ҚАЗАҚСТАН РЕСПУБЛИКАСЫ АУЫЛ ШАРУАШЫЛЫҒЫ МИНИСТРЛІГІ

Жәңгір хан атындағы Батыс Қазақстан аграрлық-техникалық университеті

А. С. Бекенова,
С. С. Бекенова

ЗАМАНАУИ БАҒДАРЛАМАЛАУ ТІЛДЕРІ

Оқу құралы

ОРАЛ
2021

ӘОЖ 004.438
КБЖ 32.973.26 - 018.1
Б 42

Жәңгір хан атындағы Батыс Қазақстан аграрлық-техникалық университетінің Оқу-әдістемелік кеңесімен басылымға ұсынылған
(хаттама №10, 26.05.2021 ж.)

Сын-пікір берушілер: Кушеккалиев А.Н. – физ.-мат.ғ.к., доцент, М.Өтемісов атындағы БҚУ
Бапиев И.М. – PhD докторы, Жәңгір хан атындағы БҚАТУ

Бекенова А.С.
Б42 Заманауи бағдарламалау тілдері: оқу құралы / **А.С. Бекенова, С.С. Бекенова**– Орал: Жәңгір хан ат. Бат. Қаз. агр.-техн. ун-ті, 2021. –205 б.

ISBN 978-601-319-293-2

Оқу құралында заманауи бағдарламалау тілдеріне шолу жасалып, нақты Python және Java бағдарламалау тілдерінің пайда болу тарихы мен дамуы, негізгі ерекшеліктері, мүмкіндіктері жан-жақты қарастырылған.

Оқу құралы білім алушылардың қазіргі кездегі бағдарламалау бойынша материалдарды жан-жақты, әрі саналы түрде меңгеруін қамтамасыз ету, қызығушылығын оятып, шығармашылығын шыңдау, бағдарламалау құралдарын өздерінің оқу әрекетінде қолдану дағдыларын қалыптастыруға бағытталған.

Оқу құралы жоғары оқу орындарының «Ақпараттық жүйелер және технологиялар» және «Кәсіптік білім беру» білім беру бағдарламасы бойынша оқитын студенттерге, магистранттарға, оқытушыларға, сонымен қатар бағдарламалауға қызығушылық танытатын оқырмандарға арналған.

ӘОЖ 004.438
КБЖ 32.973.26 - 018.1

ISBN 978-601-319-293-2

МАЗМҰНЫ

КІРІСПЕ

Ақпараттық технологиялар әлемінде көптеген бағдарламалау тілдері бар. Жыл сайын олардың саны көбейіп келеді. Мысалы, Scala, Kotlin, Go және Closure тілдері салыстырмалы түрде жақында пайда болды. Бірақ тарих олардың бірнешеуі ғана тірі қалатынын айтады.

Мұнда талқыланған тілдер бағдарламалық жасақтама жасау әлеміне үлкен үлес қосты. Сондықтан олар соңғы елу жылдың ең ықпалды тілдерінің ондығына кірді.

Тілдер - бағдарламалау индустриясының маңызды бөлігі. Олар жиі талқыланады, сынға алынады және уақыт өте келе жақсарады. Олар әркімнің аузында, бірақ бәрі де өз тарихын біледі, ең болмағанда олардың жасаушыларының есімдері қандай. Бұл танымал тілдерде аз кездеседі. Мысалы, Джеймс Гослинг Java-ның әкесі екенін бәрі біледі. Бірақ кез-келген бағдарламашы Perl, Pascal, Lisp немесе Erlang кім жасағанын біле бермейді.

Компьютерлер біздің өмірімізде соншалықты берік орын алды, біз оны онсыз елестете алмаймыз. Жарты ғасыр бұрын ғана баяу және икемсіз алғашқы компьютерлер пайда болды дегенге сену қиын. Бізде қазір өте үлкен әртүрлілік бар - ультра жұқа ноутбуктардан қуатты ойын компьютерлеріне дейін.

Енді прогресс техниканың қолданушының сұранысын интуитивті түрде түсіне алатын деңгейге жетті. Алайда, барлық бағдарламалар, браузерлер мен сайттар бағдарламалау тілдеріне негізделгенін есте ұстаған жөн.

Бүгінгі күні осындай 8 мыңға жуық тіл бар, олардың көпшілігі тек авторларға ғана түсінікті. Алайда халықаралық болып қалыптасқан және күн сайын миллиондаған адамдар қолданатын бағдарламалау тілдері бар.

Бағдарламалау тілдері не үшін қажет?

Бағдарламалау тілі болмаса, ешбір компьютер жұмыс істемейді. Оның көмегімен компьютерге қолданушының командаларын орындауға көмектесетін белгілі бір математикалық алгоритмдер құрылады.

- Бағдарламалау тілі біз тілді елестеткеннен мүлдем өзгеше көрінеді - бұл компьютер түсінетін кодқа айналдырылған әр түрлі белгілер жиынтығы.
- Бағдарламалау тілдерінің төмен деңгейден жоғары деңгейге жіктелуі тілдің адамдарға қаншалықты жақын екендігіне байланысты. Адамға бағдарламалау тілі неғұрлым аз түсінікті болса, соғұрлым оның деңгейі төмен болады.
- Көптеген тілдер аудармашы бағдарламалар көмегімен машинада оқылатын кодқа айналады. Олардың көмегімен қолданушының қандай

әрекеттерді сұрайтынын және бағдарламаның сыртқы түрін анықтайтын лексикалық, семантикалық және синтаксистік ережелер жазылады.

- Бағдарламалау тілі кез-келген компьютердің жұмысының ажырамас бөлігі болып табылады. Қазіргі әлемде олармен программисттер жұмыс істейді, олар код жазады және күрделі код құрылымдарын қолданып бағдарламалар жасайды.

Қарапайым адам үшін сайттың бірінші парағын да жазу әр түрлі таңбалардың таңғажайып тіркесімі сияқты болып көрінеді. Бағдарламалаушы үшін бұл код компьютерге қажетті команданы орнатуға және оны орындауға мүмкіндік береді. Бағдарламалау тілінде қарапайым пайдаланушы компьютермен байланысады.

Бағдарламалау тілдерінің тарихы

Бағдарламалау тілінің тарихындағы алғашқы «қарлығаштар» музыкалық жәшік немесе джаккард тоқу станогы сияқты қарапайым заттар болып саналады. Олар 18-19 ғасырларда пайда болды.

Қазіргі бағдарламалаудың басталуы 1930-1940 жылдары, лямбда калькуляциясы мен Тьюринг машинасы жазылған кезде басталды. Ол уақытты бағдарламалау тілдерінің даму тарихының бастауы деп атауға болады, өйткені компьютерлер танымал бола бастады. 1940 жылдары алғашқы электрлік есептеуіш машиналар пайда болды, сол уақытта неміс инженері К.Кузе алғашқы жоғары деңгейдегі тіл Планкулкульді жасады.

1950 жылдары бағдарламалаудың бірінші буыны болып саналатын машиналық кодтар жасалды. «Қысқа кодты» алғашқы жұмыс істейтін бағдарламалау тілі деп санау үшін оларды әр компьютер үшін бөлек жазу керек болды. Ол бірінші болып математикалық кодтар жиынтығын емес, содан кейін кодқа өңделген өрнектерді ұсынды.

Содан кейін екінші ұрпақтың тілдері пайда болды - олар адамдар үшін оңай болды, бірақ ассемблерді қолданумен шектелді. Бұл команданы машиналық тілге аударған бағдарламаның атауы, аудармашының бір түрі болды.

Ал 1950 жылдардың ортасына қарай үшінші буын тілдері пайда бола бастады. Олар кез-келген бағдарламаға сәйкес келетіндігімен ерекшеленеді, оларды әрқайсысы үшін жаңадан жазу қажет емес. Олар қазірдің өзінде жоғары деңгейдегі толыққанды тілдер болып саналады.

1960 жылдары бағдарламалау тілдері стандартталып, жетілдіріле бастады. Содан кейін төртінші және бесінші буын тілдері пайда болғанымен, олар үшінші буын тілдерінің жетілдірілген нұсқалары болып табылады.

1990 жылдары Интернет дами бастағаннан кейін веб-парақ жазуға арналған тілдер көбірек болды. Қазіргі кезде бағдарламалау тілдерін дамыту көбіне қолданушы қауіпсіздігіне бағытталған.

Заманауи бағдарламалау тілдеріне шолу

Бағдарламалаудың мыңдаған тілдері бар. Біз ең танымал және қазіргі әлемде жиі қолданылатындарды қарастырайық.

Basic

Қазіргі заманғы бағдарламалау тілдері Basic немесе BASIC жоғары деңгейлі бағдарламалау тілдерінің тобына жатады. Оны 1964 жылы Дартмут колледжінің профессорлары студенттерге өздерінің компьютерлік бағдарламаларын құруға көмектесу үшін жасады. Қазір Томас Курц пен Джон Кеменінің туындылары Windows амалдық жүйесіне арналған бағдарламалар жазылатын негізгі тілге айналды.

C

Сонау 1972 жылы Деннис Ричи бүгінгі күнге дейін танымал болып келе жатқан тіл ойлап тапты. Бағдарламашылар оны жоғары және төменгі деңгейлі бағдарламалау элементтерінің үйлесімділігі үшін жақсы көреді. Айтпақшы, сайттарды жазу үшін қолданылатын бағдарламалау тілдерінің негізі тек Си тілі болып табылады.

C ++

Бұл тіл C тілінен алынған, бірақ ол жалпы бағдарламалауға көбірек көңіл бөледі. Қазір бұл ең танымал тілдердің бірі, өйткені онда көптеген бағдарламалар жазылған. Ол амалдық жүйелерді, драйверлерді, серверлерді, ойындарды, қосымшаларды және басқаларын жасауға жарайды. C ++ - тиімді, көп функционалды және көптеген бағдарламашыларға қол жетімді әмбебап тіл.

Python

Бұл тілдің дамуын 1980 жылдары голландиялық Гидо ван Россум бастаған, бірақ оның алғашқы нұсқасы 2008 жылға дейін шыққан жоқ. Мұнда үнемі жетілдіру және белсенді пайдаланушылар қауымдастығы бар. Python - әр түрлі функциялары бар жоғары деңгейлі тіл. Ол әсіресе веб-әзірлеуде, деректерді талдау мен процестерді автоматтандыруда шебер.

PHP

Бұл тіл веб-сайттарды дамытуда қолданылатын жетекші тіл болып табылады және барлық дерлік хостинг-провайдерлер қолдайды. Ол негізінен веб-сайттар мен веб-қосымшаларды әзірлеу үшін қолданылады. PHP-ді алғаш рет 1995 жылы дат бағдарламашысы Расмус Лердорф ұсынды.

Java

Java бағдарламалау тілі байт-кодқа аударылған веб-қосымшалармен жұмыс істейді. Ол кез-келген компьютер архитектурасында жұмыс істей алады, өйткені кодты Java машинасы түрлендіреді. Ол 1996 жылы пайда болды және соңғы жылдары ол ең танымал бағдарламалау тіліне айналды. Алайда көпшілігі оның бәсекелестеріне қарағанда баяу жұмыс істейтініне наразы.

JavaScript

JavaScript Java-ға ұқсас болғанымен, ол бөлек тіл. Көбінесе ендірілген тіл ретінде қолданылады: қосымшалар веб-парақтарға оны қолдана алады. Оны пайдалану оңайырақ, тіпті бағдарламалаумен таныс емес адамдарға да беріледі. Оны пайдалану үлесі жыл сайын артып келеді.

Go (Golang)

2007 жылы Google өмірдегі мәселелерді шеше алатын өзінің бағдарламалау тілін дамыта бастады. Тілді Роб Пайк пен Кен Томпсон жасады, олар Go-ді 2009 жылы таныстырды. Google үшін бұл танымал C және C ++ тілдерін ауыстырады. Бұл серпіліс болған жоқ, бірақ ол елеулі жобалар жасау үшін қолданылады.

SQL

Бұл бағдарламалау тілі - бұл мәліметтер қорын ыңғайлы басқаруға болатын жүйе. Алғашқы әзірлемелер 1970 жылдары басталды, бірақ SQL-дің алғашқы нұсқасы 1986 жылы енгізілді. Қазіргі әлемде SQL мәліметтер базасын жетілдіруге, басқаруға және құруға ыңғайлы тіл ретінде танымал.

Swift

Сондай-ақ, Apple ондағы барлық байланысты гаджеттерге қосымшалар жасау үшін өзінің бағдарламалау тілін ойлап тапты. Apple өз тілін 2014 жылы қолданушыға ыңғайлы, сенімді, ақысыз және кез-келген бағдарламашыға қол жетімді тіл ретінде ұсынды. Ол жеке Apple өнімдері үшін жасалған.

Қазіргі әлемде бағдарламалау тілдерінсіз мүмкін емес. Бағдарламалаушы мамандығын өз бетінше таңдайтындар саны артып келеді, бірақ бұл бір қарағанда қиын болса да, ол тез дамып, жыл сайын танымал болып келеді.

Бағдарламалау тілдері - бұл компьютер әлемін құрудың күрделі, бірақ қызықты процесі!

I ТАРАУ. PYTHON БАҒДАРЛАМАЛАУ ТІЛІ

1.1 Python бағдарламалау тіліне кіріспе

Python - ең танымал бағдарламалау тілдерінің бірі. Ол дамуда кеңінен қолданылады.

Python үшін танымал қолданбалардың бірі веб-сайттар мен веб-қосымшаларды әзірлеу, атап айтқанда Django фреймворктарын қолдану болып табылады. Тағы бір кең таралған бағыт - бұл машиналық оқыту, сонымен қатар жасанды интеллектке байланысты әр түрлі бағдарламалар. Сонымен қатар, Python көмегімен басқа қосымшалар жасауға болады, мысалы, әр түрлі жұмыс үстелінің консолі және графикалық бағдарламалары, тіпті мобильді қосымшалар. Көбінесе бұл тіл басқа бағдарламалар шеңберінде тапсырыс сценарийлерін жазу үшін қолданылады.

Python - бұл аудармашы тіл, ол аудармашы орнатылған жерде жұмыс істейді. Шын мәнінде, бұл әмбебап кросс-платформалық тіл, сондықтан Python қосымшалары Windows, Linux, MacOS сияқты ең танымал платформаларда жұмыс істейтін болады.

Python - бұл әр түрлі типтегі қосымшаларды құруға арналған танымал жоғары деңгейлі бағдарламалау тілі. Бұл веб-қосымшалар, ойындар, жұмыс үстелі бағдарламалары және мәліметтер базасымен жұмыс. Python машиналық оқыту және жасанды интеллект зерттеу саласында айтарлықтай кең таралған.

Python туралы алғаш рет 1991 жылы голландтық әзірлеуші Гидо Ван Россум жариялады. Содан бері бұл тіл дамудың ұзақ жолынан өтті. 2.0 нұсқасы 2000 жылы, ал 3.0 нұсқасы 2008 жылы шыққан. Нұсқалар арасындағы осындай үлкен алшақтықтарға қарамастан, версиялар үнемі шығарылып отырады. Сонымен, осы жазу кезінде қолданыстағы нұсқасы 3,9 құрайды. Тілдің барлық шығарылымдары, нұсқалары мен өзгертулері туралы, сондай-ақ аудармашылардың өздері және жұмысқа қажетті утилиталар туралы және басқа пайдалы ақпарат туралы толығырақ ақпаратты https://www.python.org/ ресми сайтынан табуға болады.

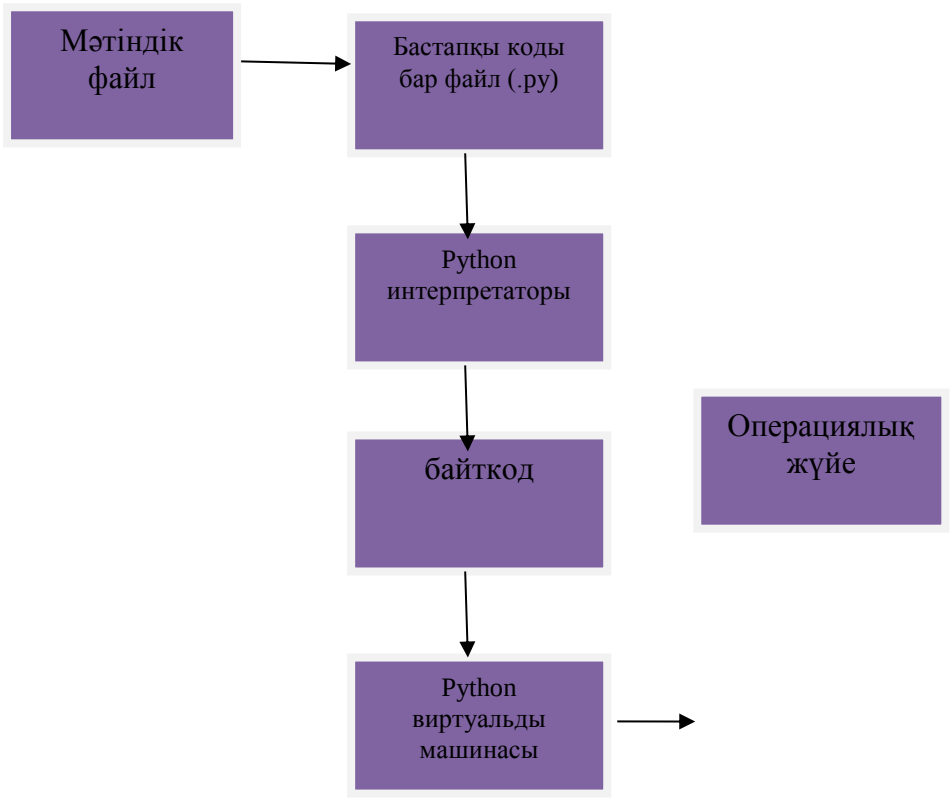
Python бағдарламалау тілінің негізгі ерекшеліктері:

- Сценарий тілі. Бағдарлама коды сценарий түрінде анықталады.
- Бағдарламалау парадигмаларының алуан түрлілігін, соның ішінде объектіге бағытталған және функционалды парадигмаларды қолдау.
- Бағдарламалардың интерпретациясы. Сценарийлермен жұмыс істеу үшін сценарийді іске қосатын және орындайтын интерпретатор керек.

Python бағдарламасының орындалуы 1.1 - суреттен көрінеді. Біріншіден, біз мәтіндік редакторға берілген бағдарламалау тілінде өрнектер жиынтығымен сценарий жазамыз. Біз бұл сценарийді орындау

үшін аудармашыға жібереміз. Аудармашы кодты аралық байт-кодқа айналдырады, содан кейін виртуалды машина алынған байт-кодты операциялық жүйе орындайтын нұсқаулар жиынтығына айналдырады.

Бұл жерде ресми кодты интерпретатордың байт-кодқа аударуы және виртуалды машинаның машиналық нұсқаулық жиынтығына аударуы формальды түрде болғанымен, яғни екі түрлі процесті бейнелейтініне қарамастан, іс жүзінде олар аудармашының өзінде біріктірілген.



Сурет 1.1 - Python бағдарламасының орындалуы

- Портативтілік және платформаға тәуелділік. Бізде қандай операциялық жүйе бар екендігі маңызды емес - Windows, Mac OS, Linux, аудармашының қатысуымен барлық осы ОЖ-де жұмыс істейтін сценарий жазу жеткілікті
- Жадыны автоматты басқару
- Динамикалық бірізділік (типизация)

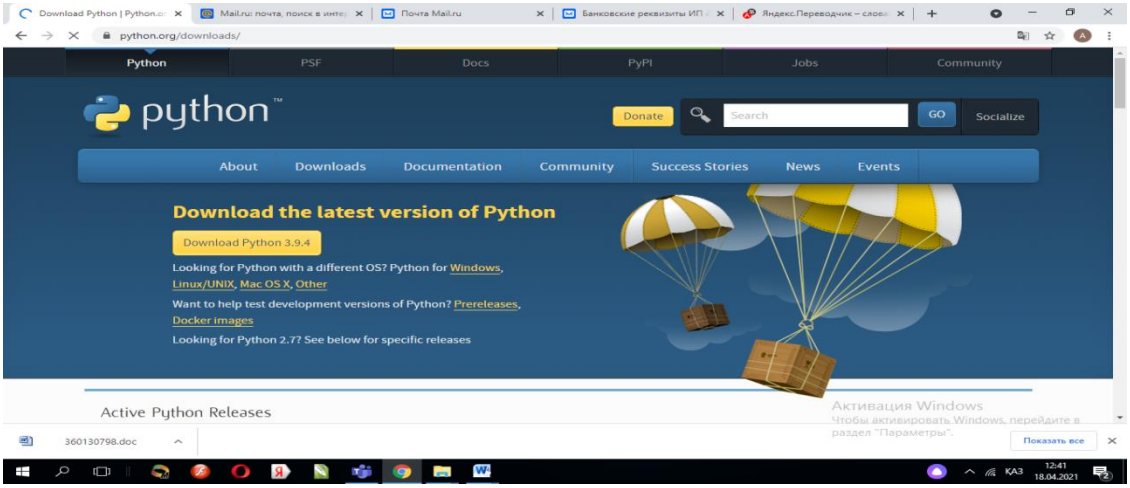
Python-өте қарапайым бағдарламалау тілі, ол қысқа және сонымен бірге қарапайым және түсінікті синтаксиске ие. Тиісінше, оны үйрену оңай, және іс жүзінде бұл оқытудың ең танымал бағдарламалау тілдерінің бірі болып табылады. Атап айтқанда, 2014 жылы ол АҚШ-та оқуға арналған ең танымал бағдарламалау тілі ретінде танылды.

Python тек оқу саласында ғана емес, сонымен қатар нақты бағдарламаларды, соның ішінде коммерциялық сипаттағы

бағдарламаларды жазуда да танымал. Сондықтан бұл тіл үшін біз қолдана алатын көптеген кітапханалар жазылған. Сонымен қатар, бұл бағдарламалау тілі өте үлкен қауымдастыққа ие, интернетте сіз осы тілден көптеген пайдалы материалдарды, мысалдарды таба аласыз және мамандардан білікті көмек ала аласыз.

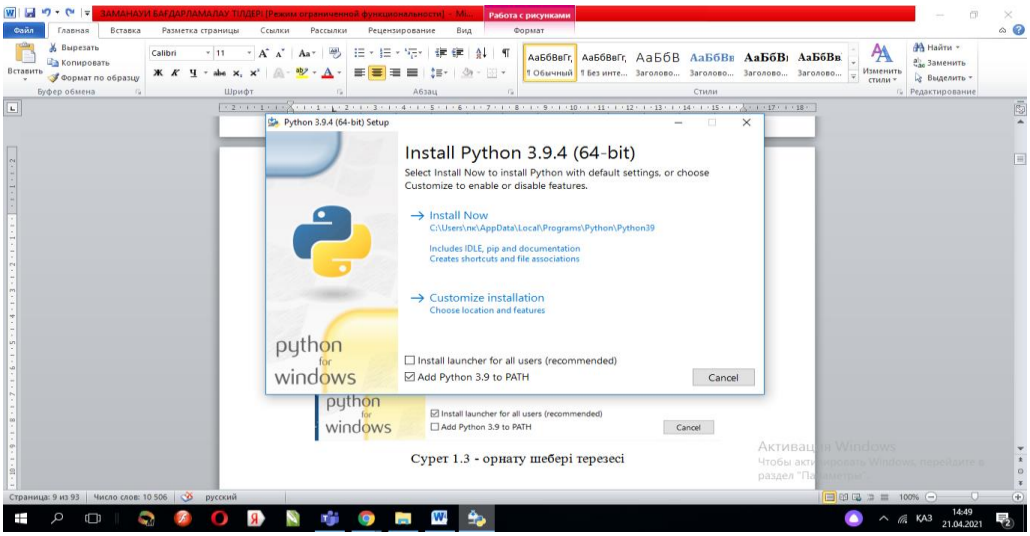
Python – ды орнату

Python бағдарламаларын жасау үшін бізге аудармашы қажет. Оны орнату үшін https://www.python.org/downloads / парағына өтіп және тілдің соңғы нұсқасын жүктеу сілтемесін (1.2 – сурет) табамыз (қазіргі уақытта бұл 3.9.0):



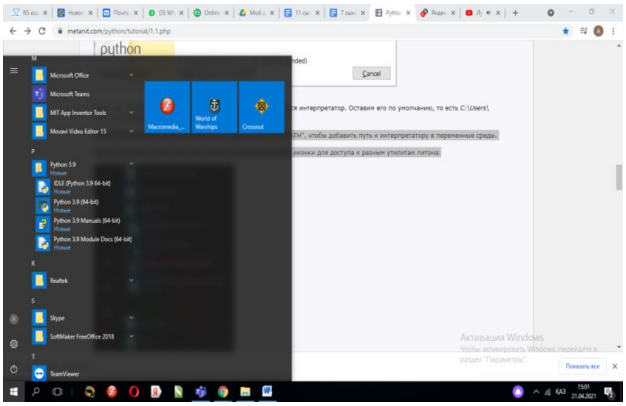
Сурет 1.2 – Бағдарламалау тілін жүктеу сілтемесі

Түймені басқанда, тиісті ағымдағы Python орнатушысы жүктеледі.Windows ОЖ-де орнатушы іске қосылған кезде орнату шебері терезесі (Сурет 1.3) іске қосылады:



Сурет 1.3 - орнату шебері терезесі

Сонымен қатар, төменгі жағында айнымалылар ортасына интерпретатордың жолын көрсететін "Add Python 3.9 to PATH" жолына белгіше қойыңыз. Windows жүйесінде Бастау мәзіріне орнатқаннан кейін біз питонның әртүрлі утилиталарына қол жеткізу үшін белгішелерді таба аламыз. (Сурет 1.4):



Сурет 1.4 – Python- ға өту

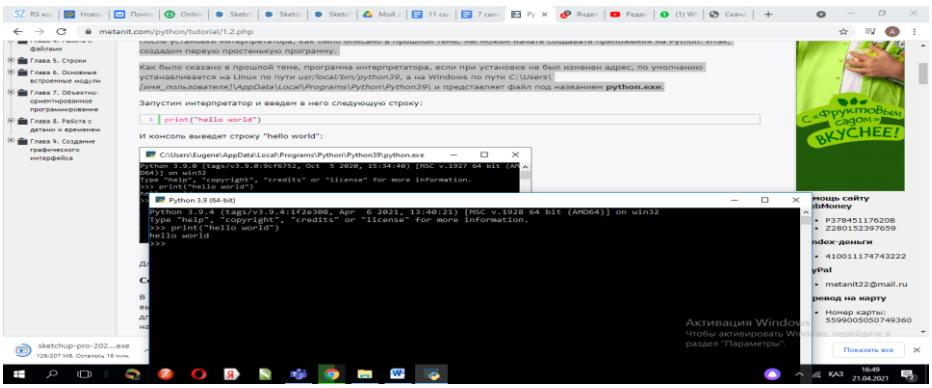
Мұнда Python 3.9 (64-bit) утилитасы сценарийді іске қосуға болатын интерпретаторды ұсынады. Файлдық жүйеде интерпретатор (аудармашының) файлын өзі орнатылған жолмен табуға болады.

Python – дағы алғашқы бағдарлама

Интерпретатор орнатқаннан кейін, біз Python-да қосымшалар жасай бастаймыз. Сонымен, алғашқы қарапайым бағдарламаны жасаңыз.

Интерпретатор бағдарламасы, егер орнату кезінде мекен-жайы өзгертілмесе, әдепкі бойынша Linux-та usr/local/bin/python3.9 жолында, ал Windows-та C:\Users\[Пайдаланушының аты]\AppData\Local\Programs\Python\Python 3.9 жол бойында орнатылады және python.exe деп аталатын файлда сақталады.

Интерпретаторды іске қосып print (“hello world”) деп жазамыз, сонда 1.5- суреттегі нәтижені аламыз:

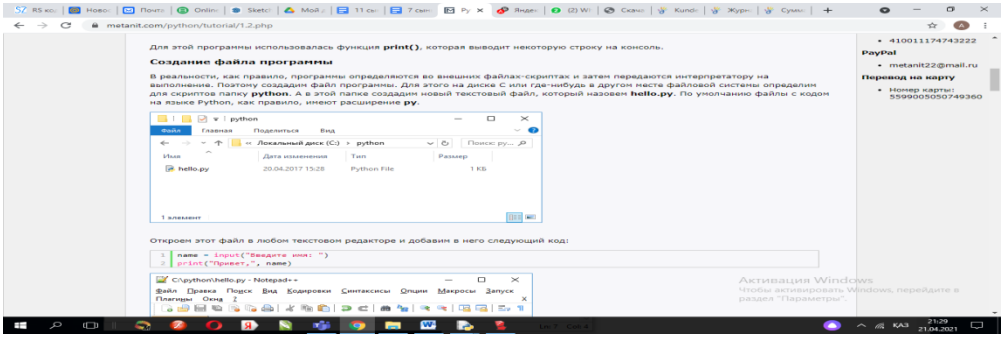


Сурет 1.5- Алғашқы бағдарлама

Бұл бағдарлама үшін консольге қандайда бір жолды шығаратын print () функциясы қолданылды.

Бағдарлама файлын жасау

Шындығында, әдетте, бағдарламалар сыртқы сценарий файлдарында анықталады, содан кейін орындау үшін аудармашыға беріледі. Сондықтан бағдарлама файлын жасаймыз. Ол үшін С дискісінде немесе файлдық жүйенің басқа жерінде сценарийлер үшін python бумасын анықтаймыз. Осы бумада біз жаңа мәтіндік файл жасаймыз, оны hello.py деп атаймыз. Python тіліндегі коды бар файлдарда кеңейтілуі .py болады. (Сурет 1.6)



Сурет 1.6 - бағдарлама файлы

Осы файлды кез келген мәтіндік редакторда ашып төмендегідей код жазамыз:

```
1   name =input("Esiminizdi engiziniz: ")
2   print("Salem,", name)
```

Скрипт екі жолдан тұрады. *Input()* функциясын қолдана отырып, бірінші жол пайдаланушының өз атын енгізуін күтеді. Содан кейін енгізілген атау name айнымалысына түседі.
Екінші жол *print ()* функциясын қолдана отырып, енгізілген атпен бірге сәлемдесуді көрсетеді.

Нәтижесінде бағдарлама есіміңізді енгізуге шақыруды, содан кейін сәлемдесуді көрсетеді.

PyCharm

Әртүрлі интеграцияланған даму орталарын немесе IDE-ді қолдануды білдіретін бағдарламаларды құрудың тағы бір әдісі бар.

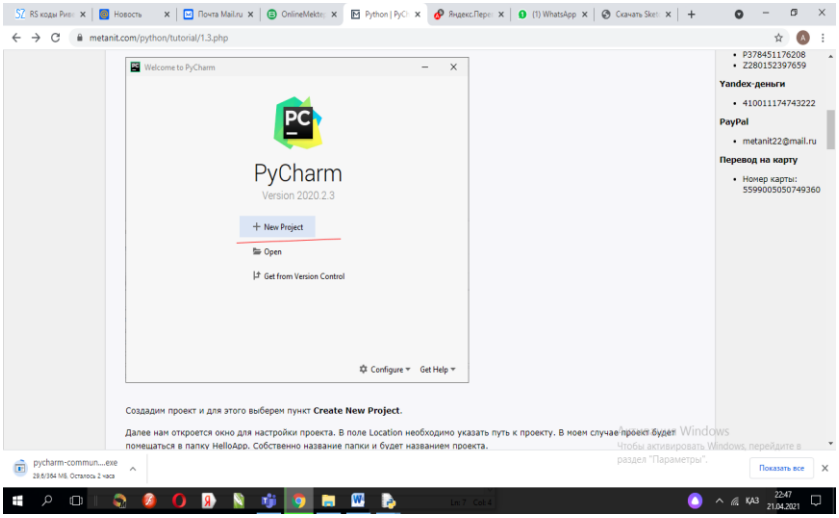
IDE бізге кодты теру үшін мәтіндік редакторды ұсынады, бірақ стандартты мәтіндік редакторлардан айырмашылығы, IDE сонымен қатар синтаксистің толық жарықтандырылуын, АВТО-толықтыруды немесе ақылды кодты ұсынуды, жасалған сценарийді бірден орындау мүмкіндігін және тағы басқаларды қамтамасыз етеді.

Python үшін әр түрлі өңдеу орталарын қолдануға болады, бірақ олардың ішіндегі ең танымалыларының бірі-JetBrains жасаған PyCharm ортасы. Бұл орта қарқынды дамып келеді, үнемі жаңартылып отырады және ең көп таралған Операциялық жүйелер - Windows, MacOS, Linux үшін қол жетімді.

Бірақ оның бір маңызды шегі бар. Атап айтқанда, ол екі негізгі нұсқада қол жетімді: ақылы кәсіби шығарылым және тегін қауымдастық. Көптеген негізгі мүмкіндіктер қауымдастықтың ақысыз шығарылымында қол жетімді. Сонымен қатар, Веб-әзірлеу сияқты бірқатар мүмкіндіктер тек ақылы Professional-да қол жетімді.

Біздің жағдайда біз Қауымдастықтың тегін шығарылымын қолданамыз.

Ол үшін https://www.jetbrains.com/pycharm/download/#section=windows жүктеу бетіне өтіп, PyCharm Community орнату файлын жүктейміз. Жүктегеннен кейін біз оны орнатуды орындаймыз.Орнату аяқталғаннан кейін бағдарламаны іске қосамыз. Бірінші іске қосу кезінде бастапқы терезе ашылады (Сурет 1.8):



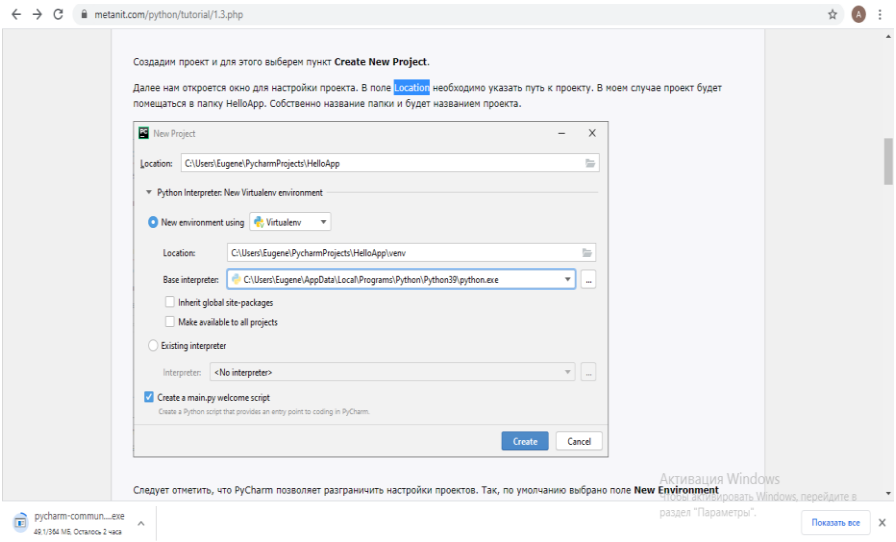
Сурет 1.8 –PyCharm алғашқы терезесі

Жобаны жасаймыз және ол үшін **Create New Project** (Сурет 1.9) тармағын таңдаймыз.

Әрі қарай, жобаны баптау үшін терезе ашылады. **Location** өрісінде жобаға жолды көрсету қажет. Біздің жағдайда, жоба HelloApp бумасына орналастырылады. Іс жүзінде буманың атауы Жобаның атауы болады.

Айта кету керек, PyCharm жоба параметрлерін бөлуге мүмкіндік береді. Сонымен, үнсіздік бойынша, **New Environment Using** өрісі таңдалады, бұл белгілі бір жоба үшін аудармашының нұсқасын орнатуға мүмкіндік береді. Содан кейін барлық орнатылған қосымша пакеттер тек ағымдағы жобаға қатысты болады. Егер біз бірнеше жоба жасасақ, бұл

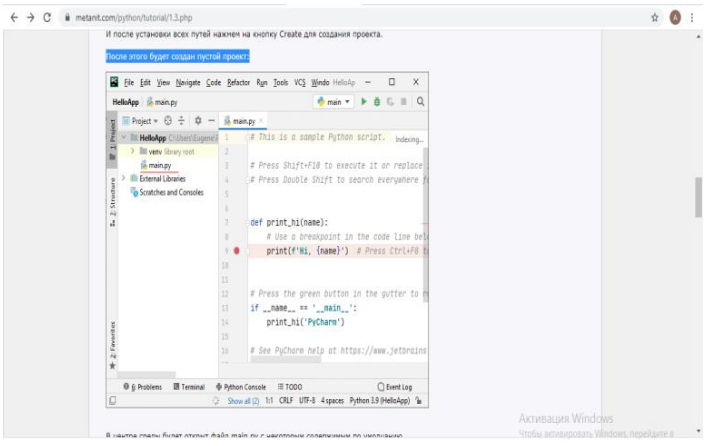
ыңғайлы, бірақ олардың әрқайсысы аудармашының белгілі бір нұсқасымен жұмыс істейді. Сонымен қатар, біз **existing Interpreter** өрісін таңдап, барлық жобалар үшін интерпретатор файлының жолын жаһандық түрде орната аламыз.



Сурет 1.9 – Жобаны баптау терезесі

Шын мәнінде, PyCharm-дегі алғашқы қарапайым бағдарлама үшін интерпертатордың қалай орнатылатыны маңызды емес. Алайда, бұл жағдайда біз әдепкі бойынша таңдалған **New Environment Using** жалаушасын қалдырамыз және оның астында **base Interpreter** өрісінде орнату бірінші тақырыпта қарастырылған аудармашы файлының жолын көрсетеміз. Соңғы жасау опциясы **main.py welcome script** сізге жобаны жасау кезінде бірден main.py файлын қосуға мүмкіндік береді. Барлық жолдарды орнатқаннан кейін жобаны құру үшін Жасау түймесін басамыз.

Осыдан кейін бос жоба (Сурет 1.10) жасалады:

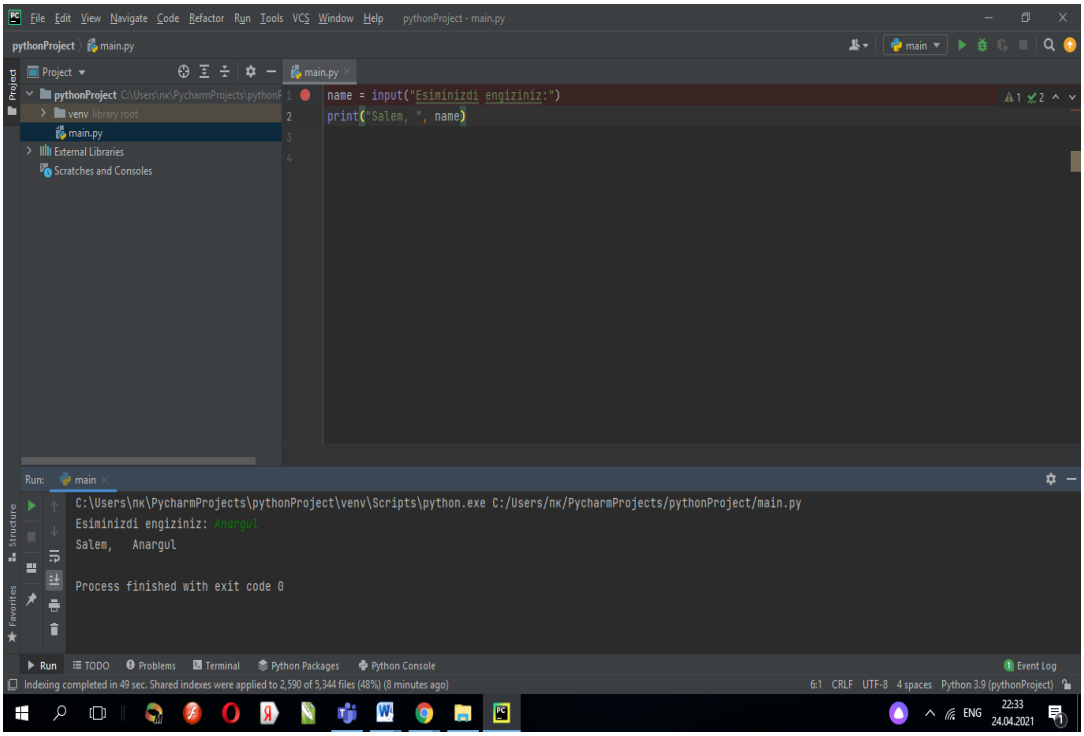


Сурет 1.10 –Бос жоба

Бағдарлама ортасында қандайда бір мазмұнды қамтитын main.py файлы ашылады. Енді қарапайым бағдарлама жасаймыз. main.py файлының кодын төмендегідей өзгертеміз:

```
name = input("Esiminizdi engiziniz: ")
print("Salem,", name)
```

Сценарийді іске қосу үшін бағдарламаның құралдар тақтасындағы жасыл көрсеткіні басамыз, сондай-ақ, іске қосу үшін Run мәзіріне өтіп, run 'main'тармақшасын басуға болады.Осыдан кейін, IDE-нің төменгі жағында шығару терезесі пайда болады, онда есіміңізді енгізу керек болады, содан кейін сәлемдесу көрсетіледі.(Сурет 1.11)



Сурет 1.11 –жоба нәтижесі

Visual Studio – дағы Python

Python-мен жұмыс істеуге мүмкіндік беретін өңдеу ортасының бірі-Visual Studio. Бұл IDE-нің PyCharm-мен салыстырғанда артықшылығы, ең алдымен, оның VS 2019 Community тегін басылымында бірқатар функциялар мен мүмкіндіктер бар, олар сол PyCharm-да тек ақылы Professional Edition-да қол жетімді. Мысалы, бұл веб-әзірлеу, оның ішінде әртүрлі фреймворктар. Сонымен қатар, Visual Studio-да Python-да әзірлеу құралдары тек Windows нұсқасында қол жетімді.

Сонымен, Visual Studio 2019 Community орнату файлын мына сілтемеден https://visualstudio.microsoft.com/downloads/ жүктеп алуға болады. Орнату файлын іске қосқаннан кейін біз Python орнатылған опцияларды таңдаймыз.

1.2 Python негіздері

Бағдарламаны жазуға кіріспе

Python тілінде бағдарлама нұсқаулықтар жиынтығынан тұрады. Әрбір нұсқаулық жаңа жолға орналастырылады. Мысалы:

```
1  print(2+3)
2  print("Hello")
```

Python-да шегіністер үлкен рөл атқарады. Дұрыс қойылмаған шегініс іс жүзінде қате болып табылады. Мысалы, код жоғарыда көрсетілгенге ұқсас болғанымен, келесі жағдайда қате пайда болады:

```
1 print(2+3)
2    print("Hello")
```

Сондықтан жаңа нұсқаулық жаңа жолдан басталу керек. Бұл Python-ның басқа тілдерден (C # немесе Java) айырмашылығы болып табылады. Бірақ тілдің кейбір құрылымы бірнеше жолдан тұрады. Мысалы, if шартты құрылымы

```
1 if1< 2:
2    print("Hello")
```

Бұл жағдайда, егер 1 2-ден кем болса, онда "Сәлем"жолы көрсетіледі. Мұнда шегініс болуы керек, өйткені print("Hello") нұсқаулығы("Сәлем") өздігінен емес, if шартты құрылымының бөлігі ретінде қолданылады. Сонымен қатар, шегініс, кодты жобалау жөніндегі нұсқаулыққа сәйкес, 4-ке көбейтілген бос орындардың санын жасаған жөн (яғни 4, 8, 16 және т.б.), егер шегініс 4 емес, 5 болса, онда бағдарлама да жұмыс істейді.

Регистрге тәуелділік

Python-бұл регистрге тәуелді тіл, сондықтан print және Print немесе PRINT өрнектері әртүрлі өрнектерді білдіреді. Егер консольге шығару үшін print әдісінің орнына Print әдісін қолдануға тырыссақ:

```
1  Print("Hello World")
```

қате шығады.

Түсініктемелер

Кодтың бір немесе басқа бөлігін жасайтын белгі үшін түсініктемелер қолданылады. Бағдарламаны тарату және орындау кезінде аудармашы түсініктемелерді елемейді, сондықтан олар бағдарламаның жұмысына әсер етпейді.
Python-дағы түсініктемелер блокты және жолды болып бөлінеді. Олардың барлығы тор белгісімен (#) алдын-ала белгіленеді.

Негізгі функциялар

Python бірқатар кіріктірілген функцияларды ұсынады. Олардың кейбіреулері әсіресе тіл үйренудің бастапқы кезеңдерінде өте жиі қолданылады.

Ақпаратты консольге шығарудың негізгі функциясы-**print ()** функциясы. Дәлел ретінде біз шығарғымыз келетін жол осы функцияға беріледі:

```
1  print("Hello Python")
```

Егер консольге бірнеше мәндерді шығару қажет болса, онда біз оларды үтір арқылы жаза аламыз:

```
1  print("Full name:", "Anar", "Bekenova")
```

Нәтижесінде барлық берілген мәндер бір жолдағы бос орындар арқылы жазылады:

```
print('Full name:', 'Anar', 'Bekenova')
```

Егер *print* функциясы шығару функциясы болса, онда *input* енгізу функциясы ақпаратты енгізуге жауап береді.

Айнымалылар және деректер типтері

Айнымалы белгілі бір деректерді сақтайды. Python-дағы айнымалы атау алфавиттік таңбадан немесе астын сызу белгісінен басталуы керек және алфавиттік-сандық таңбалар мен астын сызу белгісін қамтуы мүмкін. Сонымен қатар, айнымалы атау Python тілінің кілт сөздерінің атауымен сәйкес келмеуі керек.Кілт сөздер көп емес, оларды есте сақтау оңай: and, as, assert, break, class, continue, def, del, elif, else, except, False, finally, for, from, global, if, import, in, is, lambda, None, nonlocal, not, or, pass, raise, return, True, try, while, with, yield.

Мысалы, name айнымалысын жасайық:

name =”Anar”

Бұл жерде *”Anar”* жолын сақтайтын *name* айнымалысы анықталып тұр.

Python- да айнымалылар атауының екі типі бар: camel case және underscore notation.

camel case типінің мысалы: username = “Anar”

underscore notation типінің мысалы: user_ name = “Anar”

Сондай-ақ, регистрге тәуелділікті ескеретін болсақ, Name және Name айнымалылары әртүрлі нысандарды білдіреді.

Айнымалы бірдей деректер түрлерін сақтайды. Python-да көптеген деректер типтерінің жиынтығы бар, олар төмендегідей категорияларға бөлінеді: сандар, тізбектер, сөздіктер, жиынтықтар:

boolean - логическое значение True немесе False логикалық мәндері

int – бүтін сан, мысалы, 1, 4, 8, 50.

float – үтірмен берілетін нақты сан, мысалы, 1.2 немесе 34.76

complex – комплекс сандар

str - жолдар, мысалы "hello".

bytes - 0-255 диапазонындағы сандар тізбегі

byte array – байттар жиымы, bytes сияқты, тек өзгертуге болады

list - тізім

tuple - кортеж

set - бірегей объектілердің реттелмеген коллекциясы

frozen set – set сияқты, тек өзгермейді (immutable)

dict - сөздік, мұнда әрбір элементтің кілті мен мәні болады.

Python-динамикалық типтелген тіл. Ол берілген мән негізінде айнымалы деректер түрін анықтайды. Сонымен, қос немесе жалғыз тырнақшалардағы жолды белгілеу кезінде айнымалы str түріне ие болады. Бүтін санды тағайындау кезінде Python айнымалы түрін int ретінде автоматты түрде анықтайды. Айнымалыны float нысаны ретінде анықтау үшін оған бөлшек сан беріледі, онда бүтін және бөлшек бөлігі нүктемен бөлінеді. Өзгермелі нүкте санын экспоненциалды жазбада анықтауға болады:

```
1  x =3.9e3
2  print(x)  # 3900.0
3
4  x =3.9e-3
5  print(x)  # 0.0039
```

Float санында тек 18 маңызды таңба болуы мүмкін. Сонымен, бұл жағдайда тек екі таңба қолданылады - 3.9. Егер Сан тым үлкен немесе тым аз болса, онда экспонент көмегімен санды ұқсас белгіге жаза аламыз. Экспоненттен кейінгі сан негізгі санды 3.9- ға көбейтетін 10 санының дәрежесін көрсетеді.

Сонымен қатар, бағдарлама барысында айнымалы түрін оған басқа типтегі мән беру арқылы өзгерте аламыз:

```
1  user_id ="12tomsmith438"  # тип str
2  print(user_id)
3
4  user_id =234  # тип int
5  print(user_id)
```

Type() функциясының көмегімен ағымдағы айнымалы түрін динамикалық түрде білуге болады:

```
1  user_id ="12tomsmith438"
2  print(type(user_id))  # <class 'str'>
3
4  user_id =234
5  print(type(user_id))  # <class 'int'>
```

Сандармен операциялар

Арифметикалық операциялар

Python барлық жалпы арифметикалық амалдарды қолдайды:

Екі санды қосу:

```
1  print(6+2)  # 8
```

Екі санды азайту:

```
1  print(6-2)  # 4
```

Екі санды көбейту:

```
1  print(6*2)  # 12
```

Екі санды бөлу:

```
1  print(6/2)  # 3.0
```

Екі санды бүтін санды бөлу:

```
1  print(7/2)  # 3.5
2  print(7//2)  # 3
```

Бұл операция санның бөлшек бөлігін алып тастап, бүтінін шығарады.

Дәрежеге шығару:

```
1  print(6**2)
```

Қалдығын табу:

```
1    print(7%2)  # нәтижесі – 1
```

Төмендегідей өрнекті орындайық:

```
1  number =3+4*5**2+7
2  print(number)  # 110
```

Операциялардың ретін қайта анықтау үшін жақшаларды пайдалануға болады:

```
1  number =(3+4) *(5**2+7)
2  print(number)  # 224
```

Меншіктеуі бар арифметикалық операциялар

Бірқатар арнайы операциялар операцияның нәтижесін бірінші операндқа тағайындауға мүмкіндік береді:

+=
Қосу нәтижесін меншіктеу
-=
Азайту нәтижесін меншіктеу
*=
Көбейту нәтижесін меншіктеу
/=
Бөлу нәтижесін меншіктеу
//=
Бүтінсанды бөлу нәтижесін меншіктеу
**=
Санның дәрежесін меншіктеу
%=
Қалдықты меншіктеу

Операция мысалдары:

```
1    number =10
2    number +=5
3    print(number)  # 15
4
5    number -=3
6    print(number)  # 12
7
8    number *=4
9    print(number)  # 48
```

Сандарды түрлендіру функциялары:

Python-да бірқатар кіріктірілген функциялар сандармен жұмыс істеуге мүмкіндік береді. Атап айтқанда, int() және float() функциялары мәнді сәйкесінше int және float түріне келтіруге мүмкіндік береді.

Мысалы:

```
1  first_number ="2"
2  second_number =3
3  third_number =first_number +second_number
```

Біз "2" + 3 5-ке тең болады деп күтеміз. Алайда, бұл код ерекше жағдай туғызады, өйткені бірінші санның типі жолды білдіреді. Бәрі дұрыс жұмыс істеуі үшін int функциясын қолдана отырып, жолды санға келтіру керек():

```
1  first_number ="2"
2  second_number =3
3  third_number =int(first_number) +second_number
4  print(third_number) # 5
```

Float () функциясы сол сияқты әрекет етеді, ол санды өзгермелі нүктесі бар санға айналдырады. Бірақ тұтастай алғанда бөлшек сандармен операциялардың нәтижесі толығымен дәл болмауы мүмкін екенін ескеру қажет. Мысалы:

```
1  first_number =2.0001
2  second_number =5
3  third_number =first_number /second_number
4  print(third_number) # 0.40002000000000004
```

Бұл жағдайда нәтижені дөңгелектеу үшін round() функциясын қолдана аламыз:

```
1  first_number =2.0001
2  second_number =0.1
3  third_number =first_number +second_number
4  print(round(third_number, 4))  # 2.1001
```

Функцияның бірінші параметрі-дөңгелектелген сан, ал екіншісі-алынған сан үтірден кейін қанша таңбадан тұратындығын білдіреді.

Сандарды көрсету

Сандық айнымалының қалыпты анықтамасында ол ондық жүйеде мән алады. Бірақ Python-да ондықтан басқа, біз екілік, сегіздік және он алтылық жүйелерді қолдана аламыз. Екілік жүйеде санды анықтау үшін оның мәні алдында 0 және b префиксі қойылады:

```
1  x =0b101   # 101 екілік жүйеде 5 – ке тең
```

Сегіздік жүйеде санды анықтау үшін оның мәні алдында 0 және O префиксі қойылады:

```
1  a =0o11    # 11 сегіздік санау жүйесінде 9-ға тең
```

Он алтылық жүйеде санды анықтау үшін оның мәні алдында 0 және X префиксі қойылады:

```
1  y =0x0a        # a он алтылық санау жүйесінде 10-ға тең
```

Басқа өлшеу жүйелеріндегі сандармен арифметикалық амалдарды да жасауға болады:

```
1  x =0b101   # 5
2  y =0x0a    # 10
3  z =x +y    # 15
4  print("{0} in binary {0:08b}    in hex {0:02x} in octal {0:02o}".format(z))
```

Әр түрлі есептеу жүйелеріндегі санды шығару үшін жолда шақырылатын format функциясы қолданылады. Бұл жолға әртүрлі форматтар жіберіледі. Екілік жүйе үшін " {0: 08b}", мұнда 8 саны Сан жазбасында қанша таңба болуы керек екенін көрсетеді. Егер таңбалар Сан үшін талап етілгеннен көп болса, онда қажет емес позициялар нөлдермен толтырылады. Он алтылық жүйе үшін "{0:02x}"пішімі қолданылады. Мұнда бәрі бірдей-санның жазбасы екі таңбадан тұрады, егер бір белгі қажет болмаса, оның орнына нөл енгізіледі. Ал сегіздік санау жүйесінде жазу үшін "{0:02 o}"пішімі қолданылады.

Шартты өрнектер

Бірқатар операциялар шартты өрнектерді білдіреді. Барлық осы операциялар екі операндты қабылдайды және Python-да **boolean** түрін білдіретін логикалық мәнді қайтарады. Тек екі логикалық мән бар - **True** (өрнек шын) және **False** (өрнек жалған).

Салыстыру операциялары

Қарапайым шартты өрнектер екі мәнді салыстыратын салыстыру амалдарын білдіреді. Python келесі салыстыру әрекеттерін қолдайды:

==

Егер екі операнд тең болса, онда True мәнін қабылдайды. Әйтпесе False мәнін қабылдайды.

!=

Егер екі операнд тең емес болса True мәнін қабылдайды, басқаша жағдайда False мәнін қабылдайды.

> (артық)

Егер бірінші операнд екіншісінен артық болса True мәнін қабылдайды.

< (кем)

Егер бірінші операнд екіншісінен кем болса True мәнін қабылдайды

>= (артық немесе тең)

Егер бірінші операнд артық немесе тең болса True мәнін қабылдайды.
<= (кем немесе тең)
Егер бірінші операнд кем немесе тең болса True мәнін қабылдайды.
Салыстыру операцияларының мысалдары:

```
1  a =5
2  b =6
3  result =5==6  # операция нәтижесін айнымалыға сақтаймыз
4  print(result)  # False
5  print(a !=b)  # True
6  print(a > b)  # False
7  print(a < b)  # True
8
9  bool1 =True
10 bool2 =False
11 print(bool1 ==bool2)  # False
```

Салыстыру операциялары әртүрлі нысандарды - жолдарды, сандарды, логикалық мәндерді салыстыра алады, алайда операцияның екі операндасы да бірдей типті көрсетуі керек.

Логикалық операциялар

Күрделі шартты өрнектерді жасау үшін логикалық амалдар қолданылады. Python-да келесі логикалық операторлар бар:

Егер екі өрнек те шын болса, True қайтарады

```
1  age =22
2  weight =58
3  result =age > 21andweight ==58
4  print(result)  # True
```

Бұл жағдайда and операторы екі өрнектің нәтижелерін салыстырады: age > 21 weight == 58. Егер осы екі өрнек те True-ді қайтарса, онда and операторы да True-ді қайтарады. Сонымен қатар, салыстыру операциясы міндетті түрде өрнектердің бірі бола бермейді: бұл басқа логикалық операция немесе True немесе False сақтайтын boolean типті айнымалы болуы мүмкін.

```
 1  age =22
 2  weight =58
 3  isMarried =False
 4  result =age > 21andweight ==58andisMarried
 5  print(result)  # False, isMarried = False
```

or (логикалық өрнек)

Егер өрнектердің кем дегенде біреуі шын болса, True қайтарады

```
1  age =22
2  isMarried =False
3  result =age > 21orisMarried
4  print(result)  # True, так как выражение age > 21 равно True
```

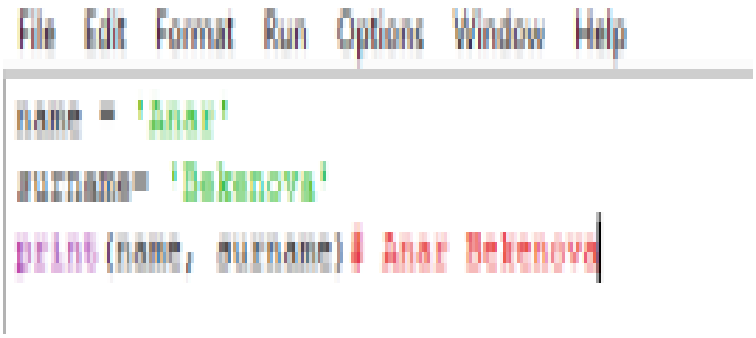
not (логикалық терістеу)

Егер өрнек жалған болса, True қайтарады.

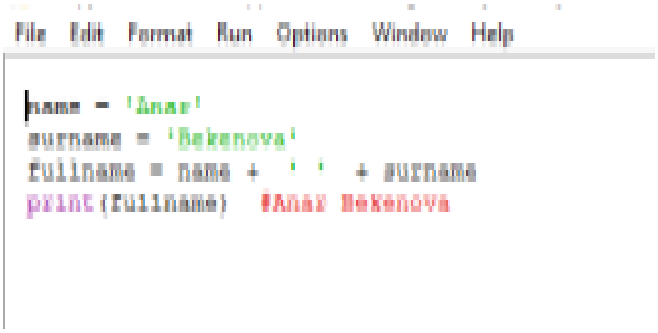
Егер and операторының операндтарының бірі жалған қайтарса, онда басқа операнд енді бағаланбайды, өйткені оператор кез-келген жағдайда жалған қайтарады. Мұндай мінез-құлық өнімділікті сәл арттыруға мүмкіндік береді, өйткені екінші операндты бағалауға ресурстарды жұмсаудың қажеті жоқ. Сол сияқты, егер or операторының операндтарының бірі True-ді қайтарса, онда екінші операнд бағаланбайды, өйткені оператор кез-келген жағдайда True-ді қайтарады.

Жолдармен операциялар

Жол тырнақшаға салынған Юникодты кодтаудағы таңбалар тізбегін білдіреді. Python-да біз жалғыз және қос тырнақшаларды қолдана аламыз:



Жолдардағы ең көп таралған операциялардың бірі-оларды біріктіру немесе байланыстыру. Жолдарды біріктіру үшін плюс белгісі қолданылады



Екі жолды біріктіргенде бәрі қарапайым, бірақ жол мен санды қосу керек болса ше? Бұл жағдайда str() функциясын қолдана отырып, санды жолға келтіру керек:

Стандартты таңбалардан басқа, жолдар арнайы түрде түсіндірілетін эскайп тізбегін басқаруды қамтуы мүмкін. Мысалы, \n тізбегі жолды келесі жолға көшіреді. Ал \t табуляция қосады .

Жолдарды салыстыру

```
1  str1 ="1a"
2  str2 ="aa"
3  str3 ="Aa"
4  print(str1 > str2)  # False
5  print(str2 > str3)  # True
```

Lower () функциясы жолды төменгі регистрге, ал **upper ()** функциясы жоғарғы регистрге ауыстырады.

If шартты конструкциясы

Шартты конструкциялар шартты өрнектерді қолданады және олардың мәніне байланысты бағдарламаның орындалуын жолдардың біріне бағыттайды. Осындай құрылымдардың бірі-*if* конструкциясы. Оның келесі ресми анықтамасы бар:

```
1  ifлогикалық өрнек:
2      нұсқаулықтар
3  [elifлогикалық өрнек:
4      нұсқаулықтар]
5  [else:
6      нұсқаулықтар]
```

Қарапайым түрде, if кілт сөзінен кейін логикалық өрнек пайда болады. Егер бұл логикалық өрнек true-ді қайтарса, онда келесі нұсқаулар блогы орындалады, олардың әрқайсысы жаңа ағыннан басталып, жолдың басынан шегіністер болуы керек:

```
1  age =22
2  ifage > 21:
3      print("рұқсат болады.")
4  print("Жұмыс аяқталды.")
```

Бұл жағдайда age айнымалысы мәні 21-ден үлкен болғандықтан, if блогы орындалады, ал консоль келесі жолдарды шығарады:

Рұқсат болады.
Жұмыс аяқталды.

Шегіністі 4 бос орынға немесе 4-ке көбейтілген бос орындар санына жасаған жөн.

"Жұмысты аяқтау" хабарын көрсететін соңғы жолдағы кодқа назар аударыңыз. Жолдың басынан ешқандай шегініс жоқ, сондықтан ол *if* блогына жатпайды және егер *if* конструкциясындағы өрнек жалған болсада кез-келген жағдайда орындалады.

Бірақ егер біз шегініс жасасақ, онда ол if конструкторына да қатысты болар еді:

```
1  age =22
2  ifage > 21:
3      print("Рұқсат болады.")
4      print("Жұмыс аяқталды.")
```

Егер кенеттен біз балама шешімді анықтауымыз керек болса, егер шартты өрнек жалған болса, онда біз **else** блогын қолдана аламыз:

```
1  age =18
2  ifage > 21:
3      print("Рұқсат беріледі")
4  else:
5      print("Рұқсат берілмейді")
```

age > 21 өрнегі ақиқат болса онда if блогы орындалады, әйтпесе else блогы орындалады.

Егер сізге бірнеше балама шарттарды енгізу қажет болса, онда сіз қосымша *elif* блоктарын қолдана аласыз, содан кейін нұсқаулық блогы бар.

```
1  age =18
2  ifage >=21:
3      print("Доступ разрешен")
4  elifage >=18:
5      print("Доступ частично разрешен")
6  else:
7      print("Доступ запрещен")
```

Кірістірілген if конструкциялары

If конструкциясы өз кезегінде кірістірілген if конструкцияларына ие болуы мүмкін:

```
1 age =18
2 ifage >=18:
3   print("17-ден артық")
4   ifage > 21:
5   print("21- ден артық")
6   else:
7   print(" 18-ден  21-ге дейін ")
```

Кірістірілген if өрнектері де шегіністерден басталуы керек, ал кірістірілген конструкциялардағы нұсқаулар да шегіністерге ие болуы керек. Дұрыс орналастырылмаған шегіністер бағдарламаның логикасын өзгерте алады. Сонымен, алдыңғы мысал келесіге ұқсас емес:

```
1  age =18
2  ifage >=18:
3      print("17- ден артық")
4  ifage > 21:
5      print("21- ден артық")
6  else:
7      print("18-ден 21- ге дейін")
```

Енді біз шартты конструкцияларды қолданатын шағын бағдарламаны жазамыз. Бұл бағдарлама айырбастау пунктінің бір түрі болады:

```
1    # Бағдарлама айырбастау пункті
2
3    usd =57
4    euro =60
5
6    money =int(input("Айырбастайтын валюта мәнін енгізіңіз: "))
7    currency =int(input("Валюта кодын көрсетіңіз (долларлар - 400, евро - 401): "))
8
9    ifcurrency ==400:
10       cash =round(money /usd, 2)
11       print("Валюта:  АҚШ доллары")
12   elifcurrency ==401:
13       cash =round(money /euro, 2)
14       print("Валюта: евро")
15   else:
16       cash =0
17       print("Белгісіз валюта")
18
19   print("Қолға алуға:", cash)
```

Input () функциясын қолдана отырып, пайдаланушы енгізген деректерді консольге аламыз. Сонымен қатар, бұл функция деректерді жол түрінде қайтарады, сондықтан енгізілген деректерді арифметикалық амалдарда қолдануға болатындай етіп int () функциясын қолдана отырып, оны бүтін санға келтіру керек.

Бағдарлама пайдаланушы айырбасталатын қаражат мөлшерін және айырбасталатын валюта кодын енгізетінін білдіреді. Валюта кодтары жеткілікті шартты: доллар үшін 400 және евро үшін 401.

If конструкторын қолдана отырып, біз валюта кодын тексеріп, тиісті валюта бағамына бөлеміз. Бөлу процесінде өзгермелі нүктесі бар өте ұзақ сан пайда болғандықтан, онда үтірден кейін көптеген белгілер болуы мүмкін, ол round() функциясын қолдана отырып, үтірден кейін екі санға дейін дөңгелектейді.

Соңында алынған мән консольге көрсетіледі. Мысалы, бағдарламаны іске қосып және кейбір деректерді енгіземіз:

Айырбастайтын валюта мәнін енгізіңіз:20000
Валюта кодын көрсетіңіз (долларлар - 400, евро - 401)
Валюта: евро
Қолға алуға: 333.33

Циклдар

Циклдар белгілі бір шартты сақтай отырып кейбір әрекеттерді қайталауға мүмкіндік береді.

while циклы

Біз қарастыратын бірінші цикл - while циклы. Ол төмендегідей формальды анықталады.

```
1 whileшартты_өрнек:
2    нұсқаулықтар
```

Циклге қатысты барлық нұсқаулар келесі жолдарда орналасқан және жолдың басынан шегініс болуы керек.

```
1 choice ="y"
2
3 whilechoice.lower() =="y":
4    print("Сәлем")
5    choice =input("Жалғастыру үшін Y басыңыз, ал шығу үшін кез келген
6 пернені басыңыз: ")
  print("бағдарлама жұмысы аяқталды")
```

Бұл жағдайда while циклі "Y" немесе "y"латын әрпі болған кезге дейін жалғасады.

Цикл блогының өзі екі нұсқаулықтан тұрады. Алдымен "сәлем" хабары көрсетіледі, содан кейін choice айнымалысы үшін жаңа мән енгізіледі. Егер пайдаланушы Y-ден басқа пернені басса, циклден шығу пайда болады, өйткені таңдау шарты choice.lower() == "y" False мәнді қабылдайды. Мұндай циклдің әр өтуі *итерация* деп аталады.

Сондай-ақ, соңғы басып шығару нұсқауында ("бағдарламаның жұмысы аяқталды") жолдың басынан шегініс жоқ, сондықтан ол циклге кірмейді.

Мысал: факториалды есептеу

```
1 #! Факториал есептеу бағдарламасы
2
3 number =int(input("Санды енгізіңіз: "))
4 i =1
5 factorial =1
6 whilei <=number:
7    factorial *=i
8    i +=1
9 print("Санның факториалы", number, "тең", factorial)
```

Мұнда ол консольден белгілі бір санды енгізеді, ал санауыш сан I енгізілген саннан үлкен болғанша, факториал санына көбейтілетін цикл орындалады.

Консольдық шығару:

Санды енгізіңіз:6

6 Санның факториалы 720 тең

for циклы

Циклдың тағы бір түрі for конструкциясы. For циклі белгілі бір сандар жиынтығындағы әр сан үшін шақырылады. Сандар жиынтығы range() функциясы арқылы жасалады. For циклінің формальды анықталуы:

```
forint_var inфункция_range:
    нұсқаулықтар
```

For кілт сөзінен кейін бүтін сандарды сақтайтын *int_var* айнымалысы келеді (айнымалы атауы кез келген болуы мүмкін), содан кейін *in* кілт сөзі, *range()* функциясын шақыру және қос нүкте.

Келесі жолдан бастап цикл нұсқауларының блогы орналасқан, олар жолдың басынан шегіністерге ие болуы керек.

Циклды орындау кезінде Python range функциясы жасаған коллекциядан барлық сандарды дәйекті түрде алады және сол сандарды int_var айнымалысында сақтайды. Бірінші өту кезінде цикл коллекциядан бірінші санды алады, екіншісінде - екінші сан және т.б., ол барлық сандарды сұрыптағанша ала береді. Коллекциядағы барлық сандар сұрыпталған кезде, цикл өз жұмысын аяқтайды.

Факториалды есептеу мысалында қарастырып көрейік:

```
#! Факториал есептеу бағдарламасы

number =int(input("Санды енгізіңіз: "))
factorial =1
for I in :range(1,number+1):
    factorial *=i
 print("Санның факториалы", number, "тең", factorial)


```

Алдымен консольден нөмірді енгізіңіз. Циклде біз range функциясы жасаған коллекциядағы сандар сақталатын I айнымалысын анықтаймыз.

Мұнда range функциясы екі аргументті қабылдайды - жинақтың бастапқы саны (мұнда 1 саны) және сандарды қосу керек сан (яғни number +1).

Консольден 6 саны енгізіледі делік, содан кейін range функциясын шақыру келесідей формада болады:

```
range(1, 6+1):
```

Бұл функция 1-ден басталатын және 7-ге дейін бүтін сандармен дәйекті түрде толтырылатын коллекция жасайды. Яғни, бұл [1, 2, 3, 4, 5, 6] жинағы болады.

Циклды орындау кезінде сандар осы коллекциядан I айнымалыға ауысады, ал циклдің өзінде I factorial айнымалыға көбейтіледі. Нәтижесінде біз санның факториалын аламыз.

Бағдарламаны консольдік шығару:

Санды енгізіңіз:6

Санның факториалы 6 тең 720

Range функциясы

Range функциясы келесі формаларға ие:

✓ range (stop): 0-ден stop-қа дейінгі барлық бүтін сандарды қайтарады

✓ range (start, stop): start (қоса) бастап stop (қоса) дейінгі аралықтағы барлық бүтін сандарды қайтарады. Жоғарыда факториалды есептеу бағдарламасында дәл осы форма қолданылды.

✓ range (start, stop, step): start (қоса алғанда) бастап stop (қоса алғанда) дейінгі аралықта бүтін сандарды қайтарады, олар step мәніне артады

range функциясын шақыру мысалдары:

```
1  range(5)          # 0, 1, 2, 3, 4
2  range(1, 5)       # 1, 2, 3, 4
3  range(2, 10, 2)   # 2, 4, 6, 8
4  range(5, 0, -1)   # 5, 4, 3, 2, 1
```

Мысалы, 0-ден 4-ке дейінгі барлық сандарды тізбекті түрде шығарамыз:

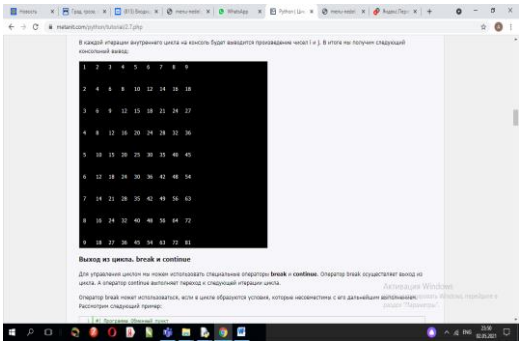
```
1  fori inrange(5):
2      print(i, end=" ")
```

Ішкі циклдар

Кейбір циклдар өздерінің ішінде басқа циклдерді қамтуы мүмкін. Көбейту кестесін шығару мысалын қарастырайық:

```
1  fori inrange(1, 10):
2      forj inrange(1, 10):
3          print(i *j, end="\t")
4      print("\n")
```

Range(1, 10) үшін сыртқы цикл 9 рет іске қосылады, өйткені range функциясымен қайтарылған коллекцияда 9 сан болады. Ішкі цикл for j in range (1, 10) сыртқы циклдің бір итерациясы үшін 9 рет, ал сыртқы циклдің барлық итерациялары үшін сәйкесінше 81 рет іске қосылады. Ішкі циклдің әр итерациясында i және j сандарының көбейтіндісі консольге шығарылады, нәтижесінде біз келесі консоль шығысын аламыз: i in range үшін сыртқы цикл(1, 10) 9 рет жұмыс істейді, өйткені range функциясы қайтарған коллекцияда 9 сан болады. Ішкі цикл for j in range (1, 10) сыртқы циклдің бір итерациясы үшін 9 рет, ал сыртқы циклдің барлық итерациялары үшін сәйкесінше 81 рет іске қосылады.Ішкі циклдің әр итерациясында і және j сандарының көбейтіндісі консольге шығарылады. (Сурет 1.12)



Сурет 1.12 – Көбейту кестесінің консольда шығарылуы

Циклдан шығу. break и continue

Циклды басқару үшін *break* және *continue* арнайы операторларын пайдалануға болады. ***Break***операторы циклден шығуды жүзеге асырады. Ал ***continue*** операторы циклдің келесі итерациясына көшуді орындайды.

Break операторын циклде оны одан әрі орындауға сәйкес келмейтін жағдайлар пайда болған кезде қолдануға болады. Келесі мысалды қарастырайық:

```
#! Бағдарлама Айырбастау пункті

print("Шығу үшін Y басыңыз")

whileTrue:
    data =input("Айырбастау мәнін енгізіңіз: ")
    ifdata.lower() =="y":
        break  # циклдан шығу
    money =int(data)
    cache =round(money /56, 2)
    print("Қолға алуға", cache, "доллар")

print("Айырбастау пунктінің жұмысы аяқталды")
```

Мұнда біз шексіз циклмен айналысамыз, өйткені while True шарты әрқашан шынайы және әрқашан орындалады. Бұл шексіз орындалуы керек бағдарламаларды құрудың танымал әдісі.

Циклде біз консольден кіріс аламыз. Пайдаланушы нөмірді енгізеді деп болжаймыз - айырбастау үшін шартты ақша сомасы. Егер пайдаланушы "Y" немесе "y" әрпін енгізсе, онда break операторының көмегімен біз циклден шығып, бағдарламаны тоқтатамыз. Әйтпесе, енгізілген соманы айырбастау бағамына бөлеміз, round функциясын қолдана отырып, нәтижені дөңгелектеп, оны консольге шығарамыз. Сонымен, пайдаланушы y пернесін басу арқылы бағдарламадан шыққысы келгенше шексіздікке дейін жалғастырамыз.

Шығу үшін Y басыңыз
Айырбастау мәнін енгізіңіз:20000
Қолға алуға: 357.14 доллар
Айырбастау мәнін енгізіңіз:Y
Айырбастау пунктінің жұмысы аяқталды

Бірақ пайдаланушы теріс санды енгізсе ше? Бұл жағдайда бағдарлама теріс нәтиже береді, бұл дұрыс емес. Бұл жағдайда, есептеуден бұрын, мәннің нөлден аз екенін тексере аламыз, ал егер аз болса, continue операторын қолдана отырып, оны аяқтамай циклдің келесі итерациясына ауысуды орындай аламыз:

```
1  #! Бағдарлама Айырбастау пункті
2
3  print("Шығу үшін Y басыңыз")
4
5  whileTrue:
6      data =input("Айырбастау мәнін енгізіңіз:")
7      ifdata.lower() =="y":
8          break  # циклдан шығу
9      money =int(data)
10     ifmoney < 0:
11         print("Қосынды оң болу керек!")
12         continue
13     cache =round(money /56, 2)
14     print("Қолға алуға", cache, "доллар")
15      print("Айырбастау пунктінің жұмысы аяқталды")
16
```

Сондай-ақ, нұсқаулық while блогына немесе кірістірілген if конструкциясына жататындығын анықтау үшін шегіністер қайтадан қолданылатынына назар аудару керек.

Бұл жағдайда біз теріс мән үшін нәтиже ала алмаймыз:

Шығу үшін Y басыңыз

Айырбастау мәнін енгізіңіз:-20000
Қосынды оң болу керек!
Айырбастау мәнін енгізіңіз:20000
Қолға алуға 357.14 доллар
Айырбастау мәнін енгізіңіз: у
Айырбастау пунктінің жұмысы аяқталды

Функциялар

Функциялар белгілі бір тапсырманы орындайтын және бағдарламаның басқа бөліктерінде қайта пайдалануға болатын код блогын білдіреді. Функцияны формальды анықтау:

```
1  def функция_аты ([параметрлер]):
2      нұсқаулықтар
```

Функцияның анықтамасы функцияның атауынан, параметрлері бар жақшалар жиынтығынан және қос нүктеден тұратын *def* өрнегінен басталады. Жақшадағы параметрлер міндетті емес. Келесі жолдан функция орындайтын нұсқаулар блогы келеді. Функцияның барлық нұсқауларында жолдың басынан шегіністер бар.

Мысалы, қарапайым функцияны анықтау:

```
1  defsay_hello():
2      print("Hello")
```

Функция say_hello деп аталады. Оның параметрлері жоқ және консольге "Hello"жолын көрсететін бір нұсқаулық бар.

Функцияны шақыру үшін функцияның атауы көрсетіледі, содан кейін жақшада оның барлық параметрлері үшін мәндер беріледі.

Мысалы:

```
1  defsay_hello():
2  print("Hello")
3
4  say_hello()
5  say_hello()
6  say_hello()
```

Мұнда say_hello функциясы қатарынан үш рет шақырылады. Нәтижесінде біз келесі консоль шығысын аламыз:

Hello
Hello
Hello

Енді параметрлермен функцияны анықтаймыз және қолданамыз:

```
1  defsay_hello(name):
2      print("Hello,",name)
3
4  say_hello("Anar")
5  say_hello("Sandu")
6  say_hello("Inabat")
```

Функция name параметрін қабылдайды және функция шақырылған кезде, параметрдің орнына мәнді жібере аламыз:

Hello Anar
Hello Sandu
Hello Inabat

Үнсіздік бойынша мәндер

Функцияның кейбір параметрлерін функцияны анықтаған кезде олар үшін үнсіздік бойынша мәндерді көрсету арқылы қосымша жасай аламыз.

Мысалы:

```
1  defsay_hello(name="Anar"):
2      print("Hello,", name)
3
4  say_hello()
5  say_hello("Anar")
```

Мұнда *name* параметрі міндетті емес. Егер функцияны шақырған кезде оған мән берілмесе, онда үнсіздік бойынша мән қолданылады, яғни "Anar"жолы.

Есімі бар параметрлер

Мәндерді беру кезінде функция оларды берілген ретпен параметрлермен салыстырады. Мысалы, келесі функция болсын:

```
1  defdisplay_info(name, age):
2      print("Name:", name, "\t", "Age:", age)
3
4  display_info("Anar", 22)
```

Функция шақырылған кезде бірінші "Anar" мәні бірінші параметрге - name параметріне, екінші мән - 22 саны екінші параметр - age параметріне беріледі. Мәселен, есімі бар параметрлерді пайдалану беру ретін қайта анықтауға мүмкіндік береді:

```
1  defdisplay_info(name, age):
2      print("Name:", name, "\t", "Age:", age)
3
4  display_info(age=22, name="Anar")
```

Есімі бар параметрлер функция шақырылған кезде оған мән бере отырып, параметр атауын көрсетуді білдіреді.

Параметрлердің белгісіз саны

Жұлдызша таңбасын пайдаланып параметрлердің белгісіз санын анықтауға болады:

```
1  defsum(*params):
2      result =0
3      forn inparams:
4          result +=n
5      returnresult
6
7
8  sumOfNumbers1 =sum(1, 2, 3, 4, 5)      # 15
9  sumOfNumbers2 =sum(3, 4, 5, 6)         # 18
10 print(sumOfNumbers1)
11 print(sumOfNumbers2)
```

Бұл жағдайда sum функциясы бір параметрді алады - *params, бірақ параметр атауының алдындағы жұлдызша, шын мәнінде, осы параметрдің орнына анықталмаған мәндер санын немесе мәндер жиынтығын бере алатынымызды көрсетеді. Функцияның өзінде for циклін қолдана отырып, сіз осы жиынтықтан өтіп, берілген мәндермен әртүрлі әрекеттер жасай аласыз. Мысалы, бұл жағдайда сандардың қосындысы қайтарылады.

Нәтижені қайтару

Функция нәтижені қайтара алады. Ол үшін функцияда қайтару операторы қолданылады, содан кейін қайтару мәні көрсетіледі:

```
1  defexchange(usd_rate, money):
2      result =round(money/usd_rate, 2)
3      returnresult
4
5  result1 =exchange(60, 30000)
6  print(result1)
7  result2 =exchange(56, 30000)
8  print(result2)
9  result3 =exchange(65, 30000)
10 print(result3)
```

Функция мәнді қайтаратындықтан, біз бұл мәнді кез-келген айнымалыға тағайындай аламыз, содан кейін оны қолдана аламыз: result2 = exchange(56, 30000).

Python-да функция бірден бірнеше мәнді қайтара алады:

```
1  defcreate_default_user():
2      name ="Tom"
3      age =33
4      returnname, age
5
6
7  user_name, user_age =create_default_user()
8  print("Name:", user_name, "\t Age:", user_age)
```

Мұнда create_default_user функциясы екі мәнді қайтарады: name және age. Функция шақырылған кезде бұл мәндер рет-ретімен user_name және user_age айнымалыларына беріледі және біз оларды қолдана аламыз.

main функциясы

Бағдарламада көптеген функцияларды анықтауға болады. Олардың барлығын ретке келтіру үшін арнайы негізгі функцияны қосу жақсы тәжірибе болып саналады, онда басқа функциялар шақырылады:

```
1   defmain():
2       say_hello("Tom")
3       usd_rate =56
4       money =30000
5       result =exchange(usd_rate, money)
6       print("К выдаче", result, "долларов")
7
8
9   defsay_hello(name):
10      print("Hello,", name)
11
12
13  defexchange(usd_rate, money):
14      result =round(money/usd_rate, 2)
15      returnresult
16
17  # Вызов функции main
18  main()
```

Айнымалылардың ауқымы

Ауқым немесе scope оны пайдалануға болатын айнымалының мәнмәтінін анықтайды. Python-да контексттің екі түрі бар: ғаламдық және жергілікті. Ғаламдық контекст айнымалы ол кез-келген функциядан тыс анықталады және бағдарламадағы кез-келген функцияға қол жетімді.

Мысалы:

```
1  name ="Anar"
2
3
4  defsay_hi():
5      print("Hello", name)
6
7
8  defsay_bye():
9      print("Good bye", name)
10
11 say_hi()
12 say_bye()
```

Мұнда *name* айнымалысы ғаламдық және ғаламдық ауқымға ие. Мұнда анықталған екі функция да оны еркін пайдалана алады.

Ғаламдық айнымалылардан айырмашылығы, жергілікті айнымалы функцияның ішінде анықталады және тек осы функциядан қол жетімді, яғни жергілікті ауқымға ие:

Бұл жағдайда екі функцияның әрқайсысында жергілікті айнымалы name анықталады. Бұл айнымалылар бірдей деп аталғанымен, бірақ бұл екі түрлі айнымалы, олардың әрқайсысы тек өз функциялары аясында қол жетімді. Сондай-ақ, say_hi функциясында surname айнымалысы анықталған, ол жергілікті, сондықтан say_bye функциясында біз оны пайдалана алмаймыз.

Жергілікті айнымалы бірдей атаумен глобалды айнымалыны жасырған кезде айнымалыны анықтаудың тағы бір нұсқасы бар:

```
1  name ="Anar"
2
3
4  defsay_hi():
5      print("Hello", name)
6
7
8  defsay_bye():
9      name ="Sandu"
10     print("Good bye", name)
11
12 say_hi()  # Hello Anar
13 say_bye()  # Good bye Sandu
```

Мұнда ғаламдық айнымалы *name* анықталады. Алайда, say_bye функциясында бірдей атауы бар жергілікті айнымалы анықталады. Егер say_hi функциясы ғаламдық айнымалыны қолданса, say_bye функциясы ғаламдық айнымалыны жасыратын жергілікті айнымалыны қолданады.

Егер біз жергілікті функцияда ғаламдық айнымалыны өзгерткіміз келсе және жергілікті айнымалыны анықтамасақ, онда *global* кілт сөзін қолдануымыз керек:

```
1  defsay_bye():
2      globalname
3      name ="Anar"
4      print("Good bye", name)
```

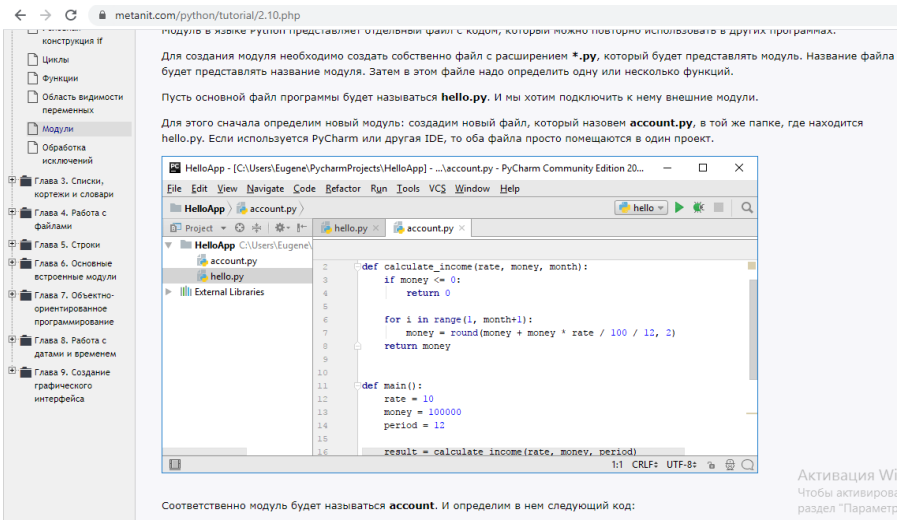
Python-да, көптеген басқа бағдарламалау тілдеріндегідей, ғаламдық айнымалыларды пайдалану ұсынылмайды. Жалғыз қолайлы тәжірибе-бұл бағдарлама барысында өзгермейтін ғаламдық тұрақтылардың аз санын анықтау.

```
1  PI =3.14
2
3
4  # дөңгелектің ауданын есептеу
5  defget_circle_square(radius):
6
7      print("радиусы бар дөңгелек ауданы", radius, "тең", PI *radius *radius)
8
9  get_circle_square(50)
```

Бұл жағдайда 3.14 саны PI тұрақтысымен ұсынылған. Бұл мән негізінен өзгермейді, сондықтан оны функциялардан шығарып, тұрақты ретінде анықтауға болады. Әдетте, тұрақты атау бас әріптермен анықталады.

Модульдер

Python тіліндегі Модуль басқа бағдарламаларда қайта пайдалануға болатын коды бар жеке файлды білдіреді. Модульді құру үшін модульді білдіретін *.py кеңейтімі бар нақты файлды жасау керек. Файл атауы модульдің атауын білдіреді. Содан кейін бұл файлда бір немесе бірнеше функцияны анықтау керек. Бағдарламаның негізгі файлы деп аталсын hello.py. біз оған сыртқы модульдерді қосқымыз келеді. Мұны істеу үшін алдымен жаңа модульді анықтаймыз: біз шақыратын жаңа файл жасаймыз account.py. орналасқан қалтада hello.py. егер PyCharm немесе басқа IDE қолданылса, онда екі файл да бір жобаға сәйкес келеді.(Сурет 1.13)



Сурет 1.13- Жаңа модуль

Тиісінше, модуль *account* деп аталады. Және онда келесі кодты анықтаймыз:

```
1  defcalculate_income(rate, money, month):
2      ifmoney <=0:
3          return0
4
5      fori inrange(1, month+1):
6          money =round(money +money *rate /100/12, 2)
7      returnmoney
```

Мұнда calculate_income функциясы анықталған, ол параметрлер ретінде салымның пайыздық мөлшерлемесін, салым сомасын және салым салынатын кезеңді алады және осы кезеңнің соңында алынған соманы есептейді.

hello.py файлында біз бұл модульді қолданамыз:

```
1  #! Бағдарлама Банк есебі
2  importaccount
3
4
5  rate =int(input("Пайыздық мөлшерлемені енгізіңіз: "))
6  money =int(input("Керекті қаражатты енгізіңіз: "))
7  period =int(input("Айдағы есеп жүргізу кезеңін енгізіңіз: "))
8
9  result =account.calculate_income(rate, money, period)
10 print("Есеп параметрлері:\n", "Қосынды: ", money, "\n", "Мөлшерлеме:
   ", rate, "\n",
11         "Кезең: ", period, "\n", "кезең соңындағы есеп қосындысы: ", result)
```

Модульді пайдалану үшін оны import операторының көмегімен импорттау керек, содан кейін модульдің аты көрсетіледі: import account.

Модульдің функционалдығына қол жеткізу үшін оның аттар кеңістігін алу керек. Үнсіздік бойынша, ол модульдің атымен сәйкес келеді, яғни біздің жағдайда ол да account деп аталады.

Модуль атауларының кеңістігін алғаннан кейін, біз оның функцияларына кеңістік схемасына сілтеме жасай аламыз:

```
1  account.calculate_income(rate, money, period)
```

Осыдан кейін біз негізгі сценарийді іске қоса аламыз hello.py және ол account.py модулін қамтиды, атап айтқанда, консольда шығыру келесідей болуы мүмкін:

```
Пайыздық мөлшерлемені енгізіңіз:10
Керекті қаражатты енгізіңіз:300000
Айдағы есеп жүргізу кезеңін енгізіңіз:6
Есеп параметрлері:
Қосынды:300000
Мөлшерлеме:10
Кезең: 6
```

Модуль аты

Жоғарыдағы мысалда модуль hello.py, ол негізгі болып табылады, account.py модульді пайдаланады. модульді іске қосу кезінде hello.py бағдарлама барлық қажетті жұмыстарды орындайды. Алайда, егер біз модульді бөлек іске қоссақ account.py өзі консольде ештеңе көрмейді. Өйткені, модуль функцияны анықтайды және басқа әрекеттерді орындамайды. Бірақ біз модульді жасай аламыз, account.py оны өздігінен де, басқа модульдерге де қосуға болады.

1.3 Тізімдер

Деректер жиынтығымен жұмыс істеу үшін Python тізімдер, түйіндер және сөздіктер сияқты кіріктірілген түрлерін ұсынады.

Тізім (list) элементтер жиынтығын немесе реттілігін сақтайтын деректер түрін білдіреді. Төртбұрышты жақшада ([]) тізімді құру үшін оның барлық элементтері үтір арқылы тізімделеді. Көптеген бағдарламалау тілдерінде массив деп аталатын ұқсас мәліметтер құрылымы бар.

Мысалы, сандар тізімін анықтайық:

```
1  numbers =[1, 2, 3, 4, 5]
```

Сондай-ақ, тізімді құру үшін **list** конструкторын пайдалануға болады():

```
numbers1 =[]
numbers2 =list()
```

Бұл тізім анықтамаларының екеуі де ұқсас-олар бос тізімді жасайды. Тізімді құру үшін list конструкторы басқа тізімді қабылдай алады:

```
numbers =[1, 2, 3, 4, 5, 6, 7, 8, 9]
numbers2 =list(numbers)
```

Тізім элементтеріне жүгіну үшін тізімдегі элемент нөмірін білдіретін *индекстерді* пайдалану керек. Индекстер нөлден басталады. Яғни, екінші элементте 1 индексі болады. Элементтерге қол жеткізу үшін -1-ден бастап теріс индекстерді қолдануға болады. Яғни, соңғы элементте -1 индексі, соңғы - -2 және т.б. болады.

```
numbers =[1, 2, 3, 4, 5]
print(numbers[0])   # 1
print(numbers[2])   # 3
print(numbers[-3])  # 3

numbers[0] =125  # тізімнің бірінші элементін өзгертеміз
print(numbers[0])   # 125
```

Егер сіз бірдей мәнді бірнеше рет қайталайтын тізімді жасауыңыз керек болса, онда жұлдызша *таңбасын пайдалануға болады. Мысалы, алты бестен тұратын тізімді анықтайық:

```
numbers =[5] *6  # [5, 5, 5, 5, 5, 5]
print(numbers)
```

Сонымен қатар, егер бізге сандардың дәйекті тізімі қажет болса, оны жасау үшін **range** функциясын қолдану ыңғайлы. Оның үш түрі бар:
range(end): 0-ден end санына дейінгі сандар жиынтығы жасалады
range(start, end): start санынан end санына дейінгі сандар жиынтығы жасалады
range(start, end, step): start санынан end санына дейінгі step қадаммен берілген сандар жиынтығы жасалады.

```
1  numbers =list(range(10))
2  print(numbers)      # [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
3  numbers =list(range(2, 10))
4  print(numbers)      # [2, 3, 4, 5, 6, 7, 8, 9]
5  numbers =list(range(10, 2, -2))
6  print(numbers)      # [10, 8, 6, 4]
```

Мысалы, тізімнің келесі екі анықтамасы ұқсас болады, бірақ range функциясы арқылы біз код көлемін азайтамыз:

```
1  numbers =[1, 2, 3, 4, 5, 6, 7, 8, 9]
2  numbers2 =list(range(1, 10))
```

Тізімде тек бірдей нысандар болуы міндетті емес. Біз бір уақытта бір тізімге жолдарды, сандарды, басқа деректер түрлерінің нысандарын орналастыра аламыз:

```
1  objects =[1, 2.6, "Hello", True]
```

Элементтерді сұрыптау үшін *for* және *while* циклдары қолданылады.

```
1  companies = ["Microsoft", "Google", "Oracle", "Apple"]
2  for item in companies:
3      print(item)
```

Мұнда **range** функциясының орнына біз қол жетімді компаниялар тізімін бірден алмастыра аламыз.

While циклі арқылы сұрыптау:

```
1  companies =["Microsoft", "Google", "Oracle", "Apple"]
2  i =0
3  whilei < len(companies):
4      print(companies[i])
5      i +=1
```

Len () функциясын пайдаланып санау үшін біз тізімнің ұзындығын аламыз.

Тізімдерді салыстыру

Екі тізім бірдей элементтер жиынтығын қамтыса, тең деп саналады:

```
numbers =[1, 2, 3, 4, 5, 6, 7, 8, 9]
numbers2 =list(range(1,10))
ifnumbers ==numbers2:
print("numbers equal to numbers2")
else:
print("numbers is not equal to numbers2")
```

Бұл жағдайда екі тізім де тең болады.

Тізімдермен жұмыс істеу әдістері мен функциялары

Элементтерді басқару үшін тізімдерде бірқатар әдістер бар. Олардың кейбіреулері:

append(item): тізімнің соңына item элементі қосылады

insert(index, item): тізімге item элементін index индексі бойынша қосады

remove(item): item элементін жояды.

clear(): тізімдегі барлық элементтерді жояды.

index(item): возвращает индекс элемента item элементінің индексін қайтарады.

pop([index]): удаляет и возвращает элемент по индексу index индексіндегі элементті жояды және қайтарады.

count(item): item элементінің тізімдегі санын анықтайды.

sort([key]): элементтерді сұрыптайды. Үнсіздік бойынша өсуі бойынша сұрыптайды. key параметрінің көмегімен сұрыптау функцияларын бере аламыз.

reverse(): тізімдегі барлық элементтерді кері ретпен жазады.

Сонымен қатар, Python тізімдермен жұмыс істеу үшін бірқатар кіріктірілген функцияларды ұсынады:

len(list): тізімнің ұзындығын көрсетеді.

sorted(list, [key]): сұрыпталған тізімді береді.

min(list): тізімнің ең кіші элементін табады.

max(list): тізімнің ең үлкен элементін табады.

Элементтерді қосу және жою

Элементті қосу үшін *append()* және *insert* әдістері қолданылады, ал жою үшін *remove(), pop()* және *clear ()* әдістері қолданылады.

Әдістердің қолданылуы:

```
users =["Tom", "Bob"]

users.append("Alice")  # ["Tom", "Bob", "Alice"]
users.insert(1, "Bill")         # ["Tom", "Bill", "Bob", "Alice"]

i =users.index("Tom")
removed_item =users.pop(i)           # ["Bill", "Bob", "Alice"]

```

```
last_user =users[-1]
users.remove(last_user)          # ["Bill", "Bob"]

print(users)

users.clear()
```

1.4 Кортеждер және сөздіктер

Кортеждер

Кортеж (tuple) элементтердің тізбегін білдіреді, олар тізімге ұқсас, тек кортеж өзгермейтін (immutable) түрі болып табылады. Сондықтан біз кортеж элементтерін қосып немесе алып тастай алмаймыз, оны өзгерте де алмаймыз. Кортежді жасау үшін оның мәндері үтірмен бөлінген жақшалар қолданылады:

```
user =("Tom", 23)
print(user)
```

Сондай-ақ, кортежді анықтау үшін біз жақшаларды қолданбай үтір арқылы мәндерді тізімдей аламыз:

```
user ="Tom", 23
print(user)
```

Егер кортеж бір элементтен тұрса, онда бір элементтен кейін үтір қою керек:

```
user =("Tom",)
```

Тізімнен кортеж жасау үшін тізімді кортежді қайтаратын tuple() функциясына беруге болады:

```
users_list =["Tom", "Bob", "Kate"]
users_tuple =tuple(users_list)
print(users_tuple)      # ("Tom", "Bob", "Kate")
```

Кортеж ішіндегі элементтерге жүгіну индекс бойынша тізімдегі сияқты жүреді. Индекстеу сонымен қатар элементтерді тізімнің басынан алған кезде нөлден басталады және элементтерді тізімнің соңынан алған кезде -1 басталады:

```
users =("Tom", "Bob", "Sam", "Kate")
print(users[0])    # Tom
print(users[2])    # Sam
print(users[-1])    # Kate

# кортеж бөлігін екінші элементтен 4- элементке дейін аламыз.
print(users[1:4])      # ("Bob", "Sam", "Kate")
```

Бірақ кортеж өзгермейтін тип болғандықтан (immutable), біз оның элементтерін өзгерте алмаймыз. Яғни, келесі жазба жұмыс істемейді:

```
users[1] ="Tim"
```

Қажет болған жағдайда, біз кортеждерді жеке айнымалыларға жіктей аламыз:

```
user =("Tom", 22, False)
name, age, isMarried =user
print(name)          # Tom
print(age)           # 22
print(isMarried)       # False
```

Функциядан бірден бірнеше мәнді қайтару қажет болған кезде кортеждерді пайдалану ыңғайлы. Функция бірнеше мәнді қайтарған кезде, ол іс жүзінде кортежге оралады:

```
defget_user():
    name ="Tom"
    age =22
    is_married =False
    returnname, age, is_married


user =get_user()
print(user[0])          # Tom
print(user[1])          # 22
print(user[2])          # False
```

Кірістірілген **Len()** функциясын қолдана отырып, кортеж ұзындығын ала аламыз:

```
user =("Tom", 22, False)
print(len(user))       # 3
```

Кортеждердің қайталануы

Кортежді қайталау үшін және *while* стандартты циклдері қолданылады. ***for***циклы арқылы:

```
user =("Tom", 22, False)
foritem inuser:
    print(item)
```

while циклы көмегімен:

```
user =("Tom", 22, False)

i =0
whilei < len(user):
    print(user[i])
    i +=1
```

Күрделі кортеждер

Бір кортежде элементтер түрінде басқа кортеждер болуы мүмкін. Мысалы:

```
countries =(
    ("Germany", 80.2, (("Berlin",3.326), ("Hamburg", 1.718))),
    ("France", 66, (("Paris", 2.2),("Marsel", 1.6)))
)

forcountry incountries:
    countryName, countryPopulation, cities =country
    print("\nCountry: {}  population: {}".format(countryName,
countryPopulation))
    forcity incities:
        cityName, cityPopulation =city
        print("City: {}  population: {}".format(cityName, cityPopulation))
```

Мұнда елдерді білдіретін *countries* кортежі әрқайсысы бөлек ел болатын кортеждерден тұрады. Кірістірілген кортеждердің үш элементі бар: елдің атауы, оның халқы мен қаласы. Қалалар жеке кортежді білдіреді, мұнда әрбір жеке Қала - бұл қаланың атауы мен оның тұрғындарының саны бар ішкі кортеж.

Сөздіктер

Тізімдер мен кортеждермен бірге, Python-да ***сөздік*** (dictionary) деп аталатын тағы бір мәліметтер құрылымы бар. Бірқатар бағдарламалау тілдерінде ұқсас құрылымдар бар (C# сөздік, PHP-дегі ассоциативті массив). Тізім сияқты, сөздік элементтер жинағын сақтайды. Сөздіктегі әр элементтің өзіндік мәні бар ерекше кілті болады.

Сөздік анықтамасының синтаксисі төмендегідей:

```
dictionary ={ ключ1:значение1, ключ2:значение2, ....}
```

Бірақ кілттер мен жолдар бірдей болуы міндетті емес. Олар әр түрлі болуы мүмкін:

```
objects ={1: "Tom", "2": True, 3: 100.6}
```

Сондай-ақ, біз бос сөздікті элементтерсіз анықтай аламыз:

```
objects ={}
```

немесе

```
objects =dict()
```

Тізімнен сөздікке түрлендіру

Сөздік пен тізім құрылымнан өзгеше болғанымен, тізімдердің жекелеген түрлері үшін оларды dict () функциясын қолдана отырып сөздікке айналдыру мүмкіндігі бар. Ол үшін тізім кірістірілген тізімдер жиынтығын сақтауы керек. Әрбір кірістірілген тізім екі элементтен тұруы керек-сөздікке айналдырылған кезде бірінші элемент кілт болады, ал екіншісі-мән:

```
users_list =[
    ["+111123455", "Tom"],
    ["+384767557", "Bob"],
    ["+958758767", "Alice"]
]
users_dict =dict(users_list)
print(users_dict)  # {"+111123455": "Tom", "+384767557": "Bob",
"+958758767": "Alice"}
```

Сол сияқты, сөздікке екі өлшемді кортеждерді түрлендіруге болады, олар өз кезегінде екі элементтен тұрады:

```
users_tuple =(
    ("+111123455", "Tom"),
    ("+384767557", "Bob"),
    ("+958758767", "Alice")
)
users_dict =dict(users_tuple)
print(users_dict)
```

Сөздік элементтеріне кіру үшін кілтті пайдалану керек:

```
dictionary[ключ]
```

Сөздіктерді көшіру және біріктіру

Copy() әдісі жаңа сөздікті қайтару арқылы сөздік мазмұнын көшіреді:

```
users ={"+1111111": "Tom","+3333333": "Bob","+5555555": "Alice"}
users2 =users.copy()
```

Update () әдісі екі сөздікті біріктіреді:

```
users ={"+1111111": "Tom","+3333333": "Bob","+5555555": "Alice"}

users2 ={"+2222222": "Sam","+6666666": "Kate"}
users.update(users2)

print(users)    # {"+1111111": "Tom", "+3333333": "Bob", "+5555555":
"Alice", "+2222222": "Sam", "+6666666": "Kate"}
print(users2)   # {"+2222222": "Sam", "+6666666": "Kate"}
```

Кешенді сөздіктер

Сандар мен жолдар сияқты қарапайым объектілерден басқа, сөздіктер күрделі нысандарды да сақтай алады - бірдей тізімдер, кортеждер немесе басқа сөздіктер:

```
users ={
    "Tom": {
        "phone": "+971478745",
        "email": "tom12@gmail.com"
    },
    "Bob": {
        "phone": "+876390444",
        "email": "bob@gmail.com",
        "skype": "bob123"
    }
}
```

Бұл жағдайда сөздіктің әр элементінің мәні өз кезегінде жеке сөздікті білдіреді.

Кірістірілген сөздіктің элементтеріне жүгіну үшін сәйкесінше екі кілтті пайдалану керек:

```
old_email =users["Tom"]["email"]
users["Tom"]["email"] ="supertom@gmail.com"
```

Бірақ егер біз сөздікте жоқ кілттің мәнін алуға тырыссақ, Python KeyError-ны шығарады:

```
tom_skype =users["Tom"]["skype"]   # KeyError
```

Қатені болдырмау үшін сөздіктегі кілттің бар-жоғын тексеруге болады:

```
key ="skype"
ifkey inusers["Tom"]:
    print(users["Tom"]["skype"])
else:
    print("skype is not found")
```

1.5 Файлдармен жұмыс

Файлдарды ашу және жабу

Python көптеген әртүрлі файл түрлерін қолдайды, бірақ оларды шартты түрде екі түрге бөлуге болады: мәтіндік және екілік. Мәтіндік файлдар, мысалы, CVS, TXT, html кеңейтімі бар файлдар, жалпы, ақпаратты мәтіндік түрде сақтайтын кез келген файлдар. Екілік файлдар-бұл суреттер, аудио және бейне файлдар және т.б. файл түріне байланысты онымен жұмыс сәл өзгеше болуы мүмкін.

Файлдармен жұмыс істеу кезінде кейбір әрекеттер тізбегін сақтау керек:

1. open() әдісі көмегімен файлды ашу
2. read() әдісі көмегімен файлды оқу немесе write() әдісі көмегімен файлды жазу
3. close() әдісі көмегімен файлды жабу

Файлмен жұмыс істеуді бастау үшін оны келесі ресми анықтамасы бар ***open()*** функциясы арқылы ашу керек:

```
1  open(file, mode)
```

Функцияның бірінші параметрі файл жолын білдіреді. Файл жолы абсолютті болуы мүмкін, яғни диск әрпінен басталады, мысалы C://somedir/somefile.txt. Немесе салыстырмалы болуы мүмкін, мысалы, somedir/somefile.txt-бұл жағдайда файлды іздеу жұмыс істеп тұрған Python сценарийінің орналасқан жеріне қатысты болады.

Екінші берілетін аргумент - *mode* режим файлды ашу режимін біз онымен не істейтінімізге байланысты орнатады. 4 жалпы режим бар:

r (Read). Файл оқу үшін ашылады. Егер файл табылмаса, FileNotFoundError хабарламасы шығады.
w (Write). Файл жазу үшін ашылады.открывается для записи. Егер файл жоқ болса, ол жасалады. Егер ұқсас файл болса, онда ол қайтадан жасалады, сәйкесінше ондағы ескі деректер жойылады.
a (Append). Файл жазу алдында ашылады. Егер файл жоқ болса, ол жасалады. Егер ұқсас файл болса, онда деректер оның соңында жазылады.
b (Binary). Екілік файлдармен жұмыс істеу үшін қолданылады. Басқа режимдермен бірге қолданылады-w немесе r.
Файлмен жұмыс аяқталғаннан кейін оны close() әдісімен жабу керек . Бұл әдіс файлға қатысты барлық ресурстарды босатады.

Мысалы, "hello.txt " мәтіндік файлын жазу үшін ашамыз:

```
1  myfile =open("hello.txt", "w")
2
3  myfile.close()
```

Файлды ашқан кезде немесе онымен жұмыс істеу кезінде біз әртүрлі ерекшеліктерге тап болуымыз мүмкін, мысалы, оған қол жетімділік жоқ және т.б. бұл жағдайда бағдарлама қатеге түседі және оны орындау close әдісін шақыруға жетпейді, сәйкесінше файл жабылмайды.

```
1   try:
2       somefile =open("hello.txt", "w")
3       try:
4           somefile.write("hello world")
5       exceptException as e:
6           print(e)
7       finally:
8           somefile.close()
9   exceptException as ex:
10      print(ex)
```

Бұл жағдайда файлмен барлық жұмыс кірістірілген *try* блогында жүреді. Егер кенеттен қандай да бір ерекшелік пайда болса, онда кез-келген жағдайда *finally* блогында файл жабылады.
Дегенмен, ыңғайлы конструкция бар-***with*** конструкциясы:

```
1  with open(file, mode) as file_obj:
2      нұсқаулықтар
```

Бұл конструкция ашық файл үшін file_obj айнымалысын анықтайды және нұсқаулар жиынтығын орындайды. Оларды орындағаннан кейін файл автоматты түрде жабылады. ***With*** блогында нұсқауларды орындау кезінде қандай да бір ерекшеліктер болса да, файл әлі де жабылады.

Сонымен, алдыңғы мысалды қайта жазамыз:

```
1  with open("hello.txt", "w") as somefile:
2      somefile.write("hello world")
```

Мәтіндік файл

Жазбада мәтіндік файлды ашу үшін W (қайта жазу) немесе a (жазба) режимін қолдану керек. Содан кейін жазу үшін жазылатын жол берілетін write(str) әдісі қолданылады. Айта кету керек, бұл жол жазылған, сондықтан сандарды, басқа типтегі деректерді жазу қажет болса, оларды алдымен жолға түрлендіру керек.

Қандай да бір ақпаратты "hello.txt" файлына жазайық:

```
1  with open("hello.txt", "w") as file:
2      file.write("hello world")
```

Егер біз қазіргі Python сценарийі орналасқан қалтаны ашсақ, онда hello .txt файлын көреміз. Бұл файлды кез-келген мәтіндік редакторда ашуға болады және қажет болса өзгертуге болады. Енді осы файлға тағы бір жолды жазамыз:

```
1  with open("hello.txt", "a") as file:
2      file.write("\ngood bye, world")
```

Жазба файлдағы соңғы таңбаға жолдар қосу сияқты көрінеді, сондықтан жаңа жолдан жазба жасау қажет болса, "\n"эскап тізбегін пайдалануға болады. Нәтижесінде hello файлы.txt-де келесі мазмұн болады:

```
1  hello world
2  good bye, world
```

Файлға жазудың тағы бір тәсілі деректерді консольға шығару үшін қолданылатын print() стандартты әдісі:

```
1  with open("hello.txt", "a") as hello_file:
2      print("Hello, world", file=hello_file)
```

print() әдісіне файлға шығару үшін Файл атауы файл параметрі арқылы екінші параметр ретінде беріледі. Ал бірінші параметр файлға жазылған жолды білдіреді.

Файлды оқу

Файлды оқу үшін ол r(Read) режимімен ашылады, содан кейін оның мазмұнын әртүрлі әдістермен оқуға болады:

- ✓ readline (): файлдан бір жолды оқиды
- ✓ read (): файлдың барлық мазмұнын бір жолға оқиды
- ✓ readlines (): файлдың барлық жолдарын тізімге оқиды

Мысалы, жоғарыда жазылған файлды жол бойынша қарастырамыз:

```
1  with open("hello.txt", "r") as file:
2      forline infile:
3          print(line, end="")
```

Біз әр жолды оқу үшін readline() әдісін нақты қолданбайтынымызға қарамастан, бірақ файлды сұрыптау кезінде бұл әдіс әр жаңа жолды алу үшін автоматты түрде шақырылады. Сондықтан циклде қолмен readline әдісін шақырудың қажеті жоқ. Жолдар "\n" жолының аударма символымен

бөлінгендіктен, басқа жолға артық берілуді болдырмау үшін end=""мәні басып шығару функциясына беріледі.
Енді жолдарды оқу үшін readline () әдісін шақырамыз:

```
1  with open("hello.txt", "r") as file:
2      str1 =file.readline()
3      print(str1, end="")
4      str2 =file.readline()
5      print(str2)
```

Консольді шығару:

```
hello world
good bye, world
```

Readline әдісін файлды while циклінде жол бойында оқу үшін пайдалануға болады:

```
1  with open("hello.txt", "r") as file:
2      line =file.readline()
3      whileline:
4          print(line, end="")
5          line =file.readline()
```

Егер файл кішкентай болса, оны бірден read әдісін қолдана отырып оқуға болады():

```
1    with open("hello.txt", "r") as file:
2        content =file.read()
3        print(content)
```

Сондай-ақ, бүкіл файлды жолдар тізіміне оқу үшін readlines () әдісін қолданамыз:

```
1  with open("hello.txt", "r") as file:
2      contents =file.readlines()
3      str1 =contents[0]
4      str2 =contents[1]
5      print(str1, end="")
6      print(str2)
```

Файлды оқу кезінде біз оның кодталуы ASCII-ге сәйкес келмейтіндігіне тап болуымыз мүмкін. Бұл жағдайда біз ***encoding*** параметрін қолдана отырып кодтауды нақты көрсете аламыз:

```
filename ="hello.txt"
with open(filename, encoding="utf8") as file:
    text =file.read()
```

Енді біз пайдаланушы енгізген жолдар массивін жазып, оны файлдан консольге қайта оқитын шағын сценарий жазамыз:

```
# file ati
FILENAME ="messages.txt"
# bos tizim aniktaimiz
messages =list()

fori inrange(4):
    message =input("zoldi engiziniz "+str(i+1) +": ")
    messages.append(message +"\n")

# tizimdi filege zhazu
with open(FILENAME, "a") as file:
    formessage inmessages:
        file.write(message)

# fileden habarlama  okimiz
print("okilgan habarlama")
with open(FILENAME, "r") as file:
    formessage infile:
        print(message, end="")
```

Бағдарлама жұмысының мысалы:

```
zoldi engiziniz 1: hello
zoldi engiziniz 2: world peace
zoldi engiziniz 3: great job
zoldi engiziniz 4: Python
okilgan habarlama hello
world peace
great job
Python
```

CSV файлдары

Ақпаратты ыңғайлы түрде сақтайтын кең таралған файл форматтарының бірі-csv пішімі. Csv файлындағы әр жол үтірмен бөлінген жеке бағандардан тұратын жеке жазбаны немесе жолды білдіреді. Сондықтан формат Comma Separated Values деп аталады. Бірақ csv пішімі мәтіндік файл пішімі болғанымен, Python онымен жұмыс істеуді жеңілдету үшін арнайы орнатылған csv модулін ұсынады.

Мысал арқылы модульдің жұмысын қарастырайық:

```
1  Importcsv
2
3  FILENAME ="users.csv"
4
5  users =[
6      ["Talgat", 28],
7      ["Alma", 23],
8      ["Ali", 34]
9  ]
10
11 with open(FILENAME, "w", newline="") as file:
12     writer =csv.writer(file)
13     writer.writerows(users)
14
15
16 with open(FILENAME, "a", newline="") as file:
17     user =["Sam", 31]
18     writer =csv.writer(file)
19     writer.writerow(user)
```

Файлға екі өлшемді тізім жазылады - іс жүзінде әр жол бір пайдаланушыны білдіретін кесте. Әр пайдаланушының екі өрісі бар - аты мен жасы. Яғни, іс жүзінде үш жол мен екі бағаннан тұратын кесте.

Жазу үшін файлды ашқан кезде үшінші параметр ретінде new line=" мәні көрсетіледі - бос жол амалдық жүйеге қарамастан файлдағы жолдарды дұрыс оқуға мүмкіндік береді.

Жазу үшін csv функциясымен қайтарылатын writer нысанын алу керек.writer(file). Бұл функцияға ашық файл жіберіледі. Ал жазбаның өзі жазу әдісі арқылы жасалады.writerow (пайдаланушылар) бұл әдіс жолдар жиынтығын қабылдайды. Біздің жағдайда бұл екі өлшемді тізім.

Егер бір өлшемді тізім болып табылатын бір жазбаны қосу қажет болса, мысалы ["Samat", 31], онда бұл жағдайда writer әдісін шақыруға болады.writerow(user)

Нәтижесінде, сценарийді орындағаннан кейін, сол қалтада users файлы болады.келесі мазмұны бар csv:

```
1  Talgat,28
2  Alma,23
3  Ali,34
4  Samat,31
```

Файлдан оқу үшін, керісінше, ***reader*** нысанын жасау керек:

```
1  Importcsv
2
3  FILENAME ="users.csv"
4
5  with open(FILENAME, "r", newline="") as file:
6      reader =csv.reader(file)
7      forrow inreader:
8          print(row[0], " - ", row[1])
```

Reader нысанын алған кезде циклде оның барлық жолдарын сұрыптай аламыз:

```
Talgat  -  28
Alma  -  23
Ali  -  34
Samat - 31
```

Бинарлық (екілік) файлдар

Екілік файлдар мәтіндік файлдарға қарағанда ақпаратты байттар жиынтығы түрінде сақтайды. Олармен жұмыс істеу үшін Python - да кірістірілген Pickle модулі қажет. Бұл модуль екі әдісті ұсынады:

- ✓ dump (obj, file): obj нысанын екілік файлға жазады
- ✓ load (file): екілік файлдан объектіге деректерді оқиды

Оқу немесе жазу үшін екілік файлды ашқан кезде жазу ("w") немесе оқу ("r") режиміне қосымша "b" режимін қолдану керек екенін ескеру қажет. Екі нысанды сақтау керек делік:

```
1   Importpickle
2
3   FILENAME ="user.dat"
4
5   name ="Talgat"
6   age =19
7
```

```
8  with open(FILENAME, "wb") as file:
9      pickle.dump(name, file)
10     pickle.dump(age, file)
11
12 with open(FILENAME, "rb") as file:
13     name =pickle.load(file)
14     age =pickle.load(file)
15     print("Имя:", name, "\tВозраст:", age)
```

Dump функциясын қолдана отырып, екі нысан дәйекті түрде жазылады. Сондықтан, load функциясы арқылы дәйекті түрде оқығанда, біз бұл нысандарды оқи аламыз. Бағдарламаны консольдік шығару:

```
Имя: Talgat          Возраст: 28
```

Сол сияқты, біз файлдан объектілер жиынтығын сақтай және алып тастай аламыз:

```
1  Importpickle
2
3  FILENAME ="users.dat"
4
5  users =[
6      ["Talgat", 28, True],
7      ["Alma", 23, False],
8      ["Ali", 34, False]
9  ]
10
11 with open(FILENAME, "wb") as file:
12     pickle.dump(users, file)
13
14
15 with open(FILENAME, "rb") as file:
16     users_from_file =pickle.load(file)
17     foruser inusers_from_file:
18         print("Esimi:", user[0], "\tZhas:", user[1], "\tUilengen(turmista):",
   user[2])
```

Dump функциясымен қандай нысанды жазғанымызға байланысты, файлды оқу кезінде сол нысанды load функциясы қайтарады.
Консольді шығару:

```
Имя: Talgat      Zhas : 28      Uilengen(turmista): True
Имя: Alma        Zhas: 23              Uilengen(turmista): False
Имя: Ali         Zhas: 34              Uilengen(turmista): False
```

1.6 Жолдар

Жолдармен жұмыс

Жол Юникодты кодтаудағы таңбалар тізбегін білдіреді. Біз квадрат жақшадағы индекс бойынша жолдың жеке таңбаларына жүгіне аламыз: Индекстеу нөлден басталады, сондықтан жолдың бірінші таңбасында 0 индексі болады. Егер Біз жолда жоқ индекске жүгінуге тырыссақ, онда IndexError хабарламасын аламыз. Мысалы, жоғарыдағы жағдайда жолдың ұзындығы 11 таңбадан тұрады, сондықтан оның таңбаларында 0-ден 10-ға дейінгі индекстер болады. Жолдың соңынан бастап таңбаларға қол жеткізу үшін теріс индекстерді қолдануға болады. Сонымен, -1 индексі соңғы таңбаны, ал -2 - соңғы таңбаны және т. б. білдіреді:

```
1  string ="hello world"
2  c1 =string[-1]  # d
3  print(c1)
4  c5 =string[-5]  # w
5  print(c5)
```

Таңбалармен жұмыс жасау кезінде жолдың өзгермейтін (өзгермейтін) түрі екенін ескеру керек, сондықтан егер біз жолдың жеке таңбасын өзгертуге тырыссақ, онда келесі жағдайдағыдай қате пайда болады:

```
1  string ="hello world"
2  string[1] ="R"
```

ord және len функциялары

Жолдар Unicode символдарын қамтитындықтан, ord() функциясының көмегімен Unicode кодындағы символдарға арналған сандық мәнді аламыз:

```
1  print(ord("A"))    # 65
```

Жолдың ұзындығын алу үшін len() функциясын қолдануға болады:

```
1  string ="hello world"
2  length =len(string)
3  print(length)   # 11
```

Жолда іздеу

Term in string өрнегін қолдана отырып, string жолындағы term ішкі жолын табуға болады. Егер ішкі жол табылса, онда өрнек шын мәнін қайтарады, әйтпесе жалған мән қайтарылады:

```
1  string ="hello world"
2  exist ="hello"instring
3  print(exist)    # True
4
5  exist ="sword"instring
6  print(exist)    # False
```

Жолдардың қайталануы

For циклінің көмегімен жолдың барлық таңбаларын сұрыптай аламыз:

```
1  string ="hello world"
2  forchar instring:
3      print(char)
```

Жолдардың негізгі әдістері

Қосымшаларда қолдануға болатын жолдардың негізгі әдістерін қарастырайық:

✓ ***isalpha()***: егер жол тек алфавиттік таңбалардан тұрса, True қайтарады

✓ ***islower()***:егер жол тек кіші әріптегі таңбалардан тұрса, True қайтарады

✓ ***isupper()***:егер барлық жол таңбалары жоғарғы регистрде болса, True қайтарады

✓ ***isdigit()***:егер жолдың барлық таңбалары сандар болса, True қайтарады

✓ ***isnumeric()***:егер жол сан болса, True қайтарады

✓ ***startswith(str):*** егер жол str ішкі жолынан басталса, True қайтарады

✓ ***endswith(str):*** егер жол strr ішкі жолында аяқталса, True қайтарады

✓ ***lower()***:жолды төменгі регистрге аударады

✓ ***upper()***:жолды жоғарғы регистрге аударады

✓ ***title()***:жолдағы барлық сөздердің бастапқы таңбалары жоғарғы регистрге аударылады

✓ ***capitalize()***:жолдың ең бірінші сөзінің бірінші әрпін бас әріпке аударады

✓ ***lstrip():***жолдан бастапқы бос орындарды жояды

✓ ***rstrip():***соңғы бос орындарды жолдан алып тастайды

✓ ***strip():***жолдан бастапқы және соңғы бос орындарды жояды

✓ ***ljust(width):*** егер жолдың ұзындығы width параметрінен аз болса, жолдың оң жағында width мәнін қосу үшін бос орындар қосылады, ал жолдың өзі сол жаққа туралanады

✓ ***rjust(width):*** егер жолдың ұзындығы width параметрінен аз болса, жолдың сол жағында бос орындар width мәнін қосу үшін қосылады, ал жолдың өзі оң жаққа тураланады

✓ ***center(width):*** егер жолдың ұзындығы width параметрінен аз болса, жолдың сол және оң жағында Бос орындар width мәнін толықтыру үшін біркелкі қосылады, ал жолдың өзі ортаға тураланады

✓ ***find(str[, start [, end])***: жолдағы ішкі жол индексін қайтарады. Егер ішкі жол табылмаса, -1 саны қайтарылады

✓ ***replace(old, new[, num])***: жолдағы бір кіші жолды басқасына ауыстырады

✓ ***split([delimeter[, num]])***: бөлгішке байланысты жолды ішкі жолдарға бөледі

✓ ***join(strs):*** жолдарды бір жолға біріктіріп, олардың арасына белгілі бір бөлгіш салады.

Мысалы, егер біз пернетақтадан нөмірді енгізуді күтетін болсақ, онда енгізілген жолды санға түрлендірмес бұрын, isnumeric() әдісін қолдана отырып, санның шын мәнінде енгізілгенін тексеруге болады, егер солай болса, онда түрлендіру әрекетін орындаймыз:

```
1  string =input("Sandy engiziniz: ")
2  ifstring.isnumeric():
3      number =int(string)
4      print(number)
```

Белгілі бір ішкі жолға жол басталатындығын немесе аяқталатындығын тексеру:

```
1          file_name ="hello.py"
2
3          starts_with_hello =file_name.startswith("hello")   # True
4          ends_with_exe =file_name.endswith("exe")          # False
```

```
1  print("iPhone 7:", "52000".rjust(10))
2  print("Huawei P10:", "36000".rjust(10))
```

Жолды бос орындармен толықтыру және туралау:
Консольды шығару:

```
iPhone 7:      52000
Huawei P10:  36000
```

Форматтау

Жолдарда анықталған format() әдісі белгілі бір мәндерді ойнатқыштардың орнына қою арқылы жолды пішімдеуге мүмкіндік береді.

Жолға кірістіру үшін фигуралы жақшалармен ({}) жиектелген арнайы параметрлер қолданылады.

Аты бар параметрлер

Форматталатын жолда параметрлерді анықтай аламыз, FORMAT() әдісінде осы параметрлер үшін мәндерді беру:

```
text ="Hello, {first_name}.".format(first_name="Talgat")
print(text)     # Hello, Talgat.

info ="Name: {name}\t Age: {age}".format(name="Ali", age=23)
print(info)     # Name: Ali  Age: 23
```

Сонымен қатар, әдіске сәйкес, аргументтер жолдағы параметрлермен бірдей атпен анықталады. Сонымен, егер параметр бірінші жағдайдағыдай first_name деп аталса, онда мән берілген аргумент те first_name деп аталады.

Сөздерді санау бағдарламасы

Сөздерді санау бағдарламасын қарастыратын шағын мысалмен жолдармен жұмысты қарастырайық:

```
#! Файлдағы сөздерді санайтын бағдарлама
importos


defget_words(filename):

    with open(filename, encoding="utf8") as file:
        text =file.read()
    text =text.replace("\n", " ")
    text =text.replace(",", "").replace(".", "").replace("?", "").replace("!", "")
    text =text.lower()
    words =text.split()
    words.sort()
    returnwords


defget_words_dict(words):
    words_dict =dict()

    forword inwords:
        ifword inwords_dict:
            words_dict[word] =words_dict[word] +1
        else:
            words_dict[word] =1
    returnwords_dict


```

```
28 defmain():
29     filename =input("filega joldy engiziniz: ")
30     ifnotos.path.exists(filename):
31         print("korsetilgen file zhok")
32     else:
33         words =get_words(filename)
34         words_dict =get_words_dict(words)
35         print("cozder sany: %d"%len(words))
36         print("unicaldi cozder sany: %d"%len(words_dict))
37         print("barlik koldanylgan cozder:")
38         forword inwords_dict:
39             print(word.ljust(20), words_dict[word])
40
41
42 if__name__ =="__main__":
43     main()
```

Мұнда get_words () функциясында мәтіннің сөздерге бастапқы сегментациясы жасалады. Бұл ретте барлық пунктуациялық белгілер жойылады, ал жолдың аудармасы бос орындарға ауыстырылады. Содан кейін мәтін сөздерге бөлінеді. Бөлгіш ретінде бос орын әдепкі бойынша қолданылады.

Әрі қарай, get_words_dict() функциясында біз сөздерден сөздік аламыз, мұнда кілт - ерекше сөз, ал мағынасы - мәтіндегі осы сөздің пайда болу саны.

Негізгі функция файлға жолды енгізіп, жоғарыда аталған функцияларды шақырып, барлық статистиканы шығарады.

Бағдарламаның консольды шығарылуы:

```
filega joldy engiziniz: C:\SomeDir\hello.txt
cozder sany: 66
unicaldi cozder sany: 54
barlik koldanylgan cozder:
Astana          2
tu              1
bank            1
siz             1
aspan           1
ony             1
birge           3
tagy            1
```

1.7 Негізгі кіріктірілген модульдер

Random модулы

Random модулы кездейсоқ сандардың пайда болуын басқарады. Оның негізгі функциялары:

random(): кездейсоқ санды 0.0-ден 1.0-ге дейін жасайды
randint():кездейсоқ санды белгілі бір диапазоннан қайтарады
randrange():кездейсоқ санды белгілі бір сандар жиынынан қайтарады
shuffle(): тізімдерді араластырады
choice(): тізімнің кез келген элементін қайтарады

Random () функциясы 0.0-ден 1.0-ге дейінгі аралықта өзгермелі нүктесі бар кездейсоқ санды қайтарады. Егер бізге үлкен ауқымның саны қажет болса, 0-ден 100-ге дейін айтайық, сәйкесінше random функциясының нәтижесін 100-ге көбейте аламыз.

```
1 Importrandom
2
3 number =random.random()  #  0.0 ден 1.0 ге дейінгі мән
4 print(number)
5 number =random.random() *100  #  0.0 ден  100.0 ге дейінгі
6 мән
  print(number)
```

Randint(min, max) функциясы кездейсоқ бүтін санды екі min және max мәндерінің арасындағы бос орынға қайтарады.

```
1 Importrandom
2
3 number =random.randint(20, 35)  # 20 дан 30ға дейінгі мән
4 print(number)
```

Randrange () функциясы белгілі бір сандар жиынынан кездейсоқ бүтін санды қайтарады. Оның үш формасы бар:

randrange (stop): кездейсоқ мән алынған сандар жиынтығы ретінде 0-ден stop санына дейінгі диапазон қолданылады

randrange (start, stop): сандар жиынтығы start санынан stop санына дейінгі ауқымды білдіреді

randrange(start, stop, step): сандар жиынтығы start санынан stop санына дейінгі диапазонды білдіреді, ал диапазондағы әр сан алдыңғы қадамнан әр түрлі болады

```
1
2 Importrandom
3
4 number =random.randrange(10)
5 print(number)
6 number =random.randrange(2, 10)  print(number)
7 number =random.randrange(2, 10, 2)
8 print(number)
```

Тізіммен жұмыс

Random модуліндегі тізімдермен жұмыс істеу үшін екі функция анықталған: **shuffle ()** функциясы тізімді кездейсоқ түрде араластырады, ал **choice()** функциясы тізімнен бір кездейсоқ элементті қайтарады:

```
1  numbers =[1, 2, 3, 4, 5, 6, 7, 8]
2  random.shuffle(numbers)
3  print(numbers)
4  random_number =random.choice(numbers)
5  print(random_number)
```

Math модулы

Python-дағы кіріктірілген ***math модулі*** математикалық, тригонометриялық және логарифмдік операцияларды орындауға арналған функциялар жиынтығын ұсынады. Модульдің кейбір негізгі функциялары:

pow(num, power): num санын power дәрежесіне көтеру
sqrt(num): num санының квадрат түбірі
ceil(num): санды ең жақын бүтін санға дөңгелектеу
floor(num): санды ең кіші бүтін санға дөңгелектеу
factorial(num): санның факториалы
degrees(rad): радианнан градусқа ауыстыру
radians(grad): градустан радианға ауысу
cos(rad): радиандағы бұрыштың косинусы
sin(rad): радиандағы бұрыштың синусы
tan(rad): радиандағы бұрыштың тангенсі
acos(rad): радиандағы бұрыштың арккосинусы
asin(rad): радиандағы бұрыштың арксинусы
atan(rad): радиандағы бұрыштың арктангенсі
log(n, base): base негізінде N санының логарифмі
log10(n): N санының ондық логарифмі

Кейбір функцияларды қолдану мысалы:

```
1  Importmath
2
3  #  2-нің 3 дәрежесі
4  n1 =math.pow(2, 3)
5  print(n1)  # 8
6
7  # осы операцияны басқаша былай орындауға болады
8  n2 =2**3
9  print(n2)
```

```
10
11 #квадрат түбір табу
12 print(math.sqrt(9))  # 3
13
14 # бүтінге дейін дөңгелектеу
15 print(math.ceil(4.56))  # 5
16
17 # ең кіші бүтін сан
18 print(math.floor(4.56))  # 4
19
20 # радианнан градусқа ауыстыру
21 print(math.degrees(3.14159))  # 180
22
23 # градустан радианға ауыстыру
24 print(math.radians(180))   # 3.1415.....
25 # косинус
26 print(math.cos(math.radians(60)))  # 0.5
27 # синус
28 print(math.sin(math.radians(90)))   # 1.0
29 # тангенс
30 print(math.tan(math.radians(0)))    # 0.0
31
32 print(math.log(8,2))    # 3.0
33 print( (100 math.log10))    # 2.0
```

Сондай-ақ, math модулі PI және E сияқты бірқатар тұрақты мәндерді ұсынады:

```
1  Importmath
2  radius =30
3  # радиусы 30болатын дөңгелек ауданы
4  area =math.pi *math.pow(radius, 2)
5  print(area)
6
7  # 10 санының натурал логарифмі
8  number =math.log(10, math.e)
9  print(number)
```

Модуль decimal

Өзгермелі нүктелі сандармен (яғни, float) жұмыс жасап , біз кейде есептеу нәтижесінде дұрыс нәтиже ала алмаймыз:

```
1  number =0.1+0.1+0.1
2  print(number)       # 0.30000000000000004
```

Мәселені санды дөңгелектейтін round() функциясын қолдану арқылы шешуге болады. Дегенмен, кіріктірілген decimal модулін қолданудың тағы бір жолы бар. Бұл модульдегі сандармен жұмыс істеудің негізгі компоненті-Decimal класы. Оны қолдану үшін біз конструктордың көмегімен оның нысанын жасауымыз керек. Жол мәні конструкторға беріледі, ол санды білдіреді:

```
1  fromdecimal importDecimal
2
3  number =Decimal("0.1")
```

Осыдан кейін Decimal нысанын арифметикалық амалдарда пайдалануға болады:

```
1  fromdecimal importDecimal
2
3  number =Decimal("0.1")
4  number =number +number +number
5  print(number)       # 0.3
```

Decimal операцияларында бүтін сандарды пайдалануға болады:

```
1  number =Decimal("0.1")
2  number =number +2
```

Алайда, float және Decimal бөлшек сандарын операцияларда араластыруға болмайды:

```
1    number =Decimal("0.1")
2    number =number +0.1
```

Алайда, сіз операцияларда float және Decimal бөлшек сандарын араластыра алмайсыз: қосымша таңбалардың көмегімен санның бөлшек бөлігінде қанша таңба болатынын анықтай аламыз:

```
1  number =Decimal("0.10")
2  number =3*number
3  print(number)
```

Сандарды дөңгелектеу

Decimal нысандарында сандарды дөңгелектеуге мүмкіндік беретін quantize () әдісі бар. Decimal нысаны да осы әдіске бірінші дәлел ретінде беріледі, ол санның дөңгелектеу форматын көрсетеді:

```
1  fromdecimal importDecimal
2
3  number =Decimal("0.444")
4  number =number.quantize(Decimal("1.00"))
5  print(number)      # 0.44
6
7  number =Decimal("0.555678")
8  print(number.quantize(Decimal("1.00")))      # 0.56
9
10 number =Decimal("0.999")
11 print(number.quantize(Decimal("1.00")))      # 1.00
```

Пайдаланылған "1.00" жолы дөңгелектеу бөлшек бөлігінде екі таңбаға дейін жүретінін көрсетеді.

ROUND_HALF_UP: егер одан кейін 5 немесе одан жоғары сан болса, санды көбею жағына қарай дөңгелектейді.

ROUND_HALF_DOWN: егер одан кейін 5-тен көп сан болса, санды көбею жағына қарай дөңгелектейді.

```
   number =Decimal("10.026")
1  print(number.quantize(Decimal("1.00"), ROUND_HALF_DOWN))      #
2  10.03
3
4  number =Decimal("10.025")
5  print(number.quantize(Decimal("1.00"), ROUND_HALF_DOWN))      #
   10.02
```

ROUND_05 UP: егер 5 болса, тек 0-ге дейін дөңгелектенеді.

```
1  number =Decimal("10.005")
2  print(number.quantize(Decimal("1.00"), ROUND_05UP))      # 10.01
3
4  number =Decimal("10.025")
5  print(number.quantize(Decimal("1.00"), ROUND_05UP))      # 10.02
```

1.8 Объектіге бағытталған бағдарламалау

Кластар мен обьектілер

Python объектіге бағытталған бағдарламалау парадигмасын қолдайды, яғни біз бағдарламаның компоненттерін класс ретінде анықтай аламыз.

Класс-бұл шаблон немесе объектінің ресми сипаттамасы, ал объект осы кластың данасын, оның нақты бейнесін білдіреді. Келесі аналогияны жасауға болады: бізде адам туралы біраз түсінік бар - екі қол, екі аяқ, бас, ас қорыту, жүйке жүйесі, ми және т.б. кейбір шаблондар бар - бұл үлгіні класс деп атауға болады. Нақты бар адам (іс жүзінде осы кластың данасы) осы кластың объектісі болып табылады.

Код тұрғысынан класс белгілі бір тапсырманы орындайтын функциялар мен айнымалылар жиынтығын біріктіреді. Класс функциялары әдістер деп те аталады. Олар кластың мінез-құлқын анықтайды. Сынып айнымалылары атрибуттар деп аталады-олар класс күйін сақтайды.

Класс ***class*** кілт сөзімен анықталады:

```
1 classкласс атауы:
2     класс әдістері
```

Класс объектісін құру үшін келесі синтаксис қолданылады:

```
1 Объект_атауы = класс_атауы([параметрлер])
```

Мысалы, біз адамды білдіретін қарапайым Person класын анықтайық:

```
1
2 classPerson:
3     name ="Talgat"
4
5     defdisplay_info(self):
6         print("Salem, menin atym", self.name)
7
8 person1 =Person()
9 person1.display_info()        # Salem, menin atym  Talgat
10 person2 =Person()
11 person2.name ="Samat"
12 person2.display_info()        # Salem, menin atym  Samat
```

Класс Person адамның атын сақтайтын name атрибутын және адам туралы ақпарат көрсетілетін display_info әдісін анықтайды.

Кез - келген кластың әдістерін анықтаған кезде, олардың барлығы конвенцияларға сәйкес self деп аталатын ағымдағы объектіге сілтемені бірінші параметр ретінде қабылдауы керек екенін ескеру қажет (бірқатар бағдарламалау тілдерінде аналогтың бір түрі бар-бұл*this* кілт сөзі). Класс ішіндегі осы сілтеме арқылы біз сол кластың әдістеріне немесе

атрибуттарына жүгіне аламыз. Атап айтқанда, өрнек арқылы self.name пайдаланушы атын алуға болады.

Person класын анықтағаннан кейін біз оның екі обьектісін жасаймыз - person1 және person2. Обьектінің атын қолдана отырып, біз оның әдістері мен атрибуттарына жүгіне аламыз. Бұл жағдайда біз әр обьектіге display_info () әдісін шақырамыз, ол консольдегі жолды көрсетеді, ал екінші объект үшін name атрибутын да өзгертеміз. Бұл жағдайда display_info әдісін шақырған кезде self параметрі үшін мәнді жіберудің қажеті жоқ.

Конструкторлар

Класс объектісін құру үшін конструктор қолданылады. Сонымен, жоғарыда біз Person класының обьектілерін жасаған кезде, біз барлық кластарға ие үнсіз конструкторды қолдандық:

```
1  person1 =Person()
2  person2 =Person()
```

Алайда, біз кластарда конструкторды __init () деп аталатын арнайы әдісті қолдана отырып анықтай аламыз. Мысалы, біз конструкторды қосу арқылы person класын өзгертеміз:

```
1   classPerson:
2
3       # конструктор
4       def__init__(self, name):
5           self.name =name  # атауын тағайындаймыз
6
7       defdisplay_info(self):
8           print("Salem, menin atym ", self.name)
9
10
11  person1 =Person("Talgat")
12  person1.display_info()        # Salem, menin atym Talgat
13  person2 =Person("Samat")
14  person2.display_info()        # Salem, menin atym Samat
```

Бірінші параметр ретінде конструктор ағымдағы нысанға - self сілтемесін алады. Көбінесе конструкторларда класс атрибуттары орнатылады. Сонымен, бұл жағдайда атрибут үшін орнатылған пайдаланушы аты конструкторға екінші параметр ретінде жіберіледі self.name. сонымен қатар, атрибут үшін Person класының алдыңғы нұсқасындағыдай класта *name* айнымалысын анықтау қажет емес. Мәнді орнату *self.name = name* name атрибутын жасайды.

```
person1 =Person("Talgat")
person2 =Person("Samat")
```

Нәтижесінде біз келесі консольды шығаруды аламыз:

```
Salem, menin atym Talgat
Salem, menin atym  Samat
```

Обьектімен жұмыс аяқталғаннан кейін біз оны жадыдан жою үшін del операторын қолдана аламыз:

```
person1 =Person("Talgat")
delperson1     # жадыдан жою
# person1.display_info()  # person1 жадыдан жойылғандықтан бұл әдіс жұмыс істемейді
```

Айта кету керек, бұл іс жүзінде қажет емес, өйткені сценарий аяқталғаннан кейін барлық обьектілер автоматты түрде жадыдан жойылады.

Инкапсуляция

Үнсіздік бойынша, кластардағы атрибуттар жалпыға қол жетімді, яғни бағдарламаның кез келген жерінен біз объектінің атрибутын алып, оны өзгерте аламыз.

Мысалы:

```
classPerson:
    def__init__(self, name):
        self.name =name    # есімін тағайындаймыз
        self.age =1        # жасын тағайындаймыз

    defdisplay_info(self):
        print("Имя:", self.name, "\tВозраст:", self.age)


talgat =Person("Talgat")
talgat.name ="Rahat"       #  name атрибутын өзгертеміз
talgat.age = -40             #  age атрибутын өзгертеміз
talgat.display_info()          # Есімі: Rahat     Возраст: -40
```

Бірақ бұл жағдайда, мысалы, адамның жасына немесе атына дұрыс емес мән бере аламыз, мысалы, теріс жасты көрсете аламыз. Мұндай мінез-

құлық жағымсыз, сондықтан объектінің атрибуттарына қол жеткізуді бақылау туралы мәселе туындайды.

Инкапсуляция ұғымы осы проблемамен тығыз байланысты.

Инкапсуляция - объектіге бағытталған бағдарламалаудың негізгі тұжырымдамасы. Бұл шақырушы кодтан объект атрибуттарына тікелей қол жеткізуге жол бермейді.

Python бағдарламалау тілінде инкапсуляцияға қатысты класс атрибуттарын жеке немесе жабық етіп жасыруға болады және оларға *қасиеттер* деп аталатын арнайы әдістер арқылы қол жеткізуді шектеуге болады.

Жоғарыдағы анықталған кластарды олардың қасиеттерін анықтап өзгертеміз:

```
1  classPerson:
2      def__init__(self, name):
3          self.__name =name      # есімін тағайындаймыз
4          self.__age =1          # жасын тағайындаймыз
5
6      defset_age(self, age):
7          ifage inrange(1, 100):
8              self.__age =age
9          else:
10             print("Мүмкін болмайтын жас")
11
12     defget_age(self):
13         returnself.__age
14
15     defget_name(self):
16         returnself.__name
17
18     defdisplay_info(self):
19         print("Esimi:", self.__name, "\tZhas:", self.__age)
20
21 talgat =Person("Talgat")
22
23 talgat.display_info()         # Esimi: Talgat Возраст: 1
24 talgat.set_age(-3486)         # Недопустимый возраст
25 talgat.set_age(25)
26 talgat.display_info()         # Esimi: Talgat  Возраст: 25
```

Жеке атрибут жасау үшін оның атауының басында қос сызық қойылады: self.__name. Біз мұндай атрибутқа тек сол кластан жүгіне аламыз. Бірақ біз бұл кластан тыс жерде бола алмаймыз. Мысалы, осы атрибутқа мән беру ештеңе бермейді:

```
1  tom.__age =43
```

Себебі бұл жағдайда динамикалық түрде жаңа атрибут __age анықталады, бірақ бұл self.__age. ешқандай қатысы жоқ.
Оның мәнін алуға тырысу орындалу қатесіне әкеледі:

```
1 print(tom.__age)
```

Дегенмен, бізге пайдаланушының жасын орнату қажет болуы мүмкін. Ол үшін қасиеттер жасалады. Бір қасиетті қолдана отырып, атрибут мәнін алуға болады:

```
1 defget_age(self):
2     returnself.__age
```

Бұл әдіс көбінесе геттер немесе аксессор деп аталады.
Жасты өзгерту үшін басқа қасиет анықталады:

```
1 defset_age(self, value):
2     ifvalue inrange(1, 100):
3         self.__age =value
4     else:
5         print("Мүмкін емес жас")
```

Мұнда біз жасымызды қайта белгілеу керек пе, жоқ па, жағдайға байланысты шеше аламыз. Бұл әдіс setter немесе mutator (mutator) деп те аталады. Әрбір жеке атрибут үшін ұқсас қасиеттер жұбын құру қажет емес. Сонымен, жоғарыдағы мысалда біз адамның атын тек дизайнерден орната аламыз. Ал алу үшін get_name әдісі анықталған.

Аннотациялар қасиеттері

Жоғарыда біз қасиеттерді қалай құру керектігін қарастырдық. Бірақ Python-да қасиеттерді анықтаудың тағы бір танымал тәсілі бар. Бұл әдіс @белгісімен алдын-ала аннотацияларды қолдануды қамтиды. қасиет-геттерді құру үшін @property аннотациясы қасиеттің үстіне қойылады. Қасиет-сеттерді құру үшін, қасиеттің үстіне есімі_қасиеті_геттера.setter. аннотациясы орнатылады.

Біз Person класын аннотацияларды қолдана отырып қайта жазамыз:

```
classPerson:
    def__init__(self, name):
        self.__name =name  # есімін тағайындаймыз
        self.__age =1      # жасын тағайындаймыз

    @property
    defage(self):
        returnself.__age

    @age.setter
    defage(self, age):
        ifage inrange(1, 100):
            self.__age =age
        else:
            print("Мүмкін емес жас")

    @property
    defname(self):
        returnself.__name

    defdisplay_info(self):
        print("Esimi:", self.__name, "\tZhas:", self.__age)


talgat =Person("Talgat")

talgat.display_info()      # Esimi: Talgat Возраст: 1
talgat.age =-3486          # Мүмкін емес жас
print(talgat.age)          # 1
talgat.age =36
talgat.display_info()      # Esimi: Talgat Возраст: 36
```

Біріншіден, қасиет-сеттерқасиет-геттерден кейін анықталатынына назар аударған жөн.

Екіншіден, сеттер де, геттер де бірдей аталады - age. Геттер age деп аталатындықтан, @age.setter аннотациясы сеттердің үстіне орнатылады.

Осыдан кейін, геттерге де, сеттерге де біз talgat.age. өрнегі арқылы жүгінеміз.

Мұрагерлік

Мұрагерлік сізге бұрыннан бар класс негізінде жаңа класс құруға мүмкіндік береді. Инкапсуляциямен қатар мұрагерлік объектіге бағытталған дизайнның негіздерінің бірі болып табылады.

Мұрагерліктің негізгі ұғымдары-субкласс және суперкласс. Субкласс барлық мемлекеттік атрибуттар мен әдістерді суперкласссқа мұра етеді. Супер класс сонымен қатар негізгі (базалық класс) немесе ата - ана (ата-ана класы) деп аталады, ал ішкі класс туынды (derived class) немесе бала класы деп аталады.

Класты мұрагерлеуге арналған синтаксис келесідей:

```
1  classішкі класс(суперкласс):
2      ішкі класс әдістері
```

Полиморфизм

Полиморфизм объектіге бағытталған бағдарламалаудың тағы бір негізгі аспектісі болып табылады және негізгі кластан мұраға қалған функционалдылықты өзгерту мүмкіндігін ұсынады.

Object класы. Обьектінің жолдық бейнеленуі

Python - ның 3-ші нұсқасынан бастап, барлық кластарда бір жалпы суперкласс бар-object және барлық кластар үнсіздікі бойынша оның әдістерін мұра етеді.

Object класының ең көп қолданылатын әдістерінің бірі - __str__() әдісі. Обьектінің жолдық бейнеленуін алу немесе обьектіні жол ретінде көрсету қажет болған кезде, Python бұл әдісті шақырады. Класты анықтаған кезде бұл әдісті қайта анықтау жақсы тәжірибе болып саналады.

Мысалы, Person класын алып, оның жолдық бейнеленуін шығарайық:

```
classPerson:
    def__init__(self, name, age):
        self.__name =name  # есімін тағайындаймыз
        self.__age =age  # жасын тағайындаймыз

    @property
    defname(self):
        returnself.__name

    @property
    defage(self):
        returnself.__age

    @age.setter
    defage(self, age):
        ifage inrange(1, 100):
            self.__age =age
        else:
            print("Мүмкін емес жас")

    defdisplay_info(self):
        print("Esimi:", self.__name, "\tZhas:", self.__age)

talgat =Person("Talgat", 23)
print(talgat)
```

Бағдарламаны іске қосу кезінде келесідей нәрсені көрсетеді:

```
<__main__.Person object at 0x0000017D2BEBDCF8>
```

Енді person класында __str__әдісін анықтаймыз:

```
classPerson:
    def__init__(self, name, age):
        self.__name =name  # есімін тағайындаймыз
        self.__age =age  # жасын тағайындаймыз

    @property
    defname(self):
        returnself.__name

    @property
    defage(self):
        returnself.__age

    @age.setter
    defage(self, age):
        ifage inrange(1, 100):
            self.__age =age
        else:
            print("Мүмкін емес жас")

    defdisplay_info(self):
        print(self.__str__())

    def__str__(self):
        return"Esimi: {} \tZhas: {}".format(self.__name, self.__age)

talgat =Person("Talgat", 23)
print(talgat)
```

_ Str _ () әдісі жолды қайтаруы керек. Бұл жағдайда біз адам туралы негізгі ақпаратты қайтарамыз. Енді консольдық шығару басқаша болады:

```
Esimi: Talgat    Zhas: 23
```

1.9 Графиктік интерфейс жасау

Tkinter. Бағдарлама терезесін жасау

Бүгінгі таңда көптеген бағдарламалар консольге қарағанда интуитивті және ыңғайлы графикалық интерфейсті қолданады. Python бағдарламалау тілін қолдана отырып, графикалық бағдарламалар жасауға болады. Ол үшін Python-да үнсіз келісім бойынша арнайы *тулкит* қолданылады - tkinter деп аталатын компоненттер жиынтығы.

Тулкит Tkinter барлық қажетті графикалық компоненттерді - түймелерді, мәтіндік өрістерді және т. б. қамтитын жеке кірістірілген модуль түрінде қол жетімді.

Графикалық бағдарламаларды құрудың негізгі кезеңі-терезе құру. Содан кейін графикалық интерфейстің барлық басқа компоненттері терезеге қосылады. Сондықтан алдымен қарапайым терезе жасаймыз. Ол үшін келесі сценарийді анықтайық:

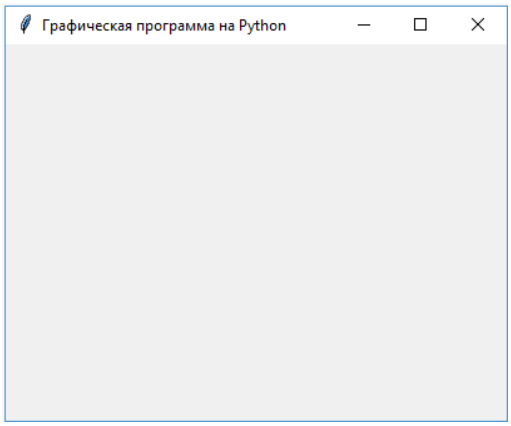
```
1 fromtkinter import*
2
3 root =Tk()
4 root.title("Графическая программа на Python")
5 root.geometry("400x300")
6
7 root.mainloop()
```

Графикалық терезені жасау үшін tkinter модулінде анықталған TK () конструкторы қолданылады. Жасалған терезе түбірлік айнымалыға тағайындалады және осы айнымалы арқылы біз терезе атрибуттарын басқара аламыз. Атап айтқанда, title () әдісін қолдана отырып, терезенің тақырыбын орнатуға болады.

Geometry () әдісін қолдана отырып терезенің өлшемін аламыз. Geometry () әдісіне өлшемді орнату үшін "ені х биіктігі"форматында жол беріледі. Егер бағдарлама терезесін құру кезінде geometry() әдісі шақырылмаса, онда терезе ішкі мазмұнды орналастыру үшін қажет кеңістікті алады.

Терезені көрсету үшін оған mainloop () әдісін шақыру керек, ол пайдаланушының өзара әрекеттесуі үшін терезе оқиғаларын өңдеу циклын бастайды.

Нәтижесінде, сценарийді іске қосқан кезде біз осындай бос терезені көреміз (Сурет 1.14):



Сурет 1.14 – Графиктік терезе

Терезенің бастапқы орны

Үнсіз келісімбойынша, терезе экранның жоғарғы сол жақ бұрышына орналастырылған. Бірақ біз geometry() әдісіне қажетті мәндерді беру арқылы оның орнын өзгерте аламыз:

```
1  fromtkinter import*
2
3  root =Tk()
4  root.title("Графическая программа на Python")
5  root.geometry("400x300+300+250")
6
7  root.mainloop()
```

Енді geometry әдісіндегі жол келесі форматқа ие: "ені X биіктігі + координатасы X + координатасы". Яғни, іске қосу кезінде терезе экранның жоғарғы сол жақ бұрышынан 300 пиксель оңға және 250 пиксель төмен болады.

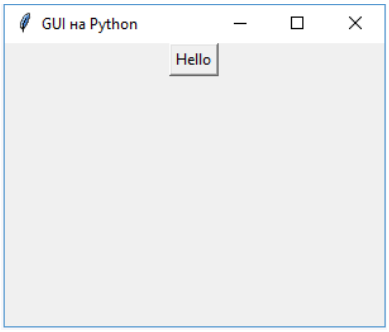
Батырмалар

Тулкит Tkinter құрамында компоненттер немесе виджеттер жиынтығы бар, олардың бірі - батырма. Терезеге батырманы қосайық:

```
1   fromtkinter import*
2
3   root =Tk()
4   root.title("GUI на Python")
5   root.geometry("300x250")
6
7   btn =Button(text="Hello")
8   btn.pack()
9
10  root.mainloop()
```

Батырманы жасау үшін Button () конструкторы қолданылады. Бұл конструкторда text параметрін пайдаланып, батырманың мәтінін орнатуға болады. Элементті көрінетін ету үшін pack () әдісі шақырылады.

Нәтижесінде терезенің жоғарғы жағында батырма пайда болады (Сурет 1.15):

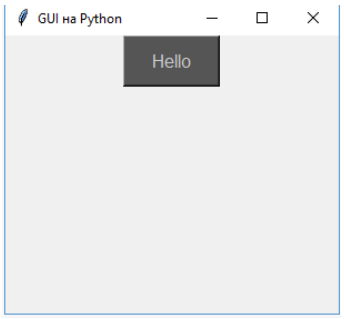


Сурет 1.15 – Батырма жасау

Әр виджет, оның ішінде батырма, оның визуализациясына әсер ететін және конструктор арқылы баптай алатын бірқатар атрибуттарға ие:

```
1   fromtkinter import*
2
3   root =Tk()
4   root.title("GUI на Python")
5   root.geometry("300x250")
6
7   btn =Button(text="Hello",          # батырмамәтіні
8               background="#555",     # батырманың фондық түсі
9               foreground="#ccc",     # мәтін түсі
10              padx="20",             # шекарадан мазмұнға көлденең шегініс
11              pady="8",              # шекарадан мазмұнға тігінен шегіну
12              font="16"              # шрифт биіктігі
13              )
14  btn.pack()
15   root.mainloop()
```

Pady, pax, font параметрлері сандық мәнді қабылдайды, ал background және foreground параметрлері он алтылық түс мәнін алады. Font параметрінде қаріп анықтамасы бар. 1.16 – суретте батырманың безендірілуі көрсетілген.



Сурет 1.16 – Батырма безендірілуі

Жалпы, Button конструкторы келесі параметрлерді қабылдай алады:

```
1  Button (master, options)
```

Master параметрі ата-ана контейнеріне сілтеме жасайды. Жоғарыдағы жағдайда бұл графикалық терезенің өзі болуы мүмкін және біз оны былай жаза аламыз:

```
1  root =Tk()
2  root.title("GUI на Python")
3  root.geometry("300x250")
4
5  btn =Button(root, text="Hello")
6  btn.pack()
```

Алайда, егер кодта бір терезе жасалса, онда батырма және кез-келген басқа элемент осы терезеде үнсіз келісім бойынша орналастырылады. Сондықтан бірінші параметрді жоғарыдағы мысалдардағыдай елемеуге болады. Егер кодта бірнеше терезе жасалса, онда біз Button конструкторына сілтемені қажетті терезеге жібере аламыз.

Батырманы басуды өңдеу

Батырманы басуды өңдеу үшін конструкторға command параметрін орнатып, оны басқан кезде іске қосылатын функцияға сілтеме беру керек:

```
1    fromtkinter import*
2
3    clicks =0
4
5
6    defclick_button():
7        globalclicks
8        clicks +=1
9        root.title("Clicks {}".format(clicks))
10
11   root =Tk()
12   root.title("GUI на Python")
13   root.geometry("300x250")
14
15   btn =Button(text="Click Me", background="#555", foreground="#ccc",
16               padx="20", pady="8", font="16", command=click_button)
17   btn.pack()
18
19   root.mainloop()
```

Мұнда басу өңдегіші ретінде click_button функциясы орнатылады. Бұл функцияда нұқу санын сақтайтын ғаламдық clicks айнымалысы өзгереді және оның мәні терезе тақырыбына шығады. Осылайша, түймені басқан сайын click_button функциясы іске қосылады және шертулер саны артады.

II ТАРАУ. JAVA БАҒДАРЛАМАЛАУ ТІЛІ

2.1. Java бағдарламалау тіліне кіріспе

Java дегеніміз не?

Java - жалпы мақсаттағы бағдарламалау тілі. Яғни, белгілі бір салада нақты мамандандырусыз, әртүрлі бағдарламалық өнімдерді жасауда қолданылатын тіл. Ол көптеген жолдармен Python, JavaScript және басқа тілдерге Java деңгейінде ұқсас. Сонымен қатар, Java C және C ++ тілдерінен синтаксис тонна алады.

Бұл - объектіге бағытталған тіл. Барлық құрылым объектілердің, кластардың, даналардың және басқа ресми нысандардың айналасында құрастырылған, олар бағдарламалық жасақтамада ООР дамыту стандарты ретінде қабылданған. Бұл әр түрлі типтегі ғимараттарды салуға болатын бөлек құрылыс блоктарын қолданатын үйлерді жобалау сияқты. Java-да жазылған бағдарламалар осындай «блоктардан» тұрады, бұл өңдеу уақытын қысқартады, сонымен қатар кодты оқуды жеңілдетеді және өзгертуді жеңілдетеді.

Java платформа рөлін де атқарады. Осы тілде жазылған код JVM виртуалды машинасында жұмыс істейді және сәйкес виртуалды машинаны қолдайтын кез-келген жүйеде проблемасыз басталады.

Java тарихы

Java тілін Sun Microsystems инженерлер тобы 1995 жылы жасаған. Компанияны Java-мен бірге кейінірек Oracle Corporation сатып алды.

Java-ны дамытудағы негізгі мәселе компьютерлерде ғана жұмыс істей алмайтын бағдарламалау тілін құру болды. Яғни, бұл тіпті тоңазытқыштың операциялық жүйесінде де жұмыс істеуге жеткілікті болды. Даму кезеңінде бұл шешім тым көреген болып көрінді, бірақ ол өзімен бірге Java үшін маңызды болып табылатын және тілді танымал еткен бірнеше маңызды архитектуралық өзгерістер әкелді.

Мысалы, тіл сізге бір рет код жазуға және кез-келген бағдарламалық жасақтама үшін өзгертусіз компиляция жасауға мүмкіндік берді. Бұл дамуды айтарлықтай жеңілдетті және, ең бастысы, кодты жазуға кететін адам-сағат санын азайтты.

Рас, Java-дың танымалдығы бұл функциямен емес, веб-парақтарға арналған мини-қосымшаларды құру мүмкіндігімен келді. Бұрын, Java болмаса, көптеген сайттар немесе олардың функциялары қол жетімді болмады, ал бәрі жоспарланған түрде жұмыс жасау үшін әзірлеушілер JRE утилитасын жүктеп алуға мәжбүр болды.

Java және JavaScript

Жаңадан жасаушылар мен қарапайым адамдар кейде бұл тілдер туыс сияқты әсер қалдырады, бірақ олар онымен байланысты емес. Олардың атауы мен С-ға негізделген синтаксисте ортақ 4 әріп бар.

JavaScript-ті 90-жылдардың ортасында Netscape әзірледі және бастапқыда LiveScript деп аталды. Тіл танымалдылықты таба алмады, өйткені ол кезде барлық назар тез дамып келе жатқан Java-ға аударылды. Сондықтан, Netscape, кем дегенде, біреу олардың туындыларына қызығушылық таныту үшін ребрендинг жасауға шешім қабылдады. Таң қаларлықтай, ол жұмыс істеді.

Қазір бұл ондаған қуатты құрылымы бар вебтің негіздерінің бірі. Айтпақшы, синтаксистегі ұқсастық дамытушыларға бір тілден екінші тілге тез ауысу мүмкіндігін береді. Егер сіз Java білсеңіз, JavaScript-ті тезірек және керісінше үйреніңіз.

Java-да жалпы кез келген бағдарламалар жазуға болады. Java-дің жақсы жағы - бұл жалпыға ортақ тіл болып табылады - негізгі конфигурацияда ол бағдарламалық платформалардың бүкіл арсеналын жасауға жарамды. Интернетке арналған виджет керек пе? Java жақсы. Windows, Linux және macOS үшін әмбебап бағдарлама жасау керек пе? Проблема емес. Android-ті дамыту да мүмкін.

Java-ны қолданудың көптеген сценарийлері бар. Тіл шынымен әмбебап және кез-келген бағдарламалық жасақтама үшін кез-келген бағдарламалық жасақтаманы жасауға жарамды. Сондықтан Java-да қандай бағдарламалар жазуға болады деген сұраққа ең жақсы жауап - өзіңіз құрғыңыз келетін бағдарламалар.

Java және Android

Тарихи тұрғыдан Java Android дамуындағы флагмандық тілдердің біріне айналды. Google мобильді операциялық жүйесінде орнатылған виртуалды машиналар Java кодын инициализациялауға мүмкіндік береді. Осы себепті, Android үшін жасалған бағдарламалық жасақтаманың керемет мөлшері Sun Microsystems тілінде жазылған.

Қазір Android қосымшаларын құруға арналған басқа тілдер бар, бірақ Java әлі күнге дейін ең танымал тілдердің бірі болып табылады.

Java-да жазылған ең жақсы бағдарламалардың мысалы:

✓ 2004 жылы НАСА инженерлері Maestro ғылыми іс-әрекеттің жоспарлаушысын Java-да жазды, Spirit планетасы қызыл планетада жүргенде оны басқарды.

✓ 20 жылдан бері ғарыш мамандары JavaFX терең ғарыштық траекториясын зерттеушіні Жерден тыс шарлау үшін қолданады.

✓ Іздеу, ең танымал веб-энциклопедияға (Википедия) енген, бастапқыда Java-да жазылған, кейін оның орнына Java-ға негізделген қозғалтқыш Elasticsearch келді.

✓ Ең танымал ойындардың бірі - Minecraft - 2009 жылы Марк Персон құрған және Java-да жазылған. Ойынға әр түрлі өзгертулер мен толықтырулар бір тілде жазылған.

✓ IntelliJ IDEA - бұл Java-ға негізделген жан-жақты дамыған орта.

Мұндай мысалдар өте көп. Java қосымшаларын ғарышкерлер, инженерлер, медицина мамандары, жүйелік әкімшілер және т.б. қолдануда.

Java-ның оң жақтары

✓ Үйрену оңай. Синтаксистің көп бөлігі C ++ тілінен алынған, бірақ жетілдірілген түрінде. Java жасаушылары барлық қайшылықтарды жойды. Нәтижесінде C ++ пайда болады, ол өте күшті және жан-жақты ғана емес, сонымен қатар ыңғайлы.

✓ Java - тұрақты және сенімді тіл. Оның объектіге бағытталған табиғаты бағдарламашылар тудыратын дамудың қателіктерін болдырмайды.

✓ Қауіпсіздік те соңғы орында емес. Java-ді құру кезінде Sun Microsystems сарапшылары тілді ғаламтор арқылы байланысатын мобильді қосымшалар жасау үшін қолдануды ойластырған. Сондықтан, олар жобалау кезеңінде Java-ны мүмкіндігінше қауіпсіз етуге ұмтылды.

✓ Ең бастысы - таңдалған платформадан толық тәуелсіздік. Жоғарыда айтқанымдай, Java кез-келген операциялық жүйеге арналған.

Java-ның минустары

✓ Java, өзінің жанкүйерлерінің үлкен қауымдастығына және ессіз танымалдығына қарамастан, жаңадан бастаушылар үшін қиынырақ.

✓ Java салыстырмалы түрде күрделі мәселелерді шешу үшін құрылды. Сондықтан оны шағын сценарийлер жазу үшін пайдалану ұсынылмайды. Бұл кішкентай саяжайдағы бақша төсегін өндірістік комбайнмен жырту сияқты. Сіз жасай аласыз, бірақ неге?

✓ Барлық кодтар виртуалды машиналарда жұмыс істейтіндіктен, Java-ны оңтайландыру қиынырақ. Сондықтан, Java қосымшаларының өнімділігі кейде айтарлықтай нашарлауы мүмкін. Атап айтқанда, бұл Android дамуының алғашқы кезеңінде байқалды. Содан кейін ол мәңгілік мұздатқыш және өте баяу өнім ретінде беделге ие болды. Бұған кінәлі Java болды.

Ұқсас тілдер

Java-ның жақын туыстарынан танымал C # бағдарламалау тілін ажыратуға болады. Олардың жақын болғаны соншалық, кейбір мектептер Java бағдарламашыларын бітіреді, кейінірек C # әзірлеушісі ретінде жұмысқа орналасады. Әңгіме тек осындай курстардағы сабақ беру тәсілінде ғана емес, техникалық ұқсастықтарда. Бұл таңқаларлық емес, өйткені Майкрософт тілі Java-ны көзбен құрды.

Сондықтан C # Java-ны алғаш құрған тапсырмаларды орындау үшін қолданылады. Жалпы, сіз оларды бір-бірін алмастыратын деп санауға болады.

Java және Python

Python-тың басты артықшылығы - оның қарапайым синтаксисі. Шынында да, қарапайым сценарий жазу немесе жаңа идеяны Python-да

байқау оңайырақ. Істерді аяқтау үшін сізге барлық бағдарламаны жазып, жинақтаудың қажеті жоқ.

Бұл кодтың кез-келген объектісімен жұмыс кезінде көрінеді. Python мен Java-дағы бірдей кластар басқаша көрінеді. Соңғысында олар Python ұсынғаннан гөрі айтарлықтай ауқымды және түсіну қиын.

Python-тан айырмашылығы, Java күшті теруді қолданады, бұл әзірлеушілерді тәртіпті болуға мәжбүр етеді және әрқашан алдын-ала қолданылатын деректер түрін жариялайды. Әйтпесе, компилятор қатемен жауап береді. Python бұл туралы талғамайды.

Python сонымен қатар кодтаудың көп нұсқаларын қолдайды, сондықтан көптеген жасаушылар оқуды жеңілдетеді.

Java-ға қарсы C ++

Синтаксис тұрғысынан осы тілдердің ұқсастығына қарамастан, Java мен C ++ тілдерінде бірқатар айырмашылықтар бар. Мысалы, C ++ компиляторды ғана қолданады. Яғни, барлық кодтарды компьютер тікелей оқитын объект құрылымына айналдыратын механизм. Java компилятордан басқа әр жолды оқитын аудармашыны қолданады және онда сипатталған нұсқауларды бірден орындайды.

C ++ операторлардың шамадан тыс жүктелуін және әдістің шамадан тыс жүктелуін, сондай-ақ struct және біріктіру сияқты түрлерін қолдайды. Жоғарыда айтылғандардың ішінен Java тек әдіс жүктеуді қолдайды.

C ++ көбінесе секвенсорлар немесе аналогтық аппараттық эмуляторлар сияқты музыкалық бағдарламалық жасақтаманы жасау үшін қолданылады. Сонымен қатар, C ++ кодын Windows және macOS амалдық жүйелерінің компоненттерінен табуға болады.

Oracle JDK және OpenJDK

Java бағдарламалау тілінде өңдеу үшін бізге JDK немесе Java Development Kit деп аталатын арнайы құралдар жиынтығы қажет. Дегенмен, JDK-дің әртүрлі енгізілімдері бар, бірақ олардың барлығы бірдей тілді - Java-ны қолданады. Екі ең танымал қосымшалар - Oracle JDK және OpenJDK. Олардың арасындағы айырмашылық неде?

Oracle JDK толығымен Oracle әзірлеген. OpenJDK Oracle және басқа да бірқатар компаниялармен бірлесіп әзірленуде.

Ең үлкен айырмашылықтары - лицензиялау. Лицензияға сәйкес Oracle JDK жеке пайдалануға, сондай-ақ қосымшаларды әзірлеу, тестілеу және көрсету үшін ақысыз пайдаланылуы мүмкін. Басқа жағдайда (мысалы, қолдау алу үшін) жазылым түрінде коммерциялық лицензия қажет. Ал OpenJDK мүлдем тегін.

Функционалдылық тұрғысынан Oracle JDK және OpenJDK мүмкіндіктерінің жиынтығы іс жүзінде ерекшеленбеуі керек. Бірақ өнімділік жоспарында Oracle JDK OpenJDK-тен сәл жылдамырақ екені атап өтілген. Сонымен қатар, кейбір әзірлеушілер OpenJDK-ға қарағанда Oracle JDK-нің тұрақты екенін атап өтеді.

Біз бұл оқулық үшін Oracle JDK-ді қолданамыз, бірақ егер сіз OpenJDK қолдансаңыз, сізге ешқандай қиындықтар туындамауы керек.

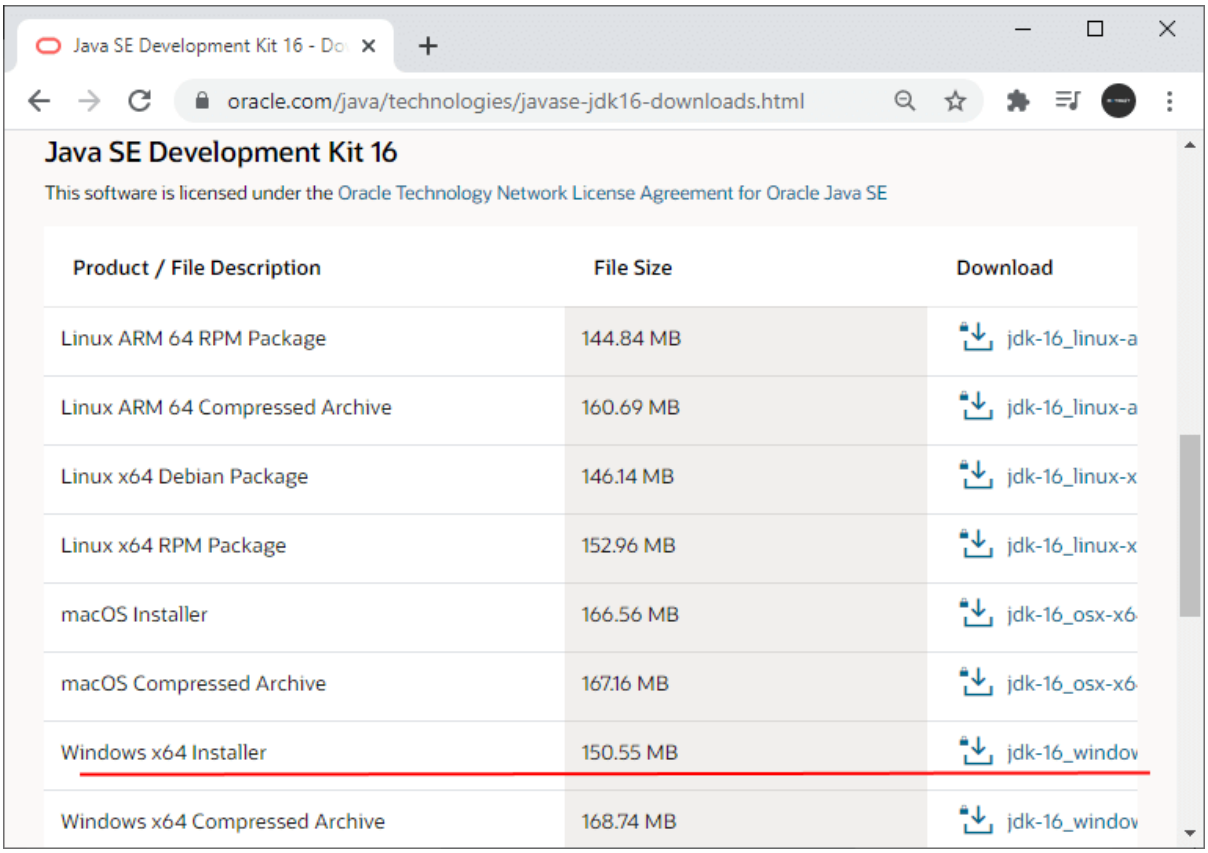
Java-ны орнату

Сонымен, Java-да бағдарлама жазу үшін бізге арнайы JDK (Java Development Kit) әзірлеу жиынтығы қажет. JDK құрамына Java бағдарламаларын құрастыруға, іске қосуға және басқа да әртүрлі функцияларды орындауға мүмкіндік беретін бірқатар бағдарламалар мен утилиталар кіреді.

JDK нұсқасын Oracle ресми веб-сайтынан жүктеуге және орнатуға болады: https://www.oracle.com/java/technologies/javase-downloads.html

Сонымен, ең соңғы - 16-шы нұсқаны https://www.oracle.com/java/technologies/javase-jdk16-downloads.html мекен-жайы бойынша жүктеу парағына өтейік.

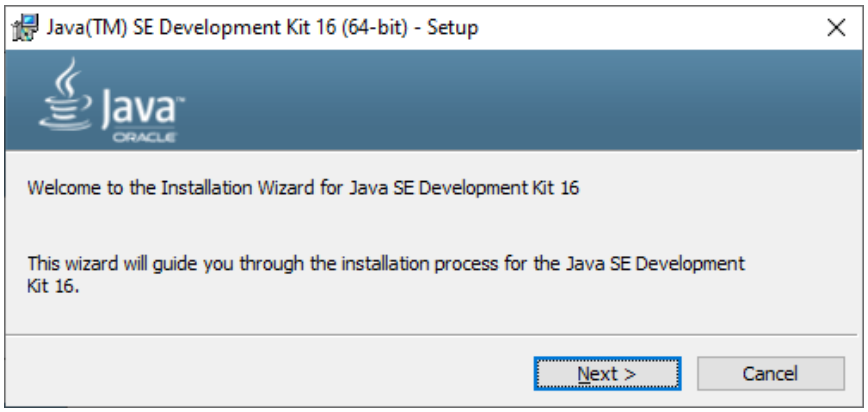
Бұл бетте біз операциялық жүйеге (Windows, MacOS немесе Linux) арналған тарату жинағын табамыз және жүктейміз (Сурет 2.1):



Сурет 2.1- Операциялық жүйелерге жіктелуі

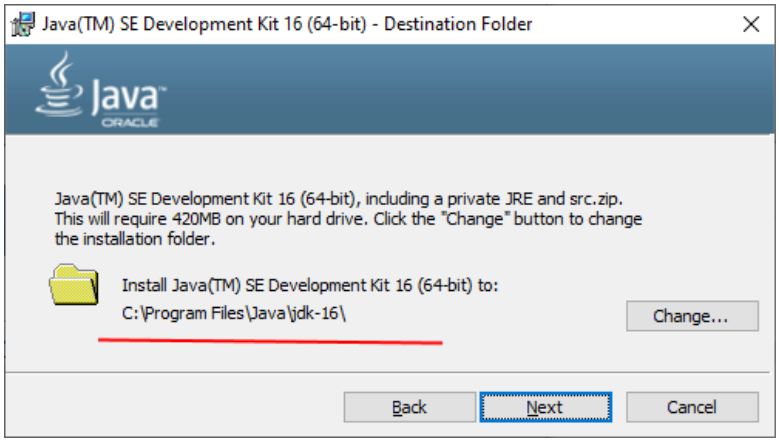
Көптеген операциялық жүйелер үшін жүктеудің екі нұсқасы бар: орнатушы ретінде немесе орнатудың қажеті жоқ мұрағат ретінде. Мысалы, менің ОЖ - Windows, сондықтан орнатушыны білдіретін jdk_16_windows-x64_bin.exe файлын жүктеймін.

Жүктеуден кейін орнатушыны іске қосыңыз (Сурет 2.2):



Сурет 2.2- орнатушыны іске қосу

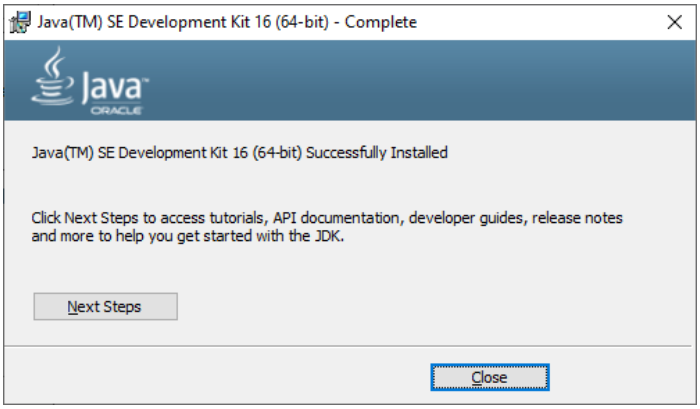
Келесі (Next) батырмасын басыңыз. Келесі экранда орнату қай бумада орындалатындығын көрсетуіңіз керек (Сурет 2.3):



Сурет 2.3- Орнату бумасы

Орнатуды аяқтау үшін үнсіздік бойынша бума таңдауынан шығып, Келесі (Next)түймесін басыңыз.

JDK орнату аяқталғаннан кейін біз келесі терезені көреміз (Сурет 2.4):



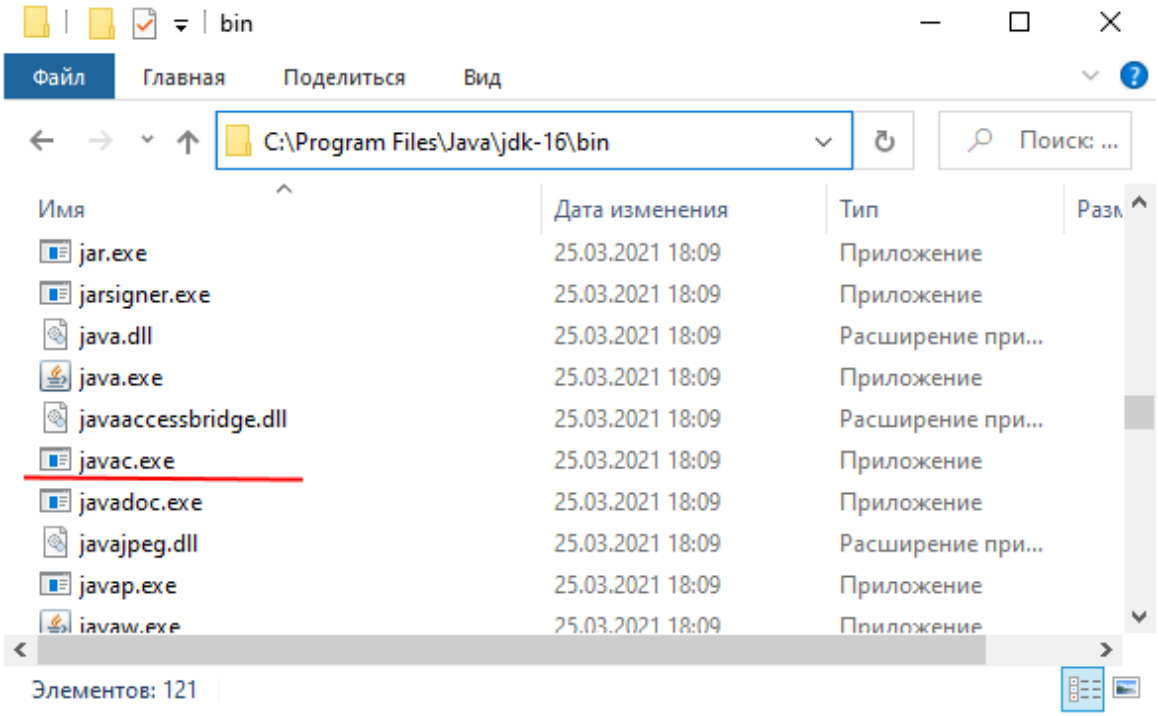
Сурет 2.4- JDK орнату

Сонымен, JDK орнатқаннан кейін бірінші Java бағдарламасын құрайық.

Java-дағы алғашқы бағдарлама

Сонымен, JDK орнатқаннан кейін алғашқы Java қосымшасын құрайық. Java-да бағдарлама құру үшін не қажет? Бізге ең алдымен бағдарлама кодын жазу керек, ал ол үшін мәтіндік редактор керек. Сіз Notepad ++ сияқты кез-келген мәтіндік редакторды қолдана аласыз.

Бағдарлама кодын орындалатын қосымшаға айналдыру үшін компилятор қажет. JDK орнатқаннан кейін барлық файлдар үнсіздік бойынша C: \ Program Files \ Java \ jdk- [version_number] каталогына орналастырылады (Windows қолданған кезде). Біздің жағдайда бұл C: \ Program Files \ Java \ jdk-16 каталогы. Егер онда bin ішкі каталогын ашсақ, онда бірқатар утилиталарды көруге болады. Бізді ең алдымен javac компиляторының утилитасы қызықтырады. Бағдарлама класын құрастыру үшін оның кодын осы компиляторға беруіміз керек.



Сурет 2.5 - Жинақталған бағдарламаны іске қосу

Бұл бумадағы назар аударатынтағы бір утилита - java.exe, ол жинақталған бағдарламаны іске қосуға мүмкіндік береді. (Сурет 2.5)

Сонымен, қатты дискіде Java тілінде бастапқы коды бар файлдар орналасқан каталог құрайық. Айталық, бұл C: / Java каталогы болсын. Содан кейін біз осы каталогта мәтіндік файл жасаймыз, оның атын

Program.java деп өзгертеміз. Осы файлды кез-келген мәтіндік редактордан ашып, оған келесі бағдарламаны терейік:

```
1   publicclassProgram{
2
3       publicstaticvoidmain (String args[]){
4
5           System.out.println("Hello Java!");
6       }
7   }
```

Java - бұл объектіге бағытталған тіл, сондықтан барлық бағдарлама өзара әрекеттесетін кластардың жиынтығы ретінде ұсынылған. Бұл жағдайда Program деген бір класс анықталады.

Класс анықтаған кезде алдымен **public** жалпыға қол жетімділік модификаторы келеді, ол бұл кластың барлығына қол жетімді болатындығын көрсетеді, яғни біз оны командалық жолдан іске қоса аламыз. Одан кейін **class** кілттік сөзі, содан кейін класс аты келеді. Яғни класс **Program** деп аталады. Атынан кейін класс мазмұны фигуралы жақшаларда орналасқан.

Класс әр түрлі айнымалылар мен әдістерді қамтуы мүмкін. Бұл жағдайда біз бір mainәдісін жарияладық. Бұл кез-келген Java бағдарламасындағы негізгі әдіс, ол бағдарламаның кіру нүктесі және барлық басқару осымен басталады. Ол бағдарламада міндетті түрде болуы керек.

Сонымен қатар main әдісі public модификаторына ие. static сөзі main әдісі статикалық екенін, ал voidсөзі оның ешқандай мән бермейтінін білдіреді. Мұның бәрі нені білдіретінін кейінірек қарастырамыз.

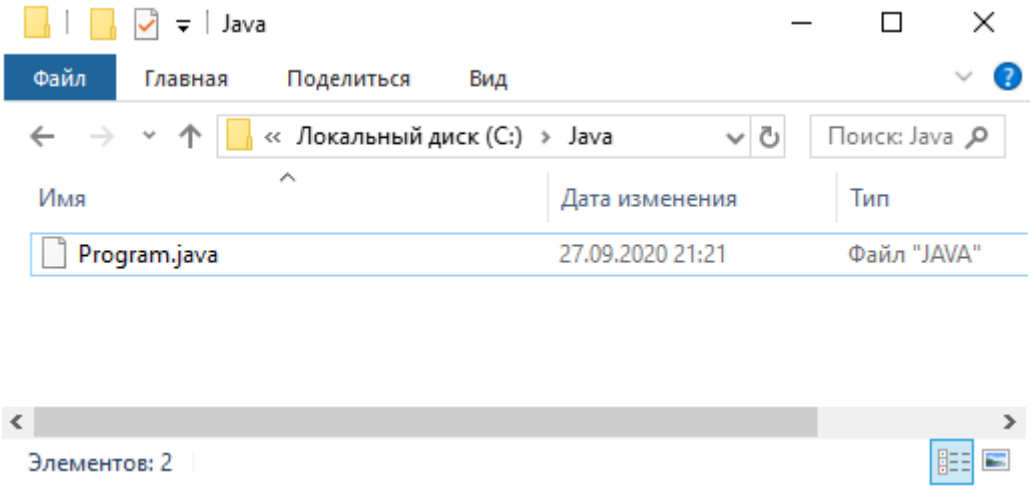
Әрі қарай, жақшада String args [] әдісі параметрлері бар - бұл String типінің мәндерін, яғни жолдарды сақтайтын argsмассиві. Бұл жағдайда бізге бұл әлі қажет емес, бірақ нақты бағдарламада бұлар бағдарлама командалық жолдан басталған кезде берілетін жол параметрлері болып табылады.

Фигуралы жақшалардағы параметрлер тізімінен кейін әдістің денесі бар - бұл әдіс орындайтын нұсқаулар. Бұл жағдайда анықтамада нақты бір ғана нұсқаулық анықталады - белгілі бір жолдың консоліне шығу. Консольға шығару үшін кірістірілген System.out.println () әдісі қолданылады. Шығару жолы осы әдіске беріледі. Әр мәлімдеме нүктелі үтірмен аяқталады.

Енді жазылған бағдарламаны құрастырайық. Командалық жолды (Windows-та) немесе Linux / MacOS-та терминал ашып, тиісті командаларды енгізейік. Алдымен, бағдарламамен бірге файл орналасқан каталогқа өтейік (Сурет 2.6):

```
cd C:\Java
```

Бұл жағдайда файл C: \ Java каталогында орналасқан.



Сурет 2.6 – Каталогта орналасу

Содан кейін команданы пайдаланып бағдарламаны құрастырамыз:

```
C:\Java>"C:\Program Files\Java\jdk-16\bin\javac" Program.java
```

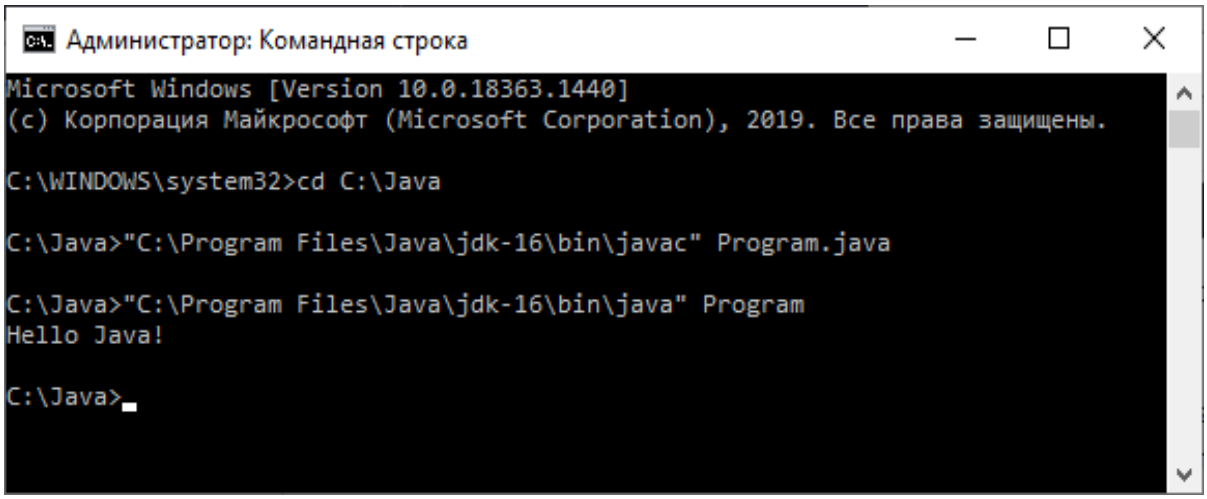
Назар аударыңыз, javac компиляторына дейінгі барлық жол тырнақшаға алынады, содан кейін бос орынмен бөлінген бағдарлама класын қамтитын файлдың аты.

Содан кейін бағдарлама байт-кодқа жинақталады және жаңа Program.class файлын C: \ Java каталогынан табуға болады. Бұл байт коды бар файл болады. Енді біз java утилитасын пайдаланып іске қосуымыз керек:

```
C:\Java>"C:\Program Files\Java\jdk-16\bin\java" Program
```

Мұнда файл кеңейтімін қолдануға болмайды.

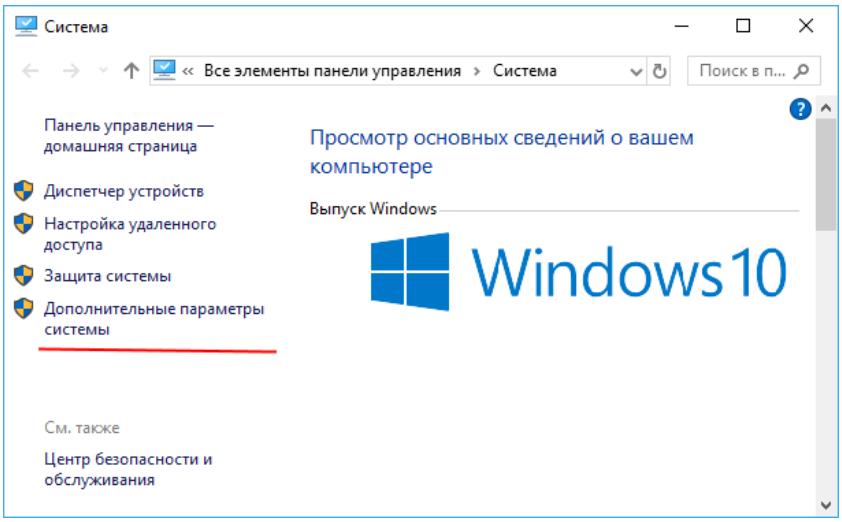
Windows ОЖ үшін барлық процесс келесідей болады (Сурет 2.7):



Сурет 2.7 – Командалық жол

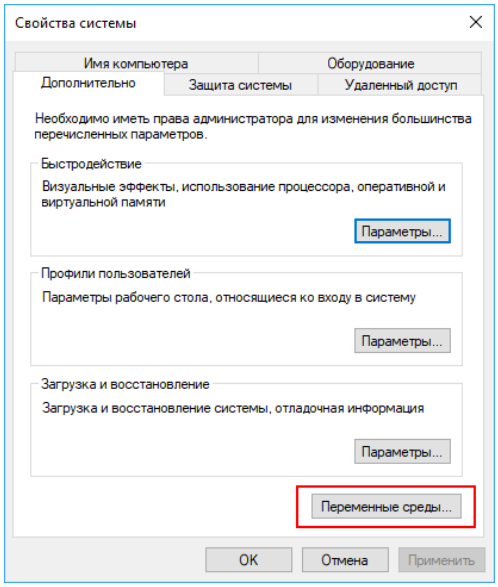
Path айнымалысына java қосу

Компиляция үшін javac компиляторына толық жолды енгізу керек, оны теру қателерімен байланыстыруға болады, сонымен қатар толық жолға әр уақытта кіру ыңғайсыз. Болашақта жағдайды жеңілдету үшін айнымалылар ортасында PATH айнымалысына JDK жолын қосайық. Егер біз Windows-та жұмыс істейтін болсақ, онда айнымалылар ортасын қосу үшін басқару тақтасын ашып (оны іздеу арқылы ашуға болады), Система пунктіне өту керек (Сурет 2.8):



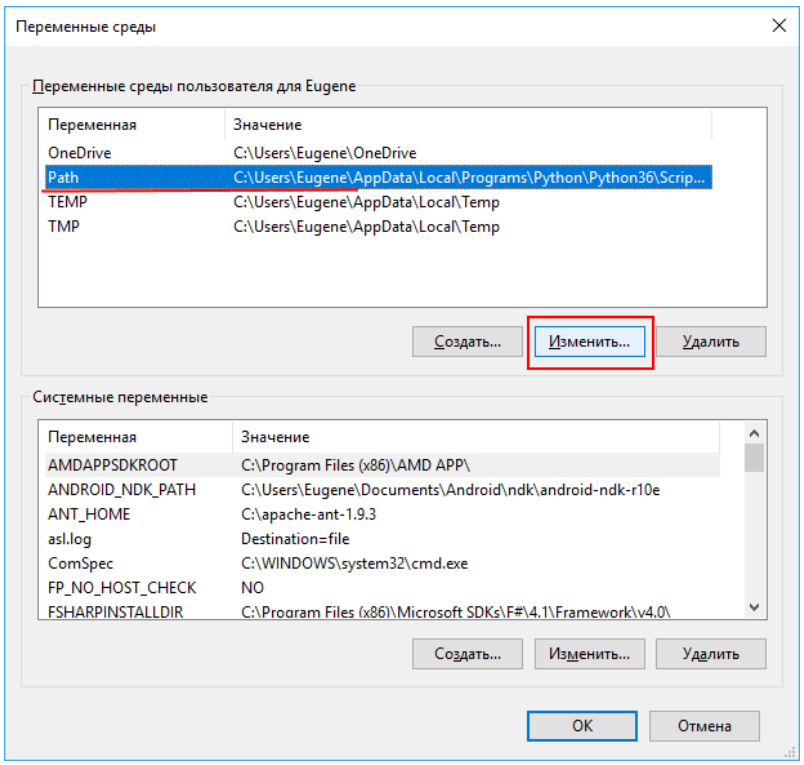
Сурет 2.8 – Жүйеге өту

Мұнда «Қосымша жүйелік параметрлер» тармағын таңдайық. Содан кейін ашылған терезеде Айнымалы орталар (Переменные среды) түймешігін басыңыз.(Сурет 2.9)



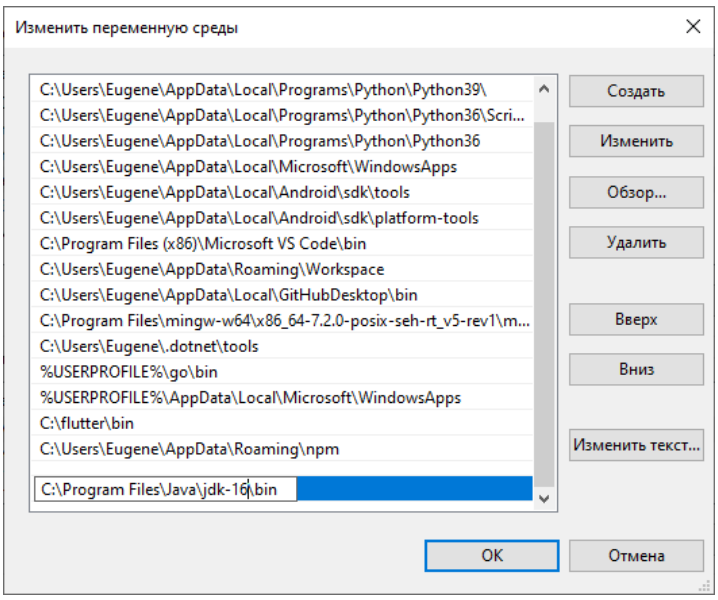
Сурет 2.9 – Жүйе қасиеттері

Әрі қарай бізге барлық айнымалыларды көруге болатын терезе ашылады (Сурет 2.10) Ағымдағы пайдаланушы үшін Path айнымалысын таңдап, «Change» батырмасын басыңыз:



Сурет 2.10 – Айнымалылар ортасы

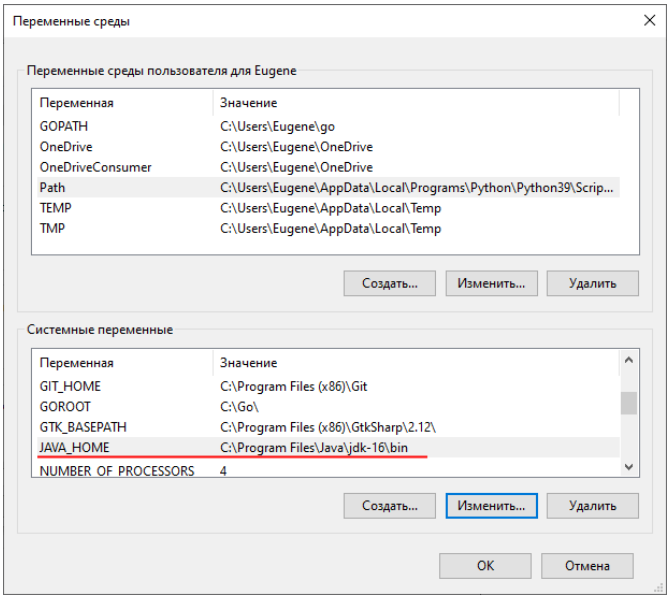
Содан кейін Path айнымалысына қосылған барлық жолдармен терезе шығады(Сурет 2.11):



Сурет 2.11 - Path айнымалысына қосылған барлық жолдар

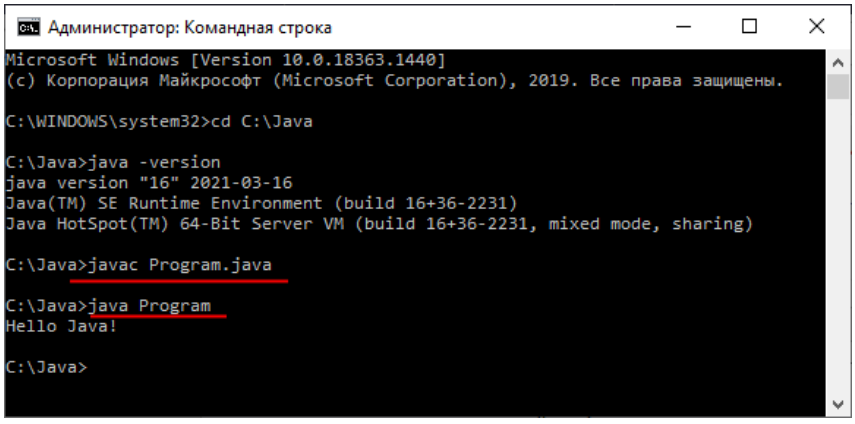
Бұл терезеде «Создать» батырмасын басыңыз. Осыдан кейін жаңа жол қосылады, онда біз jdk каталогына binбумасына жол енгіземіз. Сонымен, біздің жағдайда ол C: \ Program Files \ Java \ jdk-16 \ bin, сондықтан берілген жолды қосамыз. Содан кейін OK түймесін басамыз.

Сонымен қатар, егер Java-ны бөлек немесе басқа бағдарламалар үшін қажетті компонент ретінде орнату керек болса, онда айнымалылар ортасында жүйенің «JAVA_HOME» айнымалысын құру өте сирек емес екенін ескеру керек, ол ескі Java нұсқасына мегзейді. (Сурет 2.12) Егер сізде осындай айнымалы болса, оның жаңа sdk java нұсқасын көрсетіп тұрғанын тексеріңіз.



Сурет 2.12 - JAVA_HOME айнымалысы

Path айнымалысын орнатқаннан кейін командалық жолды қайта басыңыз. Осыдан кейін сіз утилиталардың аттарын толық жолсыз ғана енгізе аласыз (сурет 2.12):



Сурет 2.13 – утилит аттары

Сонымен қатар, java-ның дұрыс нұсқасын қолданып жатқандығымызды тексеру үшін командалық консольге енгізе аламыз:

```
java -version
```

2.2. Java бағдарламалау тілінің негіздері

Бағдарламаның құрылымы

Java тіліндегі бағдарламаның негізгі құрылыс материалы - бұл нұсқаулар(statement). Әрбір нұсқау кейбір әрекеттерді орындайды, мысалы шақыру әдістері, айнымалыларды жариялау және оларға мәндер беру. Java-да нұсқау аяқталғаннан кейін нүктелі үтір(;) қойылады. Бұл белгі компиляторға команданың соңын көрсетеді. Мысалы:

```
1System.out.println("Hello Java!");
```

Бұл жол консольге «Hello Java!» жолын басып шығаратын System.out.println әдісіне шақыруды білдіреді. Бұл жағдайда әдіс шақыру нұсқаулық болып табылады, сондықтан нүктелі үтірмен аяқталады.

Жеке нұсқаулардан басқа, кең таралған конструкциякод блогы болып табылады. Код блогы нұсқаулар жиынтығын қамтиды, ол фигуралы жақшаға алынады және нұсқаулық ашылатын және жабылатын фигуралы жақшалардың арасында орналасады:

```
1{
2System.out.println("Hello!");
3 System.out.println("Welcome to Java!");
4}
```

Бұл код блогы консольге белгілі бір жолды басып шығаратын екі операторлан тұрады.

Бағдарламаның орындалуы. Main әдісі

Java - бұл объектіге бағытталған тіл, сондықтан бүкіл бағдарламаны өзара әрекеттесетін кластар мен объектілердің жиынтығы ретінде қарастыруға болады. Бірінші тарауда алғашқы қосымшаны құру кезінде бағдарлама келесідей анықталды:

```
1public class Program{
2
3public static void main (String args[]){
4
5System.out.println("Hello Java!");
6}
7}
```

Яғни, біздің бағдарламаның негізі –Program класы.Класты анықтаған кезде алдымен public жалпыға қол жетімділік модификаторы келеді, ол бұл кластың барлығына қол жетімді болатындығын көрсетеді, яғни біз оны командалық жолдан іске қоса аламыз. Одан кейін classкілт сөзі, содан кейін класс аты келеді. Кластың атауынан кейін класс мазмұны орналасқан код блогы болады.

Java бағдарламасына кіру нүктесі Program класында анықталған main әдісі болып табылады. Онымен бірге бағдарлама басталады. Ол бағдарламада болуы керек. Оның атауы тек келесідей болуы мүмкін:

1public static void main (String args[])

Қосымшаны іске қосқан кезде, Java виртуалды машинасы бағдарламаның негізгі сыныбында ұқсас тақырыбы бар mainәдісін іздейді және ол табылған кезде оны іске қосады.

Әдістің тақырыбының алдында publicмодификатор болады, бұл әдіс сырттан қол жетімді болатындығын көрсетеді. staticсөзimain әдісі статикалық екенін, ал void сөзі оның ешқандай мән бермейтінін білдіреді. Әрі қарай, жақшада String args [] әдісі параметрлері бар - бұл String типінің мәндерін, яғни жолдарды сақтайтын argsмассиві. Бағдарламаны осы массив арқылы іске қосқанда біз бағдарламаға әр түрлі мәліметтерді жібере аламыз.

Әдістің тақырыбы орындалатын нұсқаулар жиынтығынан тұратын оның блогымен жалғасады.

Түсініктемелер (Comments)

Бағдарлама кодында түсініктемелер болуы мүмкін. Пікірлер бағдарламаның мағынасын түсінуге мүмкіндік береді, оның белгілі бір бөліктері не істейді. Компиляция кезінде түсініктемелер еленбейді және қосымшаның жұмысына және оның мөлшеріне әсер етпейді.

Java-да пікірлердің екі түрі бар: бір жолды және көп жолды. Бір жолды Түсініктеме Қос сызықтан кейін бір жолға орналастырылады//. Ал көп жолды Түсініктеме /* түсініктеме мәтіні * / таңбалары арасында жатыр. Ол бірнеше жолға орналастырылуы мүмкін. Мысалы:

```
1.  /*
2.  Көпжолды түсініктеме
3.  Программа кодын қамтитын
4.  жаңа класты хабарлау
5.  */
6.  publicclassProgram{   // Program класын хабарлау басы
7.
8.  // main әдісін анықтау
9.  publicstaticvoidmain (String args[]){    // жаңа әдісті хабарлау
10.
11. System.out.println("Hello Java!");     // консольға жолды шығару
12. }   // жаңа әдісті хабарлау соңы
```

```
13.   } // Program класын хабарлау соңы
```

Айнымалылар мен тұрақтылар

Айнымалылар бағдарламада деректерді сақтауға арналған. Айнымалы белгілі бір типтегі мәнді сақтайтын жадтың белгілі бір аймағын білдіреді. Әр айнымалының түрі, аты және мағынасы бар. Түр айнымалы немесе рұқсат етілген мәндер ауқымы қандай ақпаратты сақтай алатындығын анықтайды.

Айнымалылар келесідей жарияланады:

Мәлімет типі айнымалы аты;

Мысалы, x деп аталатын және int типіне ие болатын айнымалыны анықтаймыз:

```
intx;
```

Бұл өрнекте біз int типті x айнымалысын жариялаймыз. Яғни, x белгілі бір санды 4 Байттан аспайды.

Айнымалы атау келесі талаптарды қанағаттандыратын кез-келген ерікті атау бола алады:

- ✓ атау кез-келген алфавиттік-сандық таңбаларды, сондай-ақ астын сызу белгісін қамтуы мүмкін, ал атаудағы бірінші таңба Сан болмауы керек;
- ✓ атауында пунктуациялық белгілер мен пробел болмауы тиіс;
- ✓ атау Java тілінің кілт сөзі бола алмайды.

Сонымен қатар, жариялау және кейінгі пайдалану кезінде Java - регистрге тәуелді тіл екенін ескеру қажет, сондықтан келесі хабарландырулар int num; және int NUM; екі түрлі айнымалы болады.

Айнымалыны жариялай отырып, біз оған мән бере аламыз:

```
intx;      // айнымалыны хабарлау
x = 10;    // мән беру
System.out.println(x);  // 10
```

Сондай-ақ, ол жарияланған кезде айнымалы мәнді тағайындауға болады. Бұл процесс инициализация деп аталады:

```
intx = 10;// айнымалыны хабарлау және инициализациялау
System.out.println(x);  // 10
```

Егер біз оны қолданар алдында айнымалыға мән бермесек, онда қате пайда болуы мүмкін, мысалы, келесі жағдайда:

```
intx;
System.out.println(x);
```

Үтір арқылы бір типтегі бірнеше айнымалыны бірден жариялауға болады:

```
intx, y;
x = 10;
y = 25;
System.out.println(x);  // 10
System.out.println(y);  // 25
```

Сіз оларды бірден бастай аласыз:

```
1 intx = 8, y = 15;
2 System.out.println(x);  // 8
3 System.out.println(y);  // 15
```

Айнымалылардың ерекшелігі-біз бағдарлама барысында олардың мәнін өзгерте аламыз:

```
1 intx = 10;
2 System.out.println(x);  // 10
3 x = 25;
4 System.out.println(x);  // 25
```

var кілт сөзі

Java 10-дан бастап тілге var кілт сөзі қосылды, ол айнымалы мәнді анықтауға мүмкіндік береді:

```
var x = 10;
System.out.println(x);  // 10
```

Var сөзі деректер түрінің орнына қойылады, алайнымалы тип өзі тағайындалған мәннен алынады. Мысалы, x айнымалысына 10 саны тағайындалады, яғни айнымалы int түрін білдіреді.

Бірақ егер айнымалы var арқылы жарияланса, онда біз оны міндетті түрде инициализациялауымыз керек, яғни оны бастапқы мәнмен қамтамасыз етуіміз керек, әйтпесе қате пайда болады, мысалы, келесі жағдайда:

```
var x;      // ! Қате, айнымалы бапталмаған
x = 10;
```

Тұрақтылар

Айнымалылардан басқа, Java-да тұрақты мәндерді деректерді сақтау үшін пайдалануға болады. Айнымалы тұрақтылардан айырмашылығы, мәнді тек бір рет тағайындауға болады. Тұрақты айнымалы сияқты жарияланады, тек алдымен final кілт сөзі келеді:

```
1
2 finalintLIMIT = 5;
```

3 System.out.println(LIMIT); // 5
// LIMIT=57; // сондықтанбізендіжазаалмаймыз, өйткені LIMIT тұрақты

Әдетте, тұрақтыларда жоғарғы регистрде атаулар болады.

Тұрақтылар енді өзгермеуі керек айнымалыларды орнатуға мүмкіндік береді. Мысалы, егер бізде рі санын сақтайтын айнымалы болса, онда біз оны тұрақты деп жариялай аламыз, өйткені оның мәні тұрақты.

Деректер типтері

Java-ның басты ерекшеліктерінің бірі-бұл тіл қатаң терілген. Бұл дегеніміз, әр айнымалы мен тұрақты белгілі бір түрді білдіреді және бұл тип қатаң түрде анықталады. Деректер түрі айнымалы немесе тұрақты сақталатын мәндер ауқымын анықтайды.

Сонымен, Java-да айнымалыларды құру үшін қолданылатын ендірілген мәліметтер типтерінің жүйесін қарастырыңыз. Және ол келесі түрлермен ұсынылған.

- **boolean:** true немесе false мәнін сақтайды

booleanisActive = false;
booleanisAlive = true;

- **byte:**бүтінсанды -128-ден 127-гедейін сақтайды және 1 байттыалады

bytea = 3;
byteb = 8;

- **short**: бүтін санды -32768-ден 32767-ге дейін сақтайды және 2 байтты алады

shorta = 3;
shortb = 8;

- **int**: бүтін санды -2147483648-ден 2147483647-ге дейін сақтайды және 4 байтты алады

inta = 4;
intb = 9;

- **long**: бүтін санды -9 223 372 036 854 775 808-ден 9 223 372 036 854 775 807-ге дейін сақтайды және 8 байтты алады

longa = 5;
longb = 10;

- **double**: өзгермелі нүкте санын ±4.9*10-324-тен ±1.8 * 10308-ге дейін сақтайды және 8 байтты алады

doublex = 8.5;
doubley = 2.7;

* Бөлшек және бөлшек бөлгіш ретінде бөлшек литералдарда нүкте қолданылады.

- **float**: өзгермелі нүкте санын -3.4*1038-ден 3.4*1038-ге дейін сақтайды және 4 байтты алады.

1 floatx = 8.5F;

```
2  floaty = 2.7F;
```

- **char:** UTF-16 кодтауында бір таңбаны сақтайды және 2 байттыалады, сондықтан сақталатын мәндер диапазоны 0-ден 65535-кедейін.

Бұл жағдайда айнымалы тек оның түріне сәйкес келетін мәндерді қабылдай алады. Егер айнымалы бүтінсанды тип болса, онда ол бөлшек санды сақтай алмайды.

Бүтін сандар

Барлық бүтін әріптер, мысалы, 10, 4, -5 сандары int типті мәндер ретінде қабылданады, алайда біз бүтін сандарды басқа бүтін түрлерге тағайындай аламыз: byte, long, short. Бұл жағдайда Java автоматты түрде тиісті түрлендірулерді жүзеге асырады:

```
1 bytea = 1;
2 shortb = 2;
3 longc = 2121;
```

Алайда, егер біз *long* типті айнымалыға *int* типі үшін рұқсат етілген мәндерден асатын өте үлкен санды бергіміз келсе, онда құрастыру кезінде қате пайда болады:

```
1 longnum = 2147483649;
```

Мұнда 2147483649 саны long түрі үшін жарамды, бірақ int түрі үшін шекті мәндерден асып түседі. Барлық бүтін сандар әдепкі бойынша int сияқты мәндер ретінде қарастырылатындықтан, компилятор бізге қатені көрсетеді. Мәселені шешу үшін санға l немесе L жұрнағын қосу керек, ол санның long түрін білдіретінін көрсетеді:

```
1 longnum = 2147483649L;
```

Әдетте, бүтін сандар үшін мәндер Ондық сандар жүйесінде орнатылады, бірақ біз басқа сандық жүйелерді де қолдана аламыз.

Мысалы:

```
1 intnum111 = 0x6F; // 16-лық жүйе, 111 саны
2 intnum8 = 010; // 8-дік жүйе, 8 саны
3 intnum13 = 0b1101; // 2-лік жүйе, 13 саны
```

0x таңбаларынан кейін он алтылық мәнді орнату үшін сан он алтылық форматта көрсетіледі. Осылайша, сегіздік мән 0 таңбасынан кейін, ал екілік мән 0B таңбаларынан кейін көрсетіледі.

Сондай-ақ, бүтін сандар астын сызу белгісін қолдана отырып, санның разрядтарын бөлуді қолдайды:

```
1 intx = 123_456;
2 inty = 234_567__789;
3 System.out.println(x);  // 123456
```

```
4 System.out.println(y);  // 234567789
```

Өзгермелі нүкте сандары

Өзгермелі нүктелік бөлшек литералдың өзгермелі түрін тағайындау кезінде, мысалы, 3.1, 4.5 және т.б., Java бұл әріптерді Қос типті мән ретінде автоматты түрде қарастырады. Бұл мән float ретінде қарастырылуы керек екенін көрсету үшін біз f жұрнағын қолдануымыз керек:

```
1 floatfl = 30.6f;
2 doubledb = 30.6;
```

Бұл жағдайда екі айнымалы да бірдей мәнге ие болғанымен, бұл мәндер басқаша қарастырылады және жадта әр түрлі орын алады.

Символдар мен жолдар

Мән ретінде символдық типтегі айнымалы бір тырнақшаға салынған бір таңбаны алады: char ch = 'e';. Сонымен қатар, символдық типтегі айнымалыға 0-ден 65535-ке дейінгі бүтін санды тағайындауға болады. Бұл жағдайда айнымалы қайтадан таңбаны сақтайды, ал бүтін сан Unicode (UTF-16) таңбалар кестесіндегі таңба нөмірін көрсетеді. Мысалы:

```
1 charch=102; // символ 'f'
2 System.out.println(ch);
```

Символдық айнымалыларды белгілеудің тағы бір формасы-оналтылық форма: айнымалы "\u"таңбаларынан кейін пайда болатын оналтылық формадағы мәнді алады. Мысалы, char ch= '\u0066'; тағы да 'f' таңбасын сақтайды.

Символдық айнымалыларды жол айнымалыларымен шатастырмау керек, 'a' "a" - мен бірдей емес. Жол айнымалылары String нысанын білдіреді, Char немесе int-тен айырмашылығы ол Java-да примитивті тип емес:

```
1 String hello = "Hello...";
2 System.out.println(hello);
```

Әріптерді, сандарды, тыныс белгілерін, басқа таңбаларды білдіретін таңбалардан басқа, басқару тізбегі деп аталатын арнайы таңбалар жиынтығы бар. Мысалы, ең танымал дәйектілік - "\n". Ол келесі жолға ауысады.

Мысалы:

```
1 String text = "Hello \nworld";
2 System.out.println(text);
```

Осы кодты орындау нәтижесі:

```
Hello
world
```

Бұл жағдайда \ N тізбегі келесі жолға аудару керек деген сигнал болады.

15 нұсқасынан бастап Java мәтіндік блоктарды қолдайды (text blocks) - үш тырнақшаға салынған көп жолды мәтін. Олардың практикалық пайдасы неде екенін қарастырыңыз. Мысалы, бермеуден бастасақ және оған үлкен көпжолды мәтін:

```
1 String text = "Вот мысль, которой весь я предан,\n"+
2               "Итог всего, что ум скопил.\n"+
3               "Лишь тот, кем бой за жизнь изведан,\n"+
4               "Жизнь и свободу заслужил.";
5 System.out.println(text);
```

Операциясын қолдана отырып, біз бір мәтінге екіншісін қоса аламыз, ал мәтіннің жалғасы келесі жолда орналасуы мүмкін. Мәтінді шығару кезінде келесі жолға ауысу үшін \ N тізбегі қолданылады.

Осы кодты орындау нәтижесі:

```
Вот мысль, которой весь я предан,
Итог всего, что ум скопил.
Лишь тот, кем бой за жизнь изведан,
Жизнь и свободу заслужил.
```

JDK15-те пайда болған мәтіндік блоктар Көп жолды мәтінді жазуды жеңілдетеді:

```
1 String text = """
2               Вот мысль, которой весь я предан,
3               Итог всего, что ум скопил.
4               Лишь тот, кем бой за жизнь изведан,
5               Жизнь и свободу заслужил.
6               """;
7 System.out.println(text);
```

Бүкіл мәтіндік блок үш тырнақшаға оралған, оларды беру үшін жолдар қосылымын немесе \N тізбегін пайдаланудың қажеті жоқ. Бағдарламаны орындау нәтижесі жоғарыдағы мысалдағыдай болады.

Java-да консольді енгізу / шығару

Пайдаланушының өзара әрекеттесуінің ең оңай жолы-консоль: біз консольге кейбір ақпаратты шығара аламыз немесе керісінше консольден кейбір деректерді оқи аламыз. Java-да консольмен өзара әрекеттесу үшін System класы қолданылады және оның функционалдығы консольді енгізу мен шығаруды қамтамасыз етеді.

Консольге шығару

Шығару ағынын System сыныбына құру үшін out нысаны анықталды. Бұл нысанда println әдісі анықталған, ол консольге белгілі бір мәнді шығаруға мүмкіндік береді, содан кейін консоль курсорын келесі жолға аударады. Мысалы:

```
1 publicclassProgram {
2
3    publicstaticvoidmain(String[] args) {
4
5        System.out.println("Hello world!");
6        System.out.println("Bye world...");
7    }
8 }
```

Кез-келген мән println әдісіне, әдетте, консольге шығарылуы керек жолға беріледі. Бұл жағдайда біз келесі қорытынды аламыз:

```
Hello world!
Bye world...
```

Қажет болса, жүгіргіні келесі жолға аудармауға болады. Бұл жағдайда println-ге ұқсас **System.out.print()** әдісін қолдануға болады, тек ол келесі жолға аудармайды.

```
publicclassProgram {

    publicstaticvoidmain(String[] args) {

        System.out.print("Hello world!");
        System.out.print("Bye world...");
    }
}
```

Осы бағдарламаның шығыс консолі:

```
Hello world!Bye world...
```

System.out.print әдісін қолдана отырып, сіз каретканы келесі жолға қайтара аласыз. Ол үшін \ n қашу ретін қолданыңыз:

```
1 System.out.print("Hello world \n");
```

Жолдағы кейбір деректерді жиі ауыстыру қажет. Мысалы, бізде екі сан бар және олардың мәндерін экранға шығарғымыз келеді. Бұл жағдайда біз, мысалы, былай жаза аламыз:

```
1 publicclassProgram {
2
3    publicstaticvoidmain(String[] args) {
4
5        intx=5;
6        inty=6;
7        System.out.println("x="+ x + "; y="+ y);
8    }
9 }
```

Бағдарламаның шығыс консолі:

```
x=5; y=6
```

Сонымен қатар Java-да C тілінен мұраға қалған пішімделген шығуға арналған функция бар: System.out.printf (). Оның көмегімен біз алдыңғы мысалды келесідей жаза аламыз:

```
intx=5;
inty=6;
System.out.printf("x=%d; y=%d \n", x, y);
```

Бұл жағдайда % d белгілері аргументтің біреуі мен алмастырылатын спецификаторды білдіреді. Мұнда көптеген анықтаушылар және оларға сәйкес аргументтер болуы мүмкін. Бұл жағдайда бізде тек екі аргумент бар, сондықтан бірінші% d орнына x айнымалысының мәнін, ал екіншісінің орнына - y айнымалы мәнін ауыстырады. D әрпінің өзі бұл көрсеткіштің бүтін мәндерді шығару үшін қолданылатындығын білдіреді.

% D спецификаторынан басқа, біз басқа деректер типтері үшін бірнеше спецификаторларды қолдана аламыз:

- % x: он алтылық сандарды көрсету үшін
- % f: өзгермелі нүктелік сандарды көрсету үшін
- % e: сандарды экспоненциалды түрде көрсету үшін, мысалы 1.3e + 01
- % c: бір таңбаны басып шығару
- % s: жол мәндерін көрсетуге арналған

Мысалы:

```
1  publicclassProgram {
2
3     publicstaticvoidmain(String[] args) {
```

```
4
5         String name = "Tom";
6         intage = 30;
7         floatheight = 1.7f;
8
9  System.out.printf("Name: %s  Age: %d  Height: %.2f \n", name, age, height);
10    }
11}
```

Жылжымалы нүктелік сандарды көрсету кезінде ондық үтірден кейінгі цифрлар санын көрсете аламыз, бұл үшін% .2f бойынша көрсеткішті қолданамыз, мұндағы .2 ондық үтірден кейін екі ондық таңба болатындығын көрсетеді. Нәтижесінде біз келесі өнімді аламыз:

```
Name: Tom  Age: 30  Height: 1,70
```

Консольден енгізу

In объектісі жүйенің класында консольден кіріс алу үшін анықталады. Алайда, тікелей System.in объектісі арқылы жұмыс істеу өте ыңғайлы емес, сондықтан, әдетте, **Scanner** класы қолданылады, ол өз кезегінде System.in-ді қолданады. Мысалы, сандарды енгізуді орындайтын шағын бағдарлама жазайық:

```
1  importjava.util.Scanner;
2
3  publicclassProgram {
4
5     publicstaticvoidmain(String[] args) {
6
7         Scanner in = newScanner(System.in);
8         System.out.print("Input a number: ");
9         intnum = in.nextInt();
10
11        System.out.printf("Your number: %d \n", num);
12        in.close();
13    }
14}
```

Scanner класы java.util пакетінде болғандықтан, біз алдымен импорттау java.util.Scanner операторының көмегімен импорттаймыз.

Scanner объектісінің өзін құру үшін System.in объектісі оның конструкторына беріледі. Осыдан кейін біз кіріс мәндерін ала аламыз. Мысалы, бұл жағдайда алдымен енгізу туралы шақыруды көрсетеміз, содан кейін num айнымалысына кіріс нөмірін аламыз.

Енгізілген санды алу үшін пернетақтадан енгізілген бүтін мәнді қайтаратын in.nextInt (); әдісі қолданылады.

Бағдарламаның жұмысының мысалы:

```
Input a number: 5
```

```
Your number: 5
```

Scanner класы пайдаланушы енгізген мәндерді алуға мүмкіндік беретін бірқатар басқа әдістер бар:

next(): енгізілген жолды бірінші бос орынға дейін оқиды
nextLine (): барлық енгізілген жолды оқиды
nextInt (): енгізілген int оқылады
nextDouble (): енгізілген қосарды оқиды
nextBoolean (): логикалық мәнді оқиды
nextByte (): енгізілген байтты оқиды
nextFloat (): енгізілген флотты оқиды
nextShort (): енгізілген қысқа нөмірді оқиды

Яғни, **Scanner** класында әрбір қарабайыр типке мән енгізу әдісі анықталған.

Мысалы, адам туралы ақпарат енгізуге арналған бағдарлама құрайық:

```
1  importjava.util.Scanner;
2
3  publicclassProgram {
4
5     publicstaticvoidmain(String[] args) {
6
7   Scanner in = newScanner(System.in);
8   System.out.print("Input name: ");
9   String name = in.nextLine();
10 System.out.print("Input age: ");
11 intage = in.nextInt();
12 System.out.print("Input height: ");
13 floatheight = in.nextFloat();
14 System.out.printf("Name: %s  Age: %d  Height: %.2f \n", name, age, height);
15       in.close();
16    }
17}
```

Мұнда String, int, float типтерінің деректері дәйекті түрде енгізіліп, содан кейін барлық енгізілген мәліметтер консольге шығарылады. Бағдарламаның жұмысының мысалы:

```
Input name: Tom
Input age: 34
Input height: 1,7
Name: Tom  Age: 34  Height: 1,70
```

«1,7» саны **float** типті мәнді енгізу үшін пайдаланылатынын ескеріңіз (**double** типке де қатысты), онда бөлгіш үтір болады, ал бөлгіш нүкте

болған кезде «1,7» емес. Бұл жағдайда бәрі жүйенің ағымдағы тілдік локализациясына байланысты.

Арифметикалық амалдар

Java-дағы операциялардың көпшілігі басқа C-ге ұқсас тілдерде қолданылатынға ұқсас. Бір амалдық операциялар (бір операндада орындалады), екі операнда екілік амалдар, үш операнда үштік операциялар бар. Операнд - бұл операцияға қатысатын айнымалы немесе мән (мысалы, сан). Операциялардың барлық түрлерін қарастырайық.

Сандар арифметикалық амалдарға қатысады. Java-да екілік арифметикалық амалдар (екі операнда орындалады) және унарлы (бір операнда орындалады) бар. Екілік операцияларға мыналар жатады:

+

екі санды қосу операциясы:

```
1 int a = 10;
2 int b = 7;
3 int c = a + b;  // 17
4 int d = 4 + b;  // 11
```

-

екі санды азайту амалы:

```
1 int a = 10;
2 int b = 7;
3 int c = a - b;  // 3
4 int d = 4 - a;  // -6
```

*

екі санды көбейту амалы

```
1 int a = 10;
2 int b = 7;
3 int c = a * b;  // 70
4 int d = b * 5;  // 35
```

/

екі санды бөлу амалы:

```
1 int a = 20;
2 int b = 5;
3 int c = a / b;  // 4
4 double d = 22.5 / 4.5;  // 5.0
```

Бөлу кезінде, егер амалға екі бүтін сан қатысса, нәтиже float немесе double қос айнымалыға берілсе де, бөлудің нәтижесі бүтін санға дейін дөңгелектенетінін есте ұстаған жөн:

```
1 double k = 10 / 4;     // 2
2 System.out.println(k);
```

Нәтиже өзгермелі нүкте санын көрсету үшін операндалардың бірі өзгермелі нүкте нөмірін де көрсетуі керек:

```
1 double k = 10.0 / 4;     // 2.5
2 System.out.println(k);
```

%

қалдықты екі санды бөлуден алу:

```
1 int a = 33;
2 int b = 5;
3 int c = a % b;  // 3
4 int d = 22 % 4; // 2 (22 - 4*5 = 2)
```

Бір санда орындалатын екі унарлы арифметикалық амалдар да бар: ++ (инкремент - өсім) және - (декремент - кему). Әрбір операцияның екі түрі бар: префикс және постфикс:

++ (префиксті инкремент)

Айнымалыны бірге көбейтуді ұйғарады, мысалы z = ++ y (біріншіден, y айнымалысының мәні 1-ге арттырылады, содан кейін оның мәні z айнымалысына беріледі)

```
1 int a = 8;
2 int b = ++a;
3 System.out.println(a);  // 9
4 System.out.println(b);  // 9
```

++ (постфиксті инкремент)

Сондай-ақ айнымалының бір-біріне ұлғаюын білдіреді, мысалы z = y ++ (біріншіден, y мәні z-ге беріледі, содан кейін y мәні 1-ге арттырылады)

```
1 int a = 8;
2 int b = a++;
3 System.out.println(a);  // 9
4 System.out.println(b);  // 8
```

-- (префиксті декремент)

айнымалыны біреуіне азайту, мысалы z = - - y (біріншіден, y айнымалысының мәні 1-ге азаяды, содан кейін оның мәні z айнымалысына беріледі)

```
1 int a = 8;
2 int b = --a;
3 System.out.println(a);  // 7
4 System.out.println(b);  // 7
```

-- (постфиксті декремент)

z = y-- (алдымен y айнымалысының мәні z айнымалысына беріледі, содан кейін y айнымалысының мәні 1-ге кемітіледі)

```
1 int a = 8;
2 int b = a--;
3 System.out.println(a);  // 7
4 System.out.println(b);  // 8
```

Арифметикалық басымдылық

Кейбір операциялар басқаларынан басым болады, сондықтан алдымен орындалады. Басымдықтың төмендеу ретіндегі операциялар:

++ (инкремент - өсім), - (декремент - кему)

* (көбейту), / (бөлу),% (бөлудің қалған бөлігі)

+ (қосу), - (азайту)

Арифметикалық өрнектер жиынтығын орындау кезінде операциялардың басымдылығы ескерілуі керек:

```
1 inta = 8;
2 intb = 7;
3 intc = a + 5* ++b;
4 System.out.println(c);  // 48
```

Біріншіден, өсім ++ b операциясы орындалады, оның басымдылығы жоғары - бұл b айнымалысының мәнін жоғарылатады және нәтижесінде оны қайтарады. Содан кейін 5 * ++ b көбейту орындалады, және тек соңғы қосу + 5 * ++ b болады.

Жақшалар есептеулердің ретін қайта анықтауға мүмкіндік береді:

```
1 inta = 8;
2 intb = 7;
3 intc = (a + 5) * ++b;
4 System.out.println(c);  // 104
```

Қосу операциясының басымдылығы төмен болғанына қарамастан, көбейту көбейтілмейді, алдымен орындалады, өйткені қосу операциясы жақшаға алынған.

Операциялардың ассоциативтілігі

Басымдылықтан басқа, операциялар ассоциативтілік сияқты ұғыммен ерекшеленеді. Операциялар бірдей басымдыққа ие болған кезде, бағалау тәртібі операторлардың ассоциативтілігімен анықталады. Ассоциативтілікке байланысты операторлардың екі түрі бар:

- Солдан оңға қарай орындайтын сол жақ ассоциативті операторлар
- Оңнан солға орындайтын оң ассоциативті операторлар

Мысалы, көбейту және бөлу сияқты кейбір амалдардың басымдығы бірдей. Сонда өрнек нәтижесі қандай болады:

```
1 intx = 10/ 5* 2;
```

Бұл өрнекті (10/5) * 2 деп түсіну керек пе немесе 10 / (5 * 2) деп түсіну керек пе? Шынында да, түсіндіруге байланысты біз әртүрлі нәтижелерге қол жеткіземіз.

Барлық арифметикалық операторлар (префикстің өсуі мен кішіреюінен басқа) сол жақ ассоциативті болып табылады, яғни олар солдан оңға қарай орындалады. Сондықтан 10/5 * 2 өрнегін (10/5) * 2 деп түсіндіру керек, яғни нәтиже 4 болады.

Жылжымалы нүктелермен жұмыс

Жылжымалы нүктелік сандар қаржылық және басқа есептеулерге сәйкес келмейтінін ескеру керек, мұнда дөңгелектеу қателіктері маңызды болуы мүмкін. Мысалға:

```
1 doubled = 2.0- 1.1;
2 System.out.println(d);
```

Бұл жағдайда d айнымалысы 0,9-ға тең болмайды, өйткені бастапқыда біреу болжаған болар, бірақ 0,8999999999999999. Мұндай дәлдік қателіктері орын алады, себебі екілік жүйе өзгермелі нүктелі сандарды ұсыну үшін төменгі деңгейде қолданылады, бірақ 0,1 үшін де, басқа бөлшек мәндер үшін де екілік көрініс жоқ. Сондықтан, мұндай жағдайларда әдетте BigDecimal класы қолданылады, бұл осындай жағдайларды пысықтауға мүмкіндік береді.

Биттік операциялар

Разрядты операциялар жеке цифрларда немесе сандардың биттерінде орындалады. Бұл операцияларда операнд ретінде тек бүтін сандарды ғана пайдалануға болады.

Әр санның белгілі бір екілік көрінісі бар. Мысалы, екілік сандағы 4 саны 100, ал 5 саны 101 және т.б.

Мысалы, келесі айнымалыларды алыңыз:

```
1 byteb = 7;     // 0000 0111
2 shorts = 7;    // 0000 0000 0000 0111
```

Байт түрі сәйкесінше 8 битпен ұсынылған 1 байтты немесе 8 битті алады. Демек, b айнымалысының екілік кодтағы мәні 00000111 болады. Қысқа тип жадыда 2 байт немесе 16 битті алады, сондықтан бұл типтің саны 16 битпен ұсынылатын болады. Және бұл жағдайда екілік жүйеде s айнымалысы 0000 0000 0000 0111 мәніне ие болады.

Java-да қол қойылған сандарды жазу үшін ең маңызды битке қол қойылған екілік қосымшасы қолданылады. Егер оның мәні 0-ге тең болса, онда бұл сан оң болады, ал екілік көрінісі қол қойылмаған саннан ерекшеленбейді. Мысалы, 0000 0001 ондық жүйеде 1 болады.

Егер ең маңызды бит 1 болса, онда біз теріс санмен жұмыс істейміз. Мысалы, ондық сандағы 1111 1111 -1-ді білдіреді. Тиісінше, 1111 0011 -13-ті білдіреді.

Логикалық амалдар

Сандарға логикалық амалдар биттік операцияларды білдіреді. Бұл жағдайда сандар екілік көріністе қарастырылады, мысалы, екілік жүйеде 2 10 және екі цифрдан тұрады, 7 саны 111 және үш цифрдан тұрады.

- & (логикалық көбейту)

Көбейту биттік тәсілмен орындалады, егер екі операндта да бит мәндері 1-ге тең болса, онда амал 1-ді қайтарады, әйтпесе 0 саны қайтарылады. Мысалы:

```
1 inta1 = 2; //010
2  intb1 = 5;//101
3  System.out.println(a1&b1); // нәтиже 0
4
5 inta2 = 4; //100
6 intb2 = 5; //101
7 System.out.println(a2 & b2); // нәтиже 4
```

Бірінші жағдайда, бізде 2 және 5 деген екі сан бар, екілік таңбада 010 саны, ал 5 - 101 болады. Сандарды биттік көбейту (0 * 1, 1 * 0, 0 * 1) 000 нәтижесін береді.

Екінші жағдайда, екеудің орнына бізде 5 саны сияқты бірінші цифрда 1 болатын 4 саны болады, сондықтан мұнда операцияның нәтижесі (1 * 1, 0 * 0, 0 * 1) = 100 ондық форматтағы 4 саны болады ...

- | (логикалық қосу)

Бұл операция екілік цифрлармен де орындалады, бірақ енді егер осы биттің кем дегенде бір санында бірлік болса, бірлік қайтарылады («логикалық НЕМЕСЕ» әрекеті). Мысалға:

```
1 inta1 = 2; //010
2 intb1 = 5;//101
3 System.out.println(a1|b1); // нәтиже 7 - 111
4 inta2 = 4; //100
5 intb2 = 5;//101
6 System.out.println(a2 | b2); // нәтиже 5 – 101
```

- ^ (логикалық эксклюзивті НЕМЕСЕ)

Бұл операция XOR деп те аталады, оны қарапайым шифрлау үшін жиі қолданады:

```
1 intnumber = 45;
```

```
2 intkey = 102;
3 intencrypt = number ^ key;
4 System.out.println("шифрланған сан: "+encrypt);
5
6 intdecrypt = encrypt ^ key;
7 System.out.println("шифрланған сан: "+ decrypt);
```

Бұл жерде биттік операциялар да орындалады. Егер бізде екі сан үшін де ағымдағы цифрдың әртүрлі мәндері болса, онда 1 қайтарылады, әйтпесе 0 қайтарылады.Мысалы, 9 ^ 5 өрнегінің нәтижесі 12 саны болады. Ал санды ашуға біз қолданамыз нәтижеге кері операция.

- ~ (логикалық теріске шығару)

Санның барлық цифрларын төңкеретін биттік операция: егер разряд мәні 1 болса, онда ол нөлге айналады, ал керісінше.

```
1 bytea = 12;                // 0000 1100
2 System.out.println(~a);    //  1111 0011   или -13
```

Ауыстыру операциялары

Ауыстыру операциялары сандардың цифрларында да орындалады. Ауыстыру оңға және солға жүруі мүмкін.

- a << b - a санын солға b цифрымен ауыстырады. Мысалы, 4 << 1 өрнегі 4 санын (екіліктегі 100-ге тең) бір цифрды солға жылжытады, нәтижесінде ондықта 1000 немесе 8 шығады.
- a >> b - a санын оңға b цифрына ауыстырады. Мысалы, 16 >> 1 16 санын (екілік жүйеде 10000 болатын) бір цифрды оңға жылжытады, яғни нәтиже 1000 немесе ондық санау жүйесінде 8 саны болады.
- a >>> b - ауысымның алдыңғы түрлерінен айырмашылығы, бұл амал белгісіз жылжуды білдіреді - a санын оңға b цифрына ауыстырады. Мысалы, -8 >>> 2 өрнегі 1073741822 тең болады.

Сонымен, егер бір бағытқа немесе екіншісіне жылжу керек бастапқы санды екіге бөлсе, онда іс жүзінде көбейту немесе екіге бөлу алынады. Сондықтан мұндай операцияны тікелей көбейтудің немесе екіге бөлудің орнына қолдануға болады, өйткені аппараттық деңгейде ауысым операциясы бөлу немесе көбейту операциясына қарағанда арзанға түседі.

Шартты өрнектер

Шартты өрнектер қандай да бір шартты білдіреді және логикалық мәнді қайтарады, яғни шын (егер шарт шын болса) немесе жалған (егер шарт жалған болса). Шартты өрнектерге салыстыру және логикалық амалдар жатады.

Салыстыру операциялары

Салыстыру операциялары екі операнды салыстырады және логикалық мәнді береді - өрнек шын болса, ақиқат, ал өрнек жалған болса, жалған.

==

екі операнды теңдік үшін салыстырады және true (операндтар тең болса) және false (операндтар тең болмаса) береді.

```
1 inta = 10;
2 intb = 4;
3 booleanc = a == b;         // false
4 booleand = a == 10;        // true
```

!=

екі операндты салыстырады және операндтар тең ЕМЕС болса, true қайтарады, ал егер операндтар тең болса, false болады.

```
1 inta = 10;
2 intb = 4;
3 booleanc = a != b;         // true
4 booleand = a != 10;        // false
```

< (қарағанда аз)

Егер бірінші операнд екіншіден кіші болса, true қайтарады, әйтпесе false қайтарады.

```
1 inta = 10;
2 intb = 4;
3 booleanc = a < b;   // false
```

> (қарағанда артық)

Егер бірінші операнд екіншісінен үлкен болса, true қайтарады, әйтпесе false қайтарады.

```
1 inta = 10;
2 intb = 4;
3 booleanc = a > b;   // true
```

>= (артық немесе тең)

Егер бірінші операнд екіншіден үлкен немесе оған тең болса, true, әйтпесе false мәнін қайтарады.

```
1 booleanc = 10>= 10;    // true
2 booleanb = 10>= 4;     // true
3 booleand = 10>= 20;    // false
```

<= (кем немесе тең)

Егер бірінші операнд екіншісінен кіші немесе оған тең болса, true мәнін қайтарады, әйтпесе false мәнін береді.

```
1 booleanc = 10<= 10;    // true
2 booleanb = 10<= 4;     // false
```

3 booleand = 10<= 20; // true

Логикалық амалдар

Сондай-ақ, Java-да шартты білдіретін және true немесе false мәндерін беретін және әдетте бірнеше салыстыру операцияларын біріктіретін логикалық операциялар бар. Логикалық операцияларға мыналар жатады:

|

c=a|b; (a немесе b (немесе a мен b екеуі де) true болса, c trueболады, әйтпесе c false болады)

&

c=a&b; (егер a және b екеуі де true болса, c true, әйтпесе c false болады)

!

c=!b; (егер b false болса, c true, әйтпесе c falseболады)

^

c=a^b; (егер a немесе b (бірақ екеуі бірге емес) true болса, c true, әйтпесе c false болады.

||

c=a||b; (c = a || b; (a немесе b (немесе a мен b екеуі де) true болса, c true, әйтпесе c false болады)

&&

c=a&&b; (егер a және b екеуі де true болса, c true, әйтпесе c false болады).

Мұнда бізде екі жұп амал бар | және || (және & және && сияқты) ұқсас нәрселер жасайды, бірақ олар бірдей емес.

C = a | b өрнегі; алдымен a және b мәндерін есептеп шығарады және солардың негізінде нәтиже шығарады.

C = a || b өрнегінде; b; біріншіден, a мәні есептеледі, ал егер ол шын болса, онда b мәнін есептеу мағынасы болмайды, өйткені кез келген жағдайда c ақиқатқа тең болады. B a мәні жалған болған жағдайда ғана бағаланады.

Бұл & /&& жұп операцияларына да қатысты. C = a & b өрнегінде; a да, b де есептеледі.

C = a && b өрнегінде; алдымен a мәні есептеледі, ал егер ол жалған болса, онда b мәнін есептеу бұдан былай мағыналы болмайды, өйткені c мәні бәрібір жалған болады. B a мәні дұрыс болған жағдайда ғана есептеледі

Осылайша, операциялар || және && есептеу үшін мейлінше ыңғайлы, бұл өрнектің мәнін бағалауға кететін уақытты қысқартуға және сол арқылы өнімділікті жақсартуға мүмкіндік береді. Операциялар | және & сандарға разрядтық амалдар орындауға қолайлы.

Мысалдар:

```
1  booleana1 = (5> 6) || (4< 6); // 5 > 6 - false, 4 < 6 - true, сондықтан true
2  қайтарады;
3  booleana2 = (5> 6) || (4> 6); // 5 > 6 - false, 4 > 6 - false, сондықтан false
```

4 қайтарады;
5 booleana3 = (5> 6) && (4< 6); // 5 > 6 - false, сондықтан false қайтарады (4
6 < 6 - true, бірақ есептелмейді)
booleana4 = (50> 6) && (4/ 2< 3); // 50 > 6 - true, 4/2 < 3 - true, сондықтан true қайтарады;
booleana5 = (5> 6) ^ (4< 6); // 5 > 6 - true, сондықтан true қайтарады; 4 < 6 - false)
booleana6 = (50> 6) ^ (4/ 2< 3); // 50 > 6 - true, 4/2 < 3 - true, сондықтан false қайтарады;

Меншіктеу операциялары және операциялардың басымдығы

Соңында, қарапайым тапсырманы басқа операциялармен үйлестіретін меншіктеу операцияларын қарастырыңыз:

- =
  жай ғана бір мәнді екінші мәнге теңестіреді: c = b;
- + =
  c + = b; (c айнымалысына c және b қосу нәтижесі беріледі)
  - =
  c- = b; (c айнымалысы c-тан b-ны азайту нәтижесі беріледі)
- =
  c * = b; (c айнымалысына c және b көбейтіндісінің нәтижесі беріледі)
- / =
  c / = b; (c айнымалысына c-ны b-ге бөлудің нәтижесі беріледі)
- % =
  c% = b; (c айнымалысына c-тің b-ге бөлінуінің қалған бөлігі беріледі)
- & =
  c & = b; (c айнымалысына c & b мәні беріледі)
- | =
  c | = b; (c айнымалысына c | b мәні беріледі)
- ^ =
  c ^ = b; (c айнымалысына c ^ b мәні беріледі)
- << =
  c << = b; (c айнымалысына c << b мәні беріледі)
- >> =
  c >> = b; (c айнымалысына c >> b мәні беріледі)
- >>> =
  c >>> = b; (c айнымалысына c >>> b мәні беріледі)

Операциялардың мысалдары:

```
1 int a = 5;
2 a += 10;       // 15
3 a -= 3;        // 12
4 a *= 2;        // 24
5 a /= 6;        // 4
6 a <<= 4;       // 64
7 a >>= 2;       // 16
```

```
8 System.out.println(a);  // 16
```

Операциялардың басымдығы

Операциялармен жұмыс кезінде олардың басымдығын түсіну маңызды, оны келесі кесте арқылы сипаттауға болады:

```
expr++ expr--
++expr --expr +expr -expr ~ !
* / %
+ -
<<>>>>>
<><= >= instanceof
== !=
&
^
|
&&
||
? : (тернардыоператор)
= += -= *= /= %= &= ^= |= <<= >>= >>>= (меншіктеу операторлары)
```

Осы кестедегі оператор неғұрлым жоғары болса, оның басымдығы соғұрлым жоғары болады. Бұл жағдайда жақша өрнекте қолданылатын операцияның басымдылығын арттырады.

Мәліметтер түрінің түрлендіруі

Әрбір базалық мәліметтер типі жадтың белгілі бір санын алады. Бұл деректердің әртүрлі түрлерін қамтитын операцияларға шектеу қояды. Келесі мысалды қарастырайық:

```
1 int a = 4;
2 byte b = a; // ! қате
```

Бұл кодта біз қателікке тап боламыз. Байт және int екеуі де бүтін сандарды білдірсе де. Сонымен қатар, байт типінің айнымалысына тағайындалатын a айнымалысының мәні байт типінің мәндері шегінде (-128-ден 127-ге дейін). Алайда, біз компиляция уақытында қатеге тап болдық. Себебі бұл жағдайда біз 4 байт болатын кейбір деректерді тек 1 байт болатын айнымалыларға беруге тырысамыз.

Алайда сіздің бағдарламаңызға осы түрлендіруді жасау қажет болуы мүмкін. Бұл жағдайда түрлендіру операциясын (операция ()) қолдану қажет:

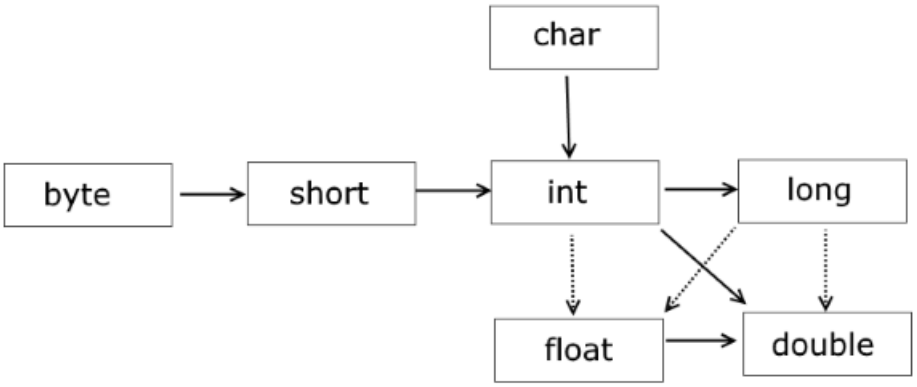
```
1 int a = 4;
2 byte b = (byte)a;  // типті түрлендіру: int типін byte типіне
3 System.out.println(b); // 4
```

Типті түрлендіру операциясы мәнді түрлендіруге тиісті типтің жақша ішіндегі нұсқауын білдіреді. Мысалы, (байт) операциясы кезінде int типі байт түріне ауыстырылады. Нәтижесінде біз байт мәнін аламыз.

Айқын және жасырын түрлендірулер

Бір операцияға әр түрлі типтегі деректер қатысқанда, түрлендіру операциясын қолдану әрдайым қажет емес. Конверсиялардың кейбір түрлері жасырын, автоматты түрде орындалады.

Автоматты түрлендіру



Сурет 2.14 Автоматты түрде түрлендіру

2.14 - суреттегі көрсеткілер түрлендірудің қандай түрін автоматты түрде орындауға болатындығын көрсетеді. Нүктелі көрсеткілер дәлдікті жоғалтатын автоматты түрлендірулерді көрсетеді.

Кеңейту трансформациялары автоматты түрде қиындықсыз жүзеге асырылады - олар объектінің жадтағы көрінісін кеңейтеді. Мысалға:

```
1 byte b = 7;
2 intd = b;  // byte типін int типіне түрлендіру
```

Бұл жағдайда жадында 1 байтты алатын байт 4 байтты алатын int-ге дейін кеңейтіледі.

Кеңейтілген автоматты түрлендірулер келесі тізбектермен ұсынылған:

```
byte -> short -> int -> long
int -> double
short -> float -> double
char -> int
```

Дәлдікті жоғалтатын автоматты түрлендіру

Кейбір түрлендірулер бірдей биттік тереңдіктегі деректер типтері арасында немесе биттік тереңдігі жоғары деректер түрінен төменгі биттік тереңдікке дейін автоматты түрде жасалуы мүмкін. Бұл келесі конверсия тізбектері: int -> өзгермелі, ұзын -> өзгермелі және ұзын -> қосарланған. Олар қатесіз шығарылады, бірақ түрлендіру кезінде ақпарат жоғалуы мүмкін.

Мысалға:

```
1 int a = 2147483647;
2 float b = a;           // int типінен float типіне
3 System.out.println(b);  // 2.14748365E9
```

Айқын түрлендірулер

Примитивті типтердің барлық басқа конверсиялары түрлендіру операциясын нақты қолданады. Әдетте, бұл бит тереңдігі жоғары түрден төменгі бит тереңдікке түрлендірулер:

```
1 long a = 4;
2 int b = (int) a;
```

Түрлендіру кезінде деректердің жоғалуы

Айқын түрлендірулерді қолдану кезінде деректердің жоғалуы мүмкін. Мысалы, келесі кодта бізде қиындықтар болмайды:

```
1 int a = 5;
2 byte b = (byte) a;
3 System.out.println(b);      // 5
```

5 саны байт типінің мәндеріне жақсы сәйкес келеді, сондықтан конверсиядан кейін b айнымалысы 5-ке тең болады. Бірақ келесі жағдайда не болады:

```
1 int a = 258;
2 byte b = (byte) a;
3 System.out.println(b);      // 2
```

Нәтижесі 2. Бұл жағдайда 258 байттың ауқымынан тыс (-128-ден 127-ге дейін), сондықтан мән қысқартылады. Неліктен нәтиже дәл 2 саны болып табылады?

258-ге тең болатын a саны екілік жүйеде 00000000 00000000 00000001 00000010-ға тең болады. Байт типінің мәндері жадында тек 8 битті алады. Демек, int-дің екілік көрінісі оң жақтағы 8 цифрға дейін қысқартылады, яғни 00000010, ондықта 2 шығады.

Рационал сандарды бүтін сандарға қысқарту

Жылжымалы нүкте мәндерін бүтін санға түрлендіру кезінде бөлшек бөлігі қысқартылады:

```
1 double a = 56.9898;
2 int b = (int)a;
```

Мұнда b 56-ға тең болады, дегенмен 57 56.9898-ге жақын болады. Мұндай оқиғаларды болдырмау үшін Java математикалық кітапханасында орналасқан дөңгелектеу функциясын пайдалану қажет:

```
1 double a = 56.9898;
2 int b = (int)Math.round(a);
```

Операциялардағы түрленулер

Көбіне әртүрлі амалдар қолдануға тура келетін жағдайлар болады, мысалы, қосу және өнім, әртүрлі типтегі мәндер бойынша. Мұнда кейбір ережелер қолданылады:

- егер операцияның бір операндысы екі типті болса, онда екінші операнд да қос түрге айналады
- егер алдыңғы шарт орындалмаса және операцияның операндаларының бірі float типіне жатса, онда екінші операнд float типіне де ауысады
- егер алдыңғы шарттар орындалмаса, операцияның бір операндысы ұзақ типті болса, онда екінші операнд та ұзақ түрге айналады
- әйтпесе, операцияның барлық операндтары int типіне ауыстырылады

Түрлендіру мысалдары:

```
1 int a = 3;
2 double b = 4.6;
3 double c = a+b;
```

Операция қос типтегі мәнді қамтитындықтан, басқа мән қосарланған типке беріледі, ал екі мәннің қосындысы a + b қос типті білдіреді.

Тағы бір мысал:

```
1 byte a = 3;
2 short b = 4;
3 byte c = (byte)(a+b);
```

Екі айнымалылар типтік байт және қысқа (екі емес, өзгермелі немесе ұзын емес), сондықтан оларды қосқан кезде олар int-ге айналады, ал олардың a + b қосындысы int-ды білдіреді. Сондықтан, егер біз осы қосынды түрін байт түріндегі айнымалыға тағайындасақ, онда қайтадан түрді байтқа түрлендіру керек.

Егер операцияларда char типіндегі мәліметтер болса, онда олар int-ке айналады:

```
1 int d = 'a' + 5;
2 System.out.println(d);  // 102
```

Шартты құрылымдар

Шартты құрылымдар көптеген бағдарламалау тілдерінің негізгі элементтерінің бірі болып табылады. Бұл конструкциялар белгілі бір жағдайларға байланысты бағдарлама жұмысын жолдардың бірімен бағыттауға мүмкіндік береді.

Java тілі келесі шартты шарттарды қолданады: if..else және switch..case

If / else конструкциясы

If / else өрнегі белгілі бір шарттың ақиқаттығын тексереді және тест нәтижесіне байланысты белгілі бір кодты орындайды:

```
1 int num1 = 6;
2 int num2 = 4;
3 if(num1>num2){
4    System.out.println("Біріншісанекіншіденүлкен");
5 }
```

Шарт **if** кілт сөзінен кейін қойылады. Егер бұл шарт орындалса, онда **if** блогында фигуралы жақшалардан кейін әрі қарай орналастырылатын код іске қосылады. Екі санды салыстыру операциясы шарт ретінде әрекет етеді.

Бұл жағдайда бірінші сан екіншісінен үлкен болғандықтан, num1> num2 өрнегі ақиқатқа айналады. Демек, фигуралы жақшалардан кейін басқару код блогына өтіп, ондағы нұсқауларды, атап айтқанда **System.out.println** («Бірінші сан екіншісінен үлкен») әдісті орындай бастайды; Егер бірінші сан екіншісінен кем немесе тең болса, онда if блогындағы операторлар орындалмайды.

Бірақ егер шарт орындалмаса, қандай-да бір шара қолданылуын қаласақ ше? Бұл жағдайда тағы бір блокты қосуға болады:

```
1 intnum1 = 6;
2 intnum2 = 4;
3 if(num1>num2){
4    System.out.println("Біріншісанекіншіденүлкен ");
5 }
6 else{
7    System.out.println("Біріншісанекіншіденаз ");
8 }
```

Бірақ сандарды салыстыру кезінде үш күйді санауға болады: бірінші сан екіншісінен үлкен, бірінші сан екіншісінен кіші, ал сандар тең. Else if операторының көмегімен біз қосымша шарттарды орындай аламыз:

```
1  int num1 = 6;
2  int num2 = 8;
3  if(num1>num2){
4      System.out.println("Біріншісанекіншіденүлкен ");
5  }
6  else if(num1<num2){
7      System.out.println("Біріншісанекіншіденаз ");
8  }
9  else{
10     System.out.println("Сандартең ");
11}
```

Логикалық операторлардың көмегімен біз бірнеше шартты бірден біріктіре аламыз:

```
1 intnum1 = 8;
2 intnum2 = 6;
3 if(num1 > num2 && num1>7){
4    System.out.println("Первое число больше второго и больше 7");
5 }
```

Мұнда егер num1> num2 ақиқат болса және сол уақытта num1> 7 ақиқат болса, if блогы орындалады.

switch конструкциясы

switch/case конструкциясы if / else конструкциясына ұқсас, өйткені ол бірнеше шартты бірден өңдеуге мүмкіндік береді:

```
1  intnum = 8;
2  switch(num){
3
4     case1:
5        System.out.println("сан 1 тең");
6        break;
7     case8:
8        System.out.println("сан 8 тең");
9        num++;
10       break;
11    case9:
12       System.out.println("сан 9 тең ");
13       break;
14    default:
```

```
15      System.out.println("сан 1, 8, 9 теңемес");
16}
```

Салыстырылатын өрнек жақша ішіндегі **switch** кілт сөзінен кейін келеді. Бұл өрнектің мәні кейстерден кейін қойылған мәндермен дәйекті түрде салыстырылады. Егер сәйкестік табылса, ол тиісті **case** блогын орындайды.

Басқа блоктардың орындалуын болдырмау үшін case блогының соңында break операторы орналастырылады. Мысалы, егер біз break операторын келесі жағдайда алып тастасақ:

```
1 case8:
2    System.out.println("сан 8 тең ");
3    num++;
4 case9:
5    System.out.println("сан 9 тең ");
6    break;
```

онда 8-case блогы орындалады (өйткені num 8-ге тең). Бірақ бұл блокта үзіліс туралы мәлімдеме болмағандықтан, 9-case блогы іске асырыла бастайды.

Егер біз сәйкестік табылмаған жағдайды шешкіміз келсе, онда біз жоғарыдағы мысалдағыдай default блогын қоса аламыз. default блогы қосымша болып табылады.

Біз қатарынан бірнеше case блоктары үшін бір әрекетті анықтай аламыз:

```
1  intnum = 3;
2  intoutput = 0;
3  switch(num){
4
5     case1:
6        output = 3;
7        break;
8     case2:
9     case3:
10    case4:
11       output = 6;
12       break;
13    case5:
14       output = 12;
15       break;
16    default:
17       output = 24;
18}
19System.out.println(output);
```

Үштік операция (Тернарная операция)

Үштік операцияның келесі синтаксисі бар: [бірінші операнд - шарт]? [екінші операнд]: [үшінші операнд]. Осылайша, бұл операцияға бірден үш операнд қатысады. Шартқа байланысты үштік операция екінші немесе үшінші операнды қайтарады: егер шарт true болса, онда екінші операнд қайтарылады; егер шарт false болса, онда үшінші. Мысалға:

```
1 intx=3;
2 inty=2;
3 intz = x<y? (x+y) : (x-y);
4 System.out.println(z);
```

Мұнда үштік операцияның нәтижесі z айнымалысы болады. Алдымен x <y шарты тексеріледі. Ал егер ол орындалса, онда z екінші операндаға тең болады ((x + y), әйтпесе z үшінші операндаға тең болады.

Цикл

Циклдар - басқару құрылымының тағы бір түрі. Ілмектер белгілі бір жағдайларға байланысты белгілі бір әрекетті бірнеше рет орындауға мүмкіндік береді. Java тілінде ілмектердің келесі түрлері бар:

- for
- while
- do...while

for циклы

For циклінің келесі ресми анықтамасы бар:

```
1 for ([есептегіш инициализациясы]; [шарт]; [есептегіш өзгерісі])
2 {
3     // әрекеттер
4 }
```

Стандартты For циклын қарастырамыз:

```
1 for (int i = 1; i < 9; i++){
2     System.out.printf("Санның квадраты %d тең %d \n", i, i * i);
3 }
```

Цикл декларациясының бірінші бөлігі - int i = 1 санауышын жасайды және инициализациялайды. Есептегіш int типінде болмауы керек. Бұл кез-келген басқа сандық тип болуы мүмкін, мысалы, өзгермелі. Цикл орындалмас бұрын санауыш 1 болады. Бұл жағдайда айнымалыны жариялаумен бірдей.

Екінші бөлім - цикл орындалатын шарт. Бұл жағдайда циклі 9-ға жеткенше жұмыс істейді.

Үшінші бөлік есептегішті бір-бірлеп арттырады. Тағы да, бізге бір-бірден ұлғайтудың қажеті жоқ. Төмендетуге болады: i--.

Нәтижесінде, цикл блогы і мәні 9-ға тең болғанша 8 рет жұмыс істейді және әр уақытта бұл мән 1-ге артады.

Бізге циклді жариялау кезінде барлық шарттарды көрсетудің қажеті жоқ. Мысалы, біз келесідей жаза аламыз:

```
1 inti = 1;
2 for(; ;){
3     System.out.printf("Квадратчисла %d равен %d \n", i, i * i);
4 }
```

Циклдің анықтамасы өзгеріссіз қалады, тек енді анықтамадағы блоктар бос: for (;;). Енді инициализацияланған санауыш айнымалысы, шарт жоқ, сондықтан цикл мәңгі жұмыс істейді - шексіз цикл.

Сонымен қатар, сіз бірқатар блоктарды алып тастай аласыз:

```
1 inti = 1;
2 for(; i<9;){
3     System.out.printf("Санның квадраты %d тең %d \n", i, i * i);
4     i++;
5 }
```

Бұл мысал бірінші мысалға балама: бізде санауыш та бар, тек ол циклден тыс жасалған. Бізде циклдің орындалу шарты бар. For блоктың өзінде есептегіш өсімі бар.

For циклы бірнеше айнымалыларды бірден анықтап, басқара алады:

```
1 intn = 10;
2 for(inti=0, j = n - 1; i < j; i++, j--){
3
4     System.out.println(i * j);
5 }
```

do циклы

Do циклі алдымен цикл кодын орындайды, содан кейін while операторындағы шартты тексереді. Бұл шарт дұрыс болғанша, цикл қайталанады. Мысалға:

```
1 intj = 7;
2 do{
3     System.out.println(j);
4     j--;
5 }
6 while(j > 0);
```

Бұл жағдайда цикл коды j нөлге дейін 7 рет жұмыс істейді. Do циклы, while операторындағы шарт дұрыс болмаса да, іс-әрекеттің кем дегенде бір орындалуына кепілдік беретінін ескеру маңызды. Сонымен, біз мынаны жаза аламыз:

```
1 intj = -1;
2 do{
3     System.out.println(j);
4     j--;
5 }
6 while(j > 0);
```

J бастапқыда 0-ден аз болса да, цикл әлі де бір рет орындалады.

While циклы

While циклі қандай да бір шарттың ақиқаттығын бірден тексереді, ал егер шарт шын болса, онда цикл коды орындалады:

```
1 int j = 6;
2 while (j > 0){
3
4     System.out.println(j);
5     j--;
6 }
```

continue және break операторлары

Break операторы циклдан кез-келген уақытта шығуға мүмкіндік береді, тіпті егер цикл өз жұмысын аяқтамаса да:

Мысалға:

```
1 for(inti = 0; i < 10; i++){
2     if(i == 5)
3         break;
4     System.out.println(i);
5 }
```

Есептегіш 5-ке жеткенде, break операторы іске қосылып, цикл аяқталады.

Енді сан 5-ке тең болса, цикл аяқталмай, тек келесі итерацияға ауысатындай етіп жасайық. Ол үшін continue операторын қолданыңыз:

```
1 for(inti = 0; i < 10; i++){
2     if(i == 5)
3         continue;
4     System.out.println(i);
5 }
```

Бұл жағдайда цикл 5 санына жеткенде, бағдарлама жай осы санды өткізіп, келесіге ауысады.

Массивтер

Массив бір типтегі мәндер жиынтығын білдіреді. Массивті жариялау бір мәнді сақтайтын тұрақты айнымалыны жариялауға ұқсас және массивті жариялаудың екі әдісі бар:

```
1 тип_данных название_массива[];
2 // либо
3 тип_данных[] название_массива;
```

Мысалы, сандар жиымын анықтайық:

```
1 int nums[];
2 int[] nums2;
```

Массивті жариялағаннан кейін оны инициалдауға болады:

```
1 int nums[];
2 nums = new int[4];  // 4 саннантұратынжиым
```

Массив келесі конструкцияны қолдану арқылы жасалады: new data_type [number_of_elements], мұнда new - жақшада көрсетілген элементтердің саны үшін жадыны бөлетін кілт сөз. Мысалы, nums = new int [4]; - бұл өрнек төрт int элементтен тұратын жиым жасайды және әр элементтің мәні 0-ге тең болады.

Сіз оны массивті жариялау кезінде бірден бастауға болады:

```
1 int nums[] = new int[4];    // 4 саннантұратынжиым
2 int[] nums2 = new int[5];   // 5 саннантұратынжиым
```

Осы инициализация кезінде массивтің барлық элементтері әдепкі мәнге ие болады. Сандық типтер үшін (оның ішінде char) бұл сан 0, логикалық үшін - жалған, ал басқа нысандар үшін - нөл. Мысалы, int үшін әдепкі мәні 0-ге тең, сондықтан жоғарыдағы сан массиві төрт нөлге тең болады.

Сонымен қатар, сіз оны құрған кезде массив элементтері үшін нақты мәндерді орната аласыз:

```
1 // бұл екі әдіс эквивалентті
2 int[] nums = newint[] { 1, 2, 3, 5 };
3
4 int[] nums2 = { 1, 2, 3, 5 };
```

Бұл жағдайда массивтің өлшемі квадрат жақшаларда көрсетілмейтіндігін ескеру керек, өйткені ол бұйра жақшалардағы элементтер санымен есептеледі.

Массивті құрғаннан кейін оның кез-келген элементтеріне массив айнымалысының атауынан кейін тік жақшаға берілетін индекс бойынша жүгіне аламыз:

```
1 int[] nums = newint[4];
2 // массив элементтерінің мәндерін орнатыңыз
3 nums[0] = 1;
4 nums[1] = 2;
5 nums[2] = 4;
6 nums[3] = 100;
7
8 // жиымның үшінші элементінің мәнін алу
9 System.out.println(nums[2]);    // 4
```

Массив элементтерін индекстеу 0-ден басталады, сондықтан бұл жағдайда массивтің төртінші элементіне сілтеме жасау үшін nums[3] өрнегін қолдану керек.

Бізде тек 4 элемент үшін ғана анықталған жиым болғандықтан, біз, мысалы, алтыншы элементке сілтеме жасай алмаймыз: nums [5] = 5. Егер біз мұны істесек, қате пайда болады.

Массивтің ұзындығы

Массивтің маңызды қасиеті - бұл массивтің ұзындығын, яғни оның элементтерінің санын қайтаратын length қасиеті:

```
1 int[] nums = {1, 2, 3, 4, 5};
2 intlength = nums.length;   // 5
```

Соңғы индекс көбіне белгісіз болады және массивтің соңғы элементін алу үшін біз бұл қасиетті қолдана аламыз:

```
1 intlast = nums[nums.length-1];
```

Көпөлшемді массивтер

Бұрын біз бір өлшемді массивтерді қарастырдық, оларды тізбек немесе бірдей типтегі мәндер тізбегі түрінде ұсынуға болады. Бірақ олар бірөлшемді массивтерден басқа көпөлшемді бола алады. Екі өлшемді массивті көрсететін кесте - ең танымал көп өлшемді массив:

```
1 int[] nums1 = newint[] { 0, 1, 2, 3, 4, 5};
2
3 int[][] nums2 = { { 0, 1, 2}, { 3, 4, 5} };
```

Көрнекі түрде екі массивті де келесідей етіп ұсынуға болады:

Біріолшемді массив nums1

0	1	2	3	4	5

Екі өлшемді массив nums2

0	1	2
3	4	5

Nums2 жиымы екі өлшемді болғандықтан, бұл қарапайым кесте. Ол сондай-ақ келесідей жасалуы мүмкін еді: int [] [] nums2 = new int [2] [3];. Төрт жақшаның саны жиымның өлшемін көрсетеді. Жақшаның ішіндегі сандар жолдар мен бағандар санына арналған. Сонымен қатар, индекстерді қолдана отырып, біз массив элементтерін бағдарламада қолдана аламыз:

```
1
2 // екінші жолдың бірінші бағанының элементін орнатыңыз
3 nums2[1][0]=44;
  System.out.println(nums2[1][0]);
```

3D жиымының анықталуы келесідей болуы мүмкін:

```
1 int[][][] nums3 = new int[2][3][4];
```

Тегістелген массив

Көпөлшемді массивтерді «тегістелген массивтер» түрінде де ұсынуға болады. Жоғарыда келтірілген мысалда екі өлшемді массивте 3 жол және үш баған болды, сондықтан тегіс кесте шықты. Бірақ біз екі өлшемді массивтегі әр элементті бөлек элементтер санымен бөлек массивке тағайындай аламыз:

```
1 int[][] nums = newint[3][];
2 nums[0] = newint[2];
3 nums[1] = newint[3];
4 nums[2] = newint[5];
```

foreach

For циклінің арнайы нұсқасы массивтер мен коллекциялар сияқты элементтер жиынын қайталауға арналған. Бұл басқа бағдарламалау тілдерінде кездесетін foreach цикліне ұқсас. Оның ресми жазылуы:

```
1 for (мәліметтер типң айнымалы_атауы: контейнер){
2     // әрекеттер
3 }
```

Мысалы,

```
1 int[] array = new int[] { 1, 2, 3, 4, 5 };
2 for (int i : array){
3
4    System.out.println(i);
5 }
```

Бұл жағдайда int типіндегі мәліметтер жиыны контейнер рөлін атқарады. Содан кейін int типті айнымалы жарияланады.

Мұны келесі нұсқадағы қарапайым нұсқада жасауға болатын еді:

```
1 int[] array = newint[] { 1, 2, 3, 4, 5};
2 for(inti = 0; i < array.length; i++){
3    System.out.println(array[i]);
4 }
```

Сонымен бірге, for циклінің бұл нұсқасы (int i: array) қарағанда икемді. Атап айтқанда, осы нұсқада біз элементтерді өзгерте аламыз:

```
1 int[] array = newint[] { 1, 2, 3, 4, 5};
2 for(inti=0; i<array.length;i++){
3    array[i] = array[i] * 2;
4    System.out.println(array[i]);
5 }
```

Циклдағы көпөлшемді массивтер қайталануы

```
1  int[][] nums = newint[][]
2  {
3     {1, 2, 3},
4     {4, 5, 6},
5     {7, 8, 9}
6  };
7  for(inti = 0; i < nums.length; i++){
8     for(intj=0; j < nums[i].length; j++){
9
10       System.out.printf("%d ", nums[i][j]);
11    }
12    System.out.println();
13 }
```

Біріншіден, жолдар арқылы қайталау үшін цикл, содан кейін бірінші цикл ішінде белгілі бір жолдың бағандары арқылы қайталау үшін ішкі цикл жасалады. Сол сияқты, сіз 3D өлшемді массивтер мен өлшемдердің көп мөлшерін қайталай аласыз.

Әдістер

Егер айнымалылар мен тұрақтылар кейбір мәндерді сақтаса, онда әдістер белгілі бір әрекеттерді орындайтын операторлар жиынтығын қамтиды.

Әдістердің жалпы анықтамасы келесідей:

```
[модификаторлар] әдіс атауының қайтарылатын мәні типі
([параметрлер]){
   // әдіс денесі
}
```

Модификаторлар мен параметрлер міндетті емес.

Әдепкі бойынша, кез-келген Java бағдарламасының негізгі класы бағдарламаға кіру нүктесі ретінде қызмет ететін негізгі әдісті қамтиды:

```
public static void main(String[] args) {
   System.out.println("салем әлем!");
}
```

public және **static** кілт сөздер модификатор болып табылады. Келесі - қайтару түрі. **Void** кілт сөзі әдіс ешнәрсе әкелмейтінін көрсетеді.

Содан кейін әдіс атауы - **main** және жақша ішіндегі әдіс параметрлері - **String[] args** шығады. Ал фигуралы жақшалар әдіс денесін қоршайды - ол орындайтын барлық әрекеттер.

Тағы бірнеше әдістер жасайық:

```
public class Program{

   public static void main (String args[]){

   }
   void hello(){

      System.out.println("Hello");
   }
   void welcome(){

      System.out.println("Welcome to Java 10");
   }
}
```

Мұнда екі қосымша әдіс анықталды: **hello** және **welcome**, олардың әрқайсысы консольге бірнеше жолды басып шығарады. Әдістер класс ішінде анықталады - бұл жағдайда **main** әдісі анықталған **Program** класы аясында.

Бірақ егер біз осы бағдарламаны құрастырып, іске қосатын болсақ, онда консольде ештеңе көрінбейді. Жоғарыда келтірілген мысалда біз екі әдісті анықтадық, бірақ біз оларды ешқайда шақырмаймыз. Әдепкі бойынша, тек **main** әдісі және оның барлық мазмұны Java бағдарламасында орындалады. Сондықтан, егер басқа әдістер де орындалғанын қаласақ, оларды **main** әдісіне шақыру керек.

Әдіс келесі түрде аталады:

```
1 имя_метода(аргументы);
```

Әдістің атауынан кейін жақшалар келтіріледі, онда аргументтер келтірілген - әдіс параметрлерінің мәндері.

Мысалы, бірнеше әдісті анықтайық және орындайық:

```
1  publicclassProgram{
2
3     publicstaticvoidmain (String args[]){
4
5          hello();
6          welcome();
7          welcome();
8     }
9     staticvoidhello(){
10
11        System.out.println("Hello");
12    }
13    staticvoidwelcome(){
14
15        System.out.println("Welcome to Java 10");
16    }
17}
```

main әдіс hello әдісін бір рет, welcome әдісін екі рет шақырады. Бұл әдістердің артықшылықтарының бірі: біз кейбір жалпы әрекеттерді жеке әдіске ауыстырып, содан кейін оларды бағдарламаның әр түрлі жерлерінде бірнеше рет шақыра аламыз. Екі әдістің де параметрлері болмағандықтан, қоңырау кезінде олардың жақшасынан бос жақша қойылады.

Сонымен қатар, main әдіспен бір класста анықталған басқа әдістерді main әдіске шақыру үшін олардың статикалық модификаторы болуы керек екенін ескеру қажет.

Нәтижесінде, бағдарламаны құрастырғаннан және орындағаннан кейін, консольден көреміз:

```
Hello
Welcome to Java 10
Welcome to Java 10
```

Әдістің параметрлері

Параметрлердің көмегімен біз есептеулер үшін қолданылатын әдістерге әр түрлі мәліметтерді жібере аламыз.

Мысалға:

```
1 staticvoidsum(intx, inty){
2
3    intz = x + y;
4    System.out.println(z);
5 }
```

Бұл функция екі параметрді - екі санды алады, оларды қосады және олардың қосындысын консольға шығарады.

Бағдарламада осы әдісті шақырған кезде біз параметрлердің орнына параметр түріне сәйкес келетін мәндерді беруіміз керек:

```
1  publicclassProgram{
2
3     publicstaticvoidmain (String args[]){
4
5        inta = 6;
6        intb = 8;
7        sum(a, b);  // 14
8        sum(3, a);  // 9
9        sum(5, 23); // 28
10    }
11    staticvoidsum(intx, inty){
12
13       intz = x + y;
14       System.out.println(z);
15    }
16}
```

sum әдісі **int** типінің екі мәнін қабылдайтын болғандықтан, параметрлер орнына **int** типінің екі мәні өту керек. Бұл **int** типін көрсететін немесе int типіне автоматты түрде айналдырылатын мәліметтер түрінің сандық литералдары немесе айнымалылары болуы мүмкін. Параметрлердің орнына берілген шамалар аргументтер деп те аталады. Параметрлерге мәндер позиция бойынша беріледі, яғни бірінші параметрге бірінші аргумент, екінші параметрге екінші аргумент және т.б.

Тағы бір мысалды қарастырайық:

```
1  publicclassProgram{
2
```

```
3     publicstaticvoidmain (String args[]){
4
5        display("Tom", 34);
6        display("Bob", 28);
7        display("Sam", 23);
8     }
9     staticvoiddisplay(String name, intage){
10
11       System.out.println(name);
12       System.out.println(age);
13    }
14}
```

display әдісі екі параметрді алады. Бірінші параметр **String** типінде, ал екіншісі **int** типінде. Сондықтан әдісті шақырған кезде алдымен оған жолды, содан кейін санды беру керек.

Айнымалы ұзындық параметрлері

Әдіс бірдей типтегі айнымалы ұзындық параметрлерін қабылдай алады. Мысалы, біз әдіске сандар жиынтығын өткізіп, олардың қосындысын есептеуіміз керек, бірақ нақты қанша сандар өтетінін білмейміз - 3, 4, 5 немесе одан да көп. Ұзындықтың айнымалы параметрлері келесі мәселені шешуге мүмкіндік береді:

```
1  publicclassProgram{
2
3     publicstaticvoidmain (String args[]){
4
5        sum(1, 2, 3);          // 6
6        sum(1, 2, 3, 4, 5);    // 15
7        sum();                 // 0
8     }
9     staticvoidsum(int...nums){
10
11       intresult =0;
12       for(intn: nums)
13          result += n;
14       System.out.println(result);
15    }
16}
```

Int ... nums параметрінің атауының алдындағы эллипс оның міндетті емес екендігін және массивті білдіретінін көрсетеді. Қосынды әдісіне біз бір санды, бірнеше сандарды бере аламыз немесе мүлдем ешқандай

параметр бере алмаймыз. Сонымен қатар, егер біз бірнеше параметрлерді өткізгіміз келсе, онда міндетті емес параметр соңында көрсетілуі керек:

```
1  public static void main(String[] args) {
2
3     sum("Welcome!", 20,10);
4     sum("Hello World!");
5  }
6  static void sum(String message, int ...nums){
7
8     System.out.println(message);
9     int result =0;
10    for(int x:nums)
11       result+=x;
12    System.out.println(result);
13}
```

Return операторы. Әдістің нәтижесі

Әдістер белгілі бір мәнді қайтара алады. Ол үшін return операторы қолданылады.

```
1 returnвозвращаемое_значение;
```

return операторынан кейін қайтарылатын мән шығады, бұл әдіс нәтижесі болып табылады. Бұл әріптік мән, айнымалының мәні немесе қандай да бір күрделі өрнек болуы мүмкін.

Мысалға:

```
1  publicclassProgram{
2
3     publicstaticvoidmain (String args[]){
4
5        intx = sum(1, 2, 3);
6        inty = sum(1, 4, 9);
7        System.out.println(x);  // 6
8        System.out.println(y);  // 14
9     }
10    staticintsum(inta, intb, intc){
11
12       returna + b + c;
13    }
14}
```

Әдіс бос түрінің **void** орнына қайтару түрі ретінде кез-келген басқа түрін қолданады. Бұл жағдайда **sum** әдісі **int** типінің мәнін қайтарады,

сондықтан бұл тип әдіс атауының алдында көрсетіледі. Сонымен қатар, егер void-тен басқасы әдіс үшін қайтару түрі ретінде анықталса, онда әдіс мәнді қайтару үшін return операторын қолдануы керек.

Бұл жағдайда қайтарылатын мән әрдайым функция анықтамасында көрсетілгендей типте болуы керек. Ал егер функция int типінің мәнін қайтарса, онда **return** операторынан кейін **int** типінің объектісі болатын бүтін мән шығады. Бұл жағдайда сияқты, бұл әдіс параметрлерінің мәндерінің қосындысы.

Әдіс кейбір жағдайларға байланысты әр түрлі мәндерді қайтару үшін return операторына бірнеше рет шақыруды қолдана алады:

```
1  publicclassProgram{
2
3     publicstaticvoidmain (String args[]){
4
5        System.out.println(daytime(7));     // Good morning
6        System.out.println(daytime(13));    // Good after noon
7        System.out.println(daytime(18));    // Good evening
8        System.out.println(daytime(2));     // Good night
9     }
10    staticString daytime(inthour){
11
12       if(hour >24|| hour < 0)
13          return"Invalid data";
14       elseif(hour > 21|| hour < 6)
15          return"Good night";
16       elseif(hour >= 15)
17          return"Good evening";
18       elseif(hour >= 11)
19          return"Good after noon";
20       else
21          return"Good morning";
22    }
23}
```

Мұнда **daytime** әдісі **String** мәнін, яғни жолды қайтарады және **hour** параметрінің мәніне байланысты, қайтарылған жол әр түрлі болады.

Әдістен шығу

Return операторы әдістің мәнін қайтару үшін, сонымен қатар әдістен шығу үшін қолданылады. Ұқсас көлемде return операторы ешнәрсе қайтармайтын әдістерде қолданылады, яғни олар void типіне ие:

```
1  publicclassProgram{
```

```
2
3     publicstaticvoidmain (String args[]){
4
5         daytime(7);     // Good morning
6         daytime(13);    // Good after noon
7         daytime(32);    //
8         daytime(56);    //
9         daytime(2);     // Good night
10    }
11    staticvoiddaytime(inthour){
12
13        if(hour >24|| hour < 0)
14            return;
15        if(hour > 21|| hour < 6)
16            System.out.println("Good night");
17        elseif(hour >= 15)
18            System.out.println("Good evening");
19        elseif(hour >= 11)
20            System.out.println("Good after noon");
21        else
22            System.out.println("Good morning");
23    }
24}
```

Егер datetime әдісіне жіберілген мән 24-тен үлкен немесе 0-ден аз болса, онда жай ғана әдістен шығыңыз. Бұл жағдайда **return** кейін қайтарылатын мәнді көрсету қажет емес.

Әдісті шамадан тыс жүктеу

Бағдарламада біз бірдей атаумен, бірақ әртүрлі типтермен және / немесе параметрлер санымен әдістерді қолдана аламыз. Бұл механизм әдісті шамадан тыс жүктеу (method overloading) деп аталады.

Мысалға:

```
1  publicclassProgram{
2
3     publicstaticvoidmain(String[] args) {
4
5         System.out.println(sum(2, 3));          // 5
6         System.out.println(sum(4.5, 3.2));      // 7.7
7         System.out.println(sum(4, 3, 7));       // 14
8     }
9     staticintsum(intx, inty){
10
```

```
11        returnx + y;
12    }
13    staticdoublesum(doublex, doubley){
14
15        returnx + y;
16    }
17    staticintsum(intx, inty, intz){
18
19        returnx + y + z;
20    }
21}
```

Мұнда sum () әдісінің үш нұсқасы немесе үш шамадан тыс жүктемесі анықталған, бірақ ол шақырылған кезде, берілген параметрлердің типіне және санына байланысты, жүйе ең қолайлы нұсқаны таңдайды.

Шамадан тыс жүктеуге параметрлердің саны мен түрлері әсер ететіндігін атап өткен жөн. Алайда, шамадан тыс жүктеме үшін қайтару түріндегі айырмашылық маңызды емес. Мысалы, келесі жағдайда әдістер қайтару түрінде ерекшеленеді:

```
1  publicclassProgram{
2
3     publicstaticvoidmain(String[] args) {
4
5        System.out.println(sum(2, 3));
6        System.out.println(sum(4, 3));
7     }
8     staticintsum(intx, inty){
9
10       returnx + y;
11    }
12    staticdoublesum(intx, inty){
13
14       returnx + y;
15    }
16}
```

Алайда, бұл шамадан тыс жүктеме деп саналмайды. Сонымен қатар, мұндай бағдарлама дұрыс емес және жай құрастырылмайды, өйткені параметрлер саны мен типі бірдей әдіс бірнеше рет анықталған.

Рекурсивті функциялар

Рекурсивті функцияларды бөлек қарастырайық. Рекурсивті функциялардың әдеттегі әдістерден басты айырмашылығы - олар өздерін шақыра алатын рекурсивті функция.

Мысалы, санның факториалын есептейтін функцияны қарастырайық:

```
1 static int factorial(int x){
2
3    if (x == 1){
4
5        return 1;
6    }
7    return x * factorial(x - 1);
8 }
```

Алдымен шарт тексеріледі: егер кіріс саны 1-ге тең болмаса, онда біз бұл санды параметр ретінде x-1 саны берілген сол функцияның нәтижесіне көбейтеміз. Яғни рекурсивті түсу пайда болады. Сонымен, параметр мәні біреуіне тең емес нүктеге жеткенше.

Рекурсивті функция міндетті түрде return операторын қолданатын және функцияның басында орналасатын негізгі нұсқасы болуы керек. Факториалды жағдайда, if (x == 1) return 1;. Барлық рекурсивті қоңыраулар ақыр соңында негізгі жағдайға жақындайтын ішкі функцияларға қол жеткізуі керек. Сонымен, функцияға оң санды беру кезінде, субфункциялардың рекурсивті шақыруларымен бірге, әр жолы оларға бір-бірден кем сан беріледі. Ақыр соңында, біз сан 1 болатын жағдайға жетеміз, ал негізгі жағдай қолданылады.

Бұл жағдайда факториалды анықтауға арналған циклдарға негізделген оңтайлы шешімдер бар екенін атап өткен жөн:

```
1 static int factorial(int x){
2    int result=1;
3    for (int i = 1; i <= x; i++)
4    {
5        result *= i;
6    }
7    return  result;
8 }
```

Рекурсивті функцияның тағы бір кең таралған мысалы - Фибоначчи сандарын есептейтін функция. Теорияда Фибоначчи тізбегінің n-ші мүшесі формула бойынша анықталады: f (n) = f (n-1) + f (n-2), және f (0) = 0, және f (1) = 1.

```
1  static int fibonachi(int n){
2
3     if (n == 0){
4         return 0;
5     }
6     if (n == 1){
```

```
7         return 1;
8     }
9     else{
10        return fibonachi(n - 1) + fibonachi(n - 2);
11    }
12}
```

Ерекше жағдайларды өңдеуге кіріспе

Бағдарламаны орындау кезінде жиі қателер болуы мүмкін, бұл міндетті түрде жасаушының кінәсінен емес. Олардың кейбірін болжау немесе болжау қиын, ал кейде мүмкін емес. Мысалы, файлды тасымалдау кезінде желілік байланыс күтпеген жерден аяқталуы мүмкін. Бұл жағдайлар ерекше жағдайлар деп аталады.

Java тілі мұндай жағдайларды шешуге арналған арнайы құралдарды ұсынады. Осындай нысандардың бірі - бұл **try...catch...finally** конструкциясы. Ерекшелік **try** блогына шығарылған кезде, басқару ерекше жағдайды басқара алатын **catch** блогына өтеді. Егер мұндай блок табылмаса, пайдаланушыға өңделмеген ерекшелік туралы хабарлама көрсетіледі, әрі қарай бағдарламаның орындалуы тоқтатылады. Мұндай тоқтау болмауы үшін, сіз **try..catch** блогын қолдануыңыз керек. Мысалға:

```
int[] numbers = new int[3];
numbers[4]=45;
System.out.println(numbers[4]);
```

Біздің **numbers** массивінде тек 3 элемент болуы мүмкін болғандықтан, **numbers[4]=45** операторы орындалған кезде консоль ерекше жағдайды көрсетеді және бағдарлама аяқталады. Енді осы ерекшелікті қолдануға тырысайық:

```
try{
   int[] numbers = new int[3];
   numbers[4]=45;
   System.out.println(numbers[4]);
}
catch(Exception ex){

   ex.printStackTrace();
}
System.out.println("Программа завершена");
```

0

Try ... catch блогын қолданған кезде, try және catch операторларының арасындағы барлық операторлар алдымен орындалады. Егер ерекшелік **try** блогына жіберілсе, қалыпты орындалу реті тоқтайды және catch

операторына ауысады. Демек, бағдарламаның орындалуы numbers[4]=45; жеткенде, бағдарлама тоқтап, catch блогына өтеді.

catch өрнегінде келесі синтаксис бар: catch (ерекшелік_типі айнымалы_аты). Бұл жағдайда Exc айнымалысы жарияланады, ол Exception типіне жатады. Бірақ егер лақтырылған ерекше жағдай catch операторында көрсетілген типтегі ерекшелік болмаса, онда ол өңделмейді және бағдарлама жай қате туралы хабарламаны іліп қояды.

Exception типі барлық ерекшеліктер үшін негізгі класс болғандықтан, catch (Exception ex) операторы барлық ерекше жағдайларды өңдейді. Ерекше жағдайды өңдеу бұл жағдайда консольға қате іздері стегін Exception класында анықталған printStackTrace () әдісі арқылы көрсетуге дейін азаяды.

catch блогы орындалуды аяқтағаннан кейін, бағдарлама өз жұмысын жалғастырады, catch блогынан кейінгі барлық операторларды орындайды.

Try..catch құрылымында finally блогы болуы мүмкін. Алайда, бұл блок қосымша болып табылады және ерекше жағдайларды қарастырған кезде оны жіберіп алуға болады. finally блогы **try** блогына ерекше жағдай қойылса да, болмаса да кез-келген жағдайда орындалады:

```
1  try{
2      int[] numbers = new int[3];
3      numbers[4]=45;
4      System.out.println(numbers[4]);
5  }
6  catch(Exception ex){
7
8      ex.printStackTrace();
9  }
10finally{
11    System.out.println("Блок finally");
12}
13System.out.println("Программа завершена");
```

Бірнеше ерекшеліктерді қолдану

Java-да көптеген ерекшеліктер бар, және біз оларды өңдеуді қосымша **catch** блоктарын қосу арқылы ажыратуға болады:

```
1  int[] numbers = new int[3];
2  try{
3      numbers[6]=45;
4      numbers[6]=Integer.parseInt("gfd");
5  }
6  catch(ArrayIndexOutOfBoundsException ex){
7
```

```
8     System.out.println("Выходзапределымассива");
9  }
10catch(NumberFormatException ex){
11
12    System.out.println("Ошибкапреобразованияизстрокивчисло");
13}
```

Егер біз белгілі бір типтегі ерекшелікті алсақ, онда ол тиісті **catch** блогына өтеді.

throw операторы

Сіз өз бағдарламаңыздағы ерекшеліктің орындалуын білдіру үшін **throw** операторын қолдана аласыз. Яғни, осы оператордың көмегімен біз өзіміз ерекше жағдай жасай аламыз және оны жұмыс кезінде шақыра аламыз. Мысалы, біздің бағдарламада сан енгізіледі, егер біз 30-дан үлкен болса, ерекше жағдай жасалғанын қалаймыз:

```
1  packagefirstapp;
2
3  importjava.util.Scanner;
4  publicclassFirstApp {
5
6     publicstaticvoidmain(String[] args) {
7
8        try{
9           Scanner in = newScanner(System.in);
10          intx = in.nextInt();
11          if(x>=30){
12             thrownewException("Число x должно быть меньше 30");
13          }
14       }
15       catch(Exception ex){
16
17          System.out.println(ex.getMessage());
18       }
19       System.out.println("Программа завершена");
20    }
21}
```

Мұнда Exception класының конструкторы ерекше жағдай туралы хабарлама жіберілетін ерекше объект құру үшін қолданылады. Егер x саны 29-дан көп болып шықса, онда ерекшелік шығарылады және басқару **catch** блогына өтеді.

Catch блогында біз айрықша хабарламаны getMessage () әдісі арқылы ала аламыз.

2.3. Объектіге бағытталған бағдарламалау

Java - бұл объектіге бағытталған тіл, сондықтан онда «класс» және «объект» сияқты ұғымдар шешуші рөл атқарады. Кез-келген Java бағдарламасын өзара әрекеттесетін объектілер жиынтығы ретінде қарастыруға болады.

Нысанның шаблоны немесе сипаттамасы класс болып табылады, ал объект сол сыныптың данасын білдіреді. Сондай-ақ келесі ұқсастықты салуға болады. Біздің бәрімізде адам туралы бірнеше түсінік бар - екі қолдың, екі аяғының, бастың, торсықтың және т.б. Кейбір үлгілер бар - бұл үлгіні класс деп атауға болады. Шынында бар адам (шын мәнінде, осы сыныптың данасы) осы сыныптың объектісі болып табылады.

class кілт сөзінің көмегімен анықталады:

```
classPerson{

}
```

Бұл жағдайда сынып Person деп аталады. Кластың атауынан кейін бұлардың арасында дененің корпусы - яғни оның өрістері мен тәсілдері орналастырылған бұйра жақшалар пайда болады.

Кез-келген объект екі негізгі сипаттамаға ие болуы мүмкін: күй - объект сақтайтын кейбір деректер және мінез-құлық - объект орындай алатын әрекеттер.

Класта объектінің күйін сақтау үшін өрістер немесе сынып айнымалылары қолданылады. Әдістер сыныптағы объектінің мінез-құлқын анықтау үшін қолданылады. Мысалы, адамды бейнелейтін Person класында келесі анықтама болуы мүмкін:

```
1 classPerson{
2
3    String name;       // имя
4    intage;          // возраст
5    voiddisplayInfo(){
6        System.out.printf("Name: %s \tAge: %d\n", name, age);
7    }
8 }
```

Person класы екі өрісті анықтайды: аты адамның атын, ал жасы олардың жасын білдіреді. Сондай-ақ displayInfo әдісі анықталған, ол ештеңе қайтармайды және тек консольға осы деректерді шығарады.

Енді біз осы класты қолданамыз. Ол үшін келесі бағдарламаны анықтаймыз:

```
1  publicclassProgram{
2
3     publicstaticvoidmain(String[] args) {
4
5        Person tom;
6     }
7  }
8  classPerson{
9
10    String name;    // имя
11    intage;        // возраст
12    voiddisplayInfo(){
13       System.out.printf("Name: %s \tAge: %d\n", name, age);
14    }
15}
```

Әдетте, кластар әр түрлі файлдарда анықталады. Бұл жағдайда қарапайымдылық үшін біз бір файлдағы екі классты анықтаймыз. Айта кету керек, бұл жағдайда тек бір класс жалпы модификаторға ие бола алады (бұл жағдайда ол Бағдарлама класы), ал кодтық файлдың өзі осы сыныптың атымен аталуы керек, яғни бұл жағдайда, файлға Program.java атауы керек.

Класс жаңа типті білдіреді, сондықтан берілген типті білдіретін айнымалыларды анықтай аламыз. Сонымен, негізгі әдісте Person класын білдіретін tom айнымалысы анықталған. Бірақ әзірге бұл айнымалы ешқандай нысанды көрсетпейді және ол әдепкі бойынша нөл болады. Жалпы, біз оны әлі қолдана алмаймыз, сондықтан алдымен Person класының объектісін құру керек.

Конструкторлар

Кәдімгі әдістерден басқа, кластар конструктор деп аталатын арнайы әдістерді анықтай алады. Осы кластың жаңа объектісі құрылған кезде конструкторлар деп аталады. Конструкторлар инициализацияны орындайды.

Егер класта конструктор анықталмаса, автоматты түрде осы класс үшін параметрсіз конструктор құрылады.

Жоғарыда анықталған Person класында ешқандай конструкторлар жоқ. Сондықтан ол үшін автоматты түрде конструктор жасалады, оны біз Person объектісін құру үшін қолдана аламыз. Нақтырақ, бір объект жасайық:

```
1  publicclassProgram{
```

```
2
3     publicstaticvoidmain(String[] args) {
4
5         Person tom = newPerson(); // созданиеобъекта
6         tom.displayInfo();
7
8         // изменяем имя и возраст
9         tom.name = "Tom";
10        tom.age = 34;
11        tom.displayInfo();
12    }
13}
14classPerson{
15
16    String name;    // имя
17    intage;        // возраст
18    voiddisplayInfo(){
19        System.out.printf("Name: %s \tAge: %d\n", name, age);
20    }
21}
```

Person объектісін құру үшін new Person () өрнегі қолданылады. Жаңа оператор Person объектісі үшін жадыны бөледі. Содан кейін ешқандай параметрлер қабылдамайтын стандартты конструктор шақырылады. Нәтижесінде, осы өрнек орындалғаннан кейін, Person объектісінің барлық деректері сақталатын бөлімге жады бөлінеді. Том айнымалысы құрылған объектіге сілтеме алады.

Егер конструктор объектінің айнымалыларының мәндерін инициализацияламаса, онда олар әдепкі мәндерді алады. Сандық типтердің айнымалылары үшін бұл сан 0-ге тең, ал типтік жолдар мен кластар үшін ол null тең (яғни іс жүзінде ешқандай мән жоқ).

Нысанды құрғаннан кейін, біз айнымалысы Томның айнымалылары арқылы том айнымалысы арқылы қол жеткізе аламыз және олардың мәндерін орнатамыз немесе аламыз, мысалы tom.name = «Tom».

Нәтижесінде біз консольден көреміз:

```
Name: null              Age: 0
Name: Tom               Age: 34
```

Егер сізге объектіні құру кезінде қандай да бір логика қажет болса, мысалы, класс өрістері белгілі бір мәндерді алуы үшін, онда сіз өзіңіздің конструкторларыңызды класта анықтай аласыз. Мысалға:

```
1   publicclassProgram{
2
3       publicstaticvoidmain(String[] args) {
```

```
4
5    Person bob = newPerson();      //
6   параметрсізбіріншіконструктордышақыру
7           bob.displayInfo();
8
9    Person tom = newPerson("Tom"); // екінші конструкторды бір
10  параметрмен шақыру
11          tom.displayInfo();
12
13  Person sam = newPerson("Sam", 25); //3 конструкторды екі
14  параметрмен шақыру
15          sam.displayInfo();
16      }
17  }
18  classPerson{
19
20      String name;    // аты
21      intage;        //жасы
22      Person()
23      {
24          name = "Undefined";
25          age = 18;
26      }
27      Person(String n)
28      {
29          name = n;
30          age = 18;
31      }
32      Person(String n, inta)
33      {
34          name = n;
35          age = a;
36      }
37      voiddisplayInfo(){
            System.out.printf("Name: %s \tAge: %d\n", name, age);
        }
    }
```

Енді класта үш конструктор бар, олардың әрқайсысы әртүрлі параметрлер санын алады және сынып өрістерінің мәндерін орнатады.

Бағдарламаның шығыс консолі:

```
Name: Undefined                 Age: 18
Name: Tom               Age: 18
Name: Sam               Age: 25
```

this кілт сөз

this кілт сөз кластың ағымдағы данасына сілтемені білдіреді. Осы кілт сөз арқылы біз айнымалыларға, объектінің әдістеріне сілтеме жасай аламыз, сонымен қатар оның конструкторларын шақыра аламыз.

Мысалға:

```
1  public class Program{
2
3      public static void main(String[] args) {
4
5          Person undef = new Person();
6          undef.displayInfo();
7
8          Person tom = new Person("Tom");
9          tom.displayInfo();
10
11         Person sam = new Person("Sam", 25);
12         sam.displayInfo();
13     }
14}
15class Person{
16
17     String name;    // имя
18     int age;        // возраст
19     Person()
20     {
21         this("Undefined", 18);
22     }
23     Person(String name)
24     {
25         this(name, 18);
26     }
27     Person(String name, int age)
28     {
29         this.name = name;
30         this.age = age;
31     }
32     void displayInfo(){
33         System.out.printf("Name: %s \tAge: %d\n", name, age);
34     }
35}
```

Үшінші конструкторда параметрлер класс өрістерімен бірдей аталады. Өрістер мен параметрлерді ажырату үшін келесі кілт сөз қолданылады:

```
1 this.name = name;
```

Сонымен, бұл жағдайда біз name параметрінің мәні name өрісіне берілгенін көрсетеміз.

Сонымен қатар, бізде бір нәрсе жасайтын үш конструктор бар: name және age өрістерін орнатыңыз. Қайталауды болдырмау үшін, this пайдаланып сіз класс конструкторларының біріне қоңырау шалып, оның параметрлері үшін қажетті мәндерді бере аласыз:

```
1 Person(String name)
2 {
3     this(name, 18);
4 }
```

Нәтижесінде бағдарламаның нәтижесі алдыңғы мысалдағыдай болады.

Инициализаторлар

Конструктордан басқа, объектіні бастапқы инициализациялау объект инициализаторының көмегімен жүзеге асырылуы мүмкін. Инициализатор кез-келген конструктордан бұрын жұмыс істейді. Яғни барлық инструкторларға ортақ инициализаторға код қоя аламыз:

```
1  publicclassProgram{
2
3     publicstaticvoidmain(String[] args) {
4
5         Person undef = newPerson();
6         undef.displayInfo();
7
8         Person tom = newPerson("Tom");
9         tom.displayInfo();
10    }
11}
12classPerson{
13
14    String name;    // имя
15    intage;        // возраст
16
17    /*начало блока инициализатора*/
18    {
```

```
19      name = "Undefined";
20      age = 18;
21  }
22  /*конец блока инициализатора*/
23  Person(){
24
25  }
26  Person(String name){
27
28      this.name = name;
29  }
30  Person(String name, intage){
31
32      this.name = name;
33      this.age = age;
34  }
35  voiddisplayInfo(){
36      System.out.printf("Name: %s \tAge: %d\n", name, age);
37  }
38}
```

Консоль шығысы:

```
Name: Undefined          Age: 18
Name: Tom       Age: 18
```

Пакеттер

Әдетте, Java-да кластар жинақталған. Пакеттер кластарды логикалық түрде жиынтықта ұйымдастыруға мүмкіндік береді. Әдепкіде, java-да java.lang, java.util, java.io және т.с.с. сияқты бірнеше кіріктірілген пакеттер бар. Сонымен қатар, пакеттерде кірістірілген пакеттер болуы мүмкін.

Кластарды пакет ретінде ұйымдастыра отырып, сіз кластар арасындағы атау қайшылықтарынан аулақ боласыз. Өйткені, әзірлеушілер өз кластарын бірдей ат қоюы ғажап емес. Пакеттегі мүшелік аттардың бір мағыналы болуын қамтамасыз етеді.

Класс белгілі бір бумаға жататынын көрсету үшін package директивасын, содан кейін буманың атын қолданыңыз:

```
package название_пакета;
```

Әдетте бума атаулары жобаның физикалық құрылымына, яғни бастапқы файлдар орналасқан каталогтарды ұйымдастыруға сәйкес келеді. Ал жоба ішіндегі файлдарға жол осы файлдар бумасының атына сәйкес келеді. Мысалы, егер кластар mypack бумасына жатса, онда бұл кластар mypack папкасында жобада орналастырылады.

Пакеттерде кластарды анықтау қажет емес. Егер класс үшін бума анықталмаса, онда бұл класс атауы жоқ әдепкі бумада болып саналады.

Мысалы, бастапқы файлдар үшін папкада study каталогын құрайық. Онда Program.java файлын келесі кодпен жасаңыз:

```
1  packagestudy;
2
3  publicclassProgram{
4
5     publicstaticvoidmain(String[] args) {
6
7        Person kate = newPerson("Kate", 32);
8        kate.displayInfo();
9     }
10}
11classPerson{
12
13   String name;
14   intage;
15
16   Person(String name, intage){
17      this.name = name;
18      this.age = age;
19   }
20   voiddisplayInfo(){
21      System.out.printf("Name: %s \t Age: %d \n", name, age);
22   }
23}
```

Файлдың басындағы package study директивасы мұнда анықталған Program және Person сыныптары study бумасына жататындығын көрсетеді.

Біз даму ортасында жұмыс істеген кезде, мысалы, Netbeans-те IDE бумалар мен олардың файлдарының барлық компиляциясына жауап береді. Тиісінше, біз тек батырманы басуымыз керек, және бәрі дайын болады. Алайда, егер біз бағдарламаны командалық жолға құрастырсақ, онда біз кейбір қиындықтарға тап болуымыз мүмкін. Сондықтан біз бұл жағын қарастырамыз.

Бағдарламаны құрастыру үшін алдымен командалық жолда / терминалда cd пәрменін қолданып, study каталогы орналасқан бумаға өтіңіз.

```
cd C:\java
```

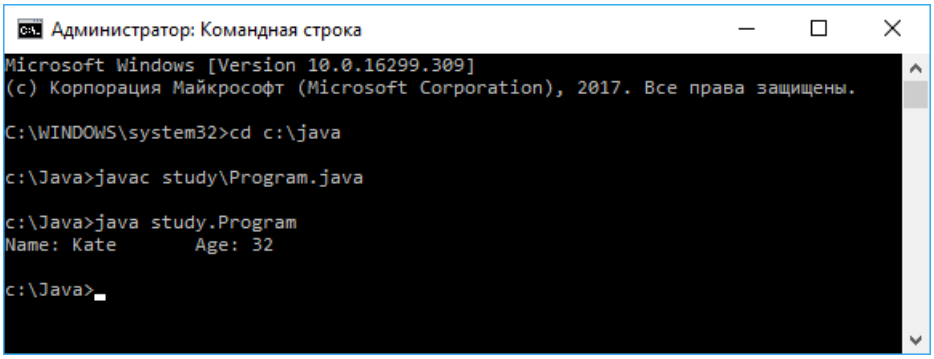
Мысалы, бұл жағдайда бұл C: \ java каталогы (яғни бастапқы код файлы C: \ java \ study \ Program.java жолында орналасқан).

Компиляциялау үшін команданы іске қосыңыз.

```
javac study\Program.java
```

Осыдан кейін Program.class және Person.class жинақталған файлдары study қалтасында пайда болады. Бағдарламаны іске қосу үшін келесі пәрменді іске қосыңыз:

```
java study.Program
```



Сурет 2.14

Пакеттер мен кластарды импорттау

Егер бізге басқа бумалардан кластарды қолдану қажет болса, онда біз бұл бумалар мен кластарды қосуымыз керек. Ерекшелік - java.lang пакетінен (мысалы, String), олар автоматты түрде бағдарламаға енеді.

Мысалы, өткен тақырыптармен таныс Scanner класы java.util бумасында, сондықтан біз оған келесідей қол жеткізе аламыз:

```
1 java.util.Scanner in = new java.util.Scanner(System.in);
```

Яғни, оның нысанын құру кезінде бумада файлға толық жолды көрсетеміз. Алайда, бума атауларының мұндай аласапыраны әрдайым ыңғайлы бола бермейді және балама ретінде біз package директивасынан кейін көрсетілген импорт директивасының көмегімен бумалар мен кластарды импорттай аламыз:

```
1  package study;
2
3  import java.util.Scanner; // импорткласса Scanner
4
5  public class Program {
6
7     public static void main(String[] args) {
8
9         Scanner in = new Scanner(System.in);
```

```
10    }
11}
```

import директивасы кодтың басында көрсетіледі, содан кейін плагин класының аты көрсетіледі (бұл жағдайда Scanner класы).

Жоғарыда келтірілген мысалда біз тек бір ғана класты қамтыдық, бірақ java.util бумасында тағы көптеген класстар бар. Әр класты бөлек байланыстырмас үшін, біз барлық пакетті дереу байланыстыра аламыз:

```
1 import java.util.*; // java.util пакетінен барлық класты импорттау
```

Енді біз java.util пакетінен кез-келген класты қолдана аламыз.

Мүмкін, біз екі бірдей пакетті екі түрлі пакеттен қолданып жатқан болармыз, мысалы, Date класы java.util бумасында да, java.sql бумасында да бар. Егер бізге осы екі классты бір уақытта қолдану қажет болса, онда біз осы кластарға толық жолды пакетте көрсетуіміз керек:

```
1 java.util.Date utilDate = new java.util.Date();
2 java.sql.Date sqlDate = new java.sql.Date();
```

Статикалық импорт

Java-да импорттың ерекше түрі - статикалық импорт бар. Ол үшін static модификаторы import директивасымен бірге қолданылады:

```
1  package study;
2
3  import static java.lang.System.*;
4  import static java.lang.Math.*;
5
6  public class Program {
7
8      public static void main(String[] args) {
9
10         double result = sqrt(20);
11         out.println(result);
12     }
13}
```

Бұл жерде System және Math кластарының статикалық импорты орын алады. Бұл кластарда статикалық әдістер бар. Статикалық импорттау операциясының арқасында біз бұл әдістерді класс атауынсыз қолдана аламыз. Мысалы, Math.sqrt (20) емес, sqrt (20) жазу керек, өйткені санның квадрат түбірін қайтаратын sqrt () функциясы тұрақты болады.

System класы үшін де солай: ол out статикалық нысанын анықтайды, сондықтан оны класс көрсетпей-ақ қолдана аламыз.

Модификаторларға және инкапсуляцияға қол жеткізу

Java класының барлық мүшелері - өрістер мен әдістер - қол жетімділік модификаторлары бар. Алдыңғы тақырыптарда біз жалпы

модификаторға тап болдық. Қатынас модификаторлары класс мүшелері үшін жарамды ауқымды, яғни берілген айнымалыны немесе әдісті қолдануға болатын контекстi орнатуға мүмкіндік береді.

Java келесі қатынасу модификаторларын қолданады:

public: қоғамдық, қоғамдық класс немесе класс мүшесі. Жалпы модификатормен бірге жарияланған өрістер мен әдістер басқа кластарға ағымдағы бумадан және сыртқы бумалардан көрінеді.

private: жалпы модификаторға қарағанда жеке класс немесе класс мүшесі. Жеке сыныпқа немесе сыныптың мүшесіне сол сыныптағы кодтан ғана қол жетімді.

protected: мұндай класқа немесе класс мүшесіне басқа класта болса да, ағымдағы кластың немесе буманың кез келген жерінен немесе туынды кластардан кіруге болады.

Әдепкі модификатор. Өрістің немесе кластың әдісінің модификаторының болмауы оған әдепкі модификатордың қолданылуын білдіреді. Мұндай өрістер немесе әдістер ағымдағы пакеттегі барлық сыныптарға көрінеді.

Мысал ретінде келесі бағдарламаны пайдаланып, кіру модификаторларын қарастырайық:

```
1  public class Program{
2
3     public static void main(String[] args) {
4
5         Person kate = new Person("Kate", 32, "Baker Street", "+12334567");
6         kate.displayName();     // норм, метод public
7         kate.displayAge();      // норм, методимеетмодификаторпоумолчанию
8         kate.displayPhone();    // норм, метод protected
9         //kate.displayAddress();  // ! Ошибка, метод private
10
11        System.out.println(kate.name);      // норм, модификаторпоумолчанию
12        System.out.println(kate.address);   // норм, модификатор public
13        System.out.println(kate.age);       // норм, модификатор protected
14        //System.out.println(kate.phone);   // ! Ошибка, модификатор private
15    }
16}
17class Person{
18
19    String name;
20    protected int age;
21    public String address;
22    private String phone;
23
```

```
24  public Person(String name, int age, String address, String phone){
25      this.name = name;
26      this.age = age;
27      this.address = address;
28      this.phone = phone;
29  }
30  public void displayName(){
31      System.out.printf("Name: %s \n", name);
32  }
33  void displayAge(){
34      System.out.printf("Age: %d \n", age);
35  }
36  private void displayAddress(){
37      System.out.printf("Address: %s \n", address);
38  }
39  protected void displayPhone(){
40      System.out.printf("Phone: %s \n", phone);
41  }}
```

Бұл жағдайда екі класс бір пакетте орналасқан - әдепкі бума, сондықтан Program класында біз әдепкі модификаторы бар, ашық және қорғалған Person класының барлық әдістері мен айнымалыларын қолдана аламыз. Бағдарлама класындағы жеке модификаторы бар өрістер мен әдістер қол жетімді болмайды.

Егер Program класы басқа бумада орналасса, онда оған жалпы модификаторы бар өрістер мен әдістер ғана қол жетімді болар еді.

Қатынас модификаторы айнымалының немесе әдіс анықтамасының қалған бөлігінің алдында тұруы керек.

Инкапсуляция

Неліктен барлық айнымалылар мен әдістерді пакетке немесе сыныпқа қарамастан, бағдарламаның кез келген жерінде қол жетімді болатындай етіп жалпы модификатормен жарияламасқа? Мысалы, жасты көрсететін жас өрісін алайық. Егер басқа класс осы өріске тікелей қол жеткізе алса, онда оған бағдарлама жұмысы кезінде қате мән, мысалы, теріс санның берілу мүмкіндігі бар. Деректерді осылай өзгерту құптарлық емес. Немесе біз консольға басып шығарылуы үшін немесе оның мәнін білу үшін кейбір деректердің тікелей қол жетімді болуын қалаймыз. Осыған байланысты, сырттан қажетсіз қол жетімділіктен қорғау үшін (мән алу үшін де, өзгерту үшін де) деректерге қол жеткізуді мүмкіндігінше шектеу ұсынылады. Әр түрлі модификаторларды қолдану деректердің бүлінбеуіне немесе дұрыс өзгертілмеуіне кепілдік береді. Деректерді белгілі бір көлемде осылай жасыру инкапсуляция деп аталады.

Осылайша, өрістерді тікелей қолданудың орнына, қатынау әдістері қолданылады. Мысалға:

```
1  public class Program{
2
3      public static void main(String[] args) {
4
5          Person kate = new Person("Kate", 30);
6          System.out.println(kate.getAge());      // 30
7          kate.setAge(33);
8          System.out.println(kate.getAge());      // 33
9          kate.setAge(123450);
10         System.out.println(kate.getAge());      // 33
11     }
12}
13class Person{
14
15     private String name;
16     private int age;
17
18     public Person(String name, int age){
19         this.name = name;
20         this.age = age;
21     }
22     public String getName(){
23         return this.name;
24     }
25     public void setName(String name){
26         this.name = name;
27     }
28     public int getAge(){
29         return this.age;
30     }
31     public void setAge(int age){
32         if(age > 0 && age < 110)
33             this.age = age;
34     }
35}
```

Содан кейін Person класындағы атау және жас өрістерімен тікелей жұмыс істеудің орнына біз осы өрістердің мәндерін орнататын және қайтаратын әдістермен жұмыс істейтін боламыз. SetName, setAge және тағы басқалар әдістері мьютейтер (mutator) деп аталады, өйткені олар өріс мәндерін өзгертеді. Ал getName, getAge және сол сияқты әдістер қол

жеткізушілер деп аталады, өйткені олардың көмегімен өрістің мәні алынады.

Сонымен қатар, бұл әдістерге қосымша логика қосуға болады. Мысалы, бұл жағдайда, жас өзгерген кезде, жаңа мәннің қолайлы аралыққа қалай сәйкес келетіндігін тексеру жүргізіледі.

Мұрагерлік

Объектіге бағытталған бағдарламалаудың негізгі аспектілерінің бірі мұрагерлік болып табылады. Мұрагерліктің көмегімен сіз жаңа функциялар қосу немесе ескіні өзгерту арқылы бұрыннан бар кластардың функционалдығын кеңейте аласыз. Мысалы, жеке адамды сипаттайтын келесі Person класы бар:

```
1  classPerson {
2
3     privateString name;
4     publicString getName(){ returnname; }
5
6     publicPerson(String name){
7
8        this.name=name;
9     }
10
11    publicvoiddisplay(){
12
13       System.out.println("Name: "+ name);
14    }
15}
```

Мүмкін кейінірек біз кәсіпорынның қызметкерін сипаттайтын тағы бір класты – Employee класын қосқымыз келеді. Бұл класс Person класымен бірдей функционалдылықты жүзеге асыратын болғандықтан, қызметкер де адам болғандықтан, Employee класын Person класының туындысы (мұрагерлік, кіші класс) ету орынды болар еді, ол өз кезегінде базалық класс, ата-ана немесе суперкласс деп аталады:

```
1 class Employee extends Person{
2 }
```

Бір класты екіншіден мұрагерлік деп жариялау үшін мұрагерлік кластың атауынан кейін негізгі кластың атауынан кейін extends кілт сөзін қолдану керек. Employee класы үшін Person негізі болып табылады, демек, Employee класы Person класындағы барлық бірдей өрістер мен әдістерді мұрагер етеді.

Кластарды қолдану:

```
1  public class Program{
2
3      public static void main(String[] args) {
4
5          Person tom = new Person("Tom");
6          tom.display();
7          Employee sam = new Employee("Sam");
8          sam.display();
9      }
10}
11class Person {
12
13    private String name;
14    public String getName(){ return name; }
15
16    public Person(String name){
17
18        this.name=name;
19    }
20
21    public void display(){
22
23        System.out.println("Name: " + name);
24    }
25}
26class Employee extends Person{
27
28}
```

Туынды класс жеке модификатормен анықталғаннан басқа, базалық кластың барлық әдістері мен өрістеріне (базалық класс басқа бумада болса да) қол жеткізе алады. Бұл жағдайда туынды класс өзінің өрістері мен әдістерін қоса алады:

```
1  public class Program{
2
3      public static void main(String[] args) {
4
5          Employee sam = new Employee("Sam", "Microsoft");
6          sam.display();  // Sam
7          sam.work();     // Sam works in Microsoft
8      }
9  }
10class Person {
```

```
11
12    private String name;
13    public String getName(){ return name; }
14
15    public Person(String name){
16
17        this.name=name;
18    }
19
20    public void display(){
21
22        System.out.println("Name: " + name);
23    }
24}
25class Employee extends Person{
26
27    private String company;
28
29    public Employee(String name, String company) {
30
31        super(name);
32        this.company=company;
33    }
34    public void work(){
35        System.out.printf("%s works in %s \n", getName(), company);
36    }
37}
```

Бұл жағдайда Employee класы жұмысшының жұмыс орнын, сонымен қатар work әдісін сақтайтын компания өрісін қосады.

Егер конструкторлар базалық класта анықталса, онда super кілт сөзін пайдаланып, туынды кластың конструкторында базалық класс конструкторларының бірін шақыру керек. Мысалы, Person класында бір параметр қабылдайтын конструктор бар. Сондықтан, Employee сыныбында Person сыныбының конструкторы конструкторда шақырылуы керек. Жақшаның ішіндегі super сөзінен кейін келтірілетін аргументтер тізімі келтірілген. Осылайша, қызметкердің атын орнату базалық класс конструкторына беріледі.

Бұл жағдайда, негізгі кластың конструкторына шақыру туынды кластың конструкторында ең басында жүруі керек.

Анықтау әдістері

Туынды класс өзінің әдістерін анықтай алады немесе базалық кластан қалған әдістерді жоққа шығара алады. Мысалы, Employee класындағы display әдісін жоққа шығарыңыз:

```
1  public class Program{
2
3     public static void main(String[] args) {
4
5        Employee sam = new Employee("Sam", "Microsoft");
6        sam.display();  // Sam
7                        // Works in Microsoft
8     }
9  }
10class Person {
11
12   private String name;
13   public String getName(){ return name; }
14
15   public Person(String name){
16
17      this.name=name;
18   }
19
20   public void display(){
21
22      System.out.println("Name: " + name);
23   }
24}
25class Employee extends Person{
26
27   private String company;
28
29   public Employee(String name, String company) {
30
31      super(name);
32      this.company=company;
33   }
34   @Override
35   public void display(){
36
37      System.out.printf("Name: %s \n", getName());
38      System.out.printf("Works in %s \n", company);
39   }
40}
```

@Override аннотациясы қайта анықталған әдіске дейін көрсетілген. Бұл аннотация негізінен міндетті емес.

Әдісті жоққа шығарған кезде оның қол жетімділік деңгейі базалық класта қол жетімділік деңгейінен кем болмауы керек. Мысалы, егер әдіс базалық сыныпта public модификаторға ие болса, онда туынды сыныптағы әдіс public модификаторға да ие болуы керек.

Алайда, бұл жағдайда, Employee display бөлігі базалық класты display әдісінен әрекеттерді қайталайтынын көре аламыз. Сондықтан біз Employee класын қысқартуға болады:

```
1  class Employee extends Person{
2
3     private String company;
4
5     public Employee(String name, String company) {
6
7        super(name);
8        this.company=company;
9     }
10    @Override
11    public void display(){
12
13       super.display();
14       System.out.printf("Works in %s \n", company);
15    }
16}
```

super кілт сөзімен біз базалық класс әдістерін жүзеге асыруға сілтеме жасай аламыз.

Мұрагерлікке тыйым салу

Мұрагерлік өте қызықты және тиімді механизм болғанымен, кейбір жағдайларда оны қолдану жағымсыз болуы мүмкін. Бұл жағдайда сіз мұрагерлікті final кілт сөзді қолдана отырып өшіре аласыз. Мысалға:

```
1 public final class Person {
2 }
```

Егер Person класы осылай анықталған болса, онда келесі код қате болып, жұмыс істемейді, өйткені біз мұрагерлікке тыйым салдық:

```
1 class Employee extends Person{ {
2 }
```

Мұраға тыйым салудан басқа, сіз жеке әдістерді жоққа шығаруға да тыйым сала аласыз. Мысалы, жоғарыдағы мысалда display () әдісі қайта анықталған, біз оны жоққа шығаруға тыйым саламыз:

```
1 public class Person {
2
3    //.....................
```

```
4
5    public final void display(){
6
7        System.out.println("Аты: " + name);
8    }
9 }
```

Бұл жағдайда Employee класы дисплей әдісін жоққа шығара алмайды.

Әдістердің динамикалық диспетчері

Мұрагерлік және әдістерді жоққа шығару мүмкіндігі бізге үлкен мүмкіндіктер ашады. Біріншіден, суперкласс айнымалысына ішкі класс объектісіне сілтеме жібере аламыз:

```
1 Person sam = new Employee("Sam", "Oracle");
```

Employee Адамнан мұра алатын болғандықтан, Employee объектісі де жеке тұлға болып табылады. Шамамен айтқанда, кәсіпорынның кез-келген қызметкері де адам болып табылады.

Алайда, айнымалы Person объектісін білдірсе де, виртуалды машина оның шынымен Employee объектісіне нұсқайтынын көреді. Сондықтан, осы объектіге әдістерді шақырған кезде, Person-та емес, Employee класында анықталған әдістің нұсқасы шақырылады. Мысалға:

```
1  publicclassProgram{
2
3     publicstaticvoidmain(String[] args) {
4
5        Person tom = newPerson("Tom");
6        tom.display();
7        Person sam = newEmployee("Sam", "Oracle");
8        sam.display();
9     }
10}
11classPerson {
12
13    privateString name;
14
15    publicString getName() { returnname; }
16
17    publicPerson(String name){
18
19       this.name=name;
20    }
```

```
21
22    publicvoiddisplay(){
23
24        System.out.printf("Person %s \n", name);
25    }
26}
27
28classEmployee extendsPerson{
29
30    privateString company;
31
32    publicEmployee(String name, String company) {
33
34        super(name);
35        this.company = company;
36    }
37    @Override
38    publicvoiddisplay(){
39
40 System.out.printf("Employee %s works in %s \n", super.getName(),
41company);
42    }
   }
```

Осы бағдарламаның шығыс консолі:

```
Person Tom
Employee Sam works in Oracle
```

Қайтарылған әдіс шақырылған кезде, виртуалды машина динамикалық түрде ішкі класста анықталған әдістің нақты нұсқасын табады және шақырады. Бұл процесті dynamic method lookup немесе динамикалық әдіс іздеу немесе динамикалық әдісті диспетчерлеу деп те атайды.

2.4. Файлдармен жұмыс

Енгізу/шығару ағындары

Көптеген бағдарламалау тілдерінің айрықша ерекшелігі - файлдармен және ағындармен жұмыс. Java-да негізгі ағындық функциялар java.io бумасынан кластарға шоғырланған.

Мұндағы негізгі түсінік - ағын. Бағдарламалаудағы «ағын» тұжырымдамасы шамадан тыс жүктелген және әр түрлі ұғымдарды білдіруі мүмкін. Бұл жағдайда файлдармен және енгізу-шығарумен жұмыс істеуге қатысты біз ағынды деректерді оқуға немесе жазуға қолданылатын

абстракция ретінде айтамыз (файлдар, розеткалар, консольдік мәтін және т.б.).

Ағын нақты физикалық құрылғыға Java I / O жүйесін пайдаланып қосылады. Бізде файлмен байланысты және ол арқылы файлды оқи немесе жаза алатын ағын болуы мүмкін. Бұл сондай-ақ желі арқылы деректерді қабылдау немесе жіберу үшін пайдаланылатын желілік розеткамен байланысты ағын болуы мүмкін. Барлық осы тапсырмалар: әртүрлі файлдарды оқу және жазу, желі арқылы ақпарат алмасу, консольдегі енгізу-шығару, біз Java-да ағындарды қолдана отырып шешеміз.

Деректерді оқуға болатын объектіні кіріс ағыны, ал мәліметтер жазуға болатын объектіні шығыс ағын деп атайды. Мысалы, егер сізге файлдың мазмұнын оқу қажет болса, онда кіріс ағыны, ал егер сізге файлға жазу керек болса, онда шығыс ағыны қолданылады.

Байт ағындарын басқаратын барлық кластардың негізінде екі абстрактілі класс жатыр: InputStream (кіріс ағындарын білдіреді) және OutputStream (шығыс ағындарын бейнелейді).

Бірақ байттармен жұмыс жасау онша ыңғайлы болмағандықтан, символдық ағынмен жұмыс істеу үшін **Reader**(символдар ағындарын оқуға) және **Writer** (кейіпкерлер ағындарын жазуға) кластары қосылды.

Ағындармен жұмыс жасайтын барлық басқа кластар осы дерексіз кластардың мұрагерлері болып табылады. Ағындардың негізгі кластары (Сурет 2.15) :

InputStream	OutputStream	Reader	Writer
FileInputStream	File OutputStream	FileReader	FileWriter
BufferedInputStream	BufferedOutputStream	BufferedReader	BufferedWriter
ByteArrayInputStream	ByteArrayOutputStream	CharArrayReader	CharArrayWriter
FilterInputStream	FilterOutputStream	FilterReader	FilterWriter
DataInputStream	DataOutputStream		
ObjectInputStream	ObjectOutputStream		

Сурет 2.15 – Ағындардың негізгі кластары

Байт ағындары

***InputStream** класы*

InputStream класы - бұл байт ағындарын басқаратын барлық сыныптар үшін негізгі класс. Оның негізгі әдістерін қарастырайық:

- ✓ int available (): ағымда оқуға болатын байт санын қайтарады
- ✓ void close (): ағынды жабады
- ✓ int read (): Ағымдағы келесі байттың бүтін көрінісін береді.

Ағынды оқуға қол жетімді байт болмаған кезде, бұл әдіс -1 санын қайтарады

✓ int read (байт [] буфер): ағыннан буферлік массивке байт оқиды. Оқылғаннан кейін оқылған байт санын қайтарады. Егер ешқандай байт оқылмаса, онда -1 саны қайтарылады

✓ int read (байт [] буфер, int offset, int length): ағыннан буферлік массивке дейінгі ұзындыққа тең байттардың санын оқиды. Бұл жағдайда оқылған байттар массивке офсеттік ығысудан басталады, яғни [буын] элементінің буферінен басталады. Әдіс сәтті оқылған байт санын қайтарады.

✓ long skip(long number): санға тең болатын оқудағы байттардың белгілі бір мөлшерін өткізіп жібереді

OutputStream *класы*

OutputStream класы - бұл екілік жазу ағындарымен жұмыс жасайтын барлық кластар үшін негізгі класс. Ол өзінің функционалдығын келесі әдістер арқылы жүзеге асырады:

✓ void close (): ағынды жабады

✓ void flush (): барлық мазмұнды жазу арқылы шығыс буферін жуады

✓ void write (int b): b бүтін параметрімен ұсынылған шығыс ағынына бір байт жазады

✓ void write (байт [] буфер): байт буферінің жиымын шығыс ағынға жазады.

✓ void write (байт [] буфер, int offset, int length): шығыс ағынына массив буферінен ұзындыққа тең байттардың санын офсеттік ығысудан бастайды, яғни [буын] элементінің буферінен бастайды.

Reader және Writer абстрактты кластары

Абстрактты Reader класы мәтіндік ақпаратты оқудың функционалдығын қамтамасыз етеді. Оның негізгі әдістерін қарастырайық:

✓ absract void close (): кіріс ағынын жабады

✓ int read (): ағындағы келесі таңбаның бүтін көрінісін береді. Егер мұндай таңбалар болмаса және файлдың соңына жеткен болса, -1 саны қайтарылады

✓ int read (char [] буфері): ағыннан буферлік массивке таңбаларды оқиды, олардың саны буферлік массивтің ұзындығына тең. Сәтті оқылған таңбалар санын қайтарады. Файлдың соңына жеткенде -1 қайтарады

✓ int read (CharBuffer буфері): ағыннан CharBuffer нысанына символдарды оқиды. Сәтті оқылған таңбалар санын қайтарады. Файлдың соңына жеткенде -1 мәнін қайтарады

✓ absract int read (char [] буфер, int ofset, int count): ағыннан бастап таңбаларды буферге жылжытудан бастайды

✓ long skip(long count): санауға тең таңбалар санын өткізіп жібереді. Сәтті өткізіп жіберілген таңбалардың санын қайтарады

Writer класы барлық символдық шығыс ағындарының функционалдығын анықтайды. Оның негізгі әдістері:

- ✓ Writer append (char c): шығыс ағынның соңына c қосады. Writer нысанын қайтарады
- ✓ Writer append (CharSequence chars): символдар таңбасын шығыс ағынының соңына қосады. Writer нысанын қайтарады
- ✓ дерексіз void close (): ағынды жабады
- ✓ абсолютті бос орын flush (): ағын буферін жуады
- ✓ void write (int c): бүтін кескіні бар ағынға бір таңба жазады
- ✓ void write (char [] буфері): ағынға символдар жиымын жазады
- ✓ absract void write (char [] буфер, int off, int len): буферлік массивтен ағынға бірнеше таңбаларды ғана жазады. Сонымен қатар, таңбалардың саны len, ал массивтен таңбаларды таңдау өшіру индексінен басталады
- ✓ void write (String str): ағынға жол жазады
- ✓ void write (String str, int off, int len): ленге тең болатын жолдан ағынға бірнеше таңбалар жазады, ал жолдан таңбаларды таңдау индекстен басталады.

Reader және Writer кластары сипаттаған функционалдылық тікелей символдар ағынының сыныптарына, атап айтқанда мәтіндік файлдармен жұмыс жасауға арналған FileReader және FileWriter кластарына беріледі.

Файлдарды оқу және жазу. FileInputStream және FileOutputStream

Файлдар мен FileOutputStream класын жазу

FileOutputStream класы файлға байт жазуға арналған. Ол OutputStream класынан шыққан, сондықтан ол барлық функционалдығын алады.

FileOutputStream класының конструкторы жазылатын файлды анықтайды. Класс бірнеше конструкторларды қолдайды:

FileOutputStream(String filePath)
FileOutputStream(File fileObj)
FileOutputStream(String filePath, boolean
append)
FileOutputStream(File fileObj, boolean append)

Файл жол жолы арқылы немесе File объектісі арқылы көрсетіледі. Екінші параметр - append әдісін көрсетеді: егер ол true болса, онда мәліметтер файл соңына қосылады, ал false болса - файл толығымен жазылады

Мысалы, файлды жолға жазайық:

```
1  importjava.io.*;
2
3  publicclassProgram {
```

```
4
5     publicstaticvoidmain(String[] args) {
6
7         String text = "Hello world!"; // жазуға арналған жол
8   try(FileOutputStream fos=newFileOutputStream("C://SomeDir//notes.txt"))
9         {
10            // жолды байтқа аудару
11            byte[] buffer = text.getBytes();
12
13            fos.write(buffer, 0, buffer.length);
14        }
15        catch(IOException ex){
16
17            System.out.println(ex.getMessage());
18        }
19        System.out.println("The file has been written");
20    }
21}
```

FileOutputStream нысанын құру үшін параметр ретінде жазу үшін файлға жол ашатын конструкторды қолданасыз. Егер мұндай файл болмаса, онда ол автоматты түрде жазу кезінде жасалады. Біз мұнда жол жазып отырғандықтан, оны алдымен байт массивіне аудару керек. Ал write әдісі арқылы жол файлға жазылады.

Файлды автоматты түрде жабу және ресурсты босату үшін FileOutputStream объектісі try ... catch конструкциясы арқылы құрылады.

Алайда, байт массивін толығымен жазу қажет емес. Write () әдісінің шамадан тыс жүктелуін қолдана отырып, сіз бір байт жаза аласыз:

```
1 fos.write(buffer[0]); // запись первого байта
```

Файлдарды және FileInputStream класын оқу

Файлдан деректерді оқу үшін, InputStream класынан мұра болатын FileInputStream класы арналған, сондықтан оның барлық әдістерін орындайды.

FileInputStream объектісін құру үшін біз бірқатар конструкторларды қолдана аламыз. Конструктордың ең көп қолданылатын нұсқасы параметр ретінде оқылатын файлға жол ашады:

```
1 FileInputStream(String fileName) throwsFileNotFoundException
```

Егер файлды ашу мүмкін болмаса, мысалы, көрсетілген жолда мұндай файл жоқ болса, онда FileNotFoundException қиыс жағдайы жіберіледі.

Бұрын жазылған файлдағы деректерді оқып, консольға шығарамыз:

```
1  importjava.io.*;
2
3  publicclassProgram {
4
5     publicstaticvoidmain(String[] args) {
6
7        try(FileInputStream fin=newFileInputStream("C://SomeDir//notes.txt"))
8        {
9           System.out.printf("File size: %d bytes \n", fin.available());
10
11          inti=-1;
12          while((i=fin.read())!=-1){
13
14             System.out.print((char)i);
15          }
16       }
17       catch(IOException ex){
18
19          System.out.println(ex.getMessage());
20       }
21    }
22}
```

Бұл жағдайда біз әрбір жеке байтты i айнымалысына оқимыз:

```
1 while((i=fin.read())!=-1){
```

Ағымда оқылатын мәліметтер болмаған кезде әдіс -1 мәнін қайтарады.

Содан кейін әрбір оқылған байт char объектісіне айналады және консольға басылады.

Дәл сол сияқты, сіз байтты массивке оқып, содан кейін оларды басқара аласыз:

```
1 byte[] buffer = newbyte[fin.available()];
2 // считаемфайлвбуфер
3 fin.read(buffer, 0, fin.available());
4
5 System.out.println("File data:");
6 for(inti=0; i<buffer.length;i++){
7
8    System.out.print((char)buffer[i]);
9 }
```

Екі сыныпты біріктіріп, бірінен оқуды және екінші файлға жазуды орындайық:

```
   importjava.io.*;
1
2
3  publicclassProgram {
4
5     publicstaticvoidmain(String[] args) {
6
7   try(FileInputStream fin=newFileInputStream("C://SomeDir//notes.txt");
8   FileOutputStream
9  fos=newFileOutputStream("C://SomeDir//notes_new.txt"))
10        {
11           byte[] buffer = newbyte[fin.available()];
12           // буферге оқимыз
13           fin.read(buffer, 0, buffer.length);
14           // буферден файлға жазамыз
15           fos.write(buffer, 0, buffer.length);
16        }
17        catch(IOException ex){
18
19           System.out.println(ex.getMessage());
20        }
21    }
   }
```

FileInputStream және FileOutputStream кластары ең алдымен екілік файлдарды жазуға, яғни байттарды жазуға және оқуға арналған. Оларды мәтіндік файлдармен жұмыс істеуге пайдалануға болатындығына қарамастан, бұл тапсырма үшін басқа сыныптар қолайлы.

Мәтіндік файлдарды оқу және жазу

Бұрын талқыланған кластар файлдарға мәтін жазу үшін қолданыла алатын болса да, олар бірінші кезекте мәліметтердің екілік ағындарымен жұмыс істеуге арналған және олардың мәтіндік файлдармен толыққанды жұмыс істеу мүмкіндіктері жеткіліксіз. Осы мақсатта Reader және Writer дерексіз кластарының мұрагерлері болып табылатын мүлдем басқа кластар қолданылады.

Файлдарды жазу. FileWriter класы

FileWriter класы Writer класынан алынған. Ол мәтіндік файлдарды жазу үшін қолданылады.

FileWriter объектісін құру үшін келесі конструкторлардың бірін пайдалануға болады:

```
1 FileWriter(File file)
2 FileWriter(File file, booleanappend)
3 FileWriter(FileDescriptor fd)
```

4 FileWriter(String fileName)
5 FileWriter(String fileName, booleanappend)

Мысалы, файлға жол ретінде жол немесе белгілі бір мәтіндік файлға сілтеме жасайтын File объектісі конструкторға беріледі. append параметрі файлдың соңына деректердің қосылуын (егер параметр true болса) немесе файлдың үстіне жазылу керек екенін анықтайды.

Файлға бірнеше мәтін жазайық:

```
1  importjava.io.*;
2
3  publicclassProgram {
4
5     publicstaticvoidmain(String[] args) {
6
7        try(FileWriter writer = newFileWriter("notes3.txt", false))
8        {
9           // барлық жолды жазу
10          String text = "Hello Gold!";
11          writer.write(text);
12          // символ бойынша жазу
13          writer.append('\n');
14          writer.append('E');
15
16          writer.flush();
17       }
18       catch(IOException ex){
19
20          System.out.println(ex.getMessage());
21       }
22    }
23}
```

Конструктор қосымша мәнінің параметрін false мәнімен қолданды - яғни файл қайта жазылады. Содан кейін мәліметтер негізгі Writer класында анықталған әдістердің көмегімен жазылады.

Файлдарды оқу. FileReader класы

FileReader класы абстрактты Reader класынан мұраға қалады және мәтіндік файлдарды оқудың функционалдығын қамтамасыз етеді.

FileReader объектісін құру үшін оның конструкторларының бірін пайдалануға болады:

1 FileReader(String fileName)
2 FileReader(File file)
3 FileReader(FileDescriptor fd)

Reader базалық класында анықталған әдістерді қолданып, файлды оқыңыз:

```
importjava.io.*;

publicclassProgram {

    publicstaticvoidmain(String[] args) {

        try(FileReader reader = newFileReader("notes3.txt"))
        {
            // символ бойынша оқимыз
            intc;
            while((c=reader.read())!=-1){

                System.out.print((char)c);
            }
        }
        catch(IOException ex){

            System.out.println(ex.getMessage());
        }
    }
}
```

Сондай-ақ, таңбалар массивінен аралық буферге оқуға болады:

```
import java.io.*;
import java.util.Arrays;

public class Program {

    public static void main(String[] args) {

        try(FileReader reader = new
FileReader("notes3.txt"))
        {
            char[] buf = new char[256];
            int c;
            while((c = reader.read(buf))>0){

                if(c < 256){
                    buf = Arrays.copyOf(buf, c);
                }
```

```
18              System.out.print(buf);
19          }
20      }
21      catch(IOException ex){
22
23          System.out.println(ex.getMessage());
24      }
25  }
   }
```

Бұл жағдайда файлдың соңына жеткенше файлды 256 символдан тұратын массивке тізбектей оқимыз, бұл жағдайда оқу әдісі -1 санын қайтарады.

Файлдың оқылған бөлігі 256 символдан аз болуы мүмкін болғандықтан (мысалы, файлда тек 73 символ бар), ал егер оқылған мәліметтердің саны буфер өлшемінен (256) аз болса, онда біз массивті Arrays.copy әдісімен көшіреміз. Яғни, біз buf массивін кесіп тастаймыз, оған тек файлдан оқылған таңбаларды қалдырамыз.

2.5 Жолдармен жұмыс

Жолдарға кіріспе. String класы

Жол - бұл символдар тізбегі. Жолдармен жұмыс істеу үшін Java жолдармен жұмыс істеудің бірқатар әдістерін ұсынатын String класын анықтайды. Физикалық тұрғыдан String объектісі - бұл жадтағы таңбалар орналасқан аймаққа сілтеме.

Жаңа жол құру үшін біз String класс конструкторларының бірін қолдана аламыз немесе жолды қос тырнақшаға тікелей тағайындай аламыз:

```
1  public static void main(String[] args) {
2
3      String str1 = "Java";
4      String str2 = new String(); // бос жол
5      String str3 = new String(new char[] {'h', 'e', 'l', 'l', 'o'});
6      String str4 = new String(new char[]{'w', 'e', 'l', 'c', 'o', 'm', 'e'}, 3, 4);//3 -
   бастапқыиндекс, 4 –символ саны
7
8      System.out.println(str1); // Java
9      System.out.println(str2); //
10     System.out.println(str3); // hello
11     System.out.println(str4); // come
12}
```

Жолдармен жұмыс істегенде, String объектісі өзгермейтінін түсіну керек(immutable). Яғни, осы сызықты өзгертетін кез-келген амалдар жаңа жолды жасайды.

Жол таңбалар жиынтығы ретінде қарастырылатындықтан, жолдың ұзындығын немесе таңбалар жиынтығының ұзындығын табу үшін length () әдісін қолдана аламыз:

```
String str1 = "Java";
System.out.println(str1.length()); // 4
```

ToCharArray () әдісін қолдана отырып, сіз жолды қайтадан символдар массивіне түрлендіре аласыз:

```
String str1 = new String(new char[] {'h', 'e', 'l', 'l', 'o'});
char[] helloArray = str1.toCharArray();
```

Жол бос болуы мүмкін. Ол үшін сіз оған бос тырнақша тағайындай аласыз немесе жолдан барлық таңбаларды алып тастай аласыз:

```
String s = "";   // строка не указывает на объект
if(s.length() == 0) System.out.println("String is empty");
```

Бұл жағдайда length () әдісімен қайтарылған жолдың ұзындығы 0-ге тең болады.

String класында жолдың бос екендігін тексеруге мүмкіндік беретін арнайы әдіс бар - isEmpty (). Егер жол бос болса, ол true қайтарады:

```
String s = "";
if(s.length() == 0) System.out.println("String is empty");
```

String айнымалысы кез-келген нысанды көрсетпеуі және null болуы мүмкін:

```
String s = null;
if(s == null) System.out.println("String is null");
```

Null бос жолға тең емес. Мысалы, келесі жағдайда біз орындау қателігіне тап боламыз:

```
String s = null;   // жол объектіге көрсетпейді
if(s.length()==0) System.out.println("String is empty");    // ! қате
```

Айнымалы кез келген String объектісін көрсетпегендіктен, біз сәйкесінше String объектісінің әдістерін шақыра алмаймыз. Мұндай қателерді болдырмау үшін жолды null екенін алдын ала тексеруге болады:

```
1 String s = null;   // жол объектіге көрсетпейді
2if(s!=null && s.length()==0) System.out.println("String is empty");
```

String класының негізгі әдістері

Негізгі жолдық амалдар String класының әдістері арқылы анықталады, олардың арасында келесілерді ажыратуға болады:

- ✓ concat(): тізбектер;

✓ valueOf (): нысанды жолға түрлендіреді;
✓ join(): бөлгіш берілген жолдарды біріктіреді;
✓ compare(): екі жолды салыстырады;
✓ charAt(): индекстегі жолдың таңбасын қайтарады;
✓ getChars(): символдар тобын қайтарады;
✓ equals(): жолдарды регистрмен салыстырады;
✓ equalsIgnoreCase(): жолдарды сезімталдығымен салыстырады;
✓ regionMatches(): жолдардағы ішкі жолдарды салыстырады;
✓ indexOf(): жолдағы ішкі жолдың бірінші пайда болу индексін табады;
✓ lastIndexOf(): жолдағы ішкі жолдың соңғы пайда болу индексін табады;
✓ startsWith(): жолдың ішкі жолдан басталатынын анықтайды;
✓ endsWith(): жолдың белгілі бір ішкі жолмен аяқталатынын анықтайды;
✓ replace(): жолдағы бір ішкі жолды басқасымен ауыстырады;
✓ trim(): жетекші және артқы кеңістікті жояды;
✓ substring(): ішкі тізбекті нақты индекстен бастап соңына дейін немесе белгілі бір индекске дейін қайтарады;
✓ toLowerCase(): жолдағы барлық таңбаларды кіші әріптерге түрлендіреді;
✓ toUpperCase (): жолдағы барлық таңбаларды бас әріпке түрлендіреді.

Осы әдістердің қалай жұмыс істейтінін талдап көрейік.

Жолдармен негізгі операциялар

Жолдарды жалғастыру

Жолдарды біріктіру үшін қосу операциясын қолдануға болады («+»):

```
1 String str1 = "Java";
2 String str2 = "Hello";
3 String str3 = str1 + " " + str2;
4
5 System.out.println(str3); // Hello Java
```

Сонымен қатар, егер жолды емес объект жолдарды қосу кезінде пайдаланылса, мысалы, сан болса, онда бұл нысан жолға айналады:

```
String str3 = "Год " + 2015;
```

Іс жүзінде жолды емес объектілерге жолдар қосқанда, String класының valueOf () әдісі шақырылады. Бұл әдіс көптеген жүктемелерге ие және барлық дерлік типтерді жолдарға түрлендіреді. Әр түрлі кластардағы объектілерді түрлендіру үшін valueOf әдісі осы кластардың toString () әдісін шақырады.

Жолдарды біріктірудің тағы бір тәсілі - бұл concat () әдісі:

```
1 String str1 = "Java";
```

```
2 String str2 = "Hello";
3 str2 = str2.concat(str1); // HelloJava
```

Concat () әдісі шақырушы жолды біріктіру үшін жолды алады және тізбектелген жолды қайтарады.

Біріктірудің тағы бір әдісі, join() әдісі жолды бөлгіш негізінде біріктіруге мүмкіндік береді. Мысалы, жоғарыдағы екі жол «HelloJava» деген бір сөзге біріктірілген, бірақ ең дұрысы біз екі астардың босорынмен бөлінгенін қалаймыз. Бұл үшін біз join () әдісін қолданамыз:

```
1 String str1 = "Java";
2 String str2 = "Hello";
3 String str3 = String.join(" ", str2, str1); // Hello Java
```

Таңбалар мен жолдарды шығару

Таңбаларды индекс бойынша алу үшін, String класы char charAt (int index) әдісін анықтайды. Ол таңбаларды шығаратын индексті алады және шығарылған таңбаны қайтарады:

```
1 String str = "Java";
2 char c = str.charAt(2);
3 System.out.println(c); // v
```

Массивтер сияқты индекстеу нөлден басталады.

Егер сізге символдар тобын немесе ішкі жолды бірден алу қажет болса, getChars (intsrcBegin, intsrcEnd, char [] dst, intdstBegin) әдісін қолдануға болады. Ол келесі параметрлерді қабылдайды:

- ✓ srcBegin: символдарды шығаруды бастайтын жолдағы индекс;
- ✓ srcEnd: символдар шығарылатын жолдағы индекс;
- ✓ dst: таңбалар алынатын символдар массиві;
- ✓ dstBegin: жолдан алынған символдарды қосатын dst массивіндегі индекс.

```
1 String str = "Hello world!";
2 intstart = 6;
3 intend = 11;
4 char[] dst=newchar[end - start];
5 str.getChars(start, end, dst, 0);
6 System.out.println(dst); // world
```

Жолдарды салыстыру

Жолдарды салыстыру үшін equals() (регистрге сезімтал) және equalsIgnoreCase () (регистрге сезімтал емес) әдістері қолданылады. Екі әдіс жолды салыстыру үшін параметр ретінде қабылдайды:

```
1 String str1 = "Hello";
```

```
2 String str2 = "hello";
3
4 System.out.println(str1.equals(str2)); // false
5 System.out.println(str1.equalsIgnoreCase(str2)); // true
```

Мәліметтердің сандық және басқа қарабайыр түрлерін салыстырудан айырмашылығы, жолдарда == тең белгісі қолданылмайды. Оның орнына equals () әдісін қолдану керек.

Тағы бір арнайы әдіс, regionMatches() екі жолдың ішіндегі жеке жолдарды салыстырады. Оның келесі формалары бар:

```
1 booleanregionMatches(inttoffset, String other, intoofset, intlen)
  booleanregionMatches(booleanignoreCase, inttoffset, String other, intoofset,
2 intlen)
```

Әдіс келесі параметрлерді алады:

- ✓ ignoreCase: салыстыру жағдайын елемеу керек пе. Егер true болса, регистр еленбейді.
- ✓ toffset: салыстыруды бастайтын шақыру жолындағы бастапқы индекс;
- ✓ other: шақырылған жолмен салыстырылатын жол;
- ✓ oofset: салыстыруды бастайтын салыстырылған жолдағы бастапқы индекс;
- ✓ len: екі жолда да салыстыруға болатын таңбалар саны.

Біз әдісті қолданамыз:

```
1 String str1 = "Hello world";
2 String str2 = "I work";
3 booleanresult = str1.regionMatches(6, str2, 2, 3);
4 System.out.println(result); // true
```

Бұл жағдайда әдіс бірінші жолдың 6-шы индексінен 3 таңбаны («wor») және екінші жолдың 2-индексінен 3 таңбаны («wor») салыстырады. Бұл ішкі тізбектер бірдей болғандықтан, true қайтарылады.

Int ComparTo (String str) және int ComparToIgnoreCase (String str) әдістерінің тағы бір жұбы екі жолды салыстыруға мүмкіндік береді, бірақ олар сізге бір жолдың екіншісінен көп екенін немесе болмайтынын білуге мүмкіндік береді. Егер қайтару мәні 0-ден үлкен болса, онда бірінші жол екіншіден, егер нөлден аз болса, онда, керісінше, екіншісі біріншіден үлкен болады. Егер жолдар тең болса, онда 0 қайтарылады.

Лексикографиялық тапсырыс бір жолды екінші жолға қарағанда көп немесе аз анықтау үшін қолданылады. Яғни, мысалы, «A» жолы «B» жолынан аз, өйткені алфавиттегі «A» таңбасы «B» таңбасынан бұрын келеді. Егер жолдардың алғашқы таңбалары тең болса, онда келесі таңбалар ескеріледі. Мысалға:

```
1 String str1 = "hello";
```

```
2 String str2 = "world";
3 String str3 = "hell";
4
5 System.out.println(str1.compareTo(str2)); // -15 - str1 меньшечем strt2
6 System.out.println(str1.compareTo(str3)); // 1 - str1 большечем str3
```

Жолдан іздеу

IndexOf () әдісі жолдағы субстриннің бірінші пайда болу индексін, ал lastIndexOf () әдісі соңғы пайда болу индексін табады. Егер ішкі жол табылмаса, онда екі әдіс те -1 қайтарады:

```
1 String str = "Hello world";
2 intindex1 = str.indexOf('l'); // 2
3 intindex2 = str.indexOf("wo"); //6
4 intindex3 = str.lastIndexOf('l'); //9
```

startsWith() әдісі жолдың белгілі бір ішкі жолдан басталатынын анықтауға мүмкіндік береді, және endWith () әдісі жолдың белгілі бір ішкі жолмен аяқталуын анықтауға мүмкіндік береді:

```
1 String str = "myfile.exe";
2 booleanstart = str.startsWith("my"); //true
3 booleanend = str.endsWith("exe"); //true
```

Жолға ауыстыру

Replace() әдісі жолдағы символдардың бір тізбегін екіншісіне ауыстыруға мүмкіндік береді:

```
1 String str = "Hello world";
2 String replStr1 = str.replace('l', 'd'); // Heddo wordd
3 String replStr2 = str.replace("Hello", "Bye"); // Bye world
```

Жолды кесу

Trim () әдісі алдыңғы және артқы кеңістікті жоюға мүмкіндік береді:

```
String str = "  hello world  ";
str = str.trim(); // hello world
```

Substring () әдісі ішкі жолды нақты индекстен соңына немесе нақты индексіне қайтарады:

```
String str = "Hello world";
String substr1 = str.substring(6); // world
String substr2 = str.substring(3,5); //lo
```

Регистрді өзгерту

ToLowerCase () әдісі жолдағы барлық символдарды кіші әріпке, ал toUpperCase () әдісі үлкен әріптерге түрлендіреді:

```
1 String str = "Hello World";
2 System.out.println(str.toLowerCase()); // hello world
3 System.out.println(str.toUpperCase()); // HELLO WORLD
```

Split

Split () әдісі жолды белгілі бір бөлгіште ішкі жолдарға бөлуге мүмкіндік береді. Бөлгіш - кез-келген символ немесе символдар жиынтығы әдіске параметр ретінде беріледі. Мысалы, мәтінді бөлек сөздерге бөлейік:

```
1 String text = "FIFA will never regret it";
2 String[] words = text.split(" ");
3 for(String word : words){
4     System.out.println(word);
5 }
```

Бұл жағдайда жол бос орынмен бөлінеді. Консоль шығысы:

```
FIFA
will
never
regret
it
```

StringBuffer және StringBuilder

String объектілері өзгермейді, сондықтан жолдарды өзгертетін кез-келген операциялар іс жүзінде жаңа жолға әкеледі, бұл қолданбаның жұмысына әсер етеді. Бұл мәселені шешу үшін Java-ға StringBuffer және StringBuilder кластары қосылды, бұл жолдармен жұмыс жасауды тиімді етеді. Олар мәні бойынша өнімділікті жоғалтпастан өзгертілетін кеңейтілетін жол тәрізді.

Бұл кластар ұқсас, қайталанатын дерлік, олардың конструкторлары бірдей, әдістері бірдей, оларды дәл осылай қолданады. Жалғыз айырмашылық - бұл StringBuffer класы синхрондалған және қауіпсіз. Яғни, StringBuffer класы көп ағынды қосымшаларда қолдануға ыңғайлы, мұнда осы кластың объектісі әр түрлі ағындарда өзгеруі мүмкін. Егер біз көп ағынды қосымшалар туралы айтпасақ, онда StringBuilder класын қолданған дұрыс, ол қауіпсіз емес, сонымен бірге бір ағынды қосымшаларда StringBuffer-ге қарағанда жылдамырақ жұмыс істейді.

StringBuffer төрт конструкторды анықтайды:

1 StringBuffer()
2 StringBuffer(intcapacity)
3 StringBuffer(String str)
4 StringBuffer(CharSequence chars)

StringBuilder ұқсас конструкторларды анықтайды:

1 StringBuilder()
2 StringBuilder(int capacity)
3 StringBuilder(String str)
4 StringBuilder(CharSequence chars)

Мысал ретінде StringBuffer функционалдығын пайдаланып, осы кластардың қалай жұмыс істейтінін қарастырайық.

Жолдардағы барлық операциялар үшін StringBuffer / StringBuilder бөлінген жадыны қайта бөледі. Жадыны жиі бөлуді болдырмау үшін, StringBuffer / StringBuilder жадының бірнеше қолданылу аймағын алдын-ала сақтайды. Параметрсіз конструктор жадыдағы 16 таңбаға арналған орынды сақтайды. Егер біз символдар саны әртүрлі болғанын қаласақ, онда символдар санын параметр ретінде қабылдайтын екінші конструкторды қолдануға болады.

Екі кластың үшінші және төртінші конструкторлары жолды және символдар жиынтығын қабылдайды, ал қосымша 16 таңбаға жадыны сақтайды.

Capacity () әдісімен біз жад сақталған символдар санын ала аламыз. Таңба буферінің минималды сыйымдылығын өзгерту үшін sureCapacity () әдісін қолданыңыз:

```
1 String str = "Java";
2 StringBuffer strBuffer = newStringBuffer(str);
3 System.out.println("Емкость: "+ strBuffer.capacity()); // 20
4 strBuffer.ensureCapacity(32);
5 System.out.println("Емкость: "+ strBuffer.capacity()); // 42
6 System.out.println("Длина: "+ strBuffer.length()); // 4
```

StringBuffer басында «Java» жолымен инициализацияланғандықтан, оның сыйымдылығы 4 + 16 = 20 таңбаны құрайды. Содан кейін strBuffer.ensureCapacity (32) шақыру арқылы буферлік сыйымдылықты арттырамыз және ең төменгі буферлік сыйымдылықты 32 таңбаға дейін арттырамыз. Алайда, соңғы сыйымдылық жоғарыға қарай өзгеруі мүмкін.

Сонымен, бұл жағдайда мен сыйымдылықты 32 емес, 32 + 4 = 36 емес, 42 таңба аламын. Шындығында, тиімділікті арттыру үшін Java қосымша жад бөле алады.

Бірақ кез-келген жағдайда, сыйымдылығына қарамай, StringBuffer-де length () әдісі бойынша алуға болатын жолдың ұзындығы өзгеріссіз қалады - 4 таңба (өйткені «Java» -да 4 таңба бар).

StringBuffer-де сақталатын жолды алу үшін біз toString () стандартты әдісін қолдана аламыз:

```
1 String str = "Java";
2 StringBuffer strBuffer = newStringBuffer(str);
3 System.out.println(strBuffer.toString()); // Java
```

StringBuffer және StringBuilder барлық операцияларында String класына ұқсайды.

Символдарды алу және орнату

CharAt () әдісі алады, ал setCharAt () әдісі символды белгілі бір индекске қояды:

```
1 StringBuffer strBuffer = newStringBuffer("Java");
2 charc = strBuffer.charAt(0); // J
3 System.out.println(c);
4 strBuffer.setCharAt(0, 'c');
5 System.out.println(strBuffer.toString()); // cava
```

GetChars () әдісі нақты индекстер арасындағы таңбалар жиынын алады:

```
1 StringBuffer strBuffer = new StringBuffer("world");
2 int startIndex = 1;
3 int endIndex = 4;
4 char[] buffer = new char[endIndex-startIndex];
5 strBuffer.getChars(startIndex, endIndex, buffer, 0);
6 System.out.println(buffer); // orl
```

Жолға қосу

Append () әдісі StringBuffer соңына ішкі жол қосады:

```
1 StringBuffer strBuffer = newStringBuffer("hello");
2 strBuffer.append(" world");
3 System.out.println(strBuffer.toString()); // hello world
```

Insert () әдісі StringBuffer-ге нақты индекстегі жолды немесе символды қосады:

```
1 StringBuffer strBuffer = new StringBuffer("word");
2
3 strBuffer.insert(3, 'l');
4 System.out.println(strBuffer.toString()); //world
5
6 strBuffer.insert(0, "s");
7 System.out.println(strBuffer.toString()); //sworld
```

Таңбаларды жою

Delete () әдісі белгілі бір индекстегі белгілі бір позицияға қатысты барлық таңбаларды жояды, ал deleteCharAt () әдісі белгілі бір индекстегі бір таңбаны жояды:

```
1 StringBuffer strBuffer = newStringBuffer("assembler");
2 strBuffer.delete(0,2);
3 System.out.println(strBuffer.toString()); //sembler
4
5 strBuffer.deleteCharAt(6);
6 System.out.println(strBuffer.toString()); //semble
```

Жолды кесу

Substring () әдісі жолды нақты индекстен соңына дейін немесе белгілі бір индекске дейін қысқартады:

```
1 StringBuffer strBuffer = newStringBuffer("hello java!");
2 String str1 = strBuffer.substring(6); // обрезка строки с 6 символа до конца
3 System.out.println(str1); //java!
4
5 String str2 = strBuffer.substring(3, 9); // обрезка строки с 3 по 9 символ
6 System.out.println(str2); //lo jav
```

Ұзындықтың өзгеруі

SetLength () әдісі StringBuffer ұзындығын өзгерту үшін қолданылады (символ буферінің сыйымдылығы емес). Егер StringBuffer ұлғаятын болса, оның тізбегі соңында бос таңбалармен толтырылады, егер ол азаятын болса, онда жол негізінен қысқартылады:

```
1 StringBuffer strBuffer = newStringBuffer("hello");
2 strBuffer.setLength(10);
3 System.out.println(strBuffer.toString()); //"hello     "
4
5 strBuffer.setLength(4);
6 System.out.println(strBuffer.toString()); //"hell"
```

Жолға ауыстыру

StringBuffer ішіндегі белгілі бір позициялар арасындағы ішкі тізбекті басқа ішкі жолмен ауыстыру үшін, ауыстыру () әдісін қолданыңыз:

```
1 StringBuffer strBuffer = newStringBuffer("hello world!");
2 strBuffer.replace(6,11,"java");
3 System.out.println(strBuffer.toString()); //hello java!
```

replace әдісінің бірінші параметрі ауыстыруды неден бастайтынын, екінші параметр - қандай позицияға дейін болатындығын, ал үшінші параметр ауыстырылатын ішкі жолды анықтайды.

Кері жол тәртібі

Reverse () әдісі StringBuffer ішіндегі тәртіпті өзгертеді:

```
1 StringBuffer strBuffer = newStringBuffer("assembler");
2 strBuffer.reverse();
3 System.out.println(strBuffer.toString()); //relbmessa
```

Тұрақты өрнектер

Тұрақты өрнектер - жолдарды өңдеудің қуатты құралы. Тұрақты өрнектер жолдың немесе ішкі жолдың сәйкес келуі керек үлгіні анықтауға мүмкіндік береді.

String класының бірнеше әдістері тұрақты өрнектерді қабылдайды және оларды жолдарда амалдар орындау үшін қолданады.

split

split() әдісі жолды ішкі жолдарға бөлу үшін қолданылады. Ол параметр ретінде жолды бөлудің критерийін білдіретін тұрақты өрнекті қабылдай алады.

Мысалы, сөйлемді сөздерге бөлейік:

```
1 String text = "FIFA will never regret it";
2 String[] words = text.split("\\s*(\\s|,|!|\\.)\\s*");
3 for(String word : words){
4    System.out.println(word);
5 }
```

Бөлу үшін «\\ s * (\\ s |, |! | \\.) \\ s *» тұрақты өрнегі қолданылады. «\\ s» кіші өрнегі пробелді білдіреді. Жұлдызша таңбаның 0-ден шексіз рет пайда бола алатынын көрсетеді. Яғни, біз жұлдызшаны қосамыз және біз анықталмаған қатардағы бос орындарды аламыз - «\\ s *» (яғни сөздер арасында қанша бос орын бар екендігі маңызды емес). Сонымен қатар, орын мүлдем болмауы мүмкін. Жақша ішіндегі өрнектер тобын көрсетеді,

олардың арасынан бос орын саны қойылуы мүмкін. Топ тік жолақ арқылы мәндер жиынтығын анықтауға мүмкіндік береді, ал ішкі жол сол мәндердің біріне сәйкес келуі керек. Яғни, «\\ s |, |! | \\» тобында. ішкі жол бос орынға, үтірге, леп белгісіне немесе нүктеге сәйкес келуі мүмкін. Сонымен қатар, нүкте арнайы реттілікті емес, нүктелік белгіні білдіретіндігімізді білдіретіндіктен, нүктенің алдына қиғаш сызықтар қоямыз.

Жолдардың сәйкестігі. matches

String класының тағы бір әдісі matches() сәйкес келеді, тұрақты өрнекті қабылдайды және егер жол осы өрнекке сәйкес келсе, true мәнін қайтарады. Басқа жағдайда false мәнін қайтарады.

Мысалы, жолдың телефон нөміріне сәйкес келетіндігін тексерейік:

```
1 String input = "+12343454556";
2 boolean result = input.matches("(\\+*)\\d{11}");
3 if(result){
4     System.out.println("It is a phone number");
5 }
6 else{
7     System.out.println("It is not a phone number!");
8 }
```

Бұл жағдайда тұрақты өрнек алдымен «(\\ + *)» тобын анықтайды. Яғни, басында қосу белгісі болуы мүмкін, бірақ ол жоқ болуы мүмкін. Әрі қарай, келесі 11 таңба сандарға сәйкес келетіндігін көреміз. «\\ d» өрнегі сандық таңбаны білдіреді, ал фигуралы жақшалардағы сан {11} - бұл символ түрін қанша рет қайталау керек. Яғни, біз қосу белгісі басында пайда болатын жолды іздейміз (немесе ол жоқ болуы мүмкін), содан кейін 11 сандық таңба бар.

Pattern класы

Java-дағы тұрақты өрнектермен жұмыс істеудің көп бөлігі java.util.regex бумасында шоғырланған.

Тұрақты өрнектің өзі жолдан сәйкестік табуға арналған үлгіні білдіреді. Java осыған ұқсас үлгіні көрсету және берілген өрнекке сәйкес жолдан ішкі жолдарды іздеу үшін Pattern және Matcher кластарын анықтайды.

Қарапайым сәйкестендіру үшін Pattern класы статикалық boolean matches(String pattern, CharSequence input) әдісін анықтайды. Егер енгізілген таңбалар тізбегі жол үлгісіне дәл сәйкес келсе, бұл әдіс шындықты қайтарады:

```
1  import java.util.regex.Pattern;
2
3  public class StringsApp {
4
```

```
5     public static void main(String[] args) {
6
7         String input = "Hello";
8         boolean found = Pattern.matches("Hello", input);
9         if(found)
10            System.out.println("Найдено");
11        else
12            System.out.println("Ненайдено");
13    }
14}
```

Бірақ, әдетте, сәйкестікті табудың тағы бір әдісі қолданылады - Matcher класын қолдану.

Matcher класы

Matcher класының негізгі әдістерін қарастырайық:

✓ boolean matches(): егер жолдың барлығы үлгіге сәйкес келсе, true мәнін қайтарады;

✓ boolean find(): егер жолда өрнекке сәйкес келетін ішкі жол болса және осы жолға өтсе, true мәнін қайтарады;

✓ String group(): табу әдісін шақыру нәтижесінде үлгіге сәйкес ішкі жолды қайтарады. Егер сәйкестік болмаса, әдіс IllegalStateException ерекше жағдайын жасайды.

✓ int start(): ағымдағы сәйкестіктің индексін қайтарады;

✓ int end(): ағымдағы сәйкестіктен кейінгі келесі сәйкестіктің индексін қайтарады;

✓ String replaceAll (String str): барлық табылған сәйкестіктерді ішкі жолмен ауыстырады және өзгертілген жолды ауыстырумен қайтарады.

Matcher класын қолданайық. Ол үшін алдымен статикалық compile() әдісін қолданып Pattern объектісін құру керек, ол өрнекті орнатуға мүмкіндік береді:

```
1 Pattern pattern = Pattern.compile("Hello");
```

Шаблон ретінде «Hello» жолы қолданылады. Compile () әдісі Pattern нысанын қайтарады, оны біз өз бағдарламамызда қолдана аламыз.

Pattern класы Matcher нысанын іздеуге және қайтаруға параметр ретінде жолды қабылдайтын matcher (String input) әдісін анықтайды:

```
1 String input = "Hello world! Hello Java!";
2 Pattern pattern = Pattern.compile("hello");
3 Matcher matcher = pattern.matcher(input);
```

Содан кейін Matcher объектісінде matches() әдісі мәтіннен үлгіге сәйкестікті табу үшін шақырылады:

```
1  importjava.util.regex.Matcher;
2  importjava.util.regex.Pattern;
3
4  publicclassStringsApp {
5
6     publicstaticvoidmain(String[] args) {
7
8        String input = "Hello";
9        Pattern pattern = Pattern.compile("Hello");
10       Matcher matcher = pattern.matcher(input);
11       booleanfound = matcher.matches();
12       if(found)
13          System.out.println("Найдено");
14       else
15          System.out.println("Ненайдено");
16    }
17}
```

Толық сәйкестікті емес, жолдағы жеке сәйкестіктерді табумен неғұрлым функционалды мысал қарастырайық:

```
1  importjava.util.regex.Matcher;
2  importjava.util.regex.Pattern;
3
4  publicclassStringsApp {
5
6     publicstaticvoidmain(String[] args) {
7
8        String input = "Hello Java! Hello JavaScript! JavaSE 8.";
9        Pattern pattern = Pattern.compile("Java(\\w*)");
10       Matcher matcher = pattern.matcher(input);
11       while(matcher.find())
12          System.out.println(matcher.group());
13    }
14}
```

Біз Java сөзінің барлық кездесетіндерін жолдан тапқымыз келеді делік. Бастапқы жолда үш сөз бар: «Java», «JavaScript» және «JavaSE». Ол үшін біз «Java (\\ w *)» үлгісін қолданамыз. Бұл үлгі тұрақты өрнек синтаксисін қолданады. Басындағы «Java» сөзі жолдағы барлық сәйкестіктер Java-дан басталуы керек дейді. (\\ w *) өрнегі «Java» -дан кейін әріптік-сандық белгілердің кез-келген санына сәйкес келетіндігін

білдіреді. \ W өрнегі әріптік-цифрлық белгіні білдіреді, ал өрнектен кейінгі жұлдызша олардың белгісіз санын көрсетеді - бір, екі, үш немесе мүлдем болмауы мүмкін. Java \ w сияқты шығып кету реті ретінде қарастырмауы үшін, мысалы, өрнек басқа қиғаш сызықпен жазылады.

Әрі қарай,Matcher класының find () әдісі қолданылады, бұл жолдағы келесі сәйкестікке өтуге мүмкіндік береді. Яғни, бұл әдіске бірінші қоңырау жолдағы бірінші сәйкестікті табады, екінші қоңырау екінші сәйкестікті табады және т.б. Яғни while (matcher.find ()) циклін қолданып, біз барлық сәйкестіктерден өте аламыз. Біз сәйкестікті matcher.group () әдісі арқылы ала аламыз. Нәтижесінде бағдарлама келесі нәтиже береді:

```
Java
JavaScript
JavaSE
```

Жолға ауыстыру

Енді replaceAll () әдісі бойынша барлық сәйкестікті ауыстырайық:

```
1 String input = "Hello Java! Hello JavaScript! JavaSE 8.";
2 Pattern pattern = Pattern.compile("Java(\\w*)");
3 Matcher matcher = pattern.matcher(input);
4 String newStr = matcher.replaceAll("HTML");
5 System.out.println(newStr); // Hello HTML! Hello HTML! HTML 8.
```

Сонымен қатар String класында ұқсас әрекеті бар replaceAll () әдісі бар екенін ескеру қажет:

```
1 String input = "Hello Java! Hello JavaScript! JavaSE 8.";
2 String myStr =input.replaceAll("Java(\\w*)", "HTML");
3 System.out.println(myStr); // Hello HTML! Hello HTML! HTML 8.
```

Жолды лексемдерге бөлу

Pattern класының String[] split(CharSequence input) әдісін қолдана отырып, сіз жолды белгілі бір бөлгіште ішкі жолдар массивіне бөле аласыз.

Мысалы, біз жолдан жеке сөздерді шығарғымыз келеді:

```
1  importjava.util.regex.Pattern;
2
3  publicclassStringsApp {
4
5     publicstaticvoidmain(String[] args) {
6
7        String input = "Hello Java! Hello JavaScript! JavaSE 8.";
8        Pattern pattern = Pattern.compile("[ ,.!?]");
```

```
9         String[] words = pattern.split(input);
10        for(String word:words)
11            System.out.println(word);
12    }
13}
```

Консоль бірнеше сөздерді басып шығарады:

```
Hello
Java

Hello
JavaScript

JavaSE
8
```

Бұл барлық бөлгіш таңбаларды жояды. Алайда, бұл бұзылу әдісі өте қолайлы емес: бізде әлі де кейбір кеңістіктер бар, олар бөлгіш емес, жетон ретінде қарастырылады. Неғұрлым дәл және күрделі бұзылу үшін біз тұрақты экспрессия элементтерін қолдануымыз керек. Сонымен, шаблонды келесімен ауыстырайық:

```
1 Pattern pattern = Pattern.compile("\\s*(\\s|,|!|\\.)\\s*");
```

Енді бізде тек қана сөздер болады:

```
Hello
Java
Hello
JavaScript
JavaSE
8
```

2.6 Лямбда өрнектері

Java тіліне JDK 8 шығарылымымен енгізілген жаңалықтардың ішінде лямбда өрнектері ерекше. Лямбда - бұл жеке айнымалыға бөлініп, содан кейін бағдарламаның әр түрлі жерлерінде бірнеше рет шақырылатын нұсқаулар жиынтығы.

Лямбда өрнегінің негізі -> стрелкасын білдіретін лямбда операторы. Бұл оператор лямбда өрнегін екі бөлікке бөледі: сол жақта өрнектің параметрлер тізімі, ал оң жақ бөліктің өзі барлық әрекеттер орындалатын лямбда өрнегінің денесін білдіреді.

Лямбда өрнегі өздігінен орындалмайды, бірақ функционалды интерфейсте анықталған әдісті жүзеге асырады. Функционалды интерфейсте іске асырусыз бір ғана әдіс болуы керек.

Мысалды қарастырайық:

```
1  public class LambdaApp {
2
       public static void main(String[] args) {
4
5          Operationable operation;
6          operation = (x,y)->x+y;
7
8          int result = operation.calculate(10, 20);
9          System.out.println(result); //30
10     }
11}
12interface Operationable{
13    int calculate(int x, int y);
14}
```

Operationable интерфейс - бұл орындалатын интерфейс, ол іске асырусыз бір әдісті - **calculate** әдісін анықтайды. Бұл әдіс екі параметрді - бүтін сандарды алады және кейбір бүтін санды қайтарады.

Шын мәнінде, лямбда өрнектері бұрын Java-да қолданылған ішкі анонимді кластар үшін стенография болып табылады. Атап айтқанда, алдыңғы мысалды келесідей қайта жазуға болады:

```
1  public class LambdaApp {
2
3     public static void main(String[] args) {
4
5         Operationable op = new Operationable(){
6
7             public int calculate(int x, int y){
8
9                 return x + y;
10            }
11        };
12         int z = op.calculate(20, 10);
13         System.out.println(z); // 30
14    }
15}
16interface Operationable{
17    int calculate(int x, int y);
```

18}

Лямбда өрнегін жариялау және қолдану үшін негізгі бағдарлама бірнеше қадамдарға бөлінеді:

1. Функционалды интерфейске сілтемені анықтау:

```
1 Operationable operation;
```

2. Лямбда өрнегін құру:

```
1 operation = (x,y)->x+y;
```

3. Лямбда өрнегін интерфейс әдісіне шақыру ретінде пайдалану:

```
1 int result = operation.calculate(10, 20);
```

Параметрді қосу операциясы лямбда өрнегінде анықталғандықтан, әдіс нәтижесі 10 және 20 сандарының қосындысы болады.

Сонымен, бір функционалды интерфейс үшін біз көптеген лямбда өрнектерін анықтай аламыз.

Мысалға:

```
1 Operationable operation1 = (int x, int y)-> x + y;
2 Operationable operation2 = (int x, int y)-> x - y;
3 Operationable operation3 = (int x, int y)-> x * y;
4
5 System.out.println(operation1.calculate(20, 10)); //30
6 System.out.println(operation2.calculate(20, 10)); //10
7 System.out.println(operation3.calculate(20, 10)); //200
```

Кешіктіріп орындау

Лямбда қолданудағы маңызды сәттердің бірі - бұл кейінге қалдыру. Яғни, біз бағдарламаның бір жерінде лямбда өрнегін анықтаймыз, содан кейін оны бағдарламаның әртүрлі бөліктерінде белгісіз рет деп атай аламыз. Кейінге қалдырылған орындау қажет болуы мүмкін, мысалы, келесі жағдайларда:

Кодты бөлек ағынмен орындау

Бір кодты бірнеше рет орындау

Кейбір оқиғалар нәтижесінде кодтың орындалуы

Кодты шынымен қажет болғанда және қажет болған жағдайда ғана орындау

Параметрлерді Lambda өрнегіне беру

Ламбда өрнегінің параметрлері функционалды интерфейстегі әдіс параметрлерімен сәйкес келуі керек. Лямбда өрнегін жазған кезде параметрлер типін жазудың қажеті жоқ, бірақ негізінен мұны істеуге болады, мысалы:

```
1 operation = (int x, int y)->x+y;
```

Егер әдіс ешқандай параметр қабылдамаса, онда бос жақша жазылады, мысалы:

```
1 ()-> 30 + 20;
```

Егер әдіс тек бір параметрді алса, онда жақшаны алып тастауға болады:

```
1 n-> n * n;
```

Терминал лямбда өрнектері

Жоғарыда біз белгілі бір мәнді беретін лямбда өрнектерін қарастырдық. Сонымен қатар, кез-келген мәнді қайтармайтын терминалды лямбда болуы мүмкін. Мысалға:

```
1  interface Printable{
2      void print(String s);
3  }
4
5  public class LambdaApp {
6
7      public static void main(String[] args) {
8
9          Printable printer = s->System.out.println(s);
10         printer.print("Hello Java!");
11     }
12}
```

Лямбда және жергілікті айнымалылар

Лямбда өрнегі сырттан жарияланатын айнымалыларды неғұрлым жалпы көлемде - лямбда өрнегі анықталған сынып немесе әдіс деңгейінде қолдана алады. Алайда, айнымалылар қалай және қай жерде анықталғанына байланысты, оларды лямбдада қолдану тәсілі әр түрлі болуы мүмкін. Бірінші мысалды қарастырайық - сынып деңгейіндегі айнымалыларды қолдану:

```
1  public class LambdaApp {
2
3      static int x = 10;
4      static int y = 20;
5      public static void main(String[] args) {
6
7          Operation op = ()->{
8
```

```
9           x=30;
10          return x+y;
11      };
12      System.out.println(op.calculate()); // 50
13      System.out.println(x); // 30 - значение x изменилось
14  }
15}
16interface Operation{
17  int calculate();
18}
```

X және y айнымалылары класс деңгейінде жарияланады, ал лямбда өрнегінде біз оларды ала аламыз, тіпті оларды өзгерте аламыз. Сонымен, бұл жағдайда өрнек орындалғаннан кейін, x айнымалысының мәні өзгереді.

Енді тағы бір мысалды қарастырайық - әдіс деңгейіндегі жергілікті айнымалылар:

```
1  public static void main(String[] args) {
2
3      int n=70;
4      int m=30;
5      Operation op = ()->{
6
7          //n=100; - так нельзя сделать
8          return m+n;
9      };
10     // n=100;  - так тоже нельзя
11     System.out.println(op.calculate()); // 100
12}
```

Біз лямбдада әдіс деңгейіндегі жергілікті айнымалыларды қолдана аламыз, бірақ олардың мәнін өзгерте алмаймыз. Егер біз мұны істеуге тырысатын болсақ, онда даму ортасы (Netbeans) бізге қате көрсете алады және мұндай айнымалыны соңғы кілт сөзімен белгілеу керек, яғни тұрақты ету керек: final int n = 70;. Алайда, бұл қажет емес.

Сонымен қатар, біз осы өрнектен тыс лямбда өрнегінде қолданылатын айнымалының мәнін өзгерте алмаймыз. Яғни, мұндай айнымалы тұрақты деп жарияланбаса да, іс жүзінде ол тұрақты болып табылады.

Лямбда өрнектеріндегі код блоктары

Лямбда өрнектерінің екі түрі бар: бір жолды өрнек және код блогы. Бір жолды өрнектердің мысалдары жоғарыда көрсетілген. Блоктық өрнектер бұйра жақшалармен қоршалған. Блоктық лямбда өрнектерінде if,

switch, ішкі кіріктірілген блоктарды, циклдарды қолдануға болады, егер операторлар болса, операторларды ауыстырады, айнымалылар жасайды және т.б. Егер блоктық лямбда өрнегі мәнді қайтаруы керек болса, онда return операторы нақты қолданылады:

```
1  Operationable operation = (int x, int y)-> {
2
3     if(y==0)
4        return 0;
5     else
6        return x/y;
7  };
8
9  System.out.println(operation.calculate(20, 10)); //2
10System.out.println(operation.calculate(20, 0)); //0
```

Жалпы функционалды интерфейс

Функционалды интерфейс жалпы болуы мүмкін, бірақ лямбда өрнегінде жалпылауға жол берілмейді. Бұл жағдайда біз интерфейс нысанын белгілі бір типпен теруіміз керек, ол кейін лямбда өрнегінде қолданылады. Мысалға:

```
1  public class LambdaApp {
2
3     public static void main(String[] args) {
4
5        Operationable<Integer> operation1 = (x, y)-> x + y;
6        Operationable<String> operation2 = (x, y) -> x + y;
7
8        System.out.println(operation1.calculate(20, 10)); //30
9        System.out.println(operation2.calculate("20", "10")); //2010
10    }
11}
12interface Operationable<T>{
13    T calculate(T x, T y);
14}
```

Осылайша, лямбда өрнегі жарияланған кезде, параметрлер қандай типті ұсынатынын және қай типке оралатынын біледі.

Лямбдалар параметрлер мен әдіс нәтижелері ретінде
Лямбдалар әдіс параметрлері ретінде

Java-дағы ламбдалардың артықшылықтарының бірі - оларды әдістерге параметрлер ретінде беруге болады. Мысалды қарастырайық:

```
1  public class LambdaApp {
2
3      public static void main(String[] args) {
4
5          Expression func = (n)-> n%2==0;
6          int[] nums = { 1, 2, 3, 4, 5, 6, 7, 8, 9 };
7          System.out.println(sum(nums, func)); // 20
8      }
9      private static int sum (int[] numbers, Expression func)
10     {
11         int result = 0;
12         for(int i : numbers)
13         {
14             if (func.isEqual(i))
15                 result += i;
16         }
17         return result;
18     }
19}
20
21interface Expression{
22     boolean isEqual(int n);
23}
```

Expression функционалды интерфейсі isEqual () әдісін анықтайды, егер n санында теңдік болса, true болады.

Бағдарламаның негізгі класы белгілі бір шартқа сәйкес келетін барлық массив элементтерінің қосындысын есептейтін **sum ()** әдісін анықтайды. Ал шарттың өзі Expression func параметрі арқылы беріледі. Сонымен, **sum** әдісін жазу кезінде біз қандай шарт қолданылатынын білмеуіміз мүмкін. Шарттың өзі лямбда өрнегі ретінде анықталады:

```
1 Expression func = (n)-> n%2==0;
```

Яғни, бұл жағдайда барлық сандар жұп болуы керек немесе олардың 2-ге бөлінуінің қалдықтары 0-ге тең болуы керек. Содан кейін бұл лямбда өрнегі **sum** әдісіне шақыруға беріледі.

Бұл жағдайда сіз интерфейстің айнымалысын анықтай алмайсыз, бірақ бірден лямбда өрнегін әдіске жібере аласыз:

```
1 int[] nums = { 1, 2, 3, 4, 5, 6, 7, 8, 9 };
2 int x = sum(nums, (n)-> n > 5); // сумма чисел, которые больше 5
3 System.out.println(x);  // 30
```

Әдістің сілтемелері әдіс параметрлері ретінде

Java-да JDK 8-ден бастап, әдіске параметр ретінде басқа әдіске сілтеме жіберуге болады. Негізінде бұл әдіс лямбда өрнегін әдіске өткізуге ұқсас.

Әдіс сілтемесі *класс_аты::статикалық_әдіс_атауы* (егер әдіс статикалық болса) немесе *класс_объектісі::әдіс_атауы*(егер әдіс статикалық емес болса) түрінде беріледі. Мысал алайық:

```
1  // функциональныйинтерфейс
2  interface Expression{
3      boolean isEqual(int n);
4  }
5  // класс, в котором определены методы
6  classExpressionHelper{
7
8      static boolean isEven(int n){
9
10         return n%2 == 0;
11     }
12
13     static boolean isPositive(int n){
14
15         return n > 0;
16     }
17}
18public class LambdaApp {
19
20     public static void main(String[] args) {
21
22         int[] nums = { -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5};
23         System.out.println(sum(nums, ExpressionHelper::isEven));
24
25         Expression expr = ExpressionHelper::isPositive;
26         System.out.println(sum(nums, expr));
27     }
28
29     private static int sum (int[] numbers, Expression func)
30     {
```

```
31        int result = 0;
32        for(int i : numbers)
33        {
34            if (func.isEqual(i))
35                result += i;
36        }
37        return result;
38    }
39}
```

Ол сонымен қатар бір әдісі бар Expression функционалды интерфейсін анықтайды. Сонымен қатар, екі статикалық әдісті қамтитын ExpressionHelper класы анықталған. Негізінде оларды бағдарламаның негізгі сыныбында анықтауға болатын еді, бірақ мен оларды бөлек сыныпқа бөлдім.

LambdaApp бағдарламасының негізгі класы қандай да бір шартқа сәйкес келетін жиым элементтерінің қосындысын қайтаратын sum () әдісін анықтайды. Шарт Expression функционалды интерфейс нысаны ретінде беріледі.

main әдісінде біз sum әдісін екі рет атаймыз, сандардың бір массивінде өтетін, бірақ шарттары әр түрлі. sum әдісін алғашқы шақыру:

```
1 System.out.println(sum(nums, ExpressionHelper::isEven));
```

Екінші параметрдің орнына ExpressionHelper :: isEven беріледі, яғни ExpressionHelper класының статикалық isEven () әдісіне сілтеме жасалады. Бұл жағдайда сілтеме жасалған әдістер параметрлер бойынша сәйкес келуі және функционалды интерфейс әдісімен нәтиже беруі керек.

sum әдісін екінші рет шақыру Expression объектісін жасайды, содан кейін әдіске өтеді:

```
1 Expression expr = ExpressionHelper::isPositive;
2 System.out.println(sum(nums, expr));
```

Параметр ретінде әдіс сілтемелерін қолдану лямбда өрнектерін қолдануға ұқсас.

Егер бізге статикалық емес әдістерді шақыру керек болса, онда сілтемеде класс атауының орнына осы класс объектісінің аты қолданылады:

```
1  interface Expression{
2      boolean isEqual(int n);
3  }
4
5  class ExpressionHelper{
6
```

```
7     boolean isEven(int n){
8
9         return n%2 == 0;
10    }
11}
12public class LambdaApp {
13
14    public static void main(String[] args) {
15
16        int[] nums = { -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5};
17        ExpressionHelper exprHelper = new ExpressionHelper();
18        System.out.println(sum(nums, exprHelper::isEven)); // 0
19    }
20
21    private static int sum (int[] numbers, Expression func)
22    {
23        int result = 0;
24        for(int i : numbers)
25        {
26            if (func.isEqual(i))
27                result += i;
28        }
29        return result;
30    }
31}
```

Конструкторларға сілтеме

Сол сияқты біз де конструкторларды қолдана аламыз: **класс_атауы::new**. Мысалға:

```
1  public class LambdaApp {
2
3      public static void main(String[] args) {
4
5          UserBuilder userBuilder = User::new;
6          User user = userBuilder.create("Tom");
7          System.out.println(user.getName());
8      }
9  }
10interface UserBuilder{
11    User create(String name);
12}
13
```

```
14class User{
15
16    private String name;
17    String getName(){
18        return name;
19    }
20
21    User(String n){
22        this.name=n;
23    }
24}
```

Конструкторларды қолдану кезінде функционалды интерфейстердің әдістері класс конструкторларымен бірдей параметрлер тізімін қабылдауы керек және сол кластың объектісін қайтаруы керек.

Лямбдалар әдістердің нәтижесі ретінде

Java-дағы әдіс лямбда өрнегін қайтара алады. Келесі мысалды қарастырайық:

```
1  interface Operation{
2      int execute(int x, int y);
3  }
4
5  public class LambdaApp {
6
7      public static void main(String[] args) {
8
9          Operation func = action(1);
10         int a = func.execute(6, 5);
11         System.out.println(a);         // 11
12
13         int b = action(2).execute(8, 2);
14         System.out.println(b);         // 6
15     }
16
17     private static Operation action(int number){
18         switch(number){
19             case 1: return (x, y) -> x + y;
20             case 2: return (x, y) -> x - y;
21             case 3: return (x, y) -> x * y;
22             default: return (x,y) -> 0;
23         }
24     }
25}
```

Бұл жағдайда **Operation** функционалды интерфейсі анықталады, онда **execute** әдісі int типінің екі мәнін қабылдайды және int типінің мәнін береді.

action әдісі параметр ретінде санды қабылдайды және оның мәніне байланысты сол немесе басқа лямбда өрнегін қайтарады. Ол қосуды, азайтуды немесе көбейтуді көрсете алады, немесе ол жай ғана 0 мәнін қайтарады. **action** әдісінің формальды қайтару түрі - бұл **Operation** интерфейсі және қайтарылған лямбда өрнегі осы интерфейске сәйкес келуі керек.

main әдісте біз бұл **action** әдіс деп атай аламыз. Мысалы, алдымен оның нәтижесін алыңыз - Operation айнымалысына тағайындалған лямбда өрнегі. Содан кейін **execute** әдісі арқылы мына лямбда өрнегін орындаңыз:

```
1 Operation func = action(1);
2 int a = func.execute(6, 5);
3 System.out.println(a);          // 11
```

Немесе сіз лямбда өрнегін дереу алуға және орындауға болады:

```
1 int b = action(2).execute(8, 2);
2 System.out.println(b);          // 6
```

Кіріктірілген функционалды интерфейстер

JDK 8-де, лямбда өрнектерінің функционалдылығымен қатар, біз көптеген әртүрлі функционалды интерфейстерді қосты, олар бізді әр түрлі жағдайларда және JDK 8 ішіндегі әр түрлі API интерфейстерінде қолдана алады. Атап айтқанда, төменде қарастырылған бірқатар интерфейстер Stream API-де кеңінен қолданылады - деректермен жұмыс істеуге арналған жаңа қолданбалы интерфейс. Осы интерфейстердің негізгісін қарастырайық:

- Predicate<T>
- Consumer<T>
- Function<T,R>
- Supplier<T>
- UnaryOperator<T>
- BinaryOperator<T>

Predicate<T>

Predicate <T> функционалды интерфейсі кейбір шарттардың орындалуын тексереді. Егер рас болса, онда **true** қайтарылады. Ламбда өрнегі параметр ретінде T типіндегі объектіні қабылдайды:

```
1 public interface Predicate<T> {
2     boolean test(T t);
3 }
```

Мысалы:

```
1  import java.util.function.Predicate;
2
3  public class LambdaApp {
4
5      public static void main(String[] args) {
6
7          Predicate<Integer> isPositive = x -> x > 0;
8
9          System.out.println(isPositive.test(5)); // true
10         System.out.println(isPositive.test(-7)); // false
11     }
12}
```

BinaryOperator <T>

BinaryOperator <T> параметр ретінде T типіндегі екі объектіні қабылдайды, оларда екілік амалдар орындайды және оның нәтижесін T типті объект ретінде де қайтарады:

```
1 public interface BinaryOperator<T> {
2     T apply(T t1, T t2);
3 }
```

Мысалы:

```
1  import java.util.function.BinaryOperator;
2
3  public class LambdaApp {
4
5      public static void main(String[] args) {
6
7          BinaryOperator<Integer> multiply = (x, y) -> x*y;
8
```

```
9         System.out.println(multiply.apply(3, 5)); // 15
10        System.out.println(multiply.apply(10, -2)); // -20
11    }
12}
```

UnaryOperator <T>

UnaryOperator <T> параметр ретінде T типіндегі объектіні қабылдайды, олармен операциялар орындайды және операциялардың нәтижесін T типті объект ретінде қайтарады:

```
1 public interface UnaryOperator<T> {
2     T apply(T t);
3 }
```

Мысалы:

```
1  import java.util.function.UnaryOperator;
2
3  public class LambdaApp {
4
5      public static void main(String[] args) {
6
7          UnaryOperator<Integer> square = x -> x*x;
8          System.out.println(square.apply(5)); // 25
9      }
10}
```

Function<T,R>

Function<T,R> функционалды интерфейсі T типті объектіден R типті объектіге өту функциясын ұсынады:

```
1 public interface Function<T, R> {
2     R apply(T t);
3 }
```

Мысалы:

```
1  import java.util.function.Function;
2
3  public class LambdaApp {
4
5      public static void main(String[] args) {
6
7   Function<Integer, String> convert = x-> String.valueOf(x) + " долларов";
8          System.out.println(convert.apply(5)); // 5 долларов
9      }
10}
```

Consumer<T>

Consumer<T>T типіндегі объектіге ешнәрсе қайтармай-ақ кейбір әрекеттерді орындайды:

```
1 public interface Consumer<T> {
2     void accept(T t);
3 }
```

Мысалы:

```
import java.util.function.Consumer;

public class LambdaApp {

    public static void main(String[] args) {

        Consumer<Integer> printer = x-> System.out.printf("%d долларов \n",
x);
        printer.accept(600); // 600 долларов
    }
}
```

Supplier<T>

Supplier<T> ешқандай дәлел келтірмейді, бірақ T типті объектіні қайтаруы керек:

```
1 public interface Supplier<T> {
2     T get();
3 }
```

Мысалы:

```
import java.util.Scanner;
import java.util.function.Supplier;

public class LambdaApp {

    public static void main(String[] args) {

        Supplier<User> userFactory = ()->{

            Scanner in = new Scanner(System.in);
            System.out.println("Введитеимя: ");
```

```
12          String name = in.nextLine();
13          return new User(name);
14      };
15
16      User user1 = userFactory.get();
17      User user2 = userFactory.get();
18
19      System.out.println("Имя user1: " + user1.getName());
20      System.out.println("Имя user2: " + user2.getName());
21  }
22}
23class User{
24
25  private String name;
26  String getName(){
27      return name;
28  }
29
30  User(String n){
31      this.name=n;
32  }
33}
```

Консоль шығысы:

```
Введите имя:
Том
Введите имя:
Сэм
Имя user1: Том
Имя user2: Сэм
```

БАҒДАРЛАМАЛАУ БОЙЫНША ПРАКТИКАЛЫҚ ТАПСЫРМАЛАР

Сызықтық алгоритмдерді бағдарламалау

Алгоритмнің құрылымдық схемесын құрып, есеп шешімінің бағдарламасын жазыңыз. Берілген деректерді пернетақтадан енгізіңіз

1. Ақпарат көлемін байтта өрнектейтін Z шамасы берілген. Z шамасын ақпарат көлемінің ірі бірлігіне ауыстырыңыз.
2. a және b нақты сандары берілген. Олардың қосындысын, айырмасын, көбейтіндісін, бөліндісін табыңыз.
3. Трапеция ауданын табыңыз. a, b, H белгілі.
4. Материальдық нүктенің энергиясын формуласы бойынша есептеңіз.
5. Радиусы берілген дөңгелектің ұзындығы мен ауданын табыңыз.
6. Екі қабырғасы берілген тік бұрышты үшбұрыштың ауданын табыңыз.
7. Қабырғасы берілген дұрыс үшбұрыштың биіктігін табыңыз.
8. Үш қабырғасы бойынша үшбұрыштың ауданын табыңыз.
9. Теріс емес a және b сандары берілген. Олардың геометриялық ортасын табыңыз.
10. Үш сан берілген. Олардың арифметикалық ортасын табыңыз.
11. Төбелерінің координаталары берілген үшбұрыштың периметрін табыңыз.
12. Қабырғалары берілген үшбұрыштың биссектрисасын анықтаңыз.
13. Цилиндр мен конустың көлемін есептейтін бағдарлама жазыңыз.
14. Ом заңын есептейтін бағдарлама жазыңыз.
15. Фаренгейт градусының мәні берілген Цельсий градусына ауыстырыңыз.

Тармақталған алгоритмдерді бағдарламалау

Алгоритмнің құрылымдық схемесын құрып, есеп шешімінің бағдарламасын жазыңыз. Берілген деректерді пернетақтадан енгізіңіз

1. Пернетақтадан екі сан енгізіңіз, олардың квадраттарының қосындысы үлкен бе әлде қосындыларының квадраты үлкен бе? Жауапты хабарлама түрінде шығарыңыз.
2. A және B нүктелері координаларымен берілген. Координалар бас нүктесінен осы нүктелердің қайсысы алшақ орналасқан. Жауапты хабарлама түрінде шығарыңыз.

3. Пернетақтадан үшбұрыштың үш қабырғасын енгізіп, үшбірыштың тік бұрышты болатындығын анықтаңыз. Жауапты хабарлама түрінде шығарыңыз.

4. Үш бүтін сан берілген. Олардың ішінен интервалға жататындарын таңдаңыз [1,3].

5. Жыл нөмірі берілген (оң бүтін сан). Осы жылы күндер санын анықтаңыз, әдеттегі жыл 365 күнді, ал секіріс — 366 күнді құрайды. Жыл 4-ке бөлінеді, 100-ге бөлінетін және 400-ге бөлінбейтін жылдарды қоспағанда (мысалы, 300, 1300 және 1900 жылдар секіріс емес, 1200 және 2000 жылдар).

6. Енгізілген өлшем бірлігінің нөмірі (1 — килограмм, 2 — миллиграмм, 3 — грамм, 4 — тонна, 5 — центнер) және М массасы бойынша килограмдағы массаның тиісті мәнін беретін бағдарлама жазыңыз.

7. Берілген 4 санның минималды косинусын табыңыз.

8. Берілген 3 санның максималды синусын экранға шығарыңыз.

9. Тең бүйірлі үшбұрышының ауданын есептеу бағдарламасын жасаңыз. Егер үшбұрыштың ауданы жұп болса, оны 2-ге бөліңіз, әйтпесе " мен 2-ге бөле алмаймын!» деген хабарлама шығарыңыз.

10. Үш сан берілген. Олардың арасында оң сандардың санын табыңыз.

11. Осы сан бойынша (1-12) оған сәйкес айдың атауын ағылшын тілінде көрсететін бағдарлама жасаңыз.

12. Шамаларды Радиан өлшемінен градусқа немесе керісінше аударатын бағдарлама жасаңыз. Бағдарлама қай аударманы жүзеге асыруды және көрсетілген әрекетті орындауды сұрауы керек.

13. X,y,z қабырғалары бар тікбұрышты үшбұрыштың бар-жоғын анықтаңыз.

14. a, b, c қабырғаларының ұзындығы бар үшбұрыштың бар – жоғын анықтаңыз, егер Иә болса, оның ауданын Герон формуласы бойынша есептеңіз.

15. Жеңілдікті ескере отырып, сатып алу құнын есептеу бағдарламасын жазыңыз. Егер сатып алу сомасы 5000 теңгеден асатын болса, 3% жеңілдік беріледі., 5% - егер сома 10000 теңгеден асатын болса.

Циклдік алгоритмдерді бағдарламалау

1. Нақты сан берілген-1 кг кәмпиттің бағасы. 1, 2, ... 10 кг кәмпиттің құнын шығарыңыз.

2. Нөлмен аяқталатын бүтін сандардың бос тізбегі берілген.

Табыңыз:

а) тізбектің барлық сандарының қосындысын;

ә) тізбектегі барлық сандар санын.

3. A және B (A<B) екі саны берілген. A-дан B-ға дейінгі барлық бүтін сандардың қосындысын табыңыз.

4. Оң санмен аяқталатын теріс бүтін сандар тізбегі берілген. Тізбектің барлық сандарының арифметикалық ортасын табыңыз (оң санды есептемегенде).

5. A және B (A<B) екі саны берілген. A-дан B-ға дейінгі барлық бүтін сандардың квадраттарының қосындысын табыңыз.

6. A-дан 200-ге дейінгі барлық бүтін сандардың арифметикалық ортасын табыңыз (A және b мәндері пернетақтадан енгізіледі; a≤ 200).

7. A-дан B-ға дейінгі барлық бүтін сандардың қосындысын табыңыз (A және b мәндері пернетақтадан енгізіледі; B ≥A).

8. Берілген бүтін сан N (> 0), бұл 2 санының белгілі бір дәрежесі: N = 2^K. бүтін санды табыңыз K — осы дәреженің көрсеткіші.

9. A-дан 50-ге дейінгі барлық бүтін сандардың квадраттарының қосындысын табыңыз (a мәні пернетақтадан енгізіледі; 0 ≤a≤50).

10. N нақты сандар тізбегі берілген. Тізбектегі бірінші сан тақ. Тақ сандар тізбегінің басында қатарынан шыққан барлық қосындыларды табыңыз.

11. Берілген бүтін сан N (>0). Бөлу операцияларын мақсатты түрде қолданып, бөлуден қалған қалдықты алып, оның цифрларының саны мен қосындысын табыңыз.

12. Сыныптың 20 оқушысының әрқайсысының физика бойынша бағалары белгілі. Орташа бағаны анықтаңыз.

13. Облыста 12 аудан бар. Әр ауданның тұрғындары (мың адам) және ауданы (км2) белгілі. Жалпы облыс бойынша халықтың орташа тығыздығын анықтаңыз.

14. Менің бай ағам маған бірінші туған күнімде бір доллар берді. Әр туған күнінде ол өзінің сыйлығын екі есе арттырды және оған менің толған жасымды қосып берді.. Сыйлықтың қай туған күнімде 100 доллардан асатынын көрсететін бағдарлама жазыңыз.

Тізімдер

1. N бүтін элементтерден тұратын бір өлшемді массив берілген. Пернетақтадан массив енгізіңіз. Максималды элементті табыңыз. Массивті кері ретпен экранға шығарыңыз.

2. Нақты сандар массивінде барлық нөлдік элементтерді массивтің барлық элементтерінің арифметикалық ортасымен ауыстырыңыз

3. N бүтін элементтерден тұратын бір өлшемді массив берілген. Пернетақтадан массив енгізіңіз. Минималды элементті табыңыз. Экранда минималды элементтің индексін көрсетіңіз

4. Бүтін сандар массиві берілген. Барлық оң элементтерді екінші массивке, ал қалғанын үшінші массивке қайта жазыңыз

5. Бір өлшемді сандық массивте d ұзындығы N тақ индекстері бар элементтердің қосындысын есептеңіз. Алынған соманы d массивіне экранға шығарыңыз.

6. Бүтін сандар массиві берілген. Массивтің Максималды элементін және оның реттік нөмірін табыңыз.

7. Бүтін типтегі бір өлшемді массив берілген. Бастапқы массивтің тақ сандарынан тұратын басқа массив алыңыз немесе мұндай сандар жоқ екенін хабарлаңыз. Алынған массив элементтердің кему ретімен көрсетіңіз.

8. 10 бүтін сандардың бір өлшемді массиві берілген. Қатар тұрған теріс сандардың жұптарын шығарыңыз.

9. 10 өлшемі10 болатын бүтін массив берілген. Барлық бірдей элементтерді алып тастап, оларды 1 рет қалдырып, жаңа массив жасаңыз.

10. Бүтін сандар массиві берілген. Жұп сандары бар элементтердің қосындысын және тақ сандары бар элементтердің көбейтіндісін табыңыз.

11. Нақты сандар массивінде барлық нөлдік элементтерді массивтің барлық элементтерінің арифметикалық ортасымен ауыстырыңыз.

12. N нақты элементтерден тұратын бір өлшемді массив берілген. Пернетақтадан массив енгізіңіз. Минималды модуль элементін табыңыз және шығарыңыз. Массивті кері ретпен экранға шығарыңыз.

13. 2-ге қалдықсыз бөлінетін тізімнің ең үлкен элементін табыңыз және оны экранға шығарыңыз.

14. Тізімнің ең кішкентай тақ элементін тауып, оны экранға шығарыңыз.

15. Тізімде қайталанатын элементтер бар-жоғын анықтаңыз, егер болса, осы мәндерді көрсетіңіз.

Жолдар

1. Орыс тіліндегі мәтінді қамтитын жол берілген. "Е" әрпінен басталатын сөздердің санын табыңыз.

2. Жолда барлық Қос нүктелер (:) пайыз (%) белгісімен ауыстырылсын. Ауыстыру санын есептеңіз.

3. Жолда нүкте таңбасын жойыңыз(.) және жойылған таңбалардың санын есептеңіз.

4. Жолда(А) әрпі (о) әрпімен ауыстырылсын. Ауыстыру санын есептеңіз. Жолда қанша символ бар екенін есептеңіз.

5. Жолда барлық бас әріптерді кіші әріптермен ауыстырыңыз.

6. Жолда барлық "А" әріптерін алып тастаңыз және жойылған таңбалардың санын есептеңіз.

7. Жол берілген. Алғашқы n/2 таңбалар арасында кездесетін барлық "п" әріптерін жұлдызшалармен ауыстыру арқылы оны түрлендіріңіз. Мұнда N-жолдың ұзындығы.

8. Нүктемен аяқталатын жол берілген. Жолда қанша сөз бар екенін анықтаңыз.

9. Берілген сөз мәтінде қанша рет кездесетінін анықтаңыз.

10. Ағылшын тілінде сөйлем-жол берілген. Әр сөз бас әріптен басталатын етіп жолды түрлендіріңіз.

11. Жол берілген . "я"әрпімен аяқталатын барлық сөздерді шығарыңыз.

12. Жолдың оңнан солға және солдан оңға қарай бірдей оқылатындығын тексеріңіз(яғни, бұл палиндром болатындығын).

13. Таңбалар жолы берілген, олардың арасында бір ашылатын және бір жабылатын жақшалар бар. Осы жақшалардың ішінде орналасқан барлық таңбаларды экранға шығарыңыз.

14. Жол берілген.. "А" әрпінен басталатын және "я" әрпімен аяқталатын барлық сөздерді шығарыңыз.

15. Мәтін жолы берілген. Жолдағы "т" әріптерінің санын есептеңіз.

Файлдар

1.Кез келген ақпаратпен мәтіндік файл жасаңыз. Файлдың мазмұнын қарауды ұйымдастырыңыз. Деректерді оқу мен өңдеуді ұйымдастырыңыз7

2.Алынған нәтижені жаңа мәтіндік файлға сақтаңыз.

Бинарлық файл жасаңыз.Компоненттері төмендегідей құрылымды қамтысын:

Білім алушының аты жөні, топ номері, үлгерімі, стипендиясы

3.Бинарлық файл жасаңыз.Компоненттері төмендегідей құрылымды қамтысын:

Завод жұмысшыларының тізімі, қызметкерлер лауазымы, жалақысы.

4.Бинарлық файл жасаңыз.Компоненттері төмендегідей құрылымды қамтысын:

Тауар атауы, тауар құны, өлшем бірлігі.

5.Бинарлық файл жасаңыз.Компоненттері төмендегідей құрылымды қамтысын:

Фирма атауы, диоганаль бойынша өлшемі, құны.

6.Бинарлық файл жасаңыз.Компоненттері төмендегідей құрылымды қамтысын:

Компьютер атауы, процессор жиілігі, жедел жады көлемі, қатты диск көлемі, құны.

Обьектіге бағытталған бағдарламалау негіздері

Екі өріс (өріс1 және өріс2) және үш әдіс (обьектіні инизиализациялауға арналған конструктор, обьект туралы ақпараты бар жолдарды қалыптастыру функциясы, жеке берілген тапсырмалар бойынша өріс міндерін өңдеу функциясы) көрсетілген класс жасаңыз.

№	Өріс 1	Өріс 2	Өрістерді өңдеу функциясы
1	Купюр номиналдары	Купюр саны	Купюр қосындысын есептеу
2	Тауар бағасы	тауар саны	Тауар жалпы құнын есептеу
3	Монета номиналы	Монета саны	Монета қосындысын есептеу
4	100 гр тағам колориясы	Тағам салмағы	Барлық тағамдағы колория
5	Минут саны	Секунд саны	Жалпы секунд есептеу
6	Сағат саны	Минут саны	Жалпы минут есептеу
7	Нақты сан: Тік бұрышты үшбұрыштың бірінші катеті	Нақты сан:тік бұрышты үшбұрыштың екінші катеті	тік бұрышты үшбұрыштың ауданын есептеу
8	Нақты сан: Тік бұрышты үшбұрыштың бірінші катеті	Нақты сан:тік бұрышты үшбұрыштың екінші катеті	Гипотенузаны есептеу
9	Нақты сан: қозғалыс жылдамдығы	Бүтін сан: қозғалыс уақыты (мин)	Жүрілген жолды (метр)
10	Бүтін сан: х	Бүтін сан: у	Х ті у ке бөлгендегі бүтінді есептеңіз
11	Бүтін сан: х	Бүтін сан: у	Кіші санның квадратын есептеңіз
12	Бүтін сан: х	Бүтін сан: у	Үлкен санның кубын есептеңіз
13	Шеңбер радиусы	Радиандағы бұрыш	Доға ұзындығын есептеңіз
14	Шеңбер радиусы	Цилиндр биіктігі	Цилиндр бетінің ауданын есептеңіз
15	Шеңбер радиусы	Конус биіктігі	Конус көлемін есептеңіз.

ПАЙДАЛАНЫЛҒАН ӘДЕБИЕТТЕР ТІЗІМІ

1. Бекенова А.С. Python бағдарламалау тілі : Оқу құралы / Бекенова А.С. – Орал : БҚАТУ, 2019. – 121 б.

2. М. Фаэлло Стильный Java. Код, который работает всегда и везде/Фаэлло М. – Питер, 2021. — 352 с.

3. Г. Энтони Изучаем Java EE 7 / Энтони Г. - М.: Питер, 2019. - 640 с.

4. Доусон М. Программируем на Python:/ Доусон М. — СПб.: Питер, 2020. — 416 с.

5. Коэльё Л. П. Построение систем машинного обучения на языке Python: пер. с англ. / Коэльё Л. П., Ричерт В. — М.: ДМК Пресс, 2016. - 234 с.

6. Маккинли У. Python и анализ данных: пер. с англ./ Маккинли У. - М.: ДМК Пресс, 2021. — 484 с.

7. Саммерфилд М. Python на практике: пер. с англ. / Саммерфилд М. - М. : ДМК Пресс, 2020. — 608 с.

8. Северенс Ч. Введение в программирование на Python / Ч. Северенс. - 2-е изд., испр. - М. : Национальный Открытый Университет «ИНТУИТ», 2016. - 231 с.

9. Сысоева М.В., Сысоев И.В. Программирование для «нормальных» с нуля на языке Python : учебник. В двух частях. Часть 1 / Сысоева М.В., Сысоев И.В. от. ред. В.Л.Черный : — М.: Базальт СПО; МАКС Пресс, 2018. – 176с.

10. Фёдоров Д. Ю. Основы программирования на примере языка Python : учебное пособие / Фёдоров Д. Ю. — СПб. : Юрайт, 2018. - 167 с.

11. Хахаев И.А. Практикум по алгоритмизации и программированию на Python : курс / И.А. Хахаев. - 2-е изд., исправ. - М. : Национальный Открытый Университет «ИНТУИТ», 2016. - 179 с.

12. Гарнаев А. WEB-программирование на Java и JavaScript/ А. Гарнаев, С. Гарнаев - Москва: СПб. [и др.] : Питер, 2017. - 718 с.

Бекенова Анаргуль Сагиндиковна
техника ғылымдарының магистрі, аға оқытушы

Бекенова Сандуғаш Сагиндиковна
техника ғылымдарының магистрі, оқытушы

ЗАМАНАУИ БАҒДАРЛАМАЛАУ ТІЛДЕРІ

Оқу құралы

26.05.2021 ж. басуға қол қойылды
Пшімі 60х84 1/16 Офсетті қағаз 80 м/г
Көлемі 13 Тапсырыс №22
Таралымы 500 дана

Дайын түпнұсқасының сапасына толық сәйкестікте басылды

Жәңгір хан атындағы Батыс Қазақстан
аграрлық-техникалық университеті
090009 Орал қ., Жәңгір хан көшесі, 51