ТЕХНОСФЕРА

Занятие №3

Основы Java

Евгений Чернов

Что такое Java?

Язык программирования

- По определению Gosling, Joy, и Steele в Java Language Specification

Платформа

- Virtual machine (JVM) definition.
- Runtime environments на различной архитектуре.

Библиотека классов

- Стандартный APIs для GUI, data storage, processing, I/O и networking.

Зачем нужна Java?

Java сильно отличается от C/C++

- Нет указателей! (garbage collection)
- Обработка ошибок
- Потоки являются частью языка
- Богатая стандартная библиотека
- Совместимость на уровне исходного кода и байткода
- Нет препроцессинга

Java для программистов C++

Все есть объект

- Каждый объект наследуется от java.lang.Object

Нет кода вне описания класса!

- Нет глобальных переменных.

Одиночное наследование

- Дополнительный тип наследования: интерфейсы

Все классы определяются в `.java` files

- Один top level public class на файл

Нет заголовочных файлов

- Объявления классов сохраняются в .class файл

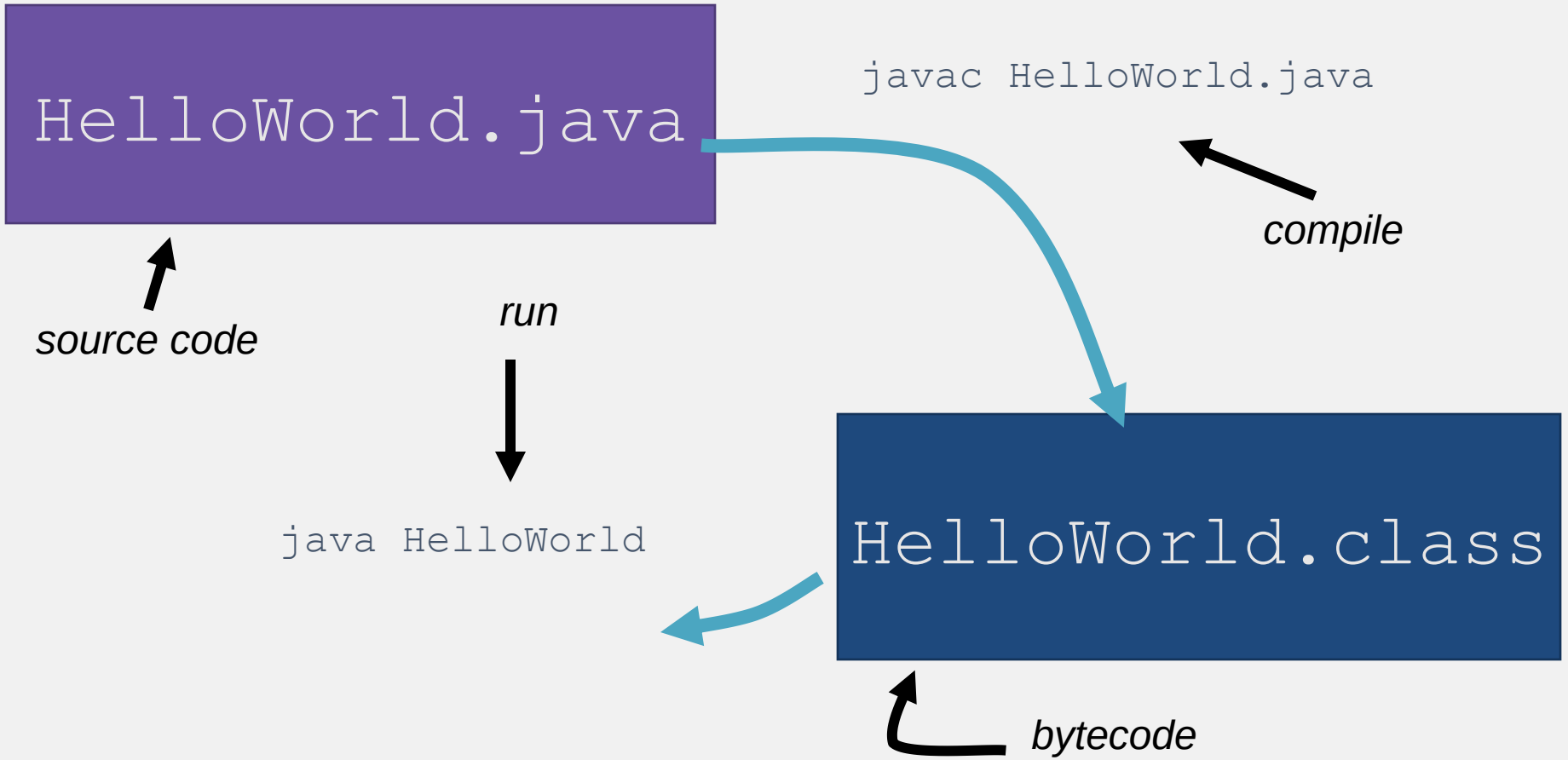
Java для программистов C++

- Похожий синтаксис
- Похожие primitive data types
  - bool это не int
- Для печати в stdout:
  - System.out.println();

Первая программа

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

Компиляция и запуск



Java bytecode и интерпретатор

bytecode – это промежуточное представление программы (class).

Интерпретатор Java запускает новую “Virtual Machine”.

VM начинает выполнение пользовательского класса путем запуска метода **`main()`**

PATH и CLASSPATH

Директория *java_home/bin* в вашем $PATH

Если вы используете другие классы, которые не в java или javax пакетах, то их местоположение должно быть включено в $CLASSPATH

Язык

1. Типы данных (Data types)
2. Операторы (Operators)
3. Управляющие структуры (Control Structures)
4. Классы и объекты (Classes and Objects)
5. Пакеты (Packages)

Типы данных в Java

Примитивы:

- **`boolean`** **`true`** или **`false`**
- **`char`** unicode! (16 bits)
- **`byte`** signed 8 bit integer
- **`short`** signed 16 bit integer
- **`int`** signed 32 bit integer
- **`long`** signed 64 bit integer
- **`float`**,**`double`** IEEE 754 floating point

Другие типы данных

Reference types (composite)

- classes
- arrays

Строки поддерживаются через встроенный класс **`String`**

Строковые литералы поддерживаются в языке (как специальный случай).

Преобразование типов (Type Conversions)

Преобразование между типами integer и floating point.

- Это включает и **`char`**

Нет автоматического преобразование «из» и «в» тип **`boolean`**!

Можно форсировать преобразование с помощью операции cast – такой же синтаксис, как в C/C++.

```
int i = (int) 1.345;
```

Операторы

Assignment: =, +=, −=, *=, …

Numeric: +, –, *, /, %, ++, −−, …

Relational: ==. !=, <, >, <=, >=, …

Boolean: &&, ||, !

Bitwise: &, |, ^, ~, <<, >>, …

Такие же как в C/C++!

Управляющие структуры

Все, что можно ожидать:

Условие: *if, if else, switch*

Цикл: *while, for, do*

break и *continue*

Исключения (Exceptions)

Терминология:

- *throw an exception*: сигнализировать, что выполнилось некоторое условие (возможно, ошибка).
- *catch an exception*: обработка ошибки или чего-либо.

В Java, обработка исключений обязательна (ошибка компиляции)!

Try/Catch/Finally

```
try {
        // code that can throw an exception
} catch (ExceptionType1 e1) {
        // code to handle the exception
} catch (ExceptionType2 e2) {
        // code to handle the exception
} catch (Exception e) {
        // code to handle other exceptions
} finally {
        // code to run after try or any catch
}
```

Обработка исключений

Исключения используются для обработки ошибок

- Вместо возврата кода ошибки некоторые методы выкидывают исключения (throw an exception).

Может произойти в любом месте стека вызова методов

Заставляет программиста озаботиться обработкой ошибок

Многопоточное программирование

Java мнотопоточная!

- Потоки (threads) легко использовать.

Два способа для создания нового потока:

- Наследование java.lang.Thread
  - Переопределение метода “run().
- Реализовать интерфейс Runnable
  - Добавить метод “run()” в свой класс.

Запуск потока

- new MyThread().start();
- new Thread(runnable).start();

Определение synchronized

Java многопоточна!

- Потоки легко использовать.

Вместо использования mutex используем synchronized:

```
synchronized ( object ) {
    // critical code here
}
```

Synchronized как модификатор

Можно также определить метод как synchronized:

```
synchronized int blah(String x) {
 // blah blah blah
}
```

Классы и объекты

“Все выражения в Java заключаются внутри методов и все методы определяются внутри классов”.

Классы в Java очень похожи на классы в C++ (такая же концепция).

Вместо “standard library” Java предоставляет множество реализаций в классах.

Определение класса

Один публичный класс верхнего уровня на.java файл.

- Обычно получается много .java файлов для одной программы.
- Один (как минимум) имеет статичный открытый метод main(.

Имя класса должно совпадать с именем файла!

- Компилятор/интерпретатор использует имена классов для определения имени файла.

Пример класса

```
public class Point {
	public double x,y;
	public Point(double x, double y) {
		this.x = x; this.y = y;
	}
	public double distanceFromOrigin(){
		return Math.sqrt(x*x+y*y);
	}
}
```

Объекты и new

Вы можете определять переменную для создания объекта:

```
Point p;
```

но это не создает новый объект!

Вам надо использовать *new*:

```
Point p = new Point(3.1,2.4);
```

Есть и другие способы создания объектов…

Использование объектов

Также, как и в C++:

- `object.method()`
- `object.field`

НО, так никогда (нет указателей!)

- `object->method()`
- `object->field`

Строки особенны

Можно инициализировать строки так:

```
String blah = "I am a literal ";
```

Или так (String operator “+”):

```
String foo = "I love " + "RPI";
```

Массивы (Arrays)

Массивы поддерживаются как еще один тип *reference type* (объекты – это другой reference type).

Хотя способ поддержки массивов в Java отличается от поддержки в C++, большинство синтаксиса совместимо.

- Однако, создание массива требует **`new`**

Примеры массивов

```
int x[] = new int[1000];

byte[] buff = new byte[256];

float[][] mvals = new float[10][10];
```

Замечания про массивы

Индекс начинается в 0.

Массивы не могут расти или сжиматься.

- Используется ArrayList вместо этого.

Каждый элемент инициализируется.

Проверка границ массива (overflow!)

- ArrayIndexOutOfBoundsException

Массивы имеют свойство *.length*

Пример обхода массива

```
int[] values;

int total=0;

for (int i=0; i<value.length; i++) {
     total += values[i];
}
```

Array Literals

Можно использовать литералы в массивах также, как и в C/C++:

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

String[] names = {“Joe”, “Sam”};
```

Ссылочные типы (Reference Types)

- Объекты и массивы являются *ссылочными типами* (*reference types)*
- Примитивные типы сохраняются по значению
- Переменные ссылочного типа хранят ссылки (указатели, которые мы не можем использовать).
- И это существенная разница!

Primitive vs. Reference Types

```
int x = 3;
int y = x;
```

There are two copies of the value 3 in memory

```
Point p = new Point(2.3,4.2);
Point t = p;
```

There is only one Point object in memory!

```
Point p = new Point(2.3,4.2);
Point t = new Point(2.3,4.2);
```

Передача аргументов в методы

Примитивные типы: метод принимает копию значения. Изменения не будут видны в методе, откуда был вызов.

Ссылочные типы: метод принимает копию ссылки, метод имеет доступ к тому же объекту!

Пример

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

void increment(int[] a) {
      for (int i=0; i<a.length; i++) {
            a[i]++;
      }
}
```

Сравнение ссылочных типов

Сравнение используя == означает:

- Эти ссылки одинаковые?
- Они ссылаются на один объект?

Иногда вы просто хотите знать, что два объекта являются идентичными копиями.

- Используйте метод **`.equals(`**
  - Требуется реализовать этот метод для своих классов!

Модификаторы контроля доступа

Public – все имеют доступ

Private – никто не имеет доступ снаружи класса

Protected – классы-наследники имеют доступ

Default – доступ внутри package

Модификатор Final

final class – не может быть родительским классом

final method – не может быть *overridden*

final field – не может изменять значение. Поля s*tatic final* являются константами времени компиляции.

final variable – не может изменять значение

Модификатор Static

static method – метод класса, к которому может быть доступ только через имя класса и он не может быть доступен через *this*.

static field – поле класса, к которому может быть доступ только через имя класса . Может быть только **одно** поле, неважно сколько существует экземпляров класса.

Пакеты (Packages)

Можно организовать несколько классов и интерфейсов в *пакет* (*package*).

- Определяет namespace который содержит все классы.

Нужно использовать некоторые java packages в своих программах

- java.lang java.io, java.util

Импорт классов и пакетов

Вместо `#include` используется `import`

Не нужно импортировать все, что угодно. Нужно знать полное имя (не только класса, но и пакета).

- Если написать `import java.io.File` то можно использовать объекты класса `File`.
- Иначе придется писать `java.io.File`.

Пример: Сумма чисел

Sum.java: считывает аргументы командной строки, преобразует в числа, суммирует и выводит результат.

Sum1.java: Та же идея – создается объект Sum1, затем конструктор выполняет всю работу (вместо main).

Пример: Sum.java

```
public class Sum {
    public static void main( String nums[] ) {
        // create an int array of the right size
        int[] ivals = new int[nums.length];
        // convert each string to integer
        for (int i=0;i<nums.length;i++) {
            ivals[i]= Integer.parseInt(nums[i]);
        }
        // sum everything up
        int total=0;
        for (int i=0;i<ivals.length;i++) {
            total += ivals[i];
        }
        System.out.println("Sum is " + total);
    }
}
```

Пример: Sum1.java

```
public class Sum1 {
    public static void main( String args[] ) {
        Sum1 s = new Sum1(args);
    }
    Sum1(String nums[]) {
        int[] ivals = new int[nums.length];
        for (int i=0;i<nums.length;i++) {
            ivals[i]= Integer.parseInt(nums[i]);
        }
        int total=0;
        for (int i=0;i<ivals.length;i++) {
            total += ivals[i];
        }
        System.out.println("Sum is " + total);
    }
}
```

Пример: Point.java

```
public class Point {
    public double x,y;

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

    public  void increment(){
        x++; y++;
    }

    public  String toString() {
        return "(" + x + "," + y + ") ";
    }
}
```

Пример: PointTest.java

```
public class PointTest extends Thread{
    Point point;
    public PointTest(Point point){
          this.point = point;
    }
    public void run(){
          point.increment();
    }
    public static void main(String args[]){
          Point myPoint = new Point(0.0,0.0);
          PointTest myThread1 = new PointTest(myPoint);
          PointTest myThread2 = new PointTest(myPoint);
          myThread1.start(); myThread2.start();
          myThread1.join(); myThread2.join();
          System.out.println(myPoint);
    }
```

Пример: Circle.java (1)

```
public class Circle {
    public Point center;
    public double radius;

    // constructors
    Circle() {
          center = new Point(0,0);
          radius = 0.0;
    }

    Circle(double x, double y, double r) {
          center = new Point(x,y);
          radius = r;
    }
```

Пример: Circle.java (2)

```
    // Compute the area
    public double area() {
          return Math.PI * radius * radius;
    }

    // toString is called when you use System.out.println
    public String toString() {
          return "Center: " + center.toString() +
              "  Radius: " + radius +
              "  Area: " + area();
    }
}
```

Пример: CircleTest.java

```
public class CircleTest {

    public static void main( String[] args ) {


          Circle c1 = new Circle();


          Circle c2 = new Circle(3.1,2.2,1.0);


          System.out.println(c1);

          System.out.println(c2);

    }

}
```

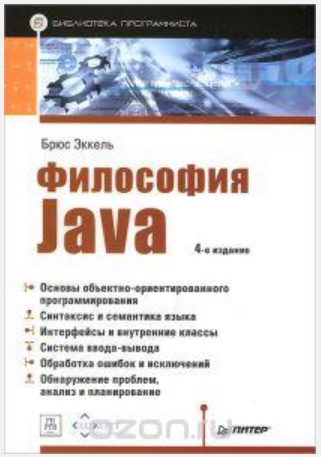
Пример: ExceptionTest.java

```
public class ExceptionTest {

    public static void main(String[] args) {

          try {

              for (int i=1; i<10; i++)

                    System.out.println(args[i]);

          } catch (ArrayIndexOutOfBoundsException iob) {

              System.out.println("Got exception : " + iob );

              iob.printStackTrace();

          } catch (Exception e) {

              System.out.println("Any other exceptions would go here");

          } finally {

              System.out.println("This is always called");

          }

    }

}
```

Полезные материалы

http://docs.oracle.com/javase/8/docs/

«Философия Java», Брюс Эккель («Thinking in Java»)



«Java. Эффективное программирование», Джошуа Блох («Effective Java. Programming Language Guide»)

ТЕХНОСФЕРА

Спасибо за внимание!

Евгений Чернов

e.chernov@corp.mail.ru

Настройка окружения

IDEA: https://www.jetbrains.com/idea/download/

Hadoop: http://hadoop.apache.org/releases.html