32.973.2
К 75

Б.О. Көчкөнбаева,
К.К. Шакиров,
Г. И. Сейитказыева



JAVA

ПРОГРАММАЛОО ТИЛИНЕ КИРИШҮҮ

Окуу-усулдук комплекси

32.973.2
К 75



Java программалоо тилине киришүү

Окуу-усулдук комплекси

КЫРГЫЗ РЕСПУБЛИКАСЫНЫН БИЛИМ БЕРҮҮ ЖАНА ИЛИМ МИНИСТРЛИГИ ОШ МАМЛЕКЕТТИК УНИВЕРСИТЕТИ
БЕЛЕК
39466
№
6316.

Ош-2024

УДК 004.43
ББК 32.973.2
К 76

Окуу-усулдук колдонмо Ош мамлекеттик университетинин Окумуштуулар кеңешинин 2024-жылдын 15-майындагы №8-жыйындын токтомунун негизинде басмага сунушталган.

Рецензенттер:

Маматкасымова А. - Ош технологиялык университетинин доценти, физ.-мат.илим.канд.

Азимов Б. – Ош мамлекеттик университетинин математика, физика, техника жана информациялык технологиялар институтунун директору, физ.-мат.илим.канд., доцент.

Көчкөнбаева, Б.О. ж.б.

К 76 Java программалоо тилине киришүү: Окуу-усулдук комплекси. / Б.О. Көчкөнбаева, К.К. Шакиров, Г. И. Сейитказыева, Ош: 2024, 240 б.

ISBN 978-9967-06-056-2

"Java программалоо тилине киришүү" окуу-усулдук комплекси Java тилин үйрөнүүнү каалаган студенттерге арналат. Окуу-усулдук комплекс 15 лекциялык жана 14 практикалык сабактардан турат. Ар бир лекция сабактын максатын, кеңири теориялык жана практикалык мисалдарды, бышыктоочу суроолорду камтып турат. Ошондой студенттин өз алдынча иштөөсү үчүн практикалык сабак бир канча мисалдардын чыгарылышы жана өз алдынча иштөө үчүн маселелердин тизмеси менен берилди.

УДК 004.43
ББК 32.973.2

ISBN 978-9967-06-056-2

© Б.О. Көчкөнбаева, К.К. Шакиров, Г. И. Сейитказыева, 2024

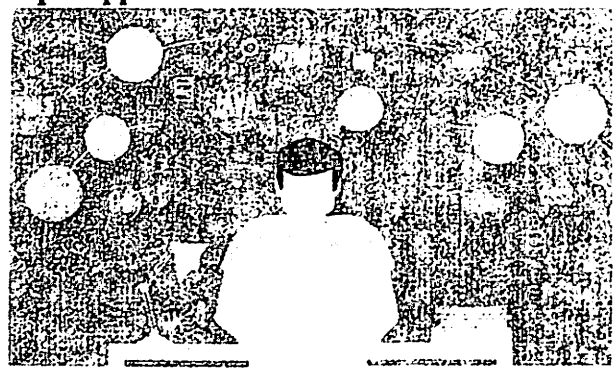
Мазмуну

Лекция 1. Киришүү 6
Java тили эмнеге жөндөмдүү? 7
Объектке багытталган программалоо 8
Лекция 2. Программалоого киришүү 9
Java Development Kit (JDK) 9
Java тилиндеги биринчи программа 9
IDE ни жүктөө жана колдоно билүү 11
1-практикалык сабак 13
Лекция 3. Өзгөрмөлөр жана турактуулар. Берилгендердин типтери 19
Программанын структурасы 19
Өзгөрмөлөр 21
Турактуулар 22
Берилгендердин типтери 23
Арифметикалык операциялар 25
Escape-код символдорду 27
Консолдук кийирүү/чыгаруу операциялары 28
Math классы жана анын методдору 32
2-практикалык сабак 34
Лекция 4. Шарттуу конструкциялар 39
Логикалык операциялар 39
if ... else конструкциясы 48
Программалык коддун блоктору 49
Switch конструкциясы 51
Тернардык оператор 53
3-практикалык сабак 55
Лекция 5. Циклдык конструкциялар 61
for циклдык конструкциясы 62
do... while циклдык конструкциясы 65
while циклдык конструкциясы 65
continue жана break операторлору 67
Камтылуучу циклдар 69
4-практикалык сабак 70
Лекция 6. Массивдер 75
Бир өлчөмдүү массивдер 75
Массивди жарыялоо 76
Массивди түзүү 77
Массивдин узундугу 77
Бир өлчөмдүү массивдердин үстүнөн амалдар 79

5- практикалык сабак 83
Лекция 7. Эки өлчөмдүү массивдер 85
Эки өлчөмдүү массивди жарыялоо жана түзүү 86
Массивди инициализациялоо 87
Тиштүү массивдер 89
foreach циклдык конструкциясы 90
Эки өлчөмдүү массивдин үстүнөн амалдар 90
6- практикалык сабак 93
Лекция 8. Методдор 95
Метод жөнүндө 96
Статикалык методдор 96
Методдун параметрлерин аныктоо 99
return оператору. Методдун жыйынтыгы 101
Методду ашыкча жүктөө 104
Рекурсиялык функциялар 105
7- практикалык сабак 107
Лекция 9. Жолчолор (строки). String классы 109
Жолчолор жөнүндө 109
Жолчолорду түзүү 109
Жолчолордун үстүнөн иштөөдө колдонулуучу негизги методдор 110
Жолчолорду салыштыруу 111
Камтылуучу жолчолорду издөө 114
StringBuffer жана StringBuilder 116
Регулярдуу туюнтмалар 120
8 - практикалык сабак 122
Лекция 10. Класстар. Объектке багытталган программалоо 125
Класс жана объекттер 125
Классты түзүү 126
Конструктор 128
this ачкыч сөзү 130
Инициализаторлор 131
static модификатору 135
9-практикалык сабак 136
Лекция 11. ОБПнун негизги түшүнүктөрү 139
Мүмкүндүк алуу модификаторлору 139
Инкапсуляция 142
Объекттер методдун параметри катары 145
Мурастоо 146
Абстрактуу класстар 149

Полиморфизм 151
Камтылуучу класстар 152
10- практикалык сабак 155
Лекция 12 Каталардын үстүнөн иштөө же өзгөчө кырдаалдарды алдын алуу 158
try...catch...finally блоктору 159
throw жана throws ачкыч сөздөрү 164
11- практикалык сабак 166
Лекция 13. Java тилинде файлдар менен иштөө 170
Кийирүү/чыгаруу агымдары 170
Файлдарды окуу жана жазуу. FileInputStream жана FileOutputStream 171
Буферленген агымдар. BufferedInputStream жана BufferedOutputStream 174
PrintStream жана PrintWriter класстары 177
File классы. Файлдар жана папкалар менен иштөө 179
12-практикалык сабак 185
Лекция 14. Коллекциялар 188
Коллекциянын түрлөрү. Коллекция интерфейси 188
Тизме(List). ArrayList классы жана List интерфейси 192
Queue (Кезектер) жана ArrayDeque классы 194
Set интерфейси жана HashSet классы 199
Map интерфейси жана HashMap классы 202
13- практикалык сабак 205
Лекция 15. Java тилинин графикалык мүмкүнчүлүктөрү 206
Графикалык интерфейсти орнотуу 206
Swing библиотекасы 212
Жогорку деңгээлдеги терезе. JFrame классы 214
Graphics классы 223
Color классы 226
Font классы 227
14-практикалык сабак 229
Колдонулган адабияттар 233
Тесттик тапшырмалар 234

Лекция 1. Киришүү



Бүгүнкү күндө Java дүйнөдөгү эң кеңири таралган жана популярдуу программалоо тилдеринин бири. Бул китеп менен биз Java'нын мүмкүнчүлүктөрүн окуп үйрөнүүгө аракет кылабыз. Алгач Java 1995-жылы Sun Microsystems компаниясы тарабынан иштелип чыгып, кийинчерээк Oracle Corporation компаниясы тарабынан сатылып алынган. Java программалоо тили ар кандай маселелер үчүн колдонула турган универсалдуу программалоо тили катары иштелип чыккан. Ал эми азыр Java тили көп жолду басып өтүп, көптөгөн ар кандай версиялары бар. Учурдагы версиясы Java 20, ал 2023-жылдын мартында чыгарылган.

Java программалоо тилин бир катар милдеттерди аткаруу үчүн колдонулган ар кандай технологияларды бириктирген өзүнүн экосистемасы бар толук кандуу платформа деп атоого болот. Бүгүнкү күндө Java тили чынында эле бардык багытта, кичинекей программаларды түзүүдөн баштап, чоң көлөмдөгү интернет порталдарга чейин колдонулуп жатат. Ошондой эле Android операциондук системасындагы көпчүлүк тиркемелер Java тилинде жазылган.

Java тили эмнеге жөндөмдүү?



Java тили атайын библиотекаларга ээ. Анын жардамында веб-сайттарды тез өнүктүрүүгө, оюндарды түзүүгө, компьютер үчүн программа жазууга, ал тургай мобилдик түзүлүштөр же планшеттер үчүн тиркеме түзүүгө да болот. Андан сырткары Java тили микроконтроллерлер үчүн да программа жазууга ыңгайлуу.

Java тилинин негизги артыкчылыгы "бир жолу жазылган код, бардык жерде иштейт" принциби. Бул бир платформада жазылган программа башка түзүлүштөрдө иштей турганын билдирет. Java тили мобилдик түзүлүштөр, аралыктагы процессорлор, зымсыз модулдар, сенсорлор жана жалпысынан дээрлик бардык электрдик түзүлүштөр үчүн тиркемелерди түзүү үчүн колдонулат.

Java тилинин синтаксиси C жана C++ тилдеринен алынган жана программанын сервердик бөлүгүн түзүүдө активдүү кодонулат. Мисалы “Amazon” толук эле Java’да түзүлгөн. Эстутумду башкаруу Java Virtual Machine (JVM) тарабынан жүргүзүлөт жана ал интеграцияланган автоматтык түрдө таштанды чогултуу механизмине ээ.

Объектке багытталган программалоо



Биз жашап жаткан дүйнө объекттерден турат. Айланабызды карасак үйлөр, бак-даректар, машиналар, эмеректер, идиш-аяктар, компьютерлер жана башка буюмдар курчап турат. Бул нерселердин баары объектилер болуп саналат жана алардын ар бири белгилүү бир мүнөздөмөлөргө, жүрүм-турумга жана максатка ээ. Биз объекттерге көнүп калганбыз жана аларды дайыма өзгөчө максаттар үчүн колдонобуз. Мисалы, жумушка барышыбыз керек болсо машинаны, тамактангыбыз келсе идиш-аякты, эс алышыбыз керек болсо ыңгайлуу диванды пайдаланабыз.

Адам күнүмдүк турмуштагы көйгөйлөрдү чечүү үчүн объективдүү ойлонууга көнүп калган. Ошондуктан объект түшүнүгүн программалоого кошкон жана бул түрдөгү программалоо ***объектке багытталган*** деп аталат.

Бышыктоочу суроолор:

1. Java тили эмнеге жөндөмдүү?
2. Объект деп эмнени түшүнөсүз?
3. Бизди курчап турган объекттер жөнүндө айтып бергиле?
4. Объектке багытталган программалоо кайдан келип чыкты?

Лекция 2. Программалоого киришүү

Максаты:

1. Java Development Kit (JDK) менен таанышуу
2. Java тилинде программа түзүүнү үйрөнүү
3. IDE ни жүктөө жана колдоно билүү

Java Development Kit (JDK)

JDK - Java тилиндеги тиркемелерди иштеп чыгууда программаларды түзүүгө жана иштетүүгө жардам берген акысыз каражаттардын топтому десек болот. Албетте, сиз кодду тексттик редактордо жаза аласыз, бирок JDKсиз программаны компиляциялап, иштетип жана мүчүлүштүктөрдү оңдоого мүмкүн эмес.

Java тилиндеги биринчи программа




```
/*
Менин биринчи программам
*/
public class Example1 {
    public static void main (String args []) {
        System.out.println ("Hello world");
    }
}
```

Бул **Блок** комментарий катары белгилүү. Бул саптар программист үчүн пайдалуу жана **Компилятор** тарабынан этибарга алынбайт

Файлды **Example1.java** деген аталышта сактаңыз. Программанын аты файлдын аталышына окшош болушу керек.

Программалар класстар деп аталат. ***Java регистрге сезимтал экенин эч качан унутпаңыз!***

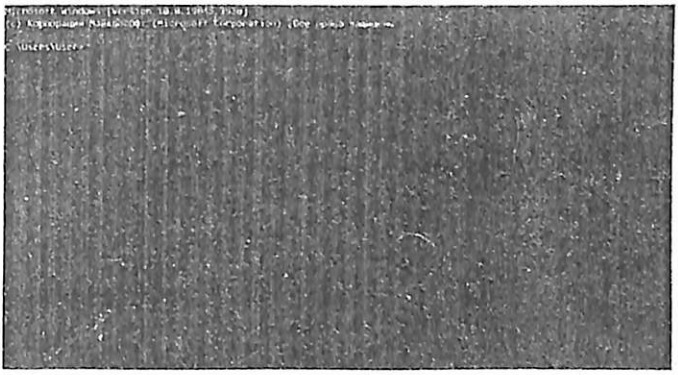
Сиз файлды "Example.java" деп атай албайсыз, андай болсо, программада "public class Example" деп жазасыз. Программист катары конкреттүү программа эмне жөнүндө экенин тез түшүнүүгө жардам берүү үчүн, программанын башталышында комментарийлерди кошуу жакшы практика болуп саналат. Комментарийдин башында "/*" жана аяктагандан кийин "*/" белгилерин коюу керек. Жогорудагы программанын болжолдуу жыйынтыгы төмөнкүдөй болот:

Hello world

Жогорудагы жыйынтыкты алуу үчүн биз алгач программаны компиляциялашыбыз керек, андан кийин компиляцияланган классты интерпретатордон өткөрүшүбүз керек. Бул жумуш үчүн талап кылынган колдонмолор JDK бөлүгү катары белгилүү:

- javac.exe – программаны компиляциялайт
- java.exe – компиляцияланган программаны аткаруу үчүн колдонулган интерпретатор

Программаны компиляциялоо жана аткаруу үчүн биз командалык жолчо терезесине өтүшүбүз керек (1-сүрөт). Windows системаларында бул ***Пуск>Выполнить>cmd*** тандоо менен жасалышы мүмкүн.



1-сүрөт Командалык жолчо терезеси

Бул жерде DOSтун кээ бир командалары керек болот:

- cd\ (каталогду өзгөртүү)
- cd\[папканын аты] керектүү папкага/каталогго өтүү үчүн

Керектүү жериңизге жеткенде, төмөнкүнү теришиңиз керек:

c:\[папканын аты]\javac Example1.java

Жогорудагы буйрук java файлын түзөт жана колдонуучуга кандайдыр бир каталарды көрсөтөт. Эгерде компиляция ийгиликтүү болсо, байт кодду камтыган жаңы файл түзүлөт: Example1.class

Андан кийин программаны аткаруу үчүн биз интерпретаторду чакырабыз:

c:\[папканын аты]\java Example1

Жыйынтык DOS терезесинде көрсөтүлөт.

IDE ни жүктөө жана колдоно билүү

Кээ бириңиздер жогорудагы программанын жыйынтыгын алууга чейин эле нааразы болушуңуздар мүмкүн. Бирок, ыңгайлуу жол бар, бул командалык жолчо терезесин унутуп, Java программалоо тили менен иштөө үчүн IDE (Integrated Development

Environment) колдонсо болот. Бир нече IDE'лер бар, алардын баары жакшы, бирок кээ бирлери менен иштөө башкаларга караганда жеңилирээк. Бул колдонуучунун программалоо деңгээлине жана табитине жараша болот! Төмөндө кээ бир IDE'лердин тизмеси келтирилген:

- BlueJ – www.bluej.org (акысыз)
- NetBeans – www.netbeans.org (акысыз/ ачык булак)
- JCreator – www.jcreator.com (акысыз версиясы бар, PRO версиясын сатып алуу талап кылынат)
- Eclipse – www.eclipse.org (акысыз/ ачык булак)
- IntelliJ IDEA – www.jetbrains.com (сыноо(trial) версиясы/сатып алуу талап кылынат)
- JBuilder – www.borland.com (сыноо версиясы/сатып алуу талап кылынат)

Java тилин жаңы баштагандар BlueJден ырахат алып, андан кийин JCreator, NetBeans, ж.б. ушул сыяктуу башка IDE'лерге өтүшү мүмкүн. Бул дагы колдонуучунун табитине жана программалык камсыздоону иштеп чыгуу чөйрөсүнө байланыштуу болот.

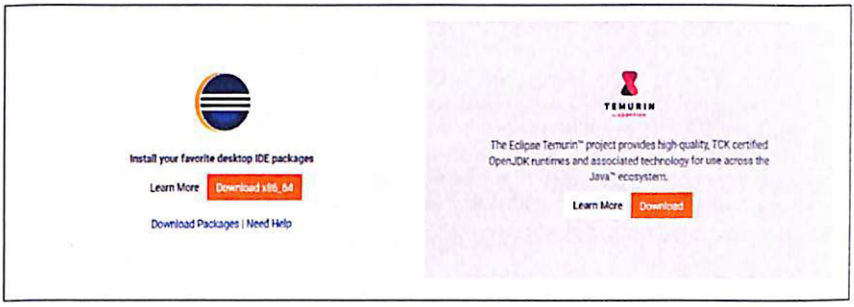
Бышыктоочу суроолор:

1. Java тилинин өзгөчөлүгү жөнүндө айтып бергиле
2. Сиз кайсы IDEни тандайсыз?
3. Java тилинде кандай программаларды түзсөк болот?
4. Азыр Java тилинин кайсы версиясы актуалдуу?
5. Блокнотто түзүлгөн java файлды кантип иштетсек болот?

1-практикалык сабак

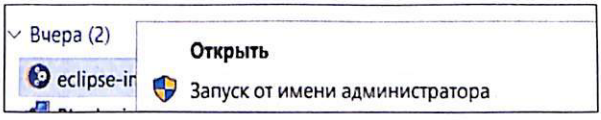
Максаты: Eclipse акысыз IDE'ни жүктөө. Биринчи программаны түзүү

Eclipse акысыз IDE'ни жүктөө үчүн браузердин адрес жолчосуна www.eclipse.org/downloads/ адресин жазабыз. Экранда төмөнкү шилтемелер ачылат (1-сүрөт):



1-сүрөт. Программаны жүктөп алуу терезеси

Өзүбүздүн операциондук системабызга жараша шилтемени тандайбыз. Биздин учурда Windows үчүн программаны тандап алдык. Бул учурда орнотуу пакети компьютерге көчүрүлөт. Кийинки этапта көчүрүлүп алынган образ файлды компьютерге орнотобуз. Образ файлга мышты алып барып, оң жак кнопкасын басабыз. Пайда болгон терезеден "Открыть" кнопкасын тандайбыз (2-сүрөт).



2-сүрөт. Программаны орнотуу

Кийинки этапта экранга төмөнкү терезе ачылат (3-сүрөт):



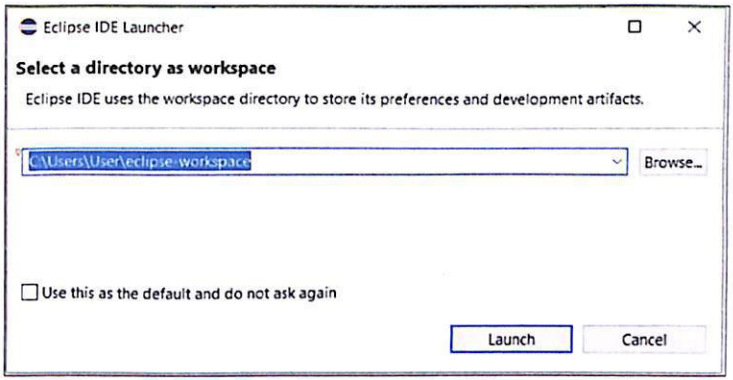
3-сүрөт. Орнотуу процесии

Eclipse IDE for java Developers менюсун тандайбыз да, кадам сайын пайда болгон терезелерге тиешелүү чечимдерди кабыл алуу менен программаны орнотуп алабыз (4-сүрөт).



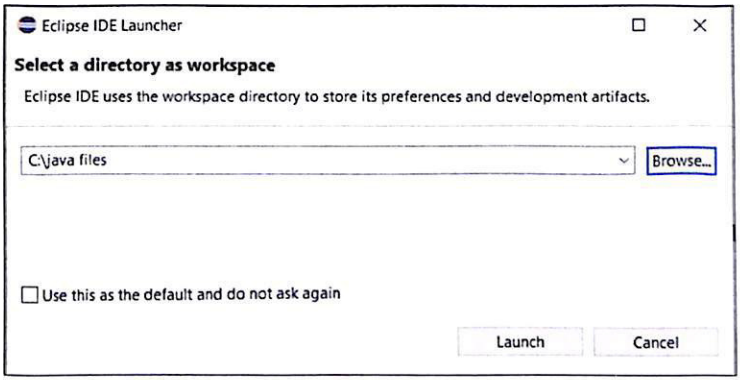
4-сүрөт. Орнотууну аягы

Орнотулган программаны иштетебиз. Алгачкы эле учурда төмөнкү терезе пайда болот (5-сүрөт):



5-сүрөт. Иштөө папкасын тандоо

Бул терезедеги жолчого биз java файлдарыбызды кайсы жерге сактай турганыбызды көрсөтөбүз. Ал үчүн "Browse" кнопкасын басып, өзүбүзгө ыңгайлуу жерди көрсөтөбүз. Мисалы, C: папкасына "java files" деген папка ачып алсаңыз да болот (6-сүрөт).



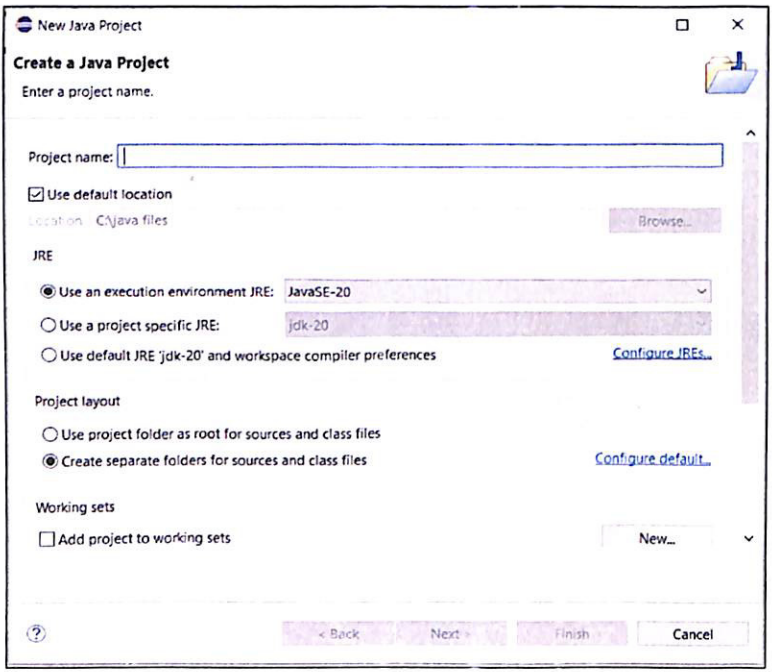
6-сүрөт. Өзүбүз түзгөн папканы тандап алуу

Андан кийин негизги терезе ачылат (7-сүрөт):



7-сүрөт. Негизги т резе

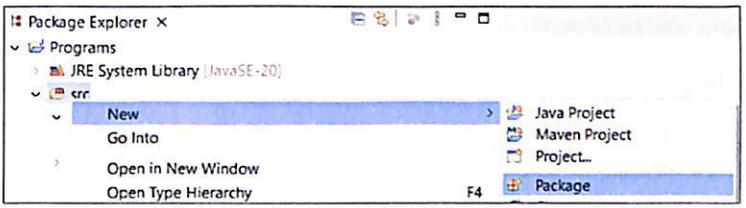
Бул терезеден же болбосо File менюсунан жаңы проект түзүү менюсун тандап алабыз.



8-сүрөт. Проектти үзүү

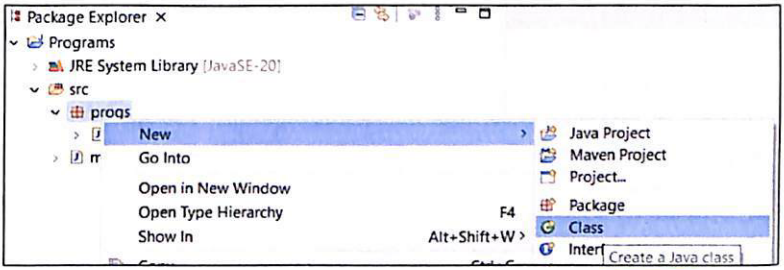
Проекттин атын өзүңүз каалагандай койсоңуз болот (мисалы, Programs деп койсоңуз болот). 8-сүрөттөн көрүнүп тургандай биз jdk-20 колдонобуз.

Андан кийин пакет түзүп алабыз. Ал үчүн негизги терезеде src папкасына барып, мыштын оң жагын басабыз New->Package менюсун тапдап алабыз да пайда болгон терезеге пакеттин атын жазып, “Finish” кнопкасын басабыз.



9-сүрөт. Пакетти түзүү

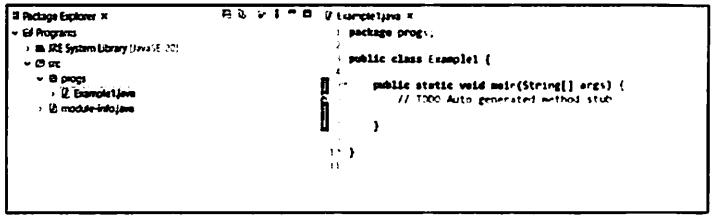
Java тилинде класстар пакеттин ичинде сакталат. Бир пакеттин ичинде бир канча класстар болушу мүмкүн. Кийинки этапта түзүп алган пакеттин үстүнө мышты алып барып, оң кнопкасын басабыз да New-> Class менюсун тандайбыз (10-сүрөт). Класстын атын биз Example1 деп коюуп алдык. (Класстын атын баш тамга менен жазасыз!)



10-сүрөт. Класс түзүү

Ошентип, класс түзүп алдык. 11-сүрөттө оң жакта класстын структурасын көрүп турасыз.

КЫРГЫЗ РЕСПУБЛИКАСЫНЫН БИЛИМ БЕРҮҮ ЖАНА ИЛИМ МИНИСТРЛИГИНИН ОШ МАМЛЕКЕТТИК УНИВЕРСИТЕТИ
39466
№



11-сүрөт. Биринчи java файл ын көрүнүшү

Биринчи программыны түзүү

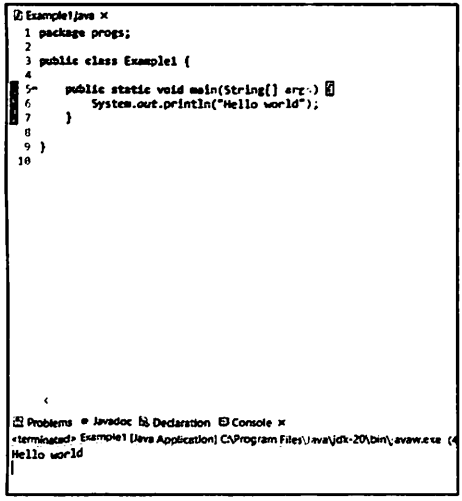
Example1.java файлыбыздын ичине төмө нкү кодду жазабыз:

```
package progs;

public class Example1 {

	public static void main(String[] args) {
		System.out.println("Hello world");
	}
}
```

Программаны иштетүү үчүн Run->R ın менюсун же Ctrl+F11 клавишаларын басабыз. Экрандын төмөн жагында жыйынтыкты көрөбүз (12-сүрөт).



12-сүрөт. Программанын ·кыйынтыгы

Эгерде ушул жыйынтыкты алып отурсаңыз, с ıз азаматсыз!

Лекция 3. Өзгөрмөлөр жана туракту уулар. Берилгендердин типтери

Максаты:

1. *Программанын структурасы менен таанышуу*
2. *Өзгөрмөлөр жана тураktуулар, арифметикалык операциялар, Escape-код символдорду колдоно билүү*
3. *Берилгендердин типтерин билүү, ыйгаруу операциясы жана операциялардын приоритети менен таанышуу*

Программанын структурасы

Java программасынын негизги бөлүгү болуп **инструкциялар** эсептелет. Ар бир инструкция методдорду чакыруу, өзгөрмөлөрдү жарыялоо жана аларга маанилерди ыйгаруу сыяктуу жумуштарды аткарат. Java тилинде инструкция аяктагандан кийин үтүрлүү чекит (;) колдонулат. Бул белги компиляторго инструкциянын аяктаганын көрсөтөт. Мисалы, биринчи программада колдонулган инструкцияны карап көрөлү:

```
System.out.println("Hello world");
```

Бул жолчодо System.out.println методу чакырылды жана ал консолго "Hello world" жолчосун чыгарып берди. Бул жерде методду чакыруу инструкция болгондуктан жолчонун аягына үтүрлүү чекит коюулду. Мындан сырткары программа бир канча инструкцияларды бириктирип туруучу блоктордон турушу мүмкүн. Бул учурда блоктун башталышы "{" белгиси менен башталып "}" белгиси менен аяктайт. Мисалы:

```
{
  System.out.println("Hello");
  System.out.println("How are you?");
}
```

Бул блокто эки инструкция бар жана алар тиешелүү жумуштарды аткарат.

Java объектиге багытталган тил болгондуктан, программаны бири-бири менен өз ара аракеттенүүчү класстардын жана объекттердин жыйындысы катары карасак болот. Кайрадан эле биринчи программабызды карап көрөлү. Программанын негизин Example1 классы түзүп турат жана ал public модификатору менен башталды. Бул класстын ачык экендигин көрсөтүп турат башкача айтканда классты командалык жолчодон иштете алабыз. Андан кийин class ачкыч сөзү классты жарыялап жатканыбызды көрсөтөт.

Программанын негизги башталгыч точкасы болуп main методу эсептелет.

public static void main(String[] args)

Программаны иштеткенде Java машина негизги класстан main методун жогорудагыдай көрүнүштө издейт жана бар болсо, ошол жерден иштеп баштайт.

Методдун ачык экендигин public модификаторунан байкасак болот, static сөзү методдун статикалык экендигин, void сөзү типти билдирет жана методдун эч кандай маани кайтарбай тургандыгын көрсөтүп турат.

Мындан сырткары программада комментарийлер да колдонулат. Биз ал жөнүндө жогоруда айтканбыз, алар бизге жолчолор жөнүндө маалыматты сактап коюуга жардам берет, бирок компилятор аларды этибарга албайт.

Комментарийлер эки түрдүү болот: бир эле жолчого коюулуучу (//), бир канча жолчого коюулуучу комментарийлер (/*...*/). Мисалы,

```
/*
   Бир канча жолчодон турган комментарий.
      Бул жерде класс түзүлүп жатат
*/
public class Example1{  // Example1 классынын жарыяланышы

  // негизги main методун аныктоо
  public static void main (String[] args

    System.out.println("Hello world"); //консолго чыгаруу
  }  // методдун аягы
} // Example1 классынын аягы
```

Өзгөрмөлөр

Программада берилгендерди сактоо үчүн ***өзгөрмөлөр*** колдонулат. Алар эс тутумдан анык бир типке ээ болгон маанини сактоого арналган белгилүү бир аймакты ээлешет. Ар бир өзгөрмөнүн тиби, аты жана мааниси бар. Тип өзгөрмө кандай маалыматты сактай аларын же мүмкүн болгон маанилердин диапазонун аныктайт.

Өзгөрмөлөр төмөнкүдөй жарыяланат:

Берилгендердин_тиби өзгөрмөнүн_аты;

Келгиле, **x** деп аталган **int** тибиндеги өзгөрмөнү жарыялап көрөлү:

int x;

Мында, **int** тибиндеги **x** өзгөрмөсү 4 байттан ашпаган бүтүн бир санды сактайт.

Өзгөрмөгө ат коюуда төмөнкү эрежелерди унутпаңыз:

• өзгөрмөнүн аты алфавиттик-цифралык белгилерди (a,b,...z,A,B,...Z, 0,1,2,...9), асты сызылган символду (_) камтышы мүмкүн;

• өзгөрмөнүн аты цифра менен башталбайт жана тыныш белгилерин же пробелди камтыбайт;

• java тилинин ачкыч сөздөрүн өзгөрмөнүн аталышына колдонууга болбойт!

Java тили регистрге сезимтал тил болгондуктан int num; жана int NUM; жолчолору эки башка өзгөрмө жарыяланганын билдирет.

Биз өзгөрмө жарыялагандан кийин гана ага маани бере алабыз.

```
int x; // өзгөрмөнү жарыялоо
x = 10; // маани берүү
System.out.println (x ); // консолго чыгаруу
```

Жогоруда өзгөрмөнү жарыялап ага маани берип көрдүк. Бул процесс өзгөрмөнү **инициализациялоо** деп аталат. Программада бир эле жолчодо өзгөрмөнү жарыялап жана инициализациялап алсак болот б.а.

```
int x = 10;
int z = 8, y = 15;
```

Туракгуулар

Java тилинде маалыматтарды сактоо үчүн өзгөрмөлөрдөн башка туракгууларды (constants) колдонсо болот. Өзгөрмөлөрдөн айырмаланып, туракгууларга бир гана жолу маани берилет жана ал өзгөрмө сыяктуу эле жарыяланат, бирок туракгуулар final ачкыч сөзү менен башталат. Мисалы:

```
final int LIMIT = 5;
```

System.out.println(LIMIT);

Эреже катары, туракгууларды жарыялоодо чоң тамгалар колдонулат.

Берилгендердин типтери

Java тилинин негизги өзгөчөлүктөрүнүн бири болуп типке каттуу көңүл бурулат. Бул ар бир өзгөрмө жана туракгуу белгилүү бир типке таандык маанилерди гана сактайт дегенди билдирет. Берилгендердин тиби өзгөрмө же туракгуу сактай ала турган маанилердин диапазонун аныктайт.

Келгиле, Java тилинде өзгөрмөлөрдү жана туракгууларды түзүү үчүн колдонулган негизги базалык типтердин тизмесин карап көрөлү (1-таблица):

Таблица 1. Берилгендердин типтери

Берилгендердин типтери	**Мааниси**
int	-2,147,483,648 ден 2,147,483,648ге чейинки 32 биттик бүтүн сан.
byte	-128ден 127ге чейинки 8 биттик бүтүн сан
short	-32,768ден 32,768ге чейинки 16 биттик бүтүн сан
long	-9,223,372,036,854,775,808 ден -9,223,372,036,854,775,808ге чейинки 64 биттик бүтүн сан
float	32 биттик чыныгы сан
double	64-биттик чыныгы сан
char	Символ, 0 дон 65,536га чейинки (Unicode) 16 биттик оң сандуу код

boolean	Логикалык тип. Чын (True) же жалган(False) маанилерин ала алат.

Жолчолор менен иштөөдө String классынын объектиси түзүлөт, ал char же int тибинен айырмаланып, Javanын примитивдүү же базалык тиби эмес. Аны төмөнкүдөй жарыялайбыз:

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

Эми көп саптуу чоң текстти көрсөтөлү:

```
String text = "Жер сагынбас эл болбос,\n"+
              "Эл сагынбас эр болбос.\n"+
              "Жатка моюн сунгандай,\n"+
              "Эр журогү кир болбос.";
System.out.println(text);
```

+ операциясын колдонуу менен биз бир текстти экинчисине улай алабыз, ал эми тексттин уландысы кийинки сапта жайгаша алат. Анткени кийинки сапка өтүүнү камсыз кылуу үчүн \n код символу колдонулат. Жыйынтыгы төмөнкүдөй болот (13-сүрөт):



13-сүрөт. Программанын жыйынтыгы

Java тили 15-версиядан баштап текст блокторун колдонуунун жаңы ыкмасын сунуштайт. Бул үч тырмакчага алынган көп саптуу текст. Келгиле, алардын практикалык мааниси кандай экенин карап көрөлү.

```
String text = """
Жер сагынбас эл болбос,
```

Эл сагынбас эр болбос.
Жатка моюн сунгандай,
Эр журогу кир болбос.
"""";
System.out.println(text);
Бул учурда биз төмөнкү жыйынтыкты алабыз (14-сүрөт):

```
Жер сагынбас эл болбос,
Эл сагынбас эр болбос.
Жатка моюн сунгандай,
Эр журогу кир болбос.
```

14-сүрөт. Программанын жыйынтыгы

String классы жөнүндө кийинчерээк кеңири карайбыз.

Арифметикалык операциялар

Арифметикалык операциялар экиге бөлүнөт: унардык(бир операнд катышуучу), бинардык (эки операнд катышуучу) операциялар.

Унардык операцияларга төмөнкүлөр кирет (2-таблица):

Таблица 2. Унардык операциялар

Оператор	Мааниси	Мисалы
+	Унардык плюс: Сан атайын көрсөтүлбөгөн учурда оң сан болуп эсептелет	+5 саны бул оң сан, аны 5 деп жазсак болот
-	Унардык минус: Сандын белгисин терс санга өзгөртөт	5 санына минус "-" белгисин кошсок -5 бул терс сан
++	Инкремент оператору:өзгөрмө нү 1 ге чоңойтот	a++ инкрементти a+1 дегенди түшүндүрөт б.а. a = a + 1

--	Декремент оператору: өзгөмөнү 1ге азайтат	a-- декременти a-1 дегенди түшүндүрөт б.а. a = a – 1
+=	Кошуу жана ыйгаруу оператору	a = a + 2 барабардыгын кошуу жана ыйгаруу операторунун жардамында a+=2 деп жазууга болот
-=	Кемитүү жана ыйгаруу оператору	a = a – 2 барабардыгын кемитүү жана ыйгаруу операторунун жардамында a-=2 деп жазууга болот
*=	Көбөйтүү жана ыйгаруу оператору	a = a * 3 барабардыгын көбөйтүү жана ыйгаруу операторунун жардамында a*=3 деп жазууга болот
/=	Бөлүү жана ыйгаруу оператору	a = a / 4 барабардыгын бөлүү жана ыйгаруу операторунун жардамында a/=4 деп жазууга болот

Бинардык операцияларга төмөнкүлөр кирет (3-таблица):

Таблица 3. Бинардык операциялар

Оператор	Мааниси	Мисалы
+	Кошуу операциясы: сандарды кошууда колдонулат	7+5
-	Кемитүү операциясы: сандан санды кемитүүдө колдонулат	7-5
*	Көбөйтүү операциясы: сандарды көбөйтүүдө колдонулат	7*5
/	Бөлүү операциясы: сандарды бөлүүдө колдонулат	20/5

%	Бөлүүнүн калдыгын алуу: Эки санды бөлүүдөн калган калдыкты алууда колдонулат	7%5=2 (7 санын 5 ке бөлгөндө калдыгы 2 ге барабар)

Escape-код символдорду

Бул символдор прогаммада жыйынтыкты экранга чыгарууда колдонулат (4-таблица).

Таблица 4. Код символдор

Код	Мааниси
\n	Жаңы жолчого өтүү
\t	Tab (4 пробел)
\r	Курсорду жолчонун башына алып барат (Enter)
\\	Тескери слеш
\'	Жалгыз тырмакча
\"	Кош тырмакча
*	Сегиздик - * санды же он алтылык цифраны билдирет
\u*	Unicode, мисалы, \u2122 = ™ (соода белгиси)

Арифметикалык операциялардын приоритети

Кээ бир операциялар приоритети чоң болгондуктан биринчи аткарылат жана ар бир кийинки операция приоритетинин төмөндөшүнө жараша аткарылат:

++ (постфикстик инкремент), -- (постфикстик декремент)

++ (префикстик инкремент), -- (префикстик декремент)

* (көбөйтүү), / (бөлүү), % (калдыгын алуу)

+ (кошуу), - (кемитүү)

Төмөнкү мисалды карап көрөлү:

```
int a = 8;
int b = 7;
int c = a + 5 * ++b;
System.out.println(c);
```

Биринчи болуп жогорку приоритетке ээ болгон инкремент операциясы ++b аткарылат, ал b өзгөрмөсүнүн маанисин бирге жогорулатып, жыйынтыгын кайтарат. Андан кийин көбөйтүү аткарылат 5 * ++b, эң акыркы болуп кошуу операциясы аткарылат. Бирок кашаалар эсептөөнүн тартибин өзгөртөт. Мисалы, 3-жолчодогу амалды

```
int c = (a + 5) * ++b;
```

деп өзгөртө турган болсок, анда көбөйтүү операциясынан башта кошуу операциясы аткарылат, анткени ал кашаанын ичине алынган.

Консолдук кийирүү/чыгаруу операциялары

Консолго чыгарууну жогоруда карап чыктык б.а. маалыматтардын агымын чыгаруу үчүн ***System*** классында ***out*** объекти аныкталган. Бул объекттин ***println*** методу маанилерди консолго чыгарып курсорду жаңы жолчого өткөрүп коет. Мисалы төмөнкү кодду карап көрөлү.

```
public class Program {

   public static void main(String[] args) {
      int x=5;
      System.out.println("x=");
      System.out.println(x);
   }
}
```

Программаны иштете турган болсок, консолдо төмөнкү жыйынтык алынат:

```
x=
5
```

Зарыл болсо, курсорду кийинки жолчого өткөрбөй койсок деле болот. Бул учурда, сиз ***System.out.print()*** инструкциясын колдонсоңуз болот, мисалы жогорудагы мисалды өзгөртүп көрөлү:

```
public class Program {

   public static void main(String[] args) {
         int x=5;
      System.out.print("x=");
      System.out.print(x);
   }
}
```

Биз төмөнкү жыйынтыкты алабыз:

```
x=5
```

Ошондой эле ***System.out.print("x="+x+"\n");*** инструкциясы менен бир эле жолчого "*x=5*" туюнтмасын чыгарып, *"\n"* жардамында курсорду жаңы жолчого өткөрүп алсак болот.

Мындан сырткары Java тилинде C тилинен мурасталып келген формattалган чыгаруу инструкциясы бар. Ал төмөкүдөй жазылат:

```
public class Program {

   public static void main(String[] args) {
      int x=2;
      int y=5;
      System.out.printf("x=%d; y=%d \n", x, y);
   }
}
```

Бул жерде %d символдору спецификаторду билдирет, анын ордуна аргументтердин бири алмаштырылат. Бизде 2 аргумент бар, алар бүтүн сандар. Ошондуктан инструкцияда эки %d символдору турат жана d символу ушул жерге бүтүн жазылаарын көрсөтүп турат

б.а. биринчи %d турган жерге x тин мааниси, экинчи %d турган жерге y тин мааниси келип жайгашат.

Жыйынтыгынла төмөнкүнү алабыз:

```
x=2; y=5
```

d символунун ордуна аргументтин тибине жараша төмөнкү символдорду колдонсок болот:

- %x: он алтылык сандарды чыгаруу үчүн
- %f: бөлчөк сандарды чыгаруу үчүн
- %e: экспоненциалдык формадагы сандарды чыгаруу, мисалы, 1.3e+01
- %c: бир символду чыгаруу үчүн
- %s: жолчолорду чыгаруу үчүн

Мисалы,

```
public class Program {

   public static void main(String[] args) {

      String name = "Akmat";
      int age = 30;
      float height = 1.7f;
System.out.printf("Name: %s  Age: %d  Height: %.2f \n", name, age,
height);
   }
}
```

кодду иштетсек, жыйынтыгы төмөнкүдөй болот.

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

Инструкциядагы *%.2f* жазуусу консолго бөлчөк сандын чыгуусун жана үтүрдөн кийин 2 сан чыгаарын көрсөтүп турат.

Ошентип, консолго каалагандай маалыматты чыгарып алууну билдик.

Эми, биз маалыматты клавиатуранын жардамында кийирүүнү карап көрөлү. Консолдон кийирүү үчүн, System классында in объекти аныкталган. Бирок, System.in объекти менен түздөн-түз иштөө өтө ыңгайлуу болбогондуктан, алар Scanner классын колдонушат жана ал өз кезегинде System.in ди колдонот. Мисалы, сандарды кийирүүчү программа жазалы.

```
import java.util.Scanner;

public class Program {

   public static void main(String[] args) {

      Scanner in = new Scanner(System.in);
      System.out.print("Санды кийириниз: ");
      int num = in.nextInt();

      System.out.printf("Сиздин сан: %d \n", num);
      in.close();
   }
}
```

Scanner классы java.util пакетинде болгондуктан, биз аны адегенде import java.util.Scanner оператору аркылуу импорттоп алдык. Негизги методдун ичинде Scanner классынын бир объекттин түзүп алдык. Ал үчүн Scanner классынын конструкторуна System.in деп жөнөттүк. Кийинки жолчодо "Санды кийириниз: " сөзүн консолго чыгарып алдык. Андан кийин, nextInt методун пайдаланып консолдон бүтүн санды окуп, num өзгөрмөсүнө жөнөттүк. Кийинки

жолчодо форматталган чыгарууну колдонуп, кийирилген маанини кайрадан чыгарып алдык. Жыйынтыгы төмөнкүдөй болот:

```
Санды кийириниз:
Сиздин сан: 5
```

Scanner классын төмөнкү методдору тиешелүү типтеги маанилерди кийирүү үчүн колдонулат:

- **next()**: биринчи пробелге чейинки жолчону окуйт
- **nextLine()**: кийирилген жолчону толук окуйт
- **nextInt()**: Int тибиндеги маанилерди окуйт
- **nextDouble()**: double тибиндеги маанилерди окуйт
- **nextBoolean()**: boolean тибиндени маанилерди окуйт
- **nextByte()**: byte тибиндеги маанилерди окуйт
- **nextFloat()**: float тибиндеги маанилерди окуйт
- **nextShort()**: short тибиндеги маанилерди окуйт

Math классы жана анын методдору

Math классы java.lang пакетинде жайгашкан жана бир катар математикалык эсептөөлөрдү жүргүзүү үчүн статикалык методдордун жыйындысын камсыз кылат.

Төмөндө Math классынын методдорун колдонуу ыңгайлуу учурлар келтирилди:

- *абсолюттук маанилерди эсептөө (модуль Math.abs())*
- *тригонометриялык функциялардын маанилерин эсептөө (синус, косинус ж.б. Math.sin(),...)*
- *ар кандай даражаларга көтөрүү (a^b - Math.pow(a,b))*
- *ар кандай даражадагы тамырларды алуу (Math.sqrt())*
- *кокус сандарды түзүү (Math.random())*

- *тегеректөө ж.б. (Math.round())*

Төмөнкү мисалды карап көрөлү:

Абсолюттук маанилерди эсептөөдө колдонсок болот.

```
public static void main(String[] args) {
    System.out.println(Math.abs(-1));     // 1
    System.out.println(Math.abs(-21.8d)); // 21.8
    System.out.println(Math.abs(4532L));  // 4532
    System.out.println(Math.abs(5.341f)); // 5.341
  }
```

Ар кандай даражага көтөрүү үчүн колдонсок болот.

```
public static void main(String[] args) {
    System.out.println(Math.pow(1,2)); // 1.0
    System.out.println(Math.pow(2,2)); // 4.0
    System.out.println(Math.pow(3,2)); // 9.0
    System.out.println(Math.pow(4,2)); // 16.0
    System.out.println(Math.pow(5,2)); // 25.0
     }
```

Бышыктоочу суроолор:

1. Өзгөрмө менен туракгуунун айырмасын мисалдар менен айтып бер?
2. Сенин жашооңдө өзгөрмөлөр барбы? Аларды кантип аныйтайсын?
3. Типтерге Java тилинде өтө каттуу көңүл бурулат . Кошуласынбы?
4. Арифметикалык операциялар кайсылар?
5. Бул темада сага эмне түшүнүксүз болду? (Түшүнбөгөн жерди шашпай кайталап окуп көрүңүз)

2-практикалык сабак

Максаты: Өзгөрмөлөр, турактуулар, берилгендердин типтери, арифметикалык операциялар жөнүндө алган маалыматтарды бышыктоо жана маселелерди чыгаруу

Келгиле, чогуу төмөнкү маселени карап көрөлү:

Маселе 1. **Алмаларды бөлүштүрүү**

Саны Nге барабар болгон мектеп окуучулары К алманы тең бөлүп алышат, бөлүнбөгөн калганы себетте калат. Ар бир мектеп окуучусу канча алма алат?

Кийирилүүчү маалымат

N (окуучулардын саны) жана K (себеттеги алмалардын саны)

Чыгуучу маалымат

Ар бир окуучу алган бүтүн сандагы алмалардын саны

Мисалы,

Кийирилүүчү маалымат

3

14

Чыгуучу маалымат

4

Чыгаруу:

Маселеде N окуучу K сандагы алманы бөлүп алышат. Мисалда көрсөтүлгөн маалыматтарды карасак N=3, K=14. 14 алманы 3 окуучуга бөлсөк 14/3=4(2) б.а. 4 бүтүндөн тийип 2 алма себетте калды.

Программаны түзүп көрөлү.

```
package progs;

import java.util.Scanner;
public class Example2 {

	public static void main(String[] args) {
		int N;
		int K;
		int d; //Ар бир окуучу алуучу алманын саны
		Scanner obj=new Scanner(System.in);
		N=obj.nextInt();
		K=obj.nextInt();
		d=K/N;
		System.out.println(d);
		obj.close();// объектти жабуу
	}
}
```

Жыйынтыгы:

```
5
14
4
```

Өз алдынча иштөөгө мисалдар

1. Гипотенуза

a жана ***b*** сандары берилсин жана бул сандар тик бурчтуу үч бурчтуктун катеттерин билдирет. Үч бурчтуктун гипотенузасын тапкыла.

Кийирилүүчү маалымат

Эки сан эки жолчого кийирилет. (1000ден ашпаган оң бүтүн сандар кийирилет).

Чыгуучу маалымат

Маселенин жообун чыгаргыла

Мисалы,

Кийирилүүчү маалымат

3
4

Чыгуучу маалымат
5.0

2. Цифралардын суммасы

Үч орундуу сан берилсин. Анын цифраларынын суммасын тапкыла.

Кийирилүүчү маалымат

Үч орундуу сан кийирилет

Чыгуучу маалымат

Маселенин жообун чыгаргыла

Мисалы,

Кийирилүүчү маалымат
456

Чыгуучу маалымат
15

3. Сабактардын аягы

Кээ бир мектептерде сабак саат 9:00дө башталат. Сабактын узактыгы - 45 мүнөт. 1, 3, 5-, ж.б. сабактардан кийин 5 мүнөттүк тыныгуу, ал эми 2, 4, 6- ж.б. сабактардан кийин - 15 мүнөттүк тыныгуу берилет. Көрсөтүлгөн сабак качан бүтөрүн аныктаңыз.

Кийирилүүчү маалымат

Сабактын номери берилет (1 ден 10 чейинки сан)

Чыгуучу маалымат

Эки бүтүн сан чыгат: сабактын аякташы саат жана минута менен

Мисалы:

Кийирилүүчү маалымат
3
Чыгуучу маалымат

11 35

Кийирилүүчү маалымат
2
Чыгуучу маалымат
10 35

4. Кийинки жуп сан

N бүтүн саны берилсин. Кийинки жуп санды басып чыгарыңыз.

(*Бул маселени чечүүдө шарттуу if операторлорун жана циклдерин колдоно албайсыз.*)

Кийирилүүчү маалымат

Натуралдык сан кийирилет

Чыгуучу маалымат

Маселенин жообун чыгарыңыз

Мисалы:

Кийирилүүчү маалымат
5
Чыгуучу маалымат
6

Кийирилүүчү маалымат
6
Чыгуучу маалымат
8

5. Сатып алуулардын баасы

Ашканада булочка a сом жана b тыйын сатылат. n булочка үчүн канча сом жана канча тыйын төлөнө тургандыгын аныктагыла.

Кийирилүүчү маалымат

Үч сан кийирилет: a,b,n

Чыгуучу маалымат

Эки сан чынат:төлөнүүчү сумма сом жана тыйын менен.

Мисалы:

Кийирилүүчү маалымат

10

15

2

Чыгуучу маалымат

20 30

6. Симметриялык сан

Төрт орундуу сан берилет. Берилген сан симметриялуу экендигин аныктаңыз. Эгерде сан симметриялуу болсо, анда 1ди, болбосо каалаган бүтүн санды басып чыгарыңыз. Санда төрт цифрадан азыраак болушу мүмкүн, анда кийирилген сандын сол жагына маанилүү змес нөлдөрдү кошуп алуу керек.

Бул маселени чечүүдө шарттуу if операторлорун жана цикл колдоно албайсыз

Кийирилүүчү маалымат

Төрт орундуу сан

Чыгуучу маалымат

Маселенин жообун чыгарыңыз

Мисалы:

Кийирилүүчү маалымат
1001
Чыгуучу маалымат
1
Кийирилүүчү маалымат

3007 ***Чыгуучу маалымат*** 2

Лекция 4. Шарттуу конструкциялар



Максаты:

1. *Логикалык операцияларды, салыштыруу амалдарын үйрөнүү*
2. *if ... else, switch ... case конструкцияларын үйрөнүү*
3. *Тернардык операторлорду, программалык коддун блокторун үйрөнүү*
4. *Өзгөрмөлөрдүн таасир этүү чөйрөсү жана жашоо узактыгын билүү*

Логикалык операциялар

Логикалык операциялар логикалык операторлордун жардамында аткарылат.

Төмөнкү мисалды карап көрөлү:

$$(a \mid b) \mid (c < 100) \ \& \ !(true) \ \hat{} \ (q == 5)$$

Бул төрт операнддан турган татаал логикалык амалды көрсөтүп турат. Алар төмөнкүлөр:

1. (a | b), мында a,b —boolean тибиндеги өзгөрмөлөр.
2. (c < 100)
3. (true)

4. (q == 5)

Ошол эле учурда жөнөкөй гана (a | b) логикалык амалы эки аргумент-операндан турат.

5. Логикалык операнд true же false маанилеринин бирин кайтаруучу логикалык амал болуп эсептелет. Мисалы,

- (2 < 1) – логикалык операнд жана анын мааниси false.
- true- логикалык операнд жана анын мааниси true.
- int a = 2 -логикалык операнд эмес, int тибиндеги өзгөрмө.
- String a = "true" - логикалык операнд эмес, бул true маанисин сактоочу жолчо.

Javada төмөнкүдөй логикалык операциялар бар:

- **Логикалык тануу оператору !**

Бул оператор унардык, б. а. ал бир логикалык туюнтмага же операндга колдонулат. Бул оператор амалды тескери маанисине өткөрөт (таблица 5).

Таблица 5. Тануу операторунун иштөөсү

а нын мааниси	**!a**
false	true
true	false

Тануу операторун программада карап көрөлү:

```
public class Example3 {
  public static void main(String[] args) {
   boolean a = true;
   System.out.println(!a);
   // Бул жерде а нын тескери мааниси false экранга чыгат.
```

```
    System.out.println(!false);
    // !false амалы true амалын кайтарат
    System.out.println(!(2 < 5));
// (2 < 5) амалы чын(true) болчу, анын тескериси жалган (false) экранга чыгат
  }
}
```

Программанын жообу төмөнкүдөй болот:

```
false
true
false
```

- **Логикалык ЖАНА & (логикалык көбөйтүү), ошондой эле && операторлору**

Логикалык ЖАНА же конъюнкция эки туюнтмага колдонулат жана анын ишинин жыйынтыгы эки операнд тең чын болгондо гана чын(true) маанисин кайтарат (таблица 6).

Таблица 6. &(ЖАНА) операторунун иштөөсү

a	b	a & b
true	true	true
true	false	false
false	true	false
false	false	false

Логикалык ЖАНА (&) операторун программада карап көрөлү:

```
public class Example4 {
  public static void main(String[] args) {
    boolean a = true;
    boolean b = false;
    boolean c = true;
```

```
        System.out.println(a & b);
// эгерде чынды(true) жалганга(false) көбөйтсөк жалган(false) чыгат.
        System.out.println(a & c);
// чынды(true) чынга(true) көбөйтсөк чын(true) чыгат
        System.out.println(false & (2 > 5));
// жалганды (false) жалганга(false) көбөйтсөк жалган(false) чыгат
            System.out.println((2 < 5) & false);
 // (2<5) амалы true маанисин кайтарат, аны false'ка көбөйтсөк false чыгат
    }
}
```

Программанын жыйынтыгы:

```
false
true
false
false
```

&& операторун кээде "кыска ЖАНА" деп аташат. Ал логикалык операнддар менен иштегенде & оператору сыяктуу эле жыйынтыкты берет. Бирок, анын ишинин өзүндө айырма бар. Демек, сиз буга чейин байкагандай эле, эгерде (a & b) туюнтмасында a операндасы жалган болсо, анда b операндынын маанисин текшерүүнүн эч кандай мааниси жок, анткени жыйынтыгы сөзсүз жалган болот. Ошентип, эгер бизге экинчи операнддын мааниси керек эмес болсо, анда && колдонуу менен биз программадагы эсептөөлөрдүн санын азайтабыз. Мисалдагы бардык & операторлорун && менен алмаштырсак, натыйжа так ошондой болот, бирок программанын өзү бир аз тезирээк иштейт.

- **Логикалык ЖЕ | (логикалык кошуу), ошондой эле || операторлору**

Java тилинде логикалык ЖЕ оператору "|" символу менен көрсөтүлөт. Логикалык ЖЕ же дизъюнкция эки туюнтмага колдонулат жана анын натыйжасы эки операнд тең жалган болгондо гана жалган болот. Бул жерде биз кандайдыр бир деңгээлде & операторундагыдай эле сүрөттү көрөбүз, бирок карама-каршы. Башкача айтканда, жок дегенде бир операнд чын болсо, анда a | b экинчи билдирүүнүн маанисине карабастан чындыкка кепилдик берилет (таблица 7).

Таблица 7. | (ЖЕ) операторунун иштөөсү

a	b	a | b
true	true	true
true	false	true
false	true	true
false	false	false

Логикалык ЖЕ (|) операторун программада карап көрөлү:

```
public class Example5{
  public static void main(String[] args) {
     boolean a = true;
     boolean b = false;
     boolean c = true;
     System.out.println(!a | b);
//!a амалы false маанисин кайтарат, ЖЕ операторунан да false жообу алынат
     System.out.println(a | c);
     System.out.println((2 < 5) | false);
//(2<5) амалы чын маанисин кайтарат, ошондуктан жыйынтык чын болот
     System.out.println((2 > 5) | true);
```

```
  }
}
```

Программанын жыйынтыгы төмөнкүдөй болот:

```
false
true
true
true
```

Эгерде биз | операторунун ордуна || операторун колдонсок, так ушундай жыйынтыкка ээ болобуз, бирок && учурундагыдай эле, ал үнөмдүү иштейт б.а. биринчи операнд чын(true) маанисин кайтраса, экинчи операндды текшерүүнүн кажети жок болот, анткени баары бир чын(true) мааниси алынат.

- **Логикалык өзгөчө ЖЕ (XOR) ^**

XOR, логикалык кемитүү, катуу дизъюнкция – булар ^ операторунун аталышы. Бул операторду эки операндга колдонууда true маанисин кайтарат, эгерде операнддар ар түрдүү болсо жана false маанисин операнддар бирдей болсо кайтарат (таблица 8) .

таблица 8. ^ (XOR) операторунун иштөөсү

Boolean a	Boolean b	a ^ b
true	true	false
true	false	true
false	true	true
false	false	false

Төмөнкү программаны карап көрөлү:

```
public class Example6{
  public static void main(String[] args) {
    boolean a = true;
    boolean b = false;
    boolean c = true;
```

```
    System.out.println(!a ^ b);
// Эки логикалык операторду колдонууну карайлы: a == true, !a болсо биз
//билгендей, false дегенди билдирет.
    System.out.println(a ^ c);
    System.out.println((2 < 5) ^ false);
    System.out.println((2 > 5) ^ true);
  }
}
```

Программанын жыйынтыгы:

```
false
false
true
true
```

Логикалык операциялар да приоритетине жараша тартип менен аткарылат. Ал төмөнкүдөй:

1. !
2. &
3. ^
4. |
5. &&
6. ||

Мисалда карап көрөлү.

```
public class Example7 {
  public static void main(String[] args) {
    boolean a = true, b = true, c = false;
    System.out.println(a | b & c);
}
```

Бул жерде (a | b & c) амалын солдон оңго карай аткарсак анда false маанисин алабыз. Бирок & операциясынын приоритети жогору болгондуктан бул жерде биринчи b & c аткарылат. Мында false жообу чыгат, андан кийин a | b амалында true|false=true келип чыгат. Ошондуктан операциялардын приоритетин билүү маанилүү!

Салыштыруу операциялары

Салыштыруу операцияларында эки операнд салыштырылат, туюнтма чын болгон учурда boolean тибинин true маанисин, жалган болгон учурда false маанисин кайтарат. Салыштыруу операцияларына төмөнкүлөр кирет:

❖ == **барабар**

Эки операндды салыштырат да true маанисин(эгер операнддар барабар болсо) же false маанисин (эгер операнддар бирдей болбосо) кайтарат. Операциянын иштөөсү төмөнкү мисалда көрсөтүлдү:

```
int a = 10;
int b = 4;
boolean c = a == b;    // false
boolean d = a == 10;   // true
```

❖ != **барабар эмес**

Эки операндды салыштырып барабар эмес болсо true маанисин, барабар болсо false маанисин кайтарат.

```
int a = 10;
int b = 4;
boolean c = a != b;     // true
boolean d = a != 10;    // false
```

❖ < **кичине**

Эгерде биринчи операнд экинчи операнддан кичине болсо, анда true, антпесе false маанисин кайтарат.

```
int a = 10;
int b = 4;
boolean c = a < b;   // false
```

❖ **> чоң**

Эгерде биринчи операнд экинчисинен чоң болсо, анда true маанисин, антпесе false маанисин кайтарат.

```
int a = 10;
int b = 4;
boolean c = a > b;   // true
```

❖ **>= чоң же барабар**

Эгерде биринчи операнд экинчисинен чоң же барабар болсо, анда true маанисин, антпесе false маанисин кайтарат.

```
boolean c = 10 >= 10;   // true
boolean b = 10 >= 4;    // true
boolean d = 10 >= 20;   // false
```

❖ **<= кичине же барабар**

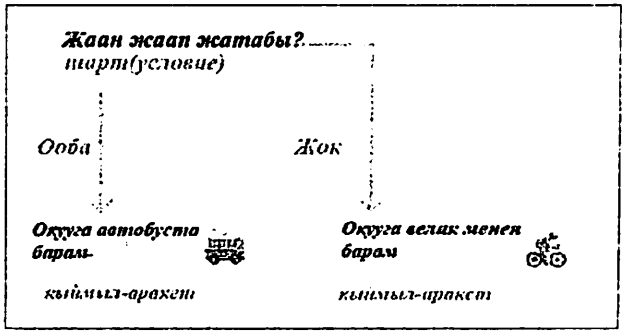
Эгерде биринчи операнд экинчисинен кичине же барабар болсо, анда true маанисин, антпесе false маанисин кайтарат.

```
boolean c = 10 <= 10;   // true
boolean b = 10 <= 4;    // false
boolean d = 10 <= 20;   // true
```

Биз жогоруда карап өткөн логикалык операцияларды жана салыштыруу операцияларын функционалдуу колдонуу үчүн Java тилинде атайын if …else, switch…case конструкциялары бар. Аларды карап көрөлү.

if … else конструкциясы

Биздин жашообузда ар кандай чечимдерди кабыл алабыз. Алар кайсы бир шартка жараша кабыл алынат. Мисалы, эгерде сыртта абаырайы жаанчыл болсо, анда автобуска чыгабыз, антпесе окууга велик менен барабыз (15-сүрөт).



15-сүрөт. Шарттын аткарылуусу

Java тилинде да ушул сыяктуу чечимдерди кабыл алуу үчүн шарттуу констукциялар колдонулат. Алар if/else жана switch/case консрукциялары.

if/else оператору белгилүү бир шарттын чындыгын текшерет жана текшерүүнүн жыйынтыгына жараша белгилүү бир кодду аткарат:

```
int num1 = 6;
int num2 = 4;
if(num1>num2){
   System.out.println("Биринчи сан экинчиден чоң");
}
```

Жогорудагы мисалды эки бүтүн сан салыштырылып жатат (6 жана 4), эгерде шарт true маанисин кайтарса (6>4), анда шарттан кийин жайгашкан операторлор иштейт да экранга “Биринчи сан экинчиден чоң” маалыматы чыгат, антпесе биздин учурда эч нерсе аткарылбайт.

Java тилинде if/else конструкциясынын синтаксиси төмөнкүдөй (16-сүрөт):

```
if (  шарт  )
    команда1 ;
else
    команда2 ;
```

16-сүрөт. if/else конструкциясынын синтаксиси

Шарт ар дайым кашанын ичинде жазылат. Эгерде берилген шарт true маанисин кайтарса, анда бир канча операторлор иштей турган болсо, алардын баарын фигуралык кашанын ичинде жазабыз. Мисалы,

```
int age = 17;
if (age < 18)
  System.out.println("Сиз жаш баласыз");
else
  System.out.println("Сиз чоң адамсыз");
```

Программалык коддун блоктору

Шарт аткарылганда (же аткарылбаганда) программаңыз бир нече буйруктарды аткарышын кааласаңыз, аларды код блогуна бириктире аласыз. Код блогун фигуралык кашанын ичине жазабыз(17-сүрөт).

```
{
    команда1 ;
    команда2 ;
    команда3 ;
}
```

17-сүрөт. Код блогунун түзүлүшү

Мисалы жогоруда каралган программалык кодду өзгөртүп көрөлү:

```
int age = 17;
if (age < 18)
{
  System.out.println("Сиз жаш баласыз");
  System.out.println("Чоңдор менен талашпа");
}
else
{
  System.out.println("Сиз чоң адамсыз");
  System.out.println("Балалык убактыңыз аяктады");
}
```

Бул жерде шарт аткарылса эки оператор иштейт, ошондой эле шарт аткарылбаса да эки оператор иштейт.

if/else конструкциясы бир канча шарттар үчүн да колдонулат. Мисалы 3 сандын чоңун табуучу программа түзүп көрөлү.

```
public class Example8 {
  public static void main(String[] args) {
          int a=7;
         int b=8;
          int c=2;
         int max;
//эгерде a саны b жана c санынан чоң болсо, анда эң чоң сан a болот
        if (a>b & a>c)
                max=a;
// антпесе a саны чоң эмес болгондуктан, эң чоң санды b жана c сандарынан
// издейбиз
        else if (b>c)
                max=b;
            else
                max=c;
```

```
        System.out.println(max);
}
```

Жыйынтыгы

```
8
```

Switch конструкциясы

switch/case конструкциясы көп бутактуу тандоону камсыз кылат. Ошентип, ал программага бир нече альтернативалардын арасынан тандоого мүмкүндүк берет. if/else операторлорун бир канча жолу пайдалануу менен көп тандоону уюштурсак болот, бирок көп учурда switch/case натыйжалуураак ыкма болуп саналат б.а. switch/case конструкциясы if/else конструкциясына окшош, анткени ал бир эле учурда бир нече шарттарды иштетүүгө мүмкүндүк берет. switch туюнтмасы char, byte, short же int тибинде болушу мүмкүн (float тибиндеги өзгөрмөлөр болбойт), case бөлүктөгү маанилердин типтери да ага шайкеш келүүсү керек.

```
int user_num = 8;
switch(user_num){

   case 1:
      System.out.println("Колдонуучу 1 санын кийирген");
      break;
   case 8:
      System.out.println("Колдонуучу 8 санын кийирген");
      user_num++;
      break;
   case 9:
      System.out.println("Колдонуучу 9 санын кийирген ");
      break;
   default:
      System.out.println("Кийирилген сан 1,8,9 эмес");
}
```

Жогорудагы программалык коддо switch ачкыч сөзүнөн кийин салыштырыла турган амал кашаанын ичинде жазылат. Бул туюнтманын мааниси тиешелүү түрдө case операторунан кийин коюлган маанилер менен салыштырылат. Ал эми дал келүү табылса, тиешелүү код блогу аткарылат. Тактап айтсак, программада user_num өзгөрмөсүнүн мааниси 8 ге барабар. Кийинки жолчодо switch оператору user_num өзгөрмөсүн текшерет. Экинчи case'те 8 мааниси болгондуктан, тиешелүү блок аткарылат жана экранга "Колдонуучу 8 санын кийирген" деген маалымат чыгат жана user_num өзгөрмөсүнүн мааниси 1 ге өсөт. break оператору switch операторунун жумушун токтотот. Эгерде break жок болсо, кийинки case уланып кетет.

Эгерде case ачкыч сөзүнөн кийинки маанилер салыштырылуучу амалдын мааниси менен бир да дал келбесе, анда default бөлүктөгү блок иштейт.

Төмөнкү мисалда көрсөтүлгөндөй, сизде бош учурлар болушу мүмкүн:

```
switch(i) {
        case 1:
        case 2:
        case 3: System.out.println("i бул 1, 2 же 3"); break;
        case 4: System.out.println("i бул 4"); break;
        }
```

Мындан сырткары камтылуучу switch да колдонулат:

```
switch(ch1) {
      case 'A': System.out.println("Бул сырткы A");
              switch(ch2) {
```

```
            case 'A':
                    System.out.println("бул ички А");
                        break;
                case 'B': // ...
                } // ички switch'тин аягы
        break;
    case 'B': // ...
```

Тернардык оператор

Тернардык оператор төмөнкү синтаксиске ээ:

[биринчи операнд - шарт] ? [экинчи операнд] : [үчүнчү операнд].

Ошентип, бул операцияга үч операнд катышат. Шартка жараша тернардык оператор экинчи же үчүнчү операндды кайтарат: эгерде шарт чын болсо (true), анда экинчи операнд кайтарылат; шарт жалган (false) болсо, анда үчүнчү операнд кайтарылат. Мисалы:

```
int x=3;
int y=2;
int z = x<y? (x+y) : (x-y);
System.out.println(z);
```

Бул жерде тернардык операциянын жыйынтыгы болуп z өзгөрмөсү саналат. Биринчиден, x<y шарты текшерилет. Эгерде шарт аткарылса, анда z экинчи операндга - (x+y) барабар болот, антпесе z үчүнчү операндга барабар болот.

Биздин учурда 3<2 шарты аткарылган жок, ошондуктан z=(x-y) амалы аткарылат жана z=3-2=1 болуп, экранга 1 чыгат.

Өзгөрмөлөрдүн таасир этүү чөйрөсү жана жашоо узактыгын билүү

Төмөнкү жөнөкөй программанын мисалында өзгөрмөлөрдү инициализациялоо, башкаруу жана программалоо каталарынан кантип сактануу керектигин көрсөк болот.

```
class Example9 {
    public static void main(String args[]) {
      int x; // main методунун ичинде бардык жерде белгилүү
    x = 10;
    if(x == 10) { // жаңы чөйрө башталды
      int y = 20; // ушул блокто гана белгилүү
      // x жана y экөө тең бул жерде белгилүү
      System.out.println("x жана y: " + x + " " + y);
      x = y * 2;
      }
   y = 100; // Ката! бул жерде y белгисиз
   // x дагы эле бул жерде белгилүү
   System.out.println("x = " + x);
   }
}
```

Бул мисалда кээ бир өзгөрмөлөр атайын учур үчүн гана түзүлүп, кийин колдонулбайт. Эгерде колдонсок, программа аны тааныбай калат.

Бышыктоочу суроолор:

1. Логикалык кайсы операцияларды үйрөндүк?
2. Салыштыруу амалдары жөнүндө айтып берчи?
3. if ... else, switch ... case конструкцияларын колдонуунун өзгөчөлүктөрү?
4. Тернардык операторду кантип колдонобуз?
5. Өзгөрмөлөрдүн колдонуу узактыгы, таасир этүү чөйрөсү кандай?

6. Жашоондон шарт менен байланышкан мисал келтирчи жана аны өзгөртүп көрчү?

3-практикалык сабак

Максаты: Салыштыруу, логикалык операциялар, if/else, switch/case конструкциялары жөнүндө алган маалыматтарды бышыктоо жана маселелерди чыгаруу

Келгиле, чогуу төмөнкү маселени карап көрөлү.

Маселе 1. **Сандын белгиси**

Математикада sign(x) (сандын белгиси) функциясы төмөнкүчө аныкталат:

sign(x) = 1 болот, эгерде x > 0 болсо,

sign(x) = -1 болот, эгерде x < 0 болсо,

sign(x) = 0 болот, эгерде x = 0 болсо.

Берилген x саны үчүн sign(x) функциясынын маанисин чыгаргыла.

Кийирилүүчү маалымат

x саны кийирилет

Чыгуучу маалымат

Маселение жообу

Мисалы:

Кийирилүүчү маалымат
5
Чыгуучу маалымат
1

Чыгаруу:

Маселеде кандайдыр бир сан кийирилсе, анын белгиси кандай экендигин сан түрүндө кайтаруу керектиги айтылып жатат б.а. эгерде оң сан кийирилсе, экранга 1 деген сан чыгуусу керек.

Кийирилген сан терс сан болсо, -1 саны, ал эми кийирилген сан 0 болсо, экранга 0 чыгуусу керек. Ушул факттарды эске алып, программаны түзүп алалы:

```
import java.util.Scanner;

public class Example10 {

        public static void main(String[] args) {

                Scanner obj=new Scanner(System.in);
                int x=obj.nextInt();
                int sign;
                if (x>0)
                        sign=1;

                else if (x<0)
                                sign=-1;
                        else
                                sign=0;

                System.out.println(sign);
                obj.close();

        }
}
```

Жыйынтыгы:

```
5
1
```

Өз алдынча иштөөгө мисалдар

1. Ладья

Шахмат талаасында көрсөтүлгөн координатада (жолчонун номери жана мамычанын номери) жайгашкан ладья фигурасы, координатасы менен берилген экинчи бир фигурага коркунуч жаратабы же жокпу аныктоо талап кылынат.

Кийирилүүчү маалымат

Төрт сан кийирилет: ладья финурасынын координаталары жана экинчи фигуранын координаталары. Координаталар 1ден 8ге чейинки бүтүн сандар.

Чыгуучу маалымат

Эгерде ладья экинчи фигураны бир ход менен жей алса, анда экранга "**Yes**" сөзүн, жей албаса "**No**" сөзүн чыгаруу керек.

Мисалы,

Кийирилүүчү маалымат
1 1 2 2
Чыгуучу маалымат
No

Кийирилүүчү маалымат
1 1 2 1
Чыгуучу маалымат
Yes

2. Слон

Шахмат талаасында көрсөтүлгөн координатада (жолчонун номери жана мамычанын номери) жайгашкан слон фигурасы, координатасы менен берилген экинчи бир фигурага коркунуч жаратабы же жокпу аныктоо талап кылынат.

Кийирилүүчү маалымат

Төрт сан кийирилет: слон финурасынын координаталары жана экинчи фигуранын координаталары. Координаталар 1ден 8ге чейинки бүтүн сандар.

Чыгуучу маалымат

Эгерде слон экинчи фигураны бир ход менен жей алса, анда экранга "Yes" сөзүн, жей албаса "No" сөзүн чыгаруу керек.

Мисалы,

Кийирилүүчү маалымат
1 1 5 5
Чыгуучу маалымат
Yes

Кийирилүүчү маалымат
3 2 2 2
Чыгуучу маалымат
No

3. Тендеме

$ax + b = 0$ тендемесин бүтүн сандар менен чечиңиз.

Кийирилүүчү маалымат

2 бүтүн сандарды киргизиңиз: a жана b

Чыгуучу маалымат

Эгерде алардын саны чектүү болсо, бардык чечимдерди, чечимдер жок болсо “NO” (тырмакчасыз) жана чексиз көп чечимдер болсо “INF” (тырмакчасыз) басып чыгаруу керек.

Мисалы,

Кийирилүүчү маалымат
6
-2
Чыгуучу маалымат
NO

Кийирилүүчү маалымат
1
1
Чыгуучу маалымат
-1

4. Метрого билет

Метродо бир сапардын билети 15 сом, 10 жолку билет 125 сом, 60 жолку билет 440 сом турат. Жүргүнчү n сапар жасоону пландаштырууда.

Сатылып алынган билеттердин жалпы баасы минималдуу жана төлөнүүчү сапарлардын жалпы саны жок дегенде n болушу үчүн, жүргүнчү ар бир түрдөгү билеттен канча сатып алышы керектигин аныктаңыз.

5. Телефон

Асандын 2000 сом акчасы бар. Ал өзүнө атайын магазинден насыяга баасы k (k>2000) болгон телефон алууну чечти. Магазин Асанга баштапкы төлөнүүчү сумманы айтты. Ал сумма k баштапкы баанын 30 пайызын түзөт. Эгерде телефондун баасы берилсе Асан болгон акчасы менен баштапкы төлөнүүчү сумманы магазинге бере алабы? Программасын түзгүлө.

6. Функциянын маанисин тапкыла

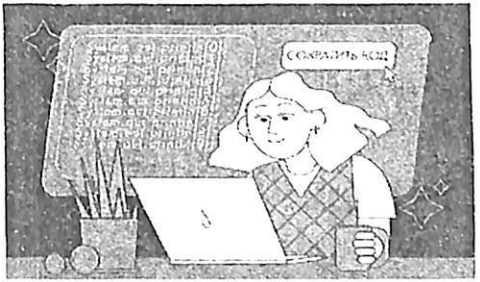
x чыныгы саны берилсин. *f(x)*ны эсептегиле.

$$f(x)=\begin{cases} 0, x \le 0 \text{ болсо} \\ x, \quad 0 \le x \le 1, \text{болсо} \\ x^4, \text{башка бардык учурда} \end{cases}$$

7. Шаардын коду

Шаарлардын кодун пайдаланып, 5 минутага созулган шаарлар аралык сүйлөшүүнүн баасын эсептеп чыгуучу программа түзгүлө. (Бишкек(312)-5.25 сом, Ош(3222)-3.43 сом, Баткен(3254)-4.30 сом, Жалал-Абад(37)-5.60 сом,Нарын (35)-4.75 сом, Талас(34)-4,34 сом, Чолпон-Ата (3943)-4.45 сом).

Лекция 5. Циклдык конструкциялар



Сабактын максаты :

1. *for циклдык конструкциясы,*
2. *do...while циклдык конструкциясы,*
3. *while циклдык конструкциясы,*
4. *break, continue операторлору,*
5. *Камтылуучу циклдар жөнүндө маалымат алуу.*

Жашообузда көбүнчө бир же бир нече иш-аракеттерди көп жолу жасоого туура келет. Мисалы, мен көп баракган турган документти сканерлешим керек. Бул учурда бир эле жумушту кайра-кайра аткарабыз: сканерге биринчи бетти коебуз, сканерлөө кнопкасын басабыз, кийинки бетти коебуз, сканерлөө кнопкасын басабыз ... Албетте бул жумушту аткаруу кыйын. Автоматташтырылган болсо кандай жакшы болмок!?

Бүгүнкү лекцияда биз Java тилинде кайталанган бир же бир нече операцияларды кантип автоматташтырууга болорун үйрөнөбүз.

Java тилинде кайталоонун өзгөчөлүгүнө жараша циклдардын төмөнкүдөй түрлөрү бар:

- for

- while
- do...while

for циклдык конструкциясы

for цикли – программанын алгоритм боюнча аткарылышын бузуп, көрсөтүлгөн кодду көп жолу аткарууга мүмкүндүк берүүчү программалык кодду башкаруучу структура десек болот.

For циклын төмөнкүдөй колдонобуз:

```
for(<баштапкы точка>; <шарт>; <итерация>) {
    // циклдын телосу
}
```

Мисалы 0дөн 5 ке чейинки сандарды катары менен консолго чыгарып көрөлү:

```
for(int i = 0; i < 5; i++) {
  System.out.println(i);
}
```

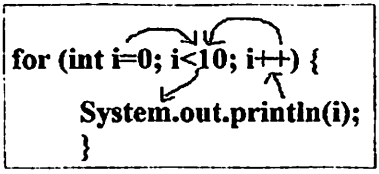
Жыйынтыгы:

```
0
1
2
3
4
```

Java тилинде жазылган бул фрагментти биздин табигый тилге которсок, анда мындай болот:

"Баштапкы мааниси 0 болгон i өзгөрмөсүн түзүңүз, ал 5ке жеткенче, ага 1ди кошуп, ар бир кадамда i өзгөрмөсүнүн маанисин консолго чыгарыңыз"

Цикл качан токтойт деп жатсаңыз, ал качан көрсөтүлгөн шарт аткарылбай калганда токтойт. for циклынын иштөөсүн схемада көрөлү (18-сүрөт):



18-сүрөт. for циклынын иштөө схемасы

Жогоруда көрсөтүлгөн схемада for циклынын иштөөсү так көрсөтүлүп турат. for ачкыч сөзүнөн кийин турган кашаанын ичиндеги туюнтмалар сөзсүз болуш керек деген талап жок б.а. сүрөт боюнча айтсак циклды башкаруучу өзгөрмө (int i;), циклды башкаруучу шарт (i<10), циклды алдыга жылдыруучу кадам (i++) жана циклдын телосу (System.out.println(i);) болуусу абзел. Бул программанын жыйынтыгында экранга 0дөн 10 го чейинки сандар чыгат. Экранга чыгаруу инструкциясы (System.out.println(i);) бир жолу жазылса да for циклынын арты менен ал 10 жолу кайталанып аткарылат.

Ушул эле программаны төмөнкүдөй жазып алсак болот:

```
int i=0;
for(; i<10; i++){
      System.out.println(i);
}
```

Бул жерде циклды башкаруучу өзгөрмө жана анын мааниси сыртта жарыяланды.

Эгерде шартты көрсөтпөй койсок же өзгөрмөнүн мааниси өзгөрбөсө, анда цикл чексиз болуп калат:

көпчүлүк учурда кайталоонун саны белгисиз болгон учурда колдонушат.

Төмөнкү маселени чыгарып көрөлү:

Маселе. Бизге N бүтүн саны берилсин. N санынан ашпаган 2 санынын даражаларын чыгаргыла.

Чыгаруу:

Бизге N саны берилсин, мисалы N = 50 деп алалы. Анда 50 санынан ашпаган 2 санынын даражаларын жазып чыгалы:

$2^0 = 1$

$2^1 = 2$

$2^2 = 4$

$2^3 = 8$

$2^4 = 16$

$2^5 = 32$

$2^6 = 64$

Биз 1 2 4 8 16 32 сандарын экранга чыгарабыз, 64 саны 50 санынан ашып кеткендиктен ошол жерден цикл токтош керек болот. Цикл канча жолу айланышын биз алдын ала айта албайбыз, ошондуктан while циклын колдонобуз.

```
import java.util.Scanner;
public class Example11 {

	public static void main(String[] args) {
		Scanner obj=new Scanner(System.in);
		int N=obj.nextInt();
		int k=1;
		while (k<N) {
			System.out.print(k+" ");
			k=k*2;
```

```
                obj.close();
            }

        }
}
```

Программаны текшерип көрүп, төмөнкү жыйынтыкты алдык:

```
1 2 4 8 16 32
```

continue жана break операторлору

break(сындыруу, токтотуу) операторун switch/case конструкцияларында салыштырууну токтотуу үчүн колдонгонбуз. Ошол сыяктуу эле break оператору цикл өз ишин аяктай элек болсо дагы, каалаган убакта циклден чыгууга мүмкүндүк берет:

```
for (int i = 0; i < 10; i++){
   if (i == 5)
      break;
   System.out.println(i);
}
```

Жыйынтык:

```
0
1
2
3
4
```

Бул жерде i өзгөрмөсүнүн мааниси 5 болгондо цикл токтойт, break оператору болбогондо i өзгөрмөсүнүн мааниси 10 го чейин жетмек.

Ушул эле мисалды кайрадан карап көрөлү. Бирок бул жолу i өзгөрмөсүнүн мааниси 5 болгондо цикл токтобой, кийинки итерацияга өтсүн дейли. Бул учурда break операторун continue (улантуу) операторуна алмаштырабыз.

```
for (int i = 0; i < 10; i++){
   if (i == 5)
      continue;
   System.out.println(i);
}
```

Жыйынтык:

```
0
1
2
3
4
6
7
8
9
```

Жыйынтыкта 5 санын көрбөй турабыз, анткени continue оператору кийинки маанiге аттап өтүүгө жардам берди.

Төмөнкү мисалды чыгарып көрөлү:

Мисал. y=1/x функциясынын маанилерин экранга чыгаргыла. x ∈[-3;3].

Чыгаруу. y=1/x функциясынын маанилерин табууда маселе келип чыгат. Кандай маселе десеңиз, бул x=0 болгондо y тин маанисин табуу үзгүлтүккө учурайт. Анткени санды 0 бөлүүгө болбойт. Ушул жерден continue операторун колдонуп, x=0 болгондо интерацияны кийинки маанiге өткөрүп жиберели. Макулсузбу?

```
public class Example12 {

    public static void main(String[] args) {
        double y;
        for (int x=-3;x<=3;x++) {
                if (x==0)
                    continue;
            y=1.0/x;
            System.out.println(y);

        }
    }
```

}

Программанын жыйынтыгы:

```
-0.3333333333333333
-0.5
-1.0
1.0
0.5
0.3333333333333333
```

Камтылуучу циклдар

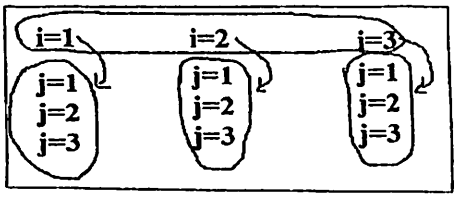
Камтылуучу циклдар жөнүндө айтпай кетсек болбойт. Анткени көпчүлүк маселелерди чыгарууда колдонулат. Төмөнкү маселени чыгарып көрөлү:

Маселе. $\sum_{i=1}^{3}\sum_{j=1}^{3} i+j$ туюнтмасынын маанисин эсептегиле.

Чыгаруу. Бул туюнтманы өзүбүзгө түшүнүктүү болгондой кылып жөнөкөйлөтүп жазып алалы:

$$\sum_{i=1}^{3}\sum_{j=1}^{3} i+j$$
$$= \big((1+1)+(1+2)+(1+3)\big)$$
$$+ \big((2+1)+(2+2)+(2+3)\big)$$
$$+ \big((3+1)+(3+2)+(3+3)\big) = 36$$

Бул жерде ички жана сырткы сумманы көрүп турабыз. Сырткы сумманы i өзгөрмөсү башкарып турат, ички сумманы j өзгөрмөсү башкарып турат б.а. 19-сүрөттө көрсөтүлгөндөй сырткы i дын ар бир маанисинде j 1ден 3 кө чейин өзгөрөт.



19-сүрөт. Камтылуучу цикл

Программада карап көрөлү:

```
public class Example13 {

    public static void main (String[] args) {
        int S=0;
        for (int i=1; i<=3; i++) {
            for (int j=1; j<=3; j++) {
                S=S+(i+j);
            }
        }
        System.out.println(S);
    }
}
```

сыртқы цикл

ички цикл

Жыйынтык:

```
36
```

Бышыктоочу суроолор:

1. Цикл түшүнүгү боюнча айтып берчи?
2. Жашоондон чектүү жана чексиз циклдарга мисал келтирчи?
3. for цикли менен while циклинин айырмасын байкадыңбы?
4. Циклды кантип токтотсок болот?
5. continue оператору эмне жумуш аткарат?

4-практикалык сабак

Максаты: for, do, while циклдары жөнүндө алган маалыматтарды бышыктоо жана маселелерди чыгаруу.

Келгиле, чогуу төмөнкү маселени карап көрөлү.

Маселе 1. **Квадраттардын тизмеси**

Берилген N санынан ашпаган натуралдык сандардын так квадраттарын басып чыгарыңыз.

Кийирилүүчү маалымат

N саны кийирилет

Чыгуучу маалымат

N санынан ашпаган натуралдык сандардын так квадраттары

Мисалы,

Кийирилүүчү маалымат

15

Чыгуучу маалымат

1
4
9

Чыгаруу:

Жогорудагы тестти карасак, 1^2=1, 2^2=4, 3^2=9 болуп, кийинки 4^2=16 саны 15 санынан чоң болуп кеткендиктен экранга 1, 4, 9 сандары гана чыкты.

Мында циклдын кайталануу саны белгисиз болгондуктан while циклын колдонуп түзөбүз:

```
public class Example13 {

    public static void main(String[] args) {
        Scanner obj=new Scanner(System.in);
        int N=obj.nextInt();
        int k=1;  //натуралдык сандардын квадрытын сактайт
        int i=1;  //натуралдык сандарды сактайт
        while (k<=N) {
            System.out.println(k);
            i++;  //кийинки натуралдык санга өтүү
            k=i*i;  //кийинки натуралдык сандын квадраты
```

```
        }

    }
}
```

Жыйынтык:



Өз алдынча иштөөгө мисалдар

1. Калдык

a, b, c, d төрт бүтүн саны берилсин. a дан b га чейинки интервалдагы сандарды d санына бөлүүдө c калдык калса экранга чыгаргыла.

Киргизилүүчү маалымат

a, b, c, d төрт бүтүн саны

Чыгуучу маалымат

a дан b га чейинки интервалдагы сандарды d санына бөлүүдө c

калдык калган сандар

Мисалы,

Киргизилүүчү маалымат

2
5
0
2

Чыгуучу маалымат

2 4

2. Цифралардын суммасы

x натуралдык саны берилсин. x санынын цифраларынын суммасын экранга чыгаргыла.

Киргизилүүчү маалымат

x саны

Чыгуучу маалымат

x санынын цифраларынын суммасы

Мисалы,

Кийирилүүчү маалымат

123

Чыгуучу маалымат

6

3. Нөлдөр

N сандагы бүтүн сандардын арасында канча нөл саны бар экендигин эсептегиле.

Кийирилүүчү маалымат

Биринчи жолчого N саны. Кийинки жолчолорго N сандагы бүтүн сандар.

Чыгуучу маалымат

Нөлдөрдүн саны

Мисалы,

Кийирилүүчү маалымат

3
1
2
3

Чыгуучу маалымат

0

4. Эң кичине бөлүнүүчү

Берилген N бүтүн сан үчүн (N>2) 1ден башка эң кичине бөлүнүүчүсүн экранга чыгарыңыз.

Кийирилүүчү маалымат

N бүтүн саны

Чыгуучу маалымат

Маселенин жообу (бир бүтүн сан)

Мисалы,

Кийирилүүчү маалымат

15

Чыгуучу маалымат

3

5. Фиббоначинин катары

Фиббоначинин катарындагы 11 элементти экранга чыгаргыла. Эскерте кетсек:

1. Фиббоначинин катарында 1 жана 2 элемент 1 ге барабар
2. Ар бир кийинки элемент алдыда жайгашкан 2 элементтин суммасына барабар б.а. Фиббоначинин катары 1, 1, 2, 3, 5, 8, 13, 21, ... тартипте кетет.

Кийирилүүчү маалымат

Маалымат кийирилбейт

Чыгуучу маалымат

1 1 2 3 5 8 13 21 34 55 89

6. Полиндром

4 орундуу сандардын ичинде канча полиндром (оң жактан да сол жактан да бирдей окулган сандар) сан бар экендигин эсептегиле.

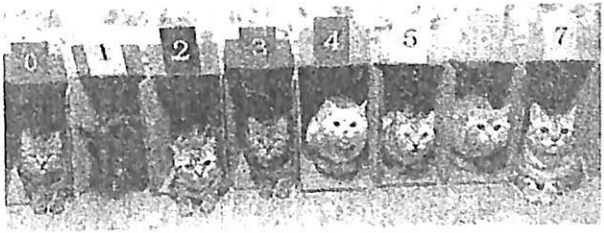
Кийирилүүчү маалымат

Маалымат кийирилбейт

Чыгуучу маалымат

Полиндром сандардын саны

Лекция 6. Массивдер



Сабактын максаты:

1. *Бир өлчөмдүү массивдерди үйрөнүү.*
2. *Бир өлчөмдүү массивдердин үстүнөн амалдарды аткаруу.*

Массив деген эмне?

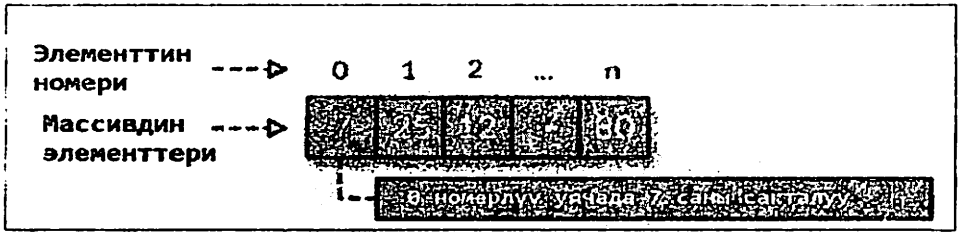
Эгерде буга чейин биз өзгөрмөнү кутучага салыштырсак (бул жерде кандайдыр бир маанини сактай аласыз), анда массив ичи бөлүктөргө бөлүнгөн кутучага окшош десек болот.

Java тилинде массивдер – бул бир эле типтеги бир нече элементтерди бир ат менен сактоого мүмкүндүк берген маалымат структуралары. Алар бир өлчөмдүү же көп өлчөмдүү болушу мүмкүн.

Бул лекцияда бир өлчөмдүү массивдер, аларды жарыялоонун жолдору, массивди түзүү, колдонулушу жөнүндө сөз кылабыз.

Бир өлчөмдүү массивдер

Бир өлчөмдүү массив бир катарга жайгашкан удаалаштыкты элестетет же аны номерленген уячалардын жыйындысы катары кароого болот, алардын ар бирине кандайдыр бир маалыматтарды сактоого болот. Мисалы, бүтүн сандардан турган массивди карап көрөлү (20-сүрөт):



20-сүрөт. Бир өлчөмдүү массив

Сүрөттө көрүнүп тургандай массив бир канча маалыматтардан турат. Аларды бир ат менен сактайт. Кайрылуу үчүн ар бир элементке номер коюулат. Номер 0 санынан башталат. Башка типтеги бир канча мисалдарды карап көрөлү:

1) 3.4, 5.6, 8.9, 3.8, 1.5 – бөлчөк сандардан турган массив

2) "bir", "eki", "uch", "tort", "besh" – жолчолордон турган массив

3) 'a', 'b', 'c', '+', '-' – символдордон турган массив

Массивди жарыялоо

Өзгөрмөнү жарыялаган сыяктуу эле массивди да жарыялоо керек. Массивди жарыялоо бир маанини сактаган кадимки өзгөрмөнү жарыялоого окшош жана массивди жарыялоонун эки жолу бар.

1) Тиби[] массивдинАты; int[] myArray;

2) Тиби массивдинАты[]; int myArray[];

Биринчиси Java тилинин стилине дал келет жана ушундай кылып жарыялоо сунуш кылынат. Экинчи учурда деле массив түзүлөт, бул C тилинен мурастап алган ыкма болуп саналат.

Java тилинде массив түзүп жатканыбызды жарыялоону билдик. Биздин учурда int тибиндеги myArray массиви жарыяланды.

Массивди түзүү

Башка объекттер сыяктуу эле, сиз new операторун колдонуп, Java тилиндеги массивди түзө аласыз, башкача айтканда, ал үчүн эстутумдан орундук сурай аласыз. Бул төмөнкүдөй аткарылат:

1) Массив жарыяланат
2) Массив үчүн эстутумдан керектүү орундук суралат:

new массивдинТиби [керектүү_орундук];

Мисалы:

1) int[] myArray; // массивди жарыялоо

2) myArray = **new int**[10];

Экинчи жолчодо жарыяланган массив үчүн эстутумдан 10 уяча бөлүп алдык б.а. myArray массиви 10 элементтен турат.

Массивди түзүүдө жогоруда көрсөтүлгөн эки жолчону бир жолчого бирктирип жазсак да болот б.а.:

int[] myArray = **new int**[10];

Массивдин узундугу

Жогоруда айткандай эле массивдин узундугу – бул массивдин элементтеринин саны. ***Аны түзүлгөндөн кийин өзгөртүүгө болбойт.***

Массивдин узундугун length өзгөрмөсү аркылуу алсак болот. Мисалы:

int[] myArray = **new int**[10];

System.out.println(myArray.length);

Жогоруда түзүлгөн массивдин узундугун консолго чыгарып алдык.

Массивди инициализациялоо жана анын элементтерине кайрылуу

Массивди түзүп алгандан кийин биз бош массивди эмес, демейки маанилер менен толтурулган массивди алабыз. Мисалы, сандык типте (int, double, float, ж.б.) түзүлгөн массивдин элементтери 0 саны менен толот, жолчо тибиндеги (String) массивдер null менен, логикалык типтеги (bollean) массивдер false менен толот.

Массивди инициализациялоо - аны белгилүү бир (түзүлгөндө автоматтык түрдө (по умолчанию) толтурулган маанилерден башка) маалыматтар менен толтуруу болуп эсептелет.

Мисалы:

```
String[] numbers  = new String[4];
numbers[0] = "One";
numbers[1] = "Two";
numbers[2] = "Three";
numbers[3] = "Four";
```

Массивдин элементтерин экранга чыгарып көрөлү:

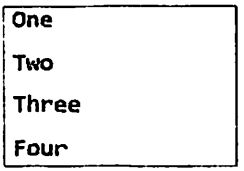
```
public class Example14 {

  public static void main(String[] args) {

        String[] numbers  = new String[] {"One", "Two", "Three", "Four"};
              for (int i = 0; i < 4; i++) {
              System.out.println(numbers[i]);
              }
        }
}
```

Жогорудагы программада массивидин маанилери белгилүү болсо, аларды массивди түзүп жатып {} кашанын ичинде жарыялоону көрүп турабыз. Массивдин ар бир элементине кайрылуу операциясын циклдын жардамында уюштурсак туура болот.

Программанын жыйынтыгы:



Бир өлчөмдүү массивдердин үстүнөн амалдар

Массивдер менен иштөө үчүн атайын **java.util.Arrays** классынын методдорун пайдалансак болот. Массивдердин үстүнөн негизинен толтуруу, керектүү элементти чыгаруу, издөө, сорттоо амалдарын аткарышат.

1. Массивди кокус сандар менен толтуруунун программасын карап көрөлү. Ал үчүн Random классынын методдору колдонулгандыктан аны жүктөп алабыз. Ал java.util пакетинде жайгашкан.

```
import java.util.Random;

public class Example15 {

    public static void main(String[] args) {

        Random random = new Random();

        int[] mass = new int[10];

        for (int i = 0; i < mass.length; i++) {

                mass[i] = random.nextInt(100) + 1;

                System.out.println(mass[i]);
```

```
            }
        }
}
```

Бул программада массивдин элементтери 1ден 100 чейинки аралыктан кокустан тандалган сандар менен толот жана консолго чыгарылат.

Программанын жыйынтыгы:

```
57
62
98
21
70
8
20
6
84
81
```

2. Бүтүн сандардан турган бир өлчөмдүү массив берилсин. Массивден 5 санын издөөчү программа түзүп көрөлү. Мисалы, 5 санын {15, 16, 19, 20, 25, 1, 3, 4, 5, 7, 10, 14} массивинен издесек 8 деген жооп алышыбыз керек. Анткени массивде 5 санынын индекси 8ге барабар.

```
public class Example16 {

    public static void main(String[] args) {

        int[] mass = new int[]
                {15, 16, 19, 20, 25, 1, 3, 4, 5, 7, 10, 14};

        for (int i = 0; i < mass.length; i++) {

                if (mass[i]==5) {

                        System.out.println(i); break;
                }
        }
    }
}
```

Бул жерде массивдин ар бир элементине кайрылып, аны 5 саны менен салыштырып чыктык. Биринчи жолуккан 5 санынын индексин экранга чыгарып, break операторунун жардамында циклдан чыгып кетебиз. Эгерде 5 саны массивде жок болсо, анда консолго эч нерсе чыкпайт.

Программанын жыйынтыгы:

```
8
```

3. Бүтүн сандардан турган бир өлчөмдүү массив берилсин. Аны 1ден 100гө чейинки кокус сандар менен толтургула. Массивдин элементтерин өсүү тартибинде сорттогула.

Маселени чыгарып көрөлү. Бизге кокустан тандалып {4, 8, 7, 2, 1, 23, 78, 54, 9, 15} элементтеринен турган массив берилсин. Аны өсүү тартибинде сорттоо менен {1, 2, 4, 7, 8, 9, 15, 23, 54, 78} массивин алабыз. Бул маселени чечүүдө көбүкчөлөр методун колдонуп көрөлү:

```
import java.util.Random;

public class Example17 {

        public static void main(String[] args) {

                Random random = new Random();


// Массивди түзүү жана толтуруу
                int[] mass = new int[10];

                for (int i = 0; i < mass.length; i++) {

                                mass[i] = random.nextInt(100) + 1;
                }

//Консолго массивди баштапкы абалда бир жолчого чыгаруу
```

```
            for (int number : mass) {
                System.out.print(number+" ");
            }

//Жаңы жолчого өтүү
            System.out.println();

//Сорттоо
            for (int i=0; i<mass.length; i++) {
                int min=i;
                for (int j=i; j<mass.length; j++) {
                    if (mass[min]>mass[j]) {
                        min=j;
                    }
                int c=mass[i];
                mass[i]=mass[min];
                mass[min]=c;
                }

            }
//Сортолгон массивди консолго чыгаруу
            for (int number : mass) {
            System.out.print(number+" ");
            }
        }
}
```

Программанын жыйынтыгы:

```
1 66 37 9 48 55 29 10 92 31
1 9 10 29 31 37 48 55 66 92
```

Бышыктоочу суроолор:

1. Массив деген эмне жана анын өзгөрмөдөн кандай айырмасын байкадың?

2. Бир өлчөмдүү массивди кантип жарыялайбыз?

3. Массивдин элементтерине кантип кайрылабыз? Индекс деген эмне?

4. Массивдин үстүнөн кандай амалдарды аткарсак болот?

5. Жашоондон массивге мисал келтирчи?

5- практикалык сабак

Максаты: бир өлчөмдүү массивдер жөнүндө алган маалыматтарды бышыктоо жана маселелерди чыгаруу.

Келгиле, чогуу төмөнкү маселени карап көрөлү.

Маселе 1.

N элементтен турган бир өлчөмдүү массивде кайталанган сандарды 0 саны менен алмаштыруучу программа түзөлү.

Кийирилүүчү маалымат

N элементтердин саны

Кийинки жолчолордо массивдин элементтери

Чыгуучу маалымат

Бир жолчодо кайталанган элементтер жок бир өлчөмдүү массивдин элементтери

Мисалы:

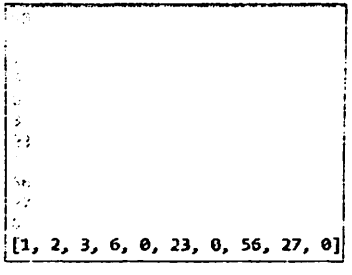
Кийирилүүчү маалымат
4
1
2
4
2
Чыгуучу маалымат
1 2 4 0

Чыгаруу:

```
package progs;
import java.util.Arrays;
import java.util.Scanner;
public class Arr {
  public static void main(String args[]) {
        Scanner obj=new Scanner(System.in);
        int N=obj.nextInt();
```

```
        int[] a=new int[N];
        int k=0;
        for(int i=0; i<N;i++)
        {k=0;
              int x=obj.nextInt();
              for(int j=0; j<=i;j++)
              {if (a[j]==x) k++;}
              if (k==0) a[i]=x;
        }
        System.out.println(Arrays.toString(a));
  }
}
```

Программанын жыйынтыгы:



Өз алдынча иштөөгө мисалдар

1. Бир өлчөмдүү n бүтүн сандардан турган массив берилсин. Массивдеги биринчи эки нолдун ортосундагы сандардын суммасын тапкыла.

Кийирилүүчү маалымат

n элементтердин саны

Кийинки жолчолордо массивдин элементтери

Чыгуучу маалымат

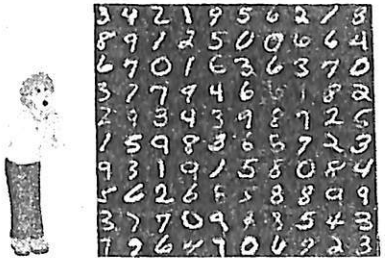
Бир жолчодо бүтүн сан - программанын жообу

Мисалы:

Кийирилүүчү маалымат
10
1 2 0 4 5 0 7 8 0 35
Чыгуучу маалымат
9

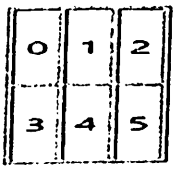
2. Бир өлчөмдүү n бүтүн сандардан турган массив берилсин. Массивдеги жуп сандардын эн кичинесин тапкыла. Эгерде андай элемент жок болсо, биринчи элементти чыгаргыла.
3. Бир өлчөмдүү бүтүн сандардан турган массив берилсин. Массивдин элементтерин өсүү тартибинде иреттегиле.
4. Бир өлчөмдүү бүтүн сандардан турган массив берилсин. Массивдеги элементтердин эң чоңунун жана эң кичинесинин катар номерин чыгаргыла.
5. $a_1 \ldots a_n$ чыныгы сандары берилсин. Бул сандардын ичинен оң сандардын суммасын тапкыла.
6. $a_1 \ldots a_n$ чыныгы сандары берилсин. Зкө бөлүнгөн жуп сандардын суммасын тапкыла.
7. $b_1, \ldots, b_n$ n натуралдык саны жана чыныгы сандары берилсин. Бул сандарды өсүү тартибинде жайгаштыргыла.

Лекция 7. Эки өлчөмдүү массивдер



Сабактын максаты:

1. *Эки өлчөмдүү массивдерди түзүү жана үстүнөн амалдарды аткаруу;*
2. *Тиштүү массивдер жөнүндө түшүнүк алуу;*
3. *foreach циклдык конструкциясы менен иштөө.*



21-сүрөт

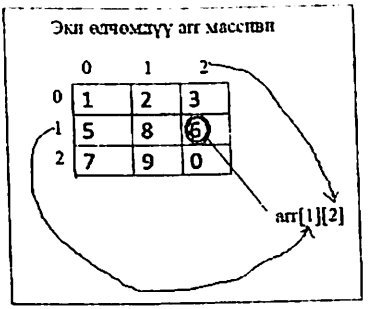
Массивдер бир өлчөмдүү жана көп өлчөмдүү болушат. Бир өлчөмдүү массивдер менен таанышып чыктык. Көп өлчөмдүү массивдердин ичинен эң көп колдонулганы эки өлчөмдүү массивдер болот. Эгерде бир өлчөмдүү массив бир жолчону элестетсе, эки өлчөмдүү массив дегенде таблицаны элестетсеңиз болот (21-сүрөт) б.а. эки өлчөмдүү массивдин жолчолорунун саны жана мамычаларынын саны берилет.

Эки өлчөмдүү массивди жарыялоо жана түзүү

Аларды төмөнкүдөй жарыялап жана түзүп алсак болот:

```
int[][] arr2 = new int[2][3];
```

Мында массивдин тиби (int), андан кийин турган 2 квадраттык кашаа ([][]) массив эки өлчөмдүү экендигин билдирет. Массивдин аты arr2, new сөзү менен массивди түзүп алабыз. Биздин учурда 2 жолчосу жана 3 мамычасы бар таблица түзүлдү. Эки өлчөмдүү массивдин элементтерине кайрылууда массивдин атынан кийин биринчи квадраттык кашаанын ичине жолчонун номерин, экинчи квадраттык кашаанын ичине мамычанын номерин жазабыз (22-сүрөт).



22-сүрөт. Эки өлчөмдүү массивдин элементине кайрылуу

Массивди инициализациялоо

Массивдин элементтерин толтурууну клавиатуранын жардамында ишке ашырып көрөлү. Ал үчүн Scanner классынын методдорун колдонобуз. Программага Scanner классын java.util пакетинен импорттоп кошуп алабыз. Аталган класстын методдорун колдонуу үчүн ал класстын объектисин түзүп алабыз жана nextInt() методу менен ар бир жолчону бүтүн сандар менен толтуруп чыгабыз. Бул учурда камтылуучу циклдар колдонулат:

```
import java.util.Scanner;
public class Example18 {

    public static void main(String[] args) {

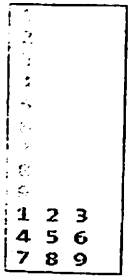
        // Scanner классынын объектисин түзүү
        Scanner obj=new Scanner(System.in);

        //2-өлчөмдүү массивди түзүү
        int [][] arr=new int[3][3];

        //массивдин элементтерин толтуруу
        for(int i=0;i<3;i++) {
            for(int j=0; j<3; j++) {
                arr[i][j]=obj.nextInt();
                }
        }
        // массивдин элементтерин консолго чыгаруу
```

```
            for(int i=0;i<3;i++) {
                  for(int j=0; j<3; j++) {
                        System.out.print(arr[i][j]+" ");
                        }
                  System.out.println();
            }
            // Scanner классынын объектисин бошотуу
            obj.close();
      }
}
```

Программанын жыйынтыгы:



Эки өлчөмдүү массивдерди да тез инициализациялап алсак болот. Мисалы:

```
int[][] months = { {31, 28, 31}, {30, 31, 30}, {31, 31, 30}, {31, 30, 31} };
```

Бул таблицада бир жылдагы айлардын узактыгы сакталган. Таблица 4 жолчодон жана 3 мамычадан турат б.а. 4x3 массиви түзүлөт жана ар бир уяча тиешелүү маанилер менен толтурулат. Программасын түзүп көрөлү:

```
public class  Example19 {
      public static void main(String[] args) {
//массивди түзүү жана толтуруу
            int[][] months = { {31, 28, 31}, {30, 31, 30}, {31, 31, 30}, {31, 30,
      31} };

  // массивдин элементтерин консолго чыгаруу
            for(int i=0;i<4;i++) {
                  for(int j=0; j<3; j++) {
                        System.out.print(months[i][j]+" ");
                        }
```

```
                System.out.println();
            }
        }
}
```

Программанын жыйынтыгы:

```
31 28 31
30 31 30
31 31 30
31 30 31
```

Тиштүү массивдер

Биз буга чейин таблица сыяктуу тегиз массивдерди карадык. Бирок биз эки өлчөмдүү массивдин ар бир жолчосуна ар кандай элементтердин саны менен өзүнчө массивди дайындай алабыз. Мисалы:

```
int[][] nums = new int[3][];
nums[0] = new int[2];
nums[1] = new int[3];
nums[2] = new int[5];
```

Бул мисалда биринчи жолчодо 2 өлчөмдүү массив түзө турганыбызды айттык, бирок жолчонун саны 3 экендигин гана көрсөттүк. Андан ары ар бир жолчо үчүн өзүнчө массив түзүп алдык. Жыйынтыгында тиштүү массив түзүлдү. Аны төмөнкүдөй көрсөтсөк болот:

1	2			
3	4	5		
6	7	8	9	10

Мындай массивдердин үстүнөн иштөөдө ар бир жолчону өзүнчө карап чыгуу керек.

foreach циклдык конструкциясы

for циклинин массивдер жана коллекциялар үчүн иштелип чыккан атайын версиясы болуп эсептелет. Ал төмөнкүдөй синтаксиске ээ:

```
for (берилгендердин_тиби өзгөрмөнүн_аты : контейнер){
  // аткарылуучу амалдар
}
```

Мисалы, төмөнкү код фрагментин карап көрөлү:

```
int[] array = new int[] { 1, 2, 3, 4, 5 };
for (int i : array){
    System.out.println(i);
}
```

Бул мисалда контейнердин ордунда бүтүн сандардан турган массив жайгашып турат, i өзгөрмөсү циклдын ар бир айлануусунда контейнерден кийинки элементти алып отурат.

Эки өлчөмдүү массивдин үстүнөн амалдар

1. Бүтүн сандардан турган квадраттык nxn массиви берилсин. Массивдин негизги диагоналын 1 саны менен толтургула. Негизги диагоналдын жогорку жагында жаткан уячаларды 0 саны менен, төмөн жактагы уячаларды 2 саны менен толтургула б.а. n=4 болгон учур үчүн төмөнкү массивди алгыла жана консолго чыгаргыла.

```
1 0 0 0
2 1 0 0
2 2 1 0
2 2 2 1
```

Чыгаруу:

Келгиле, бул маселени чечүүнүн бир нече жолдорун карап көрөлү. Эки өлчөмдүү массивди A деп атап алалы. Негизги

диагоналдан жогору турган элементтер A[i][j] элементтери болуп саналат, алар үчүн i < j, ал эми негизги диагоналдан төмөнкү элементтер үчүн i > j. Ошентип, биз i жана j маанилерин салыштырып, аларды A[i][j] маанисин аныктоо үчүн колдоно алабыз б.а. төмөнкү алгоритм аткарылат:

```
for (i = 0; i < n; ++i)
{
  for (j = 0; j < n; ++j)
  {
       if (i < j)
       {
       A[i][j] = 0;
       }
       else if (i > j)
    {
         A[i][j] = 2;
       }
       else
       {
       A[i][j] = 1;
    }
  }
}
```

Бул алгоритмди жакшы дей албайбыз. Анткени бир канча if инструкциясы колдонулуп жатат. Эгерде алгоритмди бир аз татаалдантсак бир да if колдонбой программаны түзүп алсак болот. Биринчи негизги диагоналды толтуруп алалы:

```
for (i = 0; i < n; ++i)
{
  A[i][i] = 1;
}
```

Андан кийин негизги диагоналдан жогору турган бардык элементтерди 0 мааниси менен толтурабыз, ал үчүн камтылуучу циклды колдонобуз:

```
for (i = 0; i < n; ++i)
{
  for (j = i + 1; j < n; ++j)
  {
    A[i][j] = 0;
  }
}
```

Байкаганыңардай ички циклда мамычанын номери жолчого жараша жылып отурат.

Ушундай эле жол менен негизги диагоналдан төмөнкү элементтерди 2 саны менен толтуруп алабыз:

```
for (i = 0; i < n; ++i)
{
  for (j = 0; j < i; ++j)
  {
    A[i][j] = 2;
  }
}
```

Ошондой эле сырткы циклдарды бир циклга бириктирип, төмөнкү компакттуу чечимди алабыз:

```
for (i = 0; i < n; ++i)
{  // i номерлүү жолчону толтурабыз
  for (j = 0; j < i; ++j)
  {
        A[i][j] = 2; // Биринчи диагоналдын төмөн жагына 2
                //жазабыз
  }
  A[i][j] = 1;      // Жогорку цикл аяктаганда i==j
                //болгондо, 1 жазабыз
  for (++j; j < n; ++j)  // Циклды jди 1ге көтөрүү менен
                  //баштайбыз
  {
    A[i][j] = 0;    // Диагоналдын жогорку жагына 0
                // жазабыз
  }
}
```

Бышыктоочу суроолор:

1. Эки өлчөмдүү массивди кантип жарыялайбыз?
2. Эки өлчөмдүү массивдин элементтерине кантип кайрылабыз?
3. Тиштүү массивдер кантип түзүлөт?
4. Эки өлчөмдүү массивдин бир өлчөмдүү массивден айырмасы барбы?
5. foreach циклдык конструкциясынын өзгөчөлүгү эмнеде?
6. Эки өлчөмдүү массивдер менен жашооңуздагы кандай маселелерди чечсеңиз болот?

6- практикалык сабак

Максаты: эки өлчөмдүү массивдер жөнүндө алган маалыматтарды бышыктоо жана маселелерди чыгаруу.

Келгиле, чогуу төмөнкү маселени карап көрөлү.

Маселе 1.

Бүтүн сандардан турган 4x4 эки өлчөмдүү массиви берилсин. Биринчи жана акыркы мамычаны гана экранга чыгаруучу программа түзөлү.

Чыгаруу:

```
package progs;
import java.util.Random;

public class Massiv100 {

    public static void main(String[] args) {

        Random random = new Random();

        int[][] array = new int[4][4];
        for (int i = 0; i < 4; i++) {
                        for (int j = 0; j < 4; j++) {
```

```
                    array[i][j] = random.nextInt(100) + 1;
                        }
                    }
            System.out.println("Тузулгон массив:");
            for(int i=0;i<4;i++)
            {
                for (int j = 0; j < 4; j++) {
                    System.out.print(array[i][j]+" ");
                }
                System.out.println();
            }
            System.out.println("Жыйынтык ");

                for (int i = 0; i < 4; i++) {
                            System.out.print(array[i][0]+" ");
                            System.out.print(array[i][3]+" ");
                            System.out.println();
                            }

    }
}
```

Программанын жыйынтыгы:

```
Тузулгон массив:
70 4 93 89
47 1 96 21
62 20 53 92
61 8 92 44
Жыйынтык
70 89
47 21
62 92
61 44
```

Өз алдынча иштөөгө мисалдар

1. Эки өлчөмдүү массив берилсин. Жуп индексте турган жолчолорду экранга чыгаргыла.

2. Эки өлчөмдүү бүтүн сандардан турган массив берилсин. Ар бир жолчону өсүү тартибинде иреттегиле.

3. 5×5 бүтүн сандардан турган эки өлчөмдүү массив берилсин. Массивде 7 саны канча жолу жолугаарын табуучу программа түзгүлө.

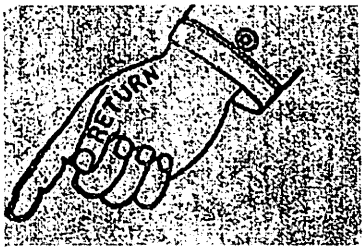
4. n×n квадраттык эки өлчөмдүү массиви берилсин. Негизги жана кошумча диагоналынын элементтеринин суммасын тапкыла.

5. n×m өлчөмүндөгү чыныгы сандардан турган матрица берилсин. Анын ар бир жолчосунун эң чоң маанилеринин суммасын тапкыла.

6. n×m өлчөмүндөгү чыныгы сандардан турган матрица берилсин. Матрицанын эң чоң элементинин индексин тапкыла.

7. Бүтүн сандардан турган a жана b эки өлчөмдүү массивдеринин тиешелүү элементтерин кошуп c массивине ыйгаргыла.

Лекция 8. Методдор



Сабактын максаты:

1. *Метод жөнүндө түшүнүк алуу*
2. *Методдун параметрлерин аныктоо*
3. *return оператору. Методдун жыйынтыгы*
4. *Методду ашыкча жүктөө*
5. *Рекурсиялык функциялар*

Метод жөнүндө...

Метод белгилүү иш-аракеттерди аткарган операторлордун жыйындысын камтып турат.

Методдор жалпысынан төмөнкүдөй аныкталат:

```
[модификаторлор] кайтарылуучу_маанинин_тиби методдун_аталышы
([параметрлер]){
    // методдун телосу
}
```

Модификаторлорду жана параметрлерди колдонуу милдетүү эмес. Java тилинде түзүлгөн программанын негизги классы программага кирүү чекити катары кызмат кылган негизги main методун камтыйт:

```
public static void main(String[] args) {
  System.out.println("Салам!");
}
```

Бул жерде public жана static ачкыч сөздөрү модификаторлор болуп эсептелет. Андан ары кайтарылуучу маанинин тиби көрсөтүлөт. Бул жерде void тиби методдун эч кандай маани кайтарбай тургандыгын аныктайт. Андан кийин методдун аты (main) жана кашаанын ичинде методдун параметрлери (String[] args) көрсөтүлүп турат. Ачылган жана жабылган фигуралык кашаалардан ортосуна методдун телосу б.а. иш-аракеттерди аткаруучу операторлор жайгашат. Биздин учурда консолго "Салам!" сөзүн чыгаруучу оператор жайгашып турат. Класстын ичинде бир канча методдор болушу мүмкүн. Методдор кадимки жана статикалык (static) болушу мүмкүн.

Статикалык методдор

Статикалык өзгөрмөлөрдөн тышкары класстарда статикалык методдор да болушу мүмкүн.

Класста кадимки методду чакыруу үчүн алгач ошол класстын объектин түзүп алып, андан кийин гана ал объект үчүн методду чакыруу керек. Кадимки методду объектсиз класста чакыруу мүмкүн эмес.

Мисалы, бизде төмөнкү класс бар дейли:

```
public class DataInfo
{
  public int getValue()
  {
    return 100;
  }
}
```

Негизги класстан DataInfo классынын getValue() методун чакыруу үчүн төмөнкүдөй жазуу туура эмес болуп калат:

```
public class Example20
{
  public static void main(String[] args)
  {
   System.out.println(DataInfo.getValue()); //бул туура эмес!
  }
}
```

Программа иштөөсү үчүн негизги класстагы main методунун ичинде DataInfo классынын объектисин түзүү керек болот жана ал объект гана getValue() методун чакыра алат.

Ал эми статикалык методду чакыруу үчүн класстын статикалык объектисинин болгону жетиштүү. main методу статикалык болгондуктан ал эч кандай объектсиз эле иштей алат. Бир канча статикалык методдорду кошуп көрөлү:

```
public class Example21 {

        public static void main (String args[]){
```

```
        hello();
        welcome();
        welcome();
    }
    static void hello(){

        System.out.println("Hello");
    }
    static void welcome(){

        System.out.println("Welcome to Java");
    }
}
```

Бул мисалда main негизги методунда hello методу бир жолу, welcome методу эки жолу чакырылганын көрүп турасыңар. Мына ушул жерде методдорду түзүүнүн жана колдонуунун артыкчылыктарын айтууга болот. Анткени кээ бир аракеттерди өзүнчө метод катары жарыялап кийин аны программанын ар кайсы бөлүгүндө бир канча жолуу колдонууга болот. Эгер метод эч кандай параметрге ээ болбосо методдун аталышынан кийин бош кашаалар коюлат.

Ошондой эле main методу менен бир класста жарыяланган башка методдорду main методунан чакыруу үчүн алар static модификаторуна ээ болуусу керек экендигин баса белгилеп айтабыз. Натыйжада, программаны түзүп, аткаргандан кийин, консолдон төмөнкү жыйынтыкты көрөбүз:

```
Hello
Welcome to Java
Welcome to Java
```

Башка класстан статикалык методду чакыруу үчүн статикалык методдун аталышынан мурун класстын атын көрсөтүшүңүз керек. Ал төмөнкүдөй форматта жазылат:

```
Тибин аты = КласстынАты.методдунАты(параметрлери)
```

Мисалы биз Math классынын төмөнкү статикалык методдорун билебиз:

```
int x = Math.min(a, b);
int y = Math.max(a, b);
int z = Math.sqrt(a);
```

Методдун параметрлерин аныктоо

Параметрлерди колдонуу менен биз методдорго эсептөөдө колдонула турган ар кандай маалыматтарды бере алабыз. Мисалы:

```
static void sum(int x, int y){

    int z = x + y;
    System.out.println(z);
}
```

Бул метод int тибиндеги эки параметрден турат жана аларды кошуп суммасын консолго чыгарып берүү жумушун аткарат. Келгиле негизги методдон ушул методду чакырып көрөлү:

```
public class Example22 {

    public static void main(String[] args) {
        int a = 6;
    int b = 8;
    sum(a, b);  //14
    sum(3, a);  //9
    sum(5, 23); //28
     }
```

```
    static void sum(int x, int y){

      int z = x + y;
      System.out.println(z);
    }

}
```

Бул жерде sum методу int тибиндеги эки маанини алгандыктан параметрлердин ордуна int тибиндеги эки маанини беришибиз керек. Параметрлердин ордуна берилүүчү маанилерди аргументтер деп да атайбыз. Маанилер параметрлерге позициясы боюнча б.а. биринчи аргумент биринчи параметрге экинчи аргумент экинчи параметрге ж.б. ыйгарылат.

Башка мисалды карап көрөлү:

```
public class Example23 {

      public static void main (String args[]){

        display("Akmat", 34);
        display("Marat", 28);
        display("Anna", 23);
      }

        static void display(String name, int age){

        System.out.println(name);
        System.out.println(age);
      }
}
```

Жогорудагы мисалда display методу эки параметрди алат. Биринчи параметрдин тиби String экинчи параметрдин тиби int. Ошондуктан методду чакыруу учурунда биринчи параметрге жолчо, экинчиге сан беришибиз керек.

Кээ бир учурда бир эле метод бир типтеги ар кандай сандагы параметрлерди кабыл алуусу мүмкүн. Алар 2,3,4 же андан да көп болуп калат. Мисалы, төмөнкү программаны карап көрөлү:

```
public class Example24 {

    public static void main (String args[]){

      sum(1, 2, 3);          // 6
      sum(1, 2, 3, 4, 5);    // 15
      sum();                 // 0
    }
    static void sum(int ...nums){

      int result =0;
      for(int n: nums)
        result += n;
      System.out.println(result);
    }
}
```

Параметрдин аталышынан мурда коюлуучу үч чекит ...nums массив экендигин көрсөтөт. Биз sum методуна бир сан же бир нече сан же эч кандай параметр бербешибиз мүмкүн. Аны жогоруда өзүңөр байкасаңар керек.

return оператору. Методдун жыйынтыгы

Биз жогоруда сөз кылгандай методдор тибине жараша маанилерди кайтарып берет. Эгерде тиби void болсо, анда эч кандай маани кайтарбай турганын билебиз. Маанилерди кайтаруу үчүн return оператору колдонулат. Анын синтаксиси төмөнкүдөй:

return кайтуучу_маани;

Методдун жыйынтыгын алуу үчүн return операторунан кийин кайтаруу керек болгон маанини көрсөтөбүз. Бул литералдык маани, өзгөрмөнүн мааниси же кандайдыр бир татаал амал болушу мүмкүн.

Мисалы:

```
public class Example25 {


      public static void main (String args[]){

     int x = sum(1, 2, 3);
     int y = sum(1, 4, 9);
     System.out.println(x);
     System.out.println(y);
   }
  static int sum(int a, int b, int c){

     return a + b + c;
   }
}
```

Программанын жыйынтыгы:

```
6
14
```

Жогоруда көрсөтүлгөн мисалда sum методунун тиби int болуп турат, бул метод бүтүн санды кайтарып берет дегенди билдирет. Методдун ичин карай турган болсок, return оператору чындыгында эле (a+b+c) амалынын жардамында бүтүн сан кайтарып жатканын көрүп турабыз.

Метод кээ бир шарттарга жараша ар кандай маанилерди кайтаруу үчүн return операторуна бир нече жолу кайрылса болот:

```
public class Example26 {

      public static void main (String args[]){

            System.out.println(daytime(7));
            System.out.println(daytime(13));
            System.out.println(daytime(18));
            System.out.println(daytime(2));
      }
      static String daytime(int hour){
```

```
        if (hour >24 || hour < 0)
            return "Туура эмес убакыт";
        else if(hour > 21 || hour < 6)
            return "Бейпил түн";
        else if(hour >= 15)
            return "Кутман кеч";
        else if(hour >= 11)
            return "Кутмандуу күнүңүз менен";
        else
            return "Кутман таң";
    }
}
```

Программанын жыйынтыгы:

```
Кутман таң
Кутмандуу күнүңүз менен
Кутман кеч
Бейпил түн
```

Жогорудагы мисалда daytime методу hour параметринин маанисине жараша ар кандай маанилерди кайтарып берет. Бул жерде күндүн ар кайсы убактысы көрсөтүлгөн жана ал убакта айтылуучу саламдашуу сөздөрү кайтарылуучу маалымат катары каралды.

Ошондой эле return оператору методдон чыгып кетүү үчүн да колдонулат. Ал учурда return операторунан кийин эч нерсе жазбайбыз. Мисалы жогоруда каралган программадагы биринчи шартты төмөнкүдөй кылып өзгөртсөк болот:

```
if (hour >24 || hour < 0)
    return;
```

Эгерде datetime методуна берилген маани 24төн чоң же 0дөн аз болсо, анда биз жөн гана методдон чыгабыз.

Методду ашыкча жүктөө

Программада бир эле аталыштагы, бирок ар кандай типтеги, ошондой эле параметрлердин саны ар кандай болгон методдорду колдоно алабыз. Бул механизм методду ашыкча жүктөө (method overloading) деп аталат. Төмөнкү мисалды карап көрөлү:

```
public class Example27 {
    public static void main(String[] args) {

        System.out.println(sum(2, 3));
        System.out.println(sum(4.5, 3.2));
        System.out.println(sum(4, 3, 7));
    }
    static int sum(int x, int y){
        return x + y;
    }
    static double sum(double x, double y){
        return x + y;
    }
    static int sum(int x, int y, int z){
        return x + y + z;
    }
}
```

Программанын жыйынтыгы:

```
5
7.7
14
```

Бул жерде *sum* методу үч жолу ар кандай параметрлер жана типтер менен ашыкча жүктөлүп жатат. Бул учурда параметрлерге жана типке жараша тиешелүү метод өз жумушун аткара берет. Аны программанын жыйынтыгынан көрүүгө болот.

Дагы бир учурду карап көрөлү:

```
public class Example28 {
    public static void main(String[] args) {
```

```
        System.out.println(sum(2, 3));
        System.out.println(sum(4, 3));
    }
    static int sum(int x, int y){

        return x + y;
    }
    static double sum(int x, int y){

        return x + y;
    }
}
```

Бул мисалда методдун параметрлеринде эч кандай айырма жок болгондуктан, бул методду ашыкча жүктөө эмес, кайталанып калган метод катары карасак болот.

Рекурсиялык функциялар

Рекурсиялык функциялардын башка методдордон негизги айырмачылыгы, алар өзүн-өзү чакыра алат.

Мисалы, сандын факториалын эсептеген функцияны карап көрөлү:

```
static int factorial(int x){

  if (x == 1){

    return 1;
  }
  return x * factorial(x - 1);
}
```

Бул жерде биринчи шарт текшерилет: эгерде кийирилген сан 1ге барабар болбосо, анда бул санды ошол эле функциянын натыйжасына көбөйтөбүз, ага параметр катары x-1 саны берилет. Башкача айтканда параметрдин мааниси бирге барабар болгонго чейин рекурсивдүү түшүү пайда болот.

Рекурсивдүү функцияда return операторун колдонгон жана функциянын башында турган кандайдыр бир негизги вариант

болушу керек. Факториалды эсептөөдө негизги вариант бул **if** (x == 1) **return** 1. Ар бир кийинки рекурсиялык чакыруулар негизги вариантка жакындап баруусу керек. Факториалды эсептөөдө x тин мааниси улам азайып отуруп 1ге барабар болот.

Рекурсивдүү функциянын дагы бир кеңири таралган мисалы - Фибоначчи сандарын эсептеген функция. Теорияда Фибоначчи ырааттуулугунун n-мүчөсү f(n)=f(n-1) + f(n-2), f(0)=0 жана f(1)=1 формуласы менен аныкталат.

```
static int fibonachi(int n){

  if (n == 0){
    return 0;
  }
  if (n == 1){
    return 1;
  }
  else{
    return fibonachi(n - 1) + fibonachi(n - 2);
  }
}
```

Бул учурда да негизги вариант же шарттын акыркы чеги бергилүү. Ал n == 0 жана n == 1 болгон учурлар. Рекурсиялык функция ошол шартка жеткенге чейин кайталанат.

Бышыктоочу суроолор:

1. Метод деген эмне жана статикалык метод жөнүндө айтып бериңиз?
2. Методдун параметрлери жөнүндө айтып бериңиз?
3. return оператору кандай максатта колдонулат?
4. Сиздин оюңузча методду ашыкча жүктөөгө болобу?
5. Рекурсивдик функциянын өзгөчөлүгү эмнеде?

7- практикалык сабак

Максаты: методдор жөнүндө алган маалыматтарды бышыктоо жана маселелерди чыгаруу.

Келгиле, чогуу төмөнкү маселени карап көрөлү.

Маселе 1.

y=ax+b функциясынын маанисин эсептөө үчүн метод түзүп, негизги main методунан чакырып көрөлү. Мында x өзгөрмө 1ден 10го чейинки маанилерди алат. a, b туракттуулар.

Чыгаруу:

```
package progs;
public class Massiv100 {

	public static void main(String[] args) {
	for(int i=1; i<=10; i++) {
		System.out.println(y(i));
	}

	}

	static int y(int x) {
		int a=2;
		int b=3;
		return a*x+b;
	}
}
```

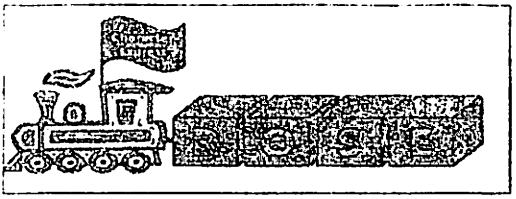
Программанын жыйынтыгы:

```
5
7
9
11
13
15
17
19
21
23
```

Өз алдынча иштөөгө мисалдар

1. Карама-каршы чокуларынын координаталары (x_1, y_1) жана (x_2, y_2) болгон жана жактары x,y координаталарына параллель болгон төрт бурчтуктун аянтын табуучу метод түзгүлө.
2. Эгерде x<0 болсо -1, x=0 болсо 0, x>0 болсо 1 маанилерин кайтаруучу бүтүн типтеги S(x) методун түзгүлө.
3. Квадраттык тендеменин $(ax^2 + bx + c = 0$ түрүндөгү) тамырларынын санын табуучу бүтүн типтеги метод түзгүлө. Тамырлардын санын дискриминанттын маанисин пайдаланып тапкыла.
4. Радиусу R болгон тегеректин аянтын табуучу чыныгы типтеги scircle(R) методун түзгүлө. Тегеректин аянтын $S = \pi R^2$ формуласы менен эсептегиле.
5. Ондук санды экилик санга өткөрүүчү бүтүн типтеги Dvoich(x) методун түзгүлө.
6. A, B чыныгы сандары жана бир арифметикалык амал кийирилсе, анда эки сандын үстүнөн берилген амалды аткаруучу Calc (A,B, Op) методун түзгүлө.

Лекция 9. Жолчолор (строки). String классы



Сабактын максаты:

1. *Жолчолор жөнүндө түшүнүк алуу*
2. *Жолчолорду түзүү*
3. *Жолчолордун устүнөн иштөөдө колдонулуучу негизги методдор*
4. *StringBuffer жана StringBuilder*
5. *Регулярдуу туюнтмалар*

Жолчолор жөнүндө...

Сиз билесизби заманбап информатиканын негизги функцияларынын бири – адамдын тилин изилдөө жана иштеп чыгуу болуп саналат. Биздин тилибиз бул сөздөр, сүйлөмдөр, тексттер б.а. жолчолор эмеспи...

Java тилинде жолчолор бул символдордун удаалаштыгы болуп эсептелет. Биз буга чейин String тиби бар экенин билгенбиз. Бул тип эң көп колдонулуучу тип болуп саналат. Анткени ал бизге тексттерди сактоого жардам берет. Java'да аларды кантип түзөбүз?

Жолчолорду түзүү

Жаңы жолчо түзүү үчүн, биз String классынын конструкторлорунун бирин же кош тырмакчанын ичине жолчону жазып алсак болот. Мисалы:

```
public static void main(String[] args) {

  String sap1 = "Hello";
  String sap2 = new String();
  String sap3 = new String(new char[] {'h', 'e', 'l', 'l', 'o'});
  String sap4 = new String(new char[]{'w', 'e', 'l', 'c', 'o', 'm', 'e'}, 3, 4);

   System.out.println(sap1); // Hello
   System.out.println(sap2); //
   System.out.println(sap3); // hello
   System.out.println(sap4); // come
}
```

Жогорудагы мисалда String классынын 4 объектиси түзүлдү.

sap1 объектиси жолчону кош тырмакчанын ичине жазуу менен түзүлдү.

sap2 объектиси курук жолчону түзөт. Анткени конструкторго эч нерсе жөнөтүлгөн жок. **new** ачкыч сөзү жаңы объектти түзүү б.а. эс тутумдан орун ажыратуу үчүн колдонулат.

sap3 объектиси символдордун массивинен түзүлдү.

sap4 объектиси ошондой эле символдор массивинен 3 индекстеги символдон баштап 4 символду кыркып алуу менен түзүлдү.

Ошентип, биз жолчолорду түзүп алууну билдик. *Жолчолор өзгөрбөй турган объекттер, башкача айтканда, жолчону өзгөрткөн бардык операциялар чындыгында жаңы сапты түзөт.*

Жолчолордун үстүнөн иштөөдө колдонулуучу негизги методдор

Жолчо символдордун жыйындысы катары каралгандыктан, жолчонун узундугун же символдор жыйындысынын узундугун табуу үчүн **length()** методун колдоно алабыз:

```
String str1 = "Java";
System.out.println(str1.length()); // 4
```

toCharArray() методун колдонуп, жолчону кайра символдордун массивине айландырсаңыз болот:

```
String str1 = new String(new char[] {'h', 'e', 'l', 'l', 'o'});
char[] helloArray = str1.toCharArray();
```

Жолчо бош болушу мүмкүн. Бул үчүн, кош тырмакчалардын ичине эч нерсе жазбай түзүп алсаңыз болот:

```
String s = "";  // бош жолчо
if(s.length() == 0) System.out.println("Жолчо бош");
```

String классында жолчонун бош экендигин текшерүү үчүн атайын **isEmpty()** методу бар. Эгерде жолчо бош болсо, анда ал true (чын) маанисин кайтарат:

```
String s = "";  // бош жолчо
if(s.isEmpty()) System.out.println("Сап бош");
```

Жолчолорду салыштыруу

Салыштыруу жолчолордун үстүнөн аткарылуучу эң кеңири таралган жумуштардын бири. String классында жолчолорду салыштыруу үчүн колдонулган ондогон методдор бар. Төмөндө биз алардын негизгилерин карап чыгабыз.

1. boolean equals(String str)

Бардык символдор дал келсе, саптар бирдей деп эсептелет. Мисалы:

```
String sap1 = "Salam";
String sap2 = "Salam";
boolean result = sap1.equals(sap2);
System.out.println(result);  //true
```

2. boolean equalsIgnoreCase(String str)

Жолчолорду тамганын регистрине (өлчөмүнө) көңүл бурбай салыштырат жана барабар болсо true жообун кайтарат. Мисалы:

```
String sap1 = "salam";
String sap2 = "Salam";
boolean result = sap1.equalsIgnoreCase(sap2);
System.out.println(result);  //true
```

3. int compareTo(String str)

Жолчолорду лексикографиялык жактан салыштырат. Эгерде жолчолор бирдей болсо, 0 кайтарат. Эгерде биринчи жолчо лексикографиялык жактан экинчиден мурун келсе, терс маанини, ал эми андан кийин келсе, оң маанини кайтарат. Мисалы:

```
String str1 = "Hello";
String str2 = "Java";
int result = str1.compareTo(str2);
System.out.println(result); //-2
```

Бул учурда -2 жообун алдык, анткени биринчи жолчо лексикографиялык жактан экинчиден мурун келет. Ал эми жолчолордун ордун алмаштырып көрсөк 2 деген жоопту алабыз:

```
String str1 = "Hello";
String str2 = "Java";
int result = str2.compareTo(str1);
System.out.println(result); //2
```

4. int compareToIgnoreCase(String str)

compareTo методуна окшош, бирок регистрдин өлчөмүнө карабай салыштырат.

5. boolean regionMatches(int toffset, String str, int offset, int len)

Жолчолордун бөлүктөрүн салыштырат. Мисалы:

```
public class Example29 {

    public static void main(String[] args) {
```

```
        String Str1 = new String("Добро пожаловать на Native_kg.net");
    String Str2 = new String("Native_kg");
    String Str3 = new String("NATIVE_KG");

    System.out.print("Кайтарылуучу маани: ");
    System.out.println(Str1.regionMatches(20, Str2, 0, 9));

    System.out.print("Кайтарылуучу маани: ");
    System.out.println(Str1.regionMatches(20, Str3, 0, 9));
        }
}
```

Программанын жыйынтыгы:

```
Кайтарылуучу маани: true
Кайтарылуучу маани: false
```

6. boolean startsWith(String prefix)

Учурдагы жолчо prefix жолчосу менен башталса true маанисин, антпесе false маанисин кайтарат. Мисалы:

```
String S="Salam Osh!";
    if (S.startsWith("Salam"))
            System.out.println("Yes");
    else
            System.out.println("No");
```

7. boolean endsWith(String suffix)

Учурдагы жолчо prefix жолчосу менен аяктаса true маанисин, антпесе false маанисин кайтарат. Мисалы:

```
Scanner in = new Scanner(System.in);
String S = in.nextLine();

if (S.endsWith(".jpg") || S.endsWith(".jpeg"))
{
  System.out.println("Бул Jpeg!");
}
else if (S.endsWith(".htm") || S.endsWith(".html"))
```

```
{
  System.out.println("Бул HTML-баракча");
}
else if (S.endsWith(".doc") || S.endsWith(".docx"))
{
  System.out.println("Бул Word документ");
}
else
{
  System.out.println("Белгисиз формат");
}
```

Камтылуучу жолчолорду издөө

Жолчону салыштыруудан кийинки эң популярдуу операция - бул бир жолчону экинчисинен издөө. String классында да бул үчүн бир нече методдор бар:

1. int indexOf(String str)

Учурдагы жолчодон str жолчосун издейт. Кезиккен жолчонун биринчи символунун индексин кайтарат. Мисалы:

```
String S="Hello";
System.out.println(S.indexOf("lo")); //3
```

2. int indexOf(String str, int index)

Биринчи index символдорду өткөрүп жиберип, учурдагы жолчодон str жолчосун издейт. Табылган жолчонун биринчи символунун индексин кайтарат, эгерде изделген жолчо табылбаса -1 санын кайтарат. Мисалы:

```
String S="telephone";
System.out.println(S.indexOf("tel",3)); //-1
```

3. int lastIndexOf(String str)

Учурдагы жолчодон str жолчосун оң жактан (жолчонун аягынан) баштап издейт. Биринчи жолуккан жолчонун индексин кайтарат. Мисалы:

```
String S="telephone tel";
System.out.println(S.lastIndexOf("tel")); //10
```

4. int lastIndexOf(String str, int index)

Биринчи index символдорду өткөрүп жиберип, учурдагы жолчодон str жолчосун аягынан баштап издейт. Эгерде изделген жолчо жок болсо, -1 санын кайтарат. Мисалы:

```
String S="Salam Osh";
System.out.println(S.lastIndexOf("Osh",4)); //-1
```

Жолчолорду түзүү

1. String substring(int beginIndex, int endIndex)

beginIndex..endIndex интервалдагы символдордон түзүлгөн жолчону кайтарып берет. Мисалы:

```
String S="Salam Osh";
String sap=S.substring(6,9);
System.out.println(sap); // Osh
```

2. String repeat(int count)

Учурдагы жолчону count жолу кайталап, жаңы жолчону түзөт. Мисалы:

```
String S="Hello";
System.out.println(S.repeat(3)); // HelloHelloHello
```

3. String replace(char oldChar, char newChar)

oldChar символун newChar символуна алмаштыруу менен жаңы сапты кайтарып берет. Мисалы:

```
String S="My number is +12345678";
System.out.println(S.replace('2', '1'));
```

4. String toLowerCase()

Сапты кичине тамгага которот. Мисалы:

```
String S="SALAM";
System.out.println(S.toLowerCase()); //salam
```

5. String toUpperCase()

Сапты чоң тамгага которот. Мисалы:

```
String S="salam";
System.out.println(S.toLowerCase()); //SALAM
```

6. String trim()

Саптын башындагы жана аягындагы бардык боштуктарды жок кылат. Мисалы:

```
public class Example30 {

        public static void main (String args[]){

         String S="  SALAM  ";
         System.out.println(S);
         System.out.println(" Пароль\n   \n".trim());
         System.out.println(S.trim());
        }
}
```

Программанын жыйынтыгы:

```
  SALAM
Пароль
SALAM
```

StringBuffer жана StringBuilder

String классынын объекттери өзгөрүлбөйт, ошондуктан жолчолорду өзгөрткөн бардык операциялар чындыгында жаңы жолчону түзөт, бул тиркемелердин иштешине таасирин тийгизет. Бул көйгөйдү чечүү үчүн, жолчолор менен иштөөнү үнөмдүү кылуу максатында Java тилине StringBuffer жана StringBuilder класстары кошулган. Алар негизинен иштөөгө таасир этпестен өзгөртүлүшү мүмкүн болгон кеңейтилүүчү жолчолорду билдирет.

Бул эки класстын методдору, иштөөсү баары окшош. Бир өзгөчөлүгү StringBuffer классы көп агымдуу программаларда ыңгайлуу, ал эми StringBuilder бир агымдуу программаларда тез иштейт.

Бул конструкторлор жолчону кабыл алып, кошумча **16 символго** эстутумду ажыратып алат. **capacity()** методунун жардамында эстутумдан орун бөлүнгөн символдордун санын билсек болот. Ал эми ensureCapacity() методу менен символдор буферинин минималдык өлчөмүн өзгөртүп алсак болот. Мисалы:

```
String str = "Java";
StringBuffer strBuffer = new StringBuffer(str);
System.out.println("Сыйымдуулук: " + strBuffer.capacity()); // 20
strBuffer.ensureCapacity(32);
System.out.println("Сыйымдуулук: " + strBuffer.capacity()); // 42
System.out.println("Узундугу: " + strBuffer.length()); // 4
```

StringBuffer эң башында "Java" жолчосу менен аныкталгандыктан, анын сыйымдуулугу 4 + 16 = 20 символду түзөт. Андан кийин минималдуу буфер сыйымдуулугун 32 белгиге чейин көбөйтүү үчүн strBuffer.ensureCapacity(32) деп чакырып буфер сыйымдуулугун жогорулатабыз. Бирок, акыркы кубаттуулугу олуттуу айырмаланышы мүмкүн. Ошентип, бул учурда 32 же 32 + 4 = 36 эмес, 42 белгиден турган кубаттуулукту алабыз. Чындыгында, эффективдүүлүктү жогорулатуу үчүн Java кошумча эстутумду бөлүштүрүп берет. Бирок жолчонун узундугу ар дайым 4 боюнча калат.

StringBuffer'де сакталган жолчону алуу үчүн toString() стандарттык методу колдонулат:

```
String str = "Java";
StringBuffer strBuffer = new StringBuffer(str);
System.out.println(strBuffer.toString()); // Java
```

charAt() методу индекс боюнча символду алат жана setCharAt() методу белгини белгилүү бир индекске орнотот:

```
StringBuffer strBuffer = new StringBuffer("Java");
char c = strBuffer.charAt(0); // J
System.out.println(c);
strBuffer.setCharAt(0, 'c');
System.out.println(strBuffer.toString()); // cava
```

getChars() методу белгилүү бир индекстердин ортосундагы символдордун жыйындысын алат:

```
StringBuffer strBuffer = new StringBuffer("world");
int startIndex = 1;
int endIndex = 4;
char[] buffer = new char[endIndex-startIndex];
strBuffer.getChars(startIndex, endIndex, buffer, 0);
System.out.println(buffer); // orl
```

append() методу StringBufferдин аягына жолчону кошот:

```
StringBuffer strBuffer = new StringBuffer("hello");
strBuffer.append(" world");
System.out.println(strBuffer.toString()); // hello world
```

insert() методу StringBuffer'ге белгилүү бир индексте жолчону же символду кошот:

```
StringBuffer strBuffer = new StringBuffer("word");
```

```
strBuffer.insert(3, 'l');
System.out.println(strBuffer.toString()); //world

strBuffer.insert(0, "s");
System.out.println(strBuffer.toString()); //sworld
```

delete() методу белгилүү бир диапозондогу бардык символдорду жок кылат, ал эми deleteCharAt() методу белгилүү бир индекстеги бир символду жок кылат:

```
StringBuffer strBuffer = new StringBuffer("assembler");
strBuffer.delete(0,2);
System.out.println(strBuffer.toString()); //sembler

strBuffer.deleteCharAt(6);
System.out.println(strBuffer.toString()); //semble
```

substring() методу жолчону белгилүү бир индекстен аягына чейин же белгилүү бир индекске кыскартат:

```
StringBuffer strBuffer = new StringBuffer("hello java!");
String str1 = strBuffer.substring(6); // жолчону 6 символдон
                                      //баштап аягына кечин кыркуу
System.out.println(str1); //java!
//3 төн 9 га чейин кыркуу
String str2 = strBuffer.substring(3, 9);
System.out.println(str2); //lo jav
```

Регулярдуу туюнтмалар

Регулярдуу туюнтмалар жолчону башкаруу үчүн күчтүү куралды камсыз кылат. Кадимки туюнтмалар жолчо же камтылуучу жолчо дал келүүчү үлгүнү көрсөтүүгө мүмкүндүк берет. String классынын кээ бир методдору кайсы бир операцияны ишке ашыруу үчүн регулярдык туюнтмаларды кабыл алат.

split методу

split() методу жолчону камтылуучу жолчолорго бөлүү үчүн колдонулат. Ал параметр катары жолчону бөлүү критерийин билдирген регулярдык туюнтманы кабыл алат. Мисалы, сүйлөмдү сөздөргө ажыратып көрөлү:

```
public class Example31 {

    public static void main (String args[]){

        String text = "Ош мамлекеттик университети";
        String[] words = text.split("\\s*(\\s|,|!|\\.)\\s*");
        for(String word : words){
          System.out.println(word);
        }

}
```

Мында "\\s*(\\s|,|!|\\.)\\s*" регулярдык туюнтмасы сүйлөмдү сөздөргө бөлүү үчүн колдонулат. "\\s" туюнтмасы боштукту билдирет. Жылдызча (*) символдун 0дөн чексиз санда болушу мүмкүн экенин көрсөтөт б.а. \\s* деп удаалаш кеткен белгисиз сандагы боштуктарды алабыз же сөздөрдүн арасында канча боштук (пробел) бар экендиги маанилүү эмес. Вертикалдык сызыкча аркылуу боштуктан башка сөздөрдү ажыратуучу символдор көрсөтүлдү б.а. "\\s|,|!|\\" тобунда сап боштукка, үтүргө, илеп

белгисине же чекитке дал келиши мүмкүн. Мындан тышкары, чекит өзгөчө символ болгондуктан, биз чекиттин белгисин айтып жатканыбызды көрсөтүү үчүн, чекиттин алдына сызыктарды коебуз.

Программанын жыйынтыгы:

```
Ош
мамлекеттик
университети
```

String классынын дагы бир методу matches() регулярдуу туюнтманы алат жана жолчо туюнтмага дал келсе, true маанисин кайтарат, антпесе false маанисин кайтарат.

Мисалы, жолчо телефон номерине дал келгенин текшерип көрөлү:

```
String input = "+12343454556";
boolean result = input.matches("(\\+*)\\d{11}");
if(result){
   System.out.println("Бул телефон номер");
}
else{
   System.out.println("Бул телефон номер эмес!");
}
```

Бул учурда, регулярдуу туюнтма алгач "(\\+*)" тобун аныктайт б.а. башында плюс белгиси болушу мүмкүн, бирок ал жок болушу да мүмкүн. Андан кийин, биз кийинки 11 белги сандарга дал келеби же жокпу, карап көрөлү. "\\d" туюнтмасы сандык символду билдирет, ал эми фигуралык кашаадагы сан - {11} - бул символдун канча жолу кайталанышы экендигин көрсөтөт б.а. башында плюс белгиси болушу мүмкүн (же ал жок болушу мүмкүн), андан кийин 11 цифра бар туюнтманы издеп жатабыз.

Бышыктоочу суроолор:

1. Жолчолор кайсы класстын объекттери болот?
2. Жолчолорду кошуучу кайсы методдор бар?
3. Жолчодон издөөчү кайсы методдор бар?
4. Жолчолорду салыштыруучу методдор кайсылар?
5. StringBuffer жана StringBuilder класстарынын кандай методдору бар?
6. Регулярдык туюнтмаларга мисал келтиргиле.

8 - практикалык сабак

Максаты: Жолчолор жөнүндө алган маалыматтарды бышыктоо жана маселелерди чыгаруу.

Келгиле, чогуу төмөнкү маселени карап көрөлү:

Сөздөрдүн саны

Бизге сөздөрдөн турган текст берилсин. Жалаң латын тамгаларынан турган сөздөрдүн санын табуучу программа түзүңүз.

Кийирилүүчү маалымат

Бир канча сөздөрдөн турган текст.

Чыгуучу маалымат

Бүтүн сан – латын тамгаларынан турган сөздөрдүн саны

Чыгаруу:

```
package progs;
import java.lang.*;
import java.util.regex.Matcher;
import java.util.regex.Pattern;
import java.util.Scanner;

public class Example {
```

```
    public static void main(String[] args) {
        Scanner obj=new Scanner(System.in);
      // Кийирилүүчү текст
      String text = obj.nextLine();

      // латын тамгаларын камтыган создор учун патерн
      Pattern pattern = Pattern.compile("\\b[a-zA-Z]+\\b");
      Matcher matcher = pattern.matcher(text);

      // сөздөрдүн санын сактоочу өзгөрмө
      int count = 0;

      // сөздөрдү издөө жана санын алуу
      while (matcher.find()) {
        count++;
      }

      // сөздөрдүн санын экранга чыгаруу
      System.out.println("Латын тамгаларын камтыган создордун саны: " + count);
    }
}
```

Программанын жыйынтыгы:

```
Hello, world! This is a test with some Latin and Cyrillic words like кат, шам, and дөң
Латын тамгаларын камтыган создордун саны: 16
```

Өз алдынча иштөөгө мисалдар

1. Эң аз сандагы символдордон түзүлгөн сөз

Ар кандай символдордун саны эң аз болгон сөздү табыңыз. Сөз тамгаларды жана сандарды камтышы мүмкүн. Андай сөздөр бир нече болсо, биринчисин табыңыз. Мисалы, "fffff ab f 1234 jkjk" сабында табылган сөз "ffffff" болушу керек.

2. Кашаалардын саны

Ар кандай символдордон турган жолчо берилсин. Жолчодогу тегерек кашаалардын санын тапкыла.

Мисалы, “2+5+(7+9)” сабында тегерек кашаалардын саны 2 ге барабар.

3. Орун алмаштыруу

Бир канча сиволдордон турган жолчо берилсин. Жолчодогу эң биринчи символ менен эң акыркы символдун ордун алмаштыргыла.

4. Сөздөрдүн саны

Кийирилген текстте канча сөз бар экендигин экранга чыгаргыла.

Мисалы, “I live in Osh” текстинде 4 сөз бар.

5. Символду тап

Бир канча сиволдордон турган жолчо берилсин. Берилген жолчодо “k” тамгасы канчанчы тураарын тапкыла.

Мисалы, “book” сабында 4 орунда турат.

6. Жолчо

'a'дан 'z'ге чейинки 80 символдон ашпаган кичинекей латын тамгаларынан турган жолчо берилсин. Бул жолчодон колдонуучу кийирген 30 символдон ашпаган сөздү табышыңыз керек. Эгерде бар болсо “Yes”, антпесе “No” сөзүн чыгарыңыз.

Мисалы:

Кийирилүүчү маалымат	Чыгуучу маалымат
abcdefg bcd	Yes
abcdefg bde	No

7. Сөздөр

Берилген M, 2≤M≤2019 натуралдык саны боюнча:

S, A, Y тамгалардан турган, M тамганы камтыган, төмөнкү шарттарды канаатандыруучу канча «сөз» бар?

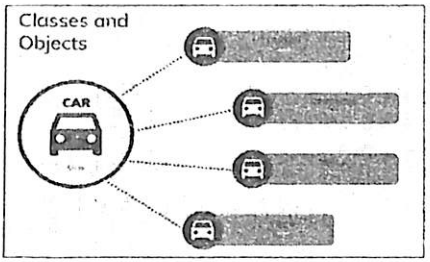
- «сөздө» бир гана үндүү бар;
- үч үндүү катары менен болбойт;
- эки үндүүнүн айкалыштарынан AA гана болуш мүмкүн.

Ал санды 1000ге бөлүп, калдыкты чыгаргыла

Мисалы, M=2: AA, SA, SY, AS, YS

Кийирилүүчү маалымат	**Чыгуучу маалымат**
2	5

Лекция 10. Класстар. Объектке багытталган программалоо



Сабактын максаты:

1. *Класстар жана объекттер жөнүндө түшүнүк алуу*
2. *Конструкторлор*
3. *Инициализаторлор*
4. *Пакеттер*

Класс жана объекттер

Java тилиндеги класс объектти түзүү үчүн шаблон, ал эми объект класстын үлгүсү. Класста өзгөрмөлөр жана методдор бар, алар класстын элементтери же мүчөлөрү деп аталат. Ал Java тилинде

инкапсуляциянын негизин түзөт. Берилген класстын ар бир объекти класс тарабынан аныкталган структураны жана методдорду камтыйт.

Java тилиндеги ар бир программаны бири-бири менен өз ара аракеттенүүчү объекттердин жыйындысы катары карасак болот. Мисалы, баарыбызда адам жөнүндө кандайдыр бир түшүнүк бар - эки колу, эки буту, башы, денеси ж.б. бөлүктөрдөн турган белгилүү бир шаблонду элестетебиз. Бул шаблонду класс деп атоого болот. Ал эми чындыгында жашап жаткан адам ошол класстын экземпляры же ошол класстын объекти десек болот.

Классты түзүү

Класс class ачкыч сөзүнүн жардамында түзүлөт:

```
class Person{

}
```

Бул жерде Person деп аталган класс түзүлдү. Класстын атынан кийин фигуралык кашаалардын ичине класстын телосу же болбосо анын касиеттери жана методдор жайгашат. Ар бир объекттин эки түрдүү мүнөздөмөсү болот: объектке тиешелүү мүнөздөмөлөрдү сактаган өзгөрмөлөр жана объект аткаруучу жумуштар (методдор). Төмөнкү классты карап көрөлү:

```
class Person{

   String name;      // аты
   int age;          // жашы

   void info(){
      System.out.printf("Name: %s \tAge: %d\n", name, age);
   }
}
```

Person (Адам) классы эки талаадан турат: name адамдын атын билдирет, ал эми age адамдын жашын билдирет. Ал эми info методу да аныкталган, ал эч нерсе кайтарбайт жана адамдын аты, жашы жөнүндөгү маалыматты жөн гана консолго чыгарат.

Эми келгиле, ушул классты колдонуп көрөлү. *Java тилинде ар бир класс өзүнчө файлда сакталат жана файлдын аты негизги класстын аты менен дал келет.* Бирок биз төмөнкү учурда эки классты бир файлда түшүндүрүү максатында жазып алдык. Мындай учурда бир класс гана public модификаторун алат жана файл негизги класстын аты менен сакталат. *Негизги класс бул main методу бар класс болот!*

```
public class Example32 {

        public static void main(String[] args) {
                Person adam;
        }
}

class Person{

   String name;
   int age;
   void info(){
     System.out.printf("Name: %s \tAge: %d\n", name, age);
   }
}
```

Жогоруда айткандай негизги класс бул учурда Example32 болот. Класс жаңы типти билдирет, ошондуктан бул типтеги өзгөрмөлөрдү түзө алабыз. Биз Person тибиндеги adam өзгөрмөсүн түзүп алдык. Анын мааниси азырынча null (эч нерсе жок). Ал эч кайсы объектке байлана элек.

Конструктор

Класста негизги методдордон башка **конструктор** деп аталган атайын методдор да жайгашат. Берилген класстын жаңы объектиси түзүлгөндө конструкторлор чакырылат б.а. конструктор объект түзүлгөндө автоматтык түрдө иштөөчү метод болуп эсептелет. Конструкторлор объекттин инициализациясын аткарышат. Эгерде класста конструктор аныкталбаса, ал класс үчүн автоматтык түрдө параметрсиз конструктор түзүлөт.

Жогоруда биз караган мисалда (Example32) конструктор түзүлгөн жок, ошондуктан ал үчүн автомагтык түрдө конструктор түзүлөт. Ушул мисалды кайрадан карап чыгалы жана объект түзөлү:

```
public class Example32 {

    public static void main(String[] args) {

        Person adam = new Person(); // объектти түзүү
        adam.info();

        // атын жана жашын өзгөртүү
        adam.name = "Asan";
        adam.age = 20;
        adam.info();
    }
}
class Person{

    String name;   // аты
    int age;       // жашы
    void info(){
      System.out.printf("Name: %s \tAge: %d\n", name, age);
    }
}
```

Жогоруда байкаганыңыздай жаңы объектти түзүү үчүн new ачкыч сөзү колдонулду (23-сүрөт).



Сүрөт 23. Объектти түзүү

Эгерде конструктор объекттин өзгөрмөлөрүнүн маанилерин ыйгарбаса, аларга демейки (по умолчанию) маанилер берилет. Сандык типтеги өзгөрмөлөр үчүн бул 0 саны, ал эми жолчо (String) типтери жана класстар үчүн null мааниси ыйгарылат.

Эгерде класстын талаалары анык бир маанини алуусун карай турган болсоңуз, анда өзүңүздүн конструкторуңузду түзө аласыз. Мисалы:

```
public class Example33 {


    public static void main(String[] args) {
      // параметрлери жок биринчи конструкторду чакыруу
      Person adam1 = new Person();
      adam1.info();

      // бир параметри бар конструкторду чакыруу
   Person adam2 = new Person("Asan");
      adam2.info();

         // эки параметри бар конструкторду чакыруу
      Person adam3 = new Person("Bolot", 25);
      adam3.info();
    }
}
class Person{

    String name;   // аты
    int age;       // жашы
    Person()
    {
```

```
        name = "Undefined";
        age = 18;
    }
    Person(String n)
    {
        name = n;
        age = 18;
    }
    Person(String n, int a)
    {
        name = n;
        age = a;
    }
    void info(){
        System.out.printf("Name: %s \tAge: %d\n", name, age);
    }
}
```

Программанын жыйынтыгы:

```
Name: Undefined         Age: 18
Name: Asan      Age: 18
Name: Bolot     Age: 25
```

Эми класста үч конструктор аныкталган, алардын ар бири ар кандай сандагы параметрлерден турат. Биз объектти түзүүдө кайсы конструкторду колдонсок, класстын талааларына тиешелүү маанилер ыйгарылат.

this ачкыч сөзү

this ачкыч сөзү класстын учурдагы объектисине шилтемени билдирет. Бул ачкыч сөз аркылуу биз өзгөрмөлөргө, объекттин методдоруна кайрыла алабыз, ошондой эле анын конструкторлорун чакыра алабыз. Мисалы:

```
class Person{

    String name;
    int age;
    Person()
    {
        this("Undefined", 18);
    }
```

```
  Person(String name)
  {
    this(name, 18);
  }
  Person(String name, int age)
  {
    this.name = name;
    this.age = age;
  }
  void info(){
    System.out.printf("Name: %s \tAge: %d\n", name, age);
  }
}
```

Үчүнчү конструктордо параметрлер класстын талаалары менен бирдей аталгандыктан аларды айырмалоо үчүн this ачкыч сөз колдонулат:

this.name=name;

Бул жерде name параметринин мааниси name талаасына ыйгарылат.

Инициализаторлор

Конструктордон тышкары, объекттин баштапкы инициализациясы объекттин инициализатору аркылуу жүргүзүлүшү мүмкүн. Инициализатор ар кандай конструктордон мурун аткарылат б.а. инициализатордо биз бардык конструкторлор үчүн жалпы кодду жайгаштырсак болот:

```
public class Example34 {

        public static void main(String[] args) {

          Person adam1 = new Person();
          adam1.info();

          Person adam2 = new Person("Asan");
          adam2.info();
```

```
    }
class Person{

  String name;   // аты
  int age;       // жашы

  /*инициализатор блогунун башы*/
  {
    name = "Undefined";
    age = 18;
  }
  /*инициализатор блогунун аягы*/

  Person(){

  }
  Person(String name){

    this.name = name;
  }
  Person(String name, int age){

    this.name = name;
    this.age = age;
  }
  void info(){
    System.out.printf("Name: %s \tAge: %d\n", name, age);
  }
}
```

Программанын жыйынтыгы:

```
Name: Undefined         Age: 18
Name: Asan      Age: 18
```

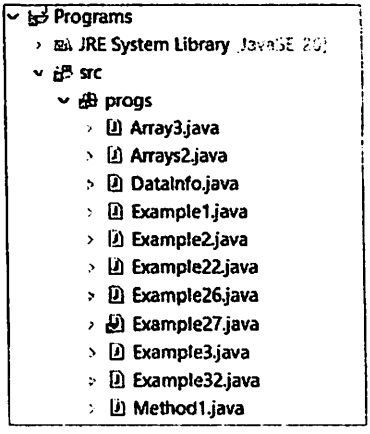
Пакеттер

Java тилинде класстар пакеттерге бириктирилет. Эсиңиздерде болсо буга чейин сырттан класстарды кошууда ал класстар жайгашкан пакеттерди көрсөткөнбүз. Аларга java.lang, java.util, java.io ж.б. пакеттерди айтсак болот.

Программисттер окшош аттагы класстарды көп түзүп алышат. Пакетти кошуу бул конфликттик учурларды алдын алуу үчүн колдонулат жана аттардын түшүнүктүү болуусун камсыздайт. Класстын кайсы пакетке тиешелүү экендигин көрсөтүү үчүн package директивасын кошуп, пакетти көрсөтөбүз:

package пакеттин_аты;

Эреже катары, пакеттердин аталыштары проекттин физикалык структурасына, башкача айтканда, баштапкы код файлдары жайгашкан каталогдордун жайгашуу ордуна туура келет. Ал эми проекттин ичиндеги файлдардын адреси бул файлдардын пакетинин атына туура келет. Мисалы, класстар **progs** пакетине таандык болсо, анда бул класстар долбоордун **progs** папкасына жайгаштырылат(24-сүрөт).



Сүрөт 24. Пакеттин структурасы

Эгерде класс үчүн пакет аныкталбаса, анда класс аты жок пакетте жайгашкан деп эсептелет.

```
package progs;

public class Example34 {
```

```
    public static void main(String[] args) {
      // операторлор

    }
}

class Person{
//класстын телосу
}
```

Бул структурада progs пакетинин ичинде эки класс камтылган: Example34, Person.

Пакеттерди импорттоп алсак болот. Жогоруда колдонгон Scanner классын карап көрөлү:

```
package progs;

import java.util.Scanner;
public class Example35 {

    public static void main(String[] args) {
        int d;
        Scanner obj=new Scanner(System.in);
        d=obj.nextInt();
        System.out.println(d);
        obj.close();
    }
}
```

Мисалда көрүнүп тургандай import директивасы менен Scanner классын java.util пакетинен программага кошуп алдык. java.util пакетиндеги бардык класстарды кошуп алуу (импорттоо) үчүн төмөнкүдөй жазабыз:

```
import java.util.*;
```

Эми биз бул пакеттеги бардык класстарды колдоно алабыз.

Биз эки башка пакеттен бир эле аталыштагы эки классты колдонуп жатсак, мисалы, Date классы java.util пакетинде да, java.sql пакетинде да кездешет. Бул эки классты бир эле учурда колдонуу

керек болсо, анда проектте бул класстарга толук адрести көрсөтүү керек:

```
java.util.Date utilDate = new java.util.Date();
java.sql.Date sqlDate = new java.sql.Date();
```

static модификатору

Java тили статикалык импорт деп аталган импорттун өзгөчө формасына ээ. Аны колдонуу үчүн static модификаторун көрсөтүү керек:

```
import static java.lang.System.*;
import static java.lang.Math.*;
```

Бул жерде System жана Math класстарында статикалык методдор бар болгондуктан static модификатору кошулуп жазылды. static модификатору коюулгандыктан бул класстардын методдору класстын аты жазылбай колдонулат:

```
package progs;

import static java.lang.System.*;
import static java.lang.Math.*;

public class Example36 {

   public static void main(String[] args) {

      double result = sqrt(20);
      out.println(result);
   }
}
```

Программанын жыйынтыгы:

```
4.47213595499958
```

Бышыктоочу суроолор:

1.Класс деп эмнени айтабыз?

2. Объект деп эмнени айтабыз?

3. Класстын талаалары жана методдору деген эмне?

4. Конструктор деп эмнени айтабыз?

5. Инициализатор кандай жумуш аткарат?

6. Пакеттер эмне үчүн колдонулат?

7. static модификатору кайсы убакта коюулат?

9-практикалык сабак

Максаты: Класстар жана объекттер жөнүндө алган маалыматтарды бышыктоо жана маселелерди чыгаруу.

Келгиле, чогуу төмөнкү маселени карап көрөлү:

Төмөнкүлөрдү камтыган Circle классын түзүңүз: 1) радиус өзгөрмөсүн; 2) тегеректин аянттын жана узундугун эсептөөчү метод түзгүлө; 3) PI константасын *Math* классынан алсаңыз болот.

Бул класстын бир нече объекттерин түзүңүз.

Чыгаруу:

Circle аталышындагы класс түзүп алалы. Ага конструктор кошолу жана радиусту параметр катары жөнөтөлү.

```
package progs;

public class Circle {
        private double radius;

          Circle(double radius) {
            this.radius = radius;
          }

          double getArea() {
```

```
        double area = Math.PI * radius * radius;
        return area;
    }
    double getLength() {
        double len=2*Math.PI*radius;
        return len;
    }
    public static void main(String[] args) {
        Circle myCircle = new Circle(10.5);
        System.out.println(myCircle.getArea());
        System.out.println(myCircle.getLength());

    }
}
```

Тапшырмада айтылгандай эки метод түзүлдү. Аянтын алуу үчүн getArea методу жана узундугун эсептөө үчүн getLength методдору колдонулат. Негизги методдо Circle классынын myCircle объектиси 10.5 радиус менен түзүлдү жана методдор чакырылды.

Программанын жыйынтыгын карап көрөлү:

```
346.36059805827474
65.97344572538566
```

Өз алдынча иштөөгө мисалдар

1. Төмөнкүлөрдү камтыган Person классын түзүңүз: а) fullName, age талаалары. б) move() жана talk() методдору в) Person() жана Person(fullName, age) эки конструкторун кошуңуз. Бул класстын эки объектисин түзүңүз. Бир объект Person() конструктору менен, экинчиси Person(fullName, age) менен инициализацияланат.
2. «Матрица» классын түзүңүз. Класста төмөнкү талаалар болушу керек: 1) чыныгы сандардын эки өлчөмдүү массиви; 2) матрицадагы жолчолордун жана мамычалардын саны. Класста

төмөнкү методдор болушу керек: 1) башка матрица менен кошуу; 2) санга көбөйтүү; 3) басып чыгаруу.

3. Китепкананын колдонуучусу жөнүндө төмөнкү маалыматтарды сактаган классты түзүңүз: аты-жөнү, китепкана картасынын номери, факультет, туулган датасы, телефону. Класстын методдору takeBook()- китепти алгандыгы, returnBook()- китепти кайтаргандыгы тууралуу маалыматты сактап турат. Бул класстын объекттеринен турган массивди түзүүчү программаны иштеп чыгыңыз. Программада окурмандар жөнүндө маалыматтарды кийирүүнү жана чыгарууну уюштуруңуз.
4. Car классын com.company.vehicles пакетинде, Engine классын com.company.details пакетинде жана Driver классын com.company.professions пакетинде түзүңүз. Driver классы төмөнкү талаалардан турат - толук аты-жөнү, айдоо тажрыйбасы. Engine классы кубаттулук, өндүрүүчү талааларынан турат. Car классы автомобилдин маркасы, автомобилдин классы, салмагы, Driver тибиндеги айдоочу, Engine тибиндеги мотордон турат. start(), stop(), turnRight(), turnLeft() методдору печатка төмөнкү сөздөрдү чыгарсын: "Кеттик", "Токто", "Оңго бур" же "Солго бур". Ошондой эле автомобиль, айдоочусу, мотору жөнүндө толук маалымат чыгаруучу printInfo() методун кошуңуз.
5. Гүлдүн аталышын жана баасын камтыган "Flower" классын түзүңүз. Баасынан айырмаланган 3 букет түзүңүз (массив колдонуңуз). Ошондой эле сатылган гүлдөрдүн санын эсептеңиз (статикалык метод).

Лекция 11. Объектке багытталган программалоонун негизги түшүнүктөрү



Сабактын максаты:

1. *Мүмкүндүк алуу модификаторлору жана инкапсуляция жөнүндө түшүнүк алуу*
2. *Объекттер методдун параметри катары*
3. *Камтылуучу класстар жөнүндө маалымат алуу*
4. *Мурастоо принциби жөнүндө таанышуу*
5. *Абстракттык класстар менен таанышуу*
6. *Полиморфизм түшүнүгү*

Мүмкүндүк алуу модификаторлору

Java тилинде класстын атрибуттары жана методдору мүмкүндүк алуу(аларга кайрылуу) модификаторлоруна ээ. Өзүңөр билгендей биз буга чейин **public** модификатору менен таанышканбыз. Ал класстын атрибуту же методу кайрылууга жана колдонууга ачык экендигин көрсөтүп турат б.а. аларга башка класстардан кайрыла алабыз. Класстар үчүн **public** (ачык) же **default** (модификатор жок болсо) модификаторду колдоно аласыз. public болгондо башка класстар үчүн ачык экендигин көрсөтөт. **default**

болгондо бир пакеттин ичиндеги класстар үчүн гана колдонууга боло турганын көрсөтүп турат. Мисалы:

```
class MyClass {
  public static void main(String[] args) {
    System.out.println("Hello World");
  }
}
```

Бул мисалда класстын модификатору жок. Ошондуктан бул класска башка пакеттен кайрылууга болбойт.

Негизинен атрибуттар, методдор жана конструкторлор үчүн төмөнкү модификаторлордун бирин колдоно аласыз:

Модификатор	Мааниси
public	Код бардык класстар үчүн жеткиликтүү
private	Код жарыяланган класстын ичинде гана жеткиликтүү
default	Код бир эле пакетте гана жеткиликтүү. Класстын атрибуттары же методу үчүн модификатордун жоктугу ага демейки(default) модификатору колдонуларын билдирет.
protected	Код бир эле пакетте жана субкласстарда жеткиликтүү.

Төмөнкү мисалдарды карап көрөлү. Биз 2 java файл менен иштейбиз. Main.java файлында Main классын жана талааларын public модификаторун колдонуп жарыялар алалы:

```
public class Main {
```

```
 public String name = "Anna";
 public String adress = "city Osh";
 public String email = "anna@gmail.com";
 public int age = 20;
}
```

Бул жерде класс public болуп турат. Анын ичинде name, address, email, age талааларынын модификатору да public б.а. ачык экендигин көрүп турабыз.

Example37.java деп аталган экинчи java файлды карап көрөлү:

```
class Example37 {
  public static void main(String[] args) {
    Main myObj = new Main();
    System.out.println("Аты: " + myObj.name);
    System.out.println("Адреси: " + myObj.adress);
    System.out.println("Почтасы: " + myObj.email);
    System.out.println("Жашы: " + myObj.age);

}
```

Бул мисалдан Example37 классында Main классынын объектисин түзүп алдык жана объектин атрибуттарын экранга чыгарып алдык.

Программанын жыйынтыгы:

```
Аты: Anna
Адреси: city Osh
Почтасы: anna@gmail.com
Жашы: 20
```

Программанын жыйынтыгы көрсөткөндөй Main классындагы атрибуттар public болгондуктан, аларга кайрылуу жана маанилерди алуу ачык ишкеашты. Эгерде ушул эле мисалда

```
public String adress = "city Osh";
```

код сабын

```
private String adress = "city Osh";
```

деп өзгөртсөк, анда программа "Main.adress талаасы көрүнбөй жатат" деп маалымат чыгарат. Ооба сиз туура байкадыңыз address талаасы private модификаторунда турат, ошондуктан ал талаага түздөн-түз кайрылуу мүмкүн эмес.

Инкапсуляция

Биз жогоруда карагандай класстага, атрибуттарга жана методдорго кайрылууну башкаруу үчүн атайын модификаторлор колдонулат. "Эмне үчүн баарын public деп түзүп алсак болбойт" деп жатсаңыз керек. Ооба андай кылсак болмок, бирок мисалы, жашты чагылдырган age талааны алалы. Эгерде башка класстын бул талаага түздөн-түз кирүү мүмкүнчүлүгү бар болсо, анда программанын иштөө учурунда ага туура эмес маани, мисалы, терс сан ыйгарылып калуусу мүмкүн. Сиз билгендей age талаасы терс маани албайт. Ошондуктан мындай маалыматты өзгөртүү сунушталбайт. Ушуга байланыштуу, аны сырттан келген керексиз кайрылуудан коргоо үчүн (маалыматты алуу үчүн да, аны өзгөртүү үчүн да) мүмкүн болушунча аларга кирүү мүмкүнчүлүгүн чектөө сунушталат. Ар кандай модификаторлорду колдонуу маалыматтарды өзгөртүлбөй, туура сакталуусун камсыз кылат. Берилиштерди ушундай жол менен жашыруу **инкапсуляция** деп аталат.

private модификатору менен сакталган талаалардын маанилерин алуу жана өзгөртүү үчүн атайын методдор кошулат. Төмөнкү мисалды карап көрөлү:

```
package progs;

public class Example38{

   public static void main(String[] args) {

      Person anna = new Person("Ana", 30);
      System.out.println(anna.getAge());
      anna.setAge(33);
      System.out.println(anna.getAge());
      anna.setAge(123450);
      System.out.println(anna.getAge());
   }
}

class Person{

   private String name;
   private int age = 1;

   public Person(String name, int age){

      setName(name);
      setAge(age);
   }
   public String getName(){
      return this.name;
   }
   public void setName(String name){
      this.name = name;
   }
   public int getAge(){
      return this.age;
   }
   public void setAge(int age){
      if(age > 0 && age < 110)
         this.age = age;
   }
}
```

Программанын жыйынтыгы:

```
30
33
33
```

Бул мисалда Person классынын 2 талаасы тең private модификаторунда жарыяланып турат б.а. аларга башка класстан түздөн-түз кайрылууга жана маанисин өзгөртүүгө болбойт. Ал талаалардын маанилерин алуу үчүн getName, getAge методдору колдонулат. Бул методдорду аксессорлор (accessor) байланыш түзүүчү деп коюушат. Ал эми маанисин өзгөртүү үчүн setName, setAge методдору колдонулду. Байкап көрсөңүз setAge методунда объектке жаш ыйгарууда атайын шарт текшерилип, жөнөтүлгөн жаш 0 дөн чоң жана 110 дон кичине болсо гана ыйгарылат. Бул методдорду мутаторлор (mutator) деп коюшат, анткени алар талаанын маанисин өзгөртүп салат.

Эмне үчүн инкапсуляцияны колдонобуз?

- Класстын атрибуттарын жана методдорун жакшыраак көзөмөлдөө үчүн.
- Класстын атрибуттары окуу үчүн гана (эгерде сиз get методун гана колдонсоңуз) же жазуу үчүн гана (эгерде сиз set методун гана колдонсоңуз) түзүлүүсү мүмкүн.
- Ийкемдүү: программист башка бөлүктөрүнө таасир этпестен коддун бир бөлүгүн өзгөртө алат.
- Маалыматтын коопсуздугу жогорулайт.

Объекттер методдун параметри катары

Класстын объектисин башка примитивдик типтеги маалыматтар сыяктуу эле методдорго параметр катары жөнөтсөк болот. Бул учурда бир өзгөчөлүк, методго объект жайгашкан эстутумдагы аймакка шилтеменин көчүрмөсү берилет. Мисалы:

```
package progs;
public class Example40{
    public static void main(String[] args) {

     Person adam = new Person("Anna");
     System.out.println(adam.getName());    // Anna
     changeName(adam);
     System.out.println(adam.getName());    // Vera
   }
   static void changeName(Person p){
      p.setName("Vera");
   }
}
class Person{

   private String name;

   Person(String name){
      this.name = name;
   }
   public void setName(String name){
      this.name = name;
   }
   public String getName(){

      return this.name;
   }
}
```

Программанын жыйынтыгы:

```
Anna
Vera
```

Бул жерде changeName методуна атын өзгөртүү үчүн Person классынын объекти жөнөтүлдү. adam өзгөрмөсү жана changeName методунун p параметри эстутумдагы бир эле объектти көрсөтөт. Ошондуктан, метод аткарылгандан кийин, методго өткөрүлүп берилген adam объектинин аты "Anna" аталышынан "Vera" болуп өзгөрөт.

Мурастоо (Наследование)

Java тилинде сиз атрибуттарды жана методдорду бир класстан экинчисине мурастай аласыз.

Мурастоо принцибинде класстар эки түрдүү болот:

1. **Туунду класс** – башка класстан атрибуттарды жана методдорду мурастап алган класс.
2. **Суперкласс** – атрибуттарды жана методдорду мурастоочу класс.

Мурасталган классты түзүү үчүн extands ачкыч сөзү колдонулат. Ал класс супер класстын бардык атрибуттарын жана методдорун, алар башка пакетте жайгашса да, кабыл алат. Бир гана учурда, атрибутар жана методдор private болсо, кабыл ала албайт. Төмөнкү мисалда Car туунду классы Vehicle супер классынан атрибуттарды жана методдорду мурастап алат:

```
class Vehicle {
  protected String brand = "Ford";//Vehicle классынын атрибуту
  public void honk() {          // Vehicle классынын методу
        System.out.println("Tuut, tuut!");
      }
    }

 class Car extends Vehicle {
  private String modelName = "Mustang"; // Car классынын
                                        //атрибуту
```

```
    public static void main(String[] args) {

      // Объект түзүү
      Car myCar = new Car();

      // myCar объекти үчүн суперкласстын honk методун чакыруу
      myCar.honk();

         // Суперкласстан мурастап алган brand атрибутун жана //субкласстын
         modelName атрибутун экранга чыгаруу
      System.out.println(myCar.brand + " " + myCar.modelName);
    }
  }
```

Программанын жыйынтыгы:

```
Tuut, tuut!
Ford Mustang
```

Vehicle классындагы protected модификаторун байкадыңызбы? Ооба brand атрибуту protected болуп турат. Эгерде ал жерде private болгондо Car классынын объектиси ал атрибутка кайрыла алмак эмес.

Эгерде базалык класста конструкторлор бар болсо, анда туунду класстын конструкторунда super ачкыч сөздү колдонуу менен базалык класстын конструкторлорунун бирин чакыруу керек.

Конструкторду чакырганда super деген сөздөн кийин берилген аргументтер кашаанын ичинде келтирилет. Бул учурда, базалык класстын конструкторуна чакыруу туунду класстын конструкторунун эң башында болушу керек. Мисалы:

```
package progs;

public class Example39{

  public static void main(String[] args) {

    Person adam1 = new Person("Asan");
```

```
        adam1.display();
        Employee adam2 = new Employee("Bolot");
        adam2.display();
    }
}

class Person {

    String name;
    public String getName(){ return name; }

    public Person(String name){

        this.name=name;
    }

    public void display(){

        System.out.println("Аты: " + name);
    }
}
class Employee extends Person{
    public  Employee(String name){
        super(name); // эгерде базалык класста
                              //конструктор аныкталса,
                //анда туунду класста аны чакыруу керек
    }
}
```

Программанын жыйынтыгы:

```
Аты: Asan
Аты: Bolot
```

Мурастоо принциби эмнеси менен жакшы?

- Бул кодду кайра колдонуу үчүн пайдалуу: жаңы классты түзгөндө бар класстын атрибуттарын жана ыкмаларын кайра колдонуу үчүн пайдаланса болот.

Абстрактуу класстар

Класстарга кайрылууну толук жабуу үчүн төмөнкү модификаторлор колдонулат:

Модификатор	Мааниси
final	Класс башка класстарга мурасталбай калат
abstract	Класс объекттерди түзүү үчүн колдонулбайт (Абстрактуу класска кирүү үчүн, ал башка класска мурастоочу класс болуш керек)

Төмөнкү мисалда final модификаторун карап көрөлү:

```
package progs;

final class Vehicle {
        protected String brand = "Ford";
        public void honk() {
          System.out.println("Tuut, tuut!");
        }
       }

       class Example35 extends Vehicle {
        private String modelName = "Mustang";
        public static void main(String[] args) {
          Example35 myFastCar = new Example35();
          myFastCar.honk();
          System.out.println(myFastCar.brand + " " + myFastCar.modelName);
        }
       }
```

Программанын жыйынтыгы:

```
Exception in thread "main" java.lang.Error: Unresolved compilation problems:
        The method honk() is undefined for the type Example35
        brand cannot be resolved or is not a field
```

Программанын жыйынтыгы көрсөткөндөй final модификатору колдонулган класстар мурасталбайт.

Төмөнкү мисалда abstract модификаторун карап көрөлү:

```
package progs;

//abstract класс
abstract class Main {
public String fname = "Asan";
public String lname = "Arapov";
public String email = "asan@gmail.com";
public int age = 20;
public abstract void study(); // abstract метод
}

//туунду класс
class Student extends Main {
public int graduationYear = 2024;
public void study() {
 System.out.println("Эртеден кечке окуу");
}
}
```

Бул Main.java файлы. Бул жерде абстрактуу класс түзүлгөн жана анын объектин түзүү үчүн мурасталган Student туунду классын түзүп алдык. Төмөнкү мисалда биз Student классынын объектин түзүүнү карайбыз:

```
package progs;

class Example35 {
        public static void main(String[] args) {
// Student классынын объектин түзүү (Main классынан //атрибуттарды жана
ыкмаларды мурастайт)
                Student myObj = new Student();

         System.out.println("Name: " + myObj.fname + " " + myObj.lname);
         System.out.println("Email: " + myObj.email);
```

```
        System.out.println("Age: " + myObj.age);
        System.out.println("Graduation Year: " + myObj.graduationYear);
        myObj.study(); // абстрактуу методду чакыруу
    }
}
```

Программанын жыйынтыгы:

```
Name: Asan Arapov
Email: asan@gmail.com
Age: 20
Graduation Year: 2024
Studying all day long
```

Полиморфизм

Полиморфизм сөзү "көп формалуу" дегенди билдирет, жана мурастоо принциби менен байланышкан көп класстардын үстүнөн иштөөдө пайда болот. Биз жогоруда айткандай мурастоо принцибинде башка класстын атрибуттарын жана методдорун мурастап алууга мүмкүндүк берилет. Ал эми полиморфизм бул методдорду ар кандай амалдарды аткаруу үчүн колдонот. Бул бизге бир эле аракетти ар кандай жолдор менен аткарууга мүмкүндүк берет. Мисалы, Animal (жаныбар) деген суперкласс болсун дейли. Анын жаныбардардын үнүн чагылдырган animalSound() методу болсун. Анда мышык, ит, канаттуу деген туунду класстардын ар биринде ушул метод болуп, ар бири ар кандай үндү чагылдырат (мышык мыёлойт, ит үрөт ж.б.).

```
class Animal {
    public void animalSound() {
      System.out.println("Ар бир жаныбардын үнү бар");
    }
}
```

```
class Cat extends Animal {
  public void animalSound() {
    System.out.println("Мышыктын үнү: мяу, мяу");
  }
}

class Dog extends Animal {
  public void animalSound() {
    System.out.println("Иттин үнү: вав, вав");
  }
}

class Main {
  public static void main(String[] args) {
    Animal myAnimal = new Animal();
    Animal myCat = new Cat();
    Animal myDog = new Dog();
    myAnimal.animalSound();
    myCat.animalSound();
    myDog.animalSound();
  }
}
```

Мисалда көрсөтүлгөндөй ар бир класстын өзүнүн объектиси түзүлдү жана ар бири үчүн ар кандай жумуш аткаруучу animalSound() методу чакырылды.

Камтылуучу класстар

Java тилинде камтылуучу (класстын ичиндеги) класстар да болот. Камтылуучу класстарды түзүүнүн максаты - бири-бирине таандык класстарды топтоштуруу болуп эсептелет жана бул сиздин кодуңузду окууга жана башкарууга ыңгайлуу кылат.

Ички класска кирүү үчүн тышкы класстын объектисин түзүп, андан кийин ички класстын объектисин түзүңүз:

```
class OuterClass { //сырткы класс
    int x = 10;
```

```
  class InnerClass {   //ички класс
    int y = 5;
  }
}

public class Main {
  public static void main(String[] args) {
    OuterClass myOuter = new OuterClass();
    OuterClass.InnerClass myInner = myOuter.new InnerClass();
    System.out.println(myInner.y + myOuter.x);
  }
}
```

Жөнөкөй класстардан айырмаланып, ички класстар private же protected модификаторунда болушу мүмкүн. Эгерде ички класстын атрибуттарына кайрылууну чектегиңиз келсе, анда ички классты private модификатору менен түзүп алыңыз. Жогорудагы мисалды төмөнкүдөй өзгөртсөк болот:

```
class OuterClass {
  int x = 10;
  private class InnerClass {
    int y = 5;
  }
}
```

Ошондой эле ички класстар static да болушу мүмкүн. Бул учурда ички класска анын объектисин түзбөй эле кайрыла алабыз:

```
class OuterClass {
  int x = 10;

  static class InnerClass {
    int y = 5;
```

```
    }
}

public class Main {
  public static void main(String[] args) {
    OuterClass.InnerClass myInner = new OuterClass.InnerClass();
    System.out.println(myInner.y);
  }
```

static ички класстар сырткы класстардын атрибуттарына кайрыла албайт.

Сырткы класстын атрибуттарына кайрылуу үчүн static модификаторун алып салуу керек:

```
class OuterClass {
    int x = 10;
    class InnerClass {
      public int myInnerMethod() {
        return x;
      }
    }
}
```

Бул жерде ички класстын методу сырткы класстын атрибутунун маанисин кайтарып берет.

Бышыктоочу суроолор:

1. Класстар, методдор жана атрибуттар менен иштөөдө кандай модифика-торлорду колдонобуз?
2. Инкапсуляция принциби жөнүндө эмне билесиз?
3. Мурастоо принцибинде кайсы ачкыч сөз колдонулат?
4. Мурастоонун кандай оң жактары бар?
5. Абстракттык класстардын объектиси болобу?

6. Полиморфизм принциби жөнүндө айтып бериңиз?

10- практикалык сабак

Максаты: Инкапсуляция, мурастоо жана полиморфизм принциптери жөнүндө алган маалыматтарды бышыктоо жана маселелерди чыгаруу.

Келгиле, чогуу төмөнкү маселени карап көрөлү.

Маселенин берилиши:

Эки сандын үстүнөн кошуу жана кемитүү амалдарын аткаруучу Calculator классын түзгүлө. Ал класстан мурасталган My_Calculator классын түзгүлө. My_Calculator классына кошумча эки санды көбөйтүү методун кошкула. Түзүлгөн My_Calculator классынын объектисин түзгүлө.

Чыгаруу:

Программаны түзүүдө биринчи Calculator классын түзүп алабыз. Класска эки сандын суммасын, айырмасын сактоо үчүн атайын талааны кошуп алабыз. Кошуу жана кемитүү үчүн методдорду кошобуз.

```
public class My_Calculator extends Calculator {
        public void mult(int a, int b) {
          c = a * b;
          System.out.println("Сандардын көбөйтүндүсү: " + c);
        }

        public static void main(String args[]) {
          int a = 5, b = 10;
          My_Calculator cal = new My_Calculator();
          cal.add(a, b);
          cal.sub(a, b);
          cal.mult(a, b);
        }
```

```
    }
class Calculator {
    int c;

    public void add(int a, int b) {
      c = a + b;
      System.out.println("Сандардын суммасы: " + c);
    }

    public void sub(int a, int b) {
      c = a - b;
      System.out.println("Сандырдын айырмасы: " + c);
    }
  }
```

Программанын жыйынтыгы:

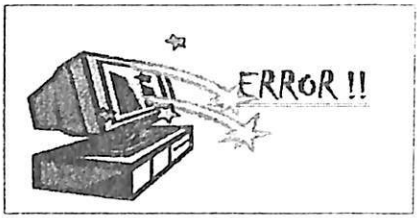
```
Сандардын суммасы: 15
Сандырдын айырмасы: -5
Сандардын көбөйтүндүсү: 50
```

Өз алдынча иштөөгө мисалдар

1. Бул жерде түстөрдүн иерархиясын ишке ашыруу керек (мисалы, ал роза, гвоздика, жоогазын ж.б. сиздин табитиңизге ылайыктуу нерсе болсун). Бир нече түстүү объекттерди түзүңүз. 3 букет чогултуп (массивди колдоносуз) жана алардын баасын аныктаңыз. Букет бир түрдөгү бир нече гүлдү камтышы мүмкүн. "Flower" суперклассында гүлдүн баасын аныктоочу метод болушу керек, бул метод мурасталган класстарында жокко чыгарылууга тийиш. Ошондой эле сатылган гүлдөрдүн санын эсептеңиз (биз статикалык методду колдонобуз).

2. Жаныбарлар классын жана аны кеңейтүүчү Dog, Cat, Horse класстарын түзүңүз. Animal классында food, location өзгөрмөлөрү жана Noise(үндөр), eat(тамактануу), sleep(уктоо) методдору бар. Dog, Cat, Horse класстары makeNoise, eat методдорун кайрадан иштеп чыгат. Dog, Cat, Horse класстарына ушул жаныбарларды гана мүнөздөгөн өзгөрмөлөрдү кошуңуз. Ветеринар классын түзүңүз, анда сиз void treatAnimal(Animal animal) методун аныктаңыз. Бул метод менен кабыл алууга келген жаныбардын тамак-ашы жана жайгашкан жери басып чыгарылсын. Негизги main методдо Animal типтеги массивди түзүңүз, ага бардык түрдөгү жаныбарларды жазыңыз. Цикл учурунда аларды ветеринарга алып барыңыз.
3. Shape супер классын жана ал класстан мурасталуучу Circle, Rectangle класстарын түзүңүз. Shape классы абстракттуу draw() методун жана түс өзгөрмөсүн камтыйт. Circle жана Rectangle класстары чекиттердин координаталарын камтыйт. Бул фигураларды камтыган массив түзүңүз. Аларды циклге тартыңыз (draw методун чакырыңыз).
4. Класстардын иерархиясын көрсөтүүчү программаны жазыңыз: геометриялык фигуралар (ромб, тик бурчтук, эллипс). Фигуранын аянтын жана периметрин эсептөөчү методдорду түзүңүз. Колдонуучуга көрсөтүү үчүн фигуранын түрүн тандоону камсыз кылуу менен класстын бардык ыкмаларынын иштешин көрсөтүңүз.

Лекция 12. Каталардын үстүнөн иштөө же өзгөчө кырдаалдарды алдын алуу



Сабактын максаты:

1. *try...catch...finally блоктору менен таанышуу;*
2. *throw жана throws ачкыч сөздөрүн колдонууну үйрөнүү.*

Бул теманы үйрөнүүдө сиз "компилятор катасы бар программаны өткөрбөйт, биз кандай каталардын үстүнөн иштейбиз?" деп ойлонушуңуз мүмкүн. Ооба каталар программа иштегенге чейинки жана иштеп жаткан учурда пайда болуучу каталар болуп бөлүнөт. Биз программа иштеп жаткан учурда пайда болуучу ар кандай каталарды алдын алуу менен программанын иштөөсүн үзгүлтүккө учуратпайбыз. Төмөнкү мисалды карап көрөлү:

```
public class Main {
        public static void main(String[ ] args) {
          int[] myNumbers = {1, 2, 3};
          System.out.println(myNumbers[10]); // ката!
        }
      }
```

Программанын жыйынтыгы:

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Index 10 out of bounds for length 3
        at Programs/progs.Example42.main(Example42.java:5)
```

Жогоруда көрсөтүлгөн программада ката массивдин индекси чектен ашып кетти деп турат. Мындай каталарды алдын алуу үчүн try...catch...finally блоктору менен таанышабыз.

try...catch...finally блоктору

Java тилинде каталарды алдын алуу атайын моделдин негизинде ишке ашат. Ката пайда болгондо, өзгөчө кырдаал жаралат жана бул программалык коддун атайын блогунда кармалып турат. Каталардын үстүнөн иштөө амалдарын Java тилинин Error жана Exception класстары ишке ашырат.

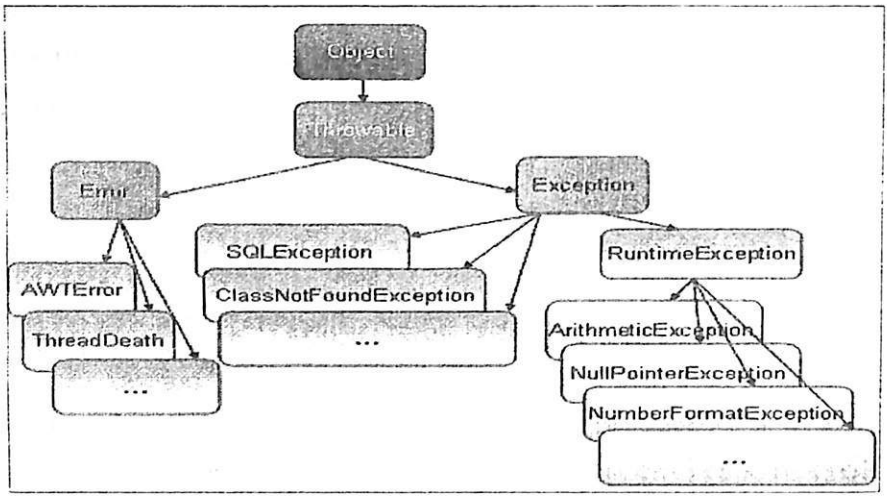
Качан өзгөчө кырдаал жаралганда, анын экземпляры түзүлөт жана ал каталардын үстүнөн иштөөчү методго жөнөтүлөт.
Error классындагы каталар ички болуп эсептелет. Мисалы флешка уланбай эле андан маалыматты окуй турган болсок, өзгөчө кырдаал жаралат.

Java тилинде каталардын же өзгөчө кырдаалдардын үстүнөн иштөөдө атайын төмөнкү ачкыч сөздөр колдонулат: try, catch, throw, throws жана finally.

Ката жаралат деп эсептеген программанын бөлүгү try блогуна орнотулат. catch ачкыч сөзүн колдонуу менен программист пайда болгон катаны тутуп, аны акыл калчап чече алат. Эгерде өзгөчө кырдаалды өзүбүз пайда кылгыбыз келсе throw ачкыч сөзүн пайдаланабыз.

Методдо өзгөчө кырдаал каралып жатканын көрсөтүү үчүн throws варианты колдонулат.

Каталарды иштеп чыгуучу методдон кайтканга чейин сөзсүз аткарылуучу блок finnaly блогу болуп эсептелет. Өзгөчө кырдаалдарды башкаруучу класстардын иерархиялык схемасы төмөнкү сүрөттө көрсөтүлдү (25-сүрөт).



25-сүрөт. Каталарды башкаруучу класстар

try…catch блокторунун иштөөсүн мисалда карап көрөлү:

```
package progs;
public class Example42 {
        public static void main(String[ ] args) {
                try {
                  int[] myNumbers = {1, 2, 3};
                  System.out.println(myNumbers[10]);
                } catch (Exception e) {
                  System.out.println("Кандайдыр бир ката бар");
                }
              }
            }
```

Программанын жыйынтыгы:

```
Кандайдыр бир ката бар
```

Жогоруда көрсөтүлгөн мисалда 3 элементтен турган бир өлчөмдүү сандык массив берилген жана ар бир элемент тиешелүү маанилерди алды. Кийинки сапта массивдин 10-элементин экранга чыгаруу командасы турат. Бирок бизде болгону 3 элементтен турган массив түзүлгөн, ошондуктан бул учурда өзгөчө кырдаал жаралат. Бул бөлүк try блогунда болгондуктан, программа башкарууну catch блогуна өткөрүп берди. Ал жерде экранга "Кандайдыр бир ката бар" деген маалыматты чыгаруу командасы турат. Ошондуктан программанын жыйынтыгында ушул маалыматты алдык.

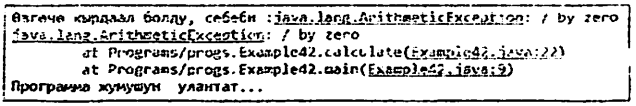
Кийинки мисалды карап көрөлү:

```
package progs;

public class Example42
{
	public static void main(String args[])
	{
		try
		{
			int num = calculate(9,0); // колдонуучунун методу
			System.out.println(num);
		}
		catch(Exception e)	// e - өзгөчө кырдаал объектиси
		{
			System.err.println("Өзгөчө кырдаал болду, себеби :" +
e.toString());
			e.printStackTrace();
		}
		System.out.println("Программа жумушун улантат...");
	}

	static int calculate( int a, int b)
	{
		int num = a / b;
		return num;
	}
}
```

Программанын жыйынтыгы:



Жогоруда келтирилген мисалда колдонуучу тарабынан түзүлгөн calculate методун көрүп турабыз. Ал метод static болгондуктан эч кандай объектсиз негизги main методунан чакырылып жатат. calculate методу эки аргументти алат жана биринчи аргументти экинчи аргументтке бөлүп тийиндини кайтарып берүү жумушун аткарат.

Негизги **main** методунан calculate методун 0 жана 9 аргументтери менен чакырууда, 0 санын 9 санына бөлүү операциясы аткарылып, 0ду санга бөлүүгө болбогондуктан өзгөчө кырдаал жаралат. Эгерде ушул программанын бөлүгү try блогуна кирбегенде программа жумушун токтотуп, башкаруу операциондук системага берилмек. Биздин учурда try блогу катаны тутуп, башкарууну catch блогуна жөнөтөт.

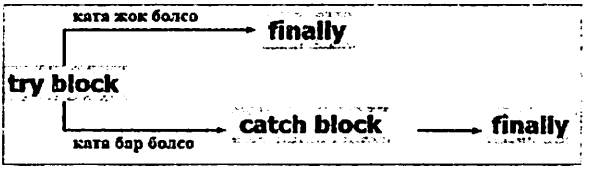
catch блогунда каталардын тибин аныктоочу e объектиси түзүлөт жана toString методунун жардамында анда сакталган маалымат текстке айланып экранга чыгат.

Программада ката пайда болду. Биз ката жөнүндө маалыматты алдык жана ошол эле учурда программа андан ары жумушун улантып жатканын көрдүк. Аны экрандагы "Программа жумушун улантат..." маалыматынан көрүп турабыз.

Программада биз үчүн жаңы болгон 2 методду көрүп турабыз: toString() - Exception объектинде сакталган маалыматтын саптык(String) көрүнүшүн кайтарып берет.

printStackTrace() - өзгөчө кырдаалдын себебин, ошондой эле катаны пайда кылган коддун сабын экранга чыгаруу үчүн колдонулат.

Каталардын үстүнөн иштөөдө дагы бир finally блогу аткарылат жана ал try менен бирге колдонулат. finally блогу өзгөчө кырдаал жаралса да, жаралбаса да аткарылат(26-сүрөт). Кээ бир учурларда аны кошпой деле коюшат. Бирок бул блок өзгөчө кырдаал болгондо, бардык эстутумду тазалоо иштери аткарылаарын камсыздайт.



26-сүрөт. try...catch...finally блокторунун иштөөсү

Мисалы:

```
package progs;
public class Example43 {
    public static void main(String[] args) {
      try {
       int[] myNumbers = {1, 2, 3};
       System.out.println(myNumbers[10]);
      } catch (Exception e) {
       System.out.println("Кандайдыр бир ката бар");
      } finally {
       System.out.println("'try catch' аяктады");
      }
    }
}
```

Программанын жыйынтыгы:

```
Кандайдыр бир ката бар
'try catch' аяктады
```

Жогоруда келтирилген мисалдан try ...finally блокторунун иштөөсүн көрсөк болот.

try блогунда бир эле убакта бир канча каталар пайда болуусу мүмкүн. Бул учурда бир канча catch блокторун колдонсок болот. Алардын аткарылуу тартиби жайгашуусуна жараша ишке ашат(27-сүрөт). Бул сүрөттө көрсөтүлгөн ArrayIndexOutOfBoundsException классы Exception классынын туунду классы болсо да биринчи болуп аткарылат.

```
.....
try{
}
catch(ArrayIndexOutOfBoundsException e) {
}
catch(Exception e) {
}
.....
```

27-сүрөт. Бир нече catch блоктору менен иштөө

throw жана throws ачкыч сөздөрү

Java тилинде throw ачкыч сөзү методдон же коддун каалаган блогунан каталарды аныктоо үчүн колдонулат. Негизинен throw ачкыч сөзү колдонуучу тарабынан түзүлүүчү өзгөчө кырдаалдар болуп эсептелет. Мисалы:

```
package progs;
import java.util.Scanner;

class Example43 {
        public static void main(String[] args) {
                Scanner in = new Scanner(System.in);
                while(true){
                        int a=in.nextInt();
                        int b=in.nextInt();
```

```
try{
        if (b==0) throw new ArithmeticException();
        double div=a*1.0/b;
        System.out.println("a/b="+div);
} catch(ArithmeticException ex){
        System.out.println("Санды 0гө бөлүүгө болбойт. Башка сан кийириңиз");
}
                }
        }
}
```

Программанын жыйынтыгы:

```
a/b=0.75
Санды 0гө бөлүүгө болбойт. Башка сан кийириңиз
```

Жогорудагы мисалда чексиз циклдын ичинде 2 бүтүн сан кийирилип жатат. Андан кийин биринчи санды экинчи санга бөлүүдө ката болуп калат деп, try блогуна жазып алдык жана жыйынтыгын экранга чыгардык.

Бул жерде if (b==0) throw new ArithmeticException(); сабын жазбасак да катаны try кийинки **double** div=a*1.0/b; сапта тутат жана башкарууну catch блогуна жөнөтөт. Биз throw ачкыч сөзү менен катаны алдын ала айтып, кандай ката пайда болорун өзүбүз жазып койдук жана b=0 болуп калса, компилятор кийинки бөлүү амалын аткарбай эле башкарууну catch блогуна өткөрүп берет. Ошентип биз катаны алдын алабыз.

throws ачкыч сөзү менен биз кайсы бир методду түзүп жатып кандай ката жараларын көрсөтүп койсок болот. Мисалы:

```
public static void main(String[] args) throws IOException {
  try {
```

```
    BufferedReader reader = new BufferedReader(new
FileReader("C:\\Users\\Username\\Desktop\\test.txt"));

    String firstString = reader.readLine();
    System.out.println(firstString);
  } catch (FileNotFoundException e) {

    System.out.println("Ката! Файл табылган жок!");
  }
}
```

Жогорудагы мисалда файлдар менен иштеп жатканда, эгерде файл эстутумда жок болуп калса, анда ката жаралат. Ошондуктан аны алдын алуу үчүн биз негизги методдун башында throws ачкыч сөзү менен ката болуп каларын көрсөтүп койдук. Java тилинде каталар менен иштөөчү 400 класс аныкталган. Албетте көп, бирок алардын ар бири программисттер кандай ката экендигин түшүнүүсү үчүн түзүлгөн.

Бышыктоочу суроолор:

1. Катанын кандай түрлөрү бар?
2. try, catch, finally блоктору эмне кызмат кылат?
3. Кайсы убакта башкаруу catch блогуна жөнөтүлөт?
4. throw жана throws ачкыч сөздөрү эмне үчүн колдонулат?

11- практикалык сабак

Максаты: Каталардын үстүнөн иштөөчү методдор жөнүндө алган маалыматтарды бышыктоо жана маселелерди чыгаруу.

Келгиле, чогуу төмөнкү маселени карап көрөлү.

Маселе:

Кокус бүтүн сандардан массивди түзө турган программа жазыңыз. Колдонуучу жолчолордун жана мамычалардын санын, ошондой эле бүтүн сандардын диапазонун көрсөтө алат. Колдонуучунун киргизүү каталарын текшериңиз.

Кийирилүүчү маалымат

n - жолчолордун саны

m - мамычалардын саны

k - бүтүн сандардын диапозону

Чыгуучу маалымат

Эгерде ката жок болсо, түзүлгөн массив экранга чыгат, болбосо катанын себеби аныкталат

Чыгаруу:

```
package progs;
import java.util.Scanner;
import java.io.IOException;
import java.util.Random;
public class Massiv100 {

    public static void main(String[] args) {
        Scanner obj=new Scanner(System.in);
        Random random = new Random();

        try {
        int n=obj.nextInt();
        int m=obj.nextInt();
        int k=obj.nextInt();
        int[][] array = new int[n][m];
        for (int i = 0; i < n; i++) {
```

```
                    for (int j = 0; j < m; j++) {
                array[i][j] = random.nextInt(k) + 1;
                    }
                }
                for (int i = 0; i < n; i++) {
                    for (int j = 0; j < m; j++) {
        System.out.print(array[i][j]+" ");
                    }
                System.out.println();
                }
            }
            catch(Exception e) {
                System.out.println("Ката пайда болду");
                e.printStackTrace();
            }
        }
}
```

Программанын жыйынтыгы:

```
3
3
100
75 31 44
16 27 71
15 11 80
```

Өз алдынча иштөөгө мисалдар

1. Логинди жана паролду текшерүү

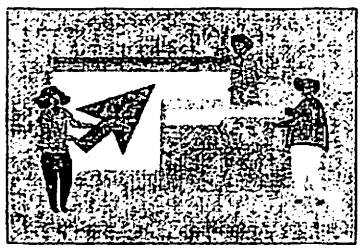
- login, password жана confirmPassword параметрлерин кабыл алуучу статикалык метод түзгүлө

- login бир гана латин тамгаларынан, цифра жана астын сызуу белгилеринен турат.

- login узундугу 20 символдон кем болуш керек. Эгерде login талаптарга жооп бербесе WrongLoginException катасы пайда болсун.
- Ошондой эле сырсөз (password) менен confirmPassword бирдей болушу керек. Эгер сырсөз бул талаптарга жооп бербесе, WrongPasswordException катаны тутуучу классы иштей баштайт.
-WrongPasswordException жана WrongLoginException - эки конструктору бар колдонуучу аныктаган класстар - бири демейки боюнча, экинчиси өзгөчө билдирүүнү кабыл алат жана аны Exception классынын конструкторуна өткөрүп берет.
- каталарды тутуу методдун ичинде ишке ашат.
- эгерде пароль туура болсо true маанисин, болбосо false маанисин кайтарат.

2. Сиз онлайн дүкөн үчүн мини тиркемени түзүңүз. Төмөнкү мүмкүнчүлүктөрдү ишке ашыруу керек: а) колдонуучунун аутентификациясы. Сизде логин-пароль жуптарын камтыган дисплей болсун. Колдонуучу туура маалыматтарды киргизсе, ал аутентификацияланат. Бул учурда логин жана паролду камтыган User объектин түзөлү. Эгерде маалыматтар туура эмес болсо, анда бош маанилер менен Колдонуучу объекти түзүлсүн. б) продукт каталогдорунун тизмесин көрүү, в) берилген критерий боюнча сорттолгон белгилүү бир каталогдон товарлардын тизмесин көрүү, г) арабага кошуу үчүн товарды тандоо, д) арабада товарларды сатып алуу. Сиз өз операцияларыңызды кошуп же табитиңизге жараша бир нерсени өзгөртө аласыз.

Лекция 13. Java тилинде файлдар менен иштөө



Сабактын максаты:

1. *Кийируу/чыгаруу агымдарын үйрөнүү*
2. *Файлдарды окуу жана жазуу. FileInputStream жана FileOutputStream класстары*
3. *BufferedInputStream жана BufferedOutputStream класстары*
4. *PrintStream жана PrintWriter класстары*
5. *File классы. Файлдар жана папкалар менен иштөө*

Кийирүү/чыгаруу агымдары

Көптөгөн программалоо тилдеринин айырмалоочу өзгөчөлүгү алардын файлдар жана агымдар менен иштөөсү. Java тилинде файлдар менен иштөө үчүн негизги функциялар **java.io** пакетинин класстарында топтолгон.

Агымдар Javanын кийирүү/чыгаруу системасы аркылуу чыныгы физикалык түзүлүш менен байланышкан. Бизде файл менен байланышкан жана файлды окуп же жаза турган агымды аныктап алсак болот. Ал ошондой эле маалыматтарды алуу же жөнөтүү үчүн колдонула турган тармактык түйүнгө байланышкан агым болушу мүмкүн. Ар кандай файлдарды окуу жана жазуу, тармак аркылуу маалымат алмашуу, консолдон маалыматтарды кийирүү маселелери Java тилинде агымдар аркылуу чечилет.

Маалыматтарды окууга мүмкүн болгон объект **кирүү агымы** деп аталат, ал эми маалыматты жазууга мүмкүн болгон объект **чыгаруу агымы** деп аталат.

Байт агымдарын башкарган бардык класстардын өзөгүн эки абстрактуу класс түзөт: ***InputStream*** (киргизүүчү агымдарды көрсөтүүчү) жана ***OutputStream*** (чыгаруу агымдарын көрсөтүүчү). Бирок байт менен иштөө анча ыңгайлуу болбогондуктан, символдор агымы менен иштөө үчүн абстрактуу **Reader** классы (символ агымын окуу үчүн) жана **Writer** (символ агымын жазуу үчүн) кошулган.

Калган класстар ушул класстардын туунду класстары болуп эсептелет жана агымдар менен иштөөдө колдонулат(28-сүрөт):

InputStream	OutputStream	Reader	Writer
FileInputStream	FileOutputStream	FileReader	FileWriter
BufferedInputStream	BufferedOutputStream	BufferedReader	BufferedWriter
ByteArrayInputStream	ByteArrayOutputStream	CharArrayReader	CharArrayWriter
FilterInputStream	FilterOutputStream	FilterReader	FilterWriter
DataInputStream	DataOutputStream		
ObjectInputStream	ObjectOutputStream		

28-сүрөт. Файлдар менен иштөөчү класстар

Файлдарды окуу жана жазуу. FileInputStream жана FileOutputStream

Файлдарды жазуу жана FileOutputStream классы

FileOutputStream классы файлга байттарды жазуу үчүн иштелип чыккан. Ал OutputStream классынан келип чыккандыктан, анын бардык функцияларын мурастап алган.

FileOutputStream классынын конструктору жазыла турган файлды аныктайт жана бир нече конструктордун түрлөрү бар:

1. FileOutputStream(String filePath)
2. FileOutputStream(File fileObj)
3. FileOutputStream(String filePath, boolean append)
4. FileOutputStream(File fileObj, boolean append)

Файл String тибиндеги файлга жетүүчү жолду көрсөтүү аркылуу же File объекти аркылуу көрсөтүлөт. Экинчи параметр - append- жазуу ыкмасын көрсөтөт: эгерде ал **true** болсо, анда маалыматтар файлдын аягына жазылат, ал эми **false** болсо, файл толугу менен кайра жазылат.

Мисалы, файлга бир сапты жазып көрөлү:

```
package progs;

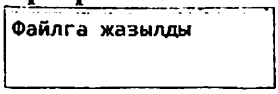
import java.io.*;
public class File1 {
        public static void main(String[] args) {

      String text = "Hello Kyrgyzstan!"; //файлга жазылуучу сап
      try(FileOutputStream fos=new FileOutputStream("text.txt"))
      {
         // сапты байтка айлантуу
         byte[] buffer = text.getBytes();

         fos.write(buffer, 0, buffer.length);
         System.out.println("Файлга жазылды");
      }
      catch(IOException ex){

         System.out.println(ex.getMessage());
      }
   }
}
```

Программанын жыйынтыгы:



Жогоруда көрсөтүлгөн мисалда **FileOutputStream** классынын объектисин түзүүдө конструкторго файлга жетүүчү жол көрсөтүлдү. Эгерде файл жок болсо, анда ал автоматтык түрдө түзүлөт. Сапты файлга жазууда ал биринчи байтка айландырылат, жана андан кийин гана **write** методунун жардамында файлга жазылат.

Файлдын үстүнөн иштөөдө, аны автоматтык түрдө жабуу үчүн try...catch конструкциясы колдонулат.

*Файлдарды окуу жана **FileInputStream** классы*

Файлдан маалыматтарды окуу үчүн **FileInputStream** классы колдонулат, ал **InputStream** классынын туундусу болуп саналат жана ошондуктан анын бардык методдорун иштете алат.

FileInputStream классынын объектин түзүү үчүн биз бир катар конструкторлорду колдоно алабыз. Конструктордун эң көп колдонулган версиясы параметр катары окула турган файлдын жолун алат:

FileInputStream(String fileName) throws FileNotFoundException

Бул жерде көрсөтүлгөндөй, эгерде файл жок болсо, **FileNotFoundException** өзгөчө абалы(ката) пайда болот.

Мисал катары башта жазылып сакталган файлды карап көрөлү:

```
package progs;
import java.io.*;
public class File2 {
        public static void main(String[] args) {
```

```
    try(FileInputStream fin=new FileInputStream("text.txt"))
    {
      int i;
      while((i=fin.read())!=-1){

        System.out.print((char)i);
       }
     }
     catch(IOException ex){

       System.out.println(ex.getMessage());
      }
   }
}
```

Программанын жыйынтыгы:

```
Hello Kyrgyzstan!
```

Жогоруда каралган мисалда ар бир байт окулап, char тибиндеги символго айланат да консолго чыгарылат.

Буферленген агымдар. BufferedInputStream жана BufferedOutputStream

BufferedInputStream классы

Кийирүү/чыгаруу операцияларын оптималдаштыруу үчүн буфердик агымдар колдонулат. Алар стандартуу эстутумга буферлерди кошуу менен маалымат алмашууну тездетет.

BufferedInputStream классы киргизүү түзүлүшүнө тынымсыз кайрылбастан, киргизилген маалыматтарды атайын буферге топтойт. **BufferedInputStream** классынын объектин түзүү үчүн 2 конструктор бар:

1. BufferedInputStream (InputStream inputStream)

2. BufferedInputStream (InputStream inputStream, int bufSize)

Биринчи параметр - буферге маалыматтар окула турган кийирүү агымы. Экинчи параметр - буфердин өлчөмү болуп саналат.

Мисалы, биз **ByteArrayInputStream** агымынан маалыматтарды буферге окуйбуз:

```
import java.io.*;
public class File3 {
      public static void main(String[] args) {

            String text = "Hello world!";
          byte[] buffer = text.getBytes();
        ByteArrayInputStream in = new ByteArrayInputStream(buffer);

        try(BufferedInputStream bis = new BufferedInputStream(in)){

            int c;
            while((c=bis.read())!=-1){

               System.out.print((char)c);
            }
          }
          catch(Exception e){

            System.out.println(e.getMessage());
          }
          System.out.println();
       }
    }
```

BufferedInputStream классынын read методун пайдалануу менен buffer массивинен маалыматтар окулуп, (**char**)c командасы менен символго айланып, экранга чыгарылат.

BufferedOutputStream классы

BufferedOutputStream классы чыгаруу агымдары үчүн буферди түзүп берет. Бул буфер түзмөккө тынымсыз кирбестен чыгуу байттарын топтойт. Ал эми буфер толгондо, маалыматтар жазылат.

Бул класстын да эки конструктору бар:

1. BufferedOutputStream (OutputStream outputStream)
2. BufferedOutputStream (OutputStream outputStream, int bufSize)

Биринчи параметр **OutputStreamден** мураска алынган чыгаруу агымы, ал эми экинчи параметр буфердин өлчөмү.

Файлга жазууну мисалда карап көрөлү:

```
package progs;
import java.io.*;
public class File4 {
    public static void main(String[] args) {

  String text = "Hello Kyrgyzstan!"; // жазылуучу сап
  try(FileOutputStream out=new FileOutputStream("text.txt");
    BufferedOutputStream bos = new BufferedOutputStream(out))
  {
    // сапты байтка айлантуу
    byte[] buffer = text.getBytes();
    bos.write(buffer, 0, buffer.length);
  }
  catch(IOException ex){

    System.out.println(ex.getMessage());
  }
```

Бул мисалда буферге топтолгон маалыматты байтка айлантып, андан кийин write методун колдонуу менен файлга жазуу амалы ишке ашты.

PrintStream жана PrintWriter класстары

PrintStream классы консолго чыгаруу үчүн колдонулуучу класс болуп саналат. System.out.println() командасы аркылуу консолго кээ бир маалыматты басып чыгарганда, биз PrintStreamди колдонобуз, анткени System классындагы out өзгөрмөсү PrintStream классынын объектисин билдирет, ал эми println() методу PrintStream классынын методу болуп саналат.

Класстын бир канча конструкторлору бар.

Мисалда карап көрөлү:

```
package progs;
import java.io.*;
public class File5 {
        public static void main(String[] args) {

                String text = "Салам дүйно!"; // жазылуучу сап
                try(FileOutputStream fos=new FileOutputStream("C://java
files//programs//text.txt");
            PrintStream printStream = new PrintStream(fos))
      {
        printStream.println(text);
        System.out.println("Файлга жазылды");
      }
      catch(IOException ex){
        System.out.println(ex.getMessage());
      }
    }
```

}

Программанын жыйынтыгы:

```
Файлга жазылды
```

Маалыматты чыгуу агымына чыгаруу үчүн PrintStream төмөнкү методдорду колдонот:

- println(): саптык маалыматты кийинки жолчого өтүү менен чыгаруу
- print(): саптык маалыматты кийинки жолчого өтпөй чыгаруу
- printf(): формatталган чыгаруу

PrintWriter классы

PrintWriter классы PrintStreamге окшош. Ал маалыматты консолго, файлга же башка чыгаруу агымына чыгаруу үчүн колдонулушу мүмкүн. Бул класста бир нече конструктор бар:

- **PrintWriter(File file)**: көрсөтүлгөн файлга маалыматты автоматтык түрдө кошот.
- **PrintWriter(File file, String csn)**: csn коддоосун эске алуу менен көрсөтүлгөн файлга маалыматты автоматтык түрдө кошот.
- **PrintWriter(OutputStream out)**: маалыматты чыгаруу үчүн, учурдагы OutputStream объекти колдонулат, ага маалыматтарды автоматтык түрдө түшүрөт.
- **PrintWriter(OutputStream out, boolean autoFlush)**: учурдагы OutputStream объекти маалыматты чыгаруу үчүн колдонулат, экинчи параметр маалыматтар OutputStreamге автоматтык түрдө кошулушу керекпи же жокпу көрсөтөт.

- **PrintWriter(String fileName):** көрсөтүлгөн ат менен файлга маалыматты автоматтык түрдө кошот.
- **PrintWriter(String fileName, String csn):** csn коддоосун колдонуп көрсөтүлгөн ат менен файлга маалыматты автоматтык түрдө кошот.
- **PrintWriter(Writer out):** маалыматты чыгаруу үчүн маалымат автоматтык түрдө жазылып жаткан Writer объектиси колдонулат.
- **PrintWriter(Writer out, boolean autoFlush):** маалыматты чыгаруу үчүн Writer объектиси колдонулат, экинчи параметр Writer'ге маалыматты автоматтык түрдө кошууга болобу экенин көрсөтөт.

Ошондой эле агымга маалыматтарды жазуу үчүн printf() жана println() методдору колдонот.

Мисалы консолго чыгаруу үчүн колдонуп көрөлү:

```
try(PrintWriter pw = new PrintWriter(System.out))
{
  pw.println("Hello world!");
}
```

Бул жерде чыгаруу агымы катары System.out колдонулат жана консолго "Hello world!" саптары басылып чыгат.

File классы. Файлдар жана папкалар менен иштөө

File классын колдонуу үчүн класстын объектин түзүп, файлдын атын же каталогдун атын көрсөтүңүз:

```
import java.io.File;  //File классын импорттоп алуу
```

File myObj = new File("filename.txt");// Файлдын атын көрсөтүңүз

File классында файлдарды түзүү жана алар жөнүндө маалымат алуу үчүн көптөгөн пайдалуу методдор бар. Мисалы:

Метод	Тиби	Мааниси
canRead()	Boolean	Файлдын окулушун же окулбагандыгын текшерет
canWrite()	Boolean	Файлга жазууга мүмкүнбү же жокпу, текшерет
createNewFile()	Boolean	Бош файлды түзөт
delete()	Boolean	Файлды өчүрөт
exists()	Boolean	Файлдын бар же жок экенин текшерет
getName()	String	Файлдын атын кайтарат
getAbsolutePath()	String	Файлдын абсолюттук адресин кайтарат
length()	Long	Файлдын өлчөмүн байт менен кайтарат
list()	String[]	Каталогдогу файлдардын массивин кайтарат
mkdir()	Boolean	Папка түзөт

Javada файлдарды түзүү жана жазуу

Javada файл түзүү үчүн createNewFile() методун колдонсоңуз болот. Бул метод логикалык маанини кайтарат: эгер файл ийгиликтүү түзүлгөн болсо true, ал эми false – эгерде файл мурунтан эле бар болсо. Метод try...catch блогунда камтылат, анткени ал ката

пайда болгондо IOException түрдөгү каталарды кармайт (эгер кандайдыр бир себептерден улам файл түзүлбөсө):

```
import java.io.File;  // File классын импорттоо
import java.io.IOException;  // Каталарды тутуу үчүн IOException
                                    //классын импорттоо
public class CreateFile {
 public static void main(String[] args) {
  try {
   File myObj = new File("filename.txt");
   if (myObj.createNewFile()) {
    System.out.println("Файл тузулду: " + myObj.getName());
   } else {
    System.out.println("Файл мурунтан эле бар.");
   }
  } catch (IOException e) {
   System.out.println("Ката кетти");
   e.printStackTrace();
  }
 }
}
```

Файлга жазуу

Төмөнкү мисалда биз **FileWriter** классын анын **write()** методу менен бирге жогорудагы мисалда түзгөн файлга текст жазуу үчүн колдонобуз. Файлга жазып бүткөндөн кийин, аны **close()** ыкмасы менен жабуу керек экенин эске алыңыз:

```
package progs;
import java.io.FileWriter;   // FileWriter классын импорттоо
import java.io.IOException; // Каталарды тутуу учун IOException
                                    // классын импорттоо
public class WriteToFile {
```

```
public static void main(String[] args) {
  try {
    FileWriter myWriter = new FileWriter("file1.txt");
    myWriter.write("Hello Kyrgyzstan!");
    myWriter.close();
    System.out.println("Файлга жазылды");
  } catch (IOException e) {
    System.out.println("Ката пайда болду");
    e.printStackTrace();
  }
 }
}
```

Программанын жыйынтыгы:

```
Файлга жазылды
```

Файлды окуу

Төмөнкү мисалда биз мурунку бөлүмдө түзгөн тексттик файлдын мазмунун окуу үчүн Scanner классын колдонобуз:

```
package progs;

import java.io.File;  // File классын импорттоо
import java.io.FileNotFoundException;  // Каталарды тутуу үчүн
import java.util.Scanner; // Тексттик файлдарды окуу үчүн Scanner
                          //классын импорттоо
public class ReadFile {
  public static void main(String[] args) {
    try {
      File myObj = new File("file1.txt");
      Scanner myReader = new Scanner(myObj);
      while (myReader.hasNextLine()) {
        String data = myReader.nextLine();
```

```
      System.out.println(data);
    }
    myReader.close();
   } catch (FileNotFoundException e) {
    System.out.println("Ката пайда болду");
    e.printStackTrace();
   }
 }
}
```

Программанын жыйынтыгы:

```
Hello Kyrgyzstan!
```

Файл жөнүндө маалымат алуу

Файл жөнүндө көбүрөөк маалымат алуу үчүн, File классынын методдорун колдонуңуз:

```
package progs;
import java.io.File;  // File классын импорттоо
public class ReadFile {
 public static void main(String[] args) {
  File myObj = new File("file1.txt");
  if (myObj.exists()) {
   System.out.println("Файлдын аты: " + myObj.getName());
   System.out.println("Файлдын абсолюттук адреси: " + myObj.getAbsolutePath());
   System.out.println("Жазууга жарамдуулугу: " + myObj.canWrite());
   System.out.println("Окууга жарамдуулугу " + myObj.canRead());
   System.out.println("Файлдын  олчому байт менен -" + myObj.length()); } else
   System.out.println("Файл табылган жок");
  }
 }
```

N бүтүн саны берилсин. n! табыңыз жана анын жыйынтыгын текстгик файлга жазыңыз.

Кийирилүүчү маалымат

N саны кийирилет.

Чыгуучу маалымат

Текстгик файлга жооп жазылат

Чыгаруу:

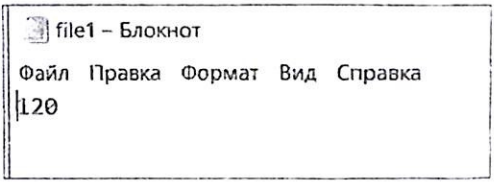
```
package progs;

import java.io.FileWriter;   // FileWriter классын импорттоо
import java.io.IOException;
import java.util.Scanner;

public class File2 {
        public static void main(String[] args) {
          try {
              Scanner obj=new Scanner(System.in);
              System.out.print("N=");
              int n=obj.nextInt();
              int S=1;
              for(int i=1;i<=n;i++)
              {S=S*i;}
           FileWriter myWriter = new FileWriter("file1.txt");
           myWriter.write(Integer.toString(S));
           myWriter.close();
           System.out.println("Файлга жазылды");
          } catch (IOException e) {
           System.out.println("Ката пайда болду");
           e.printStackTrace();
          }
```

```
    }
}
```

Программанын жыйынтыгында file2.txt тексттик файлын ачып карадык жана төмөнкү көрдүк:



Өз алдынча иштөөгө маселелер

1. f тексттик файлы берилсин. Файлдагы тексттеги үндүү тамгалардын санын таап g файлына жазгыла.

2.10 бүтүн сандарды камтыган f файлы берилсин. Сандардын эң чоңун жана эң кичинесин таап экранга чыгаргыла. (бир өлчөмдүү массивди пайдалангыла).

3.Тексттик маалыматтарды камтыган f жана g файлдары берилсин. f файлындагы маалыматты g файлындагы маалыматтын аягына улагыла.

4.Текст берилсин. Андагы сөздөрдү f файлынын ар бир жолчосуна бөлүп өзүнчө кылып жазгыла.

5. 10 орундуу сандын цифраларынын суммасын таап, f файлына жазгыла.

6. Текст берилсин. Андагы эң көп колдонулган сөздү f файлына жазгыла

Лекция 14. Коллекциялар



Сабактын максаты:

Төмөнкү түшүнүктөр менен таанышуу жана үйрөнүү:

1. *Коллекциянын түрлөрү. Коллекция интерфейси*
2. *List (Тизме) ArrayList классы жана List интерфейси*
3. *Queue (Кезектер) жана ArrayDeque классы*
4. *Set (Көптүктөр) интерфейси жана HashSet классы*
5. *Map интерфейси жана HashMap классы*

Коллекциянын түрлөрү. Коллекция интерфейси

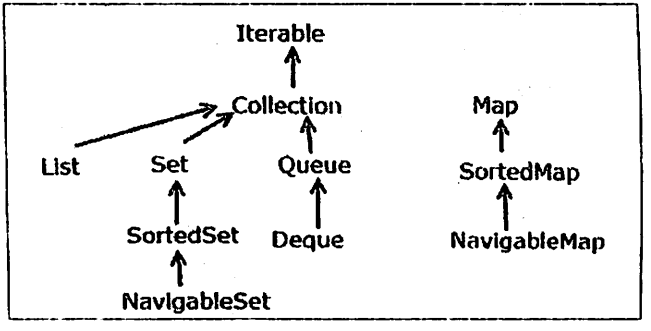
Java тилиндеги коллекциялар бири-бирине окшош бир нече маалымат структураларынын жалпы аталышы. Бул таблица же тизме сыяктуу көп сандагы маанилерди сактай турган татаал типтер.

Структуралардын жөнөкөй мисалы массивдер: алар тизме же жолчо сыяктуу бир катар маанилерди сакташат. Бирок массивдерден айырмаланып, коллекциялар динамикалык, башкача айтканда, алардын туруктуу узундугу жок. Алардын өлчөмү өзгөрүшү мүмкүн, элементтерди метод-функцияларды колдонуу менен кошууга жана өчүрүүгө болот.

Мисалы, сандардын тизмесин алалы: [1, 1, 2, 3, 5]. Бул тизме 5 элементти сактаган маалымат структурасы. Эгерде сиз ага алтынчы элементти кошо турган болсоңуз, айталы 8, бул динамикалык маалымат структурасы болуп, коллекцияларга да тиешелүү болушу мүмкүн. Сандардын ордунда жолчолор, объекттер жана башка маалыматтар түрлөрү болушу мүмкүн. Негизги нерсе элементтер бир тектүү жана бир типте болушу керек. Коллекциялардын үч негизги түрү бар: **List, Set, Queue. Map** да бар, бирок ал коллекция эмес, алар менен байланышкан маалымат структурасы.

Ар бир Java программисти коллекцияларды колдонот. Динамикалык маалымат структуралары тилдин маанилүү жана зарыл бөлүгү болуп саналат. Алар маалыматты сактоо, иштеп чыгуу жана берүү үчүн колдонулат, каалаган программа маалымат менен иштейт. Жумушка орношууда жаңы Java программисттен коллекциялар жана алар жөнүндө эмне билерин сурашат.

Схемалык түрдө коллекциялардын бүткүл системасын кыскача төмөнкүчө көрсөтүүгө болот(29-сүрөт):



29-сүрөт. Коллекциялар

29-сүрөттө көрүнүп тургандай эки түрдүү негизги интерфейс болот: Collection, Map.

Collection — жеке элементтердин (объекттердин) жыйындысын камтыган кадимки коллекция. Бул коллекцияда элементтер менен иштөөнүн негизги методдору: кошуу (add, addAll), өчүрүү (remove, removeAll, clear), издөө (contains, containsAll), коллекциянын боштугун текшерүү (isEmpty) жана өлчөмү (size).

Map — структурасы "ачкыч-маани"("ключ — значение") жуптарынан турган коллекция. Мындан тышкары, бул коллекцияда ар бир ачкыч уникалдуу болуп саналат: эки ачкыч бирдей мааниге ээ эмес. Бул жыйнак кээде сөздүк(dictionary) деп да аталат.

Map элементтери менен иштөөнүн негизги методдору:

- орнотуу (put, putAll)
- алуу (get, keySet, values, entrySet)
- өчүрүү (remove, clear)
- издөө (containsKey, containsValue)
- коллекциянын бош экенин текшерүү (isEmpty)
- өлчөмү (size)

Ар бир коллекцияны карап чыгалы.

Collection интерфейси

Collection интерфейси Iterable интерфейсин кеңейтет жана бул интерфейсте жалгыз iterator() методу бар.

Итератор – бул кандайдыр бир коллекциянын элементтерине жетүү үчүн колдонула турган атайын объект.

29-сүрөттө көрүнүп тургандай Collection интерфейсинен 3 интерфейс мурасталганын көрө аласыз: **List**, **Queue** жана **Set**. Эми алардын ар бирине кыскача токтоло кетели.

List(тизме) - бул кайталанган маанилерге жол берген иреттелген элементтердин тизмеси. Сиз бул интерфейстин башка аттарын да жолуктурасыз: ырааттуулук, тизме. Тизменин өзгочолүгү - бул элементтер номерленген жана аларга номер (индекс) боюнча кайрылууга болот.

Queue() - англис тилинен кезек деп которулат. Туура айтылышы: Queue-кью. Кезекте элементтер кезекке кошулган тартипте сакталат.

Set - тизмеден айырмаланып, элементтердин кайталанышы жок иретсиз коллекцияны сүрөттөйт. Математикадагы көптүк түшүнүгү менен дал келет.

Map интерфейсинде уникалдуу ачкычтардын байланышын көрө алабыз.

interface Map<K, V>

мында K- ачкычтар, V- сакталуучу маанилердин тиби.

Ачкыч менен биз маанилерди ала алабыз. Маалыматты кошуу үчүн ачкычты жана маанисин кошо кийирүү керек.

Келгиле, кээ бир ишке ашырууларды карап көрөлү:

HashMap - ар кандай түрдөгү ачкычтарды жана маанилерди, анын ичинде null маанисин да сактай алат. Элементтердин тартиби сакталбай калуусу мүмкүн.

LinkedHashMap - маалымат структурасы, маалыматтарды элементтердин байланышкан тизмеси түрүндө сактайт. Элементтер кошулган тартипте жайгаштырылат.

TreeMap – мындай структурада маалыматтар сорттолгон тартипте сакталат. Жаңы кошулган маалыматтар автоматтык түрдө сорттолот. Көп учурда маалыматты тез издөө үчүн колдонулат.

Тизме(List). ArrayList классы жана List интерфейси

Жөнөкөй тизмелерди түзүү үчүн List интерфейси колдонулат, ал Collection интерфейсинин функционалдуулугун кеңейтет.

Негизинен Java'да бул интерфейстин орнотулган **ArrayList** классы бар. **ArrayList** классы өз функцияларын **AbstractList** классынан мурастаган жана **List** интерфейсин колдонгон жалпы коллекцияны билдирет. Жөнөкөй сөз менен айтканда, **ArrayList** массивге окшош жөнөкөй тизме, андагы элементтердин саны так эмес.

ArrayList төмөнкү конструкторлорго ээ:

- ArrayList(): бош тизмени түзөт
- ArrayList(Collection <? extends E> col): col коллекциясынын бардык элементтери кошулган тизмени түзөт.
- ArrayList (int capacity): баштапкы capacity сыйымдуулугу бар тизмени түзөт.

Сыйымдуулук деп бул жерде маалыматтарды сактоочу массивдин өлчөмү айтылып жатат. Элементтерди кошкондо жаңы массив түзүлөт жана ага эски массивден элементтер көчүрүлүп, эстутум чындыгында кайрадан бөлүштүрүлөт. Эгерде башында ArrayListтин өлчөмүн көрсөтүп койсок, эстутумдун кайра

бөлүштүрүлүшүн азайтууга мүмкүндүк берет, ошону менен өндүрүмдүүлүктү жогорулатат.

ArrayList классын жана анын методдорун колдонуп көрөлү:

```
package progs;
import java.util.ArrayList;
public class Example50{
 public static void main(String[] args) {
    ArrayList<String> people = new ArrayList<String>();
     // тизмеге элементтерди кошобуз
     people.add("Асан");
     people.add("Болот");
     people.add("Марат");
     people.add("Анна");
     people.add(1, "Жанна"); //1 индекс боюнча элемент кошобуз

     System.out.println(people.get(1));//2-объектти алабыз
     people.set(1, "Манас"); //2-объектке жаңы маани жүктойбуз
     System.out.printf("ArrayList те %d элмент бар\n", people.size());
     for(String person : people){

       System.out.println(person);
     }
     //Элементтин бар экендигин текшеребиз
     if(people.contains("Асан")){

       System.out.println("ArrayList тизмесинде Асан бар");
     }

     // бир канча объектти очуробуз
     // белгилуу элементти очуробуз
     people.remove("Манас");
```

```
        // индекс боюнча очуруу
        people.remove(0);

        Object[] peopleArray = people.toArray();
        for(Object person : peopleArray){

            System.out.println(person);
        }
    }
}
```

Программанын жыйынтыгы:

```
Жанна
ArrayList те 5 элмент бар
Асан
Манас
Болот
Марат
Анна
ArrayList тизмесинде Асан бар
Болот
Марат
Анна
```

Бул жерде тизме String тибиндеги гана маанилерди камтып турат. Элементти тизмеге кошуу үчүн add методу колдонулду, белгилүү бир жерге кошууда индекс боюнча кошуу каралды people.add(1, "Жанна"). size() методу тизмедеги адамдардын санын билүү үчүн колдонулду. Тизмеден адамды өчүрүү үчүн remove методун колдондук people.remove("Манас").

Queue (Кезектер) жана ArrayDeque классы

Кезектер FIFO (биринчи кирген - биринчи чыгат) принцибинде иштеген маалымат структурасын билдирет. Башкача айтканда, элемент коллекцияга канчалык эрте кошулса, ал андан ошончолук

эрте алынып салынат. Бул стандарттуу бир багыттуу кезек модели. Бирок, эки багыттуулары да бар, башкача айтканда, элементти башталышына гана эмес, аягына да кошо алабыз. Ошого жараша элементти аягынан гана эмес, башынан да алып салса болот.

Кезек класстарынын өзгөчөлүгү, алар атайын Queue же Deque интерфейстерин ишке ашырат.

Queue интерфейси

Queue<E> интерфейси базалык Collection интерфейсин кеңейтет жана класстын жүрүм-турумун бир багыттуу кезек катары аныктайт. Ал төмөнкү ыкмалар аркылуу өзүнүн функционалдуулугун ачып берет:

- **E element()**: кезектин башынан элементти кайтарат, бирок аларды өчүрбөйт. Эгерде кезек бош болсо, анда NoSuchElementException катасы жаралат.
- **boolean offer(E obj)**: кезектин аягына obj элементин кошот. Эгерде элемент кошулса true, антпесе false маанисин кайтарат.
- **E peek()**: кезектин биринчи элементин өчүрбөй туруп кайтарат. Эгерде кезек курук болсо null мааниси кайтарылат.
- **E poll()**: кезектин башынан элементти өчүрүү менен кайтарат. Эгерде кезек курук болсо null маанисин кайтарат.
- **E remove()**: кезектин башынан элементти өчүрүү менен кайтарат. Эгерде кезек бош болсо, анда NoSuchElementException катасы жаралат.

Ошентип, бул интерфейсти ишке ашырган бардык класстар кезекке кошуу үчүн **offer** методу, кезектин башынан элементти алуу

үчүн **poll** методу жана кезектин башынан элементти жөн эле чыгарып алуу үчүн **peek** жана **element** методдоруна ээ болот.

Deque интерфейси

Deque интерфейси жогоруда сүрөттөлгөн Queue интерфейсин кеңейтет жана эки багыттуу кезектин жүрүм-турумун аныктайт, ал кадимки бир багыттуу кезек катары же LIFO (акыркы келип - биринчи чыккан) стек катары иштейт.

- **void addFirst(E obj)**: кезектин башына элемент кошот
- **void addLast(E obj)**: obj элементин кезектин аягына кошот
- **E getFirst()**: кезектин башынан элементти өчүрбөй туруп кайтарат. Эгерде кезек бош болсо, анда NoSuchElementException катасы жаралат.
- **E getLast()**: кезектин аягынан элементти өчүрбөй туруп кайтарат. Эгерде кезек бош болсо, анда NoSuchElementException катасы жаралат.
- **boolean offerFirst(E obj)**: кезектин башына obj элементин кошот. Эгерде элемент кошулса true, антпесе false маанисин кайтарат
- **boolean offerLast(E obj)**: кезектин аягына obj элементин кошот. Эгерде элемент кошулса true, антпесе false маанисин кайтарат.
- **E peekFirst()**: кезектин биринчи элементин өчүрбөй туруп кайтарат. Эгерде кезек курук болсо null мааниси кайтарылат.
- **E peekLast()**: кезектин акыркы элементин өчүрбөй туруп кайтарат. Эгерде кезек курук болсо null мааниси кайтарылат
- **E pollFirst()**: кезектин башынан элементти өчүрүү менен кайтарат. Эгерде кезек курук болсо null маанисин кайтарат.

- **E pollLast()**: кезектин акырынан элементти өчүрүү менен кайтарат. Эгерде кезек курук болсо null маанисин кайтарат
- **E pop()**: кезектин башынан элементти өчүрүү менен кайтарат. Эгерде кезек курук болсо NoSuchElementException катасын кайтарат
- **void push(E element)**: кезектин башына элемент кошот.
- **E removeFirst()**: кезектин башынан элементти өчүрүү менен кайтарат. Эгерде кезек курук болсо NoSuchElementException катасын кайтарат.
- **E removeLast()**: кезектин аягынан элементти өчүрүү менен кайтарат. Эгерде кезек курук болсо NoSuchElementException катасын кайтарат.
- **boolean removeFirstOccurrence(Object obj)**: биринчи жолуккан obj элементти кезектен өчүрөт. Эгерде элемент өчсө true, антпесе false маанисин кайтарат.
- **boolean removeLastOccurrence(Object obj)**: акыркы жолуккан obj элементти кезектен өчүрөт. Эгерде элемент өчсө true, антпесе false маанисин кайтарат.

Ошентип, pop жана push методдоруна ээ болуу бул элементти ишке ашырган класстарга стек катары иштөөгө мүмкүндүк берет. Ошол эле учурда, учурдагы функция эки багыттуу кезектерди түзүүгө мүмкүндүк берет, бул интерфейсти колдонгон класстарды кыйла ийкемдүү кылат.

ArrayDeque классы

Java тилинде кезектер бир нече класстар менен көрсөтүлөт. Алардын бири ArrayDeque<E> классы. Бул класс AbstractCollection классынан функцияны мурастап, Deque интерфейсин колдонуу менен жалпыланган эки багыттуу кезекти билдирет.

ArrayDeque классында төмөнкүдөй конструкторлор бар:

- ArrayDeque(): курук кезекти түзөт
- ArrayDeque(Collection<? extends E> col): col коллекциясынын элементтеринен турган кезекти түзөт.
- ArrayDeque(int capacity): capacity өлчөмүнөн турган кезекти түзөт. Эгерде биз кезектин өлчөмүн көрсөтпөсөк, анда ал 16 барабар болот.

Мисалы:

```
package progs;

import java.util.ArrayDeque;

public class Example51 {

    public static void main(String[] args) {

        ArrayDeque<String> states = new ArrayDeque<String>();
        // элементтерди стандарттык кошуу
        states.add("Germany");
        states.addFirst("France"); //элементти башына кошуу
        states.push("Great Britain"); //элементти эң башына кошуу
    states.addLast("Spain");//элементти коллекциянын аягына кошуу
        states.add("Italy");

        // биринчи элементти получаем очурбой туруп алуу
        String sFirst = states.getFirst();
        System.out.println(sFirst);     // Great Britain
        // акыркы элементти очурбой туруп алуу
        String sLast = states.getLast();
```

```
    System.out.println(sLast);      // Italy

   System.out.printf("Кезектин олчому: %d \n", states.size());

    // коллекцияны бирден карап чыгуу
    while(states.peek()!=null){
      // башынан баштап кароо
      System.out.println(states.pop());
    }

    // Person объекттеринин кезеги
    ArrayDeque<Person> people = new ArrayDeque<Person>();
    people.addFirst(new Person("Asan"));
    people.addLast(new Person("Anna"));
    // шартсыз карап чыгуу
    for(Person p : people){

      System.out.println(p.getName());
    }
  }
}
class Person{

  private String name;
  public Person(String value){

    name=value;
  }
  String getName(){return name;}
}
```

Set интерфейси жана HashSet классы

Set интерфейси Collection интерфейсин кеңейтет жана уникалдуу элементтердин жыйындысынан турат. Set жаңы методдорду кошпойт, ал мураска калгандарга гана өзгөртүүлөрдү киргизет. Тактап айтканда, add() ыкмасы коллекцияга элемент кошот жана коллекцияда андай элемент жок болсо, true маанисин кайтарат.

HashSet жалпы классы хэш таблицасын билдирет. Ал өзүнүн функционалдуулугун AbstractSet классынан мурастап, ошондой эле Set интерфейсин ишке ашырат.

HashSet классынын объектин түзүү үчүн, сиз төмөнкү конструкторлордун бирин колдоно аласыз:

- HashSet(): курук көптүктү түзөт.
- HashSet(Collection<? extends E> col): коллекциянын col элементтеринен турган хеш-таблицаны түзөт.
- HashSet(int capacity): capacity параметри баштапкы өлчөмдү көрсөтөт. Ал көрсөтүлбөсө демейдө 16га барабар болот.
- HashSet(int capacity, float koef): koef параметри же толтуруу коэффициенти 0.0 дөн 1.0 аралыгында болот жана канчалык өлчөмдө кеңейе тургандыгын көрсөтүп турат. Мисалы 0.75 болсо, анда өлчөмү ¾ кө кеңейе турганын билдирет.

HashSet классын колдонуп программа түзүп көрөлү:

```
package progs;
import java.util.HashSet;
public class Example52{
  public static void main(String[] args) {

    HashSet<String> states = new HashSet<String>();
    // тизмеге элементтерди кошолу
    states.add("Germany");
    states.add("France");
    states.add("Kyrgyzstan");
    // коллекцияда бар элементти кошуп короолу
    boolean isAdded = states.add("Germany");
```

```
        System.out.println(isAdded);    // false

        System.out.printf("Коптукто %d элемент бар \n", states.size());    // 3

        for(String state : states){

            System.out.println(state);
        }
        // элементти очуруу
        states.remove("Germany");

        // Person объекттеринин хеш-таблицасы
        HashSet<Person> people = new HashSet<Person>();
        people.add(new Person("Marat"));
        people.add(new Person("Bolot"));
        people.add(new Person("Uson"));
        for(Person p : people){

            System.out.println(p.getName());
        }
    }
}
class Person{

    private String name;
    public Person(String value){

        name=value;
    }
    String getName(){return name;}
}
```

Программанын жыйынтыгы:

```
false
Коптукто 3 элемент бар
France
Germany
Kyrgyzstan
Bolot
Uson
Marat
```

Жогоруда түзүлгөн программада HashSet классынын методдорунун иштөөсүн, коллекцияга кошуу, анын өлчөмүн кеңейтүү операцияларын карап көрдүк.

Map интерфейси жана HashMap классы

Биз жогоруда карап чыккандай эле **Map** структурасы "ачкыч-маани" ("ключ — значение") жуптарынан турган коллекция. Мындан тышкары, бул коллекцияда ар бир ачкыч уникалдуу болуп саналат: эки ачкыч бирдей мааниге ээ эмес. Бул жыйнак кээде сөздүк (dictionary) деп да аталат.

Map элементтери менен иштөөнүн негизги методдору:

- орнотуу (put, putAll)
- алуу (get, keySet, values, entrySet)
- өчүрүү (remove, clear)
- издөө (containsKey, containsValue)
- коллекциянын бош экенин текшерүү (isEmpty)
- өлчөмү (size)

Бардык интерфейстер үчүн базалык класс болуп абстракттуу класс **AbstractMap** эсептелет, ал **Map** интерфейсинин методдорунун көбүн ишке ашырат. Эң кеңири таралган **Map** классы - **HashMap**, ал **Map** интерфейсин ишке ашырат жана **AbstractMap** классынан мураска алат.

HashMap классын мисалда карап көрөлү:

```
package progs;
import java.util.*;
public class Example53{
   public static void main(String[] args) {

      Map<Integer, String> states = new HashMap<Integer, String>();
      states.put(1, "Germany");
      states.put(2, "Spain");
      states.put(4, "France");
      states.put(3, "Kyrgyzstan");

      // 2 ачкычы менен маанини алабыз
      String first = states.get(2);
      System.out.println(first);
      // бардык ачкычтарды алабыз
      Set<Integer> keys = states.keySet();
      // бардык маанилерди алабыз
      Collection<String> values = states.values();
      //элементти алмаштыруу
      states.replace(1, "Poland");
      // 2 ачкычтагы маанини очуруу
      states.remove(2);
      // элементтерди карап чыгуу
      for(Map.Entry<Integer, String> item : states.entrySet()){

         System.out.printf("Ачкыч: %d  Мааниси: %s \n", item.getKey(),
  item.getValue());
      }

      Map<String, Person> people = new HashMap<String, Person>();
      people.put("1240i54", new Person("Asan"));
      people.put("1564i55", new Person("Bolot"));
      people.put("4540i56", new Person("Marat"));

      for(Map.Entry<String, Person> item : people.entrySet()){

         System.out.printf("Ачкыч: %s  Мааниси: %s \n", item.getKey(),
  item.getValue().getName());
      }
    }
  }
  class Person{
```

```
    private String name;
    public Person(String value){

      name=value;
    }
    String getName(){return name;}
}
```

Программанын жыйынтыгы:

```
Spain
Ачкыч: 1  Мааниси: Poland
Ачкыч: 3  Мааниси: Kyrgyzstan
Ачкыч: 4  Мааниси: France
Ачкыч: 1564i55  Мааниси: Bolot
Ачкыч: 4540i56  Мааниси: Marat
Ачкыч: 1240i54  Мааниси: Asan
```

Жогоруда каралган мисалда Map классынын states объектиси түзүлдү жана ага put методу менен ачкыч: маани жазылды (states.put(1, "Germany")). Экинчи объект people төмөнкүдөй түзүлдү:

```
Map<String, Person> people = new HashMap<String, Person>();
```

Ага Person классынын жаңы объекттери түзүлүп кошулду:

```
people.put("1240i54", new Person("Asan"));
```

HashMap классынын кээ бир методдорунун иштөөсү каралды.

Бышыктоочу суроолор:

1. Коллекция деп эмнени айтабыз?
2. Массивдин коллекциядан айрмасы барбы?
3. List(тизме) интерфейси жөнүндө эмне билесиз?
4. Queue (кезек) интерфейси жөнүндө эмне билесиз?
5. Set (көптүк) интерфейси жөнүндө эмне билесиз?
6. Map интерфейси жөнүндө эмне билесиз?

13- практикалык сабак

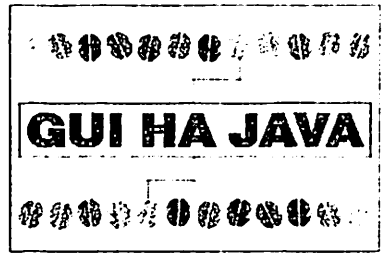
Максаты: Коллекциялар жана интерфейстер, атайын класстар жана алардын методдору жөнүндө алган маалыматтарды бышыктоо жана маселелерди чыгаруу.

Лекциялык сабакта иштелген мисалдарды карап чыккыла жана төмөнкү мисалдарды чагаргыла.

Өз алдынча иштөөгө мисалдар

1. Колдонуучу сандардын тизмесин бир жолчо катары кийирет (мисалы, "1, 2, 3, 4, 4, 5"). Жолчодогу кайталанган элементтерди алып салыңыз жана натыйжаны экранда көрсөтүңүз.
2. HeavyBox объекттерин камтыган коллекция түзүңүз. Коллекциянын элементтери кайталанган жана кутучалардын салмагын текшерген метод жазыңыз. Коробканын салмагы 300 граммдан ашса, куту башка коллекцияга которулат.
3. Оюнчуктун атын жана объектисин (Тоу классы) камтыган жуп маалыматтан турган HashMap түзүңүз. Оюнчуктардын аттарын жана маанисин (HashMap (values())) бөлүп алып чыгарыңыз.
4. Төмөнкү мүнөздөмөлөрдү камтыган Student классын түзүңүз: аты-жөнү, тайпасы, курсу, сабактар боюнча баа. Student классынын объекттерин камтыган коллекцияны түзүңүз. Бардык студенттерди кайталаңыз, ар бир окуучунун орточо баасын эсептеңиз. Орточо балл >=3 болсо, студент кийинки курска которулат, болбосо окуудан чыгарылат.
5. Кыргызча -англисче сөздүктүн программасын түзүп чыгыңыз.

Лекция 15. Java тилинин графикалык мүмкүнчүлүктөрү



Максаты: *Java тилинде колдонуучунун графикалык интерфейсин түзүү үчүн Swing библиотекасын орнотууну үйрөнүү, графикалык класстар менен таанышуу жана ар кандай терезелерди, графиктерди тургузууну үйрөнүү.*

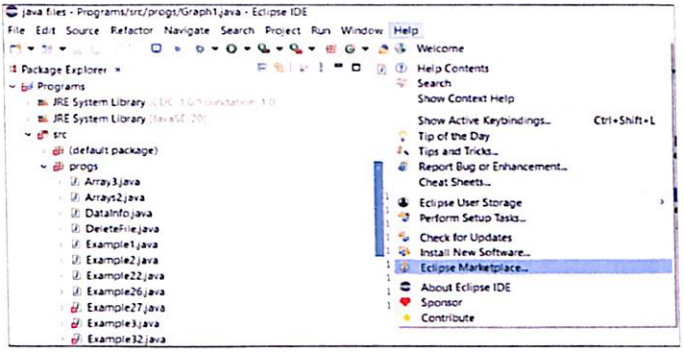
Графикалык интерфейсти орнотуу

Графикалык интерфейсти түзүү үчүн Java тилинде бир канча библиотекалар бар. Алардын ичинен төмөнкүлөрдү карап чыгабыз:

- AWT (Abstract Window Toolkit) – Платформага көз каранды графикалык интерфейс.
- Swing – Платформадан көз карандысыз графикалык интерфейс.

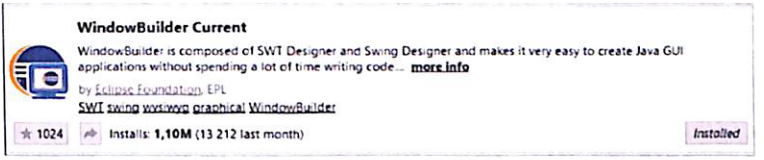
Eclipse программасында Java тилинин колдонуучу үчүн графикалык интерфейсин (GUI) жүктөө үчүн төмөнкү кадамдарды аткарабыз:

1. Eclipse программасын иштетебиз
2. Негизги менюдан Help менюсуна барабыз да ал жерден Eclipse MarketPlace камтылуучу менюсун тандайбыз (30-сүрөт).



30-сүрөт. Eclipsете GUI жүктөө

Андан кийин экранда Eclipse MarketPlace диалог терезеси пайда болот. Ал жерден "Windows Builder" библиотекасын издейбиз. Ал төмөнкүдөй болуп чыгат (31-сүрөт).



31-сүрөт. Windows Builder библиотекасы

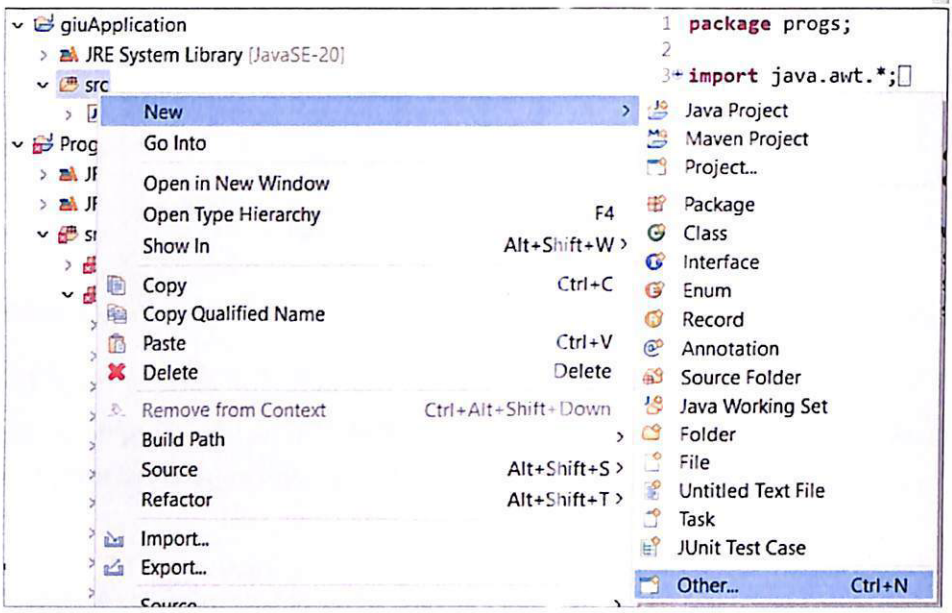
"Install" кнопкасын басуу менен жүктөп алабыз. Биздин учурда ал жүктөлгөн, ошондуктан Installed деп турат. Жүктөө учурунда пайда болгон диалогдук терезеден керектүү кызматтарды белгилеп алабыз. Керектүү компоненттер жүктөлгөндөн кийин Eclipse программасын кайрадан иштетебиз.

Андан кийин жаңы проектти түзөбүз. Мисалы, биз giuApplication деп түзүп алдык (32-сүрөт).



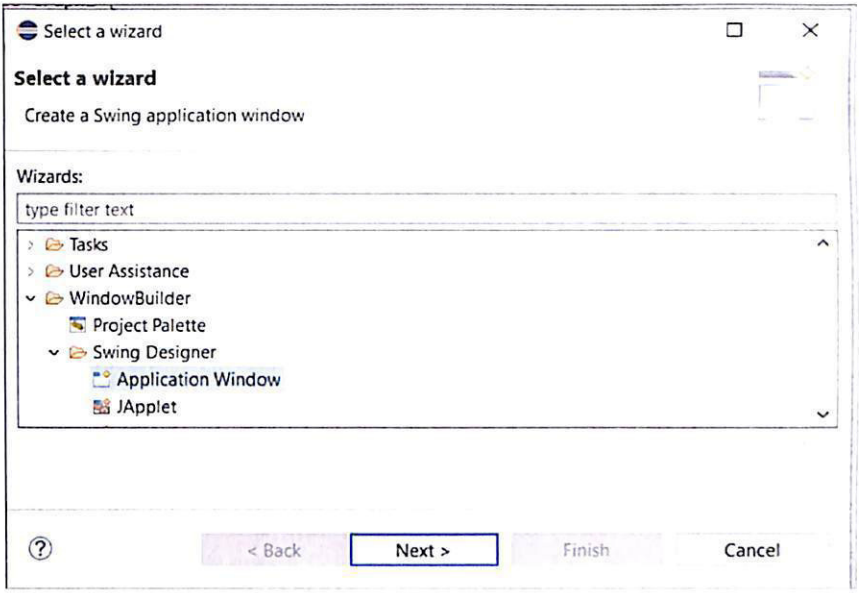
32-сүрөт. Жаңы проектти түзүү

Кийинки кадамда src->New->Other тандап алабыз (33-сүрөт).



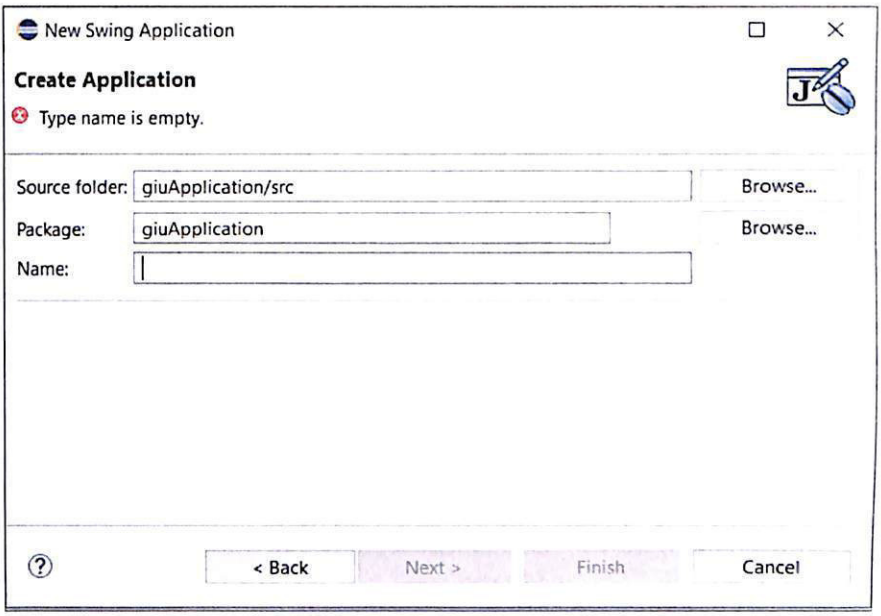
33-сүрөт. Other менюсун тандоо

"Other" менюсун тандагандан кийин экранда "Select wizard" диалогдук терезеси пайда болот. Ал жерде тизмеден "Windows Builder" табабыз, анын ичинен "Swing Designer" басабыз. Андан ары "Application Window" пунктун тандап "Next" кнопкасын басабыз (34-сүрөт).



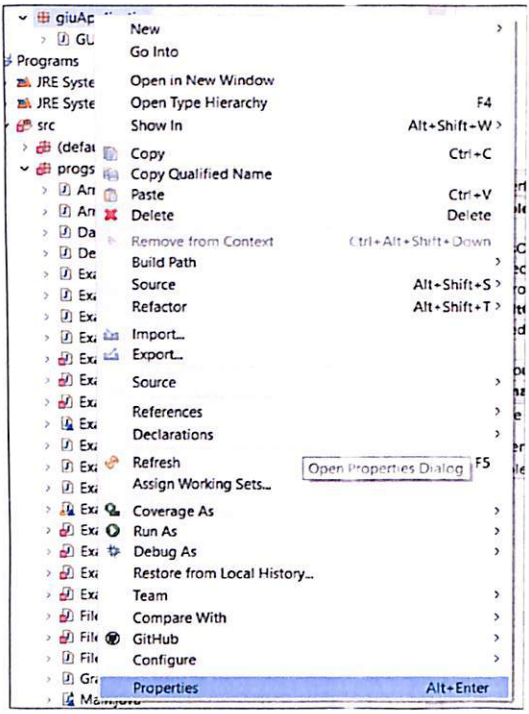
34-сүрөт. Windows тиркемесин түзүү

Кийинки кадамда төмөнкү терезе пайда болот(35-сүрөт):



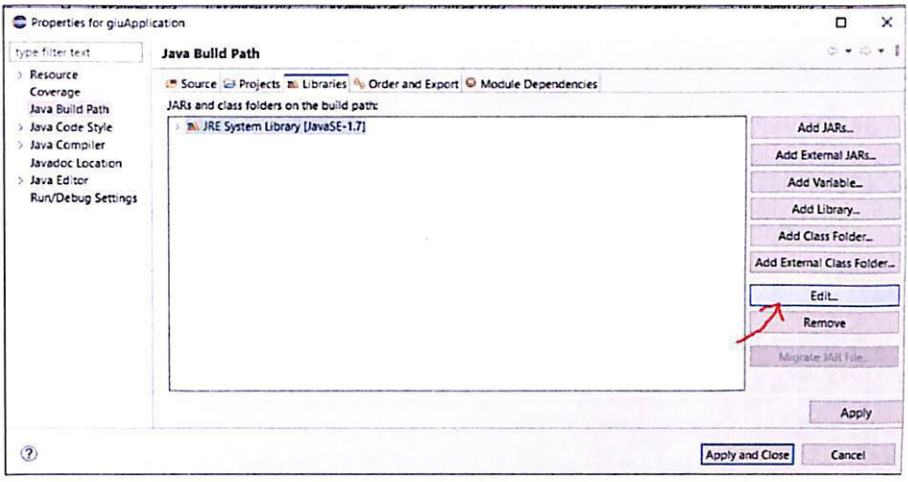
35-сүрөт. Проектти түзүү

Тиешелүү жолчого проекттин аты жазып "Finish" кнопкасын басабыз. Экранда проектин структурасы пайда болот. Ал жерден өзүбүз түзгөн пакетке барып, чычкандын оң жагын басып, пайда болгон менюдан Properties камтылуучу менюну тандайбыз (36-сүрөт):



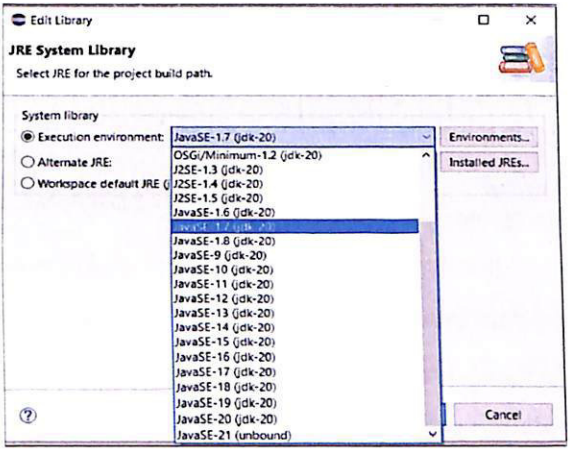
36-сүрөт. Проекттин касиеттерин өзгөртүү

Экранда проект үчүн касиеттер ачылат. Сол жактагы тизмеден "Java Build Path" деген элементти тандайбыз. Ортоңку бөлүктөн библиотеканын атын басып, оң жактагы кнопкалардын ичинен "Edit" кнопкасын басабыз (37-сүрөт).



37-сүрөт. Проект үчүн керектүү библиотеканы тандоо

Андан кийин “Edit Library” терезесинен 38-сүрөттө көрсөтүлгөндөй javaSE-1.7(jdk-20) тандайбыз да “Ok” кнопкасын басып чыгабыз.



38-сүрөт. Библиотеканы өзгөртүү

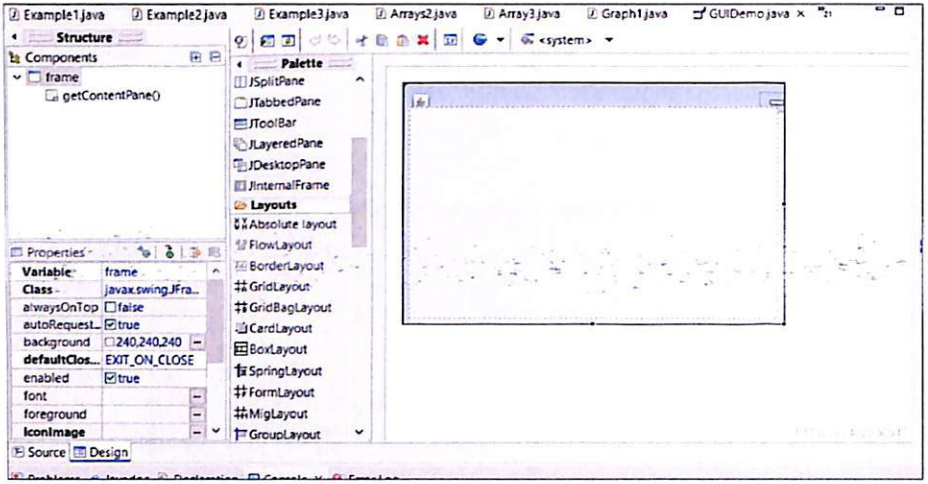
Негизги терезенин төмөн жагында 2 өтмөк пайда болот: Source, Design. Биринчи өтмөк программанын кодун көрүү үчүн колдонулса, экинчи өтмөк колдонуучу үчүн интерфейсти өзгөртүү, түзүү үчүн колдонулат.

Келиңиз Design өтмөгүн басып көрөлү (39-сүрөт).



39-сүрөт. Пайда болгон өтмөктөр

Сиз байкагандай бул учурда инструменттер панелинен турган windows терезеси конструктор режиминде пайда болот (40-сүрөт).



40-сүрөт. Java GUI интерфейси

Swing библиотекасы

Swing библиотекасынын бардык класстары javax.swing пакетинде жайгашкан. Swing библиотекасы AWT библиоткасынын негизги топтому болуп саналат жана платформадан көз карандысыз колдонуучунун интерфейсин түзүү мүмкүнчүлүгүн берет.

Анонимдүү камтылуучу класстар

AWT жана Swing библиотекаларында анонимдүү камтылуучу класстарды кеңири колдонушат. Бул класстардын аты жок жана аларды башка класстардын жана туюнтмалардын ичинде аныктоого

болот. Анонимдүү камтылуучу классты мурастоочу негизги класстан түзсө болот же интерфейсти ишке ашырса болот. Мисалы:

```
public class Example {
  public static void main(String args[]) {
    // Object объектисинин кадимки үлгүсү
    System.out.println(new Object());
    // Object классынан мураска алынган
    // анонимдүү класстын экземпляры
    Object o = new Object() {
      public String toString() {
        return "Анонимдүү класс #1";
      }
    };
    System.out.println(o);
       // Амалдын ичинде түзүлгөн
       // Object классынан мураска алынган
    // анонимдүү класстын экземпляры
    System.out.println(new Object() {
      public String toString() {
        return "Анонимдүү класс #2";
      }
    });
  }
}
```

Runnable интерфейси

Runnable интерфейси Java тилинде агымдарды реалдаштырууга багытталган. Анын жалгыз гана void run() методу агымды иштеткенде чакырылат.

```
public interface Runnable {
  public void run();
```

```
}
```

Swing библиотекасын колдонуу

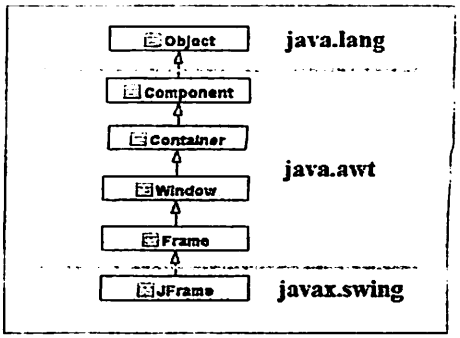
Swing библиотекасы менен бардык аракеттер Event Dispatch Thread окуяларды иштеп чыгуу агымында ишке ашырылышы керек. Ал жерге өзүбүздүн кодду кошуу үчүн атайын статикалык EventQueue.invokeLater(Runnable) методу каралган жана ал аргумент катары Runnable интерфейсин ишке ашыруучу класстын экземплярын кабыл алат.

Мисалы:

```
import java.awt.*;
import javax.swing.*;
public class Example {
  public static void main(String args[]) {
    EventQueue.invokeLater(new Runnable() {
      public void run() {
        // Swing библиотекасын колдонуу
      }
    });
  }
}
```

Жогорку деңгээлдеги терезе. JFrame классы

JFrame классы - бул аталыш тилкеси, рамкасы жана жабуу, кичирейтүү жана чоңойтуу баскычтары бар жогорку деңгээлдеги терезе (41-сүрөт).



41-сүрөт. Пакеттердин жана класстардын иерархиялык тепкичи

Jframe классынын объекттерин түзүү үчүн атайын конструкторлор 1-таблицада көрсөтүлдү.

1-таблица. JFrame классынын негизги конструкторлору

Конструктор	Мисалы	Түшүндүрмө
JFrame()	JFrame frame = new JFrame();	Конструктор бош терезени түзүп берет
JFrame(String title)	JFrame frame = new JFrame("Аталышы");	title аталышы бар бош терезени түзүп берет

Jframe классынын негизги методдору 2-таблицада көрсөтүлдү.

2-таблица. JFrame классынын кээ бир методдору

Метод	Түшүндүрмө
int getDefaultCloseOperation() void setDefaultCloseOperation(int operation)	Колдонуучу терезени жабуу баскычын басканда аткарылган аракетти алуу же орнотуу.
void setLayout(LayoutManager manager)	Windows Layout Manager орнотуу.

Component классынын негизги методдору 3-таблицада көрсөтүлдү.

3-таблица. Component классынын кээ бир методдору

Метод	Түшүндрмө
void addKeyListener(KeyListener l)	Клавиатура үчүн жаңы окуяны башкаруучу метод
void addMouseListener(MouseListener l)	Чычкан баскычын басуу окуясын иштеткичти кошот.
void addMouseMotionListener (MouseMotionListener l)	Чычканды жылдыруу окуясын иштеткичти кошот
Color getBackground() void setBackground(Color c)	Фондун түсүн кайтарат же орнотот
Color getForeground() void setForeground(Color c)	Алдынкы пландын түсүн кайтарат же орнотот
int getHeight()	Бийиктигин кайтарат
int getWidth()	Туурасын кайтарат
int getX()	x координатасын кайтрат
int getY()	y координатасын кайтрат
void repaint()	Терезени кайра тартат
void repaint(int x, int y, int width, int height)	Терезенин бөлүгүн кайра тартат
void setBounds(int x, int y, int width, int height)	Жаңы чектерди коюу менен терезенин өлчөмүн өзгөртөт жана жылдырат
void setSize(int width, int height)	Туурасын жана бийиктигин орнотот
void setVisible(boolean b)	Терезени көрсөтөт же бекитет

Container классынын негизги методдору 4-таблицада көрсөтүлдү.

4-таблица. Container классынын кээ бир методдору

Метод	Түшүндрмө
Component add(Component comp)	Контейнерге компонент кошот
Component findComponentAt(int x, int y)	Берилген чекитте жайгашкан компонентти кайтарат
Insets getInsets()	Контейнердеги боштуктарды кайтарат
void remove(Component comp)	Контейнерден компонентти өчүрөт
void removeAll()	Контейнерден бардык компоненттерди өчүрөт

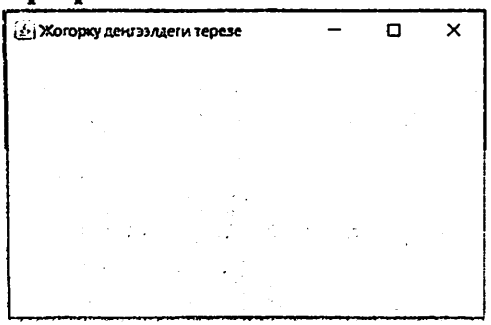
Мисал катары жогоруда каралган JFrame классынын методдорун колдонуп колдонуучунун терезесин түзүп көрөлү:

```
package progs;

import java.awt.*;
import javax.swing.*;
public class Graph1 {
  public static void main(String args[]) {
    EventQueue.invokeLater(new Runnable() {
      public void run() {
        JFrame frame =
          new JFrame("Жогорку деңгээлдеги терезе");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

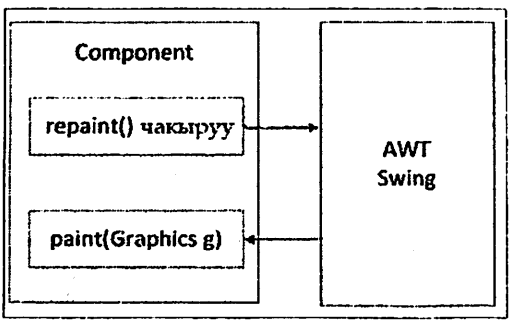
```
            frame.setSize(300, 200);
            frame.setVisible(true);
        }
    });
  }
}
```

Программанын жыйынтыгы:



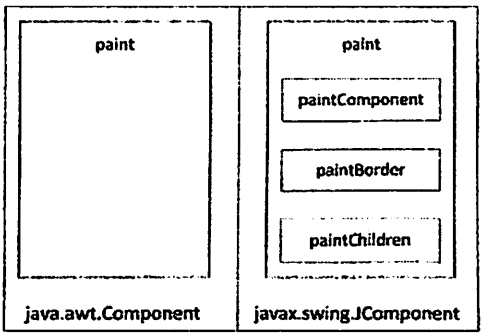
Терезеге сүрөт тартуу. Component жана JComponent класстарындагы сүрөт тартуунун өзгөчөлүктөрү

JComponent классы компонентти сүрөт тартууга жооп берген void paint(Graphics g) методун аныктайт. Бул метод компоненттин кээ бир аймагын кайра чийүү керек болгондо void repaint() методу чакырылат (42-сүрөт).



42-сүрөт. Component классынын объекттерин кайра тартуу

AWT библиотекасынын Component классынан айырмаланып, JComponent классында paint методу компонентти тартпайт, бирок тиешелүүлүгүнө жараша компоненттин өзүн, компоненттин кадрын тарткан paintComponent, paintBorder, paintChildren методдорун чакырат (43-сүрөт). JFrame классы Component классын камтыгандыктан, анын чийилиши paint методунда болушу керек. Swing классынын Jcomponent'тен мурастап алган башка класстары paintComponent методу менен чийүүнү аткарышат.



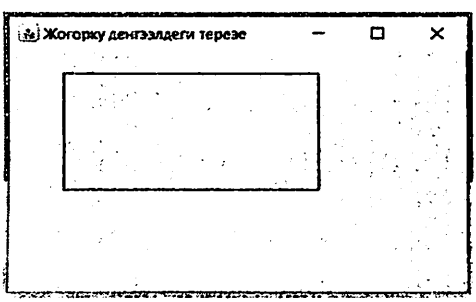
43-сүрөт. Component жана JComponent класстарынын кайра тартуу методдорунун айырмасы

Component классынын тартуу методуна мисал келтирип көрөлү:

```
package progs;
import java.awt.*;
import javax.swing.*;
public class Graph2 extends JFrame {
  public Graph2() {
    super("Жогорку денгээлдеги терезе");
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setSize(300, 200);
    setVisible(true);
```

```
  }
  public void paint(Graphics g) {
    // компонентти кайра тартуу
    super.paint(g);
    g.drawRect(50, 50, 200, 100);
  }
  public static void main(String[] args) {
    EventQueue.invokeLater(new Runnable() {
      public void run() {
        new Graph2().setVisible(true);
      }
    });
  }
}
```

Программанын жыйынтыгы:



JComponent классынын мисалында карап көрөлү:

```
package progs;


import java.awt.*;
import javax.swing.*;
public class Graph3 extends JFrame {
  public class DrawingPanel extends JPanel {
```

```
    protected void paintComponent(Graphics g) {
      super.paintComponent(g);
      g.drawRect(50, 50, 200, 100);
    }
  }
  public Graph3() {
    super("Жогорку денгээлдеги терезе");
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setSize(300, 200);
    add(new DrawingPanel());
    setVisible(true);
  }
  public static void main(String[] args) {
    EventQueue.invokeLater(new Runnable() {
      public void run() {
        new Graph3().setVisible(true);
      }
    });
  }
}
```

Жыйынтыгында ошол эле терезени алабыз.

Сүрөт тартуу областын аныктоо

JComponent рамканы камтышы мүмкүн, андыктан сүрөт тартууда сиз кадр ээлей турган областтын аянтын эсептешиңиз керек. Бул өлчөмдөрдү GetInsets() функциясы аркылуу алса болот, ал Insets классынын объектисин кайтарат.

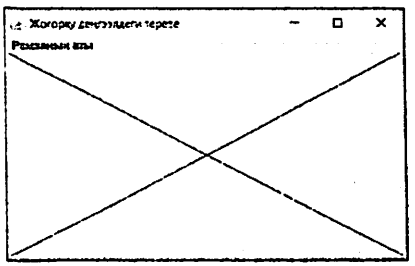
Insets классы left, right, top, bottom талааларын камтыйт, алар сол, оң, жогору жана ылдый жагындагы чегинүүнү аныктайт.

Мисалы:

```
package progs;
import java.awt.*;
import javax.swing.*;
public class Graph2 extends JFrame {
  public class DrawingPanel extends JPanel {
    protected void paintComponent(Graphics g) {
      super.paintComponent(g);
      Insets is = getInsets();
      g.drawLine(is.left, is.top, getWidth() - is.right,
          getHeight() - is.bottom);
      g.drawLine(is.left, getHeight() - is.bottom,
          getWidth() - is.right, is.top);
    }
  }
  public Graph2() {
    super("Жогорку деңгээлдеги терезе");
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setSize(300, 200);
    DrawingPanel dp = new DrawingPanel();
    dp.setBorder(BorderFactory.createTitledBorder(
      "Рамканын аты"));
    add(dp);
    setVisible(true);
  }
  public static void main(String[] args) {
    EventQueue.invokeLater(new Runnable() {
      public void run() {
```

```
            new Graph2().setVisible(true);
        }
    });
  }
}
```

Программанын жыйынтыгы:



Graphics классы

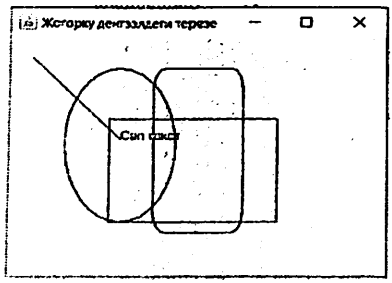
Graphics классынын объекти Component жана JComponent компоненттеринде графикалык примитивдерди тартуу үчүн жооп берет.

Мисалы:

```
package progs;
import java.awt.*;
import javax.swing.*;
public class Graph2 extends JFrame {
   public class DrawingPanel extends JPanel {
      protected void paintComponent(Graphics g) {
         super.paintComponent(g);
         g.drawLine(20, 20, 100, 100);
         g.drawOval(50, 30, 100, 150);
         g.drawRect(90, 80, 150, 100);
         g.drawRoundRect(130, 30, 80, 160, 20, 40);
         g.drawString("Сап текст", 100, 100);
```

```
    }
  }
  public Graph2() {
    super("Жогорку денгээлдеги терезе");
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setSize(300, 250);
    add(new DrawingPanel());
    setVisible(true);
  }
  public static void main(String[] args) {
    EventQueue.invokeLater(new Runnable() {
      public void run() {
        new Graph2().setVisible(true);
      }
    });
  }
}
```

Программанын жыйынтыгы:

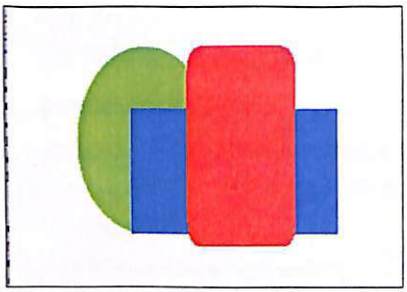


Ушул эле программаны өзгөртүп, түстөрдү колдонуп көрөлү:

```
import java.awt.*;
import javax.swing.*;
public class ForegroundWindow extends JFrame {
  public class DrawingPanel extends JPanel {
    protected void paintComponent(Graphics g) {
```

```
        super.paintComponent(g);
        g.setColor(Color.GREEN);
        g.fillOval(50, 30, 100, 150);
        g.setColor(Color.BLUE);
        g.fillRect(90, 80, 150, 100);
        g.setColor(Color.RED);
        g.fillRoundRect(130, 30, 80, 160, 20, 40);
      }
    }
    public ForegroundWindow() {
      super("Пример окна верхнего уровня");
      setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      setSize(300, 250);
      add(new DrawingPanel());
      setVisible(true);
    }
    public static void main(String[] args) {
      EventQueue.invokeLater(new Runnable() {
        public void run() {
          new ForegroundWindow().setVisible(true);
        }
      });
    }
}
```

Программанын жыйынтыгы:



Color классы

Бул класс түстөр менен иштөөдө колдонулат. Мисалы:

```
package progs;

import java.awt.*;
import javax.swing.*;
public class Graph2 extends JFrame {
   public class DrawingPanel extends JPanel {
      protected void paintComponent(Graphics g) {
         super.paintComponent(g);
         for(int i = 0; i < 256; i += 16)
            for(int j = 0; j < 256; j+= 16) {
               g.setColor(new Color(j, 128, i));
               g.fillRect(i, j, 16, 16);
            }
      }
   }
   public Graph2() {
      super("Түстөр");
      setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      setSize(300, 300);
      add(new DrawingPanel());
      setVisible(true);
   }
```

```
    public static void main(String[] args) {
        EventQueue.invokeLater(new Runnable() {
            public void run() {
                new Graph2().setVisible(true);
            }
        });
    }
}
```

Программанын жыйынтыгы:



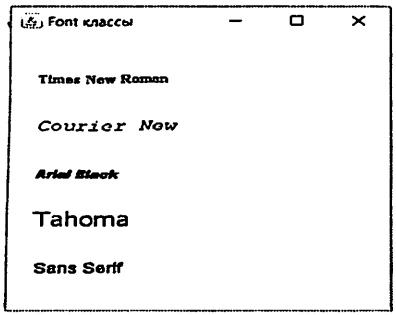
Font классы

Шрифтти башкаруу үчүн Font классы колдонулат. Мисалда карап көрөлү:

```
package progs;
import java.awt.*;
import javax.swing.*;
public class Graph2 extends JFrame {
    public class DrawingPanel extends JPanel {
        protected void paintComponent(Graphics g) {
            super.paintComponent(g);
            g.setFont(new Font("Times New Roman", Font.BOLD, 12));
            g.drawString("Times New Roman", 20, 50);
            g.setFont(new Font("Courier New", Font.ITALIC, 16));
            g.drawString("Courier New", 20, 100);
            g.setFont(new Font("Arial Black", Font.ITALIC, 10));
            g.drawString("Arial Black", 20, 150);
```

```
        g.setFont(new Font("Tahoma", Font.PLAIN, 20));
        g.drawString("Tahoma", 20, 200);
        g.setFont(new Font("Sans Serif",
          Font.ITALIC & Font.BOLD, 14));
        g.drawString("Sans Serif", 20, 250);
      }
    }
    public Graph2() {
      super("Font классы");
      setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      setSize(300, 300);
      add(new DrawingPanel());
      setVisible(true);
    }
    public static void main(String[] args) {
      EventQueue.invokeLater(new Runnable() {
        public void run() {
          new Graph2().setVisible(true);
        }
      });
    }
}
```

Программанын жыйынтыгы:



Бышыктоочу суроолор:

1. Java тилинин графика менен иштөөчү кайсы класстары бар?

2. Eclipse чөйрөсүндө колдонуучунун графикалык интерфейсин кантип жүктөп алабыз?

3. Swing жана AWT класстары боюнча айтып бериңиз?

4. Терезелер менен иштөөчү JFrame классы жөнүндө айтып бериңиз

6. Color классы боюнча эмне билесиз?

14-практикалык сабак

Максаты: Java тилинин графикалык мүмкүнчүлүктөрү, атайын класстар жана алардын методдору жөнүндө алган маалыматтарды бышыктоо жана маселелерди чыгаруу.

Келгиле, чогуу төмөнкү маселени карап көрөлү.

Маселе 1.

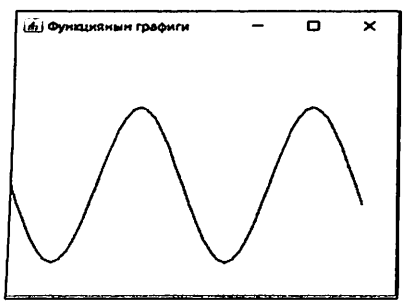
sin(x) функциясынын графигин чийип көрөлү:

```
package progs;

import java.awt.*;
import javax.swing.*;
public class Graph2 extends JFrame {
   public class DrawingPanel extends JPanel {
      protected void paintComponent(Graphics g) {
         super.paintComponent(g);
         int xPoints[] = new int[50];
         int yPoints[] = new int[50];
         for(int i = 0; i < 50; i++) {
            xPoints[i] = i * 6;
            yPoints[i] =
```

```
                (int)(80*Math.sin(i * Math.PI / 12)) + 150;
            }
            g.drawPolyline(xPoints, yPoints, 50);
        }
    }
    public Graph2() {
        super("Функциянын графиги");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(300, 300);
        add(new DrawingPanel());
        setVisible(true);
    }
    public static void main(String[] args) {
        EventQueue.invokeLater(new Runnable() {
            public void run() {
                new Graph2().setVisible(true);
            }
        });
    }
}
```

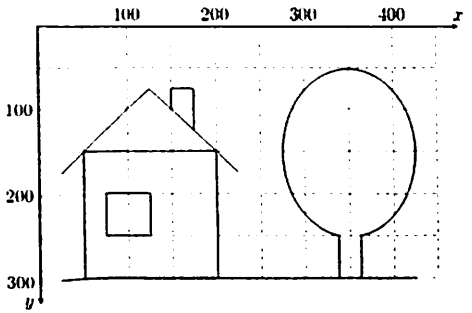
Программанын жыйынтыгы:



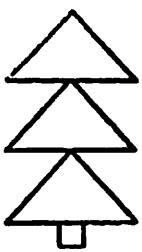
Өз алдынча иштөөгө мисалдар

Төмөнкү фигуралардын экранга чыгаруучу программа түзгүлө:

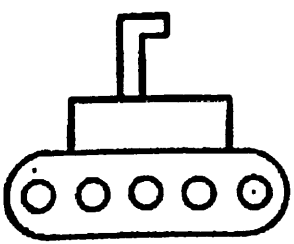
1.



2.



3.



4.



5. Java тилинде *y=cos(x)* функциясынын графигин тургузгула

6. Java тилинде калькулятордун программасын түзгүлө

Колдонулган адабияттар

1. Рикардо Флеск "Баштоочулар үчүн Java" Электрондук китеп. URL: https://www.scribd.com/document/458693839/File-1

2. Герберт Шилдт "Java™ : Толук маалымдама, Жетинчи чыгарылыш". The McGraw-Hill компаниясы. Нью-Йорк, 2007.

3. Ативонг Сучато "101 мисалдар менен JAVA компьютердик программалоо тилин үйрөнүү". Банкок, 2011

4. Дэвид Дж. Эк. "Java'ны колдонуу менен программалоого киришүү", Нью-Йорк, 2006.

5. Васильев А.Н. "Java мисалдар жана программалар менен", Илим жана Техника , Санкт-Петербург, 2016

Электрондук булактар:

1. https://proglang.su/java

2. https://javarush.com/

3. https://www.w3schools.com/java/

Тесттик тапшырмалар

1. Javaны кандай максатта колдонсо болот?

a) сайт түзүү үчүн

b) жекече компьютер үчүн программа түзүү үчүн

c) оюн программаларын түзүү үчүн

d) тиркемелерди түзүү үчүн

e) жогорудагылардын баары үчүн

2. Комментарийдин мүмкүн болгон синстаксиси кайсы?

a) /* Комментарий */

b) # Комментарий

c) /* Комментарий

d) // Комментарий

3. Төмөндө көрсөтүлгөн типтердин кайсынысы Java тилинде жок?

a) int

b) float

c) string

d) unknown

e) Double

4. Java тилинде (IDE) иштеп чыгуу чөйрөсүн колдонбостон туруп программа түзүүгө болобу?

a) Ооба

b) Жок, учурдагы кодду байт-кодго компляциялоо зарыл

5. Java тилиндеги файлдар кандай кеңейтилишке ээ?

a) <файлдын аты>.javac

b) <файлдын аты>.java

c) <файлдын аты>.class

d) <файлдын аты>.classpath

6. Файлда бирден ашык Java классы камтышы мүмкүнбү?

a) Ооба, бирок бир сырткы класста public модификатору болгондо гана

b) Ооба, сырткы класстар private модификаторуна ээ болсо

c) жок

7. Өзгөрмөнүн аталыштарын жазууда төмөндөгүлөрдүн кайсынысы синтаксистик жактан туура эмес?

a) 53someVariable

b) _someVariable

c) some-variable

d) somevariable

e) someVariable

f) Somevariable53

g) _53someVariable

8. Java тилинде түзүүлүүчү класстар кайсы класстан мурасталат?

a) Класстар **extends** ачкыч сөзүнөн кийин көрсөтүлгөн тип боюнча мурасталат. Эгер тип көрсөтүлбөсө класс мурасталбаган болуп эсептелет.

b) Object

c) Class

9. Java тилинде кайсы пакетти импорттоо автоматтык түрдө жүргүзүлөт?

a) Бардык пакеттерди анык көрсөтүү керек.

b) java.util

c) java.lang

d) java.text

10. Берилгендердин ичинен кайсы типтеги маалымат эстутумда эң чоң мейкиндикти ээлей турганын көрсөтүңүз.

a) int

b) byte

c) double

d) float

e) баары бирдей

11. Java тилинде кайсы арифметикалык оператор жок?

a) --

b) %
c) **
d) ++

12. Java тилинде кайсы салыштыруу оператору жок?

a) !=

b) ==

c) <>

d) ===

e) >=

13. Кайсы өзгөртүп түзүү типтери автоматтык түрдө ишке ашпайт?

a) byte тиби short тибине

b) int тиби long тибине

c) char тиби short тибине

d) long тиби float тибине

14. Берилгендердин кайсы тибин switch оператору колдобойт?

a) int

b) String

c) char

d) long

15. Кайсы өзгөрмө жана массив түзүү туюнтмасы синтаксистик жактан туура эмес?

a) int array[] = new int[0];

b) int[] array = new int[1];

c) int array = new int[1];

d) int[] array = new int[] {1, 2};

e) int[] array = {1, 2};

16. Методдун ашыкча жүктөмүнө эмне таасир этпейт?

a) параметрлердин саны

b) кайтарылуучу маанинин тиби

c) жеткиликтүүлүк модификатору

d) параметрлердин тиби

17. void ачкыч сөзүнүн мааниси кандай?

a) мурастоо классында метод кайра аныкталбайт

b) метод эч нерсе кайтарбайт

c) методду ашыкча жүктөө мүмкүн эмес

18. extends ачкычтык сөзүнүн кызматы кандай?

a) кайсы класстан мурастоо жүрүп жатканын көрсөтөт

b) өзгөрмөнүн кеңейтилгенин билдирет

c) мурастоо аткарылбай турганын көрсөтөт

19. Кайсы ачкыч сөз жеткиликтүүлүк модификатору эмес?

a) public

b) package

c) private

d) protected

20. class SomeClass { } классынын конструктору кантип чакырылат?

a) construct()

b) constructor()

c) SomeClass()

d) someClass()

21. Төмөндөгү типтердин кайсынысы примитивдүү типке кирбейт?

a) int

b) Byte

c) char

d) short

e) Object

f) String

22. Сандык типтерди гана белгилегиле?

a) int

b) String
c) double
d) char
e) short
f) long

23. Java тилинде өзүбүздүн примитивдик типти түзүүгө болобу?

a) ооба

b) жок

24. short тиби эс тутумдан канча бит көлөмдү ээлейт?

a) 32

b) 8

c) 16

d) 64

25. Төмөкүлөрдүн кайсынысы компилациилоо учурунда катага жолугат?

a) int a = 12;

b) int a = 014;

c) int a = 1_2;

d) int a = 0b1100;

e) баары туура

26. Программанын жыйынтыгы кандай болот?

```
public class Main {
        public static void main(String[] args) {
            int a = 10;
            some(a++);
        }

        public static void some(int a) {
            System.out.println(a);
        }
}
```

a) 11

b) 12

c) 10

27. Программанын жыйынтыгы кандай болот?

```
public class Main {
    public static void main(String[] args) {
        byte a = 127;
        System.out.println(++a);
    }
}
```

d) 11

e) 12

10

28. Жолчонун узундугун аныктоо үчүн кайсы метод колдонулат?

a) length()

b) size()

c) long()

d) count()

29. String классынын length() методу кайсы типтеги маанини кайтарат?

a) double

b) String

c) int

d) number

30. Массивди кантип туура жарыялайбыз?

a) int a = new int[] {1, 2, 3, 4, 5};

b) int[] a = new int[] {1, 2, 3, 4, 5};

c) int[] a = new int {1, 2, 3, 4, 5};

d) int a[] = 1, 2, 3, 4, 5;

e) int[] a = int[] {1, 2, 3, 4, 5};

Басууга берилди: 19.07.2024-ж.
Форматы 60x90 1/16. Көлөмү: 15 ш.б.т.
Буюртма № 32/24. Нускасы: 100 даана
"Book-дизайн" полиграфия борбору
Кыргыз Республикасы, Ош шаары, И. Сулайманов көч. 3.



Көчкөнбаева
Буажар Осмоналиевна

техника илимдеринин кандидаты Ош мамлекеттик университетинин "Информациялык системалар жана программалоо" кафедрасынын доценти. 20 дан ашык илимий макаланын, 1 окуу китебинин жана 5 окуу-усулдук колдонмонун автору.



Шакиров
Кылычбек Курбанбекович

Ош мамлекеттик университетинин STEM инновациялык колледжинин Информациялык коммуникациялык дисциплиналар циклынын башчысы. 10дон ашык илимий эмгектин автору анын ичинен бирөө окуу методикалык колдонмо.



Сейитказыева
Гулнара Имамалиевна

КРдин Жогорку Кеңешинин Ыраазычылык катынын ээси, Ош мамлекеттик университетинин математика, физика, техника жана информациялык технологиялар институтунун Автоматташтырылган системалар жана санариптик технологиялар кафедрасынын улук окутуучусу. 15ке жакын илимий эмгектин автору.

39466