Центр компьютерного обучения «СПЕЦИАЛИСТ» при МГТУ им. Н.Э.Баумана ®

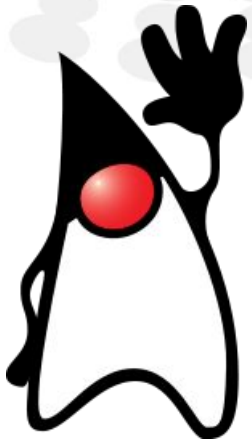
Java. Уровень 1

Основы программирования

Программа курса

1. Введение в Java-технологии
2. Введение в язык программирования Java
3. Операции и операторы Java
4. Стандартные типы Java
5. Разработка классов
6. Наследование и полиморфизм
7. Абстрактные классы и интерфейсы
8. Классы Object и Class
9. Обработка ошибок
10. Потоки данных в Java
11. Дополнительные возможности ООП
12. Коллекции
13. Сериализация
14. Работа с файловой системой
15. Шаблоны проектирования

Учебный центр компьютерного обучения «СПЕЦИАЛИСТ» при МГТУ им. Н.Э.Баумана



1. ВВЕДЕНИЕ В JAVA-ТЕХНОЛОГИИ

Учебный центр компьютерного обучения «СПЕЦИАЛИСТ» при МГТУ им. Н.Э.Баумана

1.1. ЯЗЫК И ПЛАТФОРМА JAVA

Язык Java

- ***Java (Java Programming Language)*** **–** язык программирования высокого уровня:
  - простой (simple)
  - объектно-ориентированный (object-oriented)
  - распределенный (distributed)
  - многопоточный (multithreaded)
  - динамичный (dynamic)
  - независящий от архитектуры (architecture neutral)
  - переносимый (portable)
  - высокопроизводительный (high performance)
  - надежный (robust)
  - защищенный (secure)

Платформа Java

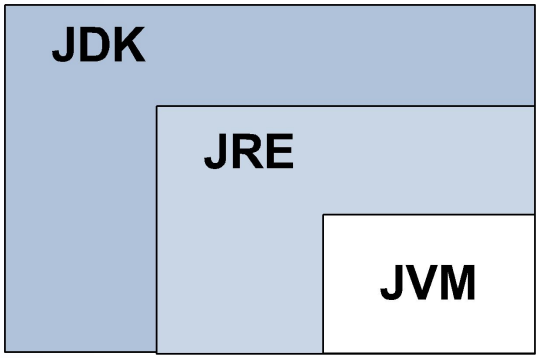
- ***Платформа Java (Java Platform)*** – программная среда, в которой работают приложения Java
- Существуют версии платформы Java для различных ОС (Windows, Linux, Solaris, Mac OS)
- Включает в свой состав:
  - ***Java Virtual Machine*** (***JVM***) – ***виртуальная машина Java*** – программа, интерпретирующая приложения Java
  - ***Java API*** - библиотека программных компонентов (классов и интерфейсов), реализующих стандартный функционал

- ***Java Platform, Standard Edition (Java SE)*** – платформа широкого назначения для рабочих станций
- ***Java Platform, Enterprise Edition (Java EE)*** – платформа для корпоративных приложений и приложений интернет
- ***Java Platform, Micro Edition (Java ME)*** – платформа для устройств с ограниченными ресурсами и мобильных устройств
- ***Java Card*** – платформа для смарт-карт

JRE и JDK

- **Java SE Runtime Environment** (***JRE***) - минимальная реализация платформы Java SE, необходимая для выполнения приложений
  - устанавливается на компьютеры конечных пользователей
  - включает в свой состав JVM и библиотеки, необходимые для выполнения программ

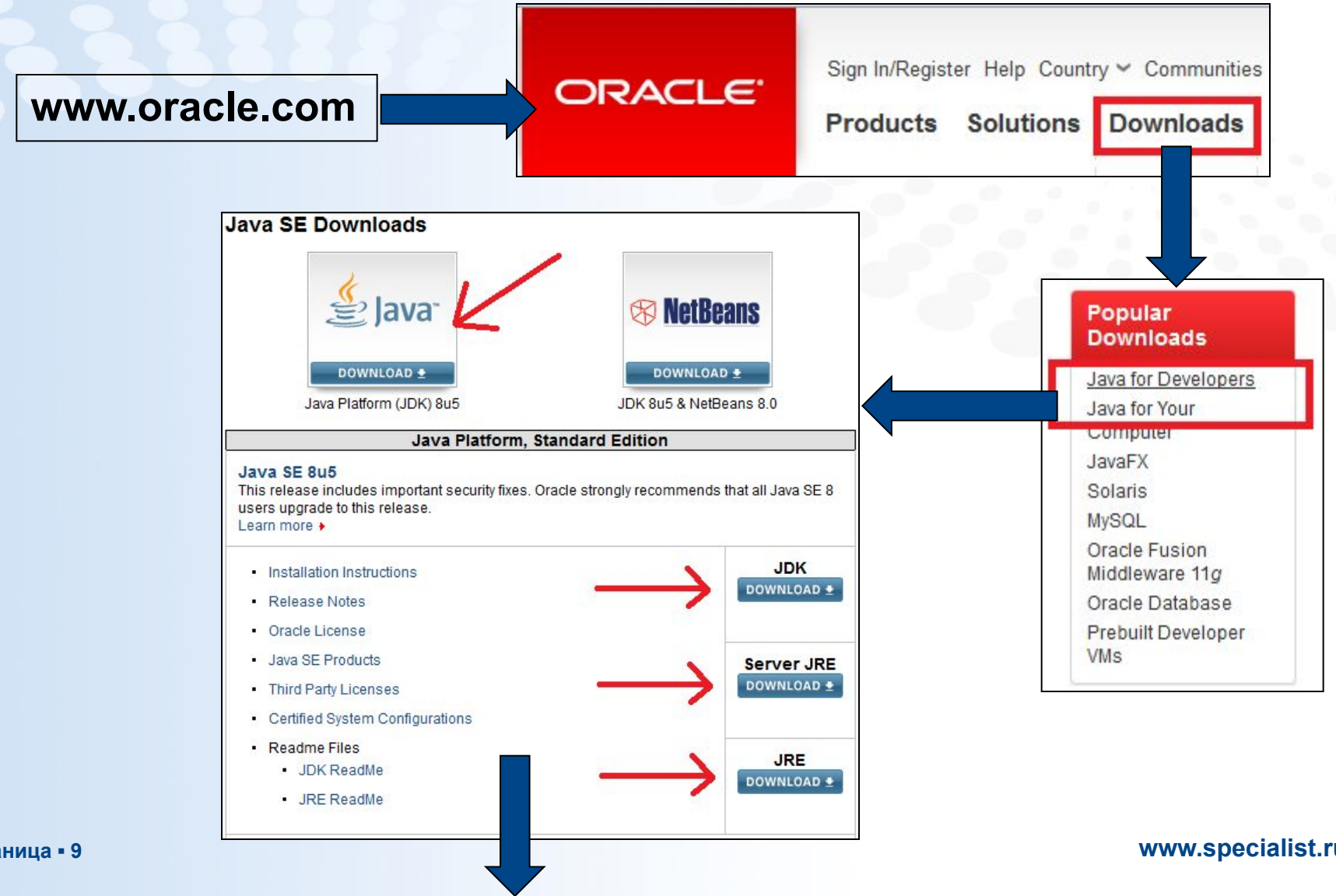
- ***Java Development Kit (JDK)*** – версия Java SE для разработки приложений
  - устанавливается на компьютеры разработчиков
  - включает в свой состав JRE, компилятор, отладчик, примеры программ, дополнительные библиотеки



История Java

- 1991 – начало работы над проектом Java
- 1995 – официальный релиз технологии Java компанией Sun Microsystems
- 1996 – выпуск JDK 1.0
- 1997 – выпуск JDK 1.1
- 1998 – выпуск J2SE 1.2
- 2000 – выпуск J2SE 1.3
- 2002 – выпуск J2SE 1.4
- 2004 – выпуск J2SE 5.0
- 2006 – выпуск Java SE 6.0
- 2010 – компания Sun вошла в состав корпорации Oracle
- 2011 – выпуск Java SE 7.0
- 2014 – выпуск Java SE 8.0

Загрузка и установка платформы Java SE (1)



Загрузка и установка платформы Java SE (2)

Java SE Runtime Environment 8 Downloads

Do you want to run Java™ programs, or do you want to develop Java programs? If you want to run Java programs, but not develop them, download the Java Runtime Environment, or JRE™.

If you want to develop applications for Java, download the Java Development Kit, or JDK™. The JDK includes the JRE, so you do not have to download both separately.

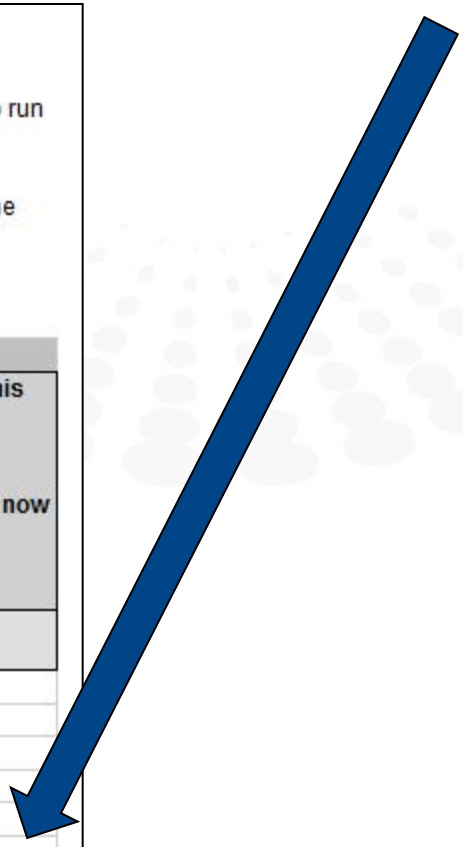
JRE MD5 Checksum

Java SE Runtime Environment 8u5

You must accept the Oracle Binary Code License Agreement for Java SE to download this software.

Thank you for accepting the Oracle Binary Code License Agreement for Java SE; you may now download this software.

Product / File Description	File Size	Download
Linux x86	40.27 MB	jre-8u5-linux-i586.rpm
Linux x86	55.46 MB	jre-8u5-linux-i586.tar.gz
Linux x64	40.4 MB	jre-8u5-linux-x64.rpm
Linux x64	54.41 MB	jre-8u5-linux-x64.tar.gz
Mac OS X x64	56.61 MB	jre-8u5-macosx-x64.dmg
Mac OS X x64	52.61 MB	jre-8u5-macosx-x64.tar.gz
Solaris SPARC 64-bit	50.32 MB	jre-8u5-solaris-sparcv9.tar.gz
Solaris x64	47.99 MB	jre-8u5-solaris-x64.tar.gz
Windows x86 Online	1.53 MB	jre-8u5-windows-i586-iftw.exe
Windows x86 Offline	29.67 MB	jre-8u5-windows-i586.exe
Windows x86	45.87 MB	jre-8u5-windows-i586.tar.gz
Windows x64	32.55 MB	jre-8u5-windows-x64.exe
Windows x64	48.87 MB	jre-8u5-windows-x64.tar.gz



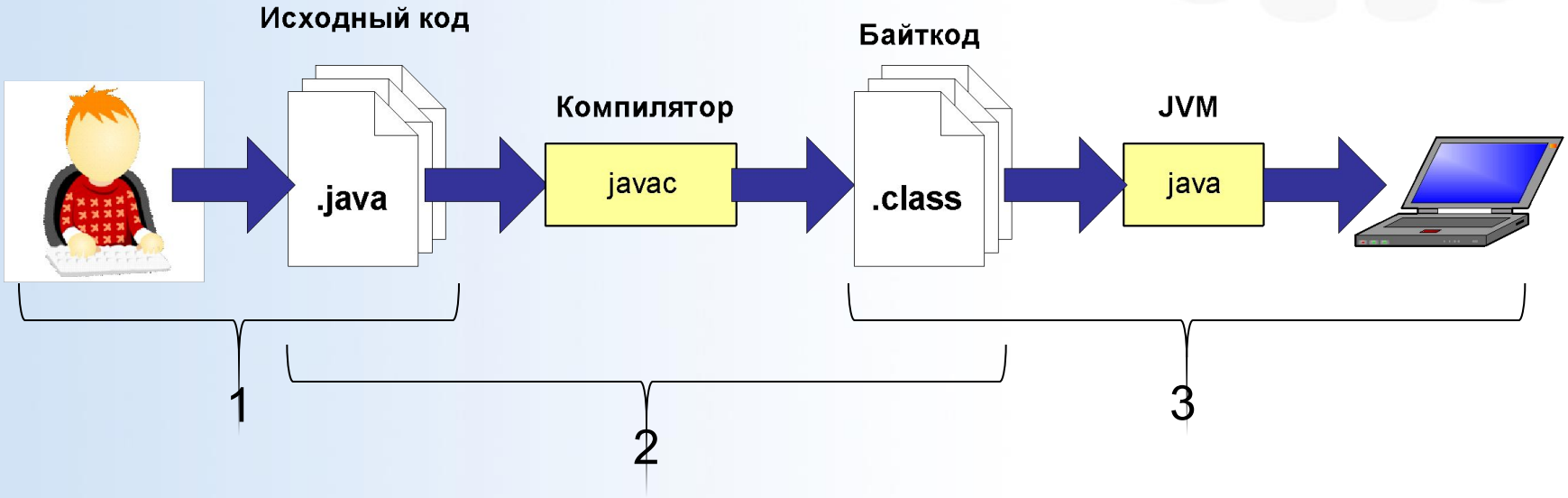
Учебный центр компьютерного обучения «СПЕЦИАЛИСТ» при МГТУ им. Н.Э.Баумана

1.2. ОСОБЕННОСТИ СОЗДАНИЯ ПРИЛОЖЕНИЙ НА JAVA

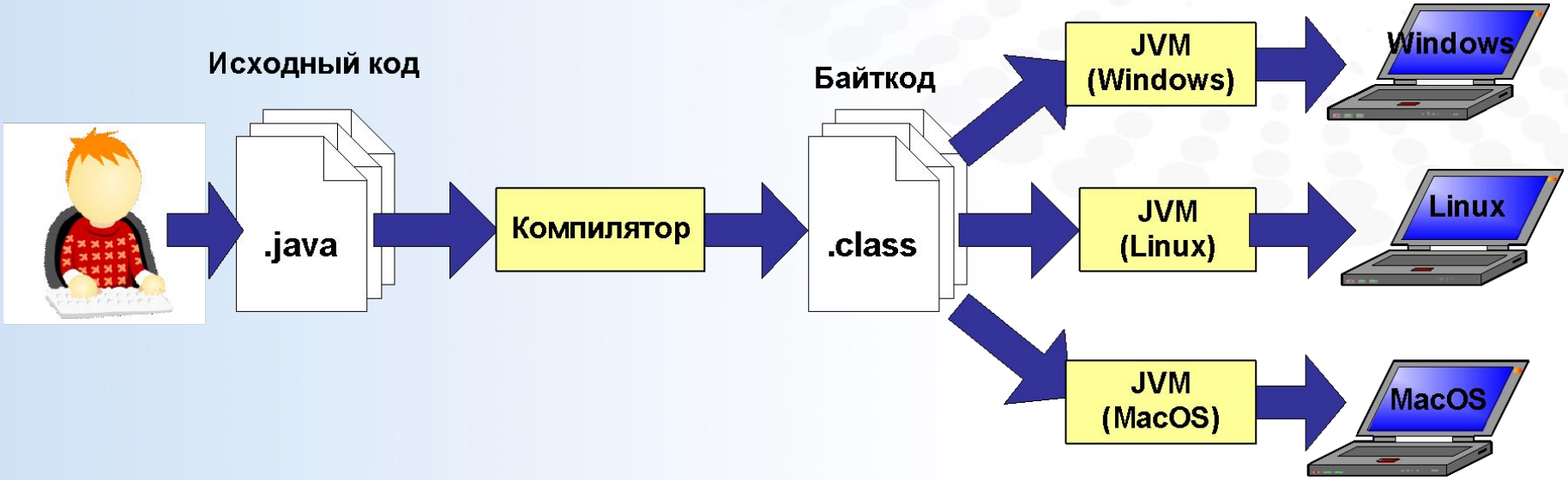
Этапы создания приложения Java

1. Разработка программного кода
2. Компиляция исходного кода в байт-код
3. Выполнение программы в JVM

- ***Байткод (bytecode)*** – машинно-независимый низкоуровневый язык виртуальной машины Java



Переносимость приложений Java



Привет, мир!

HelloWorld.java

```
package hello;

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

Преимущества программирования на Java

- Начать программировать на Java легко
- Маленький объем программного кода
- Высокая эффективность программного кода
- Приложения разрабатываются быстро
- Приложения не зависят от платформы и ОС

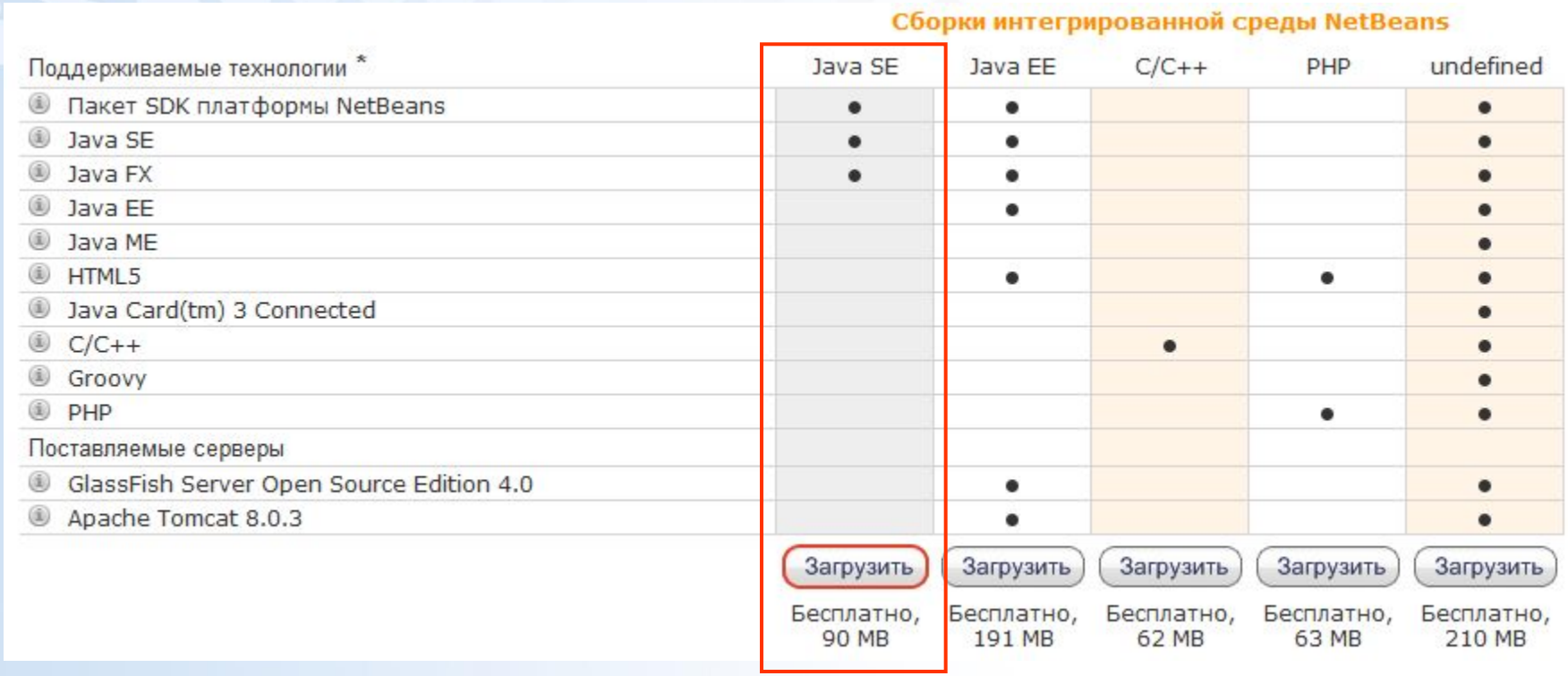
Учебный центр компьютерного обучения «СПЕЦИАЛИСТ» при МГТУ им. Н.Э.Баумана

1.3. СРЕДЫ РАЗРАБОТКИ JAVA

Среды разработки Java

- ***IDE*** – Integrated Development Environment:
- Среды разработки Java
  - NetBeans IDE
    - https://netbeans.org/
  - Eclipse IDE
    - http://www.eclipse.org
  - IntelliJ IDEA
    - http://www.jetbrains.com/idea/
  - …

NetBeans. Загрузка NetBeans

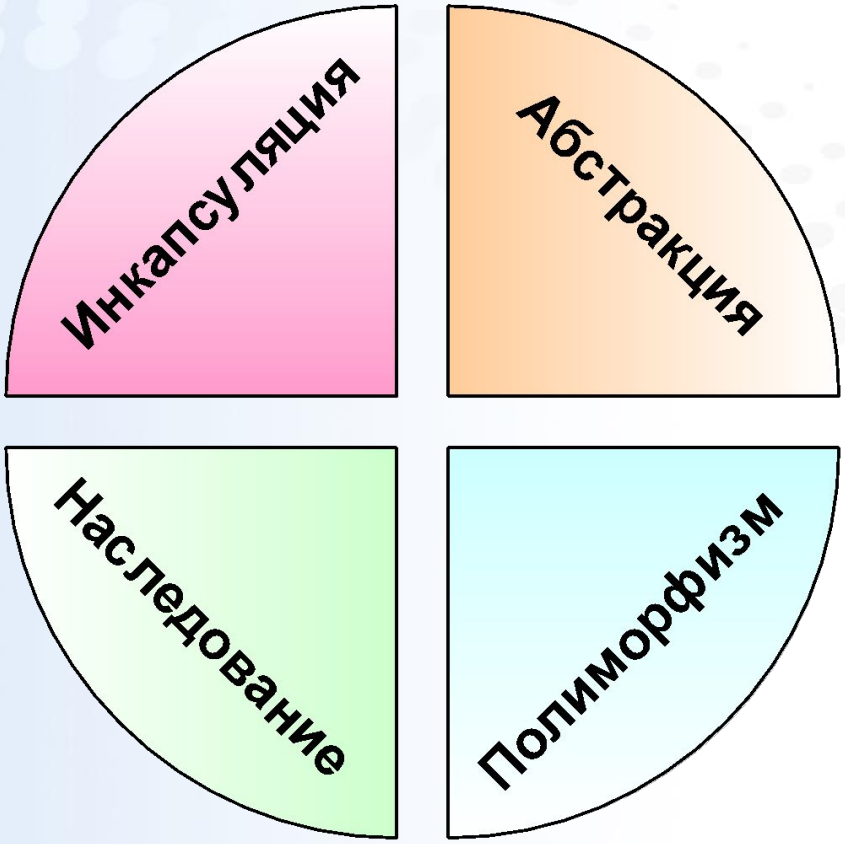


Eclipse. Загрузка Eclipse

Учебный центр компьютерного обучения «СПЕЦИАЛИСТ» при МГТУ им. Н.Э.Баумана

1.4. КОНЦЕПЦИЯ ООП В JAVA

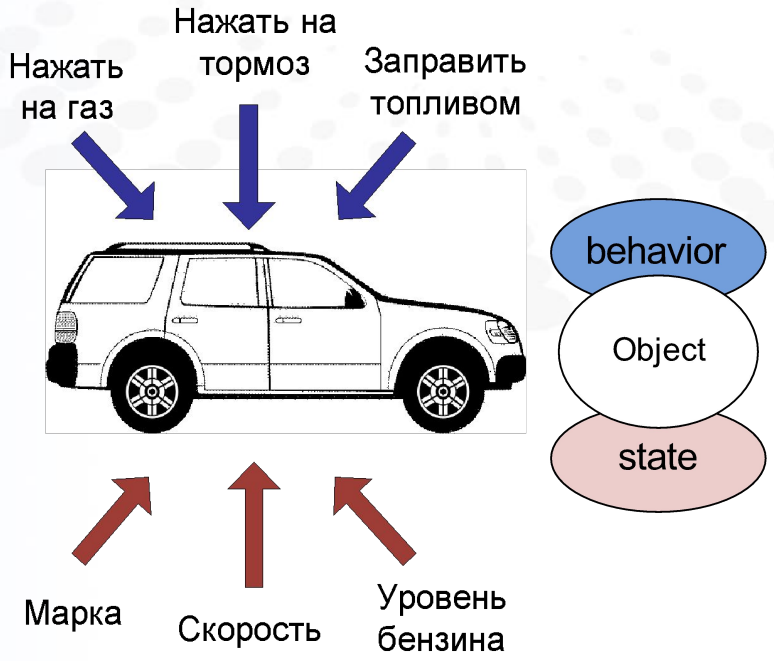
Базовые принципы ООП



Понятие объекта. Инкапсуляция

- Объект реального мира:
  - Состояние (state)
  - Линия поведения (behavior)
- Программный объект (***Object***):
  - ***Поля (fields)***
  - ***Методы (methods)***

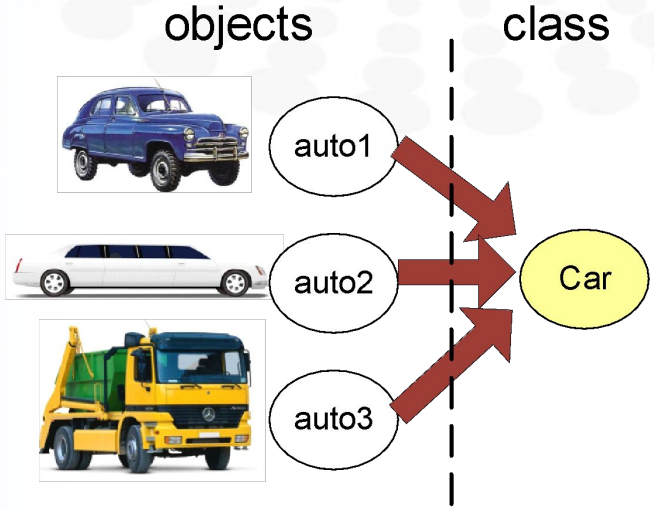
- ***Инкапсуляция (encapsulation)***
  - объединение данных и алгоритмов в рамках одной сущности (объекта)
  - разграничение доступа к элементам объекта



Понятие класса

- ***Класс*** (***class***) описывает признаки состояния и поведение множества схожих объектов
- Класс – это пользовательский ***тип данных***

```
class Car {
    String name;
    int speed;
    int fuel;

    void setName(String newName) {...}
    void speedUp(int delta) {...}
    void applyBrakes(int delta) {...}
    void addFuel(int delta) {...}
    void printState() {...}
}
```

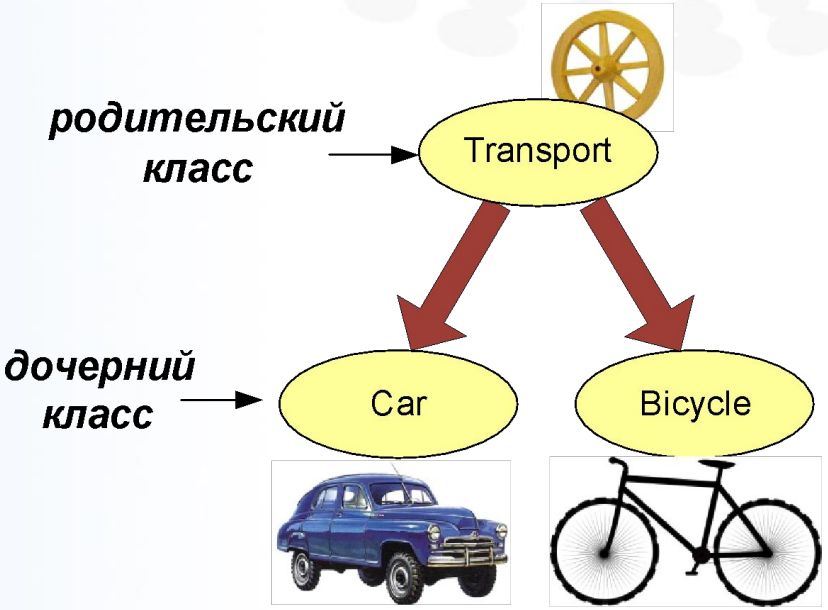



```
Car auto1 = new Car();
Car auto2 = new Car();
Car auto3 = new Car();
```

Наследование

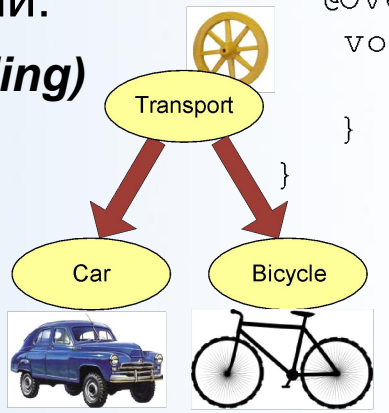
- ***Наследование*** (***inheritance***) – механизм создания новых классов на основе существующих
- При наследовании ***дочернему классу*** (***subclass***) передаются поля и методы ***родительского класса*** (***superclass***)
- У класса может быть один родитель и любое количество дочерних классов

```
class Transport {
    …
}

class Car
      extends Transport {
    …
}
```



Полиморфизм

- ***Полиморфизм (polymorphism)***
  - имеется несколько реализаций алгоритма
  - выбор реализации осуществляется в зависимости от типа объекта и типа параметров
- Механизмы реализации:
  - ***Перегрузка (overloading)*** методов
  - ***Переопределение (overriding)*** методов



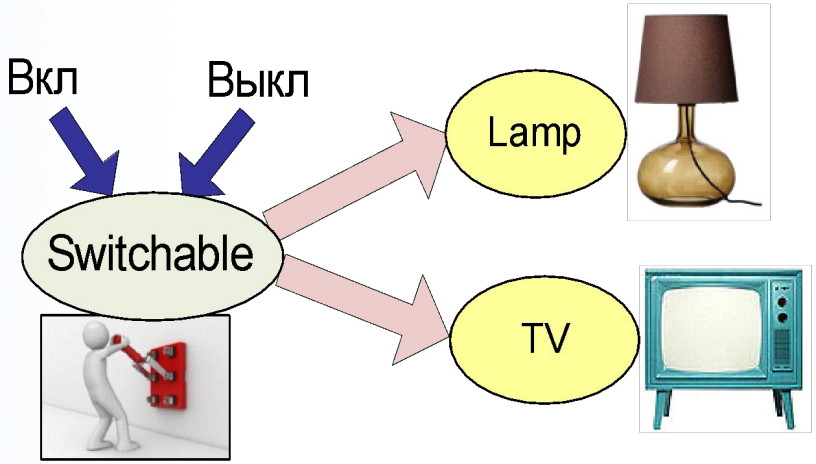

```
int max(int a, int b) {...}
int max(int a, int b, int c) {...}
int max(int[] arr) {...}
```

```
abstract class Transport{
    abstract void beep();
}

class Car extends Transport{
   @Override
   void beep() {
       System.out.println("Би-би");
   }
}

class Bicycle extends Transport{
  @Override
   void beep() {
       System.out.println("Дзынь-дзынь");
   }
}

...
Transport tr1 = new Car();
Transport tr2 = new Bicycle();
tr1.beep();  tr2.beep();
```

Понятие интерфейса

- ***Интерфейс*** (***interface***) определяет возможное поведение объектов
- Интерфейс представляет собой совокупность методов без реализации
- При объявлении класса можно указать, какие интерфейсы он будет поддерживать

```
interface Switchable {
    void switchOn();
    void switchOff();
}

class Lamp
    implements Switchable {
    …
}
```



Центр компьютерного обучения «СПЕЦИАЛИСТ» при МГТУ им. Н.Э.Баумана

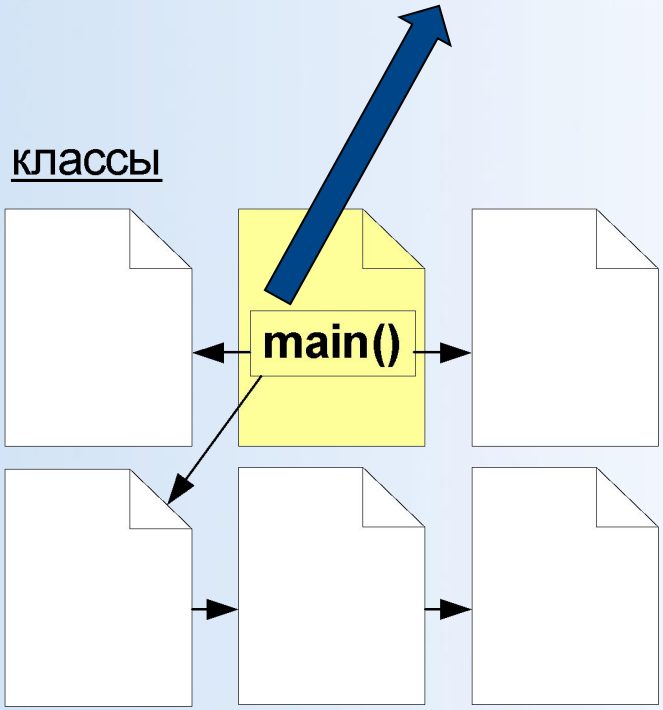
2. ВВЕДЕНИЕ В ЯЗЫК ПРОГРАММИРОВАНИЯ JAVA

Учебный центр компьютерного обучения «СПЕЦИАЛИСТ» при МГТУ им. Н.Э.Баумана

2.1. СТРУКТУРА ПРИЛОЖЕНИЙ JAVA

Java-приложение

```
public static voidmain(String [] args)
```



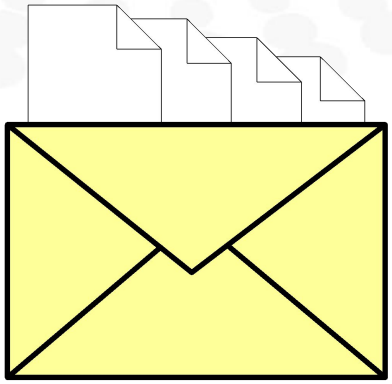
- Работа приложения Java начинается с выполнения главного метода ***main()*** одного из классов
  - метод принимает на вход массив параметров командной строки
- Класс, с которого начинается выполнение приложения java принято называть ***главным классом (main class***)

```
java hello.HelloWorld
```

Пакеты

- ***Пакет (package)*** – пространство имен в Java
- Пакет объединяет ***типы*** (классы, интерфейсы, перечисления), относящиеся к одной предметной области или одной задаче

```
package hello;
import java.util.*;

public class HelloWorld {
    public static void main(String [] args) {
        Random r = new Random();
        System.out.println("Привет, мир");
        System.out.println("Случайное число: "
                           +r.nextInt());
    }
}
```



Стандартные классы Java SE

- java.lang.String
- java.lang.Math
- java.lang.Integer
- java.lang.Thread
- …
- java.util.ArrayList
- java.util.Random
- …
- java.io.PrintWriter
- java.io.File
- …
- java.awt.Frame
- java.awt.Button
- …
- …


```
import java.util.*;

import java.util.Random;
```

Классы и пакеты в файловой системе

- Один тип (класс/интерфейс) может принадлежать только одному пакету
- Один файл исходного кода может содержать только один публичный (public) тип, доступный за пределами пакета
- Рекомендуется создавать отдельный файл исходного кода под отдельный тип
- Файл исходного кода должен называться по имени публичного типа
- Переменная окружения ***CLASSPATH*** определяет пути, по которым приложения Java будут искать пользовательские классы

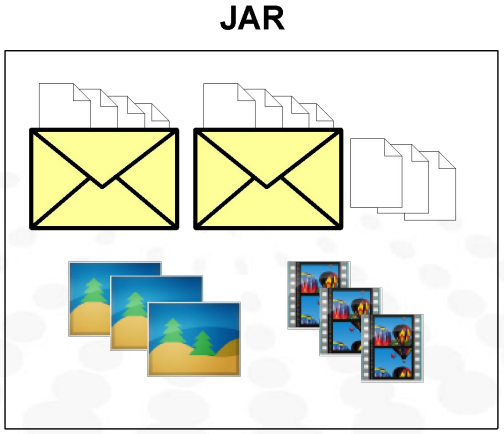
- Имя пакета ассоциируется с иерархией папок в файловой системе
- Рекомендуется использовать в названии пакета только символы нижнего регистра
- Стандартные пакеты java начинаются со слов `java.` и `javax.`
- Внешние компании используют для именования пакетов свои доменные имена (например, `com.ibm.`)

```
package com.my.graphics;
public class Rectangle {
    ...
}
```

`....\com\my\graphics\Rectangle.java`

Архивы Java

- ***Архив Java*** – ***JAR*** – архив в формате ZIP, содержащий пакеты и классы Java, а также ресурсы проекта
  - безопасность
  - эффективное хранение
  - быстрая загрузка
  - переносимость
  - целостность кода и поддержка версий




```
jar cf myapplet.jar MyApplet.class images video
```

```
jar cfve hw.jar hello.HelloWorld hello
java -jar hw.jar
```


```
Manifest-Version: 1.0
Created-By: 1.7.0_45 (Oracle
Corporation)
Main-Class: hello.HelloWorld
```

Учебный центр компьютерного обучения «СПЕЦИАЛИСТ» при МГТУ им. Н.Э.Баумана

2.2. ОСНОВНЫЕ ЯЗЫКОВЫЕ КОНСТРУКЦИИ

Ключевые слова

abstract	continue	for	new	switch
assert	default	goto	package	synchronized
boolean	do	if	private	this
break	double	implements	protected	throw
byte	else	import	public	throws
case	enum	instanceof	return	transient
catch	extends	int	short	try
char	final	interface	static	void
class	finally	long	strictfp	volatile
const	float	native	super	while

true false null

Идентификаторы

- ***Идентификатор*** (***identifier***) – имя программного объекта
  - чувствительны к регистру
  - могут быть любой длины
  - могут содержать:
    - любые буквы Юникод
    - цифры
    - символы ‘$’ и ‘_’
  - должны начинаться с буквы или символов ‘$’ и ‘_’
  - не должны совпадать с ключевыми словами Java

- Идентификаторы:
  - переменные (variables)
  - методы (methods)
  - типы (types)
    - классы (classes)
    - интерфейсы (interfaces)
    - перечисления (enums)
  - пакеты (packages)

Примитивные типы данных

- ***byte***
  - 8-битное целое
  - от -128 до 127
- ***short***
  - 16-битное целое
  - от -32 768 до 32 767
- ***int***
  - 32-битное целое
  - от -2^{31} до $2^{31}-1$
- ***long***
  - 64-битное целое
  - от -2^{63} до $2^{63}-1$
- ***float***
  - 32-битное число с плавающей точкой
- ***double***
  - 64-битное число с плавающей точкой
- ***boolean***
  - логическое значение
  - ***true*** / ***false***
- ***char***
  - 16-битный символ Unicode
  - от 0 до 65535

Переменные

- ***Переменные*** (***variable***)
  - ***Переменные экземпляра*** (instance variable, non-static fields, ***нестатические поля***)
  - ***Переменные класса*** (class variables, static fields, ***статические поля***)
  - ***Локальные переменные*** (local variable)
  - ***Параметры методов*** (method parameters)

- Рекомендации по именованию переменных
  - если значение будет меняться:
    - имя начинается с маленькой буквы
    - если имя состоит из нескольких слов, второе и последующие слова начинаются с большой буквы
  - если значение не будет меняться (***именованные константы***):
    - имя записывается в верхнем регистре
    - слова отделяются друг от друга символом '_'

```
public class HelloWorld3 {
    static final double PI_VALUE = 3.14;
    int myVariable = 20;

    public static void main(String[] args) {
        int test = 10;
        int anotherVariableWithLongName = 5;
    }
}
```

Константы

- Целочисленные
- Вещественные
- Символьные
- Строковые
- Логические (булевские)

```
int decVal   = 10;
int hexVal   = 0x10;
int binVal   = 0b10;
long longVal = 10L;
int salary    = 1_000_000;
long hexBytes = 0xFF_EC_1D_2A;

double v1 = 3.14;
double v2 = .3;
double v3 = 1.23e2;
double v4 = 3.14D;
float v5 = 3.14f;

char c1 = 'A';
char c2 = '\u0108';

System.out.println("Hello,\nworld");
String s = "Hello";

boolean c = true;
```

Преобразование типов

- без потери точности

```
int x = 100;
double y = x;
double z = (double)x;
```

- с возможными потерями точности и значения

```
double a = 10.9;
int b = (int)a;   // a = 10;

int c = 257;
byte d = (byte)c; // d = 1;
```

- явное

```
int x = 100;
double z = (double)x;
```

- неявное

```
int x = 100;
double y = x;
```

Объекты и ссылки

- ***Объект*** (***object***) – экземпляр класса
- Создание объекта осуществляется ***динамически*** при помощи оператора ***new***:

```
new Random();
```

- Работа с объектом осуществляется через переменную-ссылку

```
Random value = new Random();
```

- ***null*** – пустая ссылка

```
Random value = null;
```

- Удаление объекта осуществляется автоматически ***сборщиком мусора*** (***garbage collector***)

- Объект удаляется если все ссылки на объект утеряны, в частности при:
  - обнулении ссылки
  - выхода из области видимости ссылки

```
if (a>0)
{
    Random value =
        new Random();
    System.out.println(
        value.nextInt());
}
```

```
System.gc();
```

Методы

```
class Car {
    String name;
    int speed;

    void setName(String newName) {
        name = newName;
    }
    void speedUp(int delta) {
        speed = speed + delta;
    }
    void applyBrakes(int delta) {
        speed = speed - delta;
    }
    int getSpeed() {
        return speed;
    }
}
```

- Рекомендации по именованию методов:
  - имя метода начинается с маленькой буквы
  - первое слово в названии метода – глагол
  - второе и последующие слова в названии начинаются с большой буквы

Обращение к элементам объекта

Обращение к полю

```
имя_класса.ст_поле
ссылка_на_объект.поле
```

```
double angle = 45. / 180 * Math.PI;
double sinAngle = Math.sin(angle);
System.out.println("Синус угла 45 градусов:" + sinAngle);
Random rnd = new Random();
double doubleRandomValue= rnd.nextDouble();
System.out.println("Случайное число:" + doubleRandomValue);
```

Вызов метода

```
имя_класса.ст_метод(параметр1, параметр2, ...)
имя_класса.ст_метод_без_параметров()
ссылка_на_объект.метод(параметр1, параметр2, ...)
ссылка_на_объект.метод_без_параметров()
```

Массивы

- ***Массив*** (***array***) – множество однотипных ***элементов***
  - Размер массива определяется при создании массива
  - Размер массива не может быть измен
- Обращение к элементам осуществляется по индексу (начиная с нуля)

- Объявление массива

```
int[] arrayOfInt;
```

- Создание массива

```
arrayOfInt = new int[3];
```

- Обращение к элементам массива

```
arrayOfInt[0] = 10;
arrayOfInt[1] = 20;
arrayOfInt[2] = arrayOfInt[0]
  + arrayOfInt[1];
```

- Сокращенная форма

```
int[] arrayOfInt2 = {5,2,99};
```

Комментарии

- Строчные:

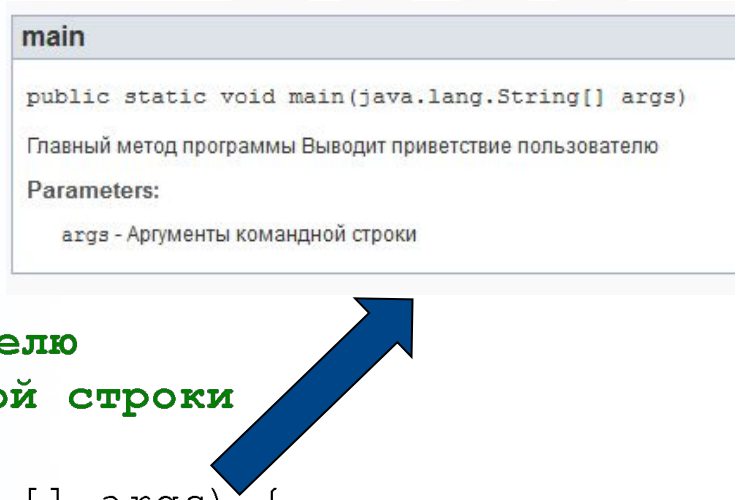
```
int a=1; // Это строчный комментарий
int b=2; // Это тоже строчный комментари
```

- Блочные:

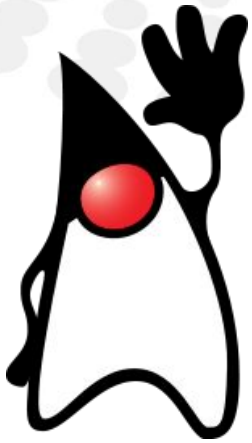
```
int a=1; /* Это
 блочный
 комментарий*/  int b=2;
```

- Документация:

```
/**
* Главный метод программы
* Выводит приветствие пользователю
* @param args Аргументы командной строки
*/
  public static void main(String [] args) {
      System.out.println("Hello, world");
  }
```



Центр компьютерного обучения «СПЕЦИАЛИСТ» при МГТУ им. Н.Э.Баумана



3. ОПЕРАЦИИ И ОПЕРАТОРЫ JAVA

компьютерного Центр обучения «СПЕЦИАЛИСТ» при МГТУ им. Н.Э.Баумана

3.1. ОПЕРАЦИИ

Типы операций

- Присваивание
- Инкремент и декремент
- Арифметические (бинарные и унарные) операции
- Операции сравнения
- Логические операции
- Побитовые операции
- Операции сдвига
- «Сложное» присваивание
- Условная операция
- Операция instanceof

Присваивание. Инкремент. Декремент

=	***Присваивание (assignment)***	`int x = 10;`
		`int y = 15;`
		`int z = x;`
		`x = z + y;`

++	***Инкремент (increment)***	`int a = 1;`
		`a++;` `// a = 2`
		`int b = ++a;` `// b = 3, a = 3`
		`int c = a++;` `// c = 3, a = 4`

--	***Декремент (decrement)***	`a--;` `// a = 3` `int d = a--;` `// d = 3, a = 2`

- Постфиксный инкремент/декремент (a++) возвращает исходное значение переменной
- Префиксный инкремент/декремент (++a) возвращает новое значение переменной

Арифметические операции

+	***сложение / конкатенация (additive / concatenation)***	`int a = 2 + 3;` `System.out.print("Hello " + "world");`
–	***вычитание (subtraction)***	`int b = 10 - a;`
*	***умножение (multiplication)***	`int c = a * b;`

/	***деление (division)***	`int d = 10 / 2;` `// d = 5`
		`float d1 = 5 / 2;` `// d1 = 2.0`
		`float d2 = 5f / 2;` `// d2 = 2.5`

%	***взятие остатка (remainder)***	`int e = 10 % 3;` `// e = 1`
+	***унарный плюс (unary plus)***	`int x = +a;` `// x = 5`
–	***унарный минус (unary minus)***	`int y = -a;` `// y = -5`

Операции сравнения. Логические операции

==	*равно (equal to)*
!=	*не равно (not equal to)*
>	*больше (greater than)*
>=	*больше или равно*
<	*меньше (less than)*
<=	*меньше или равно*

```
boolean x = 10==10; // true
boolean y = 5>10;   // false
int a=1, b=2;
if (a!=b)           // true
    System.out.print("a!=b");
```

a	b	!a	a&&b	a||b
false	false	true	false	false
false	true	true	false	true
true	false	false	false	true
true	true	false	true	true

&&	*логическое И (conditional AND)*
||	*логическое ИЛИ (conditional OR)*
!	*логическое НЕ (complement)*

```
int a=1, b=2;
if ( (a==1) && (b==2) )
    System.out.println("a is 1 AND b is 2");
if ( (a==1) || (a==10) )
    System.out.println("a is 1 OR 10");
if ( !(a==10) )
    System.out.println("a is not 10");
```

Побитовые операции. Операции сдвига

~	***Побитовое НЕ (bitwise complement)***
&	***Побитовое И (bitwise AND)***
^	***Побитовое исключающее ИЛИ (bitwise XOR)***
|	***Побитовое ИЛИ (bitwise OR)***
>>	***Сдвиг вправо (signed right shift)***
>>>	***Сдвиг вправо (unsigned right shift)***
<<	***Сдвиг влево (signed left shift)***

```
int a =     0b00000111; // a=7
int b = a | 0b00001001; // b=15
int c = a & 0b00001001; // c=1
int d = a ^ 0b00001001; // d=14
```

A	B	~A	A&B	A|B	A^B
0	0	1	0	0	0
0	1	1	0	1	1
1	0	0	0	1	1
1	1	0	1	1	0

```
int x = a >> 2;     // x=1
int y = a << 1;     // y=14
```

«Сложное» присваивание

+=
-=
*=
/=
%=
	присваивание вместе с арифметической операцией	`int a = 1;`
		`a += 10;` `// a = a + 10`
		`a *= 5;` `// a = a * 5`
		`a %= 3;` `// a = a % 3`

&= |= ^=	***присваивание вместе с побитовой операцией***	`a &= 7;` `// a = a & 7;`
<<= >>= >>>=	***присваивание вместе со сдвигом***	`a >>= 2;` `// a = a >> 2;`

Условная операция. Операция instanceof

- ***Условная операция*** возвращает одно из двух значений в зависимости от заданного условия

```
int a = (10<20)? 1 : 2;
```

- ***Операция instanceof*** проверяет принадлежность объекта заданному типу (классу)

```
String s = "Hello";
if (s instanceof java.lang.String)
System.out.println("s is a String");
```

Выражения и приоритет операций

- ***Выражение*** (***expression***) состоит из операндов и операций
- Операции выполняются в соответствии с их приоритетами
- Операции с одинаковым приоритетом выполняются в порядке:
  - (1-13) справа налево
  - (14) слева направо
- Для обозначения приоритетов операций могут использоваться круглые скобки

- Операнд:
  - константа
  - переменная (объект)
  - вызов метода
  - выражение

PRI	Операция
1	x++ x--
2	++x --x +x -x ~ !
3	* / %
4	+ -
5	<< >> >>>
6	< > <= >= instanceof
7	== !=
8	&
9	^
10	|
11	&&
12	||
13	?:
14	= += -= *= /= %= &= ^= |= <<= >>= >>>=

```
int a = 10 + 5 * 2 - 7;
double z = Math.sqrt(25) + a * (10 - 2);
boolean b =  3 > 7 || 4 > 0 && 2 == 2;
```

3.2. ОПЕРАТОРЫ

Операторы

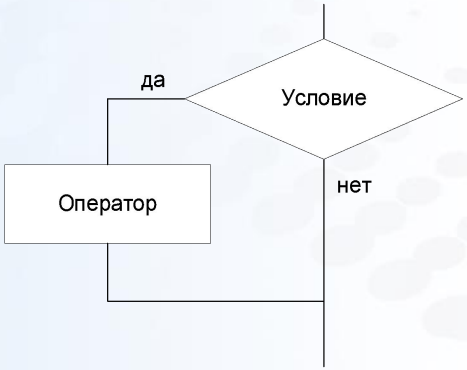
- Операторные конструкции:
  - объявление переменной
  - присваивание
  - инкремент/декремент
  - создание объекта
  - вызов методов

```
int a;
a = 5;
a++;
new HelloWorld();
System.out.println("Hi!");
```

- Условный оператор ***if***
- Условный оператор ***switch***
- Оператор цикла ***while***
- Оператор цикла ***for***
- Оператор ***break***
- Оператор ***continue***
- Оператор ***return***
- Составной оператор (блок)

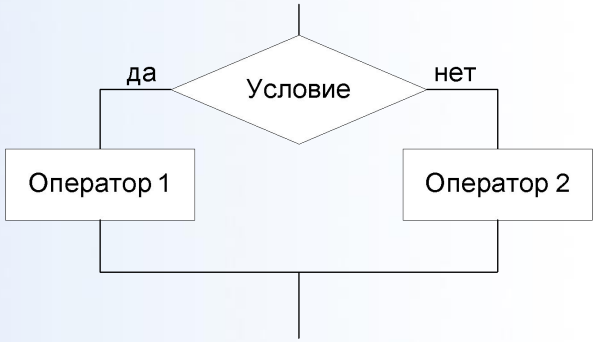
Условный оператор *if*

```
if (условное_выражение)
      оператор;
```



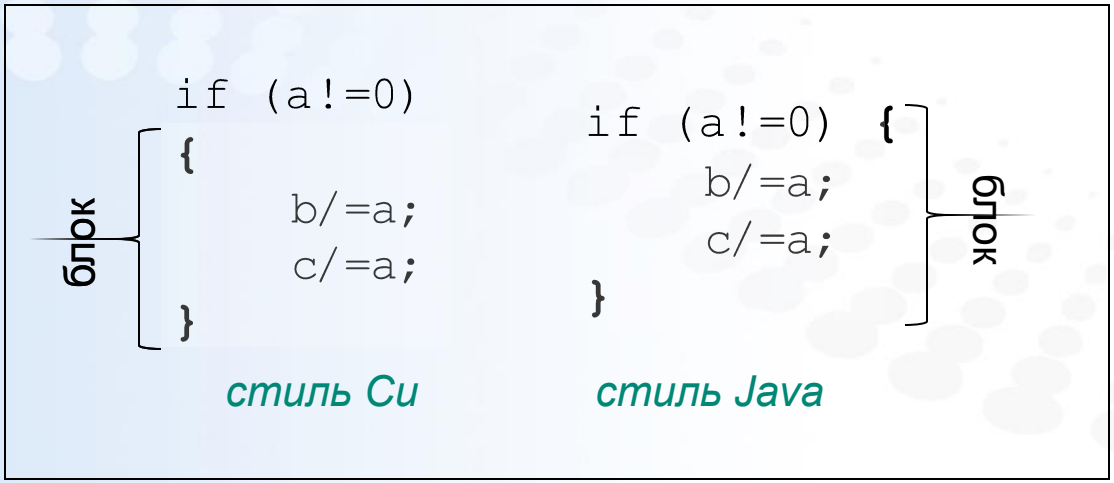

```
if (a!=0) b=b/a;

if (a!=0 && b!=0)
    c=c/(a*b);

if (a!=0)
    if (b!=0) c=c/(a*b);
```

```
if (условное_выражение)
      оператор1;
else
      оператор2;
```




```
if (a<b)
    min = a;
else
    min = b;


if (a>b && a>c)
    max = a;
else if (b>c)
    max = b;
else
    max = c;
```

Составной оператор (блок)




```
if (a!=0) {
     b/=a;
     c/=a;
}
else {
     b=0;
     c=0;
}
```

Условный оператор *switch*

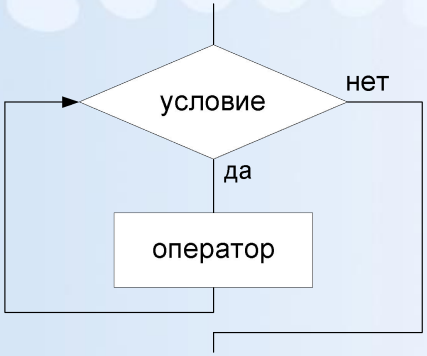
```
int day=3;
String name;
switch (day) {
    case 1: name = "ПН";
        break;
    case 2: name = "ВТ";
        break;
    case 3: name = "СР";
        break;
    case 4: name = "ЧТ";
        break;
    case 5: name = "ПТ";
        break;
    case 6: name = "СБ";
        break;
    case 7: name = "ВС";
        break;
    default: name = "???";
}
```

```
int x=3;
int y=0;
switch (x) {
    case 1:  y++;
    case 2:  y++;
    case 3:  y++;
    case 4:  y++;
    case 5:  y++;
    default: y++;
}
// y=4
```

```
int day=3;
String type;
switch (day) {
    case 1: case 2:
    case 3: case 4:
    case 5:
        type = "Будни";
        break;
    case 6: case 7:
        type = "Выходные";
        break;
    default:
        type = "???";
}
```

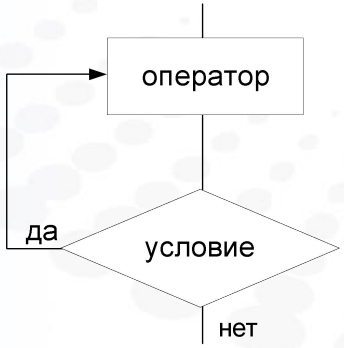
Оператор цикла *while*

```
while (условное_выражение)
    оператор;
```




```
int i = 1;
while (i < 6) {
    System.out.println("Счетчик: " + i);
    i++;
}
```

```
do
    оператор;
while (условное_выражение);
```



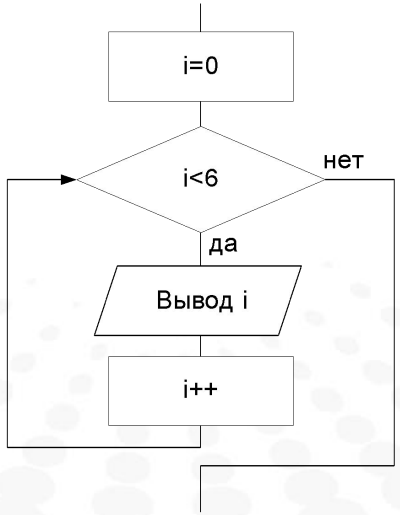

```
int i = 1;
do {
    System.out.println("Счетчик: " + i);
    i++;
}
while (i < 6);
```

```
Счетчик: 1
Счетчик: 2
Счетчик: 3
Счетчик: 4
Счетчик: 5
```

Оператор цикла *for*

```
for (инициализация; условие; модификатор)
    оператор;
```

```
for (int i=0; i<6; i++)
    System.out.println("Счетчик: " + i);
```




```
Счетчик: 1
Счетчик: 2
Счетчик: 3
Счетчик: 4
Счетчик: 5
```

```
for (int i=2; i<=10; i++)
    for (int j=2; j<=10; j++) {
        int mult = i*j;
        System.out.println(i+" x "+j+" = "+mult);
    }
```

```
for (итератор)
    оператор;
```

```
int[] numbers = {1,2,3,4,5};
for (int i : numbers)
    System.out.println("Счетчик: " + i);
```

Операторы *break, continue, return*

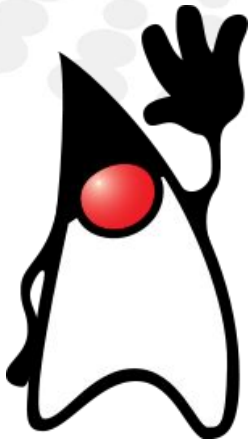
```
int[] numbers = {1, 3, 5, 10, 15};
int toFind = 5;
boolean find = false;
for (int i : numbers)
    if (i == toFind) {
        find = true;
        break;
    }
```

```
int[][] numbers = { {1, 3, 5},
                    {10, 15, 22} };
int toFind = 3;
boolean find = false;

search:
    for (int[] line : numbers) {
        for (int j : line) {
            if (j == toFind) {
                find= true;
                break search;
            }
        }
    }
```

```
int[] numbers = {1,2,3,4,6,12,15};
int sum=0;

for (int i : numbers) {
    if (i%2!=0) continue;
    sum+=i;
}
System.out.println(
        "Сумма четных чисел: "+sum);
```

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

Учебный компьютерный Центр обучения «СПЕЦИАЛИСТ» при МГТУ им. Н.Э.Баумана



4. СТАНДАРТНЫЕ ТИПЫ JAVA

Учебный центр компьютерного обучения «СПЕЦИАЛИСТ» при МГТУ им. Н.Э.Баумана

4.1. СОЗДАНИЕ И ИСПОЛЬЗОВАНИЕ МАССИВОВ

Массивы

- ***Массив*** (***array***) – множество однотипных ***элементов***
  - Размер массива определяется при создании массива
  - Размер массива не может быть измен
- Обращение к элементам осуществляется по индексу (начиная с нуля)

- Объявление массива

```
int[] arrayOfInt;
```

- Создание массива

```
arrayOfInt = new int[3];
```

- Обращение к элементам массива

```
arrayOfInt[0] = 10;
arrayOfInt[1] = 20;
arrayOfInt[2] = arrayOfInt[0]
  + arrayOfInt[1];
```

- Сокращенная форма

```
int[] arrayOfInt2 = {5,2,99};
```

Типовые операции с массивами (1)

- Проход по массиву

```
int[] arr = new int[10];
for (int i=0; i<arr.length; i++)
     arr[i] = i*10;
for (int item : arr)
     System.out.println(item);
```

- Копирование массива

```
int[] arr2;
arr2 = arr;
arr2 = new int[arr.length];
for (int i=0; i<arr.length; i++)
     arr2[i] = arr[i];

int[] arr3 = new int[arr.length];
System.arraycopy(arr, 0,
                 arr3, 0, arr.length);
```

- Сортировка массива

```
int[] arr5 = {10, 3, 7, 15, 0, 2, 16};
java.util.Arrays.sort(arr5);
```

- Заполнение массива

```
double[] arr6 = new double[10];
java.util.Arrays.fill(arr6, 0.5);
```

- Создание массива объектов

```
Car[] carArray = {new Car(),
                  new Car(), new Car()};

Car[] carArray2 = new Car[100];
for (int i=0; i<carArray2.length; i++)
     carArray2[i] = new Car();

System.out.println(carArray2[0].getName());
```

```
int[] arr4 = java.util.Arrays.copyOf(arr, arr.length);
```

Типовые операции с массивами (2)

- Преобразование в строку

```
int arr7[] = {1,2,3,4,5,6,7};
System.out.println(
        java.util.Arrays.toString(arr7));
```

- Сравнение массивов на равенство элементов

```
int arr8[] = {1,2,3,4,5,6,7};
if (Arrays.equals(arr7, arr8))
     System.out.println("Массивы равны");

if (arr7 == arr8) ...
```

Многомерные массивы

```
int[][] mdArr1 = { {1,2,3}, {4,5,6} };
int[][][] mdArr2 = { { {1,2}, {3,4} },
                     { {5,6}, {7,8} }};
System.out.println(mdArr1[1][0]);    // 4
System.out.println(mdArr2[1][0][1]); // 6
```

```
int[][] mdArr3 = new int[4][];
for (int i=0; i<mdArr3.length; i++)
     mdArr3[i] = new int[7];

for (int i=0; i<mdArr3.length; i++)
     for (int j=0; j<mdArr3[i].length; j++)
          mdArr3[i][j] = i*j;
```

```
for (int[] line : mdArr3)
{
     for (int item : line)
          System.out.print(item + " ");
     System.out.println("");
}
```

```
0 0 0 0 0 0 0
0 1 2 3 4 5 6
0 2 4 6 8 10 12
0 3 6 9 12 15 18
```

- ***Многомерный массив*** – массив, элементами которого являются массивы

4.2. РАБОТА СО СТРОКАМИ

Строки
Типовые операции со строками

- ***java.lang.String*** – класс, описывающий строку
- Объекты класса String являются неизменяемыми **(!)**
- Объявление и создание строки

```
String s1 = "Привет, мир";
char[] arr = {'П','р','и','в','е','т'};
String s2 = new String(arr);
```

- Определение длины строки

```
int len = s1.length(); // 11
```

- Работа с символами

```
for (int i=0; i<len; i++)
    if (s1.charAt(i) == 'и')
        System.out.println(
            "Символ \'и\' на позиции " + i);
```

```
Символ 'и' на позиции 2
Символ 'и' на позиции 9
```

Типовые операции со строками

- Конкатенация строк

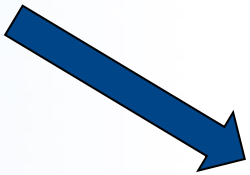
```
String s3 = "Привет";
String s4 = "мир";
String s5 = s3.concat(" ").concat(s4);
System.out.println(s5); // Привет мир

String s6 = s3 + " " + s4;
```

- Сравнение строк

```
if (s5.equals(s6))
     System.out.println("Строки равны");

String s7 = "Иванов";
String s8 = "Сидоров";
int x = s7.compareTo(s8); // -9
int y = s8.compareTo(s7); // 9
```

Форматирование строк

```
String s = "%s двух чисел %d и %d равна %d";
int a = 10;
int b = 6;
String formatString1 = String.format(s, "Сумма", a, b, a+b);
String formatString2 = String.format(s, "Разность", a, b, a-b);
System.out.println(formatString1);
System.out.println(formatString2);
```



```
Сумма двух чисел 10 и 6 равна 16
Разность двух чисел 10 и 6 равна 4
```

Преобразование в строку

- Через конкатенацию

```
int a = 10;
String s1 = "" + a;
```

- Через статический метод String.***valueOf***

```
double b = 30.5;
boolean c = true;
String s2 = String.valueOf(b);
String s3 = String.valueOf(c);
```

- Через метод java.lang.Object.***toString***

```
Integer val1 = new Integer(10);
Boolean val2 = new Boolean(true);
String sv1 = val1.toString();
String sv2 = val2.toString();
```

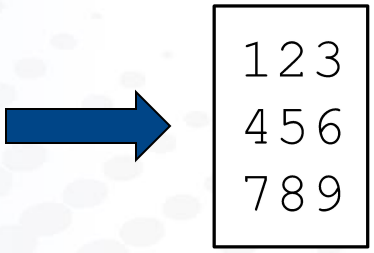
```
System.out.println(s1);
System.out.println(s2);
System.out.println(s3);
```

```
10
30.5
true
```

Использование регулярных выражений

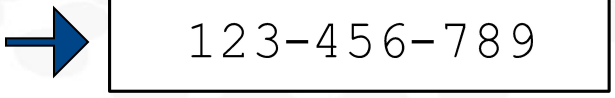
- Разбиение строки на элементы

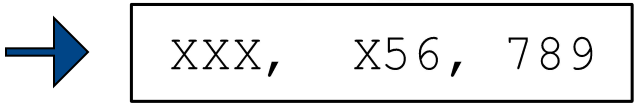
```
String s = "123,  456, 789";
String[] arr = s.split("\\s*,\\s*");
for (String item : arr)
    System.out.println(item);
```



- Замена элементов в строке

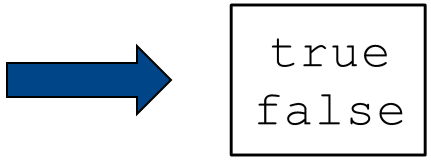
```
String s2 = s.replaceAll("\\s*,\\s*", "-");
System.out.println(s2);
  String s3 = s.replaceAll("[1-4]", "X");
  System.out.println(s3);
String s4 = s.replaceFirst(",", "ABC");
System.out.println(s4);
```





- Проверка строк на соответствие шаблону

```
String phone1 = "123-45-67";
String phone2 = "12-34-567";
String pattern = "\\d{3}-\\d{2}-\\d{2}";
System.out.println(phone1.matches(pattern));
System.out.println(phone2.matches(pattern));
```



Строки. Прочие операции

`String` **`substring`**`(int ind1, int ind2)` `String` **`substring`**`(int beginIndex)`	Возвращает подстроку между заданными номерами символов / начиная с заданного номера символа
`String` **`trim`**`()`	Возвращает строку без лидирующих и завершающих пробелов
`String` **`toLowerCase`**`()` `String` **`toUpperCase`**`()`	Возвращает строку в нижнем/верхнем регистре

`int` **`indexOf`**`(int ch)`
`int` **`lastIndexOf`**`(int ch)`
`int` **`indexOf`**`(String str)`
`int` **`lastIndexOf`**`(String str)`
	Возвращает индекс первого/последнего вхождения в строку символа `ch` / подстроки `str`

`boolean` **`contains`**`(CharSequence s)`	Проверяет, содержит ли строка заданную последовательность символов
`String` **`replace`**`(char oldC, char newC)` `String` **`replace`**`(CharSequence oldC, CharSequence newC)`	Возвращает строку, в которой произведена замена символа/последовательности символов `oldC` на `newC`
`boolean` **`equalsIgnoreCase`**`(String s)` `int` **`compareToIgnoreCase`**`(String str)`	Сравнивает сроки без учета регистра

Класс Scanner

- ***java.util.Scanner*** – текстовый сканнер, использующий регулярные выражения

```
String s = "10 20 45 77 19 abc 30";
Scanner sc = new Scanner(s);
while (sc.hasNext())
      System.out.println(sc.next());
sc.close();
```

```
10
20
45
77
19
abc
30
```

```
sc = new Scanner(s);
while (sc.hasNextInt()) {
      int k = sc.nextInt();
      System.out.print(k + " ");
}
sc.close();
System.out.println();
```

```
10 20 45 77 19
```

```
String s2 = "1 10 ab 30 abc 25 40 25 70 25";
sc = new Scanner(s2);
sc.useDelimiter("\\s*ab\\s*");
while (sc.hasNext())
      System.out.println(sc.next());
sc.close();
```

```
1 10
30
c 25 40 25 70 25
```

Класс StringBuilder

- Класс ***java.lang.StringBuilder*** используется для работы с изменяемыми строками

```
StringBuilder sb = new StringBuilder("Однажды");
sb.append(" в студеную");
sb.append(" зимнюю");
sb.append(" пору. ");
sb.append(999);
sb.append(" ");
sb.append(3.14);

System.out.println(sb);
System.out.println(sb.length());

sb.delete(9, 18);
System.out.println(sb);

String s1 = sb.toString();
String s2 = sb.substring(10, 13);
System.out.println(s2);
```

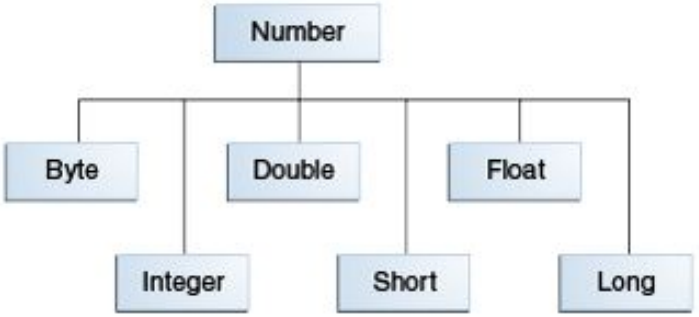
```
Однажды в студеную зимнюю пору. 999
3.14
40
Однажды в зимнюю пору. 999 3.14
зим
```

4.3. ОБЪЕКТНЫЕ ОБОЛОЧКИ ПРИМИТИВНЫХ ТИПОВ

Объектные оболочки примитивных типов

- ***java.lang.Number***
  - абстрактный класс «Число»
  - родительский класс для оболочек примитивных типов
- Потомки java.lang.Number
  - ***java.lang.Byte***
  - ***java.lang.Short***
  - ***java.lang.Integer***
  - ***java.lang.Long***
  - ***java.lang.Float***
  - ***java.lang.Double***

- Оболочки используются:
  - если вызываемый метод ожидает на вход объектную ссылку, а не значение примитивного типа
  - для преобразования значений
  - для получения специальных констант




```
Integer a = new Integer(10);
Integer b = 15;
Integer c = a + b;
System.out.println(c);
```

Основные методы оболочек

- Преобразование

```
byte byteValue()
short shortValue()
int intValue()
long longValue()
float floatValue()
double
doubleValue()
String toString()
```

- Сравнение

```
boolean equals(Object
obj)
int compareTo(...)
```

```
Integer x = 10;
Long y = x.longValue();
Long z = new Long(15);
if (x.equals(y))
     System.out.println("x = y");
System.out.println(z);


Long q = 12L;
int res = q.compareTo(z);
System.out.println(res); // -1

res = q.compareTo(y);
System.out.println(res); // 1

res = q.compareTo(q);
System.out.println(res); // 0
```

Статические поля и методы класса Integer

- Статические методы

```
static Integer decode(String s)
static int parseInt(String s)
static int parseInt(String s, int radix)
static String toString(int i)
static String toHexString(int i)
static String toBinaryString(int i)
static Integer valueOf(int i)
static Integer valueOf(String s)
static Integer valueOf(String s, int radix)
```

- Статические поля

```
static int MIN_VALUE
static int MAX_VALUE
static int SIZE
```

```
System.out.println(Integer.SIZE);       // 4
System.out.println(Integer.MIN_VALUE); // -2147483648

Integer a = Integer.decode("100");
Integer b = Integer.decode("0xFF");
System.out.println(a); // 100
System.out.println(b); // 255

int c = Integer.parseInt("500");        // c = 500
int d = Integer.parseInt("1010",2);     // d = 10

Integer e = Integer.valueOf("999");
String s  = Integer.toBinaryString(13); // s = "1101"
```

Учебный центр компьютерного обучения «СПЕЦИАЛИСТ» при МГТУ им. Н.Э.Баумана

4.4. МАТЕМАТИЧЕСКИЕ ФУНКЦИИ

Математические функции (1)

- ***java.lang.Math***

Работа с числами (static)

`double abs(double d)`
`float abs(float f)`
`int abs(int i)`
`long abs(long lng)`
	Взятие модуля числа	`double val = Math.abs(-15.3); // 15.3`

`double ceil(double d)`	Округление в большую сторону	`val = Math.ceil(20.1); // 21.0`
`double floor(double d)`	Округление в меньшую сторону	`val = Math.floor(20.99); // 20.0`

`double rint(double d)`
`long round(double d)`
`int round(float f)`
	Округление до ближайшего целого	`val = Math.rint(20.49); // 20.0`
		`val = Math.rint(20.51); // 21.0`

`double min(double a, double b)`
`float min(float a, float b)`
`int min(int a, int b)`
`long min(long a, long b)`
	Определение минимального значения	`val = Math.min(10., 20.); // 10.0`

`double max(double a, double b)`
`float max(float a, float b)`
`int max(int a, int b)`
	Определение максимального значения	`val = Math.max(10., 20.); // 20.0`

Математические функции (2)

Математические вычисления (static)

`double` **`exp`**`(double d)`	Функция экспоненты
`double` **`log`**`(double d)`	Натуральный логарифм
`double` **`pow`**`(double a, double b)`	Возведение числа **a** в степень **b**
`double` **`sqrt`**`(double d)`	Квадратный корень
`double` **`sin`**`(double d)`	Синус
`double` **`cos`**`(double d)`	Косинус
`double` **`tan`**`(double d)`	Тангенс
`double` **`asin`**`(double d)`	Арксинус
`double` **`acos`**`(double d)`	Арккосинус
`double` **`atan`**`(double d)`	Арктангенс
`double` **`toDegree`**`s(double d)`	Преобразование радиан в градусы
`double` **`toRadians`**`(double d)`	Преобразование градусов в радианы
`double` **`random`**`()`	Генерация случайной величины в диапазоне [0;1)

Константы (static)

`double` **`PI`**	Число Пи
`double` **`E`**	Число Эйлера

Учебный центр компьютерного обучения «СПЕЦИАЛИСТ» при МГТУ им. Н.Э.Баумана

4.5. РАБОТА С ДАТОЙ И ВРЕМЕНЕМ

Дата и время
Класс Date. Форматирование

- ***java.util.Date*** – класс для представления даты и времени
- ***java.text.SimpleDateFormat*** – класс для форматирования даты и времени

```
Date d1 = new Date(); // Текущие дата и время
Date d2 = new Date(0); // 01.01.1970 0:00:00 GMT
System.out.println(d2.toString());

Date d3 = new Date(31L*24*60*60*1000); // 01.02.1970
System.out.println(d3.toString());

long mills = d3.getTime(); // Количество мс с 01.01.1970
System.out.println(mills);

SimpleDateFormat sdf = new SimpleDateFormat(
                "Сегодня dd.MM.yyyy, сейчас HH:mm");
System.out.println(sdf.format(d1));

SimpleDateFormat sdf2 = new SimpleDateFormat("dd/MM/yyyy");
if (d2.before(d3))
     System.out.println(sdf2.format(d2) +
            " перед " + sdf2.format(d3));
```

```
Thu Jan 01 04:00:00 GMT+04:00 1970
Sun Feb 01 04:00:00 GMT+04:00 1970
2678400000
Сегодня 12.03.2013, сейчас 15:57
01/01/1970 перед 01/02/1970
```

Класс Calendar

- ***java.util.Calendar*** – абстрактный класс для управления датой
- ***java.util.GregorianCalendar*** – стандартный календарь, дочерний класс *Calendar*

```
SimpleDateFormat sdf= new SimpleDateFormat("d MMMM y, HH:mm");
Calendar cal = Calendar.getInstance();

cal.set(2013, 1, 15, 0, 0, 0);
Date dt = cal.getTime();
System.out.println(sdf.format(dt));

cal.set(Calendar.MONTH, Calendar.AUGUST);
System.out.println(sdf.format(cal.getTime()));

cal.set(Calendar.YEAR, 2020);
System.out.println(sdf.format(cal.getTime()));

cal.set(Calendar.HOUR_OF_DAY, 22);
System.out.println(sdf.format(cal.getTime()));

System.out.println(cal.getTimeInMillis());

int month = cal.get(Calendar.MONTH);
System.out.println(month);
```

```
15 Февраль 2013, 00:00
15 Август 2013, 00:00
15 Август 2020, 00:00
15 Август 2020, 22:00
1597514400725
7
```

4.6. КЛАСС SYSTEM

Класс System

- ***java.lang.System***

`static PrintStream out`	Поток стандартного вывода
`static InputStream in`	Поток стандратного ввода
`static PrintStream err`	Поток стандартного вывода ошибок

`static void exit(int status)`	Прерывает выполнение JVM
`static void gc()`	Запускает сборщик мусора
`static String getenv(String name)`	Возвращает значение переменной окружения
`static long currentTimeMillis()`	Возвращает текущее время в миллисекундах
`static long nanoTime()`	Возвращает время в наносекундах
`static String getProperty(String k)`	Читает системное свойство
`static void setOut(PrintStream out)`	Перенаправляет стандартный вывод
`static void setIn(InputStream in)`	Перенаправляет стандартный ввод
`...`	

Стандартный ввод-вывод

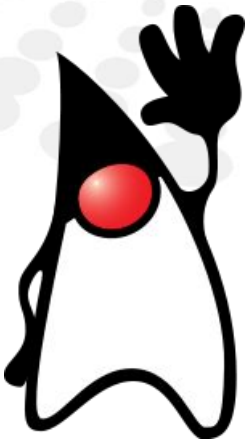
```
System.out.println("Hello, world");
System.out.println(System.getenv("TEMP"));
System.out.println(System.getProperty("os.name"));
System.out.printf("Сумма чисел %d и %d равна %d", 2, 7, (2+7));
System.err.println("Это вывод ошибки");
```

```
Hello, world
C:\Users\E786~1\AppData\Local\Temp
Windows 7
Сумма чисел 2 и 7 равна 9
Это вывод ошибки
```

```
Введите число :
23
Введено число : 23
```

```
System.out.println("Введите число: ");
Scanner sc = new Scanner(System.in);
int value = sc.nextInt();
System.out.println("Введено число: " + value);
```

Центр компьютерного обучения «СПЕЦИАЛИСТ» при МГТУ им. Н.Э.Баумана



5. РАЗРАБОТКА КЛАССОВ

Объявление классов

- ***Публичные*** (***public***) классы
  - объявляются со спецификатором ***public***
  - доступны внутри и вне пакета
  - допускается наличие только одного публичного класса в файле
- ***Непубличные (package private) классы***
  - объявляются без спецификатора доступа
  - доступны только внутри пакета
- Рекомендации по именованию классов:
  - Имя класса состоит из одного или нескольких слов
  - Первая буква каждого слова заглавная, остальные буквы – в нижнем регистре

```
public class Car {
    String name;
    int speed;

    void setName(String newName) {
        name = newName;
    }
    void speedUp(int delta) {
        speed = speed + delta;
    }
    void applyBrakes(int delta) {
        speed = speed - delta;
    }
    int getSpeed() {
        return speed;
    }
}

class Point
{
     int x;
     int y;
}
```

Элементы класса

- Элементы класса (***class members***):
  - ***Поля (fields)***
  - ***Методы (methods)***
- Спецификаторы доступа:
  - ***public*** – элемент класса является общедоступным
  - ***private*** – элемент доступен только методам класса
  - ***protected*** – элемент доступен только методам класса и дочерних классов
  - (не задан) – package-private – элемент доступен внутри пакета
- Рекомендуется скрывать поля класса от внешнего доступа

Спец.	Класс	Пакет	Потомок	Внешн
public	Y	Y	Y	Y
protected	Y	Y	Y	N
-	Y	Y	N	N
private	Y	N	N	N

```
public class Car {
    private String name;
    private int speed;

    public void setName(String newName) {
        name = newName;
    }
    public String getName() {
        return name;
    }
    public void speedUp(int delta) {
        speed = speed + delta;
    }
    public void applyBrakes(int delta) {
        speed = speed - delta;
    }
    public int getSpeed() {
        return speed;
    }
}
```

Обращение к элементам класса

```
public class Car{
    private String name;
    private int speed;

    public void setName(String newName) {
      name = newName;
    }
    public String getName() {
      return name;
    }
    ...
```

```
public class HelloWorld{
     public static void main(String[] args) {
          Car myLittleCar = new Car();
          myLittleCar.name = "Жигули";
          myLittleCar.setName("Жигули");
          System.out.println(myLittleCar.name);
          System.out.println(myLittleCar.getName());
     }
}
```

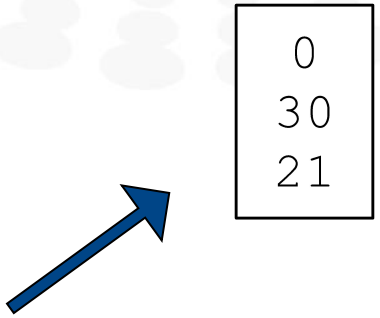
Перегрузка методов

```
public class Car {
    private String name;
    private int speed;

    public void setName(String newName) {
        name = newName;
    }
    public String getName() {
        return name;
    }
    public void speedUp(int delta) {
        speed = speed + delta;
    }
    public void applyBrakes(int delta) {
        speed = speed - delta;
    }
    public void applyBrakes() {
     speed = 0;
    }
    public int getSpeed() {
        return speed;
    }
}
```

```
public class HelloWorld{
  public static void main(String[] args) {
    Car myLittleCar = new Car();
    myLittleCar.speedUp(10);
    System.out.println(
      myLittleCar.getSpeed()); // 10
    myLittleCar.speedUp(15);
    System.out.println(
      myLittleCar.getSpeed()); // 25
    myLittleCar.applyBrakes(5);
    System.out.println(
      myLittleCar.getSpeed()); // 20
    myLittleCar.applyBrakes();
    System.out.println(
      myLittleCar.getSpeed()); // 0
  }
}
```

Методы с переменным числом аргументов

```
public class MyClass {
    int sum(int... x) {
        int result = 0;
        for (int element : x)
            result += element;
        return result;
    }
}

public class HelloWorld {
    public static void main(String[] args) {
        MyClass obj = new MyClass();
        System.out.println(obj.sum());
        System.out.println(obj.sum(24,6));
        System.out.println(obj.sum(1,2,3,4,5,6));
    }
}
```



Конструкторы

- ***Конструктор*** (***constructor***)
  - метод, предназначенный для инициализации объекта
  - вызывается автоматически при создании объекта
  - не возвращает значения
  - может принимать на вход параметры
  - может быть перегружен
  - ***конструктор по умолчанию*** (***default constructor***) – конструктор, не принимающий на вход параметров

```
public class Car{
    private String name;
    private int speed;

    public Car() {
     name = "Жигули";
     speed = 0;
    }
    public Car(String iName, int iSpeed){
     name = iName;
     speed = iSpeed;
    }
    ...
```

```
public class HelloWorld{
  public static void main(String[] args) {
    Car car1 = new Car();
    Car car2 = new Car("Volvo",90);
    System.out.println(car1.getName());
    System.out.println(car2.getName());
  }
}
```

Особенности использования конструкторов

```
public class Point{
     public int x = 0;
     public int y = 0;
}
```

```
public class HelloWorld{
     public static void main(String[] args) {
          Line l1 = new Line(0,0,3,4);
          System.out.println(l1.getLength());
          Line l2 = new Line();
          System.out.println(l2.getLength());
     }
}
```

```
public class Line{
     private Point point1;
     private Point point2;

     public Line()  {
          point1 = new Point();
          point2 = new Point();
     }
     public Line(int x1, int y1, int x2, int y2) {
          point1 = new Point();
          point2 = new Point();
          point1.x = x1; point1.y = y1;
          point2.x = x2; point2.y = y2;
     }
     public double getLength() {
          return Math.sqrt(
                    Math.pow(point1.x-point2.x, 2) +
                    Math.pow(point1.y-point2.y, 2));
     }
}
```

Ссылка *this*

- ***this*** – ссылка на текущий объект класса

```
public class Point{
     public int x = 0;
     public int y = 0;
     public void setCoord(int x, int y) {
          this.x = x;
          this.y = y;
     }
}
```

```
public class Line{
     private Point point1;
     private Point point2;

     public Line()  {
          point1 = new Point();
          point2 = new Point();
     }
     public Line(int x1, int y1, int x2, int y2)  {
          this();
          point1.x = x1; point1.y = y1;
          point2.x = x2; point2.y = y2;
     }
     public double getLength() {
          return Math.sqrt(
                    Math.pow(point1.x-point2.x, 2) +
                    Math.pow(point1.y-point2.y, 2));
     }
}
```

Статические поля

- ***Статические поля (static fields)*** являются общими для всех объектов класса

```
public class Point{
     public static int pointsCreated = 0;
     public int x;
     public int y;
     public Point(int x, int y)    {
          pointsCreated++;
          this.x = x;
          this.y = y;
     }
}
public class PointDemo{
     public static void main(String[] args) {
          Point p1 = new Point(99,99);
          Point[] ptri = {new Point(0,0), new Point(10,0),
                          new Point(10,10)};
          System.out.println(p1.x); // 99
          System.out.println(ptri[1].x); // 10
          System.out.println(Point.x);
          System.out.println(Point.pointsCreated); // 4
          System.out.println(p1.pointsCreated); // 4
     }
}
```

Статические методы

- ***Статические методы*** (***static methods***)
  - Вызываются для класса, а не для конкретного объекта
  - Могут обращаться напрямую только к статическим элементам класса

```
public class Point{
     private static int pointsCreated = 0;
     public int x;
     public int y;
     public Point(int x, int y)    {
          pointsCreated++;
          this.x = x;
          this.y = y;
     }
     public static int getPointsCount() {
          return pointsCreated;
     }

     public static void main(String [] arguments) {
          Point[] ptri = {new Point(0,0), new Point(10,0),
                          new Point(10,10)};
          System.out.println(Point.getPointsCount()); // 3
     }
}
```

Спецификатор final

- Спецификатор ***final*** используется для объявления переменных, не меняющих свое значение

```
public class HelloWorld{
     public static final double PI = 3.1415;
     public static void main(String[] args) {
          final double E = 2.7182;
          System.out.println("Число пи: " + HelloWorld.PI);
          System.out.println("Число Эйлера: " + E);
     }
}
```

```
public class Point{
     public final int x;
     public final int y;

     public Point(int x, int y) {
          this.x = x;
          this.y = y;
     }

     public static void main(String [] arguments) {
          Point p = new Point(2,3);
          p.x=2;
     }
}
```

Инициализация полей (1)

(1)
```
public class Point{
    private static int pointsCreated= 0;
    public int x = 0;
    public int y = 0;
    ...
```

---

(2)
```
public class Point{
    private static int pointsCreated;
    public int x;
    public int y;

    static {
        pointsCreated= 0;
        System.out.println("Инициализирую статическое поле");
    }

    public Point() {
        pointsCreated++;
        x = 0;
        y = 0;
        System.out.println("Инициализирую объект");
    }
```

Инициализация полей (2)

(3)
```
public class Point{
    private static int pointsCreated= initStatV();

    public int x = initInstV();
    public int y = initInstV();

    private static int initStatV() {
        System.out.println("Инициализирую статическое поле");
        return 0;
    }

    private int initInstV() {
        System.out.println("Инициализирую поле объекта");
        return 0;
    }
      ...
```

---

(4)
```
public class Point{
    public int x;
    public int y;

    {
        System.out.println("Инициализирую объект");
        x=0;
        y=0;
    }
      ...
```

Передача параметров в методы

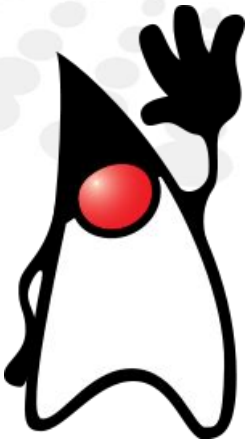
- Параметры передаются в методы ***по значению*** (***by value***)
  - в метод передается копия значения:
  - исходные переменные сохраняют вои значения
  - объекты, на которые ссылаются параметры, могут быть изменены

```
public class HelloWorld{
  public static void main(String[] args) {
    int value = 100;
    MethodTest.workWithInt(value);
    System.out.println(value); // 100

    Point pnt = new Point(10,20);
    MethodTest.workWithPoint1(pnt);
    pnt.print();  // 11;21
    MethodTest.workWithPoint2(pnt);
    pnt.print();  // 11;21

    int[] q = {1,2};
    MethodTest.workWithArray(q);
    System.out.println(q[0]+" "+q[1]); // 2 3
  }
}
```

```
public class Point{
  public int x;
  public int y;
  public Point() {x=0; y=0;}
  public Point(int nx, int ny) {x=nx; y=ny;}
  public void print() {
    System.out.println(x + ";" + y);
  }
}
```

```
public class MethodTest{
  public static voidworkWithInt(int x) {
    x = x + 1;
  }
  public static voidworkWithPoint1(Point p){
    p.x = p.x + 1;
    p.y = p.y + 1;
  }
  public static voidworkWithPoint2(Point p){
    p = new Point();
    p.x = 111;
    p.y = 222;
  }
  public static voidworkWithArray(int[] arr) {
    for (int i=0; i<arr.length; i++)
      arr[i]++;
  }
}
```

Учебный центр компьютерного обучения «СПЕЦИАЛИСТ» при МГТУ им. Н.Э.Баумана

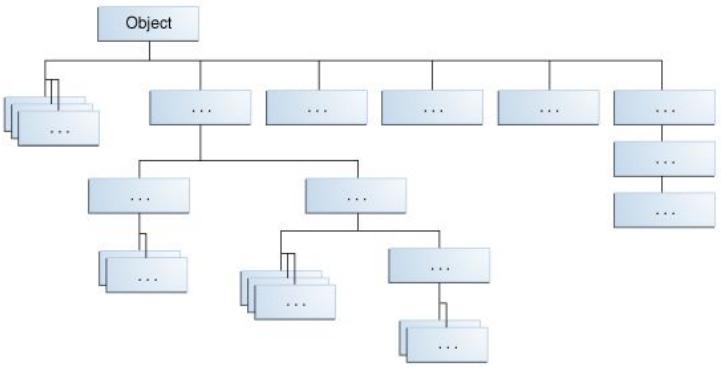


6. НАСЛЕДОВАНИЕ И ПОЛИМОРФИЗМ

Наследование

- ***Наследование*** (***inheritance***)
- У класса может быть один родитель и любое количество дочерних классов
- Прародителем всех классов Java является класс ***java.lang.Object***
- Дочернему классу передаются поля и методы родительского класса
- Дочерний класс может обращаться полям и методам родительского класса, которые:
  - объявлены со спецификатором ***public*** или ***protected***
  - объявлены без спецификатора (*package private*), при условии что дочерний класс находится в одном пакете вместе с родительским
- Дочерний класс может иметь свои собственные поля и методы, а также переопределять методы родительского класса

- Родительский класс (parent class)
  - Суперкласс (superclass)
  - Базовый класс (base class)
- Дочерний класс (child class)
  - Подкласс (subclass)
  - Наследуемый класс (derived class)

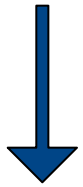


Объявление дочерних классов

```
public class Point {
  private double x = 0;
  private double y = 0;
  public double getX() {
    return x;
  }
  public double getY() {
    return y;
  }
  public void setCoord(double nx,
                       double ny) {
    x = nx; y = ny;
  }
}
```

```
public class Point3D extends Point {
  private double z = 0;
  public double getZ() {
    return z;
  }
  public void setCoord(double nx,
                  double ny, double nz) {
    z = nz;
    setCoord(nx,ny);
  }
}
```

```
public class HelloWorld {
  public static void main(String args[]) {
    Point p2d = new Point();
    p2d.setCoord(10, 20);
    System.out.println(p2d.getX() +
                       " " + p2d.getY());

    Point3D p3d = new Point3D();
    p3d.setCoord(10, 20, 30);
    System.out.println(p3d.getX() + " " +
            p3d.getY() + " " + p3d.getZ());
    p3d.setCoord(15, 24);
    System.out.println(p3d.getX() + " " +
            p3d.getY() + " " + p3d.getZ());
  }
}
```



```
10.0 20.0
10.0 20.0 30.0
15.0 24.0 30.0
```

Использование элементов *protected*

```
public class Point {
      protected double x = 0;
      protected double y = 0;
      public double getX() {
            return x;
      }
      public double getY() {
            return y;
      }
      public void setCoord(double nx, double ny) {
            x = nx; y = ny;
      }
}
            public class Point3D extends Point {
                  protected double z = 0;
                  public double getZ() {
                        return z;
                  }
                  public void setCoord(double nx, double ny, double nz) {
                        x = nx; y = ny; z = nz;
                  }
            }
```

Переопределение методов

- Методы родительского класса могут быть переопределены в дочернем классе
- Для обращения к переопределенным элементам родительского класса из дочернего класса используется ключевое слово ***super***

```
public class Point {
  ...
  public void print() {
    System.out.print(x + ", " + y);
  }
}

public class Point3D extends Point {
  ...
  @Override
  public void print() {
    System.out.print("[");
    super.print();
    System.out.print(", " + z + "]");
  }
}
```

```
public class HelloWorld{
  public static void main(
                    String args[]) {
    Point p2d = new Point();
    p2d.setCoord(10, 20);
    p2d.print();

    Point3D p3d = new Point3D();
    p3d.setCoord(10, 20, 30);
    p3d.print();
    p3d.setCoord(15, 24);
    p3d.print();
  }
}
```

```
10.0, 20.0[10.0, 20.0, 30.0][15.0, 24.0, 30.0]
```

Особенности работы со ссылками

- Переменная-ссылка, относящаяся к родительскому классу, может ссылаться на объекты дочернего класса
  - такая ссылка может быть использована только для работы с элементами, объявленными в родительском классе
  - при вызове переопределенного метода (через такую ссылку), будет выполнена реализация метода, соответствующая типу объекта (***динамический полиморфизм***)

```
public class HelloWorld {
      public static void main(String args[]) {
            Point p1 = new Point();
            p1.setCoord(10, 20);

            Point p2 = new Point3D();
            p2.setCoord(44, 38);
            p2.setCoord(44, 38, 77);

            p1.print();
            System.out.println();
            p2.print();
      }
}
```

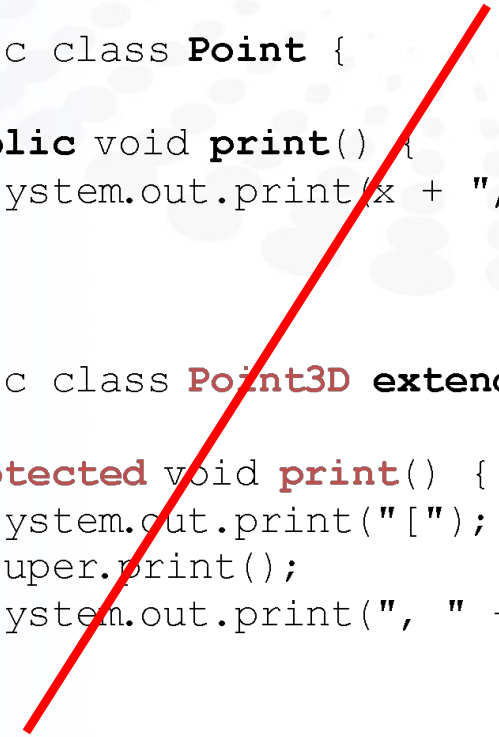
```
10.0, 20.0
[44.0, 38.0, 0.0]
```

Особенности переопределения методов (1)

- Скрытые (*private*) методы не могут быть переопределены
- При переопределении методов нельзя уменьшать их видимость

```
public class Point {
  ...
  protected void print() {
    System.out.print(x + ", " + y);
  }
}

public class Point3D extends Point {
  ...
  public void print() {
    System.out.print("[");
    super.print();
    System.out.print(", " + z + "]");
  }
}
```

```
public class Point {
  ...
  public void print() {
    System.out.print(x + ", " + y);
  }
}

public class Point3D extends Point {
  ...
  protected void print() {
    System.out.print("[");
    super.print();
    System.out.print(", " + z + "]");
  }
}
```



Особенности переопределения методов (2)

- Метод, который вызывает родительский класс, может быть переопределен

```
public class Point {
  ...
  public void print() {
    System.out.print(x + ", " + y);
  }
  public void println() {
    print();
    System.out.println();
  }
}
public class Point3D extends Point {
  ...
  @Override
  public void print() {
    System.out.print("[");
    super.print();
    System.out.print(", " + z + "]");
  }
}
```

```
public class HelloWorld {
  public static void main(
                String args[]) {
    Point p2d = new Point();
    p2d.setCoord(10, 20);
    p2d.println();

    Point3D p3d = new Point3D();
    p3d.setCoord(10, 20, 30);
    p3d.println();
  }
}
```

```
10.0, 20.0
[10.0, 20.0, 30.0]
```

Конструкторы при наследовании (1)

- При создании объекта дочернего класса вначале вызывается конструктор родительского класса, а потом – дочернего
- Конструктор родительского класса может быть вызван явно при помощи ключевого слова ***super***

```
public class Parent {
  public Parent() {
    System.out.println("Привет от контруктора родительского класса");
  }
  public Parent(double y) {
    System.out.println("Вам число "+ y + " от конструктора родительского класса");
  }
}
public class Child extends Parent {
  public Child() {
    System.out.println("Привет от контруктора дочернего класса");
  }
  public Child(int x) {
      super();
      System.out.println("Вам число "+ x + " от конструктора дочернего класса");
  }
  public Child(double y) {
      super(y);
      System.out.println("Вам число "+ y + " от конструктора дочернего класса");
  }
}
```

Конструкторы при наследовании (2)

```
public class HelloWorld {
      public static void main(String args[]) {
            new Child();
            System.out.println();
            new Child(100);
            System.out.println();
            new Child(35.6);
      }
}
```

```
Привет от контруктора родительского класса  !
Привет от контруктора дочернего класса  !

Привет от контруктора родительского класса  !
Вам число  100 от конструктора дочернего класса  !

Вам число  35.6 от конструктора родительского класса  !
Вам число  35.6 от конструктора дочернего класса  !
```

Скрытие полей

- Дочерний класс может иметь поля, совпадающие с названием полей родительского класса
- В этом случае, поля родительского класса будут *скрыты*

```
public class Parent {
  public int x = 55;
  public int y = 10;
  public void print() {
    System.out.println(x + " " + y);
  }
}
public class Child extends Parent {
  public int x;
  public double y;
  void getParentValues()
  {
    x = super.x;
    y = super.y;
  }
  public void print2() {
    System.out.println(x + " " + y);
  }
}
```

```
public class HelloWorld {
  public static void main(
                String args[]) {
    Parent par = new Parent();
    Child chi = new Child();
    par.print();
    chi.print();
    chi.print2();
    chi.getParentValues();
    chi.print2();
  }
}
```

```
55 10
55 10
0 0.0
55 10.0
```

Наследование статических методов

- Статические методы наследуются, но не переопределяются
- Если статический метод дочернего класса совпадает (по имени и параметрам) со статическим методом родительского класса, то метод родительского класса *скрывается*

```
public class Parent {
  public static void test1() {
    System.out.println("Статический метод 1 родительского класса");
  }
  public static void test2() {
    System.out.println("Статический метод 2 родительского класса");
  }
}
public class Child extends Parent {
  public static void test2() {
    System.out.println("Статический метод 2 дочернего класса");
  }
}
public class HelloWorld{
  public static void main(String args[]) {
    Child.test1();
    Child.test2();
  }
}
```

```
Статический метод  1 родительского класса
Статический метод  2 дочернего класса
```

Использование спецификатора final

- Спецификатор ***final***, в объявлении класса запрещает наследование от данного класса
- Спецификатор ***final*** в объявлении метода запрещает переопределение данного метода при наследовании
  - методы, вызываемые в конструкторе, рекомендуется объявлять со спецификатором ***final***

```
public final class MyClass {
   ...
}
```

```
public class MyClass {
     public void Do1() {
          ...
     }
     public final void Do2() {
          ...
     }
     ...
}
```

Учебный центр компьютерного обучения «СПЕЦИАЛИСТ» при МГТУ им. Н.Э.Баумана

7. АБСТРАКТНЫЕ КЛАССЫ И ИНТЕРФЕЙСЫ

компьютерного Центр обучения «СПЕЦИАЛИСТ» при МГТУ им. Н.Э.Баумана®

7.1. АБСТРАКТНЫЕ КЛАССЫ

Абстрактные классы и абстрактные методы

- ***Абстрактный класс*** (***abstract class***)
  - определяет общее поведение для порожденных им классов
  - предполагает наличие дочерних классов
  - объявляется со спецификатором ***abstract***
  - не может иметь объектов
  - может содержать или не содержать ***абстрактные методы***
- Класс должен быть объявлен как абстрактный если:
  - класс содержит абстрактные методы
  - класс наследуется от абстрактного класса, но не реализует абстрактные методы
  - класс имплементирует интерфейс, но не реализует все методы интерфейса

- ***Абстрактный метод*** (***abstract method***)
  - не имеет реализации
  - объявляется со спецификатором ***abstract***
  - переопределяется в дочерних классах

Пример абстрактного класса

```
public abstract class Figure {
      protected Point[] pnts;
      public Figure(int pointsCount) {
            pnts = new Point[pointsCount];
            for (int i=0; i<pnts.length; i++)
                  pnts[i] = new Point();
      }
      public void setCoord(int n, double x, double y) {
            if (n>=0 && n<pnts.length) pnts[n].setCoord(x, y);
      }
      public void move(double dx, double dy) {
            for (Point p : pnts)
                  p.setCoord(p.getX()+dx, p.getY()+dy);
      }
      ...
      public abstract double getArea();
}
```

```
public class Triangular extends Figure {
      public Triangular() {
            super(3);
      }
      @Override
      public double getArea() {
            ...
            return ...;
      }
}
```

Абстрактные классы как типы данных

- Абстрактный класс может использоваться при объявлении ссылок на объекты:
  - ссылка может указывать на объект неабстрактного потомка данного класса

```
Figure f1 = new Figure(3);
Figure f2 = new Triangular();
System.out.println(f2.getArea());
```

Учебный центр компьютерного обучения «СПЕЦИАЛИСТ» при МГТУ им. Н.Э.Баумана

7.2. ИНТЕРФЕЙСЫ

Объявление интерфейсов

- ***Интерфейсы*** (***interfaces***):
  - публичные (***public***)
  - непубличные – доступны внутри пакета
- Интерфейсы могут содержать:
  - абстрактные методы (методы без реализации)
  - статические константы
  - (Java SE 8) статические методы
  - (Java SE 8) *методы по умолчанию* (*default methods*) с реализацией
- Все элементы интерфейса являются публичными (*public*)
  - все поля интерфейса являются *static* и *final*
- Интерфейсы компилируются в файл *.class*

- Рекомендации по именованию интерфейсов:
  - Имя интерфейса состоит из одного или нескольких идущих подряд слов
  - Первая буква каждого слова заглавная, остальные буквы – в нижнем регистре
  - Имя интерфейса обычно заканчивается на '*able*'

```
public interface Computable {
      double compute();
}

public interface Animal {
      static final int MAX_LEGS = 4;
      int NO_LEGS = 0; // static final
      public abstract void say();
      public abstract int legsCount();
      public abstract boolean canSwim();
      public abstract boolean canRun();
      public abstract boolean canFly();
}
```

Имплементация (реализация) интерфейсов (1)

- При объявлении класса можно указать, какие интерфейсы он будет поддерживать
- Класс, реализующий интерфейс:
  - может иметь свои собственные методы (не объявленные в интерфейсе)
  - может иметь свои собственные поля
  - должен реализовать все методы интерфейса, либо быть объявлен как ***абстрактный*** (***abstract***)

```
public class NewClass
    implements Interface1, Interface2, Interface3  {
     ...
}
```

Имплементация (реализация) интерфейсов (2)

```
public class Summator implements Computable {
    private double x= 0;
    private double y= 0;
    public Summator(double nx, double ny) {
        x = nx;
        y = ny;
    }
    @Override
    public double compute() {
        return x+y;
    }
}
```

```
public class Divider implements Computable {
    private double x= 0;
    private double y= 0;
    public Divider(double nx, double ny) {
        x = nx;
        y = ny;
    }
    @Override
    public double compute() {
        return x/y;
    }
}
```

Имплементация (реализация) интерфейсов (3)

```
public class Fish implements Animal {
    @Override
    public void say() {
    }

    @Override
    public int legsCount() {
        return Animal.NO_LEGS;
    }

    @Override
    public boolean canSwim() {
        return true;
    }

    @Override
    public boolean canRun() {
        return false;
    }

    @Override
    public boolean canFly() {
        return false;
    }
}
```

```
public class Cow implements Animal {
    @Override
    public void say() {
        System.out.println("My-my");
    }

    @Override
    public int legsCount() {
        return 4;
    }

    @Override
    public boolean canSwim() {
        return false;
    }

    @Override
    public boolean canRun() {
        return true;
    }

    @Override
    public boolean canFly() {
        return false;
    }
}
```

Интерфейсы как типы данных (1)

- Интерфейс может использоваться при объявлении ссылок на объекты:
  - интерфейсная ссылка может указывать на объект класса, поддерживающего данный интерфейс
  - интерфейсная ссылка может использоваться только для вызова методов, объявленных в интерфейсе

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

          boolean summ = true;
          Computable c = null;
          double val1 = 100;
          double val2 = 25;
          if (summ)
               c = new Summator(val1, val2);
          else
               c = new Divider(val1, val2);
          System.out.println(c.compute());
     }
}
```

Интерфейсы как типы данных (2)

```
public class HelloWorld {
     public static void main(String args[]) {
          Random r = new Random();
          Animal randomAnimal = null;
          if (r.nextBoolean())
               randomAnimal = new Fish();
          else
               randomAnimal = new Cow();

          System.out.print("Животное: ");
          System.out.print(
                    randomAnimal.canSwim()? "плавает, " : "не плавает, ");
          System.out.print(
                    randomAnimal.canFly()? "летает, " : "не летает, ");
          System.out.print(
                    randomAnimal.canRun()? "бегает, " : "не бегает, ");
     }
}
```

Наследование интерфейсов

```
public interface Switchable {
      void switchOn();
      void switchOff();
}
public interface MediaPlayer extends Switchable {
      void play();
      void pause();
      void stop();
}
public class AudioPlayer implements MediaPlayer{
      @Override
      public void switchOn() {...}
      @Override
      public void switchOff(){...}
      @Override
      public void play(){...}
      @Override
      public void pause(){...}
      @Override
      public void stop(){...}
}
```

Абстрактные классы на основе интерфейсов

```
public interface Animal {
      static final int MAX_LEGS = 4;
      int NO_LEGS = 0; // static final
      public abstract void say();
      public abstract int legsCount();
      public abstract boolean canSwim();
      public abstract boolean canRun();
      public abstract boolean canFly();
}
public abstract class Bird implements Animal {
      @Override
      public int legsCount() {return 2;}
      @Override
      public boolean canRun() {return true;}
      @Override
      public boolean canFly() {return true;}
}
public class Duck extends Bird{
      @Override
      public void say() {System.out.println("Кря-кря");}
      @Override
      public boolean canSwim() { return true;}
}
```

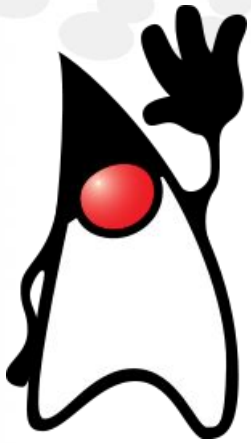
Абстрактные классы vs Интерфейсы

- Абстрактные классы
  - описывают поведение для иерархии классов
  - могут обладать состоянием
  - могут реализовывать алгоритмы
  - объектные ссылки поддерживают динамический полиморфизм
  - могут содержать скрытые и защищенные элементы
  - класс может наследоваться только от одного абстрактного класса
- Интерфейсы
  - описывают поведение для группы классов, реализующих данный интерфейс
  - не могут обладать состоянием
  - (*Java SE 7*) не могут реализовывать алгоритмы; (*Java SE 8*) могут реализовывать алгоритмы по умолчанию
  - интерфейсные ссылки поддерживают динамический полиморфизм
  - содержат только публичные элементы
  - класс может реализовывать несколько интерфейсов

Стандартные интерфейсы Java

- java.lang.Appendable
- java.lang.AutoClosable
- java.lang.CharSequence
- java.lang.Clonable
- java.lang.Comparable
- java.lang.Iterable
- java.lang.Readable
- java.lang.Runnable
- java.io.Serializable
- java.io.DataInput
- java.io.DataOutput
- …

Учебный центр компьютерного обучения «СПЕЦИАЛИСТ» при МГТУ им. Н.Э.Баумана



8. КЛАССЫ OBJECT И CLASS

Класс *java.lang.Object*

- Все классы являются потомками класса ***java.lang.Object***
- Если при объявлении класса явно не указывается родительский класс (***extends***), класс наследуется от ***Object*** напрямую

- Методы класса Object
  - `protected Object clone()`
  - `public boolean equals(Object obj)`
  - `protected void finalize()`
  - `public final Class getClass()`
  - `public int hashCode()`
  - `public String toString()`
  - `public final void notify()`
  - `public final void notifyAll()`
  - `public final void wait()`
  - `public final void wait(long timeout)`

Класс *Object*. Метод *clone*

- Метод ***clone*** создает копию объекта
- Метод ***clone*** работает для объектов классов, реализующих интерфейс ***Cloneable***
- Метод ***clone*** следует переопределять:
  - чтобы сделать его публичным
  - в случае, если копируемый объект содержит ссылки на внешние объекты

```
public class IntValue implements Cloneable {
      public int value = 0;
      @Override
      public Object clone() throws CloneNotSupportedException{
            return super.clone();
      }
}

public class HelloWorld {
      public static void main(String args[])
                  throws CloneNotSupportedException{
            IntValue a = new IntValue();
            a.value = 20;
            IntValue b = (IntValue)a.clone();
            System.out.println(b.value);
      }
}
```

Класс *Object*. Метод *equals*

- Метод ***equals*** сравнивает два объекта на равенство
- При переопределении метода ***equals*** следует также переопределять метод ***hashCode***

```
public class IntValue {
      public int value = 0;
      @Override
      public boolean equals(Object obj) {
            if (obj instanceof IntValue)
                  return ((IntValue)obj).value == value;
            else
                  return false;
      }
      ...
}
```

```
public class HelloWorld {
      public static void main(String args[]) {
            IntValue a = new IntValue();
            a.value = 20;
            IntValue b = new IntValue();
            b.value = 20;
            System.out.println(a.equals(b));
            b.value = 21;
            System.out.println(a.equals(b));
            System.out.println(a.equals("Hello"));
      }
}
```

Класс *Object*. Метод *toString*

- Метод ***toString*** преобразует объект в строку

```
public class IntValue {
      public int value = 0;
      @Override
      public String toString() {
            return "Число "+value;
      }
}

public class HelloWorld {
      public static void main(String args[])  {
            IntValue a = new IntValue();
            a.value = 20;
            System.out.println(a);
      }
}
```

Класс *Object*. Метод *finalize*

- Метод ***finalize*** вызывается однократно сборщиком мусора перед уничтожением объекта
- Метод может использоваться для освобождения ресурсов объекта
- Отсутствует гарантия вызова данного метода

```
public class IntValue {
      public int value = 0;
      @Override
      protected void finalize() {
            System.out.println("Удаляюсь");
      }
}

public class HelloWorld {
      public static void main(String args[])  {
            IntValue a = new IntValue();
            a = null;
            System.gc();
      }
}
```

Класс *Object*. Метод *hashCode*

- Метод ***hashCode*** возвращает хэш-код объекта
- Стандартная реализация метода использует в качестве хэш-кода адрес объекта в памяти
- При переопределении метода необходимо учитывать следующее соглашение:
  - для одного и того же объекта метод ***hashCode*** должен всегда возвращать одно и то же значение
  - для равных объектов (при проверке методом ***equals***) метод ***hashCode*** должен всегда возвращать одно и то же значение
  - допускается возвращение одинакового хэш-кода для различных объектов, в то же время это снижает производительность при работе с хэш-таблицами

Класс *Object*. Метод *getClass*. RTTI

- Метод ***getClass*** возвращает объект класса ***java.lang.Class***, содержащий информацию о классе объекта
- Метод не может быть переопределен

- ***Run-time type identification*** (***RTTI***) – механизм, позволяющий определить тип данных объекта во время выполнения программы
- Инструменты RTTI:
  - ***Object.getClass***
  - ***instanceof***

```
Это объект класса  Point
Это объект класса  Point3D
hello.Point3D
```

```
Object obj = new Point3D();
if (obj instanceof Point)
     System.out.println("Это объект класса Point");
if (obj instanceof Point3D)
     System.out.println("Это объект класса Point3D");
System.out.println(obj.getClass().getName());
```

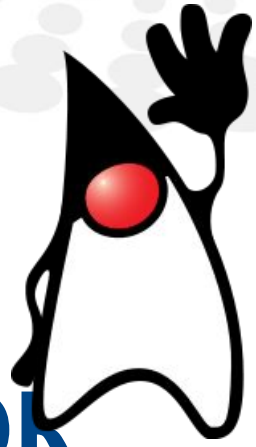
Класс Class (1)

- Методы:
  - `T cast(Object obj)`
  - `Field[] getDeclaredFields()`
  - `Method[] getDeclaredMethods()`
  - `Field[] getFields()`
  - `Method[] getMethods()`
  - `Class<?>[] getInterfaces()`
  - `Class<?> getSuperclass()`
  - `String getName();`
  - `String getSimpleName();`
  - `boolean isInterface();`
  - `T newInstance()`
  - `static Class<?> forName(String className)`
  - `...`

Класс Class (2)

```
Object obj = new Point3D();
Class<?> c = obj.getClass();

Method[] arr = c.getDeclaredMethods();
for (Method i: arr) System.out.print(i.getName() + " ");
System.out.println();

arr = c.getMethods();
for (Method i: arr)   System.out.print(i.getName() + " ");
System.out.println();

System.out.println(c.getName());
System.out.println(c.getSimpleName());
System.out.println(c.getSuperclass().getSimpleName());

Point3D pnt = (Point3D) Class.forName("hello.Point3D").newInstance();
```

```
getZ setCoord print
getZ setCoord print setCoord println getX getY getClass hashCode equals toStri ...
hello.Point3D
Point3D
Point
```



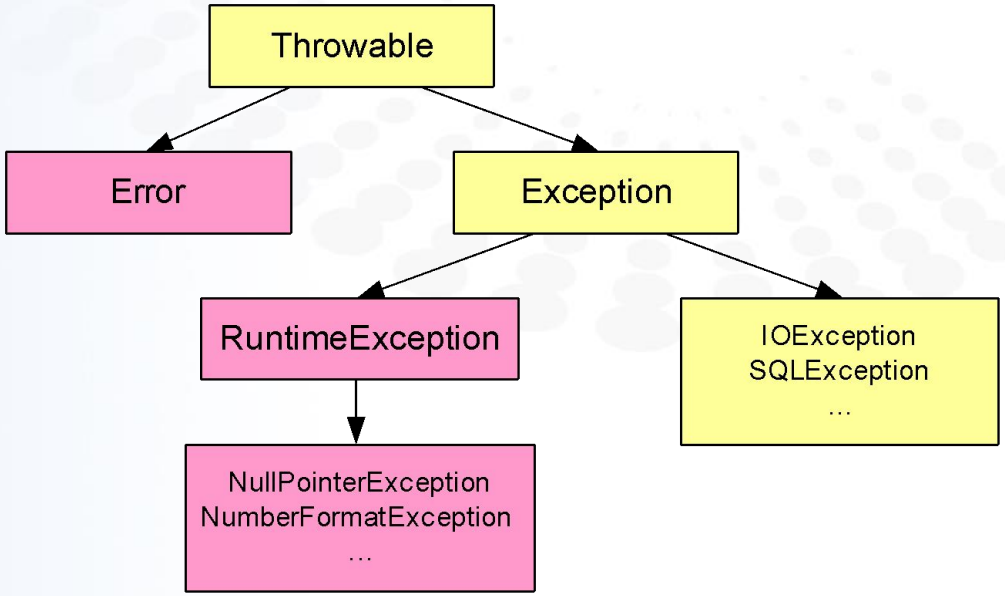
9. ОБРАБОТКА ОШИБОК

Понятие исключения

- ***Исключение*** (***exception***) – событие, возникающее в процессе выполнения программы, прерывающее ход выполнения инструкций программы
- При возникновении ошибки в работе метода, метод создает специальный объект (***exception object***) и передает его среде выполнения (*выбрасывает исключение*, ***throws an exception***)
  - *Exception object* содержит информацию о возникшей ошибке
- После выбрасывания исключения:
  - Работа метода прерывается
  - JVM ищет *обработчик* исключения (***exception handler***) в стеке вызовов метода (снизу вверх, начиная от метода выбросившего исключение, заканчивая методом *main*)
  - Если обработчик найден, управление передается ему. В таком случае принято говорить, что обработчик *«поймал» исключение* (***catch the exception***)
  - Если обработчик не найден, выполнение программы прерывается

Типы исключений. Иерархия исключений

- ***Checked exceptions***
  - возникают при вызове определенных методов
  - метод обязан отреагировать на исключение:
    - либо обработать его
    - либо выбросить его вызывающему методу



- **Unchecked exceptions**
  - могут возникнуть в любой момент
  - не подлежать обязательной обработке и декларации

Класс java.lang.Throwable

- Класс ***java.lang.Throwable*** является предком для всех «выбрасываемых» объектов (объектов *exception object*)
- Основные методы:
  - `String getMessage()` – возвращает сообщение об ошибке
  - `void printStackTrace()` – выводит сведения об ошибке и стек вызовов методов в стандартный поток вывода ошибок
  - `void printStackTrace(PrintStream s)` – выводит сведения об ошибке и стек вызовов методов в заданный поток
  - `String toString()` – возвращает краткую информацию об ошибке
- Потомки
  - ***java.lang.Error***
  - ***java.lang.Exception***

Класс *java.lang.Error*

- Класс ***java.lang.Error*** – базовый класс для описания критических ошибок, возникающих в процессе работы программы
  - ***unchecked exception***
  - категорически не рекомендованы к обработке
  - должны приводить к завершению работы JVM
- Потомки
  - java.lang.OutOfMemoryError
  - java.lang.StackOverflowError
  - java.lang.UnknownError
  - java.lang.InternalError
  - java.io.IOError
  - java.lang.NoClassDefFoundError
  - …

Класс *java.lang.Exception*

- Класс ***java.lang.Exception*** является родительским классом для всех прикладных исключений
- Исключения ***Exception*** (и его потомки, кроме *RuntimeException*)
  - ***checked exception***
  - не должны приводить к остановке программы
  - связаны с ожидаемыми проблемами при выполнении конкретных методов (ошибка подключения к базе данных, ошибка открытия файла, и т.п)

- Потомки:
  - java.io.IOException
    - java.io.FileNotFoundException
    - java.net.UnknownHostException
    - …
  - java.sql.SQLException
  - java.lang.ClassNotFoundException
  - java.lang.CloneNotSupportedException
  - java.util.concurrent.TimeoutException
  - …
  - java.lang.RuntimeException

Класс *java.lang.RuntimeException*

- ***java.lang.RuntimeException*** – исключения, связанные с ошибками выполнения программы
  - ***unchecked exception***
  - часто, вызваны ошибками в коде программы
  - не рекомендованы к обработке в тех случаях, когда они могут быть исправлены путем модификации исходного кода

- Потомки:
  - java.lang.ArithmeticException
  - java.lang.IndexOutOfBoundsException
  - java.lang.NegativeArraySizeException
  - java.lang.NullPointerException
  - java.lang.IllegalArgumentException
    - java.lang.NumberFormatException
  - ...

Обработка исключений

- Обработка исключений осуществляется с использованием блоков ***try***, ***catch*** и ***finally***
- Блок ***try*** содержит код, который может потенциально выбросить исключение
- Блок ***catch*** содержит непосредственно обработчик для заданного типа исключения
  - для различных исключений могут быть объявлены различные блоки ***catch***
  - в случае, если выполнение блока ***try*** прерывается исключением, управление передается соответствующему блоку ***catch***
- Блоку ***finally*** передается управление после завершения блока ***try*** и ***catch*** в независимости от возникших (или не возникших) исключений

Конструкция *try-catch*

```
public static void main(String args[]) {
  try {
    int sum = 0;
    for (int i=0; i<args.length; i++) {
      int value = Integer.valueOf(args[i]);
      sum  +=value;
    }
    System.out.println(
              "Сумма элементов: " + sum);
  }
  catch (NumberFormatException e) {
    System.out.println("Возникла ошибка");
    System.out.println(e.getMessage());
  }
}
```

```
try {
    ...
}
catch (E1 e) {
    ...
}
catch (E2 e) {
    ...
}
```

```
try {
    ...
}
catch (E1|E2 e) {
    ...
}
```

Конструкция *try-finally*

- Блок ***finally*** выполняется независимо от возникновения ошибок
- Блок ***finally*** выполняется даже если исключение возникает в обработчике ***catch*** или обработчик ***catch*** отсутствует

```
public static void main(String args[]) {
  int a = 10;
  int b = 0;
  int c = 15;

  try {
    c = a/b + c;
  }
  catch (ArithmeticException e) {
    c = 0;
  }
  finally {
    System.out.println("c = " + c);
  }
}
```

```
public static void main(String args[]) {
  int a = 10;
  int b = 0;
  int c = 15;
  try {c = a/b + c;  }
  finally { System.out.println("c = " + c);  }
  System.out.println("Сюда в случае ошибки уже не попадем");
}
```

Конструкция *try-with-resources*

- Конструкция ***try-with-resources*** позволяет гарантированно закрывать используемые ресурсы независимо от возникновения ошибки
- Объекты-ресурсы должны поддерживать интерфейс ***java.lang.AutoClosable***

```
try (FileInputStream input = new FileInputStream("c:\\file.txt");
     BufferedInputStream b = new BufferedInputStream(input); )
{
    int data = b.read();
    while(data != -1){
        System.out.print((char) data);
        data = b.read();
    }
}
```

Определение исключений для метода

- При объявлении метода необходимо указывать, какие исключения (типа *checked exception*) он может выбрасывать:

```
public void myMethod() throws Exception1, Exception2
```

- Метод, вызывающий другой метод с исключением, должен:
  - либо иметь обработчик для исключения
  - либо быть объявлен как выбрасывающий данное исключение

```
public void myMethod() throws Exception1, Exception2 {
   ...
}
...
public void anotherMethod() throws Exception1 {
  try {
    someObj.myMethod()
  }
  catch(Exception2 e) { ... }
}
```

Генерация исключений

- Исключения генерируются (выбрасываются) при помощи оператора ***throw***
  - после ключевого слова ***throw*** указывается ссылка на выбрасываемый объект (*exception object*)

```
public class Figure {
      protected Point[] pnts;
      public Figure(int pointsCount) {
            pnts = new Point[pointsCount];
            for (int i=0; i<pnts.length; i++)
                  pnts[i] = new Point();
      }
      ...
      public void setCoord(int n, double x, double y)
                                          throwsException {
            if (n>=0 && n<pnts.length)
                  pnts[n].setCoord(x, y);
            else
                  throw new Exception();
      }
}
```

Использование собственных исключений (1)

```
public class PointIndexException
    extends Exception {
     private final int index;
     public PointIndexException(int index) {
          super("Не корректный номер точки");
          this.index = index;
     }
     public int getIndex() {
          return index;
     }
}

public class Figure {
     protected Point[] pnts;
     ...
     public void setCoord(int n, double x, double y)
               throws PointIndexException {
          if (n>=0 && n<pnts.length)
               pnts[n].setCoord(x, y);
          else
               throw new PointIndexException(n);
     }
}
```

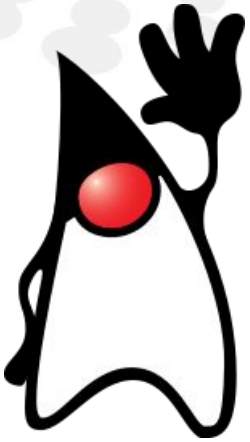
Использование собственных исключений (2)

```
Figure f = new Figure(3);
try {
      System.out.println("Работаю с первой точкой");
      f.setCoord(0, 0, 0);
      System.out.println("Работаю со второй точкой");
      f.setCoord(1, 10, 0);
      System.out.println("Работаю с третьей точкой");
      f.setCoord(2, 10, 10);
      System.out.println("Работаю с четверой точкой");
      f.setCoord(3, 0, 10);
      System.out.println("Все!");
}
catch(PointIndexException e) {
      System.out.print(e.getMessage() + ": " +
                              e.getIndex());
}
```

```
Работаю с первой точкой
Работаю со второй точкой
Работаю с третьей точкой
Работаю с четверой точкой
Не корректный номер точки : 3
```

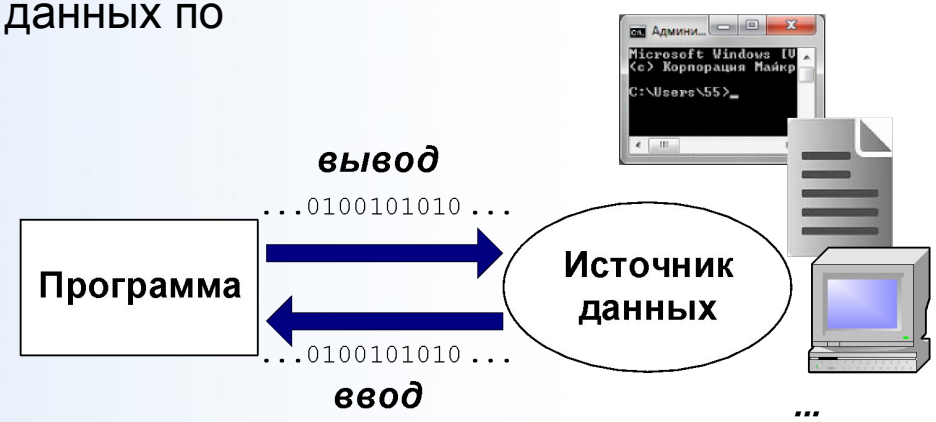
компьютерного Центр обучения «СПЕЦИАЛИСТ» при МГТУ им. Н.Э.Баумана

10. ПОТОКИ ДАННЫХ



Потоки ввода-вывода

- `java.io.*`
- Операции ввода-вывода в Java осуществляются через ***потоки (I/O streams)***
  - ***поток ввода (input stream)***
  - ***поток вывода (output stream)***
- Потоки связаны с некоторым источником данных:
  - файл на диске
  - сокет (при передаче данных по сети)
  - устройство
  - буфер в памяти
  - …
- Различные потоки могут поддерживать передачу различных данных:
  - байты
  - символы
  - значения примитивных типов
  - объекты
  - …



Учебный центр компьютерного обучения «СПЕЦИАЛИСТ» при МГТУ им. Н.Э.Баумана

10.1. БАЙТОВЫЕ ПОТОКИ

Байтовые потоки

- *Байтовые потоки* (*byte streams*) используются для передачи данных в виде последовательности байт

- ***java.io.InputStream*** – абстрактный класс, управляющий байтовым потоком ввода
- Потомки:
  - *ByteArrayInputStream*
    - *StringBufferInputStream*
  - *FileInputStream*
  - *FilterInputStream*
    - *BufferedInputStream*
    - *DataInputStream*
  - *ObjectInputStream*
  - …

- ***java.io.OutputStream*** – абстрактный класс, управляющий байтовым потоком ввода
- Потомки:
  - *ByteArrayOutputStream*
  - *FileOutputStream*
  - *FilterOutputStream*
    - *BufferedOutputStream*
    - *DataOutputStream*
    - *PrintStream*
  - *ObjectOutputStream*
  - …

Класс OutputStream. Методы

- `abstract void write(int b) throws IOException` - записывает указанный байт в поток
- `void write(byte[] b) throws IOException` - записывает байты из заданного массива в поток
- `void write(byte[] b, int off, int len) throws IOException` - записывает `len` байт из заданного массива, начиная с позиции `off`, в поток
- `void flush() throws IOException` - форсирует запись байт из буфера потока в источник данных
- `void close() throws IOException` – закрывает поток и освобождает ресурсы, связанные с потоком

Класс InputStream. Методы (1)

- `abstract int` ***`read`***`() throws IOException` –
  считывает из потока очередной байт. Возвращает байт (0-255) или -1 при достижении конца потока
- `int` ***`read`***`(byte[] b) throws IOException` –
  считывает байты из потока и записывает их в заданный массив. Возвращает количество прочитанных байт или -1 при достижении конца потока
- `int` ***`read`***`(byte[] b, int off, int len) throws IOException` –
  считывает `len` байт из потока и записывает их, начиная с позиции `off`, в заданный массив `b`. Возвращает количество прочитанных байт или -1 при достижении конца потока
- `long` ***`skip`***`(long n) throws IOException` –
  считывает заданное количество байт из потока и игнорирует их. Возвращает количество пропущенных байт

Класс InputStream. Методы (2)

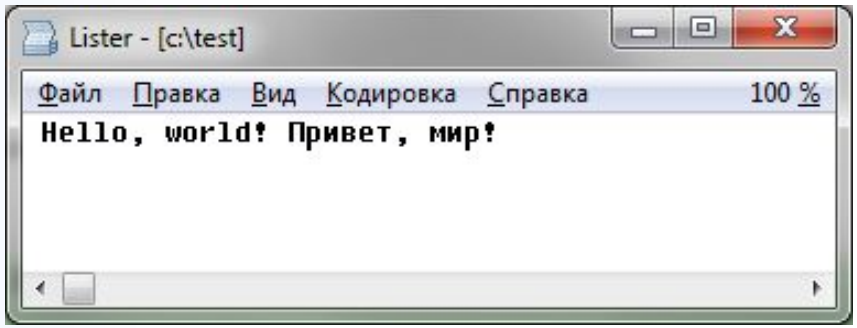
- `int` ***`available`***`() throws IOException` –
  возвращает количество байт, доступных для чтения
- `void` ***`mark`***`(int r)` –
  ставит метку в текущей позиции входного потока, которую можно будет использовать, пока из потока не будет прочитано `r` байт
- `boolean` ***`markSupported`***`()` – проверяет, поддерживает ли данный поток методы `mark` и `reset`
- `void` ***`reset`***`() throws IOException` – возвращает указатель потока на установленную метку
- `void` ***`close`***`() throws IOException` –
  закрывает поток и освобождает ресурсы, связанные с потоком

Потоки для работы с файлами. FileOutputStream (1)

- ***java.io.FileOutputStream***
- Переопределяет методы: `close, write, finalize`
- Конструкторы:
  - `public` ***`FileOutputStream`***`(String name) throws FileNotFoundException` –
    создает поток для записи данных в заданный файл
  - `public` ***`FileOutputStream`***`(String name, boolean append) throws FileNotFoundException`
    создает поток для записи данных в заданный файл. Если `append`, данные будут записываться в конец файла
  - `public` ***`FileOutputStream`***`(File file) throws FileNotFoundException`
  - `public` ***`FileOutputStrea`***`m(File file, boolean append) throws FileNotFoundException`
  - `public` ***`FileOutputStream`***`(FileDescriptor fdObj)`

Потоки для работы с файлами. FileOutputStream (2)

```
public class HelloWorld {
  public static void main(String args[]) throws IOException {
    String fileName = "c:\\test";
    FileOutputStream fout = null;
    String str = "Hello, world! Привет, мир!";
    byte [] byteArray = str.getBytes();

    try {
      fout = new FileOutputStream(fileName);
      fout.write(byteArray);
    }
    catch (IOException e) {
      e.printStackTrace();
    }
    finally {
      if (fout!=null)
        fout.close();
    }
  }
}
```



Потоки для работы с файлами. FileOutputStream (3)

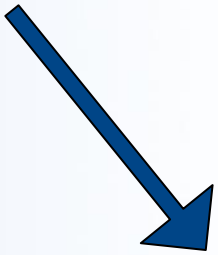
```
public class HelloWorld {
  public static void main(String args[]) {
    String fileName = "c:\\test";
    String str = "Hello, world! Привет, мир!";
    byte [] byteArray = str.getBytes();

    try (FileOutputStream fout = new FileOutputStream(fileName)) {
      fout.write(byteArray);
    }
    catch (IOException e) {
      e.printStackTrace();
    }
  }
}
```

Потоки для работы с файлами. FileInputStream (1)

- ***java.io.FileInputStream***
- Переопределяет методы: `available, close, read, skip, finalize`
- Конструкторы:
  - ***`FileInputStream`***`(String name) throws FileNotFoundException` – создает поток для чтения данных из файла с заданным названием
  - ***`FileInputStream`***`(File file) throws FileNotFoundException`
  - ***`FileInputStream`***`(FileDescriptor fdObj)`

Потоки для работы с файлами. FileInputStream (2)

```
public class HelloWorld{
  public static void main(String args[]) {
    String fileName = "c:\\test";

    try (FileInputStream fin = new FileInputStream(fileName)) {
      int b;
      do {
        b = fin.read();
          if (b!=-1) {
            System.out.print(b + " ");
          }
       } while (b!=-1);
    }
    catch (IOException e) {
      e.printStackTrace();
    }
  }
}
```



```
72 101 108 108 111 44 32 119 111 114 108 100   ...
```

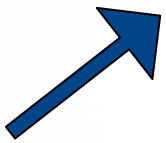
Потоки для работы с массивами байт. ByteArrayOutputStream (1)

- ***java.io.ByteArrayOutputStream***
- Переопределяет методы: `close, write`
- Методы:
  - `public void reset()`
  - `public byte[] toByteArray()`
  - `public int size()`
  - `public String toString()`
  - `public String toString(String charsetName) throws UnsupportedEncodingException`
- Конструкторы:
  - `public ByteArrayOutputStream()`
  - `public ByteArrayOutputStream(int size)`

Потоки для работы с массивами байт. ByteArrayOutputStream (2)

```
public class HelloWorld{
  public static void main(String args[]) {
    String fileName = "c:\\test";
    ByteArrayOutputStream bout = new ByteArrayOutputStream(1024);
    try (FileInputStream fin = new FileInputStream(fileName)) {
      int b;
      do {
        b = fin.read();
        if (b!=-1) bout.write(b);
      } while (b!=-1);
    }
    catch (IOException e) {
      e.printStackTrace();
    }

    byte[] byteArray = bout.toByteArray();
    for (byte ch:byteArray)
      System.out.print((char)ch);
    System.out.println();
    System.out.println(bout);     \\ bout.toString()
  }
}
```

```
Hello, world! ??????, ??? !
Hello, world! Привет, мир!
```



Потоки для работы с массивами байт. ByteArrayInputStream

- ***java.io.ByteArrayInputStream***
- Переопределяет методы: `available, close, mark, markSupported, read, reset, skip`
- Конструкторы:
  - `public` **`ByteArrayInputStream`**`(byte[] buf)`
  - `public` **`ByteArrayInputStream`**`(byte[] buf, int offset, int length)`

```
byte[] byteArr = {12, 10, 25, 3, 72, 14, 8};
ByteArrayInputStream bin = new ByteArrayInputStream(byteArr, 2, 3);
int k;
do {
      k = bin.read();
      System.out.println(k);
}
while (k!=-1);
```

```
25
3
72
-1
```

Учебный центр компьютерного обучения «СПЕЦИАЛИСТ» при МГТУ им. Н.Э.Баумана

10.2. СИМВОЛЬНЫЕ ПОТОКИ

Символьные потоки

- *Символьные потоки* (*character streams*)
  - предназначены для работы с текстовыми данными
  - обеспечивают конвертацию символов между юникодом и локальными кодировками

- ***java.io.Reader*** – абстрактный класс, управляющий чтением символьных потоков
- Потомки:
  - *BufferedReader*
  - *CharArrayReader*
  - *FilterReader*
  - *InputStreamReader*
    - *FileReader*
  - *StringReader*
  - …

- ***java.io.Writer*** – абстрактный класс, управляющий записью в символьные потоки
- Потомки:
  - *BufferedWriter*
  - *CharArrayWriter*
  - *FilterWriter*
  - *OutputStreamWriter*
    - *FileWriter*
  - *PrintWriter*
  - *StringWriter*
  - …

Класс Writer. Методы

- `abstract void write(char[] cbuf, int off, int len) throws IOException` – записывает в поток `len` символов из заданного массива `cbuf`, начиная с позиции `off`
- `void write(int c) throws IOException` – записывает в поток символ `c`
- `void write(char[] cbuf) throws IOException`
- `void write(String str) throws IOException`
- `void write(String str, int off, int len) throws IOException`
- `Writer append(CharSequence csq) throws IOException`
- `Writer append(CharSequence csq, int start, int end) throws IOException`
- `Writer append(char c) throws IOException`
- `abstract void flush() throws IOException` – форсирует запись символов из буфера потока в источник данных
- `abstract void close() throws IOException` – закрывает поток и освобождает связанные с ним ресурсы

Класс Reader. Методы

- `int read(CharBuffer target) throws IOException` – считывает символы в заданный буфер. Возвращает количество прочитанных символов или -1 в случае достижения конца потока
- `int read() throws IOException` – считывает один символ. Возвращает код символа или -1 в случае достижения конца потока
- `int read(char[] cbuf) throws IOException`
- `abstract int read(char[] cbuf, int off, int len) throws IOException`
- `long skip(long n) throws IOException` – пропускает последующие n символов в потоке. Возвращает количество пропущенных символов
- `boolean ready() throws IOException` – возвращает готовность потока к чтению
- `boolean markSupported()`
- `void mark(int readAheadLimit) throws IOException`
- `void reset() throws IOException`
- `abstract void close() throws IOException` – закрывает поток и освобождает связанные с ним ресурсы

Символьные потоки как оболочки над байтовыми (1)

- Реализация символьного ввода-вывода осуществляется через байтовые потоки
- ***java.io.InputStreamReader*** – управляет чтением символов из заданного байтового потока
  - Переопределяет методы: `close, read, ready`
  - Методы: `String` ***`getEncoding`***`()` – возвращает название кодировки
  - Конструкторы:
    - ***`InputStreamReader`***`(InputStream in, String charsetName) throws UnsupportedEncodingException`
    - ***`InputStreamReader`***`(InputStream in)`
    - `...`
- ***java.io.OutputStreamWriter*** – управляет записью символов в заданный байтовый поток
  - Переопределяет методы: `close, flush, write`
  - Методы: `String` ***`getEncoding`***`()` – возвращает название кодировки
  - Конструкторы:
    - ***`OutputStreamWriter`***`(OutputStream out, String charsetName) throws UnsupportedEncodingException`
    - ***`OutputStreamWriter`***`(OutputStream out)`
    - `...`

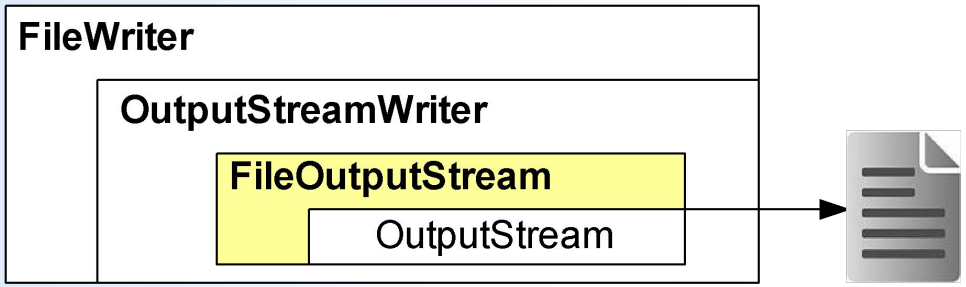
Символьные потоки как оболочки над байтовыми (2)

```
public class HelloWorld {
  public static void main(String args[]) {
      String fileName = "c:\\test";

      try (FileInputStream fin = new FileInputStream(fileName);
           InputStreamReader isr = new InputStreamReader(fin) ) {
        int b;
        do {
          b = isr.read();
          if (b!=-1) System.out.print((char)b);
        } while (b!=-1);
      }
      catch (IOException e) {
        e.printStackTrace();
      }
  }
}
```

```
Hello, world! Привет, мир!
```

Потоки для работы с файлами (1)

- ***FileWriter*** – класс, управляющий записью символов в файл
  - Конструкторы:
    - ***FileWriter***(String fileName) throws IOException
    - ***FileWriter***(String fileName, boolean append) throws IOException
    - ***FileWriter***(File file) throws IOException
    - …
- ***FileReader*** – класс, управляющий чтением символов из файла
  - Конструкторы:
    - ***FileReader***(String fileName) throws FileNotFoundException
    - ***FileReader***(File file) throws FileNotFoundException
    - …



Потоки для работы с файлами (2)

```
public class HelloWorld {
      public static void main(String args[]) throws IOException{
            String fileName = "c:\\test";
            String str = "Hello, world! Привет, мир!";

            try (FileWriter fwr = new FileWriter(fileName)) {
             fwr.write(str);
            }

            try (FileReader fr = new FileReader(fileName)) {
             int b;
             do {
                 b = fr.read();
                 System.out.print( b!=-1? (char)b:"");
             } while (b!=-1);
            }
      }
}
```

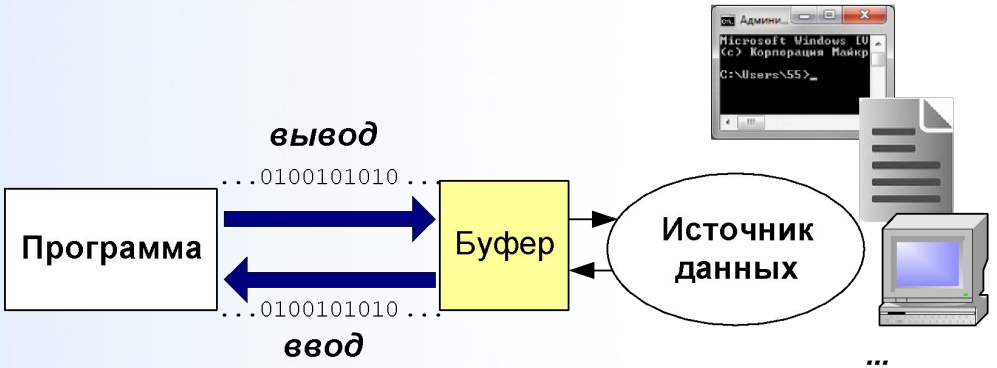
```
Hello, world! Привет, мир!
```

10.3. БУФЕРИЗОВАННЫЕ ПОТОКИ

Буферизованные потоки

- ***Буферизованные потоки*** (***buffered streams***)
  - используют буфер для промежуточного хранения данных
  - повышают производительность ввода/вывода за счет уменьшения количества обращений к источнику данных
  - выступают в качестве оболочек для небуферизованных потоков

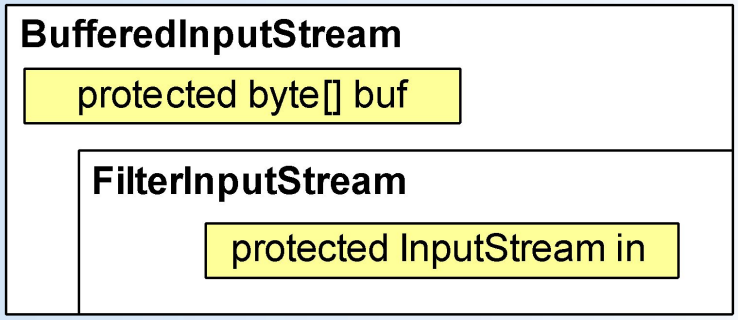
- Буферизованные потоки:
  - ***BufferedInputStream***
  - ***BufferedOutputStream***
  - ***BufferedReader***
  - ***BufferedWriter***

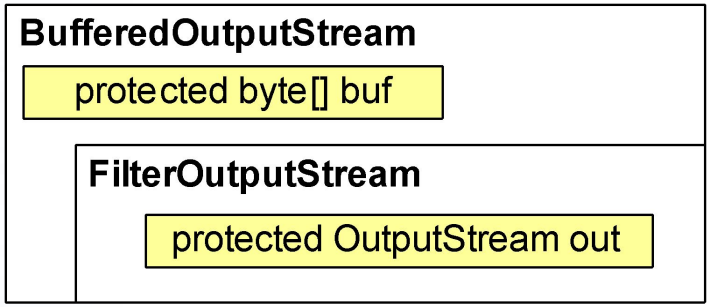


BufferedInputStream и BufferedOutputStream

- ***BufferedInputStream***
- Переопределяет методы:
  `available, close, mark, markSupported, read,reset, skip`
- Конструкторы:
  - `public` ***`BufferedInputStream`***
    `(InputStream in)`
  - `public` ***`BufferedInputStream`***
    `(InputStream in, int size)`

- ***BufferedOutputStream***
- Переопределяет методы:
  `flush, write, close`
- Конструкторы:
  - `public` ***`BufferedOutputStream`***
    `(OutputStream out)`
  - `public` ***`BufferedOutputStream`***
    `(OutputStream out, int size)`

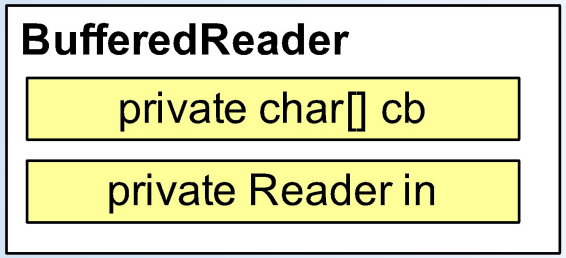


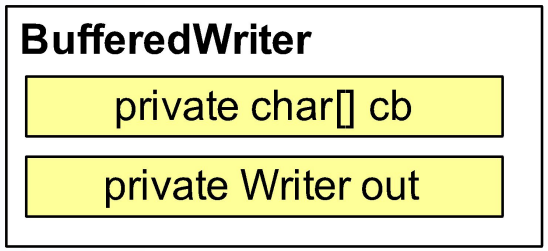


BufferedReader и BufferedWriter

- ***BufferedReader***
- Переопределяет методы: `close, mark, ready, markSupported, read, reset, skip`
- Основные методы:
  - `public String` ***`readLine`***`() throws IOException`
- Конструкторы:
  - `public` ***`BufferedReader`***`(Reader in)`
  - `public` ***`BufferedReader`***`(Reader in, int sz`

- ***BufferedWriter***
- Переопределяет методы: `close, flush, write`
- Основные методы:
  - `public void` ***`newLine`***`() throws IOException`
- Конструкторы:
  - `public` ***`BufferedWriter`***`( Writer out)`
  - `public` ***`BufferedWriter`***`( Writer out, int sz)`

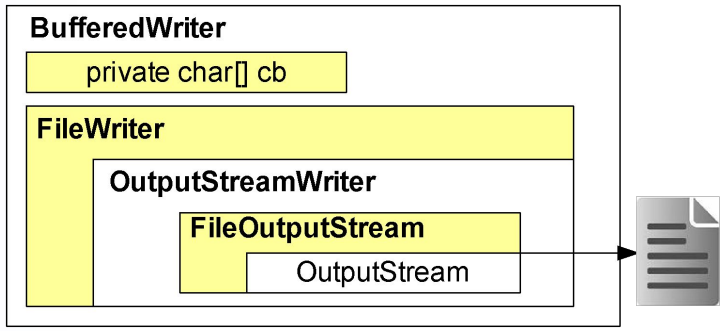




Пример использования буферизованных потоков

```
String fileName = "c:\\test";
String str = "Hello, world! Привет, мир!\n";

try (BufferedWriter bw = new BufferedWriter(
                                 new FileWriter(fileName))) {
     for (int i=0; i<5; i++)
          bw.write(str);
}

try (BufferedReader br = new BufferedReader(
                                 new FileReader(fileName))) {
     String s = null;
     while ( (s=br.readLine()) != null)
         System.out.println(s);
}
```

```
Hello, world! Привет, мир!
Hello, world! Привет, мир!
Hello, world! Привет, мир!
Hello, world! Привет, мир!
Hello, world! Привет, мир!
```



Учебный центр компьютерного обучения «СПЕЦИАЛИСТ» при МГТУ им. Н.Э.Баумана

10.4. ФОРМАТИРОВАННЫЙ ВВОД-ВЫВОД

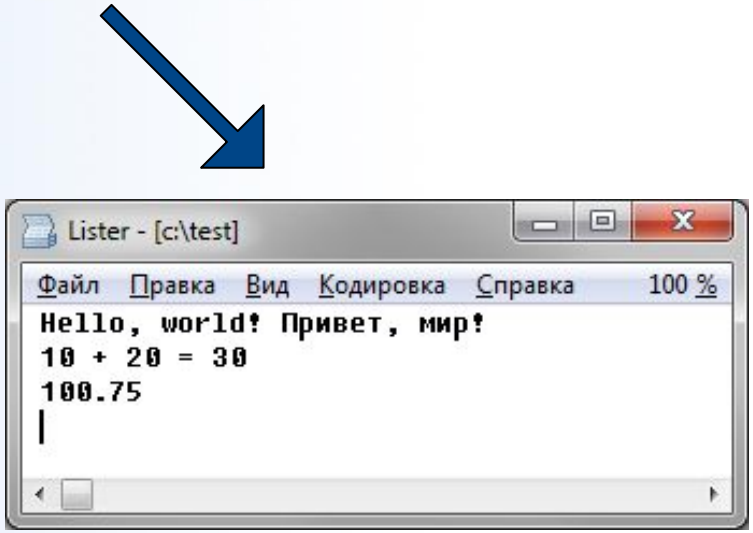
Форматированный вывод (1)

- ***java.io.PrintWriter*** - обеспечивает вывод в символьный поток
- Основные методы:
  - `void print(char x)`
  - `void print(int x)`
  - …
  - `void println()`
  - `void println(char x)`
  - `void println(int x)`
  - …
  - `PrintWriter printf(String format, Object... args)`
  - …
- Конструкторы:
  - `public PrintWriter(Writer out)`
  - `public PrintWriter(Writer out, boolean autoFlush)`
  - `public PrintWriter(OutputStream out)`
  - `public PrintWriter(String fileName) throws FileNotFoundException`
  - …

- ***java.io.PrintStream*** – обеспечивает вывод в байтовый поток
- Основные методы:
  - `void print(char x)`
  - `void print(int x)`
  - …
  - `void println()`
  - `void println(char x)`
  - `void println(int x)`
  - …
  - `PrintStream printf(String format, Object... args)`
  - …
- Конструкторы:
  - `PrintStream(OutputStream out)`
  - `PrintStream(OutputStream out, boolean autoFlush)`
  - `PrintStream(String fileName) throws FileNotFoundException`
  - …

Форматированный вывод (2)

```
public class HelloWorld {
  public static void main(String args[]) throws IOException{
      String fileName= "c:\\test";
      String str = "Hello, world! Привет, мир!";
      try (PrintWriter pw = new PrintWriter(fileName)) {
        pw.println(str);
        pw.printf("%d + %d = %d", 10, 20, (10+20));
        pw.println();
        pw.println(100.75);
      }
  }
}
```



Форматированный ввод (1)

- ***java.util.Scanner*** – управляет вводом текстовых данных на основе регулярных выражений
- Основные методы:
  - void ***close***()
  - IOException ***ioException***()
  - Scanner ***useDelimiter***(String pattern)
  - boolean ***hasNext***()
  - boolean ***hasNext***(Pattern pattern)
  - String ***next***()
  - String ***next***(String pattern)
  - boolean ***hasNextIn***t()
  - int ***nextInt***()
  - long ***nextLong***()
  - …
- Конструкторы:
  - ***Scanner***(InputStream source)
  - ***Scanner***(File source) throws FileNotFoundException
  - ***Scanner***(String source)

Форматированный ввод (2)

```
String s = "10 20 45 77 19 abc 30";
Scanner sc = new Scanner(s);
while (sc.hasNext())
      System.out.println(sc.next());
sc.close();

sc = new Scanner(s);
while (sc.hasNextInt()) {
      int k = sc.nextInt();
      System.out.print(k + " ");
}
sc.close();
System.out.println();

String s2 = "1 10 ab 30 abc 25 40 25 70 25";
sc = new Scanner(s2);
sc.useDelimiter("\\s*ab\\s*");
while (sc.hasNext())
      System.out.println(sc.next());
sc.close();
```

```
10
20
45
77
19
abc
30
```

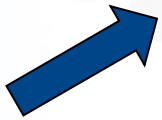
```
10 20 45 77 19
```

```
1 10
30
c 25 40 25 70 25
```

Консольный ввод-вывод. Класс System

- Поток ввода (класс ***java.lang.System***):
  - `InputStream in`
- Поток вывода (класс ***java.lang.System***):
  - `PrintStream out`
  - `PrintStream err`

```
System.out.println("Введите число: ");
Scanner sc = new Scanner(System.in);
int value = sc.nextInt();
System.out.println("Введено число: " + value);
```



```
Введите число :
23
Введено число : 23
```

```
BufferedReader br= new BufferedReader(
                   new InputStreamReader( System.in ) );
String input = br.readLine();
System.out.println("Вы ввели строку: "+input);
```

```
проверка ввода
Вы ввели строку : проверка ввода
```

Консольный ввод-вывод. Класс Console

- ***java.io.Console***
- Основные методы:
  - `public PrintWriter writer()`
  - `public Reader reader()`
  - `public Console printf(String format, Object... args)`
  - `public String readLine()`
  - `public String readLine(String fmt, Object... args)`
  - `public char[] readPassword(String fmt, Object... args)`
  - `public char[] readPassword()`
  - `public void flush()`
- Получение объекта Console
  - `java.lang.System – static Console console()`

Учебный центр компьютерного обучения «СПЕЦИАЛИСТ» при МГТУ им. Н.Э.Баумана

10.5. ВВОД-ВЫВОД ДВОИЧНЫХ ДАННЫХ

Потоки двоичных данных (1)

- Для ввода/вывода двоичных данных примитивных типов Java используются потоки *DataInputStream* и *DataOutputStream*

- ***java.io.DataInputStream***
- Конструктор:
  - `DataInputStream(InputStream in)`
- Основные методы:
  - `void readFully(byte[] b) throws IOException`
  - `boolean readBoolean() throws IOException`
  - `char readChar() throws IOException`
  - `String readUTF() throws IOException`
  - …

- ***java.io.DataOutputStream***
- Конструктор:
  - `public DataOutputStream(OutputStream out)`
- Основные методы:
  - `void writeBoolean(boolean v) throws IOException`
  - `void writeShort(int v) throws IOException`
  - `void writeChar(int v) throws IOException`
  - `void writeUTF(String str) throws IOException`
  - …

Потоки двоичных данных (2)

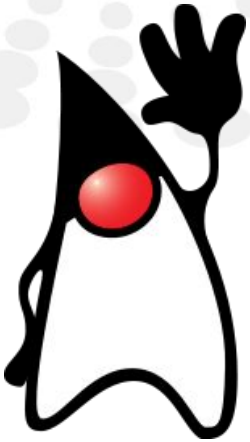
```
int[][] arr = { {1,2,3} , {4,5,6}};
String fileName = "c:\\array.bin";
try (FileOutputStream f = new FileOutputStream(fileName);
      DataOutputStream dout = new DataOutputStream(f)) {
     // Записываю размер массива
     dout.writeInt(arr.length);
     dout.writeInt(arr[0].length);
     // Записываю массив
     for (int[] line : arr)
          for (int el : line)
               dout.writeInt(el);
}
```

Потоки двоичных данных (3)

```
int[][] arr = null;
String fileName = "c:\\array.bin";
try (FileInputStream f = new FileInputStream(fileName);
      DataInputStream din = new DataInputStream(f)) {
     // Читаю размер массива
     int size1 = din.readInt();
     int size2 = din.readInt();
     // Создаю массив
     arr = new int[size1][];
     for (int i=0; i<size1; i++)
          arr[i] = new int[size2];
     // Читаю элементы
     for (int i=0; i<size1; i++)
          for (int j=0; j<size2; j++)
               arr[i][j]= din.readInt();
     // Вывожу массив
     for (int[] line : arr) {
          for (int el : line)
               System.out.print(el + " ");
          System.out.println();
     }
}
```

```
1 2 3
4 5 6
```

Учебный центр компьютерного обучения «СПЕЦИАЛИСТ» при МГТУ им. Н.Э.Баумана



11. ДОПОЛНИТЕЛЬНЫЕ ВОЗМОЖНОСТИ ООП

11.1. ПЕРЕЧИСЛЕНИЯ

Перечисления

- ***Перечисление*** (***enum type***) – пользовательский тип данных, определяющий множество констант
- Переменная типа *перечисление* должна равняться одной из заданных констант

```
public enum Day {
      MONDAY, TUESDAY, WEDNESDAY, THURSDAY,
      FRIDAY, SATURDAY, SUNDAY;
}
```

Работа с перечислениями

```
public class HelloWorld{
     public static void main(String args[]) {
          Day d = Day.WEDNESDAY;
          switch (d) {
          case MONDAY: case TUESDAY:
          case WEDNESDAY: case THURSDAY:
          case FRIDAY:
               System.out.println("Будни :(");
               break;
          default:
               System.out.println("Выходные!!!");
          }

          for (Day iter : Day.values())
               System.out.print(iter + " ");
     }
}
```

```
Будни :(
MONDAY TUESDAY WEDNESDAY THURSDAY FRIDAY SATURDAY SUNDAY
```

Методы и поля перечислений

```
public enum Day {
      MONDAY(false), TUESDAY(false), WEDNESDAY(false),
      THURSDAY(false), FRIDAY(false), SATURDAY(true), SUNDAY(true);

      private final boolean weekend;
      Day(boolean we) {weekend = we;}
      public boolean isWeekend() {return weekend;}
      public static void sayHello() {
            System.out.println("Привет от перечисления Day");
      }
}

public static void main(String args[]) {
      Day.sayHello();
      boolean isMonDayOff = Day.MONDAY.isWeekend();
      for (Day day : Day.values())
            System.out.println(day + ": " +
                  (day.isWeekend()? "выходной" : "будни"));
}
```

компьютерного Центр обучения «СПЕЦИАЛИСТ» при МГТУ им. Н.Э.Баумана®

11.2. ШАБЛОНЫ

Шаблоны

- ***Шаблон*** (***generic type***) – это класс или интерфейс, параметризованный по типам данных
- Параметрами шаблона являются ***type variables*** – переменные, значением которых являются типы данных
- ***Type variables*** могут использоваться при объявлении полей и методов шаблона
- При создании объектов класса-шаблона указываются значения ***type variables*** – типы данных, с которыми будет работать объект

```
public class MyGenerics <T> {
  private T value;
  public void setValue(T newValue)
  {
    value = newValue;
  }
  public T getValue() {
    return value;
  }
}
```

```
public class HelloWorld {
  public static void main(String args[]) {
    MyGenerics<String> g1 = new MyGenerics<String>();
    g1.setValue("Привет, мир!");
    System.out.println(g1.getValue());
    MyGenerics<Integer> g2 = new MyGenerics<>();
    g2.setValue(1_000_000);
    System.out.println(g2.getValue());
  }
}
```

компьютерного Центр обучения «СПЕЦИАЛИСТ» при МГТУ им. Н.Э.Баумана

11.3. ВЛОЖЕННЫЕ КЛАССЫ

Вложенные классы

- ***Вложенный класс*** (***nested class***) – класс, объявленный внутри другого класса
- Вложенные классы используются для:
  - логической группировки классов
  - ограничения доступа к классам
  - повышения читабельности и управляемости исходного кода

- Вложенные классы:
  - статические вложенные классы (static nested classes)
  - внутренние классы (inner classes)
  - локальные классы (local classes)
  - анонимные классы (anonymous classes)

Статические вложенные классы

- ***Статический вложенный класс*** (***static nested class***)
  - Является статическим элементом класса
  - Уровень доступа к СВК определяется спецификаторами доступа
  - СВК не может обращаться к методам и полям экземпляра внешнего класса (нестатическим)
  - Объекты СВК создаются и существуют независимо от объектов внешнего класса

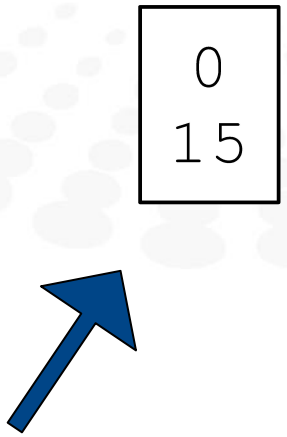
```
public class OuterClass {
  public int var1 = 0;
  ...
  public static class StaticNestedClass {
    public int var2 = 0;
    ...
  }
}
public class JavaApplication1 {
  public static void main(String[] args) {
    OuterClass o1 = new OuterClass();
    OuterClass.StaticNestedClass o2 =
                        new OuterClass.StaticNestedClass();
    o1.var1 = 10;
    o2.var2 = 20;
  }
}
```

Внутренние классы (1)

- ***Внутренний класс (inner class)***
  - Является элементом класса
  - Уровень доступа к внутреннему классу определяется спецификаторами доступа
  - Объекты внутреннего класса создаются для конкретных объектов внешнего класса
  - Внутренний класс может обращаться к методам и полям экземпляра внешнего класса
  - Внутренний класс не может содержать статических элементов

Внутренние классы (2)

```
public class OuterClass {
    private int x = 0;
    public void printX() {
        System.out.println(x);
    }

    public class InnerClass {
        public void setX(int nx) {
            x = nx;
        }
    }
}
public class JavaApplication1 {
    public static void main(String[] args) {
        OuterClass out = new OuterClass();
        out.printX();

        OuterClass.InnerClass in = out.new InnerClass();
        in.setX(15);
        out.printX();
    }
}
```



Локальные классы

```
public class JavaApplication1 {
    public static void main(String[] args) {
        int[] x = {1,2,3,4,5};

        class Power {
            private final int n;
            Power(int nval) {n = nval;}
            private int compute(int x) {
                int result = 1;
                for (int i=0; i<n; i++)
                    result*=x;
                return result;
            }
        }

        Power d3 = new Power(3);
        for (int item : x) {
            System.out.println(d3.compute(item));
        }
    }
}
```

- ***Локальный класс (local class)***
  - объявляется внутри метода или блока
  - может обращаться к элементам внешнего класса
  - может обращаться к локальным переменным блока, объявленным со спецификатором ***final***

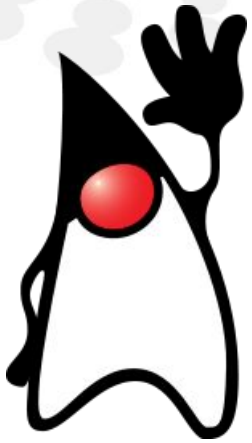
Анонимные классы (1)

- ***Анонимный класс*** (***anonymous class***)
  - локальный класс без имени
  - объявляется при создании своего объекта
  - реализует определенный интерфейс или расширяет определенный класс
  - работа с объектом анонимного класса осуществляется через интерфейсную или родительскую ссылку

Анонимные классы (2)

```
interface Computable {
    double compute(double a, double b);
}
class Summator implements Computable {
    @Override
    public double compute(double a, double b) {
        return a+b;
    }
}
public class JavaApplication1 {
    public static void main(String[] args) {
        Computable com1 = new Summator();
        Computable com2 = new Computable() {
            @Override
            public double compute(double a, double b) {
                return a-b;
            }
        };
        System.out.println(com1.compute(10, 15));
        System.out.println(com2.compute(10, 15));
    }
}
```

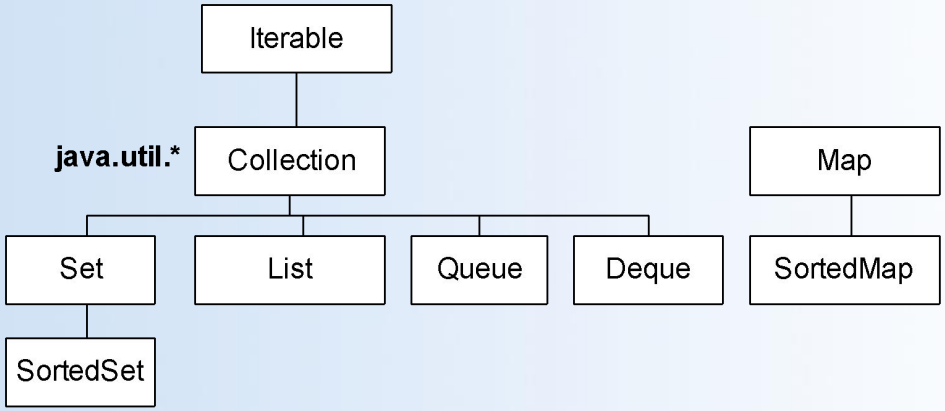
компьютерного Центр обучения «СПЕЦИАЛИСТ» при МГТУ им. Н.Э.Баумана



12. КОЛЛЕКЦИИ

Иерархия интерфейсов коллекций

- ***Коллекции*** (***collections***) или ***контейнеры*** (***containers***) предназначены для работы с группой элементов
  - Элементом коллекции является объект
  - Коллекции обеспечивают хранение элементов и доступ к ним



- Интерфейсы:
  - ***Collection*** – родительский интерфейс коллекций
  - ***Set*** – «множество» – коллекция, не допускающая наличия одинаковых элементов
  - ***List*** – «список» – коллекция элементов, следующих в определенном порядке
  - ***Queue*** – «очередь» – организует элементы в порядке FIFO
  - ***Deque*** – «двусторонняя очередь» - предоставляет доступ к элементам в порядке FIFO или LIFO
  - ***Map*** – множество элементов, доступ к которым осуществляется по ***ключу***

Интерфейс *Collection*

- ***Collection<E>***
- Основные методы:
  - `int` ***`size`***`()` – возвращает количество элементов
  - `boolean` ***`isEmpty`***`()` – проверяет, пустая ли коллекция
  - `boolean` ***`contains`***`(Object o)` – проверяет, содержится ли в коллекции заданный объект
  - `Iterator<E>` ***`iterator`***`()` – возвращает итератор
  - `Object[]` ***`toArray`***`()` – возвращает массив с элементами коллекции
  - `boolean` ***`add`***`(E e)` – обеспечивает наличие элемента в коллекции. Возвращает true, если коллекция была изменена
  - `boolean` ***`remove`***`(Object o)` – удаляет элемент из коллекции
  - `void` ***`clear`***`()` – удаляет все элементы из коллекции

Интерфейс *Set*

- ***Set<E>*** - множество уникальных объектов E
- Интерфейс содержит только методы, унаследованные от Collection
- Основные реализации:
  - ***java.util.HashSet***
  - ***java.util.TreeSet***
  - ***java.util.LinkedHashSet***

Классы *HashSet, TreeSet, LinkedHashSet*

```
Set<String> hs = new HashSet<String>();
hs.add("Иванов");
hs.add("Сидоров");
hs.add("Петров");
hs.add("Иванов");
hs.add("Алексеев");
for (String s : hs) System.out.println(s);
```

Иванов
Петров
Алексеев
Сидоров

```
Set<String> hs = new LinkedHashSet<String>();
hs.add("Иванов");
hs.add("Сидоров");
hs.add("Петров");
hs.add("Иванов");
hs.add("Алексеев");
for (String s : hs) System.out.println(s);
```

Иванов
Сидоров
Петров
Алексеев

```
Set<String> hs = new TreeSet<String>();
hs.add("Иванов");
hs.add("Сидоров");
hs.add("Петров");
hs.add("Иванов");
hs.add("Алексеев");
for (String s : hs) System.out.println(s);
```

Алексеев
Иванов
Петров
Сидоров

Интерфейс *List*

- ***List<E>*** - упорядоченная последовательность элементов E:
  - позиционный доступ (доступ по номеру элемента)
  - поиск элемента
- Основные методы:
  - `boolean` ***`add`***`(E e)` – добавляет новый элемент в конец списка
  - `void` ***`add`***`(int index, E element)` – вставляет элемент на заданную позицию
  - `boolean` ***`remove`***`(Object o)` – удаляет элемент из массива (первое вхождение)
  - `E remove(int index)` – удаляет элемент на заданной позиции
  - `void sort(Comparator<? super E> c)` – сортирует список с использованием заданного компаратора
  - `E get(int index)` – возвращает элемент на заданной позиции
  - `E set(int index, E element)` – заменяет элемент на заданной позиции
  - `int indexOf(Object o)` – возвращает индекс первого вхождения заданного элемента
  - `int lastIndexOf(Object o)` – возвращает индекс последнего вхождения заданного элемента
  - `int size()` – возвращает количество элементов в списке
  - `...`

- Реализации:
  - ***java.util.ArrayList***
  - ***java.util.LinkedList***
  - ***java.util.Vector***
  - ***java.util.Stack***
  - ...

Класс *ArrayList*

```
ArrayList<String> arr = new ArrayList<String>();
arr.add("Иванов");
arr.add("Сидоров");
arr.add("Петров");
arr.add("Иванов");
arr.add("Алексеев");
for (String s : arr) System.out.print(s + " ");
System.out.println();

System.out.println("Третий элемент: " + arr.get(3));
String toFind = "Алексеев";
int k = arr.indexOf(toFind);
System.out.println(toFind + " под номером " + k);
arr.remove(1);
arr.set(0, "Кузнецов");
for (String s : arr) System.out.print(s + " ");
```

```
Иванов Сидоров Петров Иванов Алексеев
Третий элемент : Иванов
Алексеев под номером  4
Кузнецов Петров Иванов Алексеев
```

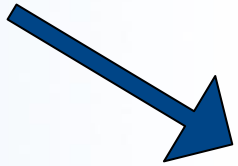
Интерфейс *Map*

- ***Map<K, V>***
  - хранит пары: значение класса V с ключом класса K
  - доступ к значениям осуществляется по ключу
  - ключ должен быть уникальным
  - в качестве ключей должны использоваться неизменяемые объекты
- Основные методы:
  - `int` ***`size`***`()` – возвращает количество пар
  - `boolean` ***`isEmpty`***`()` – проверяет наличие пар
  - `boolean` ***`containsKey`***`(Object key)` – проверяет наличие пары с заданным ключом
  - `boolean` ***`containsValue`***`(Object value)` – проверяет наличие пары с заданным ключом
  - `V` ***`get`***`(Object key)` – возвращает значение по заданному ключу
  - `V` ***`put`***`(K key, V value)` – добавляет новую пару
  - `V` ***`remove`***`(Object key)` – удаляет пару по заданному ключу и возвращает ее значение
  - `void` ***`clear`***`()` – удаляет все пары
  - `Set<K>` ***`keySet`***`()` – возвращает множество ключей
  - `Collection<V>` ***`values`***`()` – возвращает коллекцию значений
  - `Set<Map.Entry<K,V>>` ***`entrySet`***`()` – возвращает множество пар

- Реализации:
  - *java.util.HashMap*
  - *java.util.TreeMap*
  - *java.util.LinkedHashMap*
  - …

Класс *HashMap*

```
Map<Integer, String> hm = new HashMap<>();
hm.put(100, "Иванов");
hm.put(99, "Сидоров");
hm.put(2, "Петров");
hm.put(3, "Иванов");
hm.put(2, "Алексеев");
hm.put(7, "Кузнецов");

for (Map.Entry<Integer, String> e : hm.entrySet())
     System.out.println(e.getKey() + " - " + e.getValue());

System.out.println();
System.out.println(hm.get(99));
```



```
2 - Алексеев
100 - Иванов
3 - Иванов
99 - Сидоров
7 - Кузнецов

Сидоров
```

Класс *Collections*

- Класс ***java.util.Collections*** реализует стандартные алгоритмы по работе с коллекциями
- Методы:
  - `static <T> void copy(List<? super T> dest, List<? extends T> src)` – копирует элементы из одного списка в другой
  - `static int frequency(Collection<?> c, Object o)` – определяет, сколько раз заданный элемент встречается в коллекции
  - `static boolean disjoint(Collection<?> c1, Collection<?> c2)` – проверяет, что две коллекции не содержат общих элементов
  - `static <T> T max(Collection<? extends T> coll, Comparator<? super T> comp)` – находит максимальный элемент в коллекции, используя заданный компаратор
  - `static <T> T min(Collection<? extends T> coll, Comparator<? super T> comp)` – находит минимальный элемент в коллекции, используя заданный компаратор
  - `public static <T> void sort(List<T> list, Comparator<? super T> c)` – сортирует список, используя заданный компаратор
  - `static <T extends Comparable<? super T>> void sort(List<T> list)` – сортирует список по возрастанию
  - …

Сортировка списков

```
ArrayList<String> arr = new ArrayList<String>();
arr.add("Иванов");
arr.add("Сидоров");
arr.add("Петров");
arr.add("Иванов");
arr.add("Алексеев");

Collections.sort(arr);
for (String s : arr) System.out.print(s + " ");
```

```
Алексеев Иванов Иванов Петров Сидоров
```

Использование компараторов
Интерфейс *Comparator*

```
public class MyStringComp implements Comparator<String>{
  @Override
  public int compare(String s1, String s2) {
    return s2.compareTo(s1);
  }
}
```

```
ArrayList<String> arr = new ArrayList<String>();
arr.add("Иванов");
arr.add("Сидоров");
arr.add("Петров");
arr.add("Иванов");
arr.add("Алексеев");

Collections.sort(arr, new MyStringComp());
for (String s : arr) System.out.print(s + " ");
```

```
Сидоров Петров Иванов Иванов Алексеев
```

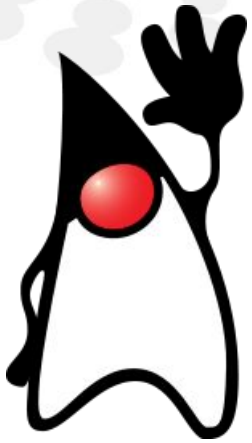
Интерфейс *Comparable*

```
public class Employee implements Comparable<Employee> {
  public String surname;
  public String name;
  public Employee(String s, String n) {
    surname = s;
    name = n;
  }
  @Override
  public int compareTo(Employee e) {
    int c = surname.compareTo(e.surname);
    if (c==0) c = name.compareTo(e.name);
      return c;
  }
}
```

```
ArrayList<Employee> arr = new ArrayList<>();
arr.add(new Employee("Сидоров","Павел"));
arr.add(new Employee("Иванов","Павел"));
arr.add(new Employee("Иванов","Игорь"));
arr.add(new Employee("Петров","Иван"));
arr.add(new Employee("Кузнецов","Иван"));
arr.add(new Employee("Иванов","Иван"));
Collections.sort(arr);
for (Employee e : arr)
   System.out.println(e.surname + " " + e.name);
```

```
Иванов Иван
Иванов Игорь
Иванов Павел
Кузнецов Иван
Петров Иван
Сидоров Павел
```

Центр компьютерного обучения «СПЕЦИАЛИСТ» при МГТУ им. Н.Э.Баумана



13. СЕРИАЛИЗАЦИЯ

Сериализация в Java
Интерфейс *Serializable*

- ***Сериализация*** (***serialization***) – преобразование состояния объекта в последовательность байт
- ***Десериализация*** (***deserialization***) – восстановление объекта из байтовой последовательности
- Объект может быть сериализован, если класс реализует интерфейс ***java.io.Serializable***

```
public class Point implements Serializable {
      private double x = 0;
      private double y = 0;

      public Point(double nx, double ny) {
            setCoord(nx, ny);
      }
      public void setCoord(double nx, double ny) {
            x = nx; y = ny;
      }
      ...
}
```

Объектные потоки

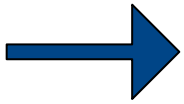
- ***java.io.ObjectInputStream***
- Конструктор:
  - `ObjectInputStream(InputStream in) throws IOException`
- Переопределяет методы InputStream:
  `read, close, ...`
- Основные методы:
  - `final Object readObject() throws IOException, ClassNotFoundException`
  - `float readFloat() throws IOException`
  - `int readInt() throws IOException`
  - `...`

- ***java.io.ObjectOutputStream***
- Конструктор:
  - `ObjectOutputStream(OutputStream out) throws IOException`
- Переопределяет методы OutputStream:
  `write, flush, close`
- Основные методы:
  - `final void writeObject(Object obj) throws IOException`
  - `void writeFloat(float val) throws IOException`
  - `void writeInt(int val) throws IOException`
  - `...`

Запись объекта в поток
Чтение объекта из потока

```
String fileName = "points.bin";
Point[] arr = {new Point(10,20), new Point(11,8),
                    new Point(7,-6)};

try (FileOutputStream fout = new FileOutputStream(fileName);
     ObjectOutputStream oout = new ObjectOutputStream(fout)) {
     oout.writeInt(arr.length);
     for (Point p : arr)
          oout.writeObject(p);
}
```

```
String fileName = "points.bin";
try (FileInputStream fin = new FileInputStream(fileName);
     ObjectInputStream oin = new ObjectInputStream(fin)) {
     int count = oin.readInt();
     for (int i=0; i<count; i++) {
          Point p = (Point)oin.readObject();
          p.println();
     }
}
```



```
10.0, 20.0
11.0, 8.0
7.0, -6.0
```

Спецификатор *transient*

- Спецификатор ***transient*** используется для обозначения полей класса, которые не будут сохраняться при сериализации

```
public class Point implements Serializable {
      private transient double x = 0;
      private double y = 0;
      ...
}
```

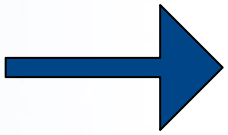
```
String fileName = "points.bin";
Point[] arr = {new Point(10,20), new Point(11,8),
                  new Point(7,-6)};

try (FileOutputStream fout = new FileOutputStream(fileName);
      ObjectOutputStream oout = new ObjectOutputStream(fout)) {
      oout.writeInt(arr.length);
      for (Point p : arr)
            oout.writeObject(p);
}
```

```
String fileName = "points.bin";
try (FileInputStream fin = new FileInputStream(fileName);
      ObjectInputStream oin = new ObjectInputStream(fin)) {
      int count = oin.readInt();
      for (int i=0; i<count; i++) {
            Point p = (Point)oin.readObject();
            p.println();
      }
}
```

```
0.0, 20.0
0.0, 8.0
0.0, -6.0
```

Некоторые особенности сериализации (1)

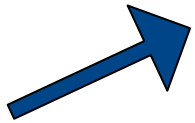
```
String fileName = "points.bin";
try (FileOutputStream fout = new FileOutputStream(fileName);
          ObjectOutputStream oout = new ObjectOutputStream(fout)) {
     Point p = new Point(99,99);
     oout.writeObject(p);
     p.setCoord(1, 1);
     oout.writeObject(p);
}

try (FileInputStream fin = new FileInputStream(fileName);
     ObjectInputStream oin = new ObjectInputStream(fin)) {
     Point p1 = (Point)oin.readObject();
     Point p2 = (Point)oin.readObject();
     p1.println();
     p2.println();
     System.out.println(p1 == p2);
     p1.setCoord(200, 200);
     p2.println();
}
```



```
99.0, 99.0
99.0, 99.0
true
200.0, 200.0
```

Некоторые особенности сериализации (2)

```
String fileName = "c:\\points.bin";
ArrayList<Point> points = new ArrayList<>();
points.add(new Point(1,1)); points.add(new Point(2,2));
Point p = new Point(3,3);
points.add(p); points.add(p);

try (FileOutputStream fout = new FileOutputStream(fileName);
      ObjectOutputStream oout = new ObjectOutputStream(fout)) {
     oout.writeObject(points);
}

try (FileInputStream fin = new FileInputStream(fileName);
     ObjectInputStream oin= new ObjectInputStream(fin)) {
     ArrayList<Point> newlist = (ArrayList<Point>)oin.readObject();
     for (Point x : newlist) x.println();
     System.out.println("-----");
     newlist.get(3).setCoord(100, 100);
     for (Point x : newlist) x.println();
}
```

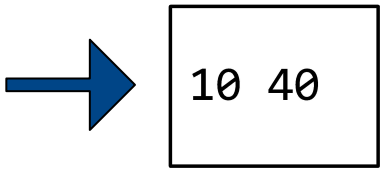


```
1.0, 1.0
2.0, 2.0
3.0, 3.0
3.0, 3.0
-----
1.0, 1.0
2.0, 2.0
100.0, 100.0
100.0, 100.0
```

Объявление собственных методов *writeObject* и *readObject*

```
public class TestSer implements Serializable{
  private int a;
  private transient int b;
  TestSer(int x, int y) {a=x;b=y;}
  public void println() {System.out.println(a + " " + b); }
  private void writeObject(ObjectOutputStream out) throws IOException{
    out.defaultWriteObject();
    out.write(b);
  }
  private void readObject(java.io.ObjectInputStream in)
          throws IOException, ClassNotFoundException {
    in.defaultReadObject();
    b = in.read()*2;
  }
}
```

```
String fileName = "c:\\test.bin";
try (FileOutputStream fout = new FileOutputStream(fileName);
      ObjectOutputStream oout = new ObjectOutputStream(fout)) {
     oout.writeObject(new TestSer(10,20));
}
try (FileInputStream fin = new FileInputStream(fileName);
     ObjectInputStream oin = new ObjectInputStream(fin)) {
     ((TestSer)oin.readObject()).println();
}
```



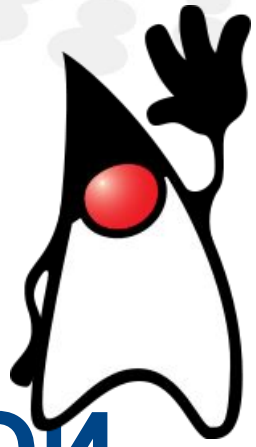
Интерфейс *Externalizable*

- Интерфейс ***Externalizable***:
  - наследуется от Serializable
  - требует определения собственных протоколов сохранения и чтения данных об объекте

```
public class TestSer implements Externalizable{
  private int a;
  private int b;
  public TestSer() {a=0;b=0;}
  public TestSer(int x, int y) {a=x;b=y;}
  ...
  @Override
  public void writeExternal(ObjectOutput out) throws IOException {
    out.writeInt(a);
    out.writeInt(b);
  }
  @Override
  public void readExternal(ObjectInput in)
         throws IOException, ClassNotFoundException {
    a = in.readInt();
    b = in.readInt();
  }
}
```

Serializable vs Externalizable

- ***Seralizable***:
  - сохранение состояния объекта осуществляется на основе стандартных алгоритмов
    - сохранение унаследованных полей осуществляется автоматически
    - сохранение внешних объектов (на которые ссылается сериализуемый объект) осуществляется автоматически
  - стандартные алгоритмы могут быть переопределны в методах ***readObject*** и ***writeObject***
  - при десериализации конструктор объекта не вызывается
- ***Externalizable***:
  - сохранение состояния объекта осуществляется пользовательскими алгоритмами
    - сохранение унаследованных полей должно реализовываться классом самостоятельно
  - при десериализации вызывается конструктор объекта по умолчанию, после чего состояние объекта восстанавливается методом ***readExternal***
  - методы ***readExternal*** и ***writeExternal*** заменяют методы ***readObject*** и ***writeObject***
  - в определенных случаях использование Externalizable позволяет повысить производительность сериализации



14. РАБОТА С ФАЙЛОВОЙ СИСТЕМОЙ

Учебный центр компьютерного обучения «СПЕЦИАЛИСТ» при МГТУ им. Н.Э.Баумана

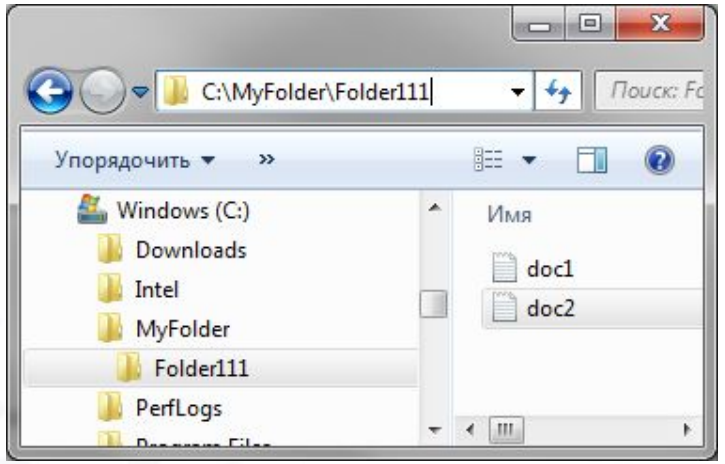
14.1. ИСПОЛЬЗОВАНИЕ КЛАССА FILE

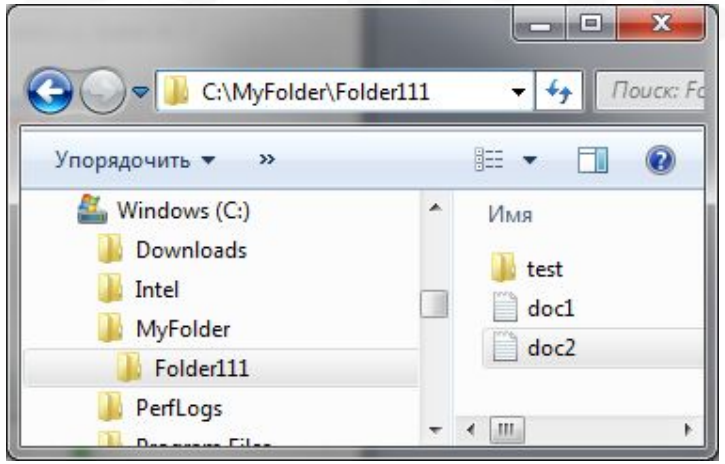
Класс *java.io.File*

- Класс ***java.io.File*** используется для работы с файлами и папками
- Конструкторы:
  - `File(String pathname)`
  - `File(String parent, String child)`
  - …
- Основные методы:
  - `String getName()`
  - `String getParent()`
  - `String getPath()`
  - `String getAbsolutePath()`
  - `boolean exists()`
  - `boolean isDirectory()`
  - `boolean isFile()`
  - `long length()`
  - `boolean createNewFile() throws IOException`
  - `boolean delete()`
  - `File[] listFiles()`
  - `boolean mkdir()`
  - `boolean mkdirs()`
  - `boolean renameTo(File dest)`
  - `Path toPath()`
  - …

Работа с классом File (1)

```
File f = new File("C:\\MyFolder",
                  "Folder111\\test");
System.out.println(f.exists());
System.out.println(f.getParent());
System.out.println(f.getName());
System.out.println(f.getPath());
System.out.println(f.mkdir());
```

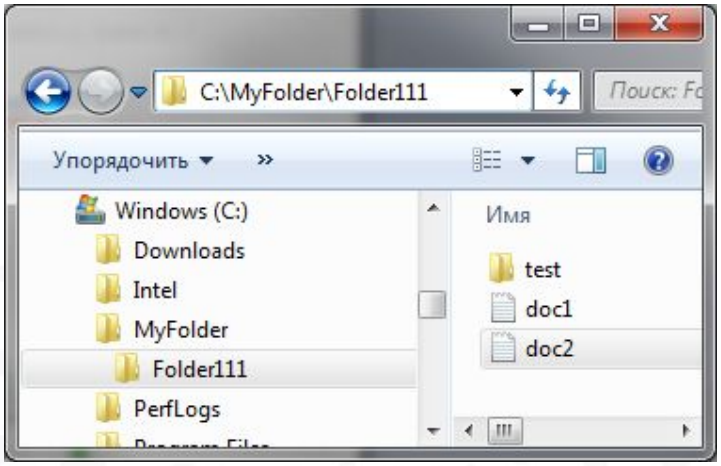





```
false
C:\MyFolder\Folder111
test
C:\MyFolder\Folder111\test
true
```

Работа с классом File (2)

```
File f = new File(
    "C:\\MyFolder\\Folder111");
File[] list = f.listFiles();
for (File item : list)
     System.out.println(
        item.getName() + " " +
        item.isDirectory() + " " +
        item.length() );
```




```
doc1.txt false 21
doc2.txt false 21
test true 0
```

Учебный центр компьютерного обучения «СПЕЦИАЛИСТ» при МГТУ им. Н.Э.Баумана

14.2. ВВЕДЕНИЕ В NIO.2

NIO

- ***Java NIO*** (*Java non-blocking input/output, Java new input/output*)
  - `java.nio.*`
  - высокопроизводительный программный интерфейс для управления файловым вводом-выводом и работы с файловой системой
  - позволяет использовать низкоуровневые возможности современных операционных систем
  - поддерживает асинхронный ввод-вывод
- Основные классы:
  - *java.nio.file.Path* – определяет размещение файла/папки
  - *java.nio.file.Paths* – управляет созданием объектов *Path*
  - *java.nio.file.Files* – управляет объектами файловой системы
  - *java.nio.Buffer* – контейнер данных, используемый для передачи информации
    - java.nio.ByteBuffer – байтовый буфер
  - *java.nio.channels.Channe*l – канал для операций ввода-вывода
    - *java.nio.channels.ByteChannel* – канал для чтения и записи байт

Классы *java.nio.file.Path* и *java.nio.file.Paths*

- ***java.nio.file.Path*** – интерфейс, используемый для определения размещения файла/папки
- Основные методы:
  - `Path` **`getRoot`**`()`
  - `Path` **`getParent`**`()`
  - `Path` **`getFileName`**`()`
  - `Path` **`getParent`**`()`
  - `int` **`getNameCount`**`()`
  - `Path` **`getName`**`(int index)`
  - `File` **`toFile`**`()`
  - …

- ***java.nio.file.Paths*** – класс, используемый для создания объектов *Path*
- Основные методы:
  - `static Path` **`get`**`(String first, String... more)`

Работа с объектами Path

```
Path p = Paths.get("C:\\MyFolder\\Folder111");
System.out.println(p);
System.out.println(p.getFileName());
System.out.println(p.getParent());
System.out.println(p.getRoot());
System.out.println(p.getNameCount());
System.out.println(p.getName(0));
System.out.println(p.getName(1));
```

```
C:\MyFolder\Folder111
Folder111
C:\MyFolder
C:\
2
MyFolder
Folder111
```

Класс Files. Управление файлами и папками (1)

- ***java.nio.file.Files*** – управляет объектами файловой системы и отвечает за операции ввода/вывода
- Методы по управлению файлами и папками:
  - `static boolean exists(Path path, LinkOption... options)`
  - `static boolean notExists(Path path, LinkOption... options)`
  - `static boolean isDirectory(Path path, LinkOption... options)`
  - `static boolean isReadable(Path path)`
  - `static boolean isWritable(Path path)`
  - `static Path createDirectory(Path dir, FileAttribute<?>... attrs) throws IOException`
  - `static Path createDirectories(Path dir, FileAttribute<?>... attrs) throws IOException`
  - `static Path createFile(Path path, FileAttribute<?>... attrs) throws IOException`
  - `static void delete(Path path) throws IOException`
  - `static Path copy(Path source, Path target, CopyOption... options) throws IOException`
  - `static Path move(Path source, Path target, CopyOption... options) throws IOException`
  - `static boolean isSameFile(Path path, Path path2) throws IOException`

Класс Files. Управление файлами и папками (2)

```
Path p1 = Paths.get("C:\\MyFolder\\Folder111\\myfile.txt");
if (Files.notExists(p1))
    Files.createFile(p1);
Path p2 = Paths.get("C:\\MyFolder\\Folder111\\folder");
if (Files.notExists(p2))
    Files.createDirectory(p2);
Path p3 = Paths.get(p2.toString(), "myfile2.txt");
Files.copy(p1, p3, StandardCopyOption.REPLACE_EXISTING);
```

Работа с атрибутами файлов

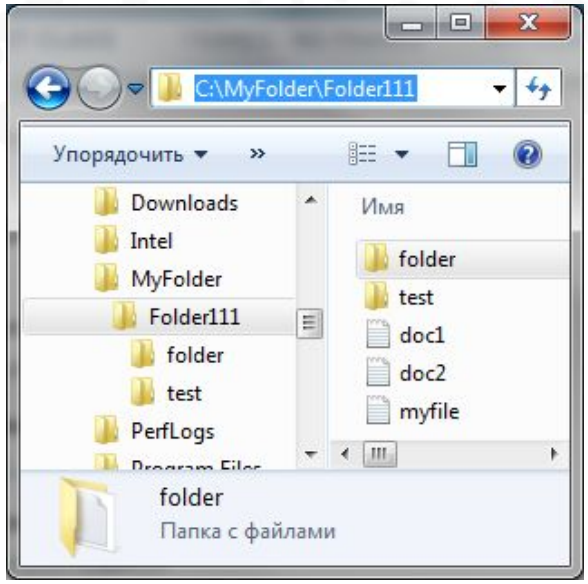
- Методы ***Files*** по работе с атрибутами файла:
  - static long **size**(Path path) throws IOException
  - `static Object getAttribute(Path path, String attribute, LinkOption... options) throws IOException`
  - `static Path setAttribute(Path path, String attribute, Object value, LinkOption... options) throws IOException`
  - `static boolean isHidden(Path path) throws IOException`
  - `static FileTime getLastModifiedTime(Path path, LinkOption... options) throws IOException`
  - `static Path setLastModifiedTime(Path path, FileTime time) throws IOException`
  - `static UserPrincipal getOwner(Path path, LinkOption... options) throws IOException`
  - `static Path setOwner(Path path, UserPrincipal owner) throws IOException`
  - `static <A extends BasicFileAttributes> A readAttributes(Path path, Class<A> type, LinkOption... options) throws IOException`

```
Path p = Paths.get("C:\\MyFolder\\Folder111\\myfile.txt");
FileTime ft1 = (FileTime)Files.getAttribute(p, "lastModifiedTime");
FileTime ft2 = (FileTime)Files.getAttribute(p, "lastAccessTime");
long size = (Long)Files.getAttribute(p, "size");
```

Обход дерева файловой системы (1)

- Методы ***Files*** для работы с дочерними элементами каталога:
  - `static DirectoryStream<Path> newDirectoryStream(Path dir) throws IOException`
  - `static Path walkFileTree(Path start, FileVisitor<? super Path> visitor) throws IOException`

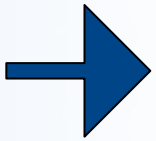
```
Path p = Paths.get("C:\\MyFolder\\Folder111");
try (DirectoryStream<Path> stream =
        Files.newDirectoryStream(p)) {
  for (Path item : stream)
    System.out.println(item);
}
```




```
C:\MyFolder\Folder111\doc1.txt
C:\MyFolder\Folder111\doc2.txt
C:\MyFolder\Folder111\folder
C:\MyFolder\Folder111\myfile.txt
C:\MyFolder\Folder111\test
```

Обход дерева файловой системы (2)

```
FileVisitor<Path> fw = new FileVisitor<Path>() {
  @Override public FileVisitResult preVisitDirectory(
      Path dir, BasicFileAttributes attrs) throws IOException {
    return FileVisitResult.CONTINUE;
  }
  @Override public FileVisitResult visitFile(
      Path file, BasicFileAttributes attrs) throws IOException {
    System.out.println(file);
    return FileVisitResult.CONTINUE;
  }
  @Override public FileVisitResult visitFileFailed(
      Path file, IOException exc) throws IOException {
    return FileVisitResult.TERMINATE;
  }
  @Override public FileVisitResult postVisitDirectory(
      Path dir, IOException exc) throws IOException {
    return FileVisitResult.CONTINUE;
  }
};
Path path = Paths.get(
    "C:\\MyFolder\\Folder111");
Files.walkFileTree(path, fw);
```



```
C:\MyFolder\Folder111\doc1.txt
C:\MyFolder\Folder111\doc2.txt
C:\MyFolder\Folder111\folder\myfile2.txt
C:\MyFolder\Folder111\myfile.txt
C:\MyFolder\Folder111\test\doc3.txt
```

Возможности ввода-вывода NIO (1)

- Методы ***Files*** по созданию потоков, каналов и вводу-выводу:
  - `static BufferedReader newBufferedReader(Path path, Charset cs) throws IOException`
  - `static BufferedWriter newBufferedWriter(Path path, Charset cs, OpenOption... options) throws IOException`
  - `static InputStream newInputStream(Path path, OpenOption... options) throws IOException`
  - `static OutputStream newOutputStream(Path path, OpenOption... options) throws IOException`
  - `static byte[] readAllBytes(Path path) throws IOException`
  - `static List<String> readAllLines(Path path, Charset cs) throws IOException`
  - `static Path write(Path path, byte[] bytes, OpenOption... options) throws IOException`
  - `static Path write(Path path, Iterable<? extends CharSequence> lines, Charset cs, OpenOption... options) throws IOException`
  - `static SeekableByteChannel newByteChannel(Path path, OpenOption... options) throws IOException`

Возможности ввода-вывода NIO (2)

```
List<String> lines = new ArrayList<>();
lines.add("Однажды, в студеную зимнюю пору,");
lines.add("Я из лесу вышел, был сильный мороз");
Path p = Paths.get("d:\\poetry.txt");
Files.write(p, lines, Charset.forName("Windows-1251"),
            StandardOpenOption.CREATE);

List<String> lines2 = Files.readAllLines(p,
            Charset.forName("Windows-1251"));
for (String line : lines2)
  System.out.println(line);
```

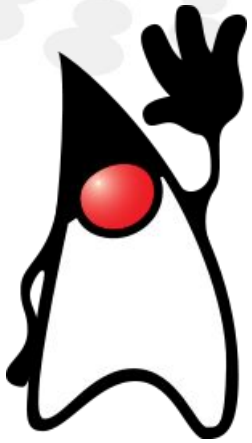
```
Однажды, в студеную зимнюю пору
Я из лесу вышел, был сильный мороз
```

Канальный ввод-вывод

- *Канальный ввод-вывод (channel I/O)*
  - ***Канал (channel)*** используется для выполнения операций ввода-вывода
  - Один канал может использоваться одновременно для ввода и вывода
  - Каналы поддерживают асинхронный ввод-вывод
  - Запись и чтение данных из канала осуществляется через ***буфер*** (***buffer***)

```
Path p = Paths.get("d:\\poetry.txt");
try (SeekableByteChannel sbc = Files.newByteChannel(
                    p,StandardOpenOption.READ)) {
  ByteBuffer buf = ByteBuffer.allocate(10);
  int count = 0;
  while ( (count = sbc.read(buf)) > 0) {
    buf.flip();
    System.out.print(
        Charset.forName("Windows-1251").decode(buf));
    buf.clear();
  }
}
```

Учебный центр компьютерного обучения «СПЕЦИАЛИСТ» при МГТУ им. Н.Э.Баумана



15. ШАБЛОНЫ ПРОЕКТИРОВАНИЯ

Шаблоны проектирования (1)

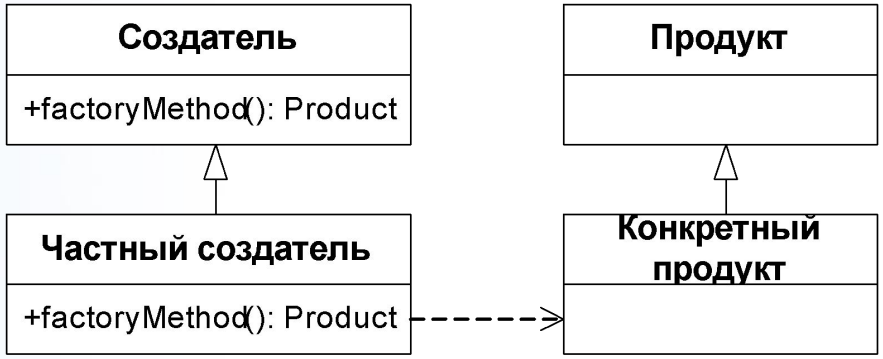
- ***Шаблоны проектирования*** (***design patterns***) – архитектурная конструкция, используемая при решении часто возникающих задач программирования
- Шаблоны показывают взаимоотношения между классами и объектами
- Широкое распространение получили 23 шаблона (***GoF patterns***) проектирования, опубликованные в книге «***Design Patterns: Elements of Reusable Object-Oriented Software***» (1994)
- Классификация шаблонов:
  - ***creational patterns*** (***порождающие шаблоны***) – шаблоны, связанные с процессом создания объектов
  - ***structural patterns*** (***структурные шаблоны***) – шаблоны, отвечающие за организацию связей между различными классами
  - ***поведенческие шаблоны*** (***behavioral patterns***) – шаблоны, определяющие алгоритмы взаимодействия объектов

Шаблоны проектирования (2)

- Creational
  - Abstract factory (абстрактная фабрика)
  - Builder (строитель)
  - Factory Method (фабричный метод)
  - Prototype (прототип)
  - Singleton (одиночка)
- Structural
  - Adapter (адаптер)
  - Bridge (мост)
  - Composite (компоновщик)
  - Decorator (декоратор)
  - Facade (фасад)
  - Flyweight (приспособленец)
  - Proxy (заместитель)
- Behavioral
  - Chain of responsibility (цепочка обязанностей)
  - Command (команда)
  - Interpreter (Интерпретатор)
  - Iterator (итератор)
  - Mediator (посредник)
  - Memento (хранитель)
  - Observer (наблюдатель)
  - State (состояние)
  - Strategy (стратегия)
  - Template method (шаблонный метод)
  - Visitor (посетитель)

Factory method (1)

- Шаблон предоставляет подклассам интерфейс для создания объектов, принадлежащих определенной иерархии классов
  - *Продукт* – абстрактный класс (или интерфейс), определяющий поведение категории целевых объектов
  - *Конкретный продукт* – наследник класса *Продукт*
  - *Создатель* (*Фабрика*) – абстрактный класс (или интерфейс), отвечающий за создание целевых объектов – объектов иерархии *Продукт*. Определяет методы для создания продуктов и работы с ними
  - *Частный создатель* – класс-наследник *Создателя*. Определяет, какой конкретный продукт нужно создавать



Factory method (2)

```
abstract class Product { }
class Product1 extends Product {}
class Product2 extends Product {}

abstract class Factory {
    public abstract Product createProduct();
}
class Factory1 extends Factory {
    @Override
    public Product createProduct() {return new Product1();}
}
class Factory2 extends Factory {
    @Override
    public Product createProduct() {return new Product2();}
}

public class JavaApplication{
    public static void main(String[] args) {
        Factory factory = new Factory1();
        Product product = factory.createProduct();
    }
}
```

Singleton

- Шаблон гарантирует, что класс будет иметь единственный экземпляр

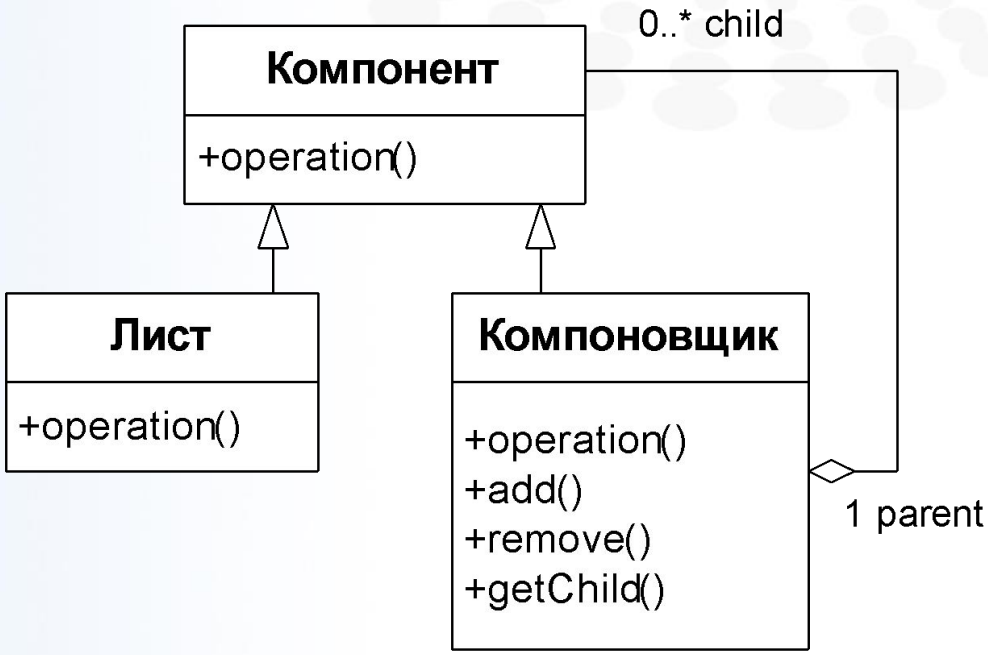
```
class Singleton {
   private static Singleton instance
                        = new Singleton();
   public static Singleton getInstance() {
       return instance;
   }
   private Singleton() {}
   ...
}
```

```
enum SingletonEnum {
    INSTANCE;

    public static SingletonEnum getInstance() {
        return INSTANCE;
    }
    ...
}
```

Composite (1)

- Используется при работе с древовидными структурами
- Позволяет одинаковым образом обращаться к отдельным объектам и к группам объектов

  – *Компонент* – родительский класс (интерфейс) для элементов дерева
  – *Лист* – одиночный элемент
  – *Компоновщик* – группа элементов

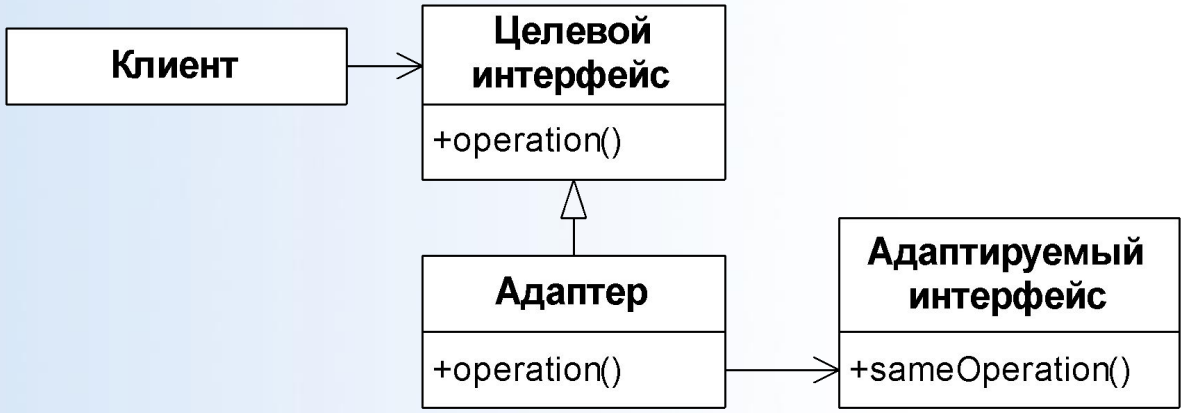


Composite (2)

```
interface FileSystemObject {
    public void delete();
    public void rename(String nName);
    ...
}
class Folder implements FileSystemObject {
    @Override public void delete() {...}
    @Override public void rename(String nName) {...}
    ...
    private List<FileSystemObject> l = new ArrayList<>();
    public void add(FileSystemObject o) { l.add(o);   }
    public void remove(FileSystemObject o) {l.remove(o); }
}
class File implements FileSystemObject {
    @Override public void delete() {...}
    @Override public void rename(String nName) {...}
    ...
}
```

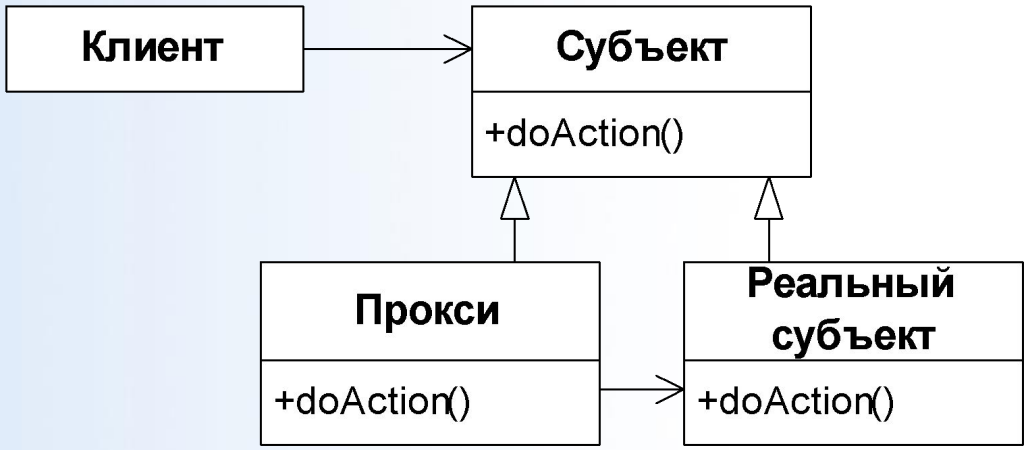
Adapter

- Адаптеры необходимы в том случае, когда класс имеет соответствующие методы для поддержки определенного поведения, но не реализует необходимые интерфейсы
  - *Клиент* умеет работать с одним интерфейсом (*Целевой интерфейс)*, при этом хочет вызвать объект, реализующий другой интерфейс (*Адаптируемый интерфейс*).
  - *Целевой интерфейс* и *Адаптируемый интерфейс* имеют схоже методы, но не являются родственниками в иерархии наследования
  - Адаптер – класс, преобразующий вызовы одного интерфейса в другой (*Целевого* в *Адаптируемый)*



Proxy

- Шаблон позволяет контролировать доступ к другому объекту, поддерживающему аналогичный интерфейс
  - *Реальный субъект* – объект, с которым должен взаимодействовать клиент
  - *Субъект* – абстрактный класс (или интерфейс), от которого наследуется *Реальный субъект*
  - *Прокси* – класс, наследуемый от *Субъекта,* выступающий в качестве обертки для *Реального субъекта*. Передает вызовы *Клиента Реальному субъекту*



Observer

- Определяет механизм для класса, который позволяет получать оповещения от объектов других классов при изменении их состояния
  - *Наблюдаемый субъект* – класс, изменение состояние которого доступно для наблюдения. Помимо основного функционала содержит методы по добавлению, удалению и оповещению наблюдателей
  - *Частный наблюдатель* – класс, отслеживающий изменение состояния *Наблюдаемого субъекта*
  - *Наблюдатель* – абстрактный класс (или интерфейс), с помощью которого *Наблюдаемый субъект* передает оповещения *Частному наблюдателю*