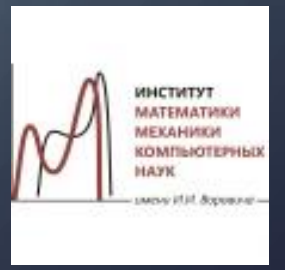


ТЕХНОЛОГИИ JAVA БИБЛИОТЕКИ МНОГОПОТОЧНОЙ ОБРАБОТКИ ДАННЫХ

ВВЕДЕНИЕ В КУРС

ЧЕРДЫНЦЕВА М.И.

2026



О ЧЕМ ЭТОТ КУРС



- Классы и интерфейсы, пакеты, тестирование
- Параметризация типов
- Коллекции
- Потоки данных
- Организация ввода-вывода
- Многопоточное программирование
- Работа в сети
- Библиотеки организации GUI

РЕЗУЛЬТАТЫ (было раньше)



- Задания (80 баллов)
- Итоговый тест (10 баллов)
- Лабораторные работы (10 баллов)

ЛИТЕРАТУРА



- Кей С. Хортсманн. Java SE 9. Базовый курс
- Кей Хорстманн. Java. Библиотека профессионала. (в 2-х томах)
- Брюс Эккель. Философия Java
- Брайан Гетц и др. Java concurrency на практике
- Официальная документация от Oracle. https://docs.oracle.com/javase/8/docs/api/

JAVA И ДРУГИЕ ЯЗЫКИ ПРОГРАММИРОВАНИЯ

- Java - универсальный язык прикладного программирования.
- Java - компилируемый, платформонезависимый, объектно-ориентированный язык с C-образным синтаксисом, со статической, явной и сильной типизацией.
- Версии Java 1.0 и 1.1 принято называть Java 1. Все

версии Java начиная с 1.2 принято называть Java 2, Java 5, Java 9 и т.п.

- Начиная с Java 8 кроме основной появляются LTS выпуски (для корпоративного использования)
- На настоящий момент последняя версия Java 25

ИСТОРИЯ

- 23 марта, 1995 — День рождения Java. James Goslings, Sun Microsystems
- Январь 1996 — Выпуск JDK 1.0
- Декабрь 1998 — Основание Java Community Process (JCP)
- 15 июня, 1999 — Декларация трех основных направлений Java платформы: J2SE, J2EE, J2ME
- Апрель 2001 — Java 2 Platform, Enterprise Edition (J2EE) SDK 1.3
- 1 октября 2004 — Выпуск Tiger (Java 5.0)
- 27 января 2010 — Oracle купил активы Sun Microsystems
- 2019 — появляется коммерческая лицензия Oracle

ПОЧЕМУ JAVA? АКТУАЛЬНОСТЬ В 2025+

- Java – язык enterprise-разработки
- Стабильно в **топ-3** рейтингов популярности
- **Миллионы** устройств по всему миру
- **Фундамент для:** Android-приложений, крупных банковских систем, веб-сервисов (LinkedIn, Airbnb, Netflix), Big Data (Hadoop, Spark), IoT

ПОЧЕМУ JAVA? АКТУАЛЬНОСТЬ В 2025+

- Кроссплатформенность (JVM)
- Богатейшая экосистема (библиотеки, фреймворки)
- Продуманность, надёжность, безопасность
- Огромное сообщество и рынок труда

ВИРТУАЛЬНАЯ JAVA-МАШИНА, БАЙТ-КОД, JIT-КОМПИЛЯЦИЯ

- Переносимость программ. Байт-код.
- Виртуальная Java-машина (JVM).
- Объектная ориентированность.
- Надёжность кода – уровень исходных кодов и уровень времени выполнения.
- JIT-компиляция (Just-In-Time).
- Автоматическая сборка мусора
- “Родной” код (native code).
- Многоязычное программирование на виртуальной машине Java (Groovy, Scala, Clojure).

ИСТОРИЯ

- 23 марта, 1995 — День рождения Java. James Goslings, Sun Microsystems
- Январь 1996 — Выпуск JDK 1.0
- Декабрь 1998 — Основание Java Community Process (JCP)
- 15 июня, 1999 — Декларация трех основных направлений Java платформы: J2SE, J2EE, J2ME
- Апрель 2001 — Java 2 Platform, Enterprise Edition (J2EE) SDK 1.3
- 1 октября 2004 — Выпуск Tiger (Java 5.0)
- 27 января 2010 — Oracle купил активы Sun Microsystems
- 2019 — появляется коммерческая лицензия Oracle

ТЕМА ДЛЯ ОБСУЖДЕНИЯ

СРАВНИВАЕМ

- Как работает компилятор C++
- Сборка программы C++
- Выполнение программы C++

- Как работает javac
- ?
- Выполнение программы java

SDK - SOFTWARE DEVELOPMENT KIT



SDK – Комплект разработки программного обеспечения.

Имеется несколько сборок для разных технологий:

- **Java ME** – комплект Java Micro Edition, предназначенный для программирования устройств с малым объемом памяти и малой разрядностью процессора.
- **Java SE** – комплект Java Standard Edition, предназначенный для программирования приложений и апплетов.
- **Java EE**– комплект Java Enterprise Edition, предназначенный для написания серверного программного обеспечения, развертывания сложных систем.
- **Java Card** —технология предоставляет безопасную среду для приложений, работающих на смарт-картах и других устройствах с очень ограниченным объёмом памяти и возможностями обработки

JRE

- JRE представляет собой комбинацию виртуальной машины Java (JVM) и библиотек, необходимых для выполнения кода во время выполнения
- Если JDK - это пакет инструментов для разработки приложений Java, то JRE - это пакет инструментов для запуска приложений Java
- JRE входит в JDK

ВИДЫ ПРОГРАММ JAVA

- Приложение (application) – аналог “обычной” прикладной программы. Может иметь оконный интерфейс. Запускается под управлением java-машины.
- Апплет (applet) – специализированная программа, работающая в окне WWW-документа под управлением браузера.
- Сервлет (servlet) - специализированная программа, работающая на стороне сервера.
- Компонент EJB (Enterprise JavaBeans) – предназначен для многократного использования серверными приложениями Java.
- Пакет– библиотека классов Java предназначена для многократного использования программами Java. Может быть в виде архива (jar).
- Модуль – агрегация пакетов и ресурсов

ПЕРЕМЕННЫЕ И ТИПЫ. ПРИМИТИВНЫЕ И ССЫЛОЧНЫЕ ТИПЫ

- **Типы** в Java делятся на **примитивные** и **ссылочные**. Существует несколько (8) предопределённых примитивных типов, все остальные – ссылочные.
- Все пользовательские типы кроме типов-перечислений являются ссылочными. Значение null соответствует ссылочной переменной, которой не назначен адрес ячейки с данными.
- Язык Java является **регистро-чувствительным**.
- **Идентификаторы** - это имена переменных, классов, методов и т.д. В идентификаторах можно применять только буквы и цифры, причём первой всегда должна быть буква, а далее может идти произвольная комбинация букв и цифр. Длина идентификатора в Java любая.
- Традиционные соглашения стиля именования (coding standards)

ПРИМЕР ПРИЛОЖЕНИЯ

```
import java.util.Random;

public class HelloWorld {
  public static void main(String args[]){
    System.out.println("Hello World!!! Java application!!!");
     for (int i=0; i<args.length; i++)
        System.out.println("Param "+i+" = "+args[i]);
   Random generator = new Random();
   System.out.println( generator.nextInt());
 }
}
```

ВЫПОЛНЕНИЕ

- Файл с текстом HelloWorld.java (имя регистрозависимое совпадает с именем класса)
- Компилируем
  - javac HelloWorld.java
- Результат компиляции – файл HelloWorld.class
- Выполнение
- Приложения
  - java HelloWorld

КЛАССЫ И ОБЪЕКТЫ

- Объект создаётся с помощью вызова *конструктора*
- Методы делятся на *методы объектов* и *методы классов*. Метод объекта можно вызывать только для объекта соответствующего типа. А метод класса может работать и при отсутствии объекта, и вызываться из класса.
- Переменные ссылочного типа содержат адреса данных, а не сами данные. Поэтому присваивания для таких переменных меняют адреса, но не данные. Все объектные типы являются ссылочными.
- На один объект может существовать множество ссылок.
- Потеря последней ссылки на объект приводит к сборке мусора.

ОПИСАНИЕ И ТЕСТИРОВАНИЕ КЛАССА

- Самотестирование

```
public class MyClass {
    ....                                    //поля
    ....                                    // методы
   public static void main ( String [ ] args) {
     MyClass  mc = new MyClass( . . .);
    // проверка методов класса
}
```

ОПИСАНИЕ И ТЕСТИРОВАНИЕ КЛАССА

- Через тестирующий класс

```
[public] class MyClassTest {
    public static void main ( String [ ] args) {
      MyClass  mc = new MyClass( . . .);
      // проверка методов класса
}
```

- Через технологию Unit-тестирования

ПАКЕТЫ

- Аналогия с библиотеками
  - Средство объединения классов
  - Механизм распространения группы классов, связанных с решением одной задачи
- Стандартная библиотека java – набор пакетов
- Иерархическая вложенность пакетов
- Пакеты обеспечивают уникальность имени класса
- Чтобы обеспечить абсолютную уникальность имени пакета, компания Sun рекомендовала использовать доменное имя компании в Интернет (которое по определению уникально), записанное в обратном порядке.
- Единственная цель вложенных пакетов— гарантия уникальности имен. С точки зрения компилятора между вложенными пакетами нет абсолютно никакой связи.

ПАКЕТЫ

```
package com.horstmann.corejava;

public class Employee{

. . .

}
```

- Каталог пакета

com\horstmann\corejava

- Путь к каталогу пакета задается переменной classpath

ПАКЕТЫ

- Класс может использовать все классы из собственного пакета и все *открытые* классы из других пакетов.

```
java.util.Date today = new java.util.Date( );
```

Или

```
import java.util. * ;   / / import  java.util.Date;
Date today = new Date();
```

ДОСТУП

- Классы, поля и методы имеют модификатор доступа (public, private, protected, в пределах пакета)
- Если ни один модификатор доступа не указан, то сущность (т.е. класс, метод или переменная) является доступной всем методам в том же самом *пакете*

МЕТОДЫ

- Передача параметров только по значению
- Допустима перегрузка методов (требуется разная сигнатура)
- Для обращения к объекту, вызвавшему метод – this
- Для вызова конструктора из другого конструктора – this()

НАСЛЕДОВАНИЕ

```
public class Base {
 .....
 }
public class Derived extends Base {
    .....
}
```

- любой класс – наследник Object
- допустимо только одиночное наследование

НАСЛЕДОВАНИЕ

- в конструкторе производного класса автоматически вызывается конструктор без параметров базового класса
- для вызова другого конструктора используется super()
- вызов конструктора базового класса должен быть первой командой в конструкторе производного класса
- в классе-наследнике допустимы и переопределение и перегрузка методов базового класса
- для обращения к методам базового класса используется super
- для всех методов, кроме final, используется позднее связывание

OBJECT

- Определяем методы, применимые к объекту любого класса

  public String toString()

  public boolean equals(Object other)

  public int hashCode()

  public Class<?> getClass()

  protected Object clone()

  protected void finalize()

  Для управления потоками Thread – wait(), notify(), notifyAll()

ПОЛИМОРФИЗМ

```
class Employee {
  public Employee(String n, double s,
                  int year){. . .}
. . .
}
class Manager extends Employee {
public Manager(String n, double s, int year)
  {
    super(n, s, year);
    bonus = 0;
  }
. . .
}
```

```
public static void main(String[] args)    {
    Manager boss = new Manager("Carl Cracker", 80000,
1987);
    boss.setBonus(5000);
    Employee[] staff = new Employee[3];
    staff[0] = boss;
    staff[1] = new Employee("Harry Hacker", 50000, 1989);
    staff[2] = new Employee("Tom Tester", 40000, 1990);
    for (Employee e : staff)
      System.out.println("name=" + e.getName() +
                         ",salary=" + e.getSalary());
}
```

АБСТРАКТНЫЕ КЛАССЫ

- Абстрактным называется класс, у которого хотя бы один метод не имеет реализации (абстрактный метод)

```
abstract class Instrument  {
        . . .
                public abstract void sound() ;
}
```

- Можно описывать переменные объекты абстрактного класса
- Нельзя создавать экземпляры объектов абстрактного класса
- Наследник абстрактного класса должен переопределить все его абстрактные методы или он тоже будет абстрактным классом

ВНУТРЕННИЕ КЛАССЫ

- **Inner classes** — внутренние классы (non static nested classes, нестатические вложенные классы)
- Объект внутреннего класса не может существовать без объекта «внешнего» класса
- У объекта внутреннего класса есть доступ к полям и методам «внешнего» класса
- При создании объекта внутреннего класса, в него незаметно передается ссылка на объект «внешнего» класса (this)

```
public class OuterClass {
...
   private InnerClass obj;

   class InnerClass {
      ... }
}
```

ЛОКАЛЬНЫЕ КЛАССЫ

```
public class OuterClass {
  public void someMethod(){
        class LocalClass{ . . .
                    }
   . . .
  }
}
```

Доступен только в методе, где он определен

У объекта локального класса есть доступ к полям и методам «внешнего» класса

АНОНИМНЫЙ КЛАСС

Локальный класс без имени. Наследует какой-то класс, или реализует какой-то интерфейс

Доступен только в методе, где он определен

У него есть доступ к полям и методам «внешнего» класса

```
public class OuterClass {
    public void someMethod(){
            Callable callable = new Callable() {
            @Override
            public Object call() throws Exception { return null; }
            };
}
```

СТАТИЧЕСКИЙ ВНУТРЕННИЙ КЛАСС

Объект статического класса **не хранит ссылку на конкретный экземпляр внешнего класса**

Объект статического вложенного класса может существовать сам по себе.

В этом плане статические классы более «независимы», чем нестатические.

Единственный момент — при создании такого объекта нужно указывать название внешнего класса

```
public class OuterClass {
  public static class StaticInnerClass{
  }
}

OuterClass.StaticInnerClass obj =
        new OuterClass.StaticInnerClass();
```

МОДИФИКАТОРЫ STATIC И FINAL

- **static** — модификатор, применяемый к полю, блоку, методу или внутреннему классу. Данный модификатор указывает на привязку к текущему классу
  - *Важно – в статическом методе нельзя использовать не статические элементы класса*
- **final** — модификатор, применяемый к классам, методам и полям. Данный модификатор указывает на запрет изменения.
- **effectively final** (не модификатор, а свойство переменной)— означает, что добавление модификатора **final** не приведет к ошибке компиляции

ИНТЕРФЕЙС

- Интерфейс описывает протокол доступа к классу
- В интерфейсе определяются только заголовки методов класса
- Если в интерфейсе определены поля, то они автоматически являются статическими. Для инициализации используется статическая секция
- Классы могут реализовывать один или несколько интерфейсов
- Если класс реализует интерфейс он должен определить реализацию всех методов этого интерфейса
- Если хотя бы один из методов остается неопределенным, класс должен быть описан как абстрактный
- Начиная с Java 9 введены методы статические (static) и по умолчанию (default)

ИНТЕРФЕЙС

```
interface StackInt {
    void push (int  value);
    bool isEmpty();
    int  pop();
    int  peek();
}
```

```
class Array100StackInt implements StackInt {
    private int array [ ];
    private int top;
    public Array100StackInt () {
            array = new int[100]; top = 0;
    }
    public void push (int  value) { array [top++]=value;}
    public bool isEmpty() { return (top ==0);}
    public int  pop() { top--; return array [top];}
    public int  peek() { return array[top-1];}
    public bool isFull() {return top==100;} //расширение
}
```

ИНТЕРФЕЙС

```
Array100StackInt  s1 = new  Array100StackInt();
StackInt   s2 = new   Array100StackInt();
```

для s2 нельзя использовать вызов

```
s2.isFull()
```