

Java Start

Курс Java Start - раскрывает базовые приемы программирования на языке Java. Рассмотрены базовые понятия используемые в программировании и их реализация в Java. Также рассматриваются базовые приемы работы с компонентами стандартной библиотеки JDK

Киев 2016

Составил: Цымбалюк А.Н.

Структура и описание курса

Курс предназначен для тех кто хочет изучить язык программирования Java. Курс рассчитан на новичка в программировании. Однако все таки для прохождения этого курса нужно обладать следующими навыками:

- Иметь базовые понятия о арифметических операциях, также знать что такое модуль числа и как извлекать из числа квадратный корень.
- Нужно иметь базовые навыки владения ПК

Курс структурирован по изучаемым темам. Это поможет с изучением последовательности базовых приемов программирования. В конце каждого основного раздела дано «домашнее задание» выполнение которого будет способствовать закреплению пройденного материала.

Курс снабжен как теоретическим материалом, так и большим количеством примеров практической реализации той или иной задачи.

Также в курсе описано много дополнительного материала не вошедшего в основной курс, но играющего важную роль в современной разработке. Однако при первом чтении его можно опустить без ущерба для понимания дальнейшего материала. Надеемся этот курс поможет вам в изучении Java и станет хорошим подспорьем для дальнейшего развития.

Для связи с автором данного курса можно использовать e-mail:
tsimbalukalexander@gmail.com

Оглавление

- 1) Введение
- 2) Дополнительный материал (Работа в IDE NetBeans and IntelliJ IDEA)
- 3) Переменные
- 4) Условные операторы
- 5) Дополнительный материал (Debugging и отладка)
- 6) Циклы
- 7) Массивы
- 8) Методы
- 9) Элементы стандартной библиотеки (Дата, Работа со строками)
- 10) Работа с файлами
- 11) Final

Java Start - Введение

Дорогу осилит идущий...

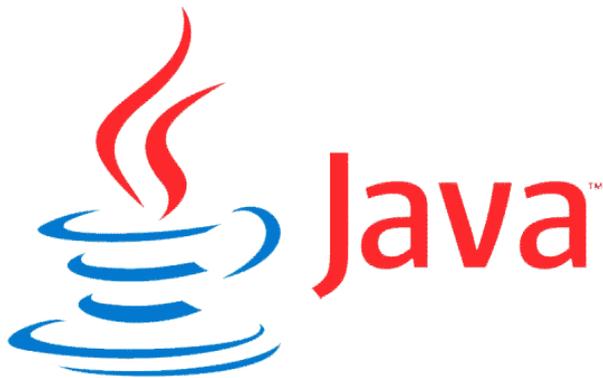
Разработал: Цымбалюк А.Н.

Краткие сведения о языке Java

Java — объектно-ориентированный язык программирования, разработанный компанией Sun Microsystems (в последующем приобретенной компанией Oracle).

Приложения Java обычно транслируются в специальный байт-код, поэтому они могут работать на любой виртуальной Java-машине вне зависимости от компьютерной архитектуры.

Дата официального выпуска — 23 мая 1995 года.



← Эмблема языка разработки Java

Достоинства языка Java

Программы на Java транслируются в байт-код JVM

- Независимость байт-кода от операционной системы и оборудования (Windows, Mac OS X, Linux etc.)
- Автоматическое управление памятью (GC)
- Гибкая система безопасности
- Скорость

Платформы Java

Java SE (J2SE) - Java для настольных компьютеров

Java EE (J2EE) — Java для работы на серверах

Java ME (J2ME) — Java для работы на микроконтроллерах

JavaFX — предназначена для создания графических интерфейсов корпоративных приложений и бизнеса.

Java Card - технология предоставляет безопасную среду для приложений, работающих на смарт-картах и других устройствах с очень ограниченным объемом памяти и возможностями обработки.

Android

Google App Engine (GAE/Java)

Почему Java хороший выбор?

Aug 2016	Aug 2015	Change	Programming Language	Ratings	Change
1	1		Java	19.010%	-0.26%
2	2		C	11.303%	-3.43%
3	3		C++	5.800%	-1.94%
4	4		C#	4.907%	+0.07%
5	5		Python	4.404%	+0.34%
6	7	^	PHP	3.173%	+0.44%
7	9	^	JavaScript	2.705%	+0.54%
8	8		Visual Basic .NET	2.518%	-0.19%
9	10	^	Perl	2.511%	+0.39%
10	12	^	Assembly language	2.364%	+0.60%
11	14	^	Delphi/Object Pascal	2.278%	+0.87%
12	13	^	Ruby	2.278%	+0.86%
13	11	v	Visual Basic	2.046%	+0.26%
14	17	^	Swift	1.983%	+0.80%

Этот язык популярен. Например по версии tiobe index <http://www.tiobe.com/tiobe-index/> он занимает 1 — е место по популярности в мире

Используется в крупных проектах

ebay

MINECRAFT

Linked in

YAHOO!

Приват 24

одноклассники

Средства Java

Дистрибутивы

- JRE — виртуальная машина (необходима для выполнения программ)
- JDK — комплект разработчика

Скачать можно :

Oracle - <http://www.oracle.com/technetwork/java/javase/downloads/index.html>

OpenJDK - <http://openjdk.java.net/>

Среды разработки

- Eclipse - <https://eclipse.org/home/index.php>
- NetBeans - <https://netbeans.org/>
- IntelliJIDEA - <https://www.jetbrains.com/idea/>

! Также программы на Java можно писать в блокноте

Цикл разработки программ на Java

Структура программы на Java

- **class** (минимальная единица)
- **package** — группа файлов
- **jar** — группа пакетов

Иерархия проектов

- **java** файлы
- Структура каталогов

Компиляция программы:

*.java -> компилятор -> *.class -> JAR

Первая программа

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

Наберите этот текст в блокноте и сохраните файл как Main.java

Компиляция Java программ (пример)

Войдите в командную строку, и в папке с файлом выполните команду

```
javac Main.java
```

После этого появится файл Main.class который можно запустить на выполнение командой

```
java Main
```

Пример компиляции и запуска java приложения

```
alexander@alexander-ThinkPad-Edge-E330:~/Документы/ls_Java$ javac Main.java
```

```
alexander@alexander-ThinkPad-Edge-E330:~/Документы/ls_Java$ java Main  
Hello world
```

IDE упрощает труд программиста

На данный момент наиболее популярными IDE для Java являются:



! В программе курса будет изучаться IDE Eclipse

Eclipse

Бесплатен. Скачать можно тут <https://www.eclipse.org/downloads/>

Внимание скачивать Eclipse нужно той же разрядности (32 и 64) что и JDK

При первом старте Eclipse выведет сообщение о расположении WorkSpace (рабочего пространства) выберите папку где будут храниться ваши проекты и нажмите ОК.

Создание проекта в eclipse

Для создания нового проекта выберите
File -> New-> Java Project

New Java Project

Create a Java Project
Enter a project name.

Project name:

Use default location
Location:

JRE

Use an execution environment JRE:

Use a project specific JRE:

Use default JRE (currently 'java-1.7.0-openjdk-1.7.0.71-2.5.3.0.fc20.i386') [Configure JREs...](#)

Project layout

Use project folder as root for sources and class files

Create separate folders for sources and class files [Configure default...](#)

Working sets

Введите имя проекта
и нажмите Finish

Выполнив щелчок мышью на проекте и вызвав контекстное меню добавляем новый проект (New->Package)

New Java Package

Java Package

Create a new Java package.

Creates folders corresponding to packages.

Source folder:

Lesson/src

Browse...

Name:

lesson1

Create package-info.java



Cancel

Finish

Дадим имя новому проекту и нажимаем Finish

Далее добавляем новый класс в проект для этого вызовем контекстное меню проекта и выберем New-> Java Class

New Java Class

Java Class
Create a new Java class.

Source folder: Lesson/src Browse...

Package: lesson1 Browse...

Enclosing type: Browse...

Name: Lesson1

Modifiers: public package private protected
 abstract final static

Superclass: java.lang.Object Browse...

Interfaces: Add...

Which method stubs would you like to create?

public static void main(String[] args)

Constructors from superclass

Inherited abstract methods

Do you want to add comments? (Configure templates and default values [here](#))

Cancel Finish

Даем имя классу

Устанавливаем галочку для создания главного метода

Нажимает Finish

Появилась окно с текстом проекта. Весь код программы должен принадлежать главному методу класса.

```
package lesson1;  
  
public class Lesson1 {  
  
    public static void main(String[] args) {  
        // TODO Auto-generated method stub  
        System.out.println("HELLO WORLD");  
    }  
  
}
```

Код программы должен быть
внутри тела главного метода



```
public static void main(String[] args) {  
    // TODO Auto-generated method stub  
    System.out.println("HELLO WORLD");  
}
```

Запустить программу на выполнение можно или выбрав Run -> Run
либо нажав на пиктограмму запуска

Некоторые замечания касающиеся кода программы

Обратите внимание на фигурные скобки. Они указывают на логические блоки программы.

```
package lesson1;
```

```
public class Lesson1 {
```

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

```
        // TODO Auto-generated method stub
```

```
        System.out.println("HELLO WORLD");
```

```
    }
```

```
}
```

Это комментарий

Каждая строка (кроме тех что являются началом или концом логического блока) должны заканчиваться точкой с запятой.

Основы компьютерной грамотности

Размерность данных:

- 1 бит : 0 или 1
- 1 байт = 8 бит (10101110)
- 1 килобайт = 1024 байт
- 1 мегабайт = 1024 килобайт
- 1 гигабайт = 1024 мегабайт
- 1 терабайт = 1024 гигабайт

Системы счисления

- Десятичная
- Двоичная - об
- Шестнадцатеричная -ох
-

Алгоритм преобразования числа из любой системы в десятичную

Основание системы (первое число состоящее из 2 цифр)

Например:

- 10-для десятичной
- 10-для двоичной
- 10- шестнадцатеричная

Значение в десятичной системе получается умножением числа на основание системы счисления в степени равной порядку следования разряда (начиная с нуля)

$Hex = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F\}$

Пример преобразования:

$$0x5A3 = 3 \cdot 16^0 + 10 \cdot 16^1 + 5 \cdot 16^2 = 1443$$

Преобразование из десятичной системы счисления в произвольную

Вычисляется частное числа и основания системы счисления, остаток от деления записывается в младший разряд, как только результатом деления станет 0, то алгоритм заканчивается.

$$A = 43 / 2 = 21, B = 43 \% 2 = 1$$

$$A = 21 / 2 = 10, B = 21 \% 2 = 1$$

$$A = 10 / 2 = 5, B = 10 \% 2 = 0$$

$$A = 5 / 2 = 2, B = 5 \% 2 = 1$$

$$A = 2 / 2 = 1, B = 2 \% 2 = 0$$

$$A = 1 / 2 = 0 \rightarrow \text{STOP!!} \rightarrow \text{ob101011}$$

Рекомендуемая литература

1. Герберт Шилд — Java Полное руководство 8-е издание
2. Брюс Эккель — Философия Java
3. Кэти Сьерра и Берт Бейтс - Изучаем Java

Полезные ссылки

1. <http://docs.oracle.com/javase/7/docs/>
2. <http://docs.oracle.com/javase/7/docs/api/>
3. <http://prog.kiev.ua/forum>

Домашнее задание

1) Разобраться с 16-й системой исчисления. Узнать какое число зашифровано в названии группы ACDC

Дополнительный материал

Работа с IDE NetBeans и IntelliJ IDEA

He Eclipse — ом единым жив Java developer

Разработал: Цымбалюк А.Н.

IDE NetBeans

NetBeans IDE — свободная интегрированная среда разработки приложений (IDE) на языках программирования Java, Python, PHP, JavaScript, C, C++, Ада и ряда других.

Проект NetBeans IDE поддерживается и спонсируется компанией Oracle, однако разработка NetBeans ведется независимым сообществом разработчиков-энтузиастов (NetBeans Community) и компанией NetBeans Org.

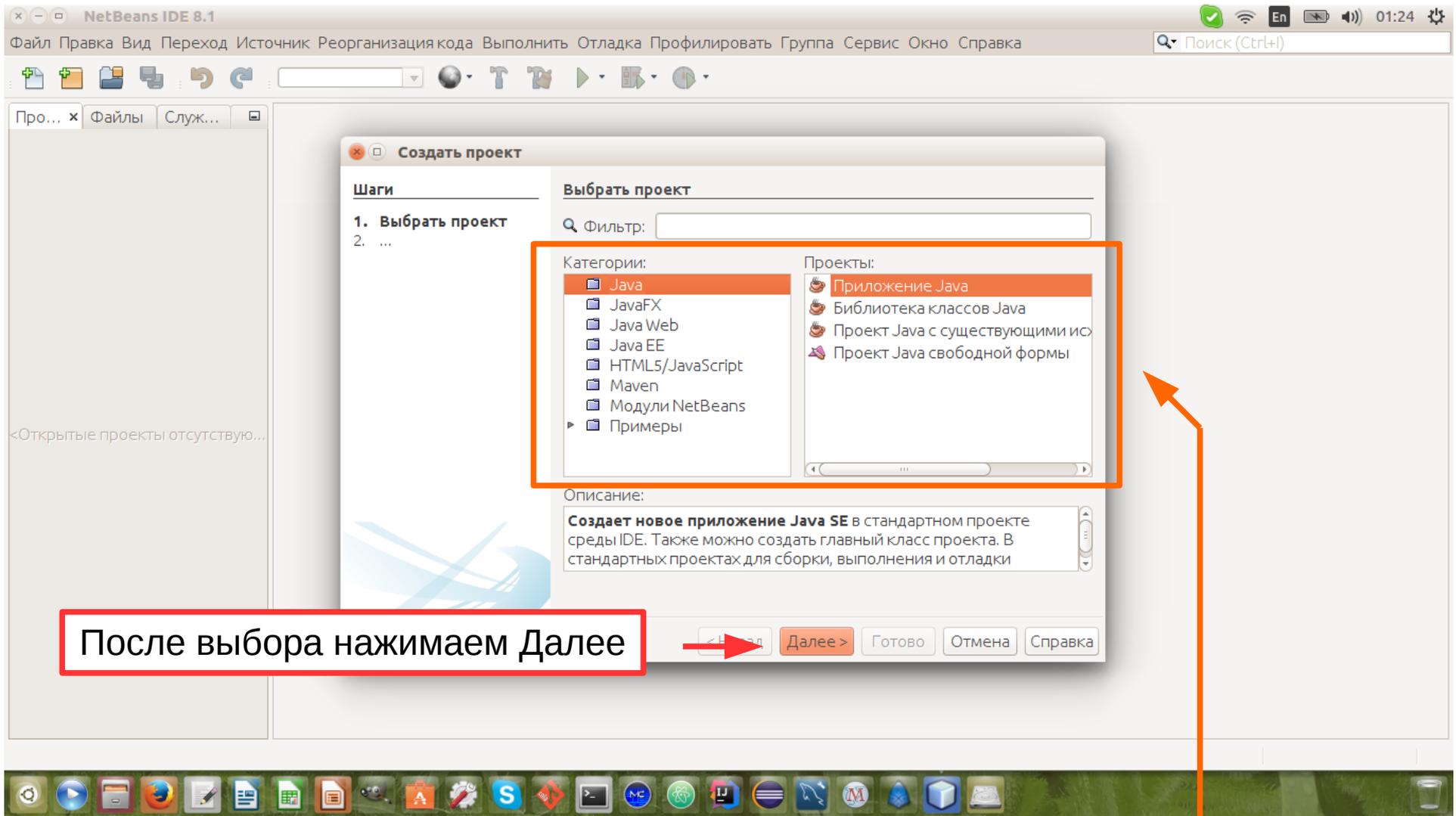
Это единственная среда разработки рекомендованная Oracle для разработки на языке Java.

Ссылка на сайт - <https://netbeans.org/>

Внимание IDE требует для работы Java. Т.е. сначала установите JDK и только потом данную IDE.

Данная IDE бесплатна во всех вариантах и скачивается с сайта производителя.

Создание нового проекта в NetBeans



После выбора нажимаем Далее

Выбираем Файл → Новый проект. В открывшемся диалоговом окне выбираем категорию Java. В проектах — приложение Java

Создание проекта в NetBeans часть 2

Новый Приложение Java

Шаги

1. Выбрать проект
2. **Имя и расположение**

Имя и расположение

Имя проекта: Lesson1

Расположение проекта: /home/alexander/NetBeansProjects Обзор...

Папка проекта: e/alexander/NetBeansProjects/Lesson1

Использовать отдельную папку для хранения библиотек

Папка с библиотеками: Обзор...

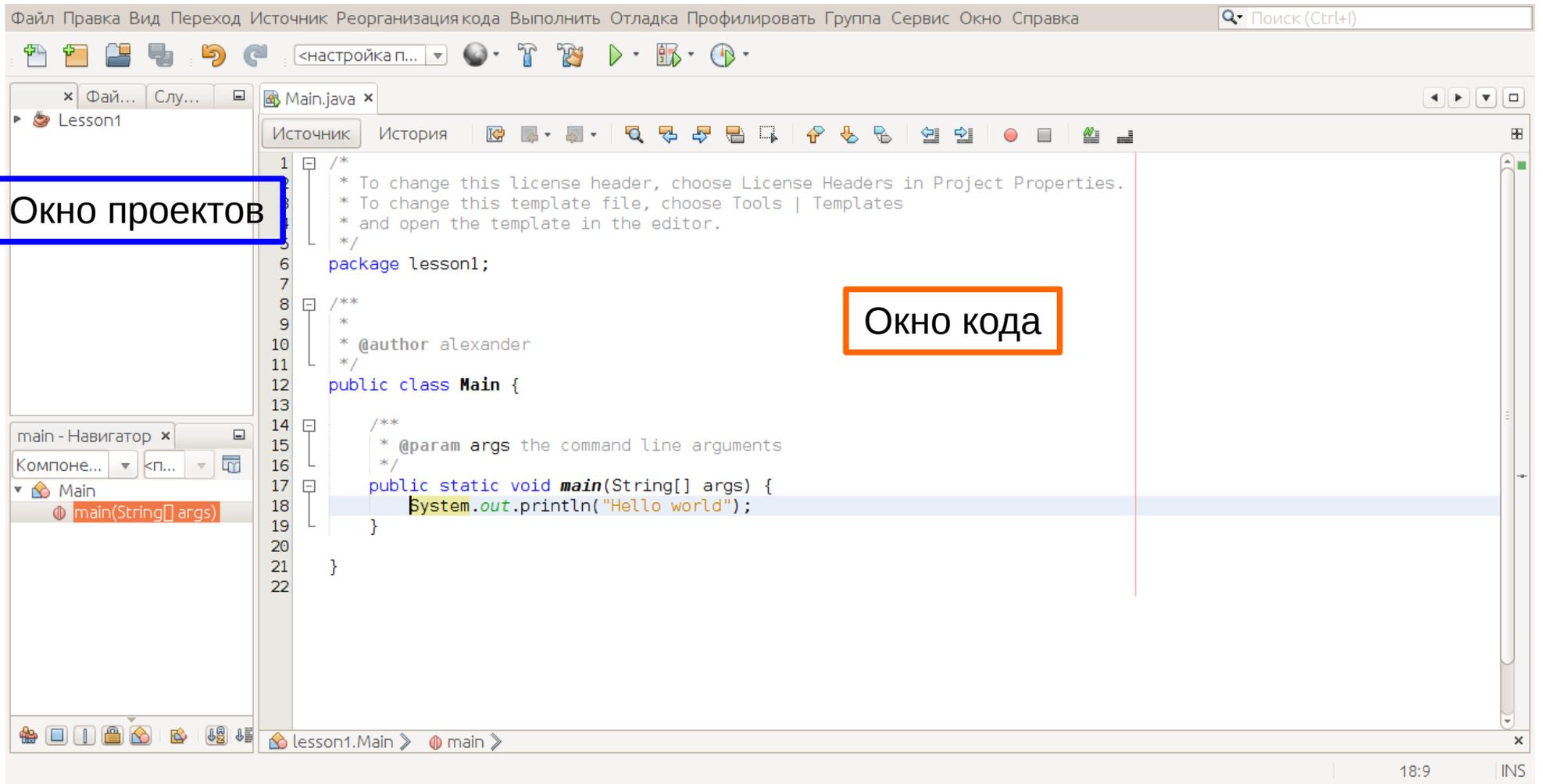
Разные пользователи и проекты могут совместно использовать одни и те же библиотеки компиляции (дополнительная информация находится в справочной системе).

Создать главный класс lesson1.Main

< Назад Далее > **Готово** Отмена Справка

Даем имя проекту и имя главному классу и нажимаем готово.

Работа в IDE NetBeans



Запустить приложение: Выполнить → Запустить проект. Или нажать F6.

IDE IntelliJ IDEA

IntelliJ IDEA — интегрированная среда разработки программного обеспечения на многих языках программирования, в частности Java, JavaScript, Python, разработанная компанией JetBrains.

Начиная с версии 9.0, IntelliJ IDEA доступна в двух версиях: Community Edition и Ultimate Edition. Community Edition является полностью свободной версией, доступной под лицензией Apache 2.0. В ней реализована полная поддержка Java SE, Groovy, Scala, а также интеграция с наиболее популярными системами управления версиями. В версии Ultimate Edition реализована поддержка Java EE, UML-диаграмм, подсчёт покрытия кода, а также поддержка других систем управления версиями, языков и фреймворков.

Ссылка на сайт - <https://www.jetbrains.com/idea/>

Внимание IDE требует для работы Java. Т.е. сначала установите JDK и только потом данную IDE.

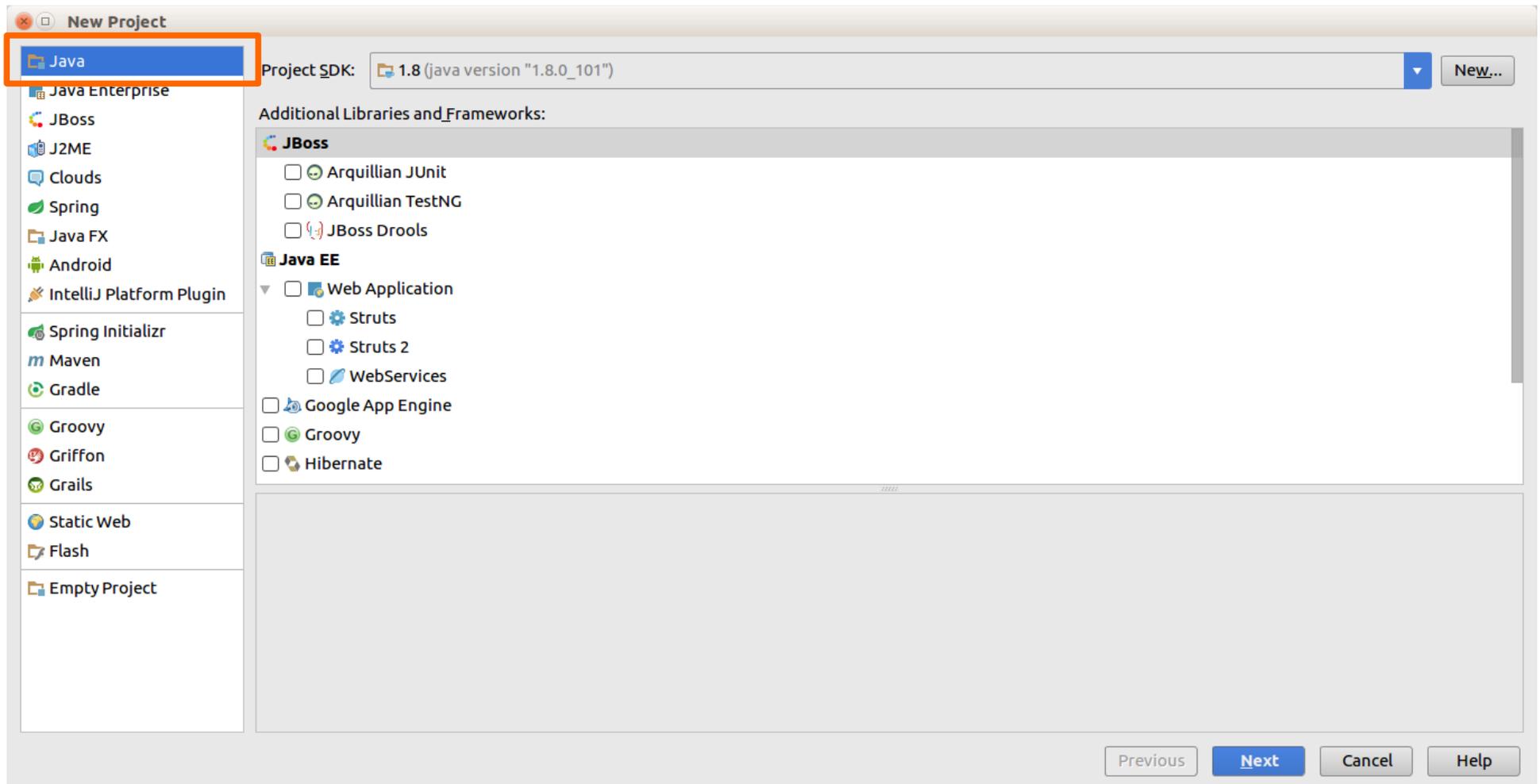
Данная IDE скачивается с сайта производителя бесплатно (только Community Edition).

Создание нового проекта в IntelliJ IDEA



Выберите в меню → Create New Project

Создание нового проекта в IntelliJ IDEA часть 2



Выбираем Java и при необходимости устанавливаем Project SDK. Потом нажимаем Next. В следующем окне (создание проекта из шаблона) просто нажимаем Next.

Создание нового проекта в IntelliJ IDEA часть 3

New Project

Project name: ProgaTwo

Project location: /home/alexander/Документы/Is_Java

More Settings

Module name: ProgaTwo

Content root: /home/alexander/Документы/Is_Java

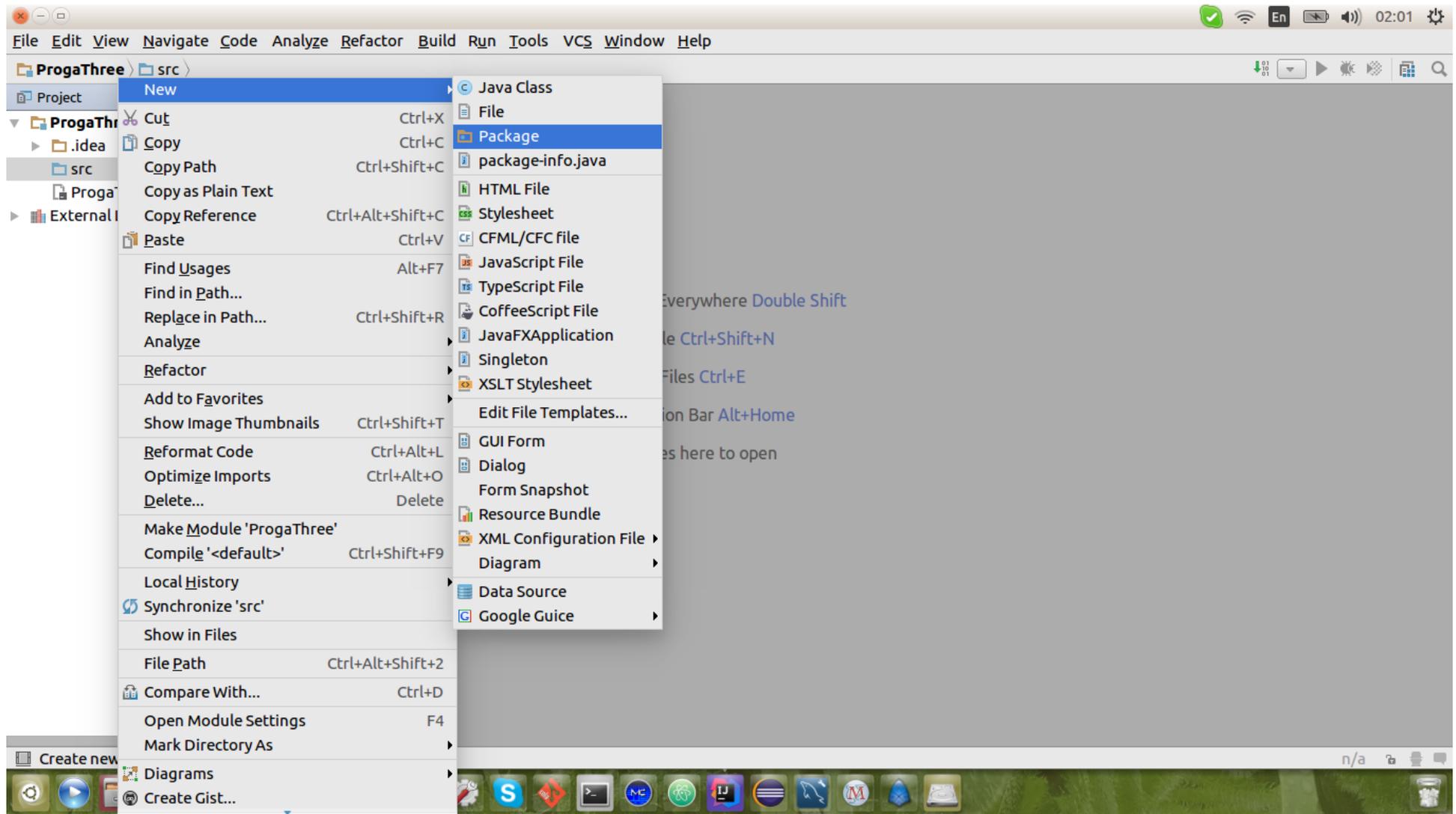
Module file location: /home/alexander/Документы/Is_Java

Project format: .idea (directory based)

Previous Finish Cancel Help

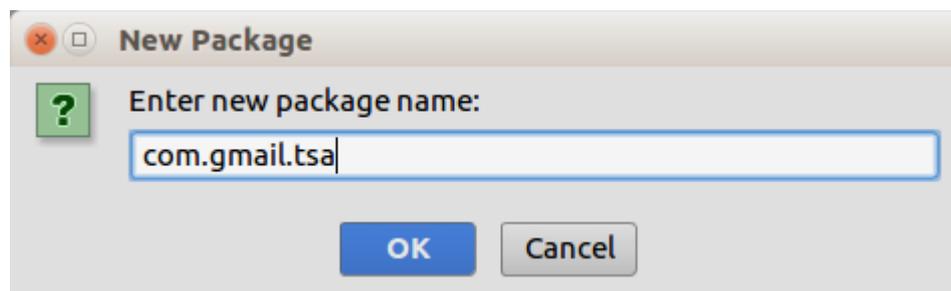
Даем имя проекту и нажимаем Finish

Создание нового проекта в IntelliJ IDEA часть 4



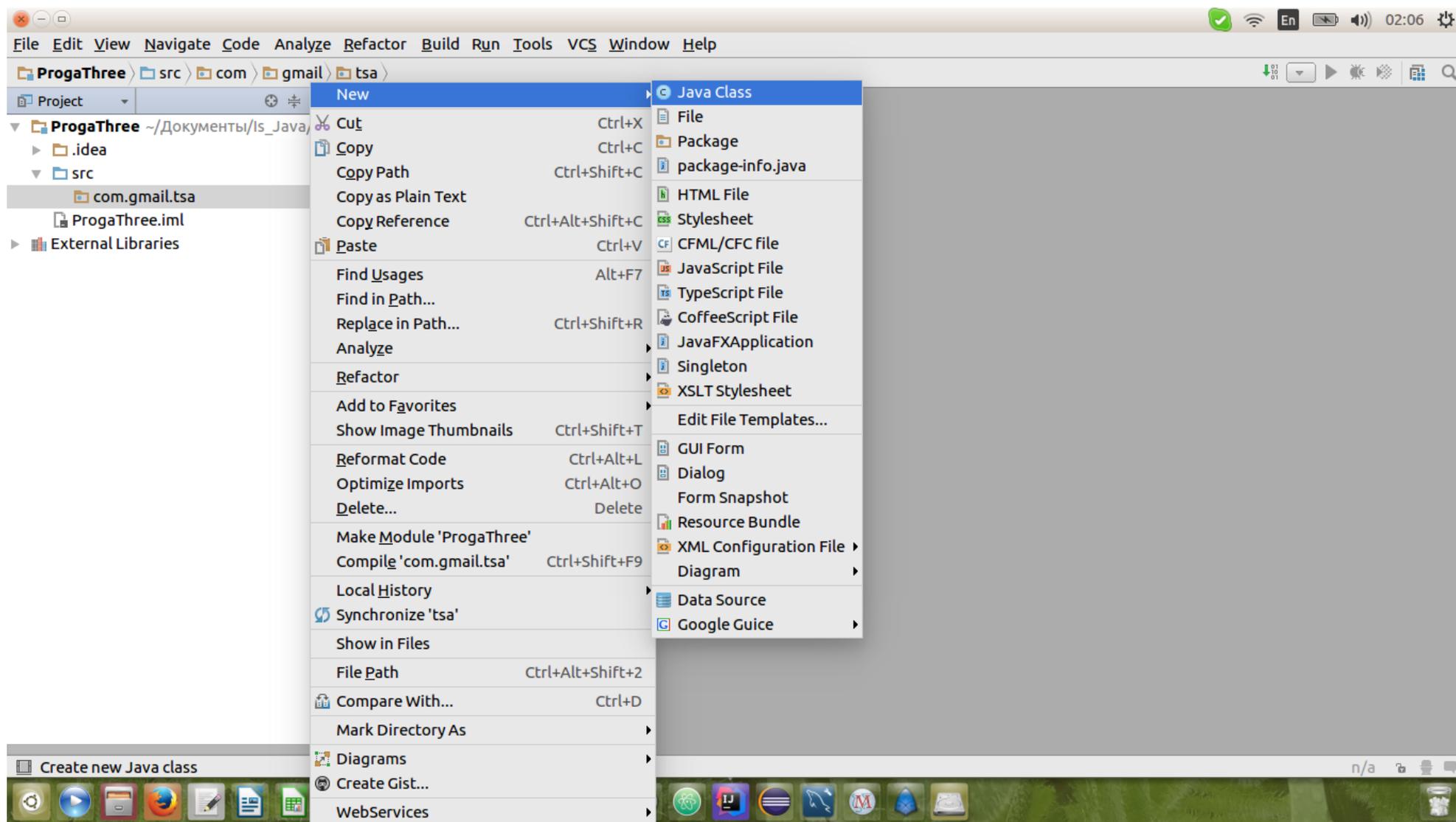
Выполните щелчок на папке src. Вызовите контекстное меню. В нем New → Package

Создание нового проекта в IntelliJ IDEA часть 5



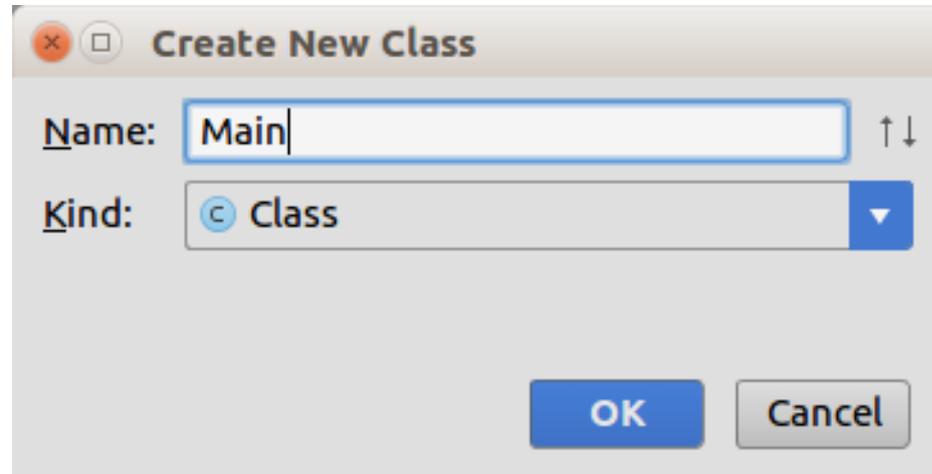
Введите имя пакета и нажмите ОК

Создание нового проекта в IntelliJ IDEA часть 6



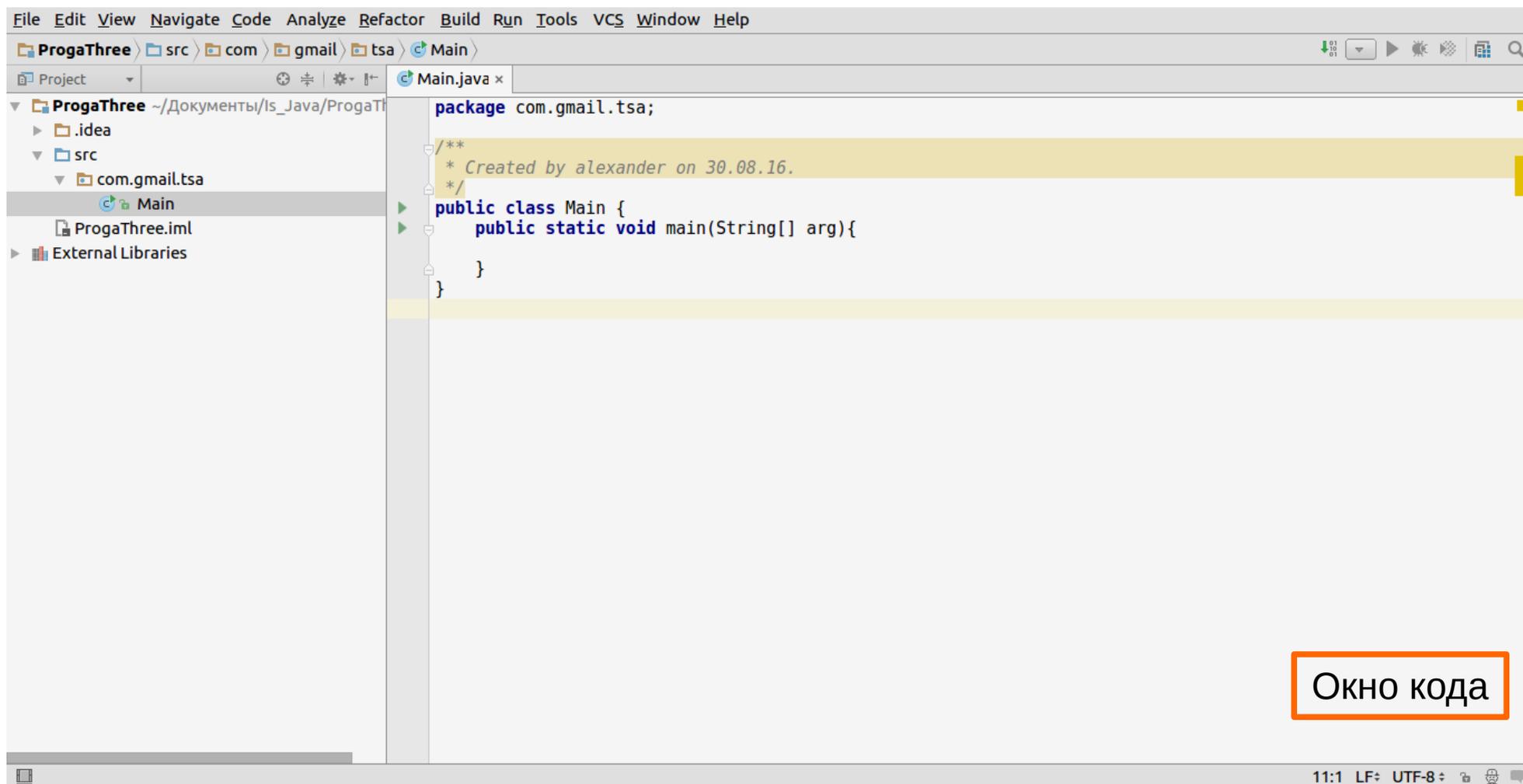
Щелчок на созданном пакете, вызов контекстного меню → New → JavaClass

Создание нового проекта в IntelliJ IDEA часть 7



Выбираем имя нового класса и его тип.

Создание нового проекта в IntelliJ IDEA часть 8



Запустить приложение на выполнение: Run → Run...
Или комбинацией клавиш Alt+Shift+F10

Краткие итоги Лекции

Существует 3 наиболее популярные среды разработки

- **Eclipse**
- **NetBeans**
- **IntelliJ IDEA**

Первые 2 бесплатны, 3 — я бесплатна только в урезанном виде.
В лекции рассмотрена работа с **NetBeans** и **IntelliJ IDEA**. Работа с **Eclipse** рассмотрена в основном курсе.

Java Start

(Переменные)

Переменная – именованная область памяти которую можно использовать для осуществления доступа к данным.

Данные, находящиеся в переменной, называются **значением** этой переменной.

Объявление переменных в Java

В Java переменные объявляются конструкцией вида - **Тип Имя**;

Тип (встроенные типы в Java — примитивные):

- **Числовые**
 - byte
 - short
 - int
 - long
 - float
 - double
- **Символьные**
 - char
- **Логические**
 - boolean

Имя — имя которое вы даете переменной !

Числовой тип переменных

Тип переменной	Размер (бит)	Диапазон
byte	8	-128..127
short	16	-32768..32767
int	32	-2 147 483 648.. 2 147 483 647
long	64	-9 223 372 036 854 775 808.. ..9 223 372 036 854 775 807
float	32	$-3.4 \cdot 10^{38} \dots 3.4 \cdot 10^{38}$
double	64	$-1.8 \cdot 10^{308} \dots 1.8 \cdot 10^{308}$

Символьный и логический тип переменных

Тип переменной	Размер (бит)	Диапазон
char	16	0..65535
boolean	32	true/ false;

Присваивание значений переменных

Имя=**Значение**;

Значение - значение подходящее соответствующему типу переменной.

Инициализация -
присваивание объявленной переменной значения.

! Можно объявлять переменную одновременно с инициализацией.

Тип Имя=**Значение**;

Пример объявления и инициализации переменных

```
package repetishion2;
```

```
public class Prim1 {
```

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

```
        int x;  
        x=5;
```

Объявление переменной x
Инициализация переменной x

```
        x=15;
```

Изменение значения переменной x

```
        double y=3.5;
```

Объявление с присваиванием

```
    }  
}
```

Обратите внимание что разделителем целой и дробной части вещественного числа является точка.

Переменные числового типа:

- Целочисленные (выражаются целыми числами) :
 - byte
 - short
 - int
 - long
- Вещественные (выражаются вещественными числами):
 - float
 - double

! Существуют префиксы для указания системы счисления:
(указывается перед значением)

- 0x - шестнадцатеричная система
- 0 - восьмеричная система
- 0b — двоичная система

Автоматическое приведение типов переменных

Если в выражении используется две и более переменных или литерала, то тип переменной которую возвращает выражение определяется максимальным рангом используемым в выражении. В таблицы со списком числовых данных, они описаны по возрастанию ранга сверху в низ. Т.е. byte имеет минимальный ранг, double максимальный.

! Выражение обычно расширяется в сторону более широкой переменной.

- Если один операнд имеет тип double, другой тоже преобразуется к типу double.
- Иначе, если один операнд имеет тип float, другой тоже преобразуется к типу float.
- Иначе, если один операнд имеет тип long, другой тоже преобразуется к типу long.
- Иначе оба операнда преобразуются к типу int.

Пример: преобразование типов

```
package repetishion2;  
  
public class Prim1 {  
  
    public static void main(String[] args) {  
  
        int x=1;  
        double y=2.3;  
  
        double z=x+y-5.2; ← Преобразование типов  
  
        System.out.println("z= "+z);  
    }  
}
```

Вывод значения переменной в консоль

Результат работы программы

z= -1.900000000000000004

Некоторые следствия из автоматического приведения типов

Целочисленные литералы в Java имеют тип int

```
package com.gmail.tsa;  
  
public class Main {  
    public static void main(String[] args) {  
        byte a = 7;  
        a = a + 12;  
    }  
}
```



Этот код работать не будет. Так как в строке 2 операнда, то тип результата определяется максимальным рангом. А у целочисленного литерала он int. Следовательно тип возвращаемого выражения тоже должен быть int. Однако это не так.

Некоторые следствия из автоматического приведения типов

Вещественные литералы имеют тип double

```
package com.gmail.tsa;  
  
public class Main {  
    public static void main(String[] args) {  
        float a = 7;  
        a = a + 12.5;  
    }  
}
```



Этот код работать не будет. Так как в строке 2 операнда, то тип результата определяется максимальным рангом. У вещественного литерала он double. Следовательно тип возвращаемого выражения тоже должен быть double. Однако это не так.

Некоторые следствия из автоматического приведения типов

Для указания того что целочисленный литерал имеет тип long в конце нужно поставить символ L

```
package com.gmail.tsa;  
  
public class Main {  
  
    public static void main(String[] args) {  
  
        long a = 12345678910L;  
  
    }  
  
}
```

Для указания того что целочисленный литерал имеет тип float в конце нужно поставить символ F

```
package com.gmail.tsa;  
  
public class Main {  
  
    public static void main(String[] args) {  
  
        float a = 3.45F;  
  
    }  
  
}
```

Переполнение целочисленных переменных

! Как только значение переменной превысит максимальный размер ее диапазона, значение станет равно минимальному значению диапазона + значение этой переменной .

Можно привести переменную с более высоким рангом к переменной с более низким для этого можно выполнить принудительное приведение типов. Сначала нужно указать к какому типу приводиться (в скобках) после чего указать что приводиться.

! Указание на принудительное преобразование

(byte)
(short)
(int)
(long)

Пример: переполнение

```
package repetishion2;  
  
public class Prim1 {  
  
    public static void main(String[] args) {  
  
        byte x=127;  
        System.out.println("x= "+x);  
        x=(byte)(x+1);  
        System.out.println("x="+x);  
    }  
}
```

← Переполнение так как результат принудительно приводя к byte

Результат работы программы:

x= 127

x=-128

Строковый тип данных

Тип — `String`;

Значение — "Текст заключенный в кавычки "

Пример:

```
String s1="HELLO";
```

Операторы

Запись	Описание
+	Сложение
-	Вычитание
*	Умножение
/	Деление
%	Деление по модулю
++	Инкремент
--	Декремент

! Сокращенная запись некоторых операторов

Запись	Альтернатива
<code>a+=1;</code>	<code>a=a+1;</code>
<code>a-=1;</code>	<code>a=a-1;</code>
<code>a*=2;</code>	<code>a=a*2;</code>
<code>a/=2;</code>	<code>a=a/2;</code>
<code>a%=2</code>	<code>a=a%2;</code>

Операторы

Для численных данных действие операторов аналогично их математическому представлению.

! Внимание для целочисленных переменных / - целочисленное деление - Например $5/3=1$

Для строковых переменных используется только оператор «+»

В таком случае результатом будет строка равная первой строке к которой дописана вторая строка.
Например : "abc"+"def"="abcdef";

НУЖНО БОЛЬШЕ МАТЕМАТИКИ !!!

Дополнительные математические операции

Синтаксис	Математическое значение
<code>Math.sqrt(x)</code>	\sqrt{x}
<code>Math.abs(x)</code>	$ x $
<code>Math.pow(x,y)</code>	x^y
<code>Math.exp(x)</code>	e^x
<code>Math.log(x)</code>	$\ln(x)$
<code>Math.cos(x)</code>	$\cos(x)$
<code>Math.round(x)</code>	Округление

И еще много полезных вещей в пакете Math

Ввод данных с клавиатуры

```
package repetishion2;
```

```
import java.util.Scanner;
```

Обязательные строки
(пока нужно запомнить)

```
public class Prim1 {
```

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

```
        Scanner sc=new Scanner(System.in);
```

```
        int x;
```

```
        System.out.println("Input x");
```

```
        x=sc.nextInt();
```

Считывание переменной с клавиатуры

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

```
    }
```

```
}
```

Что и как считывать

Тип переменной	Как считывать
byte	sc.nextByte()
short	sc.nextShort()
int	sc.nextInt();
long	sc.nextLong()
float	sc.nextFloat()
double	sc.nextDouble()
String	sc.nextLine()

Преобразование строки в число и обратно

Классы:

Integer, Long, Short, Boolean, Char, Byte.

Преобразование числа в строку:

```
String t = Integer.toString(123);  
String t = Long.toString(123);  
String t = Integer.toString(123, 16);
```

Основание системы
счисления



Преобразование строки в число:

```
int n = Integer.parseInt("123");  
long n = Long.parseLong("123");  
double n = Double.valueOf("123");
```

Краткие итоги лекции

Для хранения данных вашей программы в памяти нужно использовать переменные. Перед использованием переменной ее нужно объявить. Т.е. указать ее тип и дать ей имя. В Java все переменные делятся на переменные примитивного и ссылочного типа.

К переменным примитивного типа относятся:

- **byte**
- **short**
- **int**
- **long**
- **float**
- **double**
- **char**
- **boolean**

С переменными примитивного типа (числового и символьного) можно проводить основные математические операции.

Для считывания данных с клавиатуры используется Scanner

Литература

- 1) Герберт Шилд — Java 8. Полное руководство. 9-е издание, «Вильямс» — 2015, стр. 75-93
- 2) Кей Хорстман, Гари Корнел — Библиотека профессионала Java. Том 1. Основы. 9-е издание, «Вильямс» — 2014, стр. 57 — 75

Домашнее задание

1) Написать программу которая считывает 5-и значное число с клавиатуры и выводит цифры из которого оно состоит.

Например : Считывается число 54698

Выводиться:

5

4

6

9

8

2) Написать программу которая вычислит и выведет на экран площадь треугольника если известны его стороны.

3) Написать программу которая вычислит и выведет на экран длину окружности, если ее радиус считывается с клавиатуры.

Java Start

(Условные операторы)

Условные операторы используются для того, чтобы разветвлять ход выполнения программы, основываясь на изменениях состояния программы.

Оператор if

```
if (Логическое условие) {  
    Блок операторов которые выполняются в случае  
    когда логическое условие истинно  
}
```

Логическое условие — это сравнение между собой двух операндов (чаще всего переменных и литералов). Логическое условие возвращает один из двух возможных результатов:

- **true** — условие правдиво с точки зрения математики.
- **false** — с точки зрения математики условие ложно.

Если логическое условие вернет true, то блок операторов заключенный в фигурные скобки оператора if выполняться. Если же логическое условие вернет false то они будут пропущены

Логическое условие — один или более операторов сравнения (для переменных числового типа или символьного) или использования переменной типа `boolean`.

Список операторов используемых в логических условиях.

Запись оператора	Описание
<code>==</code>	Равно
<code>!=</code>	Не равно
<code><</code>	Меньше
<code>></code>	Больше
<code><=</code>	Меньше или равно
<code>>=</code>	Больше или равно

Пример использования оператора if

```
package com.gmail.tsa;
```

```
public class Main {
```

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

```
        int price = 450;
```

```
        int money = 500;
```

```
        if (money > price) {
```

```
            System.out.println("I will buy it");
```

```
        }
```

```
    }
```

```
}
```

Логическое условие. Вернет true, так как 500 больше 450

Этот оператор выполнится так логическое условие вернуло true

Оператор if с блоком else

Если возникает необходимость выполнить один блок операторов если условие истинно, и другой блок операторов если оно ложно то можно оператор if дополнить блоком else.

```
if (Логическое условие) {  
    Блок операторов которые выполняются в случае  
    когда логическое условие истинно  
}  
else {  
    Блок операторов которые выполняются в случае  
    когда логическое условие оператора if ложно.  
}
```

Внимание блок else для оператора if может быть только один. Блок else без оператора if не существует.

Пример использования оператора if с блоком else

```
package com.gmail.tsa;  
  
public class Main {  
  
    public static void main(String[] args) {  
  
        int price = 450;  
  
        int money = 300;  
  
        if (money > price) {  
            System.out.println("I will buy it");  
        } else {  
            System.out.println("I'm too poor");  
        }  
    }  
}
```

Логическое условие. Вернет false, так как 300 не больше чем 450

Выполниться этот оператор, так как логическое условие оператора if вернуло false

Блок схемы

Блок-схема — распространенный тип схем (графических моделей), описывающих алгоритмы или процессы, в которых отдельные шаги изображаются в виде блоков различной формы, соединенных между собой линиями, указывающими направление последовательности.



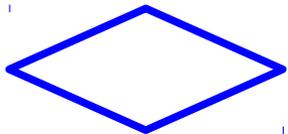
Процесс - символ отображает функцию обработки данных любого вида (выполнение определенной операции или группы операций, приводящее к изменению значения, формы или размещения информации)



Данные - Символ отображает данные, носитель данных не определен.



Предопределенный процесс - Символ отображает предопределенный процесс, состоящий из одной или нескольких операций или шагов программы, которые определены в другом месте (в подпрограмме, модуле). В программировании – вызов процедуры или функции.



Условный оператор (Решение) - Символ отображает решение или функцию переключательного типа, имеющую один вход и ряд альтернативных выходов, один и только один из которых может быть активизирован после вычисления условий, определенных внутри этого символа.



Терминатор - Символ отображает выход во внешнюю среду и вход из внешней среды (начало или конец схемы программы, внешнее использование и источник или пункт назначения данных).

Блок схема условных операторов

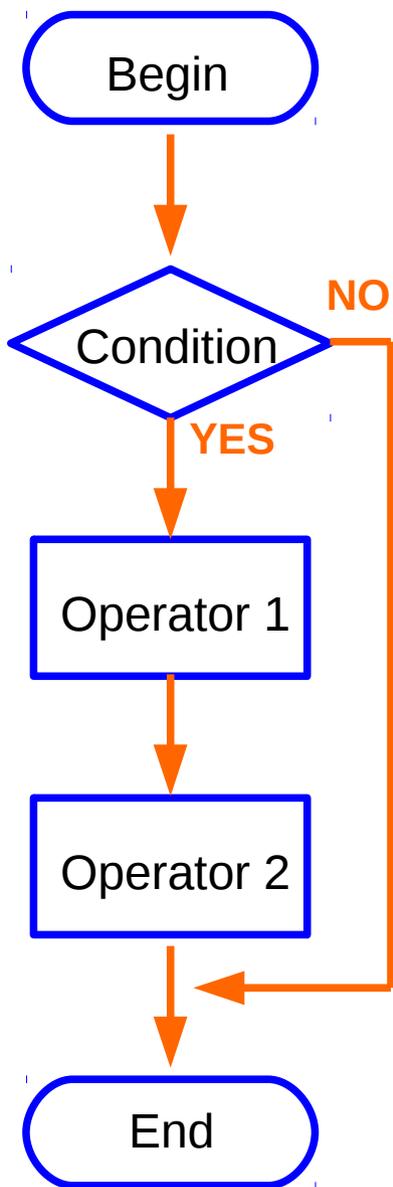


Диаграмма оператора if

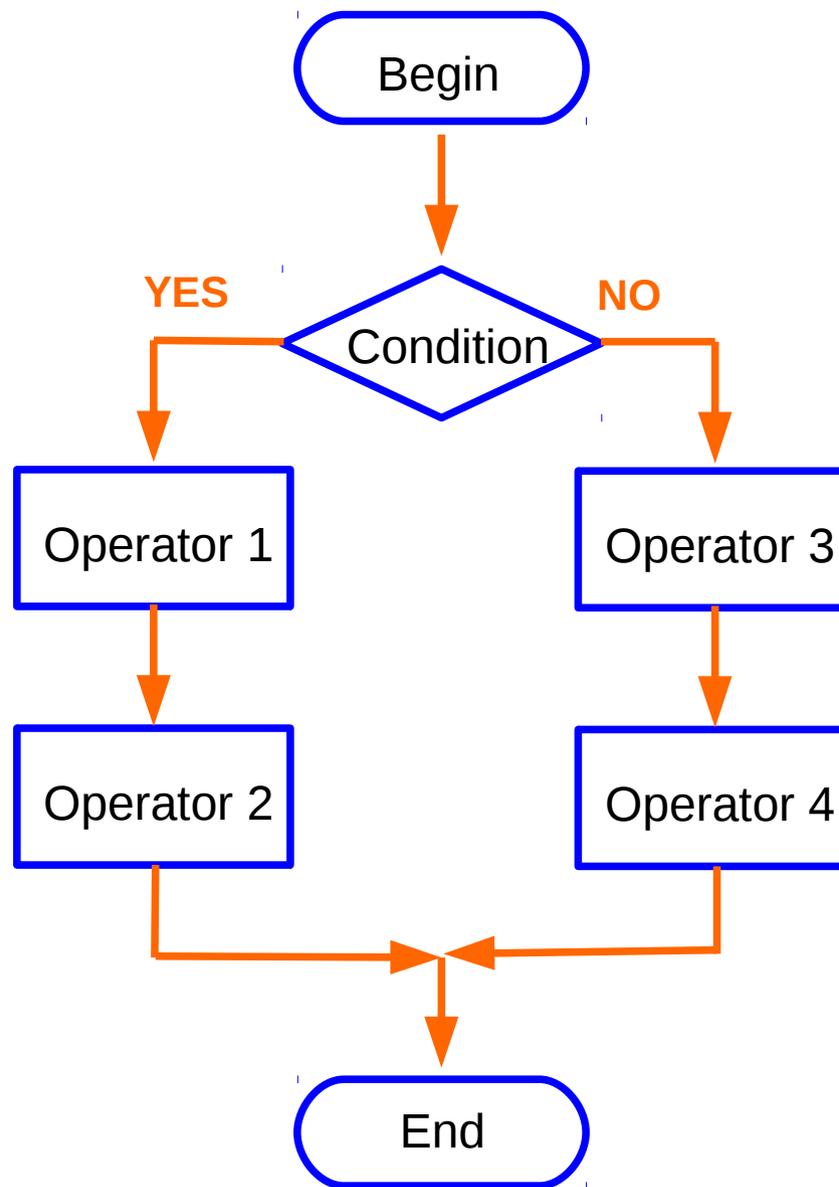


Диаграмма оператора if - else

Тернарный оператор ?:

Для сокращения записи условного оператора применяют тернарный оператор. Его синтаксис - Логическое Условие ? выражение1 : выражение2

Если Логическое Условие равно **true**, то вычисляется **выражение1** и его результат становится результатом выполнения всего оператора. Если же Логическое Условие равно **false**, то вычисляется **выражение2**, и его значение становится результатом работы оператора. Оба операнда выражение1 и выражение2 должны возвращать значение одинакового (или совместимого) типа.

```
package com.gmail.tsa;

public class Main {

    public static void main(String[] args) {

        int a = 10;
        int b = 5;
        int c;

        if (a < b) {
            c = 0;
        } else {
            c = 1;
        }
        System.out.println(c);
    }
}
```

```
package com.gmail.tsa;

public class Main {

    public static void main(String[] args) {

        int a = 10;
        int b = 5;
        int c;

        c = a < b ? 0 : 1;

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

Присвоение в зависимости от условия реализовано обычным условным оператором и с помощью тернарного.

Вложенный операторы if

Вложенный *if* – это оператор *if*, который размещен внутри другого *if* – или *else*-оператора. Вложение *if* – обычный прием в программировании. Когда вы вкладываете *if*, нужно помнить, что оператор *else* всегда относится к самому близкому оператору *if*, который находится в том же блоке, что и *else*.

```
package com.gmail.tsa;

public class Main {

    public static void main(String[] args) {

        int a = 10;
        int b = 5;
        int c;

        if (a < b) {
            c = 0;
        } else {

            if (a > 9) {
                c = 1;
            } else {
                c = 2;
            }

        }

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

← Вложенный условный оператор

Пример на использование вложенного оператора if

```
package lesson3;
import java.util.Scanner;
public class Main {

    public static void main(String[] args) {

        Scanner sc=new Scanner(System.in);
        int a;
        int b;
        System.out.println("Input a");
        a=sc.nextInt();
        System.out.println("Input b");
        b=sc.nextInt();

        if (a<b){
            System.out.println("a<b");
        }
        else {
            if (a>b){
                System.out.println("a>b");
            }
            else {
                System.out.println("a=b");
            }
        }
    }
}
```

Логическое условие внешнего оператора if

Пара if- else содержит вложенный if-else

Пара if -else является вложенной для внешнего if

Логическое условие вложенного оператора if

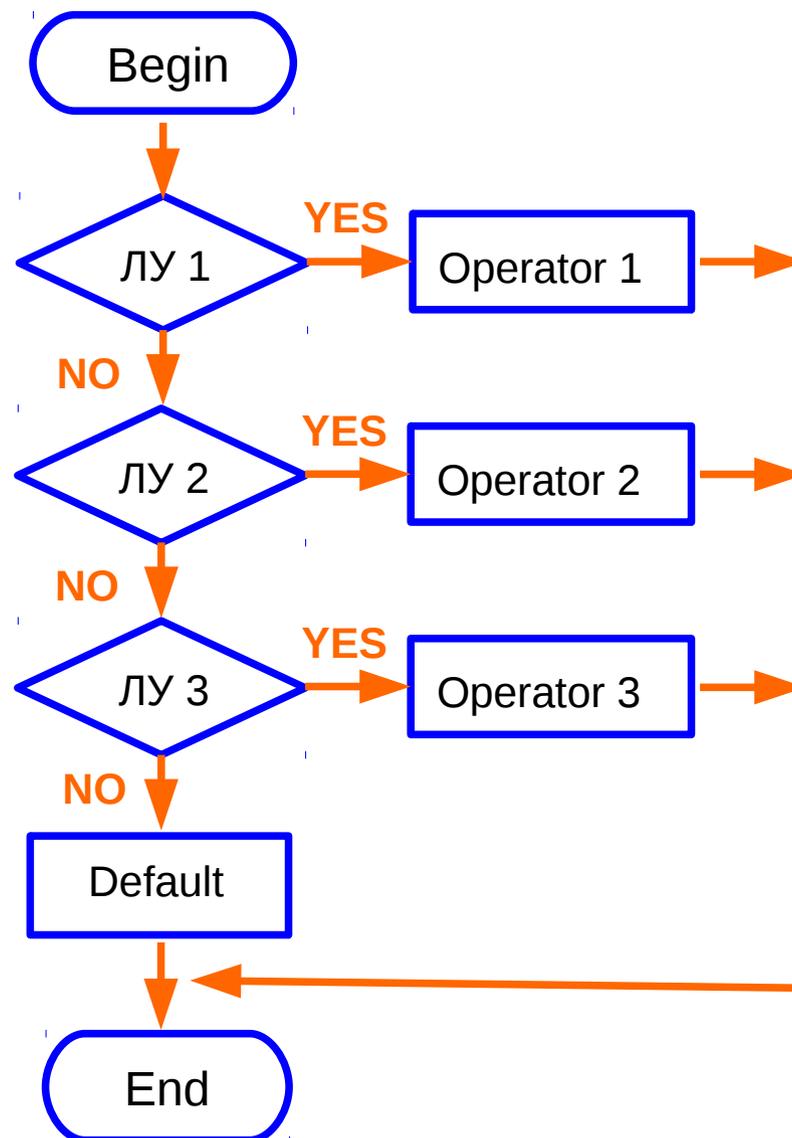
Многозвенный if-else-if

Общую программную конструкцию, которая основана на последовательности вложенных if, называют многозвенным (ladder) if-else-if.

Запись многозвенного if-else -if

```
if(ЛУ 1) {  
  блок операторов 1  
}  
  
else if(ЛУ 2) {  
  блок операторов 2  
}  
  
else if(ЛУ 3) {  
  блок операторов 3  
}  
....  
else {  
  блок операторов N  
}
```

Диаграмма (блок схема) многозвенного if-else -if



Примеры на использование структуры if-else-if

```
package com.gmail.tsa;
```

```
public class Main {
```

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

```
        int a = 10;
```

```
        int c;
```

```
        if (a < 3) {
```

```
            c = 0;
```

```
        } else if (a < 5) {
```

```
            c = 1;
```

```
        } else if (a < 7) {
```

```
            c = 2;
```

```
        } else {
```

```
            c = 3;
```

```
        }
```

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

```
    }
```

```
}
```

← Логическое Условие 1

← Логическое Условие 2

← Логическое Условие 3

← Default

Логические (булевы) операторы

Для объединения нескольких логических условий используются логические (булевы) операторы. Они позволяют объединить несколько условий в одно. Приоритет также можно устанавливать с помощью скобок.

Оператор	Описание
&&	Оператор «И» «AND» - вернет true если первое логическое условие вернет true и второе логическое условие также вернет true
	Оператор «ИЛИ» «OR» - вернет true если хоть одно из логических условий вернет true
^	Оператор «Исключающее ИЛИ» «XOR» работает как OR, за исключением того что если оба условия вернут true оператор вернет false
!	Оператор инверсии. Т.е. то что было true станет false и наоборот.

У операторов && и || есть полный вариант записывается как & и |

Операторы & и | всегда проверяют значение обоих операндов. && и || носят название операторов короткой схемы, так как если результат булевого выражения может быть определен из левого операнда, правый операнд не вычисляется.

Оператор &&

Синтаксис:

if (ЛУ1 && ЛУ2)

Таблица истинности для оператора &&

ЛУ1	true	false	false	true
ЛУ2	false	true	false	true
Результат	false	false	false	true

Оператор && вернет правду если оба логических условия вернут правду

Пример использования оператора &&

```
package com.gmail.tsa;  
  
public class Main {  
  
    public static void main(String[] args) {  
  
        int a = 10;  
        int b = 15;  
        int c;  
  
        if (a > 9 && b > 9) {  
            c = 0;  
        } else {  
            c = 1;  
        }  
  
        System.out.println(c);  
    }  
}
```

Использование оператора && так как и первое условие вернет true (a>9) и второе условие вернет true (b>9) то и результатом будет true

Оператор ||

Синтаксис:

if (ЛУ1 || ЛУ2)

Таблица истинности для оператора ||

ЛУ1	true	false	false	true
ЛУ2	false	true	false	true
Результат	true	true	false	true

Оператор || вернет true если хотя бы одно из условий вернет true

Пример использования оператора ||

```
package com.gmail.tsa;  
  
public class Main {  
  
    public static void main(String[] args) {  
  
        int a = 10;  
        int b = 15;  
        int c;  
  
        if (a > 9 || b > 20) {  
            c = 0;  
        } else {  
            c = 1;  
        }  
  
        System.out.println(c);  
    }  
}
```

Использование оператора || так как и первое условие вернет true (a>9), а второе условие вернет false (b>20) то результатом будет true. Так как в || достаточно одного true;

Оператор \wedge

Синтаксис:

if (ЛУ1 \wedge ЛУ2)

Таблица истинности для оператора \wedge

ЛУ1	true	false	false	true
ЛУ2	false	true	false	true
Результат	true	true	false	false

Оператор \wedge вернет true если хотя бы одно из условий вернет true, но не оба одновременно.

Пример использования оператора ^

```
package com.gmail.tsa;  
  
public class Main {  
  
    public static void main(String[] args) {  
  
        int a = 10;  
  
        int c;  
  
        if ((a > 5) ^ (a > 7)) {  
  
            c = 0;  
  
        } else {  
  
            c = 1;  
  
        }  
        System.out.println(c);  
    }  
}
```

Использование оператора ^ так как и первое условие вернет true ($a > 5$) и второе условие вернет true ($a > 7$) то результатом будет false.

Оператор !

Синтаксис:

If (!ЛУ1)

Таблица истинности для оператора !

ЛУ 1	true	false
Результат	false	true

Оператор ! вернет true если условие вернет false

Пример использования оператора !

```
package com.gmail.tsa;

public class Main {

    public static void main(String[] args) {

        int a = 10;
        int c;

        if (!(a < 9)) {
            c = 0;
        } else {
            c = 1;
        }

        System.out.println(c);

    }

}
```

Использование оператора !. Так как условие вернет false (a<9) то и результатом будет true (этот оператор инвертирует результат).

Оператор switch

Оператор *switch* –это *java* - оператор множественного ветвления.

Общая форма оператора *switch*:

```
switch (expression) {  
    case value1:  
        //последовательность операторов1  
        break;  
    case value2:  
        //последовательность операторов2  
        break;  
    ...  
    case valueN:  
        //последовательность операторов N  
        break;  
    default:  
        //последовательность операторов по умолчанию  
}
```

! expression должно иметь тип `byte`, `short`, `int` или `char`; каждое value, указанное в операторах `case`, должно иметь тип, совместимый с типом выражения. Каждое значение `case` должно быть уникальной константой. Дублирование значений `case` недопустимо.

Работа оператора `switch`

Оператор `switch` работает следующим образом. Значение выражения сравнивается с каждым из указанных значений в `case`-операторах. Если соответствие найдено, то выполняется последовательность операторов, следующая после этого оператора `case`. Если ни одна из `case`-констант не соответствует значению выражения, то выполняется оператор `default`. Однако оператор `default` необязателен. Если согласующихся `case` нет, и `default` не присутствует, то никаких дальнейших действий не выполняется.

Оператор `break` используется внутри `switch`, чтобы закончить последовательность операторов. Когда встречается оператор `break`, выполнение передается к первой строке кода, которая следует за полным оператором `switch`. Он создает эффект досрочного выхода из `switch`.

Пример использования switch

```
package lesson3;

import java.util.Scanner;

public class Main {

    public static void main(String[] args) {
        Scanner sc=new Scanner(System.in);
        int month=sc.nextInt();
        String season;
        switch(month){
            case 12:
            case 1:
            case 2:
                season="Winter";
                break;
            case 3:
            case 4:
            case 5:
                season="Spring";
                break;
            case 6:
            case 7:
            case 8:
                season="Summer";
                break;
            case 9:
            case 10:
            case 11:
                season="Autumn";
                break;
            default:
                season="unknown month";
        }
        System.out.println(month+" month is "+season);
    }
}
```

Переменная которая сравнивается с значениями в операторах case

Блок оператора switch

Действие по умолчанию

Краткие итоги лекции

Для изменения хода программы в зависимости от условий, используются условные операторы.

В Java реализованы следующие условные операторы:

- **if**
- **if** с блоком **else**
- Тернарный оператор (по сути сокращение блока `if — else`)
- **switch**

Также часто используются вложенные в друг друга условные операторы, что дает возможность реализовать более сложную логику работы.

Для объединения нескольких логических условий используются булевы операторы :

- **&&** - AND
- **||** - OR
- **^** - XOR
- **!** - NOT

Существует полная и краткая форма операторов AND и OR.

Литература

- 1) Герберт Шилд — Java 8. Полное руководство. 9-е издание, «Вильямс» — 2015, стр. 118-128
- 2) Кей Хорстман, Гари Корнел — Библиотека профессионала Java. Том 1. Основы. 9-е издание, «Вильямс» — 2014, стр. 97 — 100

Домашнее задание — Уровень 1

- 1) Написать программу которая считает 4 числа с клавиатуры и выведет на экран самое большое из них.
- 2) Есть девятиэтажный дом, в котором 4 подъезда. Номер подъезда начинается с единицы. На одном этаже 4 квартиры. Напишите программу которая получит номер квартиры с клавиатуры, и выведет на экран на каком этаже, какого подъезда расположена эта квартира. Если такой квартиры нет в этом доме то нужно сообщить об этом пользователю.
- 3) Определить количество дней в году, который вводит пользователь. В високосном годе - 366 дней, тогда как в обычном 365. Високосный год определяется по следующему правилу:
Год високосный, если он делится на четыре без остатка, но если он делится на 100 без остатка, это не високосный год. Однако, если он делится без остатка на 400, это високосный год.
- 4) Треугольник существует только тогда, когда сумма любых двух его сторон больше третьей. Дано: a , b , c – стороны предполагаемого треугольника. Напишите программу которая укажет существует такой треугольник или нет.

Домашнее задание - Уровень 2

- 1) Есть круг с центром в начале координат и радиусом 4. Пользователь вводит с клавиатуры координаты точки x и y . Написать программу которая определит лежит ли эта точка внутри круга или нет.
- 2) Дан треугольник координаты вершин $A(0,0)$, $B(4,4)$, $C(6,1)$. Пользователь вводит с клавиатуры координаты точки x и y . Написать программу которая определит лежит ли эта точка внутри треугольника или нет.
- 3) Проверить, является ли четырехзначное число счастливым билетом. Дано четырехзначное число. Проверить, является ли оно «счастливым билетом». Примечание: счастливым билетом называется число, в котором - при четном количестве цифр в числе сумма цифр его левой половины равна сумме цифр его правой половины. Например, рассмотрим число 1322. Его левая половина равна 13, а правая – 22, и оно является счастливым билетом (т. к. $1 + 3 = 2 + 2$)
- 4) С клавиатуры вводится шестизначное число. Проверить, является ли оно палиндромом. Примечание: палиндромом называется число, слово или текст, которые одинакового читаются слева направо и справа налево. Например, это числа 143341, 5555, 7117 и т. д.

Дополнительный материал (Debugging и отладка)

Debugging (отладка) – это этап разработки программы, в ходе которого обнаруживают, локализуют и исправляют баги (ошибки).

Чтобы определить, где находится ошибка, нужно:

- Узнать текущее значение переменных
- Выявить, по какому пути выполнялась программа

Есть два взаимодополняющих метода отладки:

Использование отладчиков – специализированного ПО, которое включает в себя интерфейс пользователя для пошагового выполнения программы с остановками на некоторых строках кода, либо при достижении конкретного условия.

Вывод текущего состояния программы с помощью операторов вывода, расположенных в критических точках. Вывод производится на экран, в файл или на принтер. Вывод в файл еще называют журналированием.

Отладчик – это специализированный инструмент, который позволяет программисту наблюдать за выполнением исследуемой программы, перезапускать ее и останавливать, прогонять в замедленном темпе и т. д.

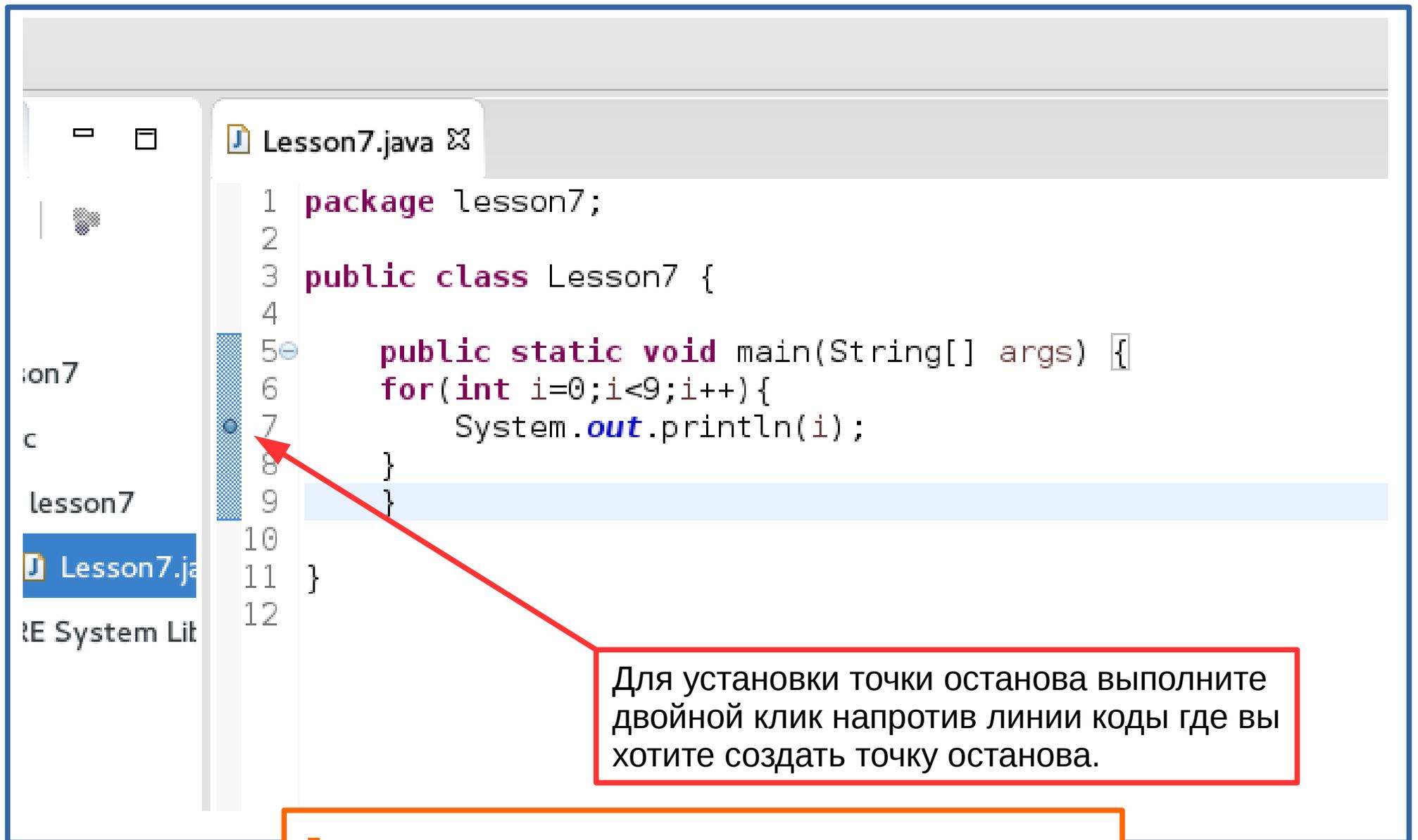
Использование встроенного отладчика в Eclipse

Для использования отладчика установите сначала точки останова в программе.

В программировании, точка останова (англ. breakpoint) — это преднамеренное прерывание выполнения программы, при котором выполняется вызов отладчика (одновременно с этим, программа сама может использовать точки останова для своих нужд). После перехода к отладчику, программист может исследовать состояние программы (логи, состояние памяти, регистров процессора, стека и т. п.), с тем чтобы определить, правильно ли ведет себя программа. После остановки в отладчике, программа может быть завершена либо продолжена с того же места где произошел останов.

На практике, точка останова определяется как одно или несколько условий, при которых происходит прерывание программы. Наиболее часто используется условие останова при переходе управления к указанной инструкции программы (instruction breakpoint). Другое условие останова — операция чтения, записи или изменения указанной ячейки или диапазона ячеек памяти (data breakpoint или watchpoint).

Создание точек останова



```
1 package lesson7;
2
3 public class Lesson7 {
4
5     public static void main(String[] args) {
6         for(int i=0;i<9;i++){
7             System.out.println(i);
8         }
9     }
10
11 }
12
```

Для установки точки останова выполните двойной клик напротив линии кода где вы хотите создать точку останова.

! Внимание точек останова может быть несколько

Для запуска выполнения в режиме отладчика нажмите на иконку или выберите Run-> Debug As -> Java application

tor Navigate Search Project Run Window Help

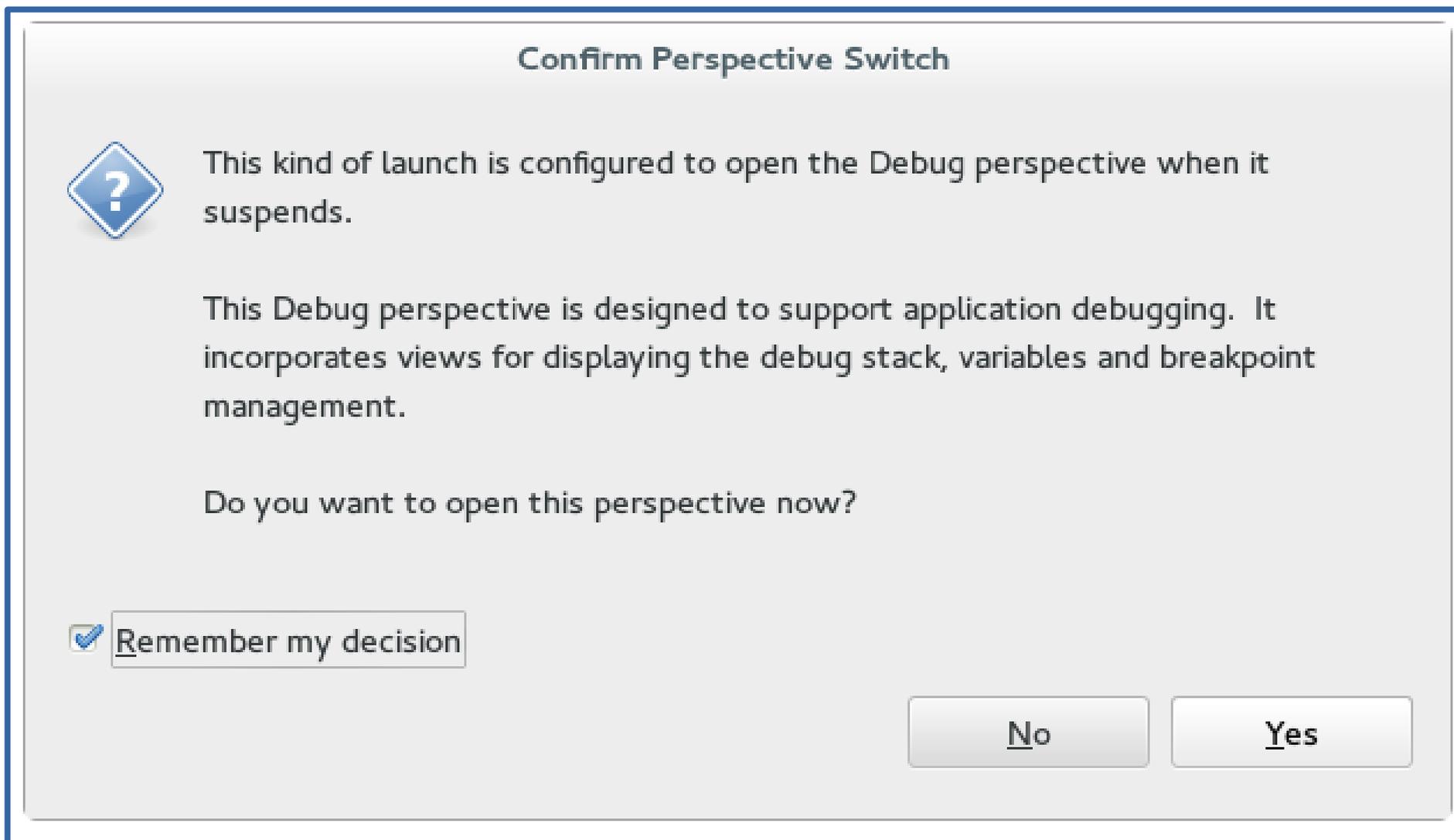


] Lesson7.java ✖

```
1 package lesson7;
2
3 public class Lesson7 {
4
5     public static void main(String[] args) {
6         for(int i=0;i<9;i++){
7             System.out.println(i);
8         }
9     }
10
11 }
12
```

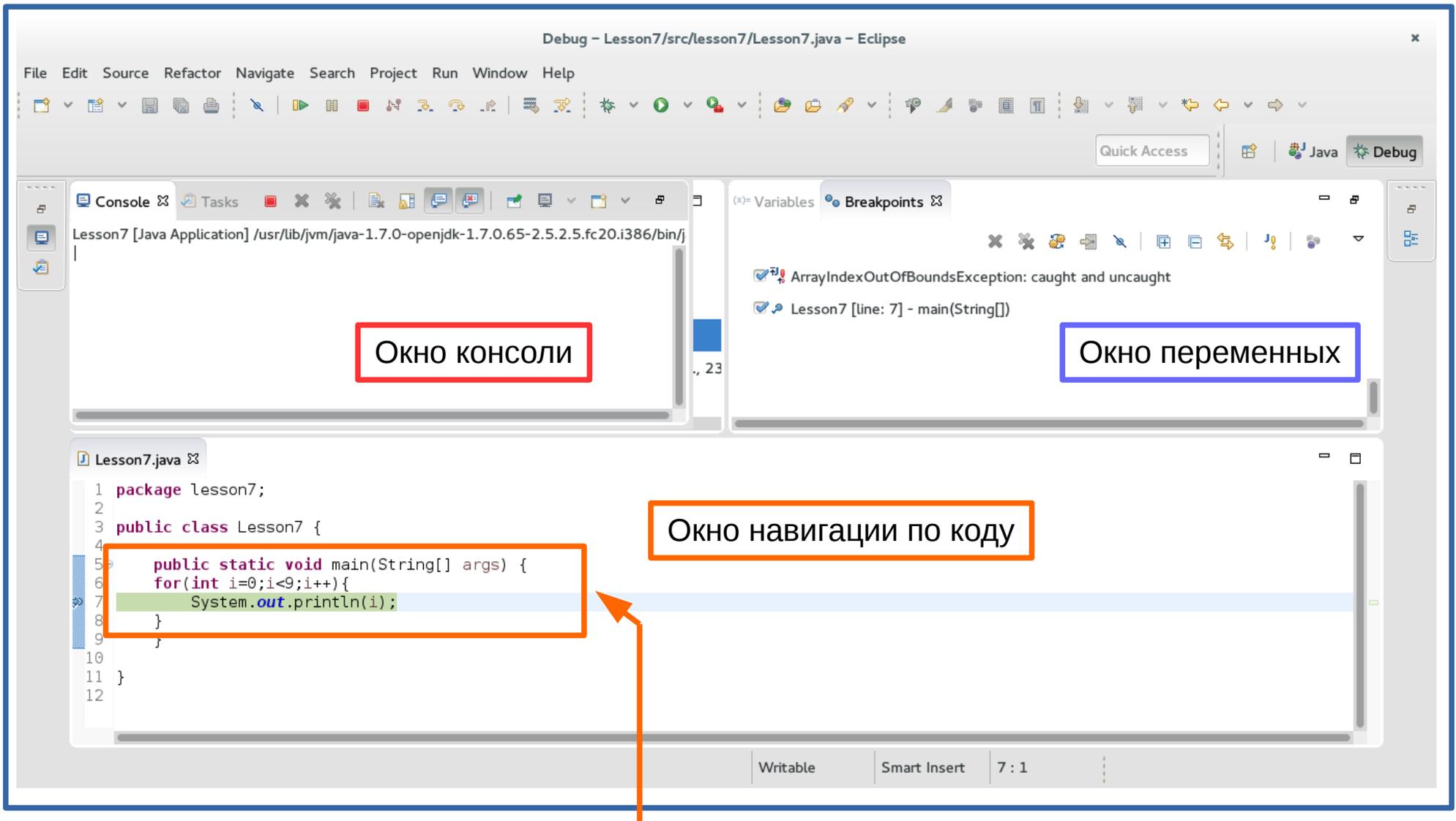
Иконка старта режима отладки

Потом возникнет диалоговое окно которое предложит перевести Eclipse в режиме отладки.



Выбираем YES и попадаем в режим отладки

Вид окна отладки Eclipse



В окне навигации по коду, вы сможете увидеть как в пошаговом режиме выполняется ваша программа. Строка которая на данный момент активна выделяется цветом.

При старте отладки есть возможность просматривать значения переменных

The screenshot displays the Eclipse IDE interface during a debug session. The top menu bar includes File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, and Help. The toolbar contains various icons for file operations and debugging. The main workspace is divided into several panes:

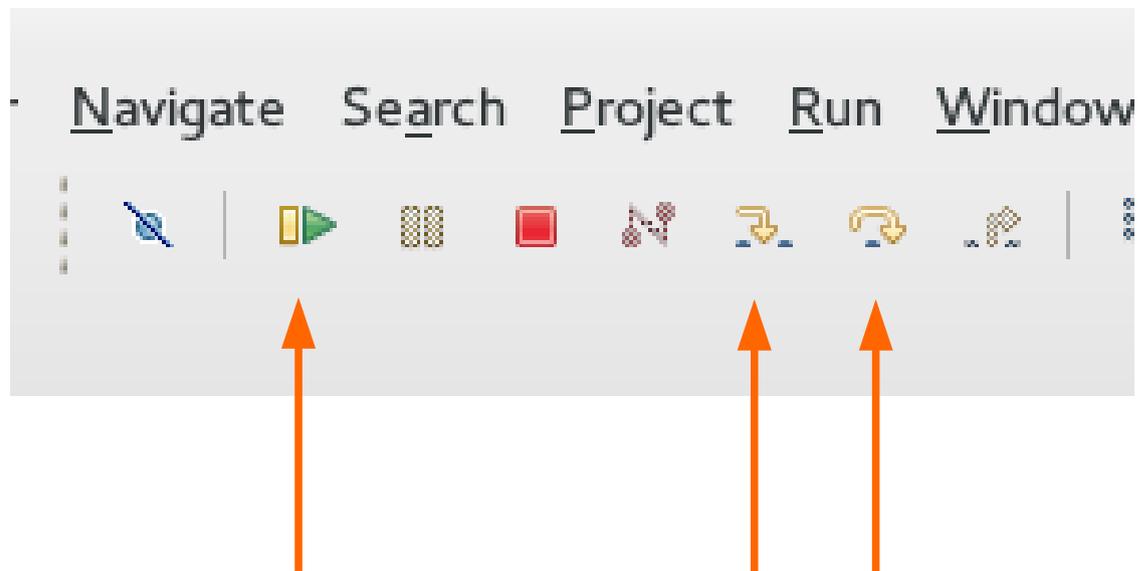
- Debug Console:** Shows the current thread as 'Thread [main] (Suspended (breakpoint at line 7 in Lesson7))' and the current execution point at 'Lesson7.main(String[]) line: 7'.
- Variables View:** A table showing the current state of variables:

Name	Value
args	String[0] (id=15)
k	3
i	2
- Code Editor:** Shows the source code of 'Lesson7.java'. The current line of execution is line 7, which is the start of a for loop:

```
for(int i=0;i<9;i++){
```

! Выполнив щелчок правой кнопкой мыши по переменной и выбрав пункт меню Change Value.. можно изменить значение некоторых переменных.

Пошаговое выполнение кода



Иконки для пошагового выполнения операторов

Использование директивы assert

Директива `assert` — вызывает преднамеренное завершение программы, при выполнении определенного логического условия.

Применяется для поиска логических ошибок на этапе проектирования ПО.

! Для применения этой директивы сначала нужно установить параметр `-ea` для виртуальной машины Java.

Добавление параметров VM в Eclipse

Для установки конфигурации виртуальной машины выполните такую последовательность действий
Run - > Run configuration

В появившемся диалоговом окне перейдите на вкладку (x)=Arguments в поле VM arguments введите -ea

Установка ургументов виртуальной машины

The image shows a configuration window for a virtual machine in an IDE. The window title is "Lesson1 (1)". The "Arguments" tab is selected, and the "VM arguments" field contains the text "-ea".

Вкладка для установки аргументов

Установка аргумента виртуальной машины

Apply Revert

Объявление директивы assert

assert логическое условие

assert логическое условие:"Текст сообщения"

Как только логическое условие станет false — программа экстренно завершит свою работу

```
package com.gmail.tsa;
```

Пример использования директивы assert

```
public class Main {
```

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

```
        for(int i=0;i<10;i++){
```

```
            assert i<=5:"ERROR";
```

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

```
        }
```

```
    }
```

```
}
```

Краткие итоги Лекции

Отладчик инструмент для поиска ошибок в программе. Он не является частью инфраструктуры Java и поставляется средой разработки. Позволяет выполнить программу пошагово, в каждый момент видеть значение переменных и какая строка кода является активной.

Работа с отладчиком начинается с установки точки останова. После ее установки войти в режим отладки можно Run → Debug As Java Application.

Java Start

ЦИКЛЫ

Цикл — разновидность управляющей конструкции в высокоуровневых языках программирования, предназначенная для организации многократного исполнения набора инструкций.

Разработал: Цымбалюк А.Н.

Операторы цикла.

Цикл — разновидность управляющей конструкции в высокоуровневых языках программирования, предназначенная для организации многократного исполнения набора инструкций. Также циклом может называться любая многократно исполняемая последовательность инструкций, организованная любым способом (например, с помощью условного перехода).

Последовательность инструкций, предназначенная для многократного исполнения, называется **телом цикла**. Единичное выполнение тела цикла называется **итерацией**. **Выражение** определяющее, будет в очередной раз выполняться итерация, или цикл завершится, называется условием выхода или условием окончания цикла (либо условием продолжения в зависимости от того, как интерпретируется его истинность — как признак необходимости завершения или продолжения цикла). Переменная, хранящая текущий номер итерации, называется **счётчиком итераций цикла** или просто **счётчиком** цикла. Цикл не обязательно содержит счётчик, счётчик не обязан быть один — условие выхода из цикла может зависеть от нескольких изменяемых в цикле переменных, а может определяться внешними условиями (например, наступлением определённого времени), в последнем случае счётчик может вообще не понадобиться.

Виды циклов

В информатике используются следующие разновидности операторов цикла

- **Цикл с предусловием** — цикл, который выполняется пока истинно некоторое условие, указанное перед его началом. Это условие проверяется до выполнения тела цикла, поэтому тело может быть не выполнено ни разу (если условие с самого начала ложно).
- **Цикл с постусловием** — цикл, в котором условие проверяется после выполнения тела цикла. Отсюда следует, что тело всегда выполняется хотя бы один раз.
- **Цикл со счётчиком** — цикл, в котором некоторая переменная изменяет своё значение от заданного начального значения до конечного значения с некоторым шагом, и для каждого значения этой переменной тело цикла выполняется один раз.

Язык Java реализует все указанные выше разновидности циклов

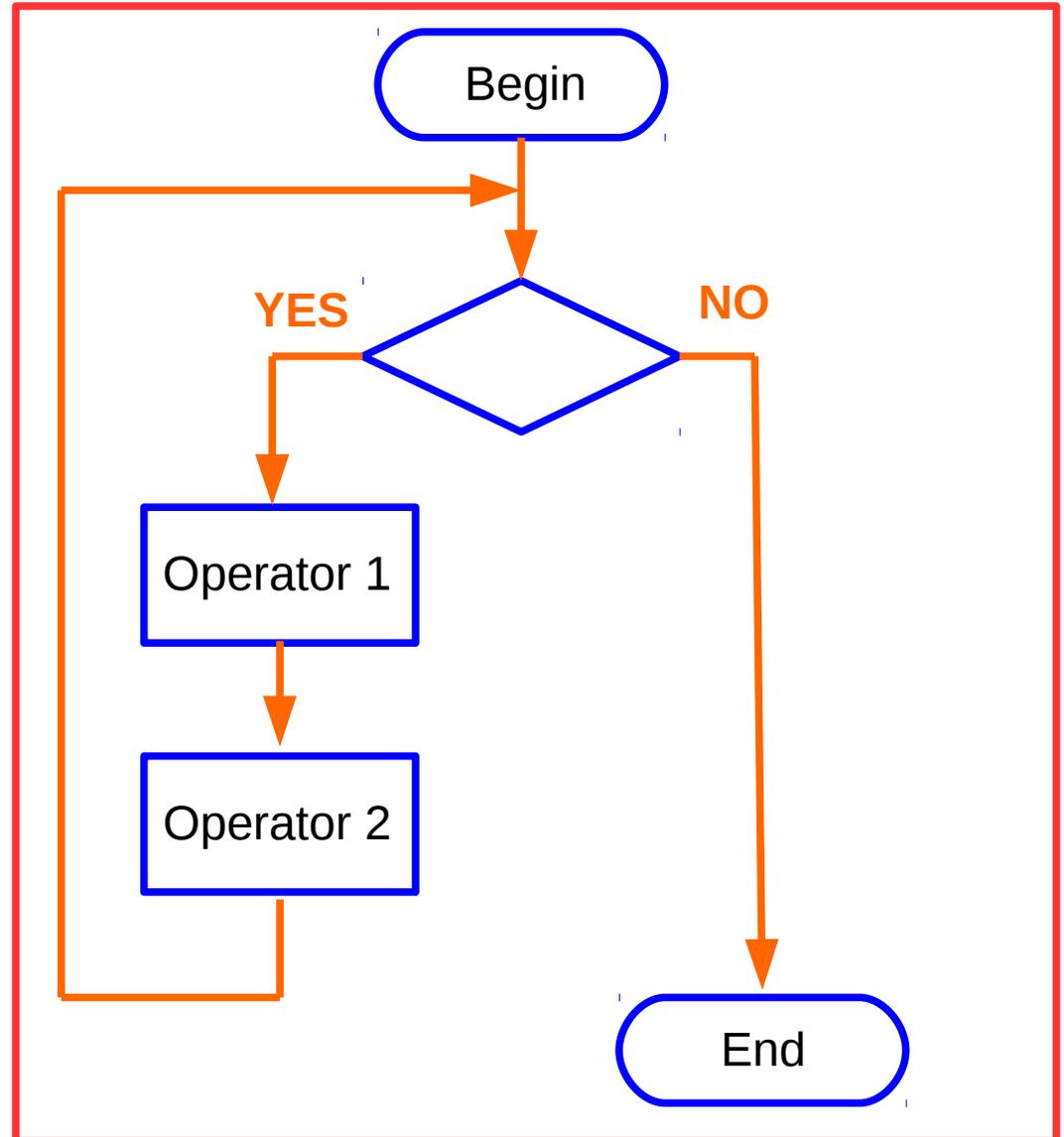
Цикл с предусловием - while

сначала проверяется условие потом выполняется блок операторов

Структура записи в Java

```
while (Логическое условие){  
    Блок операторов  
}
```

Алгоритм выполнения цикла while



Пример цикла while

```
package com.gmail.tsa;  
  
import java.util.Scanner;  
  
public class Main {  
  
    public static void main(String[] args) {  
        Scanner sc=new Scanner(System.in);  
        int n=sc.nextInt();
```

Логическое условие
Продолжения цикла

```
while(n>0){
```

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

```
    n=n-1;
```

```
}
```

Тело цикла

Цикл while

! Переменная связанная с Логическим Условием обязательно должна изменяться в теле цикла. В противном случае выполнение тела цикла будет бесконечным.

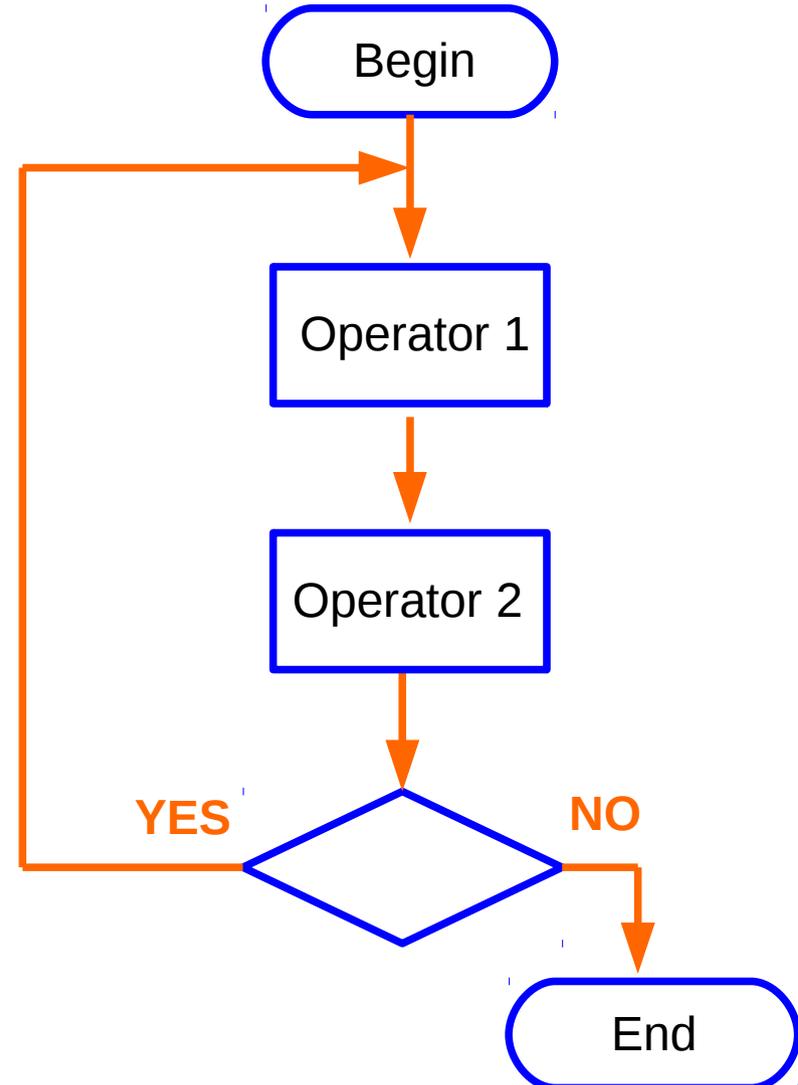
Цикл с постусловием do — while

сначала выполняется блок операторов потом проверяется условие

Структура записи в Java

```
do{  
    Блок операторов  
}while (Логическое условие)
```

Алгоритм выполнения цикла do-while



Пример цикла do - while

```
package com.gmail.tsa;  
  
import java.util.Scanner;  
  
public class Main {  
  
    public static void main(String[] args) {  
        Scanner sc=new Scanner(System.in);  
        int n=sc.nextInt();
```

```
do{
```

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

```
    n=n-1;
```

Тело цикла

```
}while(n>0);
```

Логическое условие продолжения цикла

Цикл do- while

! Переменная связанная с Логическим Условием обязательно должна изменяться в теле цикла. В противном случае выполнение тела цикла будет бесконечным.

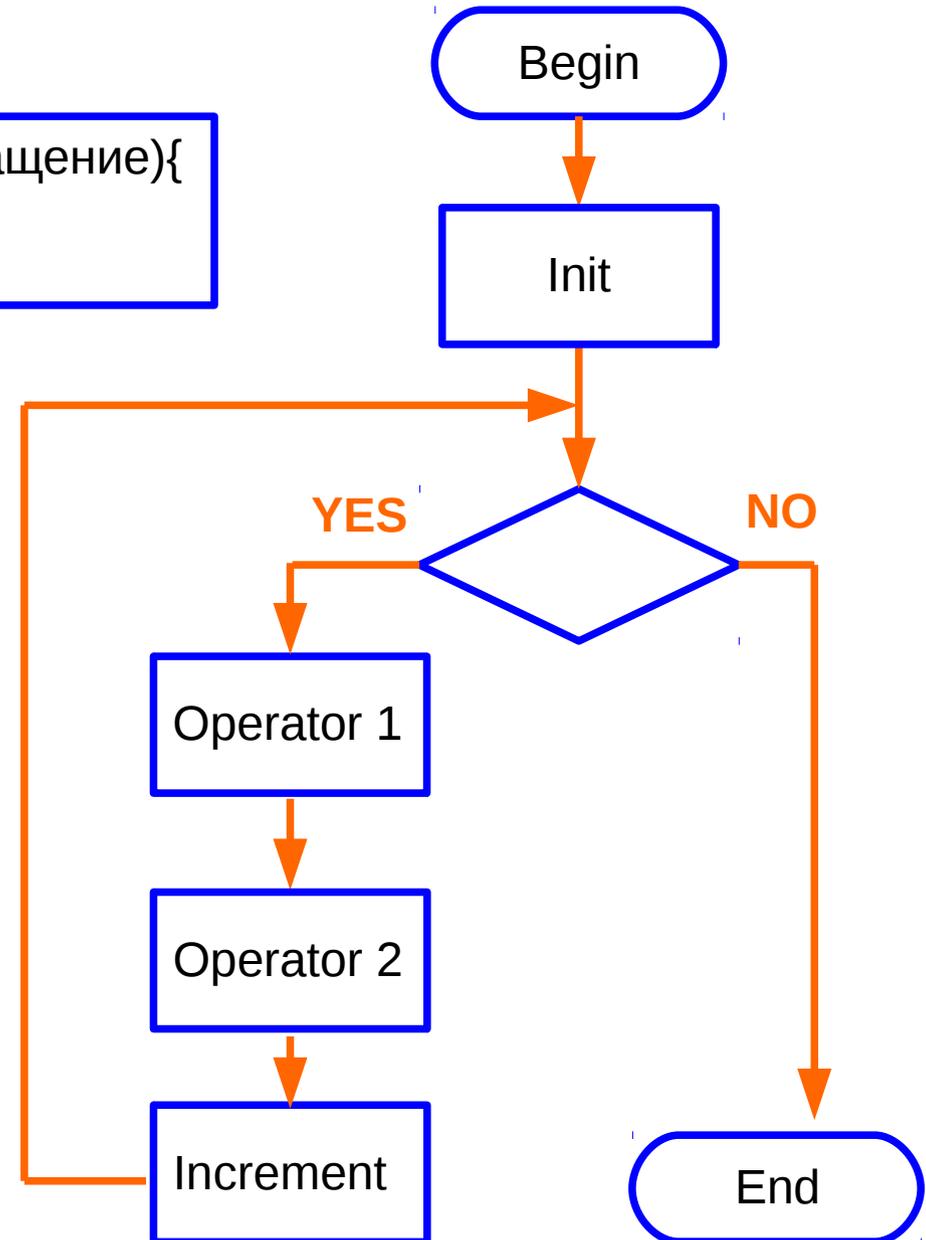
Цикл со счётчиком for

количество итераций заведомо известно

Структура записи в Java

```
for(инициализация; Логическое Условие; приращение){  
    блок операторов  
}
```

Алгоритм выполнения
цикла for



Пример цикла for

```
package com.gmail.tsa;  
  
import java.util.Scanner;  
  
public class Main {  
  
    public static void main(String[] args) {  
        Scanner sc=new Scanner(System.in);  
        int n=sc.nextInt();
```

Инициализация

Логическое условие
окончания цикла

Приращение переменной
цикла

```
for(int i=0;i<n;i++){
```

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

```
}
```

Тело цикла

Цикл for

```
}  
}
```

Цикл for — более подробный взгляд.

Не обязательно объявлять переменную цикла

Приращения может и не быть.
Также может выполняться 2 и более действия

`for(; ;)`

Условие не обязательно связано с переменной цикла. Или вообще отсутствует.

! Строго говоря цикл for — может легко заменить цикл while и do-while

Пример использования цикла for в качестве цикла while

```
package com.gmail.tsa;  
  
import java.util.Scanner;  
  
public class Main {  
  
    public static void main(String[] args) {  
        Scanner sc=new Scanner(System.in);  
        int n=sc.nextInt();  
        int i=0;  
        for( ;i<n; ){  
  
            System.out.println(i);  
            i++;  
  
        }  
  
    }  
}
```

Осталось только условие продолжения цикла.
Т.е. теперь это цикл while

Вложенные циклы

Как и в случае вложенных условных операторов, циклы также могут быть вложены друг в друга. Наибольшее распространение получили вложенные циклы for. В основном потому что именно они используются для прохода по двумерным массивам.

```
for(инициализация; Логическое Условие; приращение){  
  операторы  
  ....  
  for(инициализация; Логическое Условие; приращение){  
    блок операторов  
  }  
  ....  
}
```

↑
Внутренний цикл

↑
Внешний цикл

Следует помнить правило — за одну итерацию внешнего цикла, внутренний пройдет все свои итерации.

Пример вложенных циклов for (рисуем прямоугольник)

```
package com.gmail.tsa;

import java.util.Scanner;

public class Main {

    public static void main(String[] args) {

        Scanner sc=new Scanner(System.in);
        int n=sc.nextInt();
        for(int i=0;i<n;i++){
            for(int j=0;j<n;j++){
                System.out.print("*");
            }
            System.out.println();
        }
    }
}
```

← Внутренний цикл

↑ Внешний цикл

Операторы прерывания цикла

Иногда необходимо досрочно прекратить выполнение итерации цикла или закончить цикл совсем. Для этого используют операторы:

- **continue** – прекращает итерацию цикла
- **break** – завершает цикл

! Внимание оператор **break** прерывает только цикл из которого он был вызван.

Пример использования операторов continue и break

```
package com.gmail.tsa;
public class Main {

    public static void main(String[] args) {
        int n=10;
        for(int i=0;i<n;i++){

            if (i==3){
                continue;
            }
            if(i==7){
                break;
            }
            System.out.println(i);
        }
    }
}
```

← 3-я итерация пропускается

← На 7-й цикл прекратиться

Результат работы программы →

0
1
2
4
5
6

Область действия переменных объявленных в цикле

Переменная объявленная внутри цикла видна только в теле цикла. Видимость переменной означает возможность обращения к ней по имени с целью записи и считывания ее значения.

```
package com.gmail.tsa;
```

```
public class Main {
```

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

```
        int sum = 0;
```

```
        for (int i = 0; i < 5; i++) {
```

```
            int j = i;
```

```
        }
```

```
        sum = sum + j;
```

```
    }
```

```
}
```

← Эта переменная видна только внутри цикла

← В этой строке ошибка, т. к. переменная j — объявлена в цикле

Внимание данный пример не скомпилируется!!!!

Краткие итоги лекции

Для обеспечения возможности многократного повторения операторов используется операторы цикла. Циклы делятся по типам:

- Цикл с предусловием
- Цикл с постусловием
- Цикл со счетчиком

В Java реализованы все три вида циклов:

- Цикл с предусловием — цикл **while**
- Цикл с постусловием — цикл **do-while**
- Цикл со счетчиком — цикл **for**

Как и в случае с условными операторами возможна вложенность циклических операторов. В этом случае внутренний цикл проведет все свои итерации, за одну итерацию внешнего.

Переменные объявленные внутри цикла, имеют область видимости тело цикла.

Для досрочного прерывания цикла существуют операторы:

- **continue** — прерывает текущую итерацию
- **break** — прерывает цикл

Литература

- 1) Герберт Шилд — Java 8. Полное руководство. 9-е издание, «Вильямс» — 2015, стр. 133-153
- 2) Кей Хорстман, Гари Корнел — Библиотека профессионала Java. Том 1. Основы. 9-е издание, «Вильямс» — 2014, стр. 100 — 108

Домашнее задание — Уровень 1

- 1) С помощью циклов нарисовать «обои». Причем количество полос должно вводиться с клавиатуры. В примере 7 полос.

```
***+++***+++***+++***
***+++***+++***+++***
***+++***+++***+++***
***+++***+++***+++***
***+++***+++***+++***
```

- 2) Вычислить с помощью цикла факториал числа - n введенного с клавиатуры ($4 < n < 16$). Факториал числа это произведение всех чисел от этого числа до 1. Например $5! = 5 * 4 * 3 * 2 * 1 = 120$
- 3) Напечатайте таблицу умножения на 5. предпочтительно печатать $1 \times 5 = 5$, $2 \times 5 = 10$, а не просто 5, 10 и т. д.
- 4) Выведите на экран прямоугольник из *. Причем высота и ширина прямоугольника вводятся с клавиатуры. Например ниже представлен прямоугольник с высотой 4 и шириной 5.

```
*****
*      *
*      *
*****
```

Домашнее задание - Уровень 2

1) С помощью цикла (только одного) нарисовать такую фигуру. Причем максимальная высота этой фигуры вводится с клавиатуры (в примере максимальная высота - 4)

```
*  
**  
***  
****  
***  
**  
*
```

2) С помощью циклов вывести на экран все простые числа от 1 до 100. Простое число — число которое делиться нацело только на единицу или само на себя. Первые простые числа это — 2,3,5,7...

3) Выведите на экран «песочные часы» максимальная ширина которых считывается с клавиатуры (число нечетное). В примере ширина равна 5.

```
*****  
  ***  
   *  
  ***  
*****
```

Java Start

(Строки и массивы)

Массив — это конечная последовательность упорядоченных элементов одного типа, доступ к каждому элементу в которой осуществляется по его индексу.

Размер или длина массива — это общее количество элементов в массиве. Размер массива задается при создании массива и не может быть изменен в дальнейшем, т. е. нельзя убрать элементы из массива или добавить их туда, но можно в существующие элементы присвоить новые значения.

Массивы в Java

В Java **массивы** являются **объектами**. Это значит, что имя, которое дается каждому массиву, лишь указывает на адрес какого-то фрагмента данных в памяти. Кроме адреса в этой переменной ничего не хранится. Индекс массива, фактически, указывает на то, насколько надо отступить от начального элемента массива в памяти, чтоб добраться до нужного элемента.

Индекс начального элемента — 0, следующего за ним — 1 и т. д.
Индекс последнего элемента в массиве — на единицу меньше, чем размер массива.

Минимально возможное количество индексов необходимое для точного указания на элемент массива называется размерностью.

Объявление и инициализация массивов

Объявление

Тип данных которые хранятся в массиве **ИМЯ** []

Тип данных которые хранятся в массиве [] **ИМЯ**

Инициализация

ИМЯ=new Тип данных которые хранятся в массиве[размер]

ИМЯ={значение 1, значение 2,...}

! Объявление можно объединить с инициализацией

Примеры

Массив из 10 целых чисел .

```
int[] a=new int[10];
```

Объявление и инициализация массива вещественных чисел.

```
double [] b={1,2,3,4,5.5};
```

! Для определения длины массива используется функция `length` (сначала указывается имя массива потом «.» и `length`.
Например - `a.length`

! Если при создании массива используется оператор `new` то массив заполняется значениями по умолчанию. Для числовых типов — 0. Для логического — `false` . Для символьного - `'\u0000'`. Для строкового — `null`.

О внутреннем устройстве массивов

Массив состоит из двух частей — ссылки и данных хранящихся в ОЗУ

`int [] array` ← ссылка на целочисленный массив

`{1,2,3,4,5}` ← Данные массива



Т.е. ссылка только указывает на тип данных массива и где они лежат, но при этом сама она данными массива не является.

Как обратиться к отдельному элементу массива

Чтобы обратиться к какому-то из элементов массива для того, чтобы прочитать или изменить его значение, нужно указать имя массива и за ним индекс элемента в квадратных скобках. Элемент массива с конкретным индексом ведет себя также, как переменная.

Например:

```
System.out.println(b[1]);
```

```
System.out.println(b[b.length-1]);
```

! Внимание если обратиться к несуществующему элементу массива возникнет ошибка выполнения.

Циклы основной инструмент при работе с массивами

```
package com.gmail.tsa;

public class Main {

    public static void main(String[] args) {

        int[] mas=new int[10]; ← Объявление массива 10 целых чисел

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

            mas[i]=i; ← Присвоение значения элементу массива

            System.out.print(mas[i]+" "); ← Вывод элемента массива на экран

        }

    }

}
```

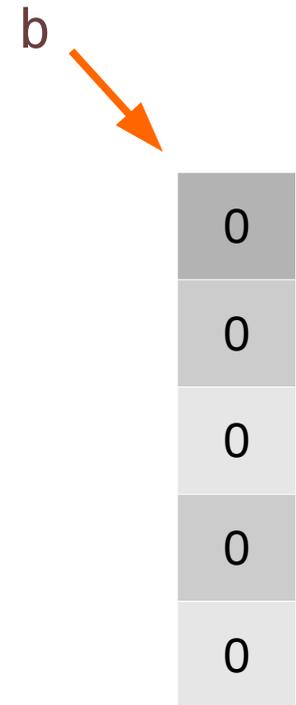
Внимание попытка подставить массив в качестве аргумента метода System.out.println() приведет к выводу на экран его хеш-кода, а не данных массива.

Массивы ссылочный объект данных

Если присвоить одной ссылке на массив, значение другой ссылки то получим две ссылки на один и тот же массив.

```
int [] a = {1,2,3,4,5};
```

```
int [] b = new int [5];
```



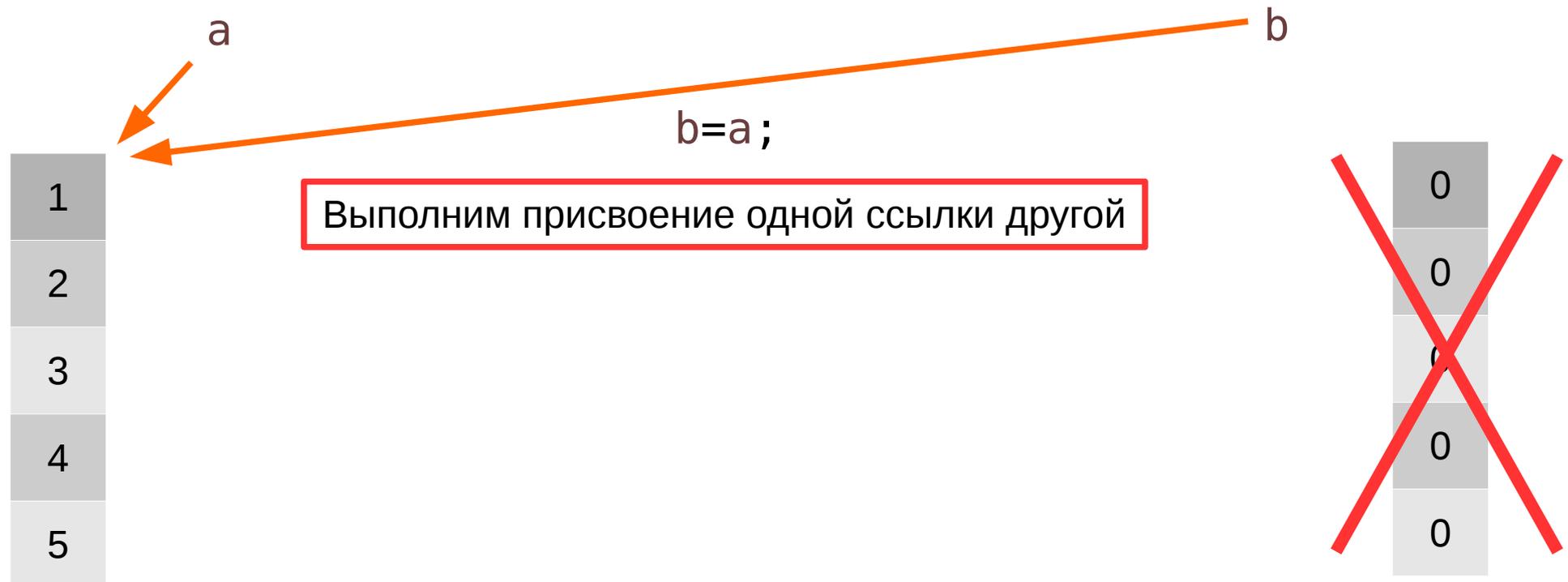
При таком объявлении каждая ссылка указывает на свои данные.

Массивы ссылочный объект данных

Если присвоить одной ссылке на массив, значение другой ссылки то получим две ссылки на один и тот же массив.

```
int [] a = {1,2,3,4,5};
```

```
int [] b = new int [5];
```



Теперь обе ссылки указывают на одни и те же данные. А данные массива b будут удалены сборщиком мусора. Теперь если изменить данные по ссылке b это автоматически будет означать изменение данных для массива a.

Пример присвоения ссылок на массивы

```
package com.gmail.tsa;
```

```
public class Main {
```

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

```
        int[] a = { 1, 2, 3, 4, 5 };
```

```
        int[] b = new int[5];
```

```
        b = a;
```

← Теперь у двух ссылок одинаковый адресат

```
        b[0] = 10;
```

← Изменяем данные по ссылке b

```
        System.out.println(a[0]);
```

← Это же нашло отображение по ссылке a

```
    }
```

```
}
```

Копирование массивов

Копировать массивы можно с помощью поэлементного копирования

```
package com.gmail.tsa;  
  
public class Main {  
  
    public static void main(String[] args) {  
  
        int[] a={3,7,8,1,2,78,33,45,8,10};  
  
        int[] b=new int[10];  
  
        for(int i=0;i<a.length;i++){  
            b[i]=a[i];  
        }  
    }  
}
```

Присваивает i — му элементу b , значение i — го элемента a

Либо с помощью встроенной функции `arraycopy`

Формат вызова

`System.arraycopy`(какой массив копируется, начальный индекс с которого начинается копирование, в какой массив копируется, начальный индекс вставки, длина скопированного участка массива)

```
int[] a={3,7,8,1,2,78,33,45,8,10};  
int[] b=new int[10];  
System.arraycopy(a, 0, b, 0, a.length);
```

Длина скопированного
Участка равна длине a

Какой массив копируется

С какого индекса начинается вставка

В какой массив копируется

С какого индекса начинается копирование

Дополнительные функции для работы с массивами

Для вызова необходимо импортировать — `import java.util.Arrays`

Копирование:

`Arrays.copyOfRange`(какой массив копируется, начальный индекс, конечный индекс)

Копирование и увеличение размера

`Arrays.copyOf`(какой массив копируется, новый размер)

Заполнение:

`Arrays.fill`(какой массив заполняется, чем заполняется)

Сортировка:

`Arrays.sort`(какой массив сортируют)

`Arrays.sort`(какой массив сортируют, начальный индекс, конечный индекс)

Преобразование:

`Arrays.toString`(какой массив преобразуется)

Пример использования функция для работы с массивами

```
package com.gmail.tsa;
```

```
import java.util.Arrays;
```

```
public class Main {
```

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

```
        int[] a = { 0, 2, 5, 9, 14 };
```

```
        int[] b = Arrays.copyOfRange(a, 0, 3);
```

В массив b скопированы
элементы из a с 0 по 3

```
        System.out.println(Arrays.toString(b));
```

```
        int[] c = Arrays.copyOf(b, 7);
```

Массив c скопировал b и имеет размер 7

```
        System.out.println(Arrays.toString(c));
```

```
        Arrays.sort(c);
```

Сортировка массива c

```
        System.out.println(Arrays.toString(c));
```

```
        Arrays.fill(c, 7);
```

Заполнение массива c числом 7

```
        System.out.println(Arrays.toString(c));
```

Создание строки на основе массива c и вывод ее на экран

```
    }
```

```
}
```

Специальный вид цикла для работы с массивами

Цикл - foreach

Объявление:
for(переменная: массив)

Переменная принимает последовательно значение элементов массива

```
package com.gmail.tsa;  
  
public class Main {  
    public static void main(String[] args) {  
        int[] a = { 0, 2, 5, 9, 14 };  
        for (int i : a) {  
            System.out.print(i + " ");  
        }  
    }  
}
```

Цикл for-each. Переменная i по очереди примет значения элементов массива

Вывод на экран: 0 2 5 9 14 .

Если в цикле изменить переменную цикла, то на данные массива это не повлияет.

Двухмерные массивы

Объявление:

```
Тип [][] имя = new Тип [строки][столбцы]
```

Инициализация

```
int [][] a={{1,2,3},{4,5,6},{7,8,9}}
```

Примером двухмерного массива является матрица (или морской бой ☺)

Первый элемент это номер строки, второй элемент — номер столбца соответственно

Пример объявления, заполнения и вывода на экран 2-х мерного массива

```
package com.gmail.tsa;
```

```
public class Main {
```

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

```
        int [][] b=new int[4][5];
```

Объявление

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

Цикл по строкам

```
            for(int j=0;j<b[0].length;j++){
```

Цикл по столбцам

```
                b[i][j]=i+j;
```

Заполнение значениями

```
                System.out.print(b[i][j]+" ");
```

Вывод на экран

```
            }
```

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

```
    }
```

```
}
```

```
}
```

Как устроены многомерные массивы в Java

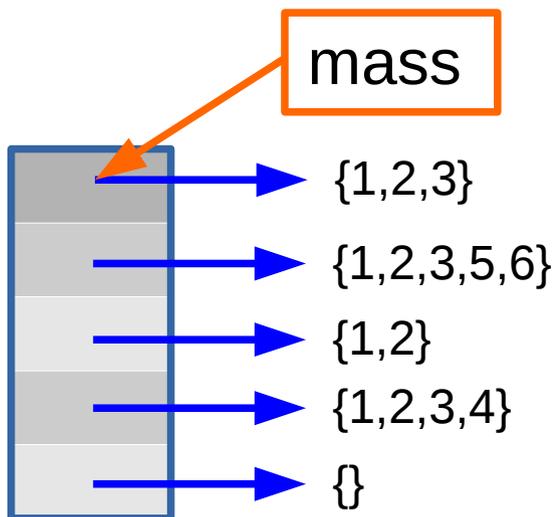
```
int [][] mass
```

Объявление «двухмерного» массива типа int.

Объявление ссылки на одномерный массив

Тип данных хранящихся в массиве — ссылки на одномерные массивы типа int.

Т.е. двухмерный массив — это массив массивов.



Рваный двухмерный массив типа int

Создание «рваных» массивов

При создании 2-х мерных массивов при создании достаточно объявить первый размер, а второй размер объявлять динамически

```
package com.gmail.tsa;

public class Main {

    public static void main(String[] args){

        int [][] b=new int[8][];
        for(int i=0;i<b.length;i++){
            b[i]=new int[i];
            for(int j=0;j<b[i].length;j++){
                b[i][j]=i+j;
                System.out.print(b[i][j]+" ");
            }
            System.out.println();
        }
    }
}
```

Вывод программы

```
1
2 3
3 4 5
4 5 6 7
5 6 7 8 9
6 7 8 9 10 11
7 8 9 10 11 12 13
```

Строки

Объявление строк
`String имя = "Текст";`

Конкатенация строк — объединение строк с помощью операции `+`. Каждая строка дописывается вправо.

Пример

```
String s=" Java"+" the"+" best"
```

! Конкатенация строк возможна с числовыми типами переменных.

Методы для работы со строками

Выделение символов

getChars(int начало источника, int конец, char[] цель, int начало цели)
charAt(индекс символа)
toCharArray()

Сравнение строк

String1.equals(String2)

String1.equalsIgnoreCase(String2) — без учета регистра

String1.compareTo(String2) — возвращает положительное или отрицательное число в зависимости от результата сравнения строк

Поиск

indexOf(символ) — ищет индекс первого вхождения символа

lastIndexOf(подстрока) — ищет индекс последнего вхождения подстроки

Выделение части строк

substring(начальный индекс, конечный индекс)

Примеры работы со строками

```
package com.gmail.tsa;

import java.util.Arrays;

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

        String text = "Hello";
        String text1 = "World";

        char[] t = text.toCharArray();
        System.out.println(Arrays.toString(t));

        char l = text.charAt(0);
        System.out.println(l);

        System.out.println(text.equals(text1));

        int i = text.indexOf("ll");

        String subtext = text.substring(0, 3);
        System.out.println(subtext);
    }
}
```

Создаем 2 строки

Получаем массив символов из строки

Получаем 0-символ из строки

Ищем строку «ll» в строке и получаем индекс

Вырезаем кусок строки

Метод split для работы со строками

Используется для разбиения строки на части на основе разделителей. В результате получается массив строк.

```
package com.gmail.tsa;

import java.util.Arrays;

public class Main {
    public static void main(String[] args) {
        String text = "123,345,657";
        String[] t = text.split(",");

        for (String temp : t) {
            System.out.println(temp);
        }
    }
}
```

Исходная строка

Разбиваем ее на части на основе Разделителя - «,»

Выводим получившийся массив на экран с помощью цикла for - each

Краткие итоги лекции

Массивы — структуры данных, хранящие последовательность элементов одного типа. Доступ к каждому элементу осуществляется по индексу. Значение индекса начинается с 0. Попытка обращения к несуществующему индексу приведет к аварийному завершению программы.

Массивы в Java — ссылочный тип данных. Поэтому состоят из двух частей ссылки на массив и самих данных. Память под данные выделяются с помощью оператора **new**.

Существует целый ряд методов направленных на более удобную работу с массивами.

Двумерные массивы в Java, являются одномерными массивами каждый элемент которого ссылка на одномерный массив. Эта особенность позволяет создавать «рваные» массивы.

Строки — это массивы символов. В Java существует много методов для работы со строками.

Литература

- 1) Герберт Шилд — Java 8. Полное руководство. 9-е издание, «Вильямс» — 2015, стр. 94 - 102
- 2) Кей Хорстман, Гари Корнел — Библиотека профессионала Java. Том 1. Основы. 9-е издание, «Вильямс» — 2014, стр. 116 — 127

Домашнее задание -Уровень 1

- 1) Дан массив {0,5,2,4,7,1,3,19} — написать программу для подсчета нечетных цифр в нем.
- 2) Написать код для возможности создания массива целых чисел (размер вводится с клавиатуры) и возможности заполнения каждого его элемента вручную. Выведите этот массив на экран.
- 3) Создать массив случайных чисел (размером 15 элементов). Создайте второй массив в два раза больше, первые 15 элементов должны быть равны элементам первого массива, а остальные элементы заполнить удвоенных значением начальных. Например
Было → {1,4,7,2}
Стало → {1,4,7,2,2,8,14,4}
- 4) Введите строку текста с клавиатуры — реализуйте программу для возможности подсчета количества символа — 'b' в этой строке, с выводом результат на экран.

Домашнее задание — Уровень 2

1) «Перевернуть массив». Т.е. написать программу которая повернет базовый массив на 90,180,270 градусов по часовой стрелке. (При выполнении задания использовать дополнительный массив нельзя). В примере показан поворот на 90 градусов - (3 часа)

Было

```
[1, 2, 3, 4, 5, 6]
[1, 2, 3, 4, 5, 6]
[1, 2, 3, 4, 5, 6]
[1, 2, 3, 4, 5, 6]
[1, 2, 3, 4, 5, 6]
[1, 2, 3, 4, 5, 6]
```

Стало

```
[1, 1, 1, 1, 1, 1]
[2, 2, 2, 2, 2, 2]
[3, 3, 3, 3, 3, 3]
[4, 4, 4, 4, 4, 4]
[5, 5, 5, 5, 5, 5]
[6, 6, 6, 6, 6, 6]
```

2) Написать код для зеркального переворота массива (7,2,9,4) -> (4,9,2,7). - массив может быть произвольной длины. (При выполнении задания использовать дополнительный массив нельзя)(1 час)

3) Ввести с клавиатуры число (до миллиарда) которое обозначает количество долларов и центов пользователя. Вывести это количество прописью. (4 часа)

Например:

How much money do you have?

123,34

You have: one hundred twenty three dollars thirty four cents

Java Start

(Методы)

Метод — это именованный обособленный блок кода выполняющий последовательность каких либо операций.

Разработал: Цымбалюк А.Н.

Применение методов

- Когда нужно разбить задачу на более мелкие подзадачи
- Когда нужно многократно провести один набор операций над сходными данными
- Когда нужно написать и протестировать метод для дальнейшего использования в ваших проектах

Объявление методов в Java

```
Тип_возвращаемого_значения имя (тип параметр 1, тип параметр 2, ...){  
Тело метода  
}
```

Тип_возвращаемого_значения - любой тип переменных в Java, в теле метода обязательно должно быть ключевое слово **return** после которого идти переменная или значение такого же типа.

! Метод может и не возвращать значения, тогда его тип возвращаемого значения - **void**

Имя - имя метода придуманное пользователем (внимание слова зарезервированные в Java использовать нельзя). Желательно что бы имя метода начиналось с маленькой буквы.

Параметр — любой тип переменных Java

Тело метода — последовательность операций

Объявление метода

Перед типом переменной нужно указать слово **static** - это только пока не начнете использовать классы. Методы объявляются вне главного метода, а используются внутри его.

Пример объявления метода

Метод возвращает целочисленную переменную

```
int sum(int a, int b) {  
    return a+b;  
}
```

Имя метода

Два целочисленных значения - параметры

Оператор return
вернет целочисленное значение

Пример объявления и использования метода

```
package com.gmail.tsa;
```

```
public class Main {
```

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

```
        int d=5;
```

```
        int c=7;
```

```
        System.out.println(sum(d,c));
```

```
    }
```

```
    static int sum(int a,int b){
```

```
        return a+b;
```

```
    }
```

```
}
```

Вызов метода

Фактические параметры

Формальные параметры

Объявление и описание
метода

Формальные параметры позиционно приравниваются фактическим и получают их значение.

Особенности функционирования методов в Java

Методы — обособленные структуры. Из этого следует что переменная объявленная внутри одного метода, не «видна» в теле другого. Т.е. В двух разных методах можно объявить переменную с одним и тем же именем. Это будут две разные переменные.

```
package com.gmail.tsa;
```

```
public class Main {
```

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

```
        int a = 5;
```

```
        int b = 6;
```

```
        int c = 20;
```

```
        System.out.println(sum(a, b));
```

```
    }
```

```
    public static int sum(int a, int b) {
```

```
        int c = a + b;
```

```
        return c;
```

```
    }
```

```
}
```

← Это переменная с метода main()

← Это переменная с метода sum()

И хотя эти переменные носят одно имя они разные, так как объявлены в разных методах

Особенности функционирования методов в Java

Сигнатура метода — это имя метода плюс параметры (причем порядок параметров имеет значение).

В одном классе не может быть методом с одинаковой сигнатурой. Однако может быть произвольное количество методов с одинаковым именем, в таком случае эти методы должны различаться списком параметров (количеством или типом). Такие методы называют **перегруженными**.

Внимание перегрузить метод по типу возвращаемого значения нельзя.

Пример перегруженных методов в Java

```
package com.gmail.tsa;
```

```
public class Main {
```

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

```
        int a = 5;
```

```
        int b = 6;
```

```
        System.out.println(sum(a, b));
```

Вызов метода с 2 параметрами

```
        System.out.println(sum(1, 2, 3));
```

Вызов метода с 3 параметрами

```
    }
```

```
public static int sum(int a, int b) {
```

```
    int c = a + b;
```

```
    return c;
```

```
}
```

```
public static int sum(int a, int b, int c) {
```

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

```
}
```

Перегруженный метод sum. В данном примере по количеству параметров.

```
}
```

Особенности функционирования методов в Java

Можно объявить метод принимающий произвольное количество параметров одного типа. Для этого параметр объявляется как

Тип ... имя

В таком случае фактические параметры неявно упаковываются в массив. В любом случае такой параметр должен быть последним в списке параметров.

Пример метода с произвольным количеством параметров одного типа

```
package com.gmail.tsa;
```

```
public class Main {
```

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

```
        System.out.println(sum(1, 2));
```

```
        System.out.println(sum(1, 2, 3));
```

Вызов с разным количеством параметров

```
        System.out.println(sum(1, 2, 3, 4));
```

```
    }
```

Переменные неявно упакуются в массив a

```
    public static int sum(int... a) {
```

```
        int sum = 0;
```

```
        for (int i : a) {
```

```
            sum += i;
```

```
        }
```

```
        return sum;
```

```
    }
```

```
}
```

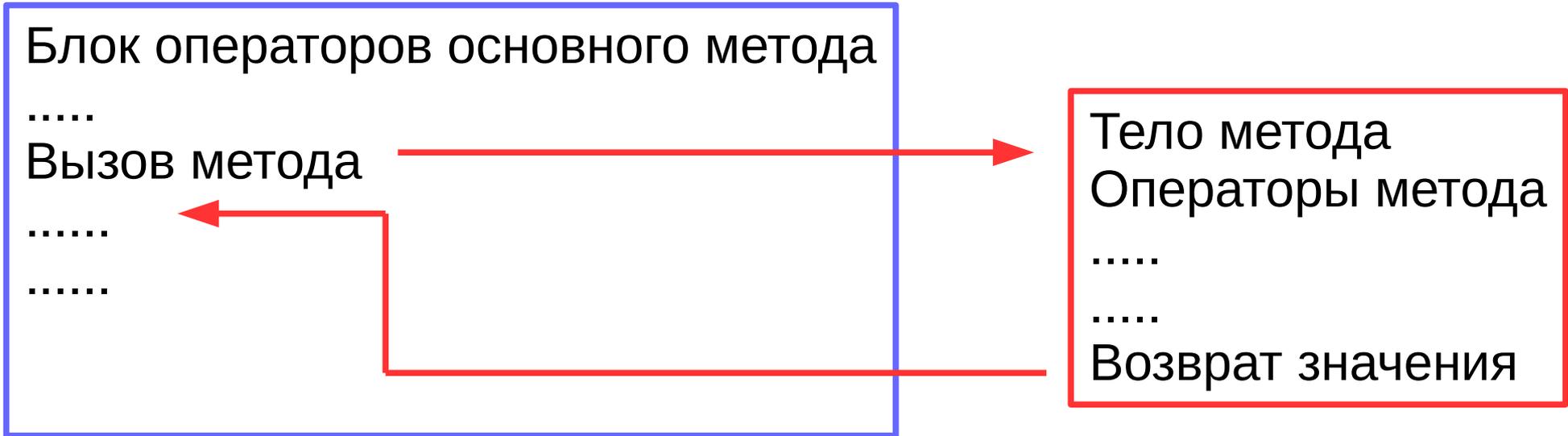
Пример объявления и использования void метода

```
package com.gmail.tsa;  
  
public class Main {  
  
    public static void main(String[] args) {  
  
        int d=5;  
        int c=7;  
  
        sum(d,c);  
  
    }  
  
    static void sum(int a,int b){  
        System.out.println(a+b);  
    }  
  
}
```

Использование void метода

Объявление и описание метода

Как работают методы



Передача параметров в методы

В Java все типы данных передаются по значению. Однако есть разница между примитивными и ссылочными типами данных.

Все примитивные типы передаются по значению — т.е. при передаче создается локальная копия переменной и в методе используется копия. Т.е. при изменении значение переменной в методе в главном методе ее значение не меняется.

Ссылочные типы также передаются по значению — однако это касается только ссылки (т. е. адресата ссылки сменить нельзя), данные же на которые она ссылается изменить можно.

Пример метода с параметром примитивного типа

```
package pr3;

public class Pr3 {

    public static void main(String[] args) {

        int a=5;
        int b=3;

        System.out.println(a);

        System.out.println(sum(a,b));

        System.out.println(a);
    }

    static int sum(int a,int b){
        a=a+3;
        System.out.println(a);
        return a+b;
    }
}
```

И хотя в этом методе переменная a, меняется это не оказывает влияния на переменную a метода main

Пример метода с параметром ссылочного типа

```
package com.gmail.tsa;

import java.util.Arrays;

public class Main {

    public static void main(String[] args) {

        int[] a = { 1, 2, 3, 4, 5, 67, 8 };
        System.out.println(Arrays.toString(a)); ← До вызова метода
        sum(a);
        System.out.println(Arrays.toString(a)); ← После вызова метода

    }

    static void sum(int[] a) {
        a[3] = a[3] + 3;
        System.out.println(Arrays.toString(a));

    }
}
```

И хотя ссылка в методе не изменилась, данные по этой ссылке изменить можно. Так по ссылке изменен 3 элемент массива.

Некоторые замечания о методах

В теле метода могут вызываться другие методы

```
package com.gmail.tsa;

public class Main {

    public static void main(String[] args) {

        System.out.println(calculateVolume(2.1,3));

    }

    static double calculateArea(double r){

        return Math.PI*r*r;

    }

    static double calculateVolume(double r,double h){

        return h*calculateArea(r);

    }

}
```

↑

Вызов другого метода

Некоторые замечания о методах

Параметром метода может быть результат другого метода

```
package com.gmail.tsa;

public class Main {

    public static void main(String[] args) {

        System.out.println(calculateVolume(calculateArea(2.1),3));
    }

    static double calculateArea(double r){
        return Math.PI*r*r;
    }

    static double calculateVolume(double s,double h){
        return h*s;
    }

}
```



Результат метода как параметр

Использование рекурсивных методов

В программировании **рекурсия** — вызов функции (процедуры) из нее же самой, непосредственно (простая рекурсия) или через другие функции (сложная или косвенная рекурсия), например, функция вызывает функцию , а функция — функцию . Количество вложенных вызовов функции или процедуры называется глубиной рекурсии.

Рекурсивная программа позволяет описать повторяющееся или даже потенциально бесконечное вычисление, причем без явных повторений частей программы и использования циклов.

Пример реализации рекурсивного метода

```
package com.gmail.tsa;  
  
public class Main {  
    public static void main(String[] args) {  
        System.out.println(fact(5));  
    }  
  
    public static int fact(int n) {  
        return n <= 1 ? 1 : n * fact(n - 1);  
    }  
}
```



Рекурсивный метод вычисления факториала. Если параметр меньше либо равен 1, то возвращаем 1. Если нет то умножаем параметр на результат вызова этого же метода по с параметром на 1 меньше.

! На самом деле этот пример можно отнести к примеру когда рекурсия не стоит использовать (есть более простой и быстрый метод вычисления факториала).

Краткие итоги лекции

Методы — обособленные блоки кода, которые можно выполнить используя короткое имя. Методы применяются для разбиения задачи на более мелкие части, повторного использования кода, и упрощения отладки приложения.

Описание метода в Java:

- Тип возвращаемого значения (для методов которые описаны в главном классе — сначала указывают модификатор **static**).
- Название метода (с маленькой буквы).
- Список параметров заключенный в круглые скобки.
- Тело метода заключенное в фигурные скобки.

Сигнатура метода — название метода и список параметров. В одном классе не может быть двух методов с одинаковой сигнатурой. Следовательно если у методов одинаковые имена, то они должны различаться количеством или типом параметров.

Рекурсия — возможность метода вызвать самого себя.

Литература

- 1) Герберт Шилд — Java 8. Полное руководство. 9-е издание, «Вильямс» — 2015, стр. 162 - 166
- 2) Кей Хорстман, Гари Корнел — Библиотека профессионала Java. Том 1. Основы. 9-е издание, «Вильямс» — 2014, стр. 160 — 164

Домашнее задание — Уровень 1

- 1) Напишите метод который вернет максимальное число из массива целых чисел.
- 2) Реализуйте метод параметрами которого являются - целое число, вещественное число и строка. Возвращает он конкатенацию строки с суммой вещественного и целого числа.
- 3) Реализуйте метод рисующий на экране прямоугольник из звездочек «*» — его параметрами будут целые числа которые описывают длину и ширину такого прямоугольника.
- 4) Напишите метод который реализует линейный поиск элемента в массиве целых чисел. Если такой элемент в массиве есть то верните его индекс, если нет то метод должен возвращать число - «-1»
- 5) Напишите метод который вернет количество слов в строке текста.

Домашнее задание — Уровень 2

1) Существуют такие последовательности чисел

0,2,4,6,8,10,12

1,4,7,10,13

1,2,4,8,16,32

1,3,9,27

1,4,9,16,25

1,8,27,64,125

Реализуйте программу которая выведет следующий член этой последовательности (либо подобной им). Например пользователь вводит строку 0,2,4,6,8,10,12 ответом программы должно быть число 14. (6 часов)

2) Число-палиндром с обеих сторон (справа налево и слева направо) читается одинаково. Самое большое число-палиндром, полученное умножением двух двузначных чисел – $9009 = 91 \times 99$.

Найдите самый большой палиндром, полученный умножением двух трехзначных чисел. (4 часа)

3) Существует массив $\{1,2,3,4,5\}$ — размер массива может быть произвольным. Напишите программу которая выведет на экран все возможные комбинации из этих цифр. Внимание повторений быть не должно. (2 часа)

Java Start

(Элементы стандартной библиотеки)

Рассмотрены элементы стандартной библиотеки для работы с датой и временем, обработки и форматирования строк.

Разработал: Цымбалюк А.Н.

Получение даты в Java

Для работы необходимо подключить дополнительный пакет - `java.util.Date`;

Потом нужно создать объект типа `Date` и использовать его

```
package com.gmail.tsa;
import java.util.Date;
public class Main {
    public static void main(String[] args) {
        Date date = new Date(); ← Создание переменной типа Date
        System.out.println(date);
    }
}
```

Переменная созданная таким образом будет содержать системное время на момент выполнения данной строки кода

Создание переменной типа Date с произвольной датой

В таком случае применяется следующий синтаксис:

```
Date имя = new Date(long t)
```

Имя — имя которое вы даете объекту

t- переменная с количеством миллисекунд прошедших с 1 января 1970 года (четверг); время с этого момента называют «эрой UNIX» (англ. Unix Epoch).

Пример

```
package com.gmail.tsa;  
import java.util.Date;  
public class Main {  
    public static void main(String[] args) {  
        long t = 1234567891011L;  
        Date date = new Date(t);  
        System.out.println(date);  
    }  
}
```

Создание переменной типа Date, с помощью long значения времени

Не самый удобный способ задания даты, однако есть способ разбора строки.

Форматированный вывод даты и времени

Для работы необходимо подключить дополнительный пакет

```
java.text.SimpleDateFormat
```

Потом нужно создать объект типа SimpleDateFormat на основе специального шаблона

```
SimpleDateFormat stf=new SimpleDateFormat("yy:MM:dd");
```

Имя которое вы дали объекту

Пример шаблона

Создать форматированную строку на основе созданного объекта

```
String res=stf.format(date);
```

Все управляющие символы в шаблоне будут заменены на соответствующее значение даты или времени.

Шаблон даты и времени

Символ	Что означает
G	Эра
y	Год(4 числа)
yy	Год(последние 2 цифры)
yyyy	Год(4 числа)
M	Номер месяца без лидирующих нулей
MM	Номер месяца (с лидирующими нулями если номер месяца < 10)
MMM	Четырех буквенное сокращение месяца
MMMM	Полное название месяца
w	Неделя в году без лидирующих нулей
ww	Неделя в году с лидирующими нулями
W	Неделя в месяце без лидирующих нулей
WW	Неделя в месяце с лидирующим нулем
D	День в году
d	День месяца без лидирующих нулей
dd	День месяца с лидирующими нулями

Символ	Что означает
F	День недели в месяце без лидирующих нулей
FF	День недели в месяце с лидирующими нулями
E	День недели (сокращение)
EEEE	День недели (полностью)
a	AM/PM указатель
H	Часы в 24-часовом формате без лидирующих нулей
HH	Часы в 24-часовом формате с лидирующим нулем
k	Количество часов в 24-часовом формате
K	Количество часов в 12-часовом формате
h	Время в 12-часовом формате без лидирующих нулей
hh	Время в 12-часовом формате с лидирующим нулем
m	Минуты без лидирующих нулей
mm	Минуты с лидирующим нулем
s	Секунды без лидирующих нулей
ss	Секунды с лидирующим нулем
Z	Часовой пояс
Z	Часовой пояс в формате RFC 822

Пример

```
package com.gmail.tsa;

import java.text.SimpleDateFormat;
import java.util.Date;

public class Main {

    public static void main(String[] args) {

        Date date = new Date(); ← Создание нового объекта Date

        SimpleDateFormat sdf = new SimpleDateFormat("dd:MM:yyyy");
        String text = sdf.format(date);
        System.out.println(text); ← Создание форматированной строки
    }
} ← Создание объекта SimpleDateFormat
```

Описание шаблона

The diagram consists of several colored boxes with arrows pointing to specific parts of the Java code. A blue box labeled 'Создание нового объекта Date' has an arrow pointing to 'new Date()'. A red box labeled 'Создание форматированной строки' has an arrow pointing to 'System.out.println(text);'. An orange box labeled 'Описание шаблона' has an arrow pointing to the format string '"dd:MM:yyyy"'. A red box labeled 'Создание объекта SimpleDateFormat' has an arrow pointing to the closing brace of the main method. Another red box is positioned around the 'SimpleDateFormat sdf = new SimpleDateFormat(...)' line, with an arrow pointing to the 'new' keyword.

Организация удобного ввода даты

Нужно подключить дополнительный пакет - `java.text.ParseException`

Выполнить преобразование строки в переменную типа `Date` с помощью шаблона

В программе может одновременно быть несколько шаблонов, например один для вывода даты, а второй для ввода.

Внимание метод разбора строки с генерации новой даты помечен как генерирующий проверяемое исключение поэтому его нужно вызывать в блоке `try - catch`.

Организация удобного ввода даты

```
package com.gmail.tsa;

import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.Scanner;

public class Main {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);
        Date date = new Date();
        SimpleDateFormat sdf = new SimpleDateFormat("dd:MM:yyyy");
        String dateText = "";
        System.out.println("Input date");
        dateText = sc.nextLine();
        try {
            date = sdf.parse(dateText);
        } catch (ParseException e) {
            System.out.println(e);
        }
        SimpleDateFormat sdfOne = new SimpleDateFormat("dd MMMM yyyy");
        String text = sdfOne.format(date);
        System.out.println(text);
        sc.close();
    }
}
```

Описание шаблона для ввода даты



Разбор строки на основании шаблона



Описание шаблона для вывода даты

Подробнее о исключениях и их обработке

В момент выполнения ошибочной (аварийной с точки зрения интерпретатора) строки кода создается объект типа исключение. Если нет участка кода который обработает этот объект то программа автоматически завершит свою работу. Однако если строка заключена в блок `try — catch` то исключение может быть перехвачено (для этого необходимо совпадение типов сгенерированного и обрабатываемого исключения) и обработано, в таком случае выполнение программы продолжится в штатном режиме.

Исключения делятся на:

- Проверяемые — которые обязательно проверять (т. е. Если метод помечен как генерирующий проверяемое исключение то его невозможно вызвать за пределами блока `try — catch`)
- Не проверяемые — не обязательные к проверке в коде.

```
try {  
    date = sdf.parse(dateText);  
} catch (ParseException e) {  
    System.out.println(e);  
}
```

Метод `parse` помечен как генерирующий проверяемое исключение. Поэтому его можно вызвать только в блоке `try - catch`

Тип исключения которое перехватывает этот блок

Методы объекта Date

Метод	Описание
boolean after(Date name)	Вернет - «истина» если вызывающий объект старше name
boolean before(Date name)	Вернет - «истина» если вызывающий объект младше name
int compareTo(Date name)	Сравнивает даты, если они равны то вернется 0. Отрицательное значение если name старше, положительное если наоборот
long getTime()	Вернет кол-во миллисекунд с 1 января 1970 года
void setTime(long time)	Устанавливает время

Использование класса Calendar

Для работы необходимо импортировать дополнительный пакет

```
java.util.Calendar
```

Создать объект класса Calendar

```
Calendar cl=Calendar.getInstance();
```

Некоторые методы по получению и установки даты и времени

Получение данных —

```
get(Calendar.Date )  
get(Calendar.MONTH)  
get(Calendar.YEAR)  
get(Calendar.HOUR)  
get(Calendar.MINUTE)  
get(Calendar.SECOND)
```

Установка данных -

```
set(Calendar.HOUR, значение)  
set(Calendar.MINUTE, значение)  
set(Calendar.SECOND, значение)
```

Пример работы с Calendar

```
package com.gmail.tsa;
```

```
import java.util.Calendar;
```

```
import java.util.Date;
```

```
public class Main {
```

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

```
        Calendar cl = Calendar.getInstance();
```

Установка свойств



```
        cl.set(Calendar.YEAR, 1981);  
        cl.set(Calendar.MONTH, Calendar.OCTOBER);  
        cl.set(Calendar.DAY_OF_MONTH, 11);
```

```
        Date date = cl.getTime();
```

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



Создание объекта типа Date на основе объекта типа Calendar

```
    }
```

```
}
```

Использование классов `StringBuilder` и `StringBuffer`

Эти классы используются для комфортной работы со строками

Для работы необходимо объявить объект нужного типа

```
StringBuffer sb=new StringBuffer("HELLO WORD");
```

! Также можно не указывать параметров или вместо них указать размер.

```
StringBuffer sb=new StringBuffer();
```

```
StringBuffer sb=new StringBuffer(10);
```

Методы классов StringBuilder и StringBuffer

Метод	Что означает
setLength(int размер)	Устанавливается длина. Все символы не попавшие в диапазон теряются.
charAt(int где)	Извлекает символ по индексу
setCharAt(int где, char символ)	Установить символ по индексу
getChars(int нач.,int кон., char [] куда, int с какого индекса)	Копирует подстроку символов
append(что добавить)	Добавляет значение переменных к существующему классу
insert(int индекс, что)	Вставляет одну строку в другую начиная с индекса
reverse()	Меняет порядок символов
delete(int нач,инт конец)	Удаляет кусок текста
deleteCharAt(int индекс)	Удаляет символ по индексу
replace(int нач., int конц., String строка)	Заменяет один набор символов другим
substring(int нач,инт кон)	Вырезает часть строки
int indexOf(String строк ,int нач)	Поиск подстроки (если найдет то первый индекс, если нет то -1)
void trimToSize()	Уменьшает размер для соответствия содержимому

Пример работы с StringBuilder

```
package com.gmail.tsa;
```

```
public class Main {
```

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

```
        StringBuilder sb = new StringBuilder();
```

```
        sb.append("Hello world");
```

```
        sb.insert(6, "my little ");
```

```
        String text = sb.toString();
```

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

```
        text = sb.reverse().toString();
```

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

```
    }
```

```
}
```

Создание объекта StringBuilder

Добавляем текст

Вставляем текст начиная с 6 позиции

Получение обработанной строки

Получение инвертированной строки

В чем же разница ?

StringBuilder — более быстрый, но не синхронизированный.

StringBuffer — более безопасный, но и более медленный.

Так как пока все программы пишутся в однопоточном режиме то предпочтительнее использовать **StringBuilder**

Использование класса Formatter

Предназначен для создания форматированных строк

Для работы необходимо подключить дополнительный пакет

```
java.util.Formatter;
```

Создать объект Formatter

```
Formatter fr=new Formatter();
```

Пример создания объекта Formatter

```
package com.gmail.tsa;
```

```
import java.util.Formatter;
```

```
public class Main {
```

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

```
        Formatter fr=new Formatter();
```

```
    }
```

```
}
```

Методы класса Formatter

Метод	Описание
<code>void close()</code>	Закрывает объект Formatter
<code>void flush()</code>	Сбрасывает буфер формата
<code>Formatter format (String формстрока, аргументы)</code>	Форматирует аргументы в соответствии с строкой форм. строки
<code>String toString()</code>	Возвращает отформатированную строку

Подробнее о строке форматирования

Строка форматирования представляет собой строку где нужно вставить отформатированные символы на определенные места указанные с помощью спецификаторов формата.

Спецификатор формата	Тип аргумента
%a, %A	Шеснадцетиричное с плавающей точкой
%b, %B	Булево
%c	Символ
%d	Десятичное целое
%e, %E	Научная нотация
%f	Десятичное с плавающей точкой
%o	Восьмеричное целое
%n	Вставляет символ перевода строки
%s, %S	Строка
%t, %T	Время и дата
%x, %X	Шеснадцетеричное целое
%%	Вставляет символ %

Использование форматирования

Метод `format (String text, Object ...arg)`

`text` — Строка форматирования

`arg` — Список аргументов форматирования, их должно быть столько как и спецификаторов формата, и того же типа что и спецификатор.

Для каждого спецификатора возможно указание параметров. (См. Таблицу ниже)

После того как строка будет сформирована, ее можно получить используя метод `toString()` у объекта класса `Formatter`.

Использование форматирования

Указание точности(используется со спецификатором %f, %e, %s)

Для числового типа : кол-во десятичных разрядов. кол-во чисел после запятой.

Для символьного типа: мин длинна . max длинна

Спецификаторы **формата даты времени**
(работает только с объектами класса **Calendar**)

Суффикс	Заменяется на
a	Сокращенное название дня недели
A	Полное название дня недели
b	Сокращенное название месяца
B	Полное название месяца
c	День:Месяц:Год
C	Первые два знака года
D	Месяц/день/год
h	Сокращенное название месяца
N	Наносекунды

Пример применения Formatter

```
package com.gmail.tsa;

import java.util.Formatter;

public class Main {

    public static void main(String[] args) {

        Formatter form = new Formatter();

        form.format("%.4f", Math.PI);

        String text = form.toString();

        System.out.println(text);

    }

}
```

Число Пи округленное до 4
символом после запятой

Получение
отформатированной строки

Побитовые операции (можно применять к целочисленным переменным)

Побитовые операции Java

Операция	Описание
	OR
&	AND
^	XOR
~	NOT

Операторы сдвига Java

Операция	Описание
<<	Сдвиг влево
>>	Сдвиг вправо
>>>	Беззнаковый сдвиг вправо

Операция |

Вернет 1 если хоть один из битов равен 1.

```
0101011
0010000
-----
0111011
```

Операция &

Вернет 1 если оба бита равны 1.

```
0101011
0010000
-----
0000000
```

Операция \wedge

Вернет 1 если хотя бы один из битов равен 1. Однако если оба бита равны 1 то результатом будет 0.

```
1101011
1010000
-----
0111011
```

Операция \sim

Инвертирует биты

```
1101011
-----
0010100
```

Операция сдвига

Значение << количество

Сдвигает биты в значение на указанное количество раз

$000001110 \ll 2 = 000111000$

Значение >> количество

Сдвигает биты в значение на указанное количество раз

$000001110 \gg 2 = 000000011$

Значение >>> количество

Сдвигает биты в значение на указанное количество раз без учета знака

$000001110 \ggg 2 = 000000011$

Пример применения побитовых операций

```
public class Main {  
  
    public static void main(String[] args) {  
        String binary[]={  
            "0000", "0001", "0010", "0011", "0100",  
            "0101", "0110", "0111", "1000", "1001", "1010", "1011",  
            "1100", "1101", "1110", "1111"};  
        int a=3;  
        int b=6;  
        int c=a | b;  
        int d=a & b;  
        int e=a ^ b;  
        int f=(~a&b)|(a& ~b);  
        System.out.println("a="+binary[a]);  
        System.out.println("b="+binary[b]);  
        System.out.println("a|b="+binary[c]);  
        System.out.println("a&b="+binary[d]);  
        System.out.println("a^b="+binary[e]);  
        System.out.println("(~a&b)|(a& ~b)"+binary[f]);  
    }  
}
```

Краткие итоги лекции

В этой лекции рассмотрены некоторые инструменты стандартной библиотеки Java

- Date — для работы с датой (в основном текстовое представление)
- Calendar — для работы с датой (числовое представление)
- StringBuilder — для работы со строками (обработка строк)
- Formatter — для работы со строками (Форматирование строк)

Также рассмотрены побитовые операции в Java

- & - AND
- | - OR
- ^ - XOR
- ! - NOT
- << - Сдвиг влево
- >> - Сдвиг вправо
- >>> - Без знаковый сдвиг

Литература

- 1) Герберт Шилд — Java 8. Полное руководство. 9-е издание, «Вильямс» — 2015, стр. 659-666, 489-496, 680 - 695

Домашнее задание — Уровень 1

- 1) Написать программу которая вернет количество миллисекунд прошедших от такого же числа, но в прошлом месяце до сегодняшней даты. Например если сегодня 3 августа, то узнать сколько миллисекунд прошло с 3 июля.
- 2) Написать свой вариант метода `Arrays.toString()` для `int[]`.
- 3) Ввести с консоли число в бинарном формате. Перевести его в десятичный и вывести на экран ("10" -> 2).
- 4) Выведите на экран 10 строк со значением числа Пи. Причем в первой строке должно быть 2 знака после запятой, во второй 3, в третьей 4 и т.д.

Домашнее задание — Уровень 2

- 1) Ввести с консоли дату. Сравнить ее с текущей датой в системе. Вывести отличающиеся части (год, месяц) на экран.
- 2) Расстояние Хэмминга между двумя целыми числами - это число позиций, в которых биты различаются .

Для примера:

$$117 = 01110101$$

$$17 = 00010001$$

 $H = 0+1+1+0+0+1+0+0 = 3$ - расстояние Хэмминга между (117,17)

Даны два положительных целых числа (N, M) в десятичном виде. Вам необходимо подсчитать расстояние Хэмминга между этими двумя числами.

- 3) Вовочка сидя на уроке писал подряд одинаковые числа. Когда Мария Ивановна забрала у него тетрадь там было несколько рядов чисел. Напишите программу которая определит минимальное число которое писал Вовочка например:

$$11111111 \Rightarrow 1$$

$$12121212 \Rightarrow 12$$

$$121121121 \Rightarrow 121$$

Java Start

(Работа с файлами)

Лучший метод уменьшения размеров файлов: "Del *.*" - 100% сжатие.

Неизвестный автор.

Разработал: Цымбалюк А.Н.

Основы работы с файловой системой — IO

При использовании IO основным инструментом по работе с файловой системой является класс File

Расположен в пакете java.io.File

Конструктор принимает в качестве параметра абсолютный или относительный адрес объекта в виде строковой переменной

```
File f1=new File("a.txt");
```



Адрес файла

Объект ассоциированный с файлом

Внимание если файл по указанному адресу существует то объект будет связан с файлом на диске. Если же по указанному адресу файл не существовал по объект будет связан с виртуальным файлом в ОЗУ.

Методы класса File

Метод	Описание
String getName()	Вернет имя файла
String getParent()	Вернет имя каталога файла
String getAbsolutePath()	Вернет путь к файлу
boolean exist()	Вернет true если файл существует
boolean renameTo(File новое имя)	Переименовывает файл
boolean delete()	Удаляет файл
boolean isDirectory()	Проверяет является ли объект каталогом
boolean mkdir()	Создает каталог
boolean createNewFile()	Создать новый файл
boolean mkdirs()	Создает каталог путь для которого еще не создан
String [] list()	Только для каталогов. Список остальных объектов внутри него в виде массива строк
File [] listFiles()	Возвращает массив файлов и подкаталогов, которые находятся в определенном каталоге
long length()	возвращает размер файла в байтах

И файлы и каталоги в Java IO описываются с помощью объекта класса File

Пример использования переменных типа File

```
package com.gmail.tsa;  
import java.io.File;  
import java.io.IOException;
```

```
public class Main {
```

```
    public static void main(String[] args) {  
        File fileTwo = new File("a.txt");
```

```
        try {  
            fileTwo.createNewFile();  
        } catch (IOException e) {  
            System.out.println(e);  
        }
```

Создание файла на
диске по указанному адресу

```
        File fileThree = new File("b.txt");
```

```
        fileTwo.renameTo(fileThree);
```

Перемещение файла

```
        fileThree.delete();
```

Удаление файла

```
        File folder = new File("lib");  
        folder.mkdirs();
```

Создание папки на
диске по указанному адресу

```
        File fileOne = new File(".");  
        File[] arrayFile = fileOne.listFiles();  
        for (File file : arrayFile) {  
            System.out.println(file);  
        }
```

```
    }
```

```
}
```

Процедура для записи информации в файл

Создать объект File

Создать объект PrintWriter

Указать File или просто имя файла как назначение PrintWriter

Вывод информации по аналогии с консолью

Запись должна проводиться только в пределах блока try..catch

Удобным классом для записи данных в файл является класс **PrintWriter**

Конструктор класса

`PrintWriter(OutputStream выходной_поток)`

`PrintWriter(OutputStream выходной_поток, boolean сброс_при_новой_строке)`

`PrintWriter(Writer выходной_поток)`

`PrintWriter(Writer выходной_поток, boolean сброс_при_новой_строке)`

выходной_поток — Определяет объект класса `OutputStream`, который примет вывод

`PrintWriter(File выходной_файл)`

`PrintWriter(File выходной_файл, String набор_символов)`

`PrintWriter(String имя_файла)`

`PrintWriter(String имя_файла, String набор_символов)`

! Удобство заключается в возможности использования методов `print()` и `println()`

Блок try — с - ресурсами

Для закрытия потока можно либо вызвать явно метод `close()` либо использовать введенный в JDK 7 оператор `try` с ресурсами.

Объявление блока `try` с ресурсами

```
try (описание ресурса){  
    Использование ресурса  
}
```

- 1) Ресурсы управляемые оператором `try` должны быть объектами классов, реализующих интерфейс `AutoCloseable`
- 2) Ресурс, объявленный в блоке `try`, является неявно финальным
- 3) Можно управлять несколькими ресурсами, отделив каждый из них в объявлении точкой с запятой.

Пример использования PrintWriter

```
package com.gmail.tsa;
```

```
import java.io.FileNotFoundException;
```

```
import java.io.PrintWriter;
```

```
public class Main {
```

Создание PrintWriter для записи в файл c.txt

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



```
try (PrintWriter a = new PrintWriter("c.txt")) {
```

```
    for (int i = 0; i < 10; i++) {
```

```
        a.println(i);
```

```
    }
```

```
    a.println();
```

Запись данных в файл

```
    a.println("Hello WORLD");
```

```
    } catch (FileNotFoundException e) {
```

```
        System.out.println("ERROR FILE WRITE");
```

```
    }
```

```
}
```

```
}
```

Вычитка данных из файла

Вычитать данные (строки) из файла можно несколькими путями например:

- 1) Используя **Scanner** — для этого нужно указать, что вычитку следует производить из файла.
- 2) Используя **BufferedReader**.

Плюсами использования **Scanner** является возможность перевода текстовых данных в переменные типа `int`, `double` и т. д. Однако этот инструмент работает сравнительно медленно.

BufferedReader — вычитывает только строки, однако имеет более высокую скорость работы.

Считывание данных из файла с помощью Scanner

Создать объект File

Создать объект Scanner указав в качестве данных переменную типа File

Используя методы вычитки Scanner считать данные

Закрыть файл. (Или использовать try с ресурсами)

Считывание данных из файла с помощью Scanner

```
package com.gmail.tsa;

import java.io.File;
import java.io.IOException;
import java.util.Scanner;

public class Main {

    public static void main(String[] args) {
        String text = loadFromFile(new File("a.txt"));
        System.out.println(text);
    }

    public static String loadFromFile(File file) {
        StringBuilder sb = new StringBuilder();

        try (Scanner sc = new Scanner(file)) {

            for (; sc.hasNextLine();) {
                sb.append(sc.nextLine()).append(System.lineSeparator());
            }

        } catch (IOException e) {
            System.out.println(e);
        }

        return sb.toString();
    }
}
```

В качестве источника данных
используем переменную типа File

```
for (; sc.hasNextLine();) {
    sb.append(sc.nextLine()).append(System.lineSeparator());
}
```

Используя методы вычитки Scanner вычитываем
строки в StringBuilder

Процедура для считывания информации из файла с помощью BufferedReader

Создать объект File

Создать объект BufferedReader

Создать объект FileReader в качестве источника данных указать file

Считать данные

Закрывать файл (Или использовать try с ресурсами)

Вычитка данных из файла с помощью BufferedReader

```
package com.gmail.tsa;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;

public class Main {

    public static void main(String[] args) {
        String text = loadFromFile(new File("a.txt"));
        System.out.println(text);
    }
```

Создание BufferedReader



```
try (BufferedReader br = new BufferedReader(new FileReader(file))) {
```

```
    String text = "";
```

```
    for (; (text = br.readLine()) != null;) {
        sb.append(text).append(System.lineSeparator());
    }
```



```
    } catch (IOException e) {
        System.out.println(e);
    }
    return sb.toString();
}
```

Вычитка строк из файла и добавление в StringBuider

```
}
```

Краткие итоги лекции

Для работы с объектами файловой системы в Java используется переменные типа **File**. При их создании указывается адрес файла. Если по указанному адресу существовал файл или папка то созданная переменная будет ассоциирована с ним, в противном случае с виртуальным файлом.

Для записи данных в текстовый файл проще всего использовать **PrintWriter**. В этом случае достаточно указать адрес файла в качестве цели записи. И используя методы `print` и `println()` занесите данные в файл. Запись осуществляется в пределах блока `try — catch`.

Для чтения данных из файла можно использовать **Scanner** и **BufferedReader**. Второй обладает более высокой скоростью работы, но работает только со строками.

Литература

- 1) Герберт Шилд — Java 8. Полное руководство. 9-е издание, «Вильямс» — 2015, стр. 363-370

Домашнее задание — Уровень 1

- 1) Создайте консольный «текстовый редактор» с возможностью сохранения набранного текста в файл.
- 2) Напишите метод для сохранения в текстовый файл двумерного массива целых чисел.
- 3) Реализуйте метод который выведет на экран список всех каталогов которые «лежат» в каталоге который будет параметром этого метода.

Домашнее задание — Уровень 2

- 1) Напишите метод для считывания двумерного массива из файла (размер массива заранее неизвестен, определите его сами на основе данных в файле).
- 2) Считайте из текстового файла текст на английском языке и выведите статистику по частоте использования букв в тексте (т. е. буква — количество использований), причем первыми должны выводиться буквы используемые чаще всего.
- 3) Напишите метод для создания в файле ASCII — арта, т. е. фигуры размером 40x40 символов заполненных символами образующими узор.

Java Start (Final)

- Куда мне отсюда идти?
- А куда ты хочешь попасть?
- А мне все равно, только бы попасть куда-нибудь.
- Тогда все равно куда идти. Куда-нибудь ты обязательно попадешь.

Льюис Кэрролл - «Алиса в стране чудес»

Спасибо за ваше внимание.

Надеюсь данный курс поможет вам в дальнейшем продвижении в сферу программирования на одном из интереснейших языков Java. Если курс вам понравился можете оставить отзыв, или наоборот если есть замечания выскажите их нам мы всегда открыты для диалога.

Логичным дальнейшим шагом в изучении Java является изучение ООП — (объектно ориентированной парадигмы) и ее реализации в языке программирования Java. В дальнейшем следует продолжить изучение использования стандартной библиотеки. Также немалую пользу принесет изучение базовых алгоритмов и структур данных.

Хочется пожелать вам удачи в дальнейшем изучении этого языка программирования.

С уважением Цымбалюк А.Н.