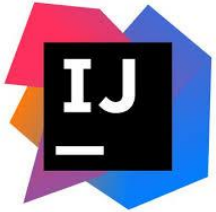
Основы Java

Татьяна Берленко, tatiana.berlenko@gmail.com

Немного истории

- Java разработана компанией Sun Microsystems;
- дата официального выпуска: 23 мая 1995 года;
- Java 7 была куплена компанией Oracle;
- На данный момент последняя версия языка - Java 8 (релиз 19 марта 2014 года).

Что можно писать на Java



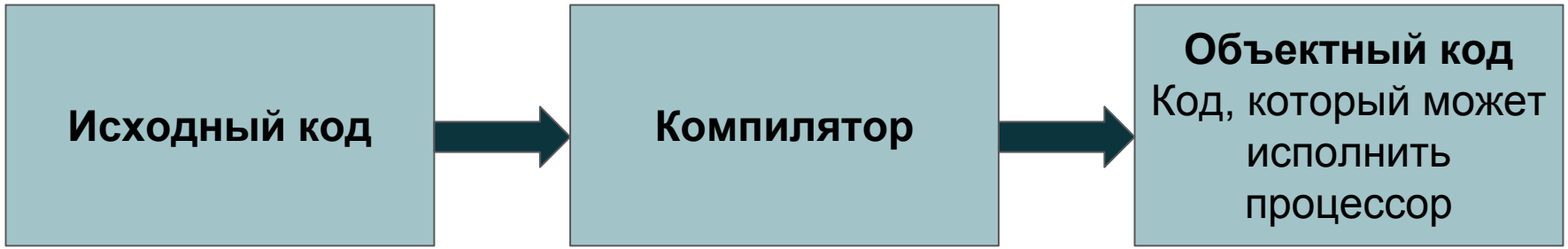
amazon

Вопросы аудитории

1. Что такое компиляция/компилятор?

Компиляция - процесс преобразования программы с исходного языка высокого уровня в эквивалентную программу на машинном языке.
Компилятор - программа, которая осуществляет компиляцию.

2. Что поступает компилятору на вход, а что он возвращает в качестве результата?



Вопросы аудитории

3. Бывают ли языки без компилятора?

Бывают.

4. Что такое интерпретация и чем она отличается от компилятора?

Интерпретация - пооператорный анализ, обработка и выполнение исходной программы или запроса.

5. Есть ли компилятор у языка Java? JavaScript? C++?

У Java и C++ да, у JavaScript нет.

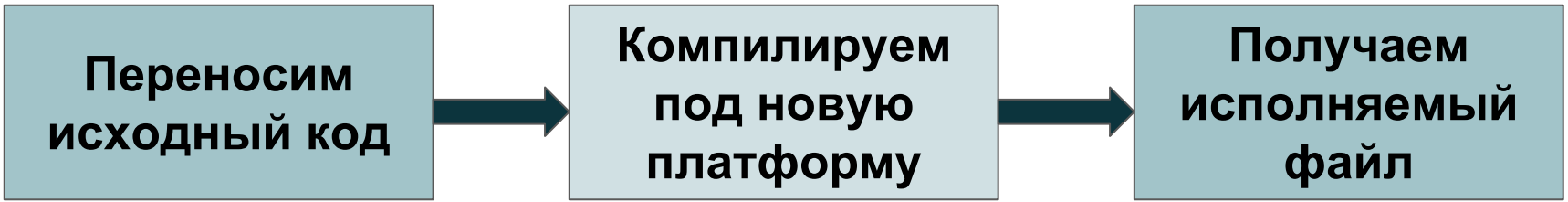
Вопросы аудитории

6. Если скомпилировать программу на C++ на Windows, запустится ли она на MacOS/Linux? А если язык Java? Python?

C++ - нет, Java - да, Python - да.

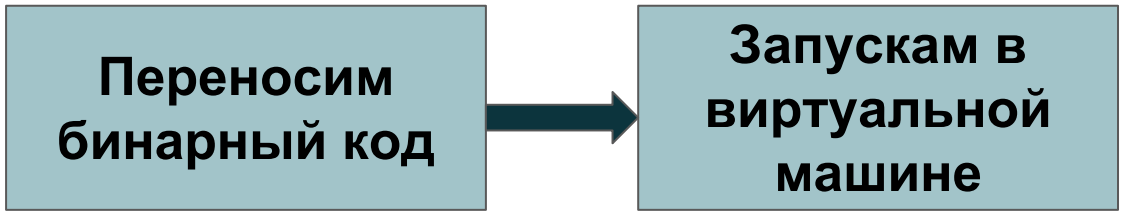
Виды переносимости программ

Программа перекомпилируется под новую платформу.



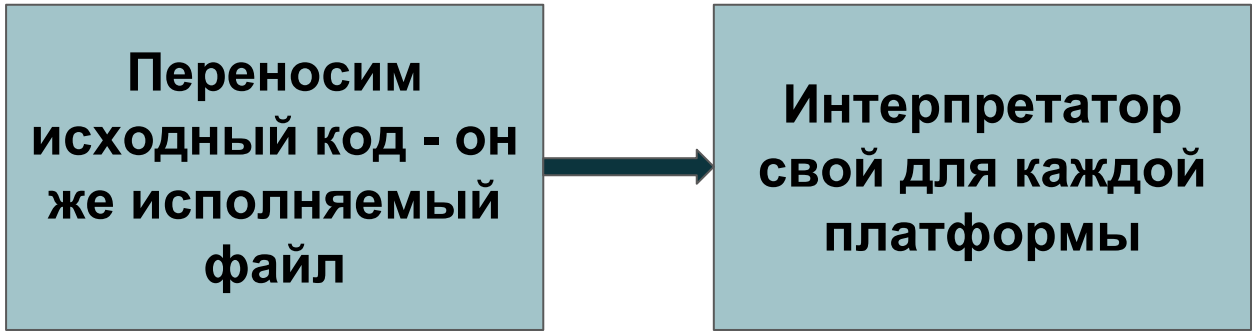
Виды переносимости программ

Переносим бинарный файл (бинарные файлы). Не будут работать только специфичные функции конкретной платформы.

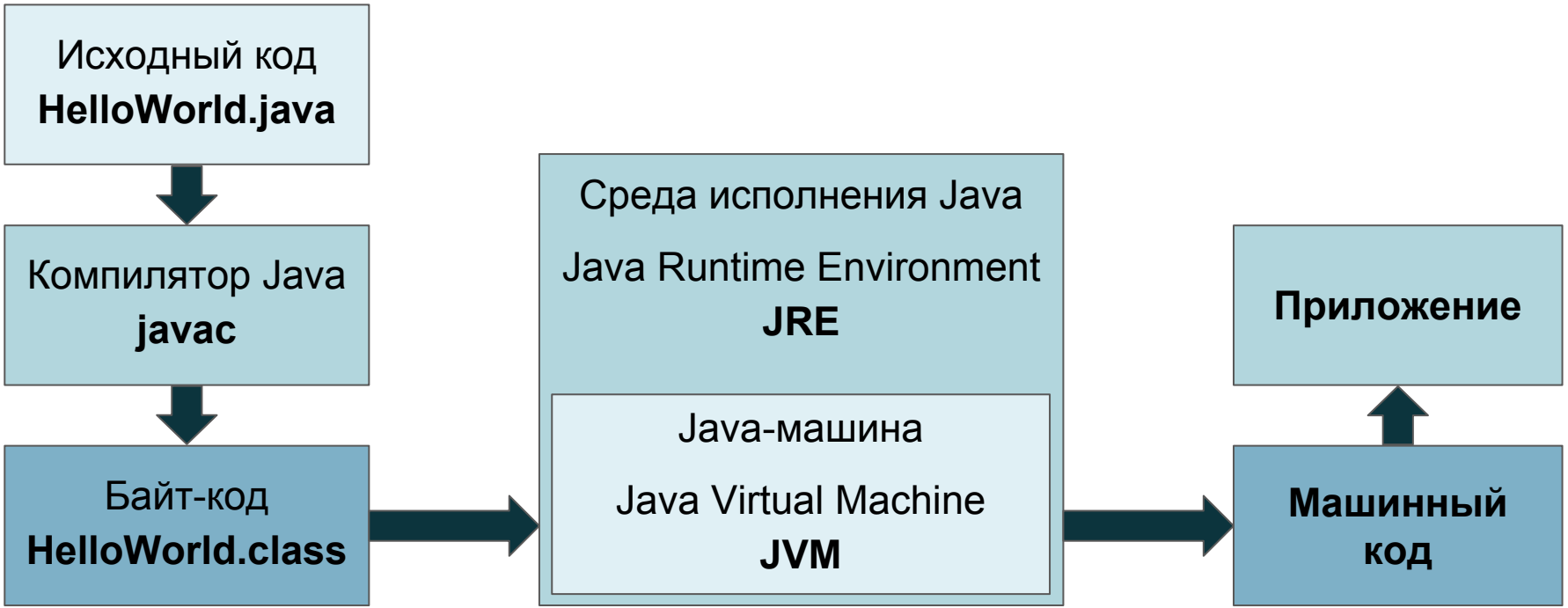


Виды переносимости программ

Программа интерпретируется.



Write once, run anywhere!



JDK & JRE

Если вы хотите просто **запустить** ваше приложение, написанное на Java, вам нужно скачать **JRE** (Java Runtime Environment). Это Java-машина (виртуальная машина для запуска приложений) + библиотека Java-классов.

Если же вы хотите **разрабатывать** приложения на языке Java, вам необходимо скачать **JDK** (Java Development Kit). **JDK** включает в себя компилятор **javac**, **JRE**, стандартные библиотеки классов Java, примеры, документацию.

Поскольку **JDK** включает в себя **JRE**, скачивать и то и то **не нужно**.

DEMO
HELLO WORLD IN INTELLIJ IDEA

Пример программы “Hello World!”

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

Байт-код для программы HelloWorld

```
public class HelloWorld {
  public HelloWorld();
    Code:
       0: aload_0
       1: invokespecial #1                  // Method java/lang/Object."<init>":()V
       4: return

  public static void main(java.lang.String[]);
    Code:
       0: getstatic     #2                  // Field java/lang/System.out:Ljava/io/PrintStream;
       3: ldc           #3                  // String Hello World!
       5: invokevirtual #4                  // Method java/io/PrintStream.println:(Ljava/lang/String;)V
       8: return
}
```

Что можно почитать

The Java® Virtual Machine Specification

https://docs.oracle.com/javase/specs/jvms/se7/html/index.html

Пример программы с вводом имени пользователя

```
import java.util.Scanner;

public class HelloUser {
    public static void main(String[] args) {
        System.out.println("Please, input your name");
        String userName = getUserName();
        System.out.println("Hello, " + userName);
    }

    public static String getUserName(){
        Scanner scanner = new Scanner(System.in);
        String userName = scanner.nextLine();
        return userName;
    }
}
```

Примитивные типы данных

Целые числа - byte, short, char, int, long
Числа с плавающей точкой (иначе дробные) - float, double
Логический тип - boolean

```
int intExample = 10;
System.out.println(intExample); // 10

double doubleExample = 10;
System.out.println(doubleExample); // 10.0

float floatExample = 10f; // эквивалентно 10F
System.out.println(floatExample); // 10.0
```

Оператор условия if

```
if(a == 100)
    a += 100;
else if(a > 100)
    a = 0;
else
    a *= 10;
```

```
if(a == 100){
    a += 100;
    b = 0;
}
else
    a *= 10;
```

Циклы

while

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

do...while

```
int i = 0;
do{
  i ++;
 System.out.println(i);
} while(i > 10)
```

Циклы

for

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

for each

```
int [] array = {10,11,12,13,14,15};
for (int i:array) {
   System.out.println(i);
}
```

Оператор switch

```
int season = 2;
switch (season){
   case 1:
      System.out.println("Зима");
      break;
   case 2:
      System.out.println("Весна");
      break;
   case 3:
      System.out.println("Лето");
      break;
   case 4:
      System.out.println("Осень");
      break;
   default:
      System.out.println("Ваше число неверно");
}
```

```
int season = 2;
switch (season){
   case 1:
      System.out.println("Зима");
      break;
   case 2:
      System.out.println("Весна");
   case 3:
      System.out.println("Лето");
      break;
   case 4:
      System.out.println("Осень");
      break;
   default:
      System.out.println("Ваше число неверно");
}
```

DEMO
IN INTELLIJ IDEA

Пример программы с подсчетом суммы чисел в массиве

```
public class NumberInArray {
   public static void main(String[] args) {
      int numbers[] = {50, -9, -10, 45, 20, -4, 10, 0, 20, 25, 10, 25};
      for (int number:numbers) {
         System.out.print(element + " ");
      }
      int length = numbers.length;
      int sum = 0;
      int count = 0;
      for (int i = 0; i < length; i++) {
         if(sum >= 100)
            break;
         sum += numbers[i];
         count++;
      }
      System.out.println("\nЧтобы получить сумму 100, нужно сложить " + count + " чисел");
   }
}
```

Задачи для самостоятельного решения

1. Напишите программу, которая получает на вход массив чисел размера 10 и выводит максимальное число.
2. Напишите программу, которая получает на вход число и выводит все простые числа, которые меньше этого числа.

Строки. Примеры создания

```
String testString = "this is my new string";
```

```
String testString = new String("this is my new string");
```

```
char[] array = {'t', 'h', 'i', 's', ' ', 'i', 's', 'm', 'y', 'n', 'e', 'w', ' ', 's', 't', 'r', 'i', 'n', 'g'};
String testString = new String(array);
```

Строки. Как получить число из строки

```
int intNumber = Integer.parseInt("1111");
```

```
float floatNumber = Float.parseFloat("25.0f");
```

```
double d = Double.parseDouble("25.0");
```

Строки. Как получить строку из числа

```
String fromInt = Integer.toString(1);

String fromFloat = Float.toString(2.0f);

String fromDouble = Double.toString(2.0);
```

DEMO STRINGS
IN INTELLIJ IDEA

Пример работы со строками

```
public class StringsExample {
  public static void main(String[] args) {
      String testString = new String("Это моя новая строка");
      System.out.println("testString = " + testString);
      System.out.println("\ntestString[4] вызовет ошибку!");
      System.out.println("\nСимвол строки по номеру символа можно получить так: <строка>.charAt(<номер>).\nНапример, четвертый символ testString:
testString.charAt(4) // = " + testString.charAt(4));
      System.out.println("Длину строки можно получить так: <строка>.length().\nНапример, длина строки testString: testString.length() // = " + testString.length());
      System.out.println("Если мы будем сравнивать строки таким образом: testString == \"Это моя новая строка\", то результат будет: " + (testString == "Это моя новая
строка"));
      System.out.println("Правильно сравнивать строки нужно с помощью <строка>.equals(): testString.equals(\"Это моя новая строка\"), результат: " +
testString.equals("Это моя новая строка"));
      System.out.println("Для того, чтобы прибавить к строке другую строку можно использовать <строка_1>.concat(<строка_2>)");
      System.out.println("Результат такого сложения обязательно нужно присвоить какой-нибудь переменной, например: <строка_1> =
<строка_1>.concat(<строка_2>)");
      System.out.println("Пример: testString = testString.concat(\" с новыми символами\") // = " + testString.concat(" с новыми символами"));
      System.out.println("Это аналогично записи testString += \" с новыми символами\"");
      System.out.println("Найти индекс символа 'o' с начала строки: testString.indexOf('o') = " + testString.indexOf('o'));
      System.out.println("Индекс символа 'o' с конца строки: testString.lastIndexOf('o') = " + testString.lastIndexOf('o'));
      System.out.println("Индекс несуществующего символа 's': testString.indexOf('s') = " + testString.indexOf('s'));
      System.out.println("Для замены в строке одного символа на другой используйте testString.replace(<символ_который_заменяем>,
<символ_которым_заменяем>)");
      System.out.println("Пример: заменим в строке testString пробел на перевод каретки");
      System.out.println("testString.replace(\" \", \"\\n\") \nРезультат:\n" + testString.replace(" ", "\n"));
  }
}
```

Основы ООП в Java

Основные понятия. Объект

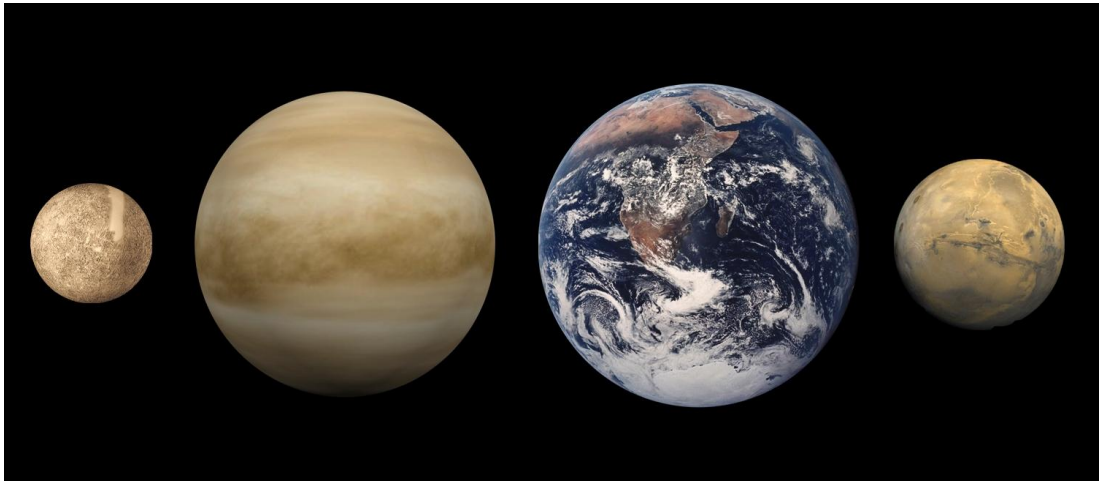
Объект - конкретная сущность предметной области.

Основные понятия. Класс

Класс - это тип объекта. Или говорят, что объект - экземпляр класса.

Класс: **Планета**.

Объекты: **Меркурий, Венера, Земля, Марс**.



Основные понятия. Поля и методы

Поле - характеристика (атрибут) класса. Для каждого объекта поле будет принимать конкретное значение.

Например: класс **Планета** содержит поле **среднийРадиус**.

среднийРадиус Венеры 6051,8 км

среднийРадиус Земли 6371,0 км

среднийРадиус Марса 3389,5 км

Основные понятия. Поля и методы

Метод - функция класса.

Например: класс **Планета** содержит метод **получитьСреднийРадиус()**

Для объекта **Венера** метод вернет 6051,8 км

Для объекта **Земля** метод вернет 6371,0 км

Для объекта **Марс** метод вернет 3389,5 км

Пример

Planet.java

```
public class Planet {

   String averageRadius;

   String getAverageRadius(){
      return averageRadius;
   }

}
```

Создание объектов. Конструктор

Конструктор - специальный метод, который создает объект.

У конструктора в описании **отсутствует** тип возвращаемого значения.

Конструкторов может быть несколько, но тогда у них должны быть разные параметры.

```
Planet(String averageRadius) {
   this.averageRadius = averageRadius; // текущий экземпляр класса
}

Planet() {
   this.averageRadius = "0.0";
}
```

Создание объектов. Конструктор

Конструктор вызывается при вызове ключевого слова **new**.

```
Planet Earth = new Planet("6371,0 км");
```

По умолчанию у вас всегда есть конструктор по умолчанию без параметров. Например, для нашего класса Planet конструктор по умолчанию, который бы вызывался java-машиной, был бы такой:

```
Planet() {
  this.averageRadius = "";
}
```

Соответственно, вызов:

```
Planet Earth = new Planet();
```

Пример использования класса Planet

```
public class PlanetUsage {

   public static void main(String[] args) {

      Planet Earth = new Planet("6371,0 км");

      String averageRadiusEarth = Earth.getAverageRadius();

      System.out.println(averageRadiusEarth);

   }

}
```

Задачи для самостоятельного решения

Задание: создать свой класс для хранения библиотечной карточки и класс для её использования (вызов конструктора, вызов методов).

Уничтожение объектов. Garbage Collector

В некоторых языках существует метод, который вызывается для уничтожения объектов. Такой метод называется деструктор.

В таких языках управление памятью полностью доверено программисту.

В Java существует Garbage Collector (сборщик мусора). Это такой процесс, который уничтожает объекты, которые больше не будут нужны программе.

Таким образом, в Java не нужно думать об освобождении памяти.

Структура программы на языке Java

- Любая программа должна содержать хотя бы один класс.
- Классы объединяются в **пакеты**.
- Подключение классов из разных пакетов выполняется с помощью ключевого слова import.

Пример

пакет java_examples

пример импорта класса Scanner

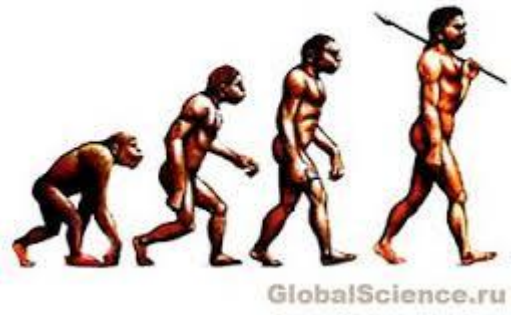
```
package java_examples;

import java.util.Scanner;

public class HelloUser {
   public static void main(String[] args) {
      System.out.println("Please, input your name");
      String userName = getUserName();
      System.out.println("Hello, " + userName);
   }
   public static String getUserName(){
      Scanner scanner = new Scanner(System.in);
      String userName = scanner.nextLine();
      return userName;
   }
}
```

Наследование

Наследование - это возможность создания нового класса на основе существующего.

В Java класс, **от** которого наследуются (**класс-предок**), называется **суперкласс**.

В Java наследоваться можно *только от одного класса*.



Пример

```
public class Monkey {

   String name;
   int age;

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

}
```

Пример

```
public class Human extends Monkey {
   Human(){
      super("Big Big Monkey", 1); // вызов конструктора суперкласса обязателен
   }
}
```

Модификаторы доступа

Модификатор	Видимость везде	Видимость у наследников	Видимость в пакете
public	+	+	+
protected	-	+	+
Отсутствие модификатора	-	-	+
private	-	-	-

Пример

```
public class PublicPrivateProtectedModificatorsExample {

    public String name;
    protected String phone;
    private String pinCode;

    public void sayName() {
        System.out.println(name);
    }
    protected void sayPhone() {
        System.out.println(phone);
    }
    private void sayPinCode(){
        System.out.println(pinCode);
    }
}
```

доступно везде

доступно наследникам и в пакете

доступно только в классе

доступен везде

доступен наследникам и в пакете

доступен только в классе

Модификаторы доступа

Модификатор	Класс	Поле	Метод
final	**Нельзя создать класс-наследник**	**Нельзя изменять значение поля**	**В классе-наследнике нельзя переопределить метод**
static	**Не применим (к классам верхнего уровня)**	**Поле одно для всех экземпляров класса**	**Метод может быть вызван без создания объекта класса**

Пример

```
public class FinalStaticModificatorsExample {

    private static int i = 0;
    private final String msg = "HELLO, WORLD!";

    public final void sayHi() {
        System.out.println(msg);
    }

    public static void main(String[] args) {
        System.out.println(i);
    }

}
```

значение одинаково во всех объектах

значение поля нельзя изменить

метод нельзя переопределить в наследниках

метод можно вызвать, не создавая объекта класса

Интерфейсы

- Все поля - публичные константы.
- Все методы - публичные и не имеют реализации.
- Классы могут наследоваться от множества интерфейсов (говорят, что класс может реализовать интерфейс).

Явно писать модификатор public необязательно!

Пример

```
public interface MammalInterface {
   public static final boolean isCheeksPresent = true; // У всех млекопитающих есть щёки
   void sleep();
}
```

```
public class Monkey implements MammalInterface {
   private String name;

   public void sleep() { // Этот метод обязательно должен быть реализован!
      System.out.println("zzzzzzzz");
   }

   public boolean getIsCheeksPresent(){
      return isCheeksPresent; // все поля интерфейса доступны наследникам
   }
}
```

Полиморфизм

Возможность использования объектов разных классов одинаковым образом, при этом не задумываясь о типе объекта.

Полиморфизм позволяет работать с классами-наследниками таким же образом, как с классом-родителем.

Пример полиморфизма

```
public class Mammal {
    public void sleep(){
        System.out.println("I am sleeping");
    }
}
```

```
public class Horse extends Mammal{
    public void sleep() {
        System.out.println("hrrrrr");
    }
}
```

```
public class Monkey extends Mammal {
    public void sleep() {
        System.out.println("zzzzzzzz");
    }
}
```

Пример полиморфизма

```
public class PolymorphismExample {
    public static void main(String[] args) {
        Mammal monkey = new Monkey();
        Mammal horse = new Horse();
        monkey.sleep(); // zzzzzzzz
        horse.sleep(); // hrrrrr
    }
}
```

Examples

https://github.com/OSLL/JB_school_practice_2017_public

Спасибо за внимание!

Пример getters & setters

```
public class GettersAndSettersExample {
        private String name; // поля принято делать приватными

        public String getName() { // геттер возвращает значение приватного поля
                return name;
        }

        public void setName(String name) { // сеттер устанавливает значение приватного поля
                this.name = name;
         }
 }
```

Некоторые особенности дробных примитивных типов

```
float a = 1.0f;

float b = 2.0f;

System.out.println(a/b);      // 0.5

System.out.println(a/0);      // Infinity

System.out.println(-a/0);     // -Infinity

System.out.println(0/0);      // Ошибка! нельзя делить на 0 целое число

System.out.println(0.0/0);    // NaN (Not a Number)
```

Конструктор базового класса

Если в базовом классе нет ни одного конструктора или же есть конструктор по умолчанию, т.е. **без параметров,** он будет вызван в конструкторе класса-наследника автоматически.

Пример

```
public class Monkey {
   String name;
   int age;
   Monkey(){ // конструктор по умолчанию
      this.name = “”;
      this.age = 0;
   }
}
```

Пример

```
public class Human extends Monkey {
   Human(){
      // вызов конструктора суперкласса необязателен
   }
}
```