

# Введение в анализ данных

Java: основы языка

Сергей Рыбалкин

# План лекции

---



1. Основные моменты
2. Архитектура
3. Типы данных и базовый синтаксис
4. Классы, интерфейсы, наследование

# План лекции

---



1. Основные моменты
2. Архитектура
3. Типы данных и базовый синтаксис
4. Классы, интерфейсы, наследование

# Основные моменты



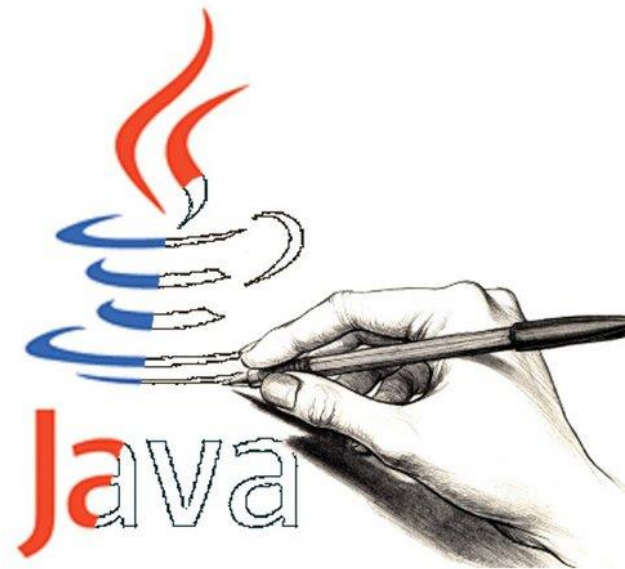
Языку Java 20 лет

Является языком ООП

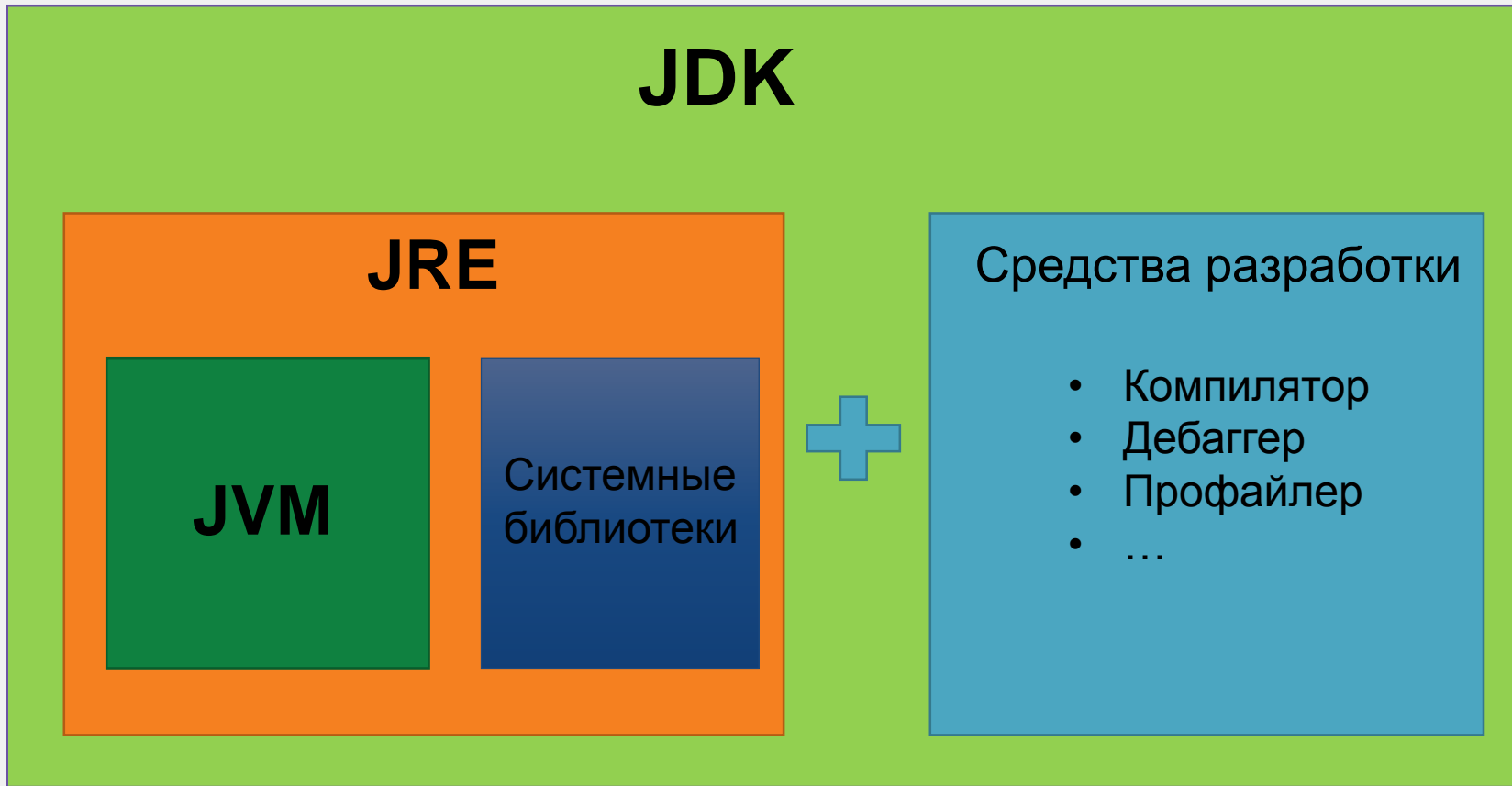
Код транслируется не в машинные команды, а в байт-код, который затем выполняет JVM

Имеет сборщик мусора

МНОГОПОТОЧНЫЙ  
Переносимый  
Надежный  
Безопасный  
Простой  
Объектно-ориентированный  
Платформенно-независимый



# Типы распространения



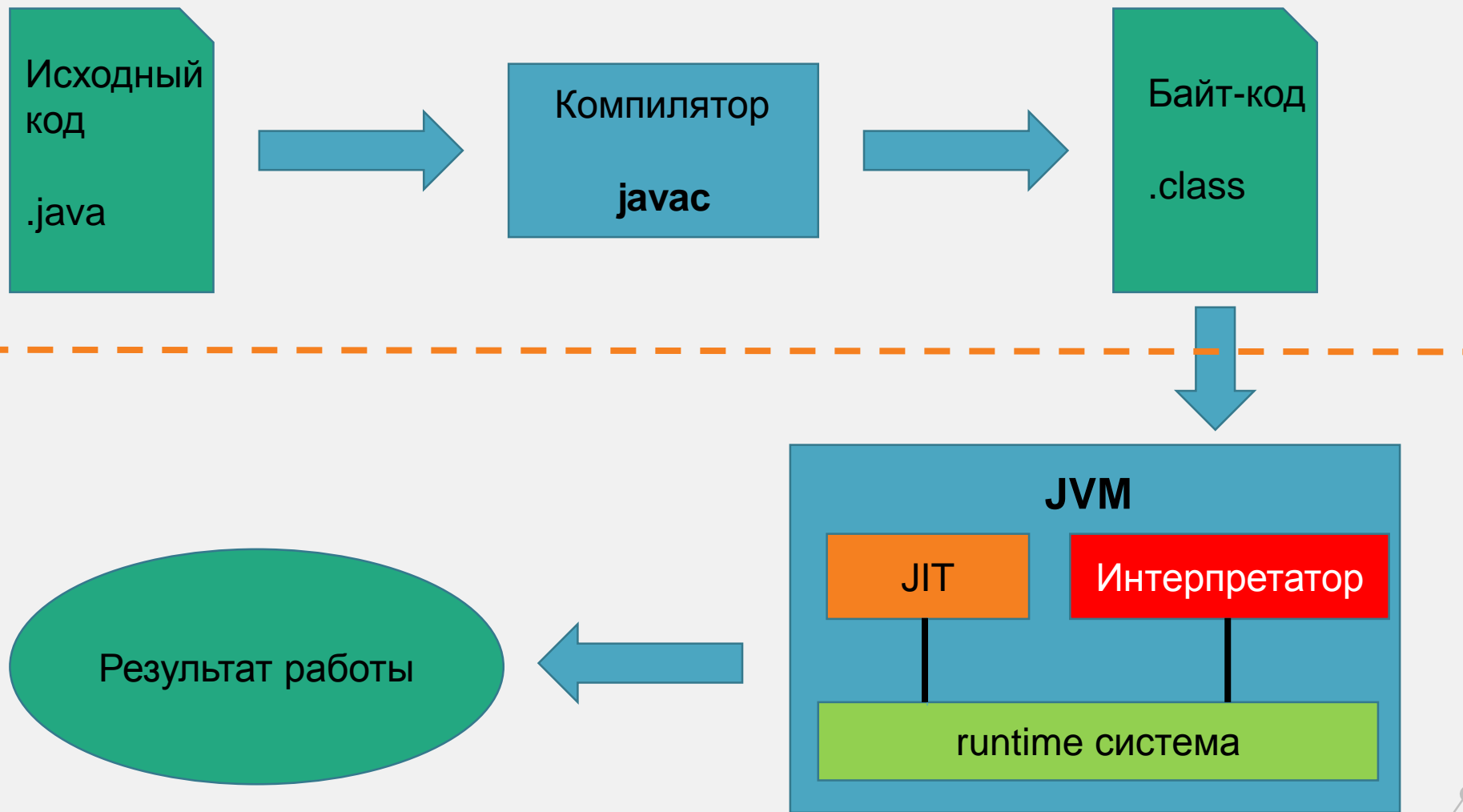
# План лекции

---



1. История
2. Архитектура
3. Типы данных и базовый синтаксис
4. Классы, интерфейсы, наследование

# Архитектура



# План лекции

---



1. История
2. Архитектура
3. Типы данных и базовый синтаксис
4. Классы, интерфейсы, наследование



# Типы данных



## Базовые типы:

Название	Размер	Диапазон значений
boolean	undefined	true/false
char	2 байта	\u0000 - \uffff
byte	1 байт	-128 - 127
short	2 байта	-32,768 – 32,767
int	4 байта	$-2^{31} - 2^{31}-1$
long	8 байт	$-2^{63} - 2^{63}-1$
float	4 байта	IEEE 754
double	8 байт	IEEE 754

## Ссылочные типы:

- Тип класса
- Тип Интерфейса
- Тип массива

# Операторы



Тип операторов	Операторы
Присвоение	=, +=, *= ...^=
Арифметические	+, -, *, /, %
Сравнения	<, >, <=, >=, ==, !=
Логические	&&,
Побитовые	&,  , ^, >>, <<, >>>
Унарные	++, --, +, -, !
Проверка типа	instanceof

# Выражения, блоки



```
1.int value = 0;
2.anArray[0] = 100;
3.System.out.println("Element 1 at index 0: " +
    anArray[0]);
4.int result = 1 + 2;
5.if (value1 == value2)
    System.out.println("value1 == value2");
```

```
1.int countOfApple = 0;
2.{ //начало блока
3.    int countOfFruit = countOfApple + 1;
4.    System.out.println(String.format("I have %d fruit.",
5.                                     countOfFruit));
6.} //конец блока
7./*
8.    а здесь countOfFruit уже нет
9.*/
```

# Условные операторы



## if – then - else

```
1. if(1 == countOfApple){  
2.     // у нас есть 1 яблоко  
3. } else if(countOfApple > 1){  
4.     // у нас больше 1 яблока  
5. } else {  
6.     // у нас меньше 1 яблока  
7. }
```

## switch

```
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: // прочие случаи  
8.         break;  
9. }
```

# Циклы



```
1. while(condition) {  
2. // делаем что-то до тех пор,  
3. // пока true == condition  
4. }
```

```
1. do {  
2. // делаем что-то до тех пор,  
3. // пока true == condition,  
4. // но как минимум 1 раз  
5. } while(condition)
```

# Циклы



```
1. for(int i=0; i < countOfApple; i++) {  
2.     // цикл выполнится countOfApple раз,  
3.     // если countOfApple >= 0  
4. }
```

```
1. for( ; ; ) {  
2.     // а это бесконечный цикл  
3. }
```

```
1. List<Integer> digits = Arrays.asList(1, 2, 3, 4);  
2. for(Integer i : digits ) {  
3.     /* Проход по всем элементам коллекции.  
4.     digits должен реализовывать интерфейс  
5.     java.lang.Iterable<T> ,  
6.     иначе будет ошибка компиляции или быть массивом */  
7. }
```

# break / continue



```
1.int countOfApple = 10;
2.int i=0;
3.for(; i < countOfApple; i++) {
4.    if(5 == i){
5.        break;
6.    }
7.}
8.// 5 == i
```

```
1.List<Integer> digits = new ArrayList<Integer>();
2.for(int i=0; i<10; i++) {
3.    if(5 == i){
4.        continue;
5.    }
6.    digits.add(i);
7.}
8.System.out.println(digits.contains(5)); //false
```

# break / continue



В java нет **goto**, но есть метки, что можно использовать с **break / continue**

```
1. FIRST_LOOP:
2. for(int i=0; i < 10; i++) {
3.     for(int y=0; y < 10; y++){
4.         if(5 == i){
5.             continue FIRST_LOOP;
6.         }
7.         if(i == y) {
8.             System.out.print(i);
9.         }
10.    }
11.}
12. //1234678910
```

```
1. FIRST_LOOP:
2. for(int i=0; i < 10; i++) {
3.     for(int y=0; y < 10; y++){
4.         if(5 == i){
5.             break FIRST_LOOP;
6.         }
7.         if(i == y) {
8.             System.out.print(i);
9.         }
10.    }
11.}
12. //1234
```





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

# Функции



Есть такое понятие, как перегрузка функций. Например,

```
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. По типу возвращаемого значения или по модификатору доступа – нет.

# План лекции

---



1. История
2. Архитектура
3. Типы данных и базовый синтаксис
4. Классы, интерфейсы, наследование

# Классы и объекты



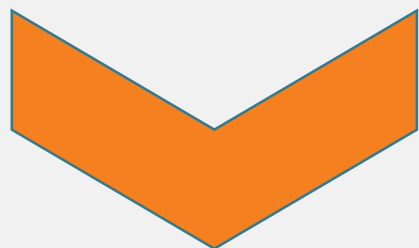
Java – объектно-ориентированный язык программирования.

```
1.class MyClass {  
2.    // поля, конструкторы, и  
3.    // методы и блоки инициализации  
4. }
```

```
1.class MyClass extends MySuperClass  
2.    implements YourInterface {  
3.    // поля, конструкторы, и  
4.    // методы и блоки инициализации  
5. }
```

У класса могут быть конструкторы, но нет деструкторов.  
Наиболее близким понятием к деструктору является реализация метода `finalize()`

# Модификаторы доступа



**public**

При наследовании уровень доступа может изменяться только в сторону большей видимости



**protected**



**default**



**private**

# Классы и объекты



```
1. class Human extends Animal{
2.     private int luckyNumber;
3.     public static short AVERAGE_HEIGHT = 170;
4.     public static final long COUNT_OF_POPULATION;
5.
6.     static{
7.         COUNT_OF_POPULATION = 7000000000;
8.     }
9.
10.    private Human(int myLuckyNumber){
11.        this.luckyNumber = myLuckyNumber;
12.    }
13.
14.    public Human(int[] luckyNumbersCandidates){
15.        this(selectLuckyNumber(luckyNumbersCandidates));
16.    }
17.
18.    protected final selectLuckyNumber(int[] candidates){
19.        int result = 0;
20.        //some logic
21.        return result;
22.    }
23.}
```

# Прочие модификаторы

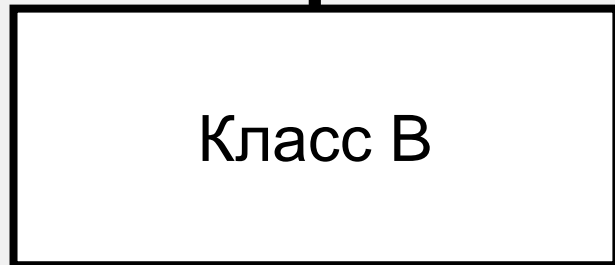


- **static**
- **final**
- **abstract**
- **default**
- **strictfp** — гарантирует единообразия выполнения операций над числами с плавающей точкой
- **transient** — маркирует поля, которые не будут сериализоваться
- **volatile** — гарантирует атомарность операций чтения/записи
- **synchronized** — обеспечивает синхронизацию выполнения блоков кода
- **native** — метод реализован в нативном коде

# Наследование



```
1.class A{  
2.    // поля, конструкторы, и  
3.    // методы и блоки инициализации  
4. }
```



```
1.class B extends A{  
2.    // поля, конструкторы, и  
3.    // методы и блоки инициализации  
4. }
```

```
1.B b = new B();  
2.System.out.println(b instanceof A);  
3. //true
```



# Наследование



Класс `java.lang.Object` является суперклассом для всех классов (даже для массивов)

```
1. class Object{
2.
3.     protected Object clone() throws CloneNotSupportedException {
4.         //some logic}
5.
6.     public boolean equals(Object obj) { //some logic }
7.
8.     protected void finalize() throws Throwable { //some logic }
9.
10.    public final Class getClass() { //some logic };
11.
12.    public int hashCode() { //some logic }
13.
14.    //... Прочие методы
15.
16.    public String toString() { //some logic }
17.}
```

# Наследование



```
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.}
10.class B extends A{
11.    int i;
12.    public B(int i){
13.        super(i);
14.        this.i = i / 2;
15.    }
16.    @Override
17.    public int getI(){
18.        return this.i;
19.    }
20.    public int getSuperI(){
21.        return super.i;
22.    }
23.}
```

# Наследование



```
1.class A{
2.    public printName(){
3.        System.out.println("My name is A");
4.    }

5.    static public sayHellow (){
6.        System.out.println("Hello");
7.    }
8.}
9.class B extends A{
10.    @Override
11.    public printName(){
12.        System.out.println("My name is B");
13.    }
14.
15.    static public sayHellow (){
16.        System.out.println("Hello");
17.    }
18.}
```

# Наследование



---

Также поля класса могут быть отмечены, как `final` и `static`



В Java множественное наследование реализовано через механизм интерфейсов. Т.е. дочерний класс может наследовать только одному родительскому классу, но многим интерфейсам сразу.

```
1.class B
2.    extends A
3.    implements Writable, Readable, Mutable{
4.        // поля, конструкторы, и
5.        // методы и блоки инициализации
6.    }
```



## Default методы

```
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. }
```

# Абстрактные классы



```
1. abstract class ParrotAbstact {  
2.     int age;  
3.     public ParrotAbstact(int age){  
4.         this.age = age;  
5.     }  
6.     public void sayHello(){  
7.         System.out.println("Hello!");  
8.     }  
9.     abstract void sayHi();  
10. }
```

# Абстрактные классы vs Интерфейсы



Где сравниваем?	Интерфейс	Абстрактный класс
Наследование	Класс может реализовать множество интерфейсов	Класс может наследовать только один базовый класс
Поля данных	Только public static константы	Как в обычном классе
Модификаторы доступа методов	Только public	Есть ограничение только на abstract методы (не могут быть private)
Конструктор	Не может иметь конструктор	Нет ограничений

Если коротко, интерфейс – это все-таки больше «контракт», а абстрактный класс – реализация, хоть и не полная.



# Переопределение vs Перегрузка



---

Важно помнить:

- При **переопределении** выбор функции для вызова осуществляется во **время выполнения**.
- При **перегрузке** при **компиляции**.

# Создание объектов



Объект класса можно создать при помощи ключевого слова **new**:

```
1. Integer a = new Integer(1);  
2. String a = new String("1");
```

## Порядок создания объекта:

1. Вызывается блок статической инициализации базового класса (если он есть и класс не был загружен ранее)
2. Вызывается блок статической инициализации создаваемого класса (если он есть и класс не был загружен ранее)
3. Вызывается конструктор класса
4. Вызывается конструктор базового класса
5. Происходит инициализация переменных в порядке их определения
6. Вызывается остальной код конструктора.

# Перечисления



```
1. public enum DayOfWeek{
2.     SUNDAY, MONDAY, TUESDAY, WEDNESDAY,
3.     THURSDAY, FRIDAY, SATURDAY,
4. }
```

## Примеры использования:

```
1. public static void main(String... args){
2.     for(DayOfWeek day : DayOfWeek.values()){
3.         System.out.println(day.name());
4.     }
5. }
```

```
1. switch(day){
2.     case SUNDAY:
3.         makeClean();
4.         break;
5.     case SATURDAY:
6.         sleepIn();
7.         break;
8. }
```

# Перечисления



А можно и так:

```
1. public enum DayOfWeek{
2.     MONDAY (1),
3.     TUESDAY (2),
4.     WEDNESDAY (3),
5.     THURSDAY (4),
6.     FRIDAY (5),
7.     SATURDAY (6),
8.     SUNDAY (7),
9.
10.    private int number;

11.    public DayOfWeek(int numberOfDay){
12.        this.number = numberOfDay;
13.    }
14.
15.    public getNumber(){
16.        return this.number;
17.    }
18.}
```



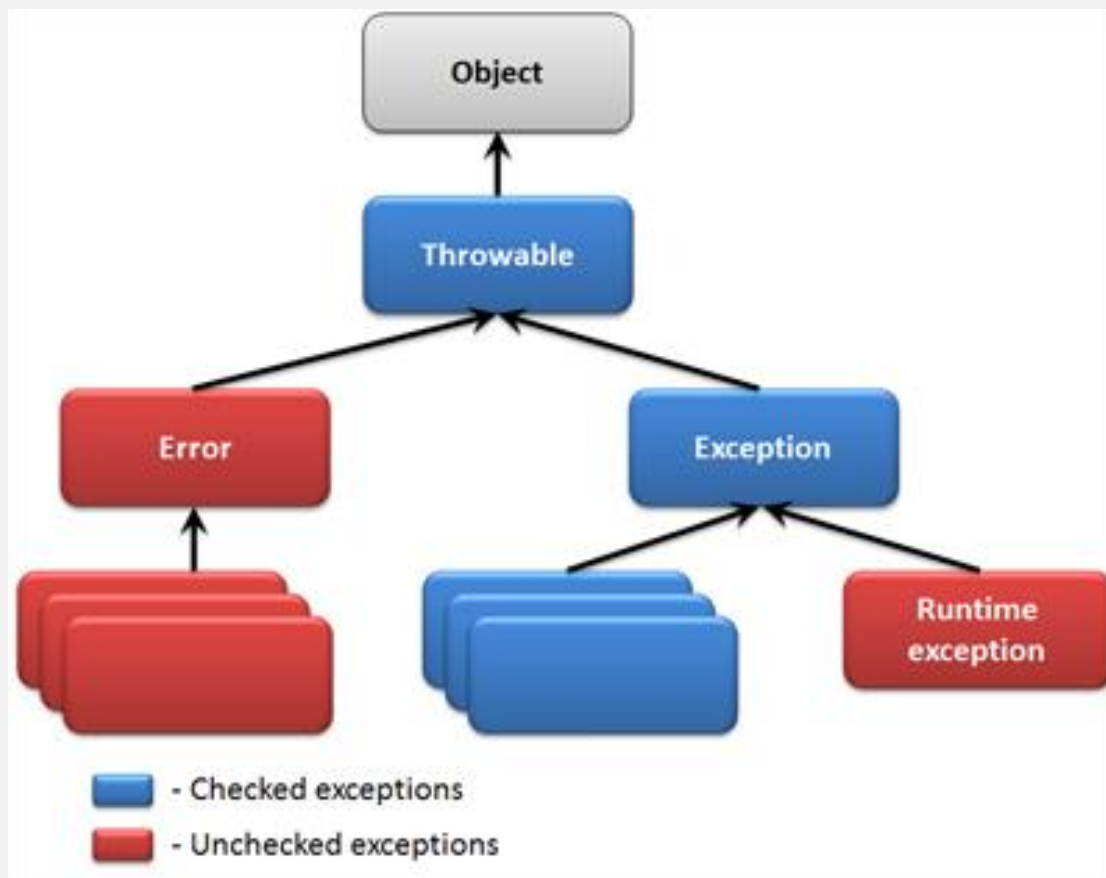
Java пакеты – механизм, позволяющий организовывать Java – классы в пространстве имен

```
1. //Пример указания текущего пакета
2.package ru.mail.sfera.ida.lab1
```

```
1. //Пример импорта
2.import java.util.List; //Теперь в коде можем писать просто List
3.import java.util.*; //то же самое, что и выше, но теперь
4.                      //доступны все классы из java.util
5.import static java.lang.Math.PI; //Теперь можем статическую
6.                               // константу PI в коде
7.import static java.lang.Math.*;  // так тоже можно
```



## Иерархия основных типов исключений



# try/catch/finally



## try – catch - finally

```
1. try{
2.     //Do something
3. } catch(Exception e){
4.     e.printStackTrace();
5.     System.err.println("Error:" + e.getMessage());
6. }
7. finally{
8.     // этот код будет выполнен в любом случае.
9.     //Ну, почти в любом :)
10. }
```

## try – with - resources

```
1. static String readFirstLineFromFile(String path)
2.     throws IOException {
3.     try (BufferedReader br =
4.         new BufferedReader(new FileReader(path))) {
5.         return br.readLine();
6.     }
7. }
```

# Компиляция/запуск



В случае одного файла

1. `javac Имя_файла.java`
2. `java -classpath . Имя_файла`

Имя файла совпадает с именем класса

В случае пакетов:

1. `javac -sourcepath ./src -d bin/src/.../Имя_класса.java`
2. `java -classpath ./bin Полное_имя_класса`

Хорошая статья по теме:

<http://habrahabr.ru/post/125210/>





**Спасибо за  
внимание!**

**Сергей Рыбалкин**

[s.rybalkin@corp.mail.ru](mailto:s.rybalkin@corp.mail.ru)