.

Description maydonida loyiha tavsifini belgilash mumkin.

Project type maydoni loyiha turini belgilaydi. Odatda *Application* qiymati o'rnatiladi (ya'ni loyiha ilova yaratish uchun mo'ljallangan). Ushbu qiymatni standart sifatida belgilanadi.

Organization maydonida ilovalar paketi nomini belgilash mumkin. Taklif qilingan nomni qoldirish yoki o'zgartirish mumkin. Misol uchun, com.doshanova.

Android language maydoni Android uchun tilni belgilaydi. Bu yerda ham standart qiymatni qoldirish mumkin – *Kotlin*.

iOS language maydoni iOS platformasi uchun tilni belgilaydi. Bu yerda ham standart qiymatni qoldirish mumkin – *Swift*.

Platforms maydonida loyiha yaratiladigan platformalarni belgilash mumkin. Odatda, Android va iOS punktlari belgilanadi, ammo boshqa mavjud platformalarni ham tanlash mumkin. Shunday qilib, yuqoridagi skrinshotda ko'rinib turibdiki, web uchun loyiha yaratishda "Web" bandini tanlash lozim.

Keyin barcha sozlamalarni o'ratgandan so'ng, loyihani to'g'ridan-to'g'ri yaratish uchun "Finish" tugmasi bosiladi. Yaratilgandan so'ng darhol Android Studio yaratilgan loyihani ochadi (7.23-rasm):



7.23-rasm. Android Studio-da yaratilgan yangi loyihani ko'rinishi

Yaratilgan loyiha konsolda *flutter create* buyrug'i yordamida yaratilgan tuzilishga ega bo'ladi. *main.dart* fayli haqiqiy dastur kodini o'z ichiga olgan Android Studio markazida ochiladi.

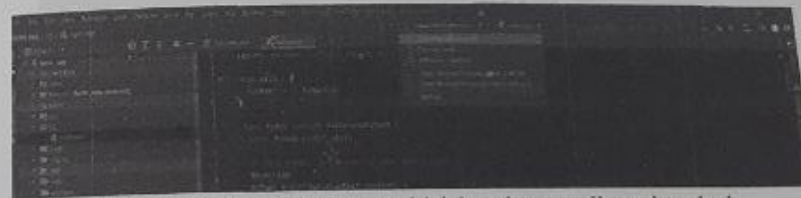
Shuningdek, Android qurilmani kompyuterga ulab (yoki emulyatorlardan foydalanish mumkin) va Android Studio panelida ilovani ishga tushirish uchun yashil uchburchak bosiladi (7.24-rasm).

Flutter Demo Home Page

You have pushed the button this many times
0

7.24-rasm. Android qurilmada ilovaning ko'rinishi

Xuddi shunday, Android Studio-da boshqa "qurilmalarda", masalan, web brauzerda loyihani ishga tushirish mumkin. Buning uchun asboblard panelida tegishli qurilmani tanlash kifoya (7.25-rasm):



7.25-rasm. Ilovani ishga tushirish uchun qurilmani tanlash

Visual Studio Code-da birinchi ilova yaratish

Bugungi kunda eng mashhur matn muharrirlaridan biri bu Visual Studio Code hisoblanadi va Google buning uchun Flutterda ishlab chiqish plaginini taqdim etadi. Uni o'rnatish uchun *View -> Extensions* menyu bandiga o'tish lozim (7.26-rasm).



7.26-rasm. View -> Extensions menyus bandiga o'tish

Shundan so'ng, kengaytmalar paneli ochiladi. Qidiruv maydoniga "Flutter" so'zini kiritiladi (7.27-rasm):



7.27-rasm. Flutter plaginini qidirish

Kerakli "Flutter" plagin topilgandan so'ng uni o'rnatish uchun "O'rnatish" tugmasi bosiladi.

Ushbu plagin Dart plaginiga bog'liq bo'lgani uchun Dart plagini ham avtomatik ravishda o'rnatiladi.

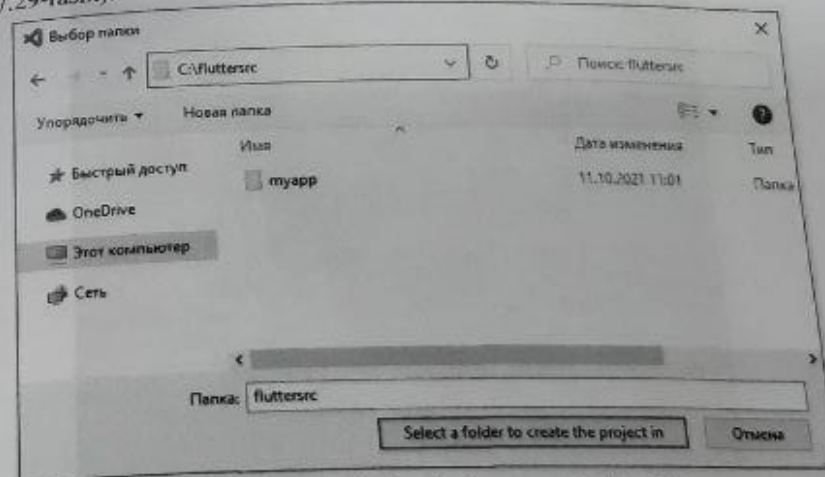
Plaginni o'rnatgandan so'ng, birinchi loyihani yaratamiz. Buning uchun menyuning View -> Command Palette bandiga o'tiladi. Ochilgan

qidiruv oynasida "Flutter" so'zi kiritiladi va natijalar orasidan Flutter: New Application Project tanlanadi (7.28-rasm).



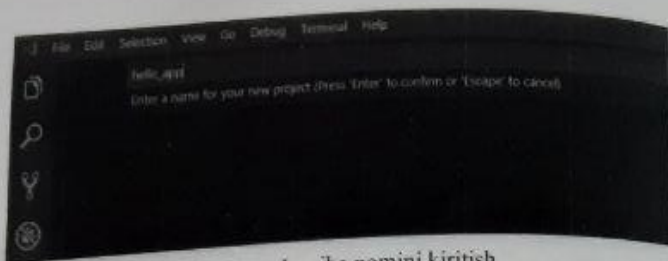
7.28-rasm. Flutter: New Application Project ni tanlash

Keyingi qadamda yangi oynada loyiha joylashgan papka tanlanadi (7.29-rasm):



7.29-rasm. Loyiha joylashgan papkani tanlash

Keyin yangi oynada loyiha nomi kiritiladi, masalan, "hello_app" va Enter tugmasi bosiladi (7.30-rasm).



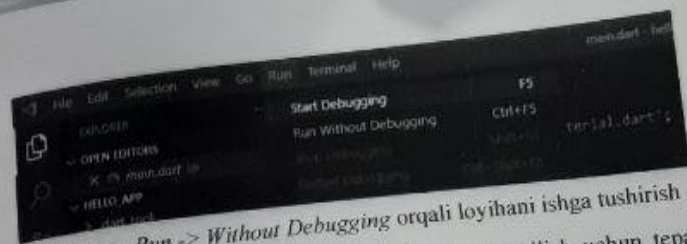
7.30-rasm. Loyiha nomini kiritish

Natijada, xuddi shunday tuzilishga ega bo'lgan loyiha yaratiladi. Va *main.dart* fayli muharrirda ochiladi (7.31-rasm):



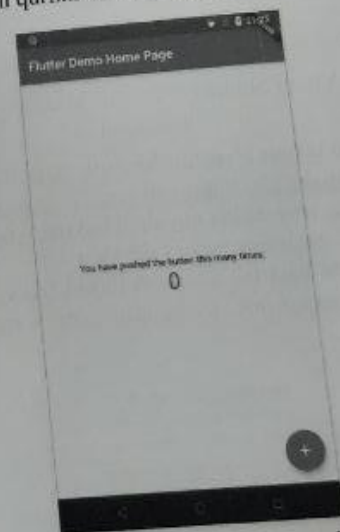
7.31-rasm. Loyihaning Visual Studio Code-dagi ko'rinishi

Ilovani ishga tushirish uchun Android mobil qurilmasini kompyuterga ulash lozim va Visual Studio Code-da *Run -> Start Debugging* menyusi bandiga o'tiladi. Loyihani ishga tushirishning muqobil varianti *Run -> Without Debugging* buyrug'ini bajarish kerak (7.32-rasm).



7.32-rasm. *Run -> Without Debugging* orqali loyihani ishga tushirish

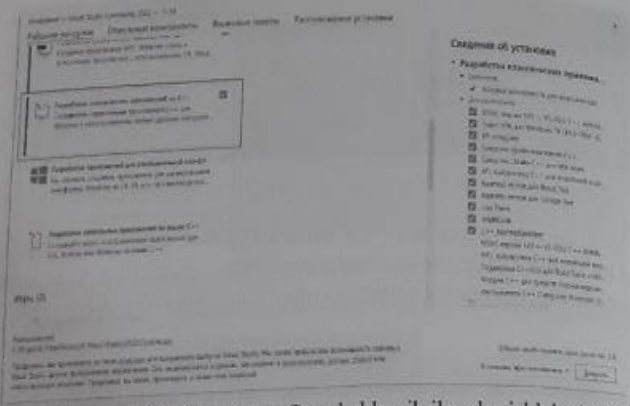
Natijada, ilova yig'iladi. VS Code-da otladka qilish uchun tepada panel ko'rsatiladi va pastda esa barcha nosozliklarni tuzatish ma'lumotlarini ko'rsatadi. Ilova esa mobil qurilmada ishga tushiriladi (7.33-rasm).



7.33-rasm. Ilovaning ishga tushishi

Flutter yordamida Windows uchun ilova yaratish
2.20 versiyasidan boshlab Flutter freymworki Windows OT uchun ilovalar yaratishni qo'llab-quvvatlaydi. Biroq, freymworkning o'ziga qo'shimcha ravishda, Windows OT-ga ilovalar yaratish uchun bir qator qadamlarni bajarish kerak:

Visual Studio o'rnatuvchisidan foydalanib, "C++ da klassik ilovalar ishlab chiqish" ni tanlash lozim (7.34-rasm).

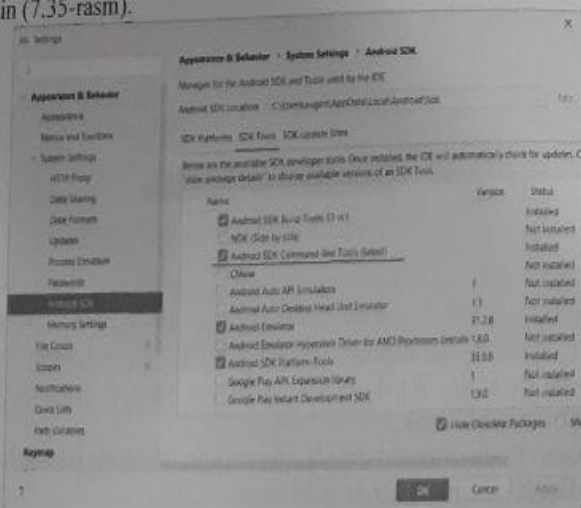


7.34-rasm. Visual Studioda "C++ da klassik ilovalar ishlab chiqish" ni tanlash

Visual Studio uchun o'ratuvchini quyidagi manzildan yuklab olish mumkin: <https://visualstudio.microsoft.com/ru/downloads/>.

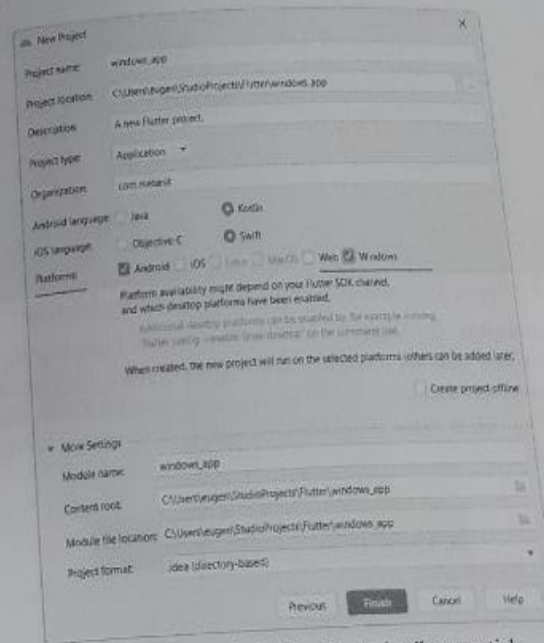
Shundan so'ng, terminalda quyidagi buyruqni ishga tushirish kerak: flutter config --enable-windows-desktop

Ushbu buyruqni bajarish uchun Android SDK buyruq qatori orqali Android SDK Command-line tools utilitasini o'rnatish kerak bo'lishi mumkin (7.35-rasm).



7.35-rasm. Android SDK Command-line tools ni o'rnatish

Agar utilita to'g'ri o'rnatilgan va sozlangan bo'lsa, Android Studio-da loyiha yaratishda Windows elementi Platformalar maydonida mavjud bo'ladi (7.36-rasm):

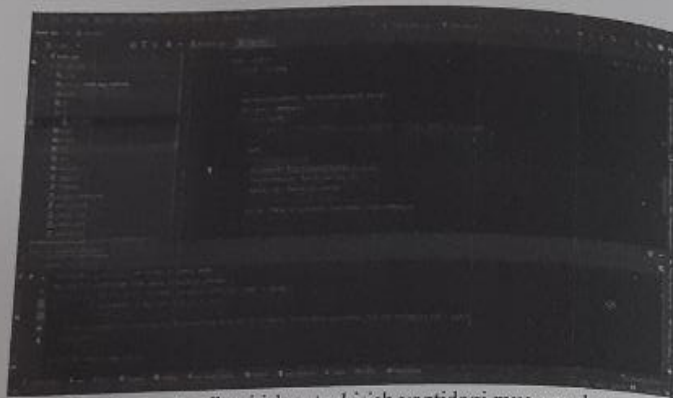


7.36-rasm. Android Studio-da loyiha yaratish

Windows OT-ga ilova yaratish uchun ushbu katakchani belgilash va yangi loyiha yaratish lozim. Loyihani yaratgandan so'ng, uning tarkibida Windows OT uchun mo'ljallangan fayllar uchun windows katalogi yaratiladi.

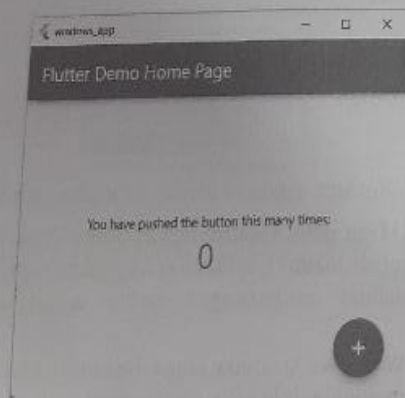
Loyihani Windows tizimida ishga tushirish uchun ishga tushirish platformalari maydonida Windows (desktop) ni tanlash va loyihani ishga tushirish lozim.

Ishga tushirish vaqtida app.so fayli ma'lum bir katalogda topilmasligi bilan bog'liq muammoga duch kelish mumkin (7.37-rasm):



7.37-rasm. Loyihani ishga tushirish vaqtidagi muammolar

Bu muammo o'rnatilgan Visual Studio versiyasi bilan bog'liq va bu holda, yechim sifatida `build\windows` katalogdan `app.so` faylini `..build\windows\runner\Debug\data` katalogiga nusxalash kerak. Shundan so'ng loyihani qayta ishga tushirganda, ekranda ilova oynasi ko'rsatiladi (7.38-rasm):



7.38-rasm. Ilovani ko'rinishi

Shu bilan bir qatorda, loyihani muqobil sifatida *Profil Mode* rejimida yoki *Release Mode* rejimida ishga tushirish mumkin.

7.2. Flutter-da vidjetlar bilan ishlash

Flutter ilovasining markaziy elementi vidjetlar hisoblanadi. Aslida, bu grafik interfeysni tashkil etuvchi vizual komponentlardir. Oddiy misolni ko'rib chiqamiz. Avvalo, yangi Flutter loyihasi yaratiladi (7.39-rasm).



7.39-rasm. Loyiha ko'rinishi

Odatda dastur kodi `lib/main.dart` faylida joylashadi. Ushbu faylni ochib uning tarkibini quyidagicha o'zgartiriladi:

```
import 'package:flutter/material.dart';
void main() {
  runApp(const MyApp());
}
class MyApp extends StatelessWidget {
  const MyApp({super.key});
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Demo',
      theme: ThemeData(
        colorScheme: ColorScheme.fromSeed(seedColor:
Colors.deepPurple),
        useMaterial3: true,
      ),
      home: const MyHomePage(title: 'Flutter Demo Home Page'),
    );
  }
}
```

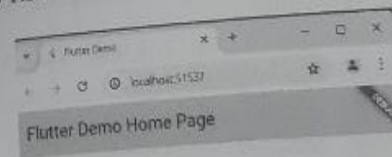
```

    } }
    class MyHomePage extends StatefulWidget {
      const MyHomePage({super.key, required this.title});
      final String title;
      @override
      State<MyHomePage> createState() => _MyHomePageState();
    }
    class _MyHomePageState extends State<MyHomePage> {
      int _counter = 0;
      void _incrementCounter() {
        setState(() {
          _counter++; });
      }
      @override
      Widget build(BuildContext context) {
        return Scaffold(
          appBar: AppBar(
            backgroundColor:
              Theme.of(context).colorScheme.inversePrimary,
            title: Text(widget.title),
          ),
          body: Center(
            child: Column(
              mainAxisAlignment: MainAxisAlignment.center,
              children: <Widget>[
                const Text(
                  'Bu misol - Mobil ilovalar ishlab chiqish - kitobi uchun
                    keltirilgan.'),
                Text(
                  'Flutterdan Salom!',
                  textDirection: TextDirection.ltr,
                  textAlign: TextAlign.center,
                  // '$_counter',
                  style: Theme.of(context).textTheme.headlineMedium,
                ),
              ],
            ),
            floatingActionButton: FloatingActionButton(
              onPressed: _incrementCounter,
              tooltip: 'Increment',
              child: const Icon(Icons.add),
            ),
          );
        }
      }
    }

```

Flutter ilovasining bajarilishi asosiy funksiyadan boshlanadi. Grafik interfeys yaratish uchun bu funksiya boshqa o'rnatilgan funksiyani chaqiradi - *runApp*(Vidjet ilovasi). U ma'lum bir vidjetni ekranga o'rnatadi.

Ya'ni, qurilma ekranida dastur ishga tushirganida bu ko'rinadi. O'rnatiladigan vidjet parametr sifatida *runApp()* funksiyasiga uzatiladi. Barcha vidjetlar *Widget* sinfidan meros bo'lib keladi. Bunday holda, *Text* vidjeti matnni ekranda ko'rsatish uchun mo'ljallangan *runApp* funksiyasiga o'tkaziladi. Ushbu sinf konstruktori orqali uning uchta xususiyati o'rnatiladi. Birinchi, "Flutterdan Salom!" qatori uzatiladigan matn. Shuningdek, matnning yo'nalishini belgilovchi va *TextDirection.ltr* (matn yo'nalishi chapdan o'ngga) qiymatini qabul qiluvchi *textDirection* xossasi va matnning gorizontal tekislanishini o'rnatuvchi va *TextAlign.center* qiymatini qabul qiluvchi *textAlign* xossasi ham o'rnatiladi (ya'ni matn markazga to'g'ri keladi). Kodni o'zgartirgandan so'ng, Android Studio test fayllari bilan jildni noto'g'ri deb ko'rsatishi mumkin, chunki test fayli o'zgartirilmagan dastur kodi o'zgartirildi. Ammo ba'zan bularni e'tiborsiz qoldirish mumkin. Bundan tashqari, *widget_test.dart* fayli butunlay o'chirilishi mumkin. Shunday qilib, oddiy Flutter ilovasini yaratdik. Natijada, dasturni ishga tushirganda, qurilmaning yuqori markazida matnni ko'rish mumkin (7.40-rasm):



Bu misol - Mobil ilovalar ishlab chiqish - kitobi uchun keltirilgan.
Flutterdan Salom!

7.40-rasm. Ilovaning ko'rinishi

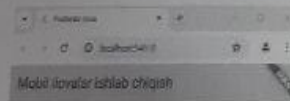
Ba'zi vidjetlar boshqa vidjetlarni o'z ichiga olishi mumkin va shu bilan daraxt shaklidagi ierarxik tizimni tashkil qiladi. Masalan, matnni vertikal ravishda ekranning o'rtasiga joylashtirish uchun *Align* vidjetidan foydalaniladi:

```
import 'package:flutter/material.dart';
void main() {
  runApp(
    Align(
      alignment: Alignment.center,
      child: Text(
        'Flutterdan Salom!',
        textDirection: TextDirection.ltr, ), ), ); }
```

Endi *runApp* funksiyasiga uzatiladigan ildiz vidjet *Align* vidjetidir. *Alignment* xususiyati ichki o'rnatilgan elementlarning tekislanishini o'rnatadi. Bunday holda, *Alignment.center* qiymati tekislash gorizontal va vertikal ravishda markazlashtirilishini bildiradi.

Boshqa xususiyat, *child*, ichma-ich o'rnatilgan vidjetni ifodalaydi - bu avval ishlatilgan *Text* vidjetidir.

Ilovani ishga tushirganda, uning ichida *Text* vidjeti bilan *Align* vidjetini ko'rish mumkin (7.41-rasm).



Flutterdan Salom!

7.41-rasm. Ilovaning ko'rinishi

Konteynerlar va komponentlarni boshqarish
Bitta vidjet yoki vidjetlar to'plamini ma'lum bir tarzda tashkil qilish va tartibga solish uchun maxsus vidjetlar qo'llaniladi, bular - vidjet - konteynerlar tartibini boshqaradigan joylashtirish konteynerlari. Flutter-da ushbu vidjetlar guruhi juda keng tarqalgan. Ulardan ba'zilarini ko'rib chiqamiz.

Align
Align vidjeti ichma-ich joylashgan elementni konteynerning ma'lum bir tomoniga nisbatan joylashtirish imkonini beradi. Odatda, u konteynerning butun kengligi va balandligi bo'ylab cho'zilib, uning barcha bo'sh joyini to'ldiradi. Vidjetni yaratish uchun quyidagi konstruktordan foydalaniladi:

```
Align({Key key, AlignmentGeometry alignment: Alignment.center, double widthFactor, double heightFactor, Widget child})
```

Konstruktordan ko'rinib turibdiki, uning barcha parametrlari ixtiyoriydir. Ichma-ich joylashgan vidjetning ma'lum bir chetiga tekislash qo'llaniladigan ichki elementni o'rnatish uchun *child* parametridan foydalaniladi - bu har qanday *Widget* obyekt, ya'ni har qanday vidjet bo'lishi mumkin.

Qo'shimcha parametrlar *widthFactor* va *heightFactor* - ichki o'rnatilgan yordamchi elementga nisbatan *Center* vidjetining kengligi va balandligidagi o'zgarish koeffitsientidir. Masalan, agar *widthFactor* 2,0 bo'lsa, u holda *Center* vidjetining kengligi 2,0 ga ko'paytirilgan *child* elementining kengligiga teng bo'ladi. Xuddi shu narsa *heightFactor* uchun ham amal qiladi, faqat u balandlikni o'zgartiradi.

Tekislanishni o'rnatish uchun *AlignmentGeometry* sinfini ifodalovchi va quyidagi qiymatlarni olishi mumkin bo'lgan *alignment* parametridan foydalaniladi:

- *Alignment.bottomCenter*: markazda gorizontal tekislash, konteynerning pastki chetida vertikal (pastki markaz). *Alignment* bilan bir xil (0,0, 1,0);

- *Alignment.bottomLeft*: chap chetiga gorizontal tekislash, konteynerning pastki chetiga vertikal tekislash (pastki chap). *Alignment* bilan bir xil (-1,0, 1,0);

- *Alignment.bottomRight*: o'ng chetiga gorizontal tekislash, konteynerning pastki chetiga vertikal tekislash (pastki o'ng). *Alignment* bilan bir xil (1,0, 1,0);

- *Alignment.center*: Gorizantal va vertikal markazni tekislash. *Alignment* bilan bir xil (0,0, 0,0);

- *Alignment.centerLeft*: chap chetiga gorizantal tekislash, konteyner markaziga vertikal tekislash. *Alignment* bilan bir xil (-1,0, 0,0);

- *Alignment.centerRight*: o'ng chetiga gorizantal tekislash, konteyner markaziga vertikal tekislash. *Alignment* bilan bir xil (1,0, 0,0);

- *Alignment.topCenter*: markazda gorizantal tekislash, konteynerning yuqori chetida (yuqori markazda) vertikal tekislash. *Alignment* bilan bir xil (0,0, -1,0);

- *Alignment.topLeft*: chap chetiga gorizantal tekislash, konteynerning yuqori chetiga vertikal tekislash (yuqori chap). *Alignment* bilan bir xil (-1,0, -1,0);

- *Alignment.topRight*: o'ng chetiga gorizantal tekislash, konteynerning yuqori chetiga vertikal tekislash (yuqori o'ng). *Alignment* bilan bir xil (1,0, -1,0).

Konstruktorda *Alignment* parametri *AlignmentGeometry* sinfini ifodalasa-da, bu yerda konstantalar *AlignmentGeometry*-dan meros bo'lib qolgan *Alignment* sinfini ifodalaydi.

Quyidagi misolda matnni chap markazda joylashtirish uchun *Align* vidjetidan foydalanilgan:

```
import 'package:flutter/material.dart';
void main() {
  runApp(Align(
    alignment: Alignment.centerLeft,
    child: Text(
      'Matn chapdan ongga ',
      textDirection: TextDirection.ltr, // matn chapdan o'ngga
      style: TextStyle(fontSize: 24) // shrift balandligi 24
    )
  ));
}
```

Fractional Offset

FractionalOffset sinfi *Alignment* sinfidan meros bo'lib, elementni joylashtirish uchun qo'shimcha imkoniyatlarni taqdim etadi. U quyidagi konstruktorga ega:

FractionalOffset(juft dx, double dy)

bu erda *dx* - gorizantal siljish va *dy* - vertikal siljish. Ofsetlar 0,0 dan 1,0 gacha bo'lgan kasrlarda ifodalanadi, masalan, *FractionalOffset* (1,0, 0,0) konteynerning yuqori o'ng burchagini, *FractionalOffset* (0,0, 1,0) esa pastki chap burchakni ifodalaydi. Shunga ko'ra, *FractionalOffset*(0,5, 0,5) markaz

(gorizantal va vertikal) hisoblanadi. Kerakli ofsetni ko'rsatib, elementni konteynerning ma'lum bir qismiga joylashtirish mumkin. Masalan:

```
import 'package:flutter/material.dart';
void main() {
  runApp(Align(
    alignment: FractionalOffset(0.2, 0.3),
    child: Text(
      'Hello Flutter',
      textDirection: TextDirection.ltr, // matn chapdan o'ngga
      style: TextStyle(fontSize: 20) // shrift balandligi 20
    )
  ));
}
```

FractionalOffset(0.2, 0.3) qiymati ichki o'rnatilgan *Text* elementining yuqori burchagi quyidagi koordinatalarda bo'lishini bildiradi: $X = Align_container_width * 0.2$, $Y = Align_container_height * 0.3$ (7.42-rasm).



7.42-rasm. Matnga *FractionalOffset*-ni qo'llash

Center

Center ichma-ich joylashtirilgan elementni markazlashtiradi. U *Align* sinfidan voris bo'lib o'tgan, uning funkcionalligini oladi. Odatda u konteynerning butun kengligi va balandligi bo'ylab cho'ziladi va uning barcha bo'sh joyini to'ldiradi. Vidjetni yaratish uchun quyidagi konstruktordan foydalaniladi:


```
Center({Key key, double widthFactor, double heightFactor, Widget child})
```

Markazlash qo'llaniladigan ichma-ich o'rnatilgan elementni o'rnatish uchun *child* parametri ishlatiladi. Har qanday *Widget* obyekti, ya'ni har qanday vidjet uning vazifasini bajarishi mumkin.

widthFactor va *heightFactor* qo'shimcha parametrlari *Center* vidjetining kengligi va balandligi uning ichma-ich joylashgan elementiga nisbatan o'zgarishi koeffitsiyentini o'rnatadi. Masalan, agar *widthFactor* 2,0 bo'lsa, u holda *Center* vidjetining kengligi 2,0 ga ko'paytirilgan ichki elementning kengligiga teng bo'ladi. Xuddi shu narsa *heightFactor* uchun ham amal qiladi, faqat u balandlikni o'zgartiradi.

Masalan, *Center* vidjeti yordamida kichik matnni markazlashtirish quyidagicha amalga oshiriladi (7.43-rasm):

```
import 'package:flutter/material.dart';
void main() {
  runApp(Center(
    child:Text(
      'Hello Flutter',
      textDirection: TextDirection.ltr, ) ) );
}
```



7.43-rasm. Matnga *Center* -ni qo'llash

Aslida bu kod quyidagiga teng bo'ladi:

```
import 'package:flutter/material.dart';
void main() {
  runApp(Align(
    alignment: Alignment.center,
    child:Text(
      'Hello Flutter!',
      textDirection: TextDirection.ltr ) ) );
}
```

Padding vidjeti ichma-ich o'rnatilgan element uchun otstup (chekinish) ni o'rnatish imkonini beradi. U quyidagi konstruktordan foydalanadi:

```
Padding({Key key, @required EdgeInsetsGeometry padding, Widget child})
```

Konstruktorda otstupni o'rnatish uchun majburiy parametr bo'lgan *padding* parametridan foydalaniladi. U *EdgeInsetsGeometry* sinfini ifodalaydi. Otstupni o'rnatish uchun ushbu sinf konstruktorlaridan birini ishlatish mumkin:

- *EdgeInsets.all(double value)*: Barcha to'rtta otstup uchun bitta *double* qiymatni o'rnatadi (chap, yuqori, o'ng va pastki);

- *EdgeInsets.fromLTRB(double left, double top, double right, double bottom)*: to'rt tomonning har biri uchun otstup qiymatini o'rnatadi;

- *EdgeInsets.fromWindowPadding(WindowPadding padding, double devicePixelRatio)*: *padding* parametriga mos keladigan bo'sh joyini belgilaydi;

- *EdgeInsets.only({double left: 0.0, double top: 0.0, double right: 0.0, double bottom: 0.0})*: To'rtta tomonning har birini nolga teng bo'lmagan otstup qiymatlarini o'rnatadi;

- *EdgeInsets.symmetric ({double vertical: 0.0, double horizontal: 0.0})*: vertikal yuqori va pastki otstupni o'rnatadi va *horizontal* chap va o'ng otstupni o'rnatadi.

Matnga hech qanday otstup qo'llanilmaydigan misol quyida keltirilgan:

```
import 'package:flutter/material.dart';
void main() {
  runApp(Align(
    alignment: Alignment.topCenter,
```

```

child: Text(
  'Hello Flutter',
  textDirection: TextDirection.ltr ) ) );
}

```

Dasturdan ko'rinib turganidek, ilova butun ekranni, jumladan, smartfonning yuqori panelini egallaydi, bu yerda turli ko'rsatkichlar va joriy vaqt joylashadi.

Quyidagi dasturda otstup ilovaning yuqori paneliga qo'llanilgan:

```

import 'package:flutter/material.dart';
void main() {
  runApp(Padding(
    padding: EdgeInsets.all(40),
    child: Align(
      alignment: Alignment.topCenter,
      child: Text(
        'Hello Flutter',
        textDirection: TextDirection.ltr
      ) ) ) );
}

```

Bunday holda, *Align* konteyneri matn bilan boshqa konteynerga – *Padding*-ga joylashtiriladi. *Padding* barcha to'rtta otstup uchun bitta qiymatni o'rnatadi - 40. Natijada, ilovadagi matn yuqori holat satridan 40 birlik pastga siljiydi.

EdgeInsets konstruktorlaridan foydalanib, turli tomonlar uchun otstupni o'rnatish mumkin. Masalan:

```

import 'package:flutter/material.dart';
void main() {
  runApp(Padding(
    padding: EdgeInsets.only(top: 40, bottom: 10, left: 10, right: 10),
    child: Align(
      alignment: Alignment.topCenter,
      child: Text(
        'Hello Flutter',
        textDirection: TextDirection.ltr ) ) ) );
}

```

ConstrainedBox

ConstrainedBox vidjeti o'rnatilgan vidjet joylashtirilgan to'rtburchaklar maydon (kenglik va balandlik) parametrlarini belgilaydi.

ConstrainedBox obyektini yaratish uchun quyidagi konstruktordan foydalaniladi:

```

ConstrainedBox({Key key, @required BoxConstraints constraints,
Widget child});

```

Constraints parametr cheklovlari *BoxConstraints* sinfini ifodalaydi, unda quyidagi konstruktorlar mavjud:

- *BoxConstraints*({double minWidth: 0.0, double maxWidth: double.infinity, double minHeight: 0.0, double maxHeight: double.infinity}); minimal kenglik (*minWidth*), maksimal kenglik (*maxWidth*), minimal (*minHeight*) va maksimal (*maxHeight*) balandlikni oladi;

- *BoxConstraints.expand*({double width, double height}); konteyner kengaytira oladigan kenglik va uzunlikni oladi;

- *BoxConstraints.loose*(*Size size*): *size* parametridan katta bo'lmasligi kerak bo'lgan konteyner yaratadi;

- *BoxConstraints.tight*(*Size size*): *size* parametri bilan aynan bir xil o'lchamda bo'lishi kerak bo'lgan konteyner yaratadi;

- *BoxConstraints.tightFor*({double width, double height}); konteyner yaratilishi kerak bo'lgan aniq kenglik va uzunlikni oladi;

- *BoxConstraints.tightForFinite*({double width: double.infinity, double height: double.infinity}); agar ular *double.infinity* ga teng bo'lmasa, aniq kenglik va uzunlik qiymatlarini oladi.

Ushbu turdagi konteyner vidjetni ma'lum bir hududga cheklash kerak bo'lganda ishlatiladi.

Masalan, *Text* vidjeti sukut bo'yicha konteynerning butun uzunligi va kengligi bo'ylab cho'ziladi:

```

import 'package:flutter/material.dart';
void main() {
  runApp(Center(
    child: Text(
      'Hello Flutter',
      textDirection: TextDirection.ltr,
      textAlign: TextAlign.center,
      style: TextStyle(fontSize: 26)
    ) ) );
}

```

Ilovada matn maydonini cheklash uchun *ConstrainedBox*-dan foydalaniladi:

```

import 'package:flutter/material.dart';

```

```

void main() {
  runApp(
    Center(
      child: ConstrainedBox(
        constraints: BoxConstraints.tightFor(width: 300, height: 100),
        child: Text(
          'Hello Flutter',
          textDirection: TextDirection.ltr,
          textAlign: TextAlign.center,
          style: TextStyle(fontSize: 26)
        )
      )
    )
  );
}

```

Bunday holda, *BoxConstraints.tightFor(width: 300, height: 100)* konstruktoridan foydalanib, *ConstrainedBox* maydoni va shunga mos ravishda uning kengligi 300 va balandligi 100 birlik bo'lgan to'rtburchak bilan chegaralanadi.

Agar vidjet tarkibi ko'proq joy egallasa, u ko'rinadigan maydonga qisqartiriladi. Masalan, ko'p qatorli matnni akslantirish quyidagicha amalga oshiriladi:

```

import 'package:flutter/material.dart';
void main() {
  runApp(
    Center(
      child: ConstrainedBox(
        constraints: BoxConstraints.tightFor(width: 320, height: 80),
        child: Text(
          'Assalomu aleykum\nBu misol mobil ilovalar ishlab
chiqish, noquv qollanmasi uchun keltirilgan.\nBu qator ilovada
korinmasligi mumkin.',
          textDirection: TextDirection.ltr,
          style: TextStyle(fontSize: 22)
        )
      )
    )
  );
}

```

Bunday holda, oxirgi qator vidjet uchun ajratilgan maydonga to'g'ri kelmaydi, shuning uchun u ilova ishga tushganda ko'rinmaydi.

Container

Container shunday vidjetni ifodalaydiki, u faqat bitta ichma-ich joylashtirilgan elementni o'z ichiga olishi mumkin, lekin ayni paytda ichma-ich o'rnatilgan vidjetlarning fonini, joylashishini va hajmini sozlash uchun qo'shimcha imkoniyatlarni taqdim etadi. Asosan, *Container* boshqa

vidjetlarning imkoniyatlarini birlashtiradi – *Padding*, *ConstrainedBox*, *Align*.

Container konstruktori ekranning ma'lum jihatlarini sozlash imkonini beruvchi bir nechta parametrlarni qabul qiladi:

```

Container({Key key, AlignmentGeometry alignment,
EdgeInsetsGeometry padding, Color color, Decoration decoration,
Decoration foregroundDecoration, double width, double height,
BoxConstraints constraints, EdgeInsetsGeometry margin, Matrix4
transform, Widget child, Clip clipBehavior: Clip.none})

```

Bu konstruktorning ba'zi parametrlari quyida keltirilgan:

– *key*: element kaliti;

– *alignment*: *Align* vidjetidagi tekislash sozlamalariga o'xshash *AlignmentGeometry* obyektini sifatida ichki o'rnatilgan element uchun tekislash sozlamalari;

– *padding*: *Padding* vidjetidagi o'yustupni o'rnatishga o'xshash konteyner chegaralaridan ichki o'rnatilgan elementni cheklash uchun sozlash;

– *color*: konteyner rangi;

– *constraints*: uzunlik va kenglik cheklovlari *BoxConstraints* obyektini sifatida o'rnatilgan vidjetga qo'llaniladi. *ConstrainedBox*-da o'lichamlarni o'rnatishga o'xshash bo'ladi;

– *margin*: joriy *Container* vidjetining chetlarini to'ldirish parametrini o'rnatishga o'xshash tashqi konteyner chegaralaridan o'rnatadi;

– *width*: konteyner kengligi;

– *height*: konteyner balandligi.

Quyidagi dasturda oddiy *Container* vidjetini yaratilgan:

```

import 'package:flutter/material.dart';
void main() {
  runApp(
    Container(
      color: Colors.lightBlueAccent,
      alignment: Alignment.center,
      child: Text(
        'Hello Flutter',
        textDirection: TextDirection.ltr
      )
    )
  );
}

```

Bunday holda, rang sifatida *Colors.lightBlueAccent* qiymati bilan tavsiflangan o'rnatilgan rang (och ko'k rang tusi) ishlatiladi. *Alignment.center* qiymati pastki vidjetni markazlashtirish uchun ishlatilgan.

Margin va padding otstuplari
Dasturda *Margin* va *padding* otstuplaridan foydalanish quyidagi misolda keltirilgan:

```
import 'package:flutter/material.dart';  
void main() {  
  runApp(Container(  
    color: Colors.lightBlue,  
    alignment: Alignment.topLeft,  
    padding: EdgeInsets.all(40),  
    margin: EdgeInsets.only(top:30),  
    child: Text(  
      'Hello Flutter',  
      textDirection: TextDirection.ltr  
    ) ) );  
}
```

Bunday holda, otstup qiymati yuqoridagi 30 birlikdan iborat bo'lgan chekinishga o'rnatiladi. Ya'ni, *Container* vidjeti ekranning yuqori chegarasidan 30 birlik pastda joylashgan bo'ladi. Shuning uchun, smartfon ekranining yuqori qismida *Container* vidjeti bilan to'ldirilmagan qora chiziqni ko'rish mumkin. *Container* chegaralariga nisbatan o'rnatilgan *Text* vidjeti uchun otstup ham 40 birlikka o'rnatiladi.

Column

Column konteyneri elementlarni vertikal ravishda ustunga joylashtiradi. Vidjet yaratish uchun quyidagi konstruktordan foydalanish lozim:

```
Column({Key key, MainAxisAlignment mainAxisAlignment:  
MainAxisAlignment.start,      MainAxisAlignment mainAxisAlignment:  
MainAxisAlignment.max,      CrossAxisAlignment crossAxisAlignment:  
CrossAxisAlignment.center, TextDirection textDirection, VerticalDirection  
verticalDirection: VerticalDirection.down, TextBaseline textBaseline,  
List<Widget> children: const []})
```

Konstruktor quyidagi parametrlarga ega:

- *key*: vidjet kaliti;
- *mainAxisAlignment*: vertikal tekislashni o'rnatadi;
- *mainAxisSize*: asosiy o'q bo'ylab vidjet egallagan joyni belgilaydi;
- *crossAxisAlignment*: gorizontal tekislashni o'rnatadi;
- *textDirection*: ichki joylashtirilgan elementlarning gorizontal tartibini belgilaydi;

- *verticalDirection*: ichki joylashtirilgan elementlarning vertikal tartibini aniqlaydi;

- *textBaseline*: elementlarni tekislash uchun asosiy chiziqni o'rnatadi;

- *children*: ichki o'rnatilgan elementlar to'plami;

Column vidjeti uchun oddiy dastur quyida keltirilgan:

```
import 'package:flutter/material.dart';
```

```
void main() {  
  runApp(Container(  
    padding: EdgeInsets.all(30),  
    color: Colors.teal,  
    child: Column(  
      children: <Widget>[  
        Text('Java',  
          textDirection: TextDirection.ltr),  
        Text('Kotlin',  
          textDirection: TextDirection.ltr),  
        Text('Swift',  
          textDirection: TextDirection.ltr),  
        Text('Dart',  
          textDirection: TextDirection.ltr)  
      ], ) ) );  
}
```

Column vidjeti prokrutkani qo'llab-quvvatlamaydi. Shuning uchun, agar ko'rinadigan bo'shliqdan tashqariga chiqadigan vidjetlarga kirishni ta'minlash kerak bo'lsa, unda *Column* o'rniga boshqa konteynerdan, masalan, *ListView*-dan foydalanish yaxshiroqdir.

CrossAxisAlignment

Column konstruktoridagi *crossAxisAlignment* parametri ichma-ich o'rnatilgan vidjetlar kesishuvchi o'q bo'ylab qanday joylashishini belgilaydi. Ushbu parametr quyidagi qiymatlarni olishi mumkin:

CrossAxisAlignment.center: ichma-ich joylashtirilgan elementlarni gorizontal ravishda markazlashtiradi;

CrossAxisAlignment.end: ichma-ich joylashtirilgan elementlarni gorizontal o'qning oxiriga joylashtiradi. Ustunda *textDirection* konstruktorining boshqa parametri *TextDirection.ltr* bo'lsa (ya'ni matn chapdan o'ngga), u holda elementlar o'ng tomonga tekislanadi. Agar u *TextDirection.rtl*-ga teng bo'lsa (ya'ni, matn o'ngdan chapga), u holda elementlar chapga tekislanadi;

CrossAxisAlignment.start: kesishuvchi o'q boshida joylashgan elementlarni joylashtiradi. Ustunda *textDirection* konstruktorining boshqa parametri *TextDirection.ltr* bo'lsa (ya'ni matn chapdan o'ngga), u holda elementlar chap tomonga tekislanadi;

CrossAxisAlignment.stretch: ustun konteynerining to'liq kengligi bo'ylab ichna-ich joylashgan elementlarni cho'zib beradi;

CrossAxisAlignment.baseline: ichna-ich joylashgan elementlarni asosiy chiziq bo'ylab tekislaydi, ya'ni kesishgan o'q bo'yicha. Ustun konteyneri uchun u aslida *CrossAxisAlignment.start* bilan bir xil. *textBaseline* parametrini o'rnatishni talab qiladi.

Oldingi misolda, *CrossAxisAlignment.center* sukut bo'yicha qo'llaniladi, ya'ni barcha ichki o'rnatilgan vidjetlar markazlashtirilgan. Quyidagi dasturda boshqa qiymatdan foydalanilgan:

```
import 'package:flutter/material.dart';
void main() {
  runApp(Container(
    padding: EdgeInsets.all(30),
    color: Colors.teal,
    child: Column(
      textDirection: TextDirection.ltr,
      crossAxisAlignment: CrossAxisAlignment.start,
      children: <Widget>[
        Text('Java',
          textDirection: TextDirection.ltr),
        Text('Kotlin',
          textDirection: TextDirection.ltr),
        Text('Swift',
          textDirection: TextDirection.ltr),
        Text('Dart',
          textDirection: TextDirection.ltr)
      ],
    ));
}
```

CrossAxisAlignment qiymati yordamida *CrossAxisAlignment.start* qiymati konteynerning chap chetiga tekislanadi. Biroq, matn yo'nalishi har xil bo'lishi mumkinligi sababli - o'ng va chap tomon, *textDirection: TextDirection.ltr* matn yo'nalishini ham ko'rsatish kerak. Matnning yo'nalishiga qarab, boshi chap chekka (chapdan o'ngga) yoki o'ng chetiga (o'ngdan chapga) bo'lishi mumkin.

Text

Text vidjeti matn qatorini ko'rsatish uchun mo'ljallangan. Vidjet yaratish va uni o'rnatish uchun quyidagi konstruktordan foydalanish mumkin:

```
Text(String data, {Key key, TextStyle style, StrutStyle strutStyle,
TextAlign textAlign, TextDirection textDirection, Locale locale, bool
softWrap, TextOverflow overflow, double textScaleFactor, int maxLines,
String semanticsLabel, TextWidthBasis textWidthBasis,
TextHeightBehavior textHeightBehavior})
```

Text vidjetning asosiy parametrlari quyidagilar:

- *text*: vidjetning matni;
- *style*: matn rangi, fon rangi, shrift o'lehami va boshqalar kabi matn uslubini belgilaydi. *TextStyle* sinfining obyektini ifodalaydi;
- *strutStyle*: asosiy chiziqqa nisbatan minimal chiziq balandligini o'rnatadi. *StrutStyle* sinfini ifodalaydi;
- *textAlign*: gorizontal tekislashni o'rnatadi. *TextAlign* ro'yxatini ifodalaydi va quyidagi qiymatlarni olishi mumkin:
 - *TextAlign.center*: markazga tekislash;
 - *TextAlign.left*: konteynerning chap chetiga tekislash;
 - *TextAlign.right*: konteynerning o'ng chetiga tekislash;
 - *TextAlign.justify*: matnni chiziqning butun uzunligi bo'ylab cho'zish;
 - *TextAlign.end*: satr oxiridagi tekislash. Chapdan o'ngga matn uchun bu konteynerning o'ng qirrasi va o'ngdan chapga matn uchun bu konteynerning chap chetidir;
 - *TextAlign.start*: satr boshida tekislash. Chapdan o'ngga matn uchun bu konteynerning chap qirrasi va o'ngdan chapga matn uchun bu konteynerning o'ng tomonidir.
- *textDirection*: matn yo'nalishini o'rnatadi. *TextDirection* ro'yxatini ifodalaydi. Asosiy qiymatlar: *TextDirection.ltr* (matn chapdan o'ngga) va *TextDirection.rtl* (o'ngdan chapga);
- *locale*: mahalliy yoki taxminan tilni o'rnatadi. *Locale* sinfining obyektini ifodalaydi;
- *softWrap*: test belgilangan uzunlikka yetganda keyingi qatorga o'tish o'tmasligini aniqlaydi;
- *overflow*: matn mavjud uzunlikdan oshib ketganda qanday qisqartirilishini aniqlaydi. *TextOverflow*-ni ifodalaydi;
- *textScaleFactor*: har bir mantiqiy piksel uchun siljish piksellar sonini belgilaydi;

- *maxLines*: matn satrlarining maksimal sonini belgilaydi;
- *textWidthBasis*: matn kengligi qanday o'lanishini belgilaydi, *TextWidthBasis*-ni ifodalaydi;
- *textHeightBehavior*: *TextStyle.height* qiymati matnning birinchi va oxirgi satrlariga qanday qo'llanilishini aniqlaydi, *TextHeightBehavior* sinfini ifodalaydi.

Birinchi *String* parametri talab qilinadi va ko'rsatiladigan haqiqiy xabarni ifodalaydi. Qolgan parametrlar ixtiyoriy hisoblanadi. Quyida *Text* vidjetining bir qator xususiyatlarini ifodalovchi dastur kodi keltirilgan:

```
import 'package:flutter/material.dart';
void main() {
  runApp(Container(
    padding: EdgeInsets.only(top:25, left:10, right:10),
    color: Colors.white,
    child: Text("Hello Flutter",
      textDirection: TextDirection.ltr,
      textAlign: TextAlign.center,
      style: TextStyle(color: Colors.green,
        fontSize: 26,
        backgroundColor: Colors.black87 )
    ) ) );
}
```

Matn stili

Matn uslubi *TextStyle* sinfini ifodalovchi *style* parametri bilan tavsiflanadi. Ushbu sinf ko'plab matnlarni ko'rsatish sozlamalarini o'rnatish imkonini beradi. U quyidagi konstruktorga ega:

```
TextStyle({bool inherit: true, Color color, Color backgroundColor,
double fontSize, FontWeight fontWeight, FontStyle fontStyle, double
letterSpacing, double wordSpacing, TextBaseline textBaseline, double
height, Locale locale, Paint foreground, Paint background, List<Shadow>
shadows, List<FontFeature> fontFeatures, TextDecoration decoration,
Color decorationColor, TextDecorationStyle decorationStyle, double
decorationThickness, String debugLabel, String fontFamily, List<String>
fontFamilyFallback, String package})
```

Bu konstruktorning ba'zi parametrlari quyida keltirilgan:

- *inherit*: berilgan uslub asosiy vidjetning uslub xususiyatlarini meros qilib olish-olmasligini bildiradi;
- *color*: matn rangi;

- *backgroundColor*: matnning fon rangi;
- *fontSize*: shrift hajmi;
- *fontWeight*: shrift qalinligi. *FontWeight* sinfini ifodalaydi va uning konstantalaridan birini qiymat sifatida qabul qilishi mumkin: *FontWeight.w100*, *FontWeight.w200*, *FontWeight.w300*, *FontWeight.w400* (*FontWeight.bold* ga ekvivalent), *FontWeight.w500*, *FontWeight.w600*, *FontWeight.w700*, *FontWeight.w800* va *FontWeight.w900*. Konstantadagi raqamli qiymat qanchalik katta bo'lsa, shrift mos ravishda qalinroq bo'ladi;
- *fontStyle*: shrift stilini o'rnatadi. *FontStyle* ro'yxatini ifodalaydi va quyidagi qiymatlarni qabul qilishi mumkin: *FontStyle.normal* va *FontStyle.italic* (kursiv shrift);
- *letterSpacing*: so'zlar orasidagi masofani o'rnatadi, manfiy qiymat so'zlarni bir-biriga yaqinlashtiradi;
- *wordSpacing*: belgilar orasidagi masofani o'rnatadi, salbiy qiymat belgilarni bir-biriga yaqinlashtiradi;
- *textBaseline*: matnning tayanch chizig'i;
- *height*: matn elementining balandlik koeffitsiyenti. Haqiqiy balandlikni aniqlash uchun bu koeffitsient shrift balandligi *fontSize* bilan ko'paytiriladi;
- *locale*: matnning mahalliy yoki til turini o'rnatadi;
- *foreground*: matnni to'ldirish uchun rasm yoki *Paint* obyektini belgilaydi;
- *background*: tasvir yoki *Paint* obyektini vidjet foni sifatida o'rnatadi;
- *shadows*: shrift uchun soyalarni *Shadow* obyektlari to'plami sifatida o'rnatadi;
- *decoration*: matnni bezash (chiziq, tagiga yoki ustiga chizish). *TextDecoration* sinfidan quyidagi konstantalarni qiymat sifatida qabul qilishi mumkin:
 - o *TextDecoration.lineThrough*: chizilgan matn;
 - o *TextDecoration.overline*: matnning tagiga chizish;
 - o *TextDecoration.underline*: tagiga chizilgan;
 - o *decorationColor*: bezak rangi;
 - o *decorationStyle*: bezatish uslubi. *TextDecorationStyle* ro'yxatini ifodalaydi va quyidagi qiymatlarni qabul qilishi mumkin:
 - o *TextDecorationStyle.dashed*: punktir chiziq;
 - o *TextDecorationStyle.dotted*: nuqtalar;
 - o *TextDecorationStyle.double*: ikkitalik chizish;
 - o *TextDecorationStyle.solid*: oddiy chiziq;

- o `TextDecorationStyle.wavy`: to'liqlik chiziq;
- `decorationThickness`: bezak qalinligi;
- `fontFamily`: ishlatiladigan shrift nomi. Odatda Android faqat "Roboto" shriftidan foydalanadi.

Quyidagi dasturda bezaklarni foydalanilishi keltirilgan:

```
import 'package:flutter/material.dart';
void main() {
  runApp(Container(
    padding: EdgeInsets.only(top:25, left:10, right:10),
    color: Colors.teal,
    child: Column(children: <Widget>[
      Text("Hello Flutter",
        textDirection: TextDirection.ltr,
        style: TextStyle(
          fontSize: 26,
          decoration: TextDecoration.lineThrough,
          decorationStyle: TextDecorationStyle.double ),
        Text("Hello Flutter",
          textDirection: TextDirection.ltr,
          style: TextStyle(
            fontSize: 26,
            decoration: TextDecoration.underline,
            decorationStyle: TextDecorationStyle.wavy,
            decorationColor: Colors.blue,
            decorationThickness: 2 ),
        Text("Hello Flutter",
          textDirection: TextDirection.ltr,
          style: TextStyle(
            fontSize: 26,
            decoration: TextDecoration.underline,
            decorationStyle: TextDecorationStyle.dotted,
            decorationColor: Colors.red,
            decorationThickness: 3 )) ) ) );
}
```

Ko'p qatorli matn, matni keyingi qatorga o'tkazish va kesish
Vidjet matni unga ajratilgan joyga sig'masligi mumkin va bu holat uchun turli xil strategiyalar mavjud. Shunday qilib, *overflow* parametri matn

qatorini tugatish formatini o'rnatishga imkon beradi. U *TextOverflow* ro'yxatini ifodalaydi va quyidagi qiymatlarni olishi mumkin:

- *TextOverflow.clip*: matn konteyner uzunligiga mos ravishda kesiladi;
- *TextOverflow.ellipsis*: tarkibdagi matnning oxiriga ellips qo'shadi;
- *TextOverflow.fade*: matn oxiri o'chib ketadi;
- *TextOverflow.visible*: matn konteyner tashqarisida ko'rinadi.

Masalan, uzun matndan keyin ko'pnuqtani ishlatilishi quyidagi dasturda keltirilgan:

```
import 'package:flutter/material.dart';
void main() {
  runApp(Container(
    padding: EdgeInsets.only(top:25, left:10, right:10),
    color: Colors.white,
    child: Text("Emulyator ishga tushaman deguncha ",
      textDirection: TextDirection.ltr,
      style: TextStyle(color: Colors.black87, fontSize: 20),
      overflow: TextOverflow.ellipsis ) );
}
```

SoftWrap parametri matni keyingi qatorga o'tkazish imkonini beradi (agar *true* bo'lsa). Odatda, u *false* qiymatga ega bo'ladi, ya'ni matn keyingi qatorga o'tkazilmaydi.

```
import 'package:flutter/material.dart';
void main() {
  runApp(Container(
    padding: EdgeInsets.only(top:25, left:10, right:10),
    color: Colors.white,
    child: Text("Emulyator ishga tushaman deguncha, "
      "bir qancha jarayonlar bajariladi, buni terminalda kuzatish mumkin ",
      textDirection: TextDirection.ltr,
      style: TextStyle(color: Colors.black87, fontSize: 20),
      softWrap: true ) ) );
}
```

maxLines parametri yordamida konstruktordagi qatorlar sonini aniqlash ham mumkin. Test avtomatik ravishda keyingi qatorga o'tadi (to'liq maksimal qatorlar sonini to'ldirguncha).

```
import 'package:flutter/material.dart';
void main() {
  runApp(Container(
    padding: EdgeInsets.only(top:25, left:10, right:10),
```

```

color: Colors.white,
child: Text("Emulyator ishga tushaman deguncha, "
" bir qancha jarayonlar bajariladi, buni terminalda kuzatish mumkin ",
textDirection: TextDirection.ltr,
style: TextStyle(color: Colors.black87, fontSize: 20),
maxLines: 4 ) ) );
}

```

RichText

RichText vidjeti matni turli uslublar bilan ko'rsatish imkonini beradi. Bunga *RichText*-dagi har bir alohida matn qismi *TextSpan* obyektini ifodalaganligi sababli erishiladi.

RichText yaratish uchun quyidagi konstruktordan foydalaniladi:

```

RichText({Key key, @required InlineSpan text, TextAlign textAlign:
TextAlign.start, TextDirection textDirection, bool softWrap: true,
TextOverflow overflow: TextOverflow.clip, double textScaleFactor: 1.0, int
maxLines, Locale locale, StrutStyle strutStyle, TextWidthBasis
textWidthBasis: TextWidthBasis.parent, TextHeightBehavior
textHeightBehavior})

```

Konstruktor parametrlarining aksariyati *Text* sinfi konstruktor parametrlari bilan bir xil. Shunga ko'ra, ularni *Text* vidjetini yaratish va sozlashda bo'lgani kabi ishlatish mumkin va faqat farq qiladigan parametrlarni ko'rib chiqish mumkin. Birinchidan, matn *InlineSpan* obyektini ifodalovchi matn parametri yordamida aniqlanadi. Aslida, qoida tariqasida, ushbu parametr *TextSpan* sinfining obyektini (*InlineSpan*-dan meros bo'lib qolgan) tomonidan uzatiladi. Aslida *TextSpan* matnning bir qismidir. *TextSpan* yaratish uchun quyidagi konstruktordan foydalaniladi:

```

TextSpan({String text, List<InlineSpan> children, TextStyle style,
GestureRecognizer recognizer, String semanticsLabel})

```

Birinchi parametr, matn, obyektning haqiqiy matnini ifodalaydi. Bundan tashqari, *children* parametridan foydalanib, matni ham o'z ichiga olishi mumkin bo'lgan qo'shimcha *InlineSpan* obyektlarini (jumladan, *TextSpan* obyektlarini) joylashtirish mumkin. Uchinchi parametr - *style* - matn uchun muayyan uslubni o'rnatish imkonini beradi. *RichText* vidjeti qo'llanilgan dastur quyida keltirilgan:

```

import 'package:flutter/material.dart';
void main() {
runApp(Container(
padding: EdgeInsets.only(top:25, left:10, right:10),

```

```

color: Colors.teal,
child: RichText(
textDirection: TextDirection.ltr,
text: TextSpan(
text: "Hello Flutter",
style: TextStyle(fontSize: 20),
children: <TextSpan>[
TextSpan(text: " from ", style: TextStyle(color: Colors.red)),
TextSpan(text: "Tuit.uz", style: TextStyle(fontWeight:
FontWeight.bold)),
], ) ) );
}

```

Bunday holda, *RichText*-dagi barcha matn ichma-ich o'rnatilgan *TextSpan* vidjetida inkapsulyatsiya qilingan bo'lib, u o'z navbatida boshqa *TextSpan* obyektini o'z ichiga oladi. Darhaqiqat, ikkala *TextSpan*-dan olingan matn oddiygina asosiy *TextSpan*-dan matnga qo'shiladi.

Asosiy *TextSpan* uslubi avtomatik ravishda barcha *TextSpan* obyektlari uchun tarqaladi. Shunday qilib, yuqoridagi misolda shrift balandligi uslubini o'rnatish ko'rsatilgan: *TextStyle(fontSize: 20)* aslida barcha *TextSpan* obyektlari uchun amal qiladi. Har bir kichik *TextSpan* darajasida maxsus uslubni belgilash yoki asosiy *TextSpan* sinfidan meros qilib olingan uslub sozlamalarini o'zgartirish mumkin.

Stack

Stack konteyneri ba'zi elementlarni boshqalarning ustiga joylashtirish imkonini beradi. *Stack* vidjetini yaratish uchun quyidagi konstruktordan foydalanilad:

```

Stack({Key key, AlignmentGeometry alignment:
AlignmentDirectional.topStart, TextDirection textDirection, StackFit fit:
StackFit.loose, Overflow overflow: Overflow.clip, Clip clipBehavior:
Clip.hardEdge, List<Widget> children: const <Widget>{}})

```

Konstruktor quyidagi parametrlarga ega:

- *key*: vidjet kaliti;
- *alignment*: ichki o'rnatilgan vidjetlar o'rmini o'rnatadi;
- *textDirection*: ichma-ich joylashtirilgan elementlarning gorizontalar tartibini belgilaydi;
- *fit*: o'rnatilgan vidjetlar uchun o'lchamlarni belgilaydi;
- *overflow*: ichki o'rnatilgan kontentni kesish yoki kesish kerakligini belgilaydi;

- *clipBehavior*: ichki oʻrnatilgan elementlar qanday qirqib olinishini belgilaydi;

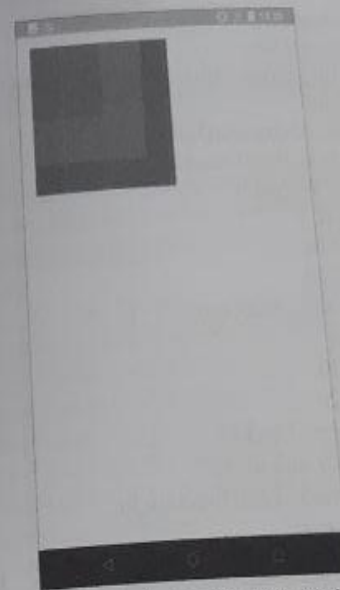
- *children*: ichma-ich oʻrnatilgan elementlar toʻplami.

Konteyner vidjetlari toʻplamini oʻz ichiga oladigan *Stack* vidjeti qoʻllanilgan dastur quyida berilgan:

```
import 'package:flutter/material.dart';
void main() {
  runApp(Container(
    color: Colors.white,
    padding: EdgeInsets.only(top:40, bottom: 10, left: 20, right: 20),
    child: Stack(
      textDirection: TextDirection.ltr,
      children: <Widget>[
        Container(
          width: 200,
          height: 200,
          color: Colors.blueGrey,
        ),
        Container(
          width: 160,
          height: 160,
          color: Colors.cyan,
        ),
        Container(
          width: 100,
          height: 100,
          color: Colors.blue,
        ),
      ],
    ));
}
```

Bunday holda, *Stack* ketma-ket bir-birining ustiga chiqadigan uchta konteyner elementini oʻz ichiga oladi (7.43-rasm).

Shuni taʼkidlash kerakki, bu holda *textDirection* xususiyatini ham oʻrnatish kerak. Odatda ichma-ich joylashgan vidjetlar *Stack* elementining yuqori chap burchagida joylashgan (dastlabki tekislash xususiyati *AlignmentDirectional.topStart*-ga oʻrnatiladi), lekin tabiiyki, *AlignmentGeometry* turini ifodalovchi *alignment* xususiyati bilan oʻrnatilgan joyni quyidagi qiymatlardan biri yordamida qayta aniqlash mumkin:



7.43-rasm. *Stack*-ning ilovada akslanishi

- *AlignmentDirectional.topStart*: element boshida yuqori tekislash (agar matn yoʻnalishi chapdan oʻngga boʻlsa, bosh chap chegara, matn yoʻnalishi oʻngdan chapga boʻlsa, boshi oʻng chegaradir);
 - *AlignmentDirectional.topEnd*: element oxirida yuqori tekislash (agar matn yoʻnalishi chapdan oʻngga boʻlsa, boshlangʻich oʻng chegara, agar matn yoʻnalishi oʻngdan chapga boʻlsa, boshlangʻich chap chegaradir);
 - *AlignmentDirectional.topCenter*: yuqori markazga tekislash;
 - *AlignmentDirectional.bottomStart*: element boshida pastki tekislash;
 - *AlignmentDirectional.bottomEnd*: element oxiridagi pastki tekislash;
 - *AlignmentDirectional.bottomCenter*: pastki markazni tekislash;
 - *AlignmentDirectional.center*: gorizontaal va vertikal markaz boʻyicha tekislash;
 - *Alignment.centerStart*: konteyner boshida markazga tekislash;
 - *Alignment.centerEnd*: konteyner oxirida markazga tekislash.
- Quyidagi dasturda vidjetlar markazga joylashtirilishiga misol keltirilgan:

```
import 'package:flutter/material.dart';
void main() {
```

```

runApp(Container(
  color: Colors.white,
  padding: EdgeInsets.only(top:40, bottom: 10, left: 20, right: 20),
  child: Stack(
    alignment: AlignmentDirectional.center,
    textDirection: TextDirection.ltr,
    children: <Widget>[
      Container(
        width: 240,
        height: 240,
        color: Colors.blueGrey,
      ),
      Container(
        width: 220,
        height: 220,
        color: Colors.black12,
      ),
      Text("Flutter atdt.uz da",
        textDirection: TextDirection.ltr,
        softWrap: true,
        style: TextStyle(fontSize: 20,
          ),
        ],
      )
    )
  )
)

```

Nazorat savollari:

1. Flutter-da qanday vidjetlar mavjud?
2. Text vidjetining qanday parametrlari mavjud?
3. Flutter-da qanday konteyner turlari mavjud?
4. Flutter-da ishlash uchun qanday dasturiy vositalar kerak?
5. Padding vidjeti nimaga ishlatiladi va uning qayday xususiyatlari mavjud?

GLOSSARIY

Abstract Window Toolkit (AWT) – u platformaga xos usullar yordamida amalga oshirilgan grafik komponentlarning standart to'plami. Ushbu komponentlar barcha platformalar uchun umumiy bo'lgan funksiyalar to'plamini qo'llab-quvvatlaydi.

Abstract – bu sinfning ta'rif bo'lib, uni yaratib bo'lmaydi, lekin ayni paytda boshqa sinflar tomonidan meros qilib olinadi. Abstrakt sinf o'z kichik sinflarida amalga oshirilishi kerak bo'lgan bajarilmagan (mavhum) usullarni o'z ichiga olishi mumkin.

Abstrakt sinf – bu bir yoki bir nechta mavhum usullarni o'z ichiga olgan sinf, shuning uchun sinfning hech qanday nusxasini yaratib bo'lmaydi. Mavhum sinflar boshqa sinflar mavhum usullarni qo'llash orqali ularni kengaytira olishlari va yaratishlari uchun aniqlanadi.

Abstrakt usul – bu amalga oshirilmaydigan usul.

API (Application Programming Interface) – bu foydalanuvchilar uchun mo'ljallangan va obyektlar va sinflarning xususiyatlari va holatiga kirish usullarini tavsiflovchi spetsifikatsiya.

Applet – bu veb-brauzerda yoki boshqa appletlarni ko'rish dasturida ishlaydigan komponent.

Buytkod – Java kompilyatori tomonidan yaratilgan va Java tarjimoni tomonidan bajariladigan mashinadan mustaqil koddir.

Mobil operatsion tizim – mobil qurilmani boshqarish uchun mo'ljallangan tizimdir.

Linux Kernel – apparat qatlami va dasturiy ta'minot stegi o'rtasidagi mavhum qatlamdir.

Android Runtime – ilovalar uchun eng muhim asosiy funktsionallikni ta'minlaydi, Dalvik virtual mashinasini va Android ilovalarini ishga tushirish uchun zarur bo'lgan asosiy Java kutubxonalarini o'z ichiga oladi.

Application Framework – dasturchilarga kutubxona darajasidagi tizim komponentlari tomonidan taqdim etilgan API-larga kirishni ta'minlaydigan dastur ramkasi qatlami.

Applications – bu dastur qatlami, oldindan o'rnatilgan asosiy ilovalar to'plami.

Surface Manager – kompozit oyna boshqaruvchisidir. Kiruvchi chizma buyruqlari ramkadan tashqari buferda to'planadi, ular ma'lum kompozitsiyani tashkil qiladi va keyin ekranda ko'rsatiladi. Bu tizimga qiziqarli uzluksiz effektlar, oyna shaffofligi va silliq o'tishlarni yaratishga imkon beradi.

Media Framework – bu PacketVideo OpenCORE asosida amalga oshirilgan kutubxonalar. Audio va video kontentni yozib olish va ijro etish, shuningdek, statik tasvirlarni ko'rsatish uchun ishlatiladi. Qo'llab-quvvatlanadigan formatlar: MPEG4, H.264, MP3, AAC, AMR, JPG va PNG.

SQLite – bu Android tomonidan asosiy ma'lumotlar bazasi mexanizmi sifatida ishlatiladigan engil va yuqori samarali relyatsion ma'lumotlar bazasi mexanizmi.

FreeType – bitmaplar bilan ishlash, shriftlarni rasterlash va ular ustida amallarni bajarish uchun kutubxona.

LibWebCore – bu Google Chrome va Apple Safari brauzerlarida ishlatiladigan WebKit brauzer mexanizmi kutubxonalari.

SGL (Skia Graphics Engine) ochiq manbali 2D grafik dvigatelidir. Grafik kutubxona Google kompaniyasining mahsulotidir va boshqa dasturlar tomonidan qo'llaniladi.

SSL – xuddi shu nomdagi kriptografik protokolni qo'llab-quvvatlash uchun kutubxona.

Libc – bu Linux-ga asoslangan qurilmalarda ishlash uchun sozlangan C standart kutubxonasi.

Vidjetlar – ish stolida grafik obyekt sifatida ko'rsatiladigan ilovalardir.

Activity – bu grafik foydalanuvchi interfeysini ko'rsatish uchun mas'ul bo'lgan ilovaning ko'rinadigan qismi.

Servis – bu fonda ishlaydigan, ko'p vaqt talab qiladigan operatsiyalarni bajaradigan yoki masofaviy jarayonlar uchun ishlaydigan komponent.

Kontent provayder – mobil ilova ma'lumotlarining taqsimlangan to'plamini boshqarishdir.

Broadcast Receivers – translyatsiya bildirishnomalariga javob beruvchi komponent.

Intents – bu ilovalarning bir-biri bilan va operatsion tizim bilan bog'lanishiga imkon beruvchi tizim xabarlarini.

Hodisa – bu sahifa yoki ekran ko'rinishidan mustaqil ravishda kuzatilishi mumkin bo'lgan kontent bilan foydalanuvchi o'zaro aloqasi.

View – foydalanuvchi ko'rgan interfeys turi.

ViewModel – bu ma'lumotlarni qayta ishlash va foydalanuvchi interfeysida taqdim etish, View va Modelni bog'laydi.

Data Handlers – bu dastur ishlaydigan modellarda tarmoq orqali uzatiladigan yoki diskda saqlanadigan shakldan ma'lumotlarni tayyorlash.

Mobil veb-sayt – bu shaxsiy kompyuterda ham, mobil qurilmalarda ham ko'rish mumkin bo'lgan sezgir dizaynga ega bo'lgan maxsus veb-sayt.

Mobil ilova – bu mobil operatsion tizimlar uchun mo'ljallangan maxsus dastur. Ushbu operatsion tizimlar: Windows Phone OS, Android OS va iOS OS.

XML (Extensible Markup Language) matnlar yoki matnli hujjatlar uchun kengaytiriladigan belgilash tilidir.

Interpolatsiya – bu turli xil harflar va qiymatlarni boshqa ma'lumotlar tiplarini ifodalashi mumkin bo'lgan qatorga birlashtirishdir.

ADABIYOTLAR

1. Wei-Meng Lee. "Beginning Android TM 4 Application Development". Radha Offset, Delhi. 2013.
2. Wei-Meng Lee. "Android TM Application Development Cookbook". Sharda Offset Press, Delhi. 2013.
3. Zigurd Mednieks, Laird Dornin, G. Blake Meike, Masumi Nakamura. "Programming Android Second Edition". 2012.
4. Reto Meier. Professional Android 4 Application Development. John Wiley & Sons, 2013. – 816 p.
5. Wei-Meng Lee. Android™ Application development cookbook. John Wiley & Sons, 2013. – 410 p.
6. Голощапов А. Л. Google Android: программирование для мобильных устройств. СПб.: БХВ-Петербург, 2011. - 448 с.
7. Усов В. Swift. Основы разработки приложений под iOS. — СПб.: Питер, 2016. - 304 с.
8. П. Дейтел, Х. Дейтел, Э. Дейтел, М. Моргано. Android для программистов. Создаем приложения. Prentice Hall, -2013. – 560 с.
9. Колисниченко Д. Android для пользователя. Полезные программы и советы. СПб.: БХВ-Петербург, 2013. - 256 с.
10. Дошанова М.Ю. «Программные средства для мобильных устройств». Учебное пособие. – ТУИТ "Алокачи", 2020

RESURSLAR:

1. <http://www.tutorialspoint.com/Android/index.htm>
2. <http://www.metanit.com>
3. <http://www.amazon.com>
4. <http://dl.e-book-free.com>
5. <https://flutter.su/>

MUNDARIJA

KIRISH	3
1. MOBIL. ILOVALARNI YARATISH ASOSLARI VA HAYOT SIKLI	5
1.1. Mobil ilovalarni yaratish uchun dasturlash muhitlari	5
1.1.1. Mobil ilovalarni ishlab chiqish dasturlari	5
1.1.2. Android IDE.....	7
1.1.3. Intel XDK.....	8
1.1.4. Marmelad SDK.....	8
1.2. Mobil ilovalarni yaratish uchun dasturlash tillari	9
1.2.1. Android operatsion tizimi uchun Java tili.....	9
1.2.2. Android operatsion tizimi uchun Dart tili.....	11
1.3. Mobil ilovalarning hayotiy sikli.....	12
1.3.1. Ilovaning hayotiy sikli	12
1.3.2. Hayotiy siklni boshqarish.....	16
1.3.3. Activity holati	21
1.3.4. Jarayonlar va oqimlar	23
Nazorat savollari	28
2. MOBIL QURILMALAR BILAN ISHLASH.....	29
2.1. Mobil operatsion tizim tushunchasi.....	29
2.1.1. Mobil qurilmalarda operatsion tizim tushunchasi.....	29
2.1.2. Symbian OT	29
2.1.3. Windows Mobile.....	30
2.1.4. Android operatsion tizimi	30
2.1.5. iPhone OT	31
2.1.6. Palm OT	31
2.1.7. BlackBerry OT.....	32
2.2. Mobil operatsion tizimlar platformasi va arxitekturasi.....	32
2.2.1. Android platformasi va uning asoslari	32

2.2.2. Android ilovalarining asosiy turlari	36
2.2.3. Hova arxitekturasi, asosiy komponentalar	38
2.2.4. Asboblarni o'rnatish va sozlash	41
2.3. Mobil operatsion tizim platformasiga mos dasturlash tillari	53
2.3.1. Android OT tillari	53
2.3.2. iOS OT tillari	55
2.3.3. Platformalararo tillari	56
Nazorat savollari:	59
3. ANDROID UCHUN JAVA DASTURLASH TILIDA ILOVALAR YARATISH	60
3.1. Java dasturlash tilining asosiy konstruksiyalari	60
3.1.1. Ma'lumotlar turlari	60
3.1.2. Maxsus sinflar va funksiyalar	61
3.2. Sinflar va obyektlar	63
3.2.1. Sinf tavsifi	63
3.2.2. Konstruktorlar	65
3.2.3. <i>This</i> kalit so'z	67
3.2.4. Initsializatorlar	68
3.3. Java dasturlash tilida mobil ilovalarni ishlab chiqish	68
Nazorat savollari:	83
4. MOBIL ILOVALARDA MA'LUMOT BAZASI BILAN ISHLASH, GEOLOKATSIYA BILAN ISHLASH	84
4.1. Ma'lumotlar bazasi bilan ishlash	84
4.1.1. Androidda ma'lumotlar bazasiga kirish	84
4.1.2. Ma'lumotlar bazasini yaratish	84
4.1.3. So'rovlarni yaratish	89
4.2. Kontent provayderlar va ulardan foydalanish	92
4.3. Mobil ilovalarda tarmoqli dasturlash	101
4.4. Server bilan ishlash	105

4.5. Foydalanuvchining joylashuvini aniqlash	114
Nazorat savollari:	123
5. MOBIL DATCHIKLAR BILAN ISHLASH	125
5.1. Androidning sensor imkoniyatlari	125
5.2. Sensorlar turi va ular bilan ishlash	132
5.3. Mobil ilovalarda mobil datchiklardan foydalanish	140
Nazorat savollari:	149
6. IOS UCHUN SWIFT DASTURLASH TILIDA ILOVALARNI YARATISH	150
6.1. Swift tiliga kirish	150
6.2. Xcode da Swift bilan ishlash	151
6.3. Swift dasturlash tilining asosiy konstruksiyalari	155
6.4. Swift dasturlash tilida mobil ilovalarni ishlab chiqish	237
Nazorat savollari:	254
7. FLUTTERDA ANDROID OT UCHUN MOBIL ILOVA ISHLAB CHIQISH	255
7.1. Flutter-ga kirish	255
7.2. Flutter-da vidjetlar bilan ishlash	281
Nazorat savollari:	306
GLOSSARIY	307
ADABIYOTLAR:	310

Doshanova M. Yu.

MOBIL ILOVALARINI ISHLAB CHIQISH

O'QUV QO'LLANMA

Toshkent - "METODIST NASHRIYOTI" - 2024

Muharrir: Bakirov Nurmuhammad

Texnik muharrir: Tashatov Farrux
Musahhih: Hazratqulova Ruxshona
Dizayner: Ochilova Zarnigor

Bosishga 08.07.2024.da ruxsat etildi.
Bichimi 60x90. "Times New Roman" garniturasida.
Ofset bosma usulida bosildi.
Shartli bosma tabog'i 20. Nashr bosma tabog'i 19,75.
Adadi 300 nusxa.

"METODIST NASHRIYOTI" MCHJ matbaa bo'limida chop etildi.
Manzil: Toshkent shahri, Shota Rustaveli 2-vagon tor ko'chasi, 1-uy.



+99893 552-11-21

Nashriyot rozilgisiz chop etish ta'qiqlanadi.

ISBN 978-9910-03-150-2



9 789910 031502

