

Java

Основы языка

Сергей Рыбалкин

Немного о себе

Java – 5+ лет

C++ – 3 года

Люблю backend и зеленые билды.

Не люблю javascript.

s.rybalkin@corp.mail.ru

<http://stackoverflow.com/users/6375041>

<https://github.com/rybalkinsd>



Немного о вас





Формат курса



Архитектура языка



Базовый синтаксис



Классы



Формат курса



Архитектура языка



Базовый синтаксис



Классы

1. Java intro
2. Java web
3. Java persistence
4. Java client – server communication
5. Итоговый проект

Total: 100

3 рубежных контроля = $16 + 12 + 12$

Задачи семинаров = 30

Сдача итогового проекта = 30

Сертификаты

3: 40+

4: 60+

5: 80+



Формат курса



Архитектура языка



Базовый синтаксис



Классы

- Языку Java 20 лет
- Является языком ООП
- Код транслируется не в машинные команды, а в байт-код, который затем выполняет JVM
- Строгая типизация
- Автоматическое управление памятью

МНОГОПОТОЧНЫЙ
Переносимый
Надежный
Безопасный
Простой
Объектно-ориентированный
Платформенно-независимый

JDK

JRE

JVM

Системные
библиотеки

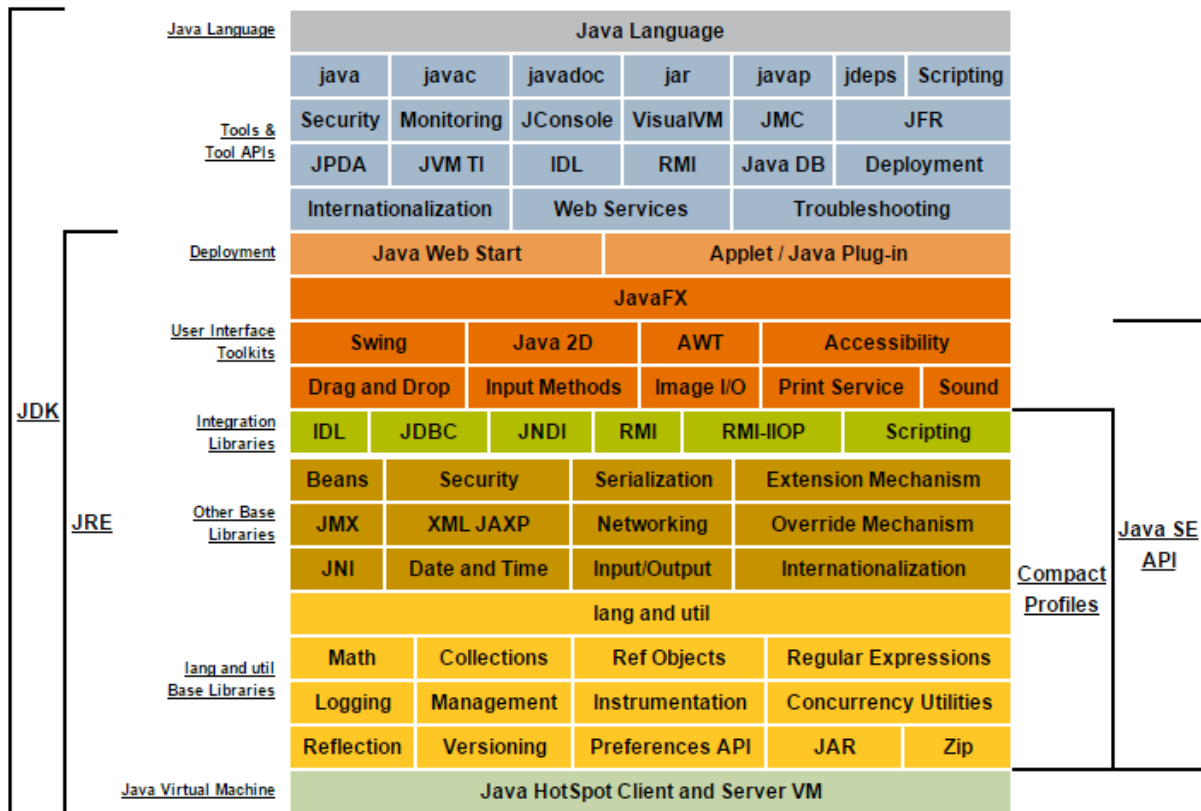


Средства разработки

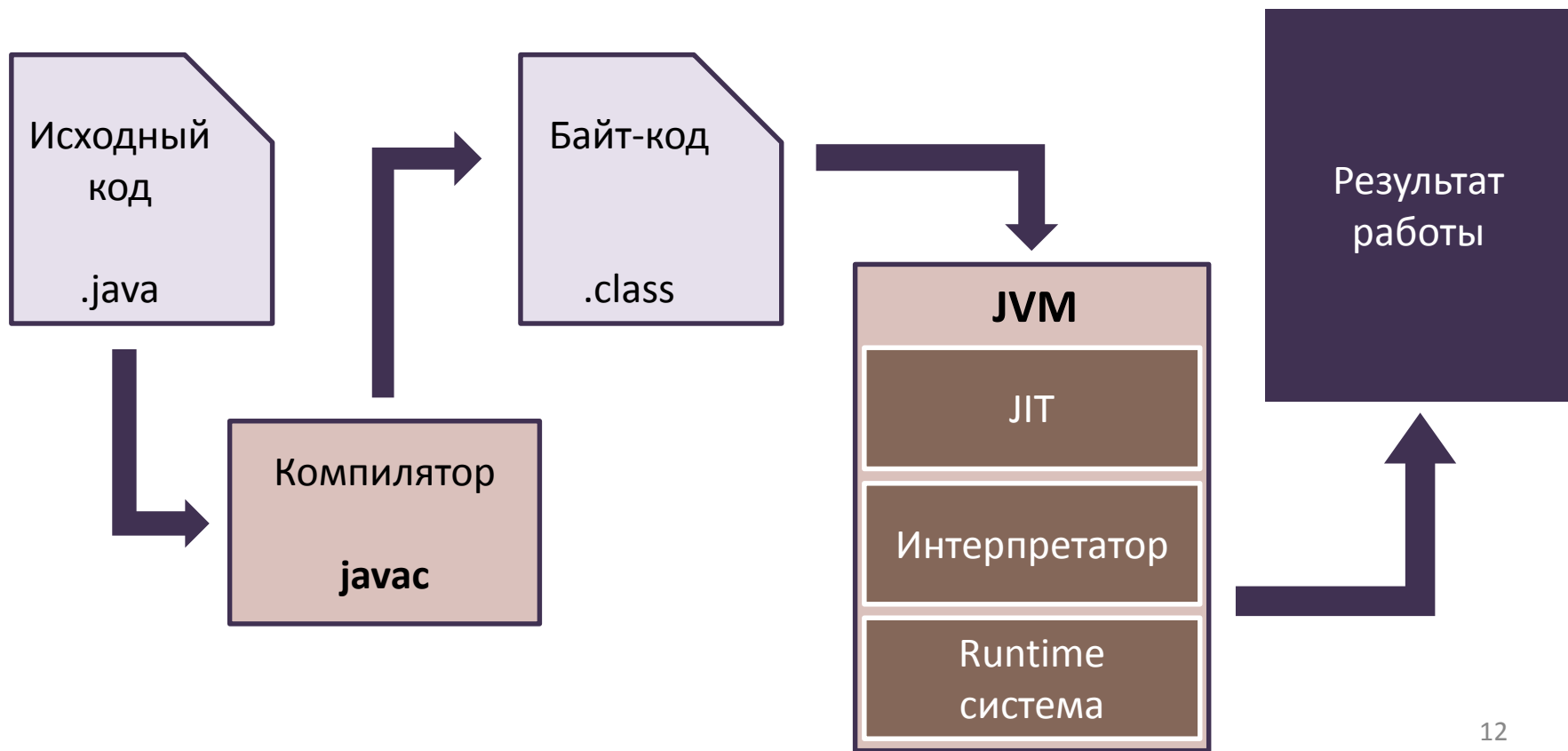
- Компилятор
- Дебаггер
- Профайлер
- ...

Типы распространения

JDK
=
JRE
+
Tools



JRE
=
JVM
+
Lang
+
Libs





Формат курса



Архитектура языка



Базовый синтаксис



Классы

- Primitive types
- Reference types

Type	Size	Range	Type	Size	Range
boolean	undefined	true/false	int	4 bytes	$-2^{31} - 2^{31} - 1$
byte	1 byte	-128 – 127	long	8 bytes	$-2^{63} - 2^{63} - 1$
char	2 bytes	\u0000 – \uffff	float	4 bytes	IEEE 754
short	2 bytes	-32'768 – 32'767	double	8 bytes	IEEE 754

Operator type	Operator
Assignment	=, +=, *= ...^=
Arithmetical	+, -, *, /, %
Relational	<, >, <=, >=, ==, !=
Logical	&&,
Bitwise	&, , ^, >>, <<, >>>
Unary	++, --, +, -, !
Relational2	instanceof

```
1. int value = 0;

2. array[0] = 100;

3. System.out.println("Hello, world!");

4. int result = 1 + 2;

5. if (value1 == value2)
    System.out.println("value1 == value2");
```


Block defines variable scope

```
1.int commonVariable = 0;
2.if (commonVariable > -42) { // ← начало блока
3.    int innerVariable = commonVariable + 1;
4.    System.out.println(String.format("Inner variable is %d",
5.                                     innerVariable));
6.} // ← конец блока
7./*
8.    а здесь innerVariable уже нет
9.*/
```

if – else if – else

```
1.if (18 == yourAge) {  
2.    // у вас всё хорошо  
3.}else if (yourAge > 18  
4.    && yourAge <= 25) {  
5.    // бывало и лучше  
6.}else {  
7.    // "\(\ツ)\/"  
8.}
```

switch - case

```
1. switch (countOfApple) {  
2.     case 1: // у нас есть 1 яблоко  
3.         break;  
4.     case 2: // у нас есть 2 яблока  
5.         break;  
6.     ...  
7.     default:  
8.         // прочие случаи  
9.         break;  
10. }
```

`while (expression) statement`

`do { statement } while (expression)`

`for (initialization; termination; increment)
statement`

```
1. for (int i = 0; i < numberOfObjects; i++) {  
2.     // iterates numberOfObjects times,  
3.     // if numberOfObjects >= 0  
4. }
```

```
1. int[] digits = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}  
2. for (int i : digits ) {  
3.     System.out.println("Digit: " + digit);  
4. }
```

```
1.IntStream.range(0, 10)
2.          .forEach(digit -> System.out.println(digit));
```

```
1.IntStream.range(0, 10)
2.          .forEach(System.out::println);
```

* Java 8, stream, lambda, method reference

Method signature – method name + argument list.

```
1. public int getCountOfApples(List<Integer> boxes,  
2.                               Integer[] numberOfBoxes)  
3.     throws Throwable {  
  
4.     Integer sumOfApples = 0;  
5.     for (Integer i : numberOfBoxes) {  
6.         sumOfApples += boxes.get(i);  
7.     }  
8.     return sumOfApples;  
9. }
```

- Access modifier → public
- Return type → int
- Method name → getCountOfApples
- Parameter list → (...)
- Exception list → Throwable
- Method body → { ... }


```
1. void sayDigit(int digit){  
2.     System.out.println(String.format("The digit is %d", digit));  
3. }  
4.  
5. void sayDigit(float digit){  
6.     System.out.println(String.format("The digit is %f", digit));  
7. }
```

Перегрузка может осуществляться **только** по набору аргументов.



Формат курса



Архитектура языка



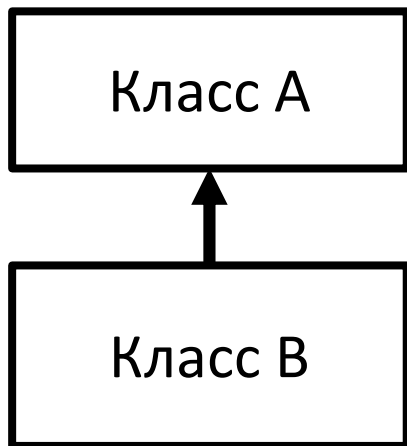
Базовый синтаксис



Классы

Everything is an object.
Nothing outside of a class.

```
1.class Clazz extends SuperClass
2.           implements YourInterface {
3.
4.     private int id;
5.     public Clazz() { ... }
6.}
```



Отношение **Is A**

```
1. class A{
2.     // fields, constructors,
3.     // methods and initialize blocks
4. }
```

```
1. class B extends A{
2.     // fields, constructors,
3.     // methods and initialize blocks
4. }
```

```
1. class A {  
2.     int i;  
3.     public A(int i){  
4.         this.i = i;  
5.     }  
6.     public int getI(){  
7.         return i;  
8.     }  
9. }
```

```
1. class B extends A {  
2.     int i;  
3.     public B(int i){  
4.         super(i);  
5.         this.i = i / 2;  
6.     }  
7.     @Override  
8.     public int getI(){  
9.         return this.i;  
10.    }  
11.    public int getSuperI(){  
12.        return super.i;  
13.    }  
14. }
```

java.lang.Object is a superclass for all classes.

```
1. class Object {  
2.     protected Object clone() throws CloneNotSupportedException  
  
3.     public boolean equals(Object obj)  
4.  
5.     protected void finalize() throws Throwable  
  
6.     public final Class getClass()  
  
7.     public int hashCode() { //some logic }  
  
8.     public String toString()  
9.     ...  
10. }
```

```
1. class Human extends Animal {
2.     public static short AVERAGE_HEIGHT = 170;
3.     public static final long COUNT_OF_POPULATION;
4.     static {
5.         COUNT_OF_POPULATION = 7_000_000_000;
6.     }
7.     private int luckyNumber;
8.
9.     private Human(int myLuckyNumber){
10.         this.luckyNumber = myLuckyNumber1;
11.     }
12.
13.     public Human(int[] luckyNumbersCandidats){
14.         this(selectLuckyNumber(luckyNumbersCandidats));
15.     }
16. }
```

- **static**
- **final**
- **abstract**
- **default**
- **transient** – marker for non-serializable fields
- **volatile** – guarantees atomicity of read/write operations
- **synchronized** – guarantees that block or method will be synchronized
- **native** – native code marker


```
1. abstract class AbstractBird {  
2.     private int age;  
3.     public AbstractBird(int age) {  
4.         this.age = age;  
5.     }  
  
6.     public abstract void sayHi();  
7. }  
  
8. new AbstractBird() {  
9.     @Override  
10.    public void sayHi() {  
11.        // payload  
12.    }  
13.}
```

One class – one superclass.

One class – many interfaces.

```
1.class B extends A
2.    implements Writable, Readable, Mutable {
3.
4.    // payload
5.}
```

```
1. interface Talking {  
2.     void sayHello();  
3.  
4.     default void sayHi(){  
5.         System.out.println("Hi!");  
6.     }  
7. }  
  
8. class Parrot implements Talking {  
9.     public void sayHello(){  
10.         System.out.println("Hello!");  
11.     }  
12. }
```

	Interface	Abstract class
Inheritance	implement many	extend one
Fields	public static only	no limits
Access modifiers	public only	no abstract private methods
Constructor	no constructors	no limits

Спасибо за внимание!

Сергей Рыбалкин

s.rybalkin@corp.mail.ru