Faculty of Software Engineering and Computer Systems

Programming

Lecture #5.
Exceptions. Functional syntax

Instructor of faculty
Pismak Alexey Evgenievich
Kronverksky Pr. 49, 1331 room

pismak@itmo.ru

Saint-Petersburg

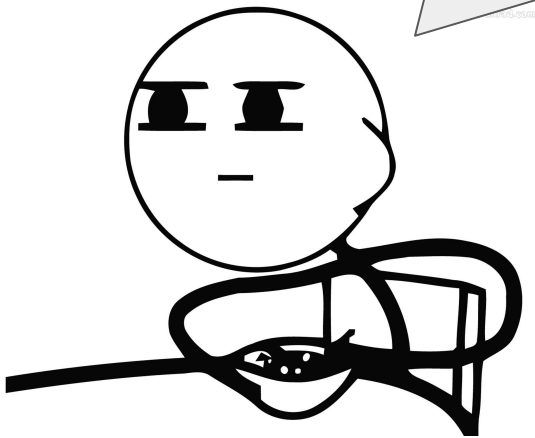
Исключительные ситуации (ошибки)

- Как было раньше?
  - прерывание
  - флаг ошибки
  - код возврата
  - специальное значение

Исключения (exceptions)

```
public static void main(String[] s) {

    int x = …. // читаем из файла
    int y = 5 / x;

}
```

Если вдруг **x** окажется равным **0**, то что нам предложит объектно-ориентированная Java ?

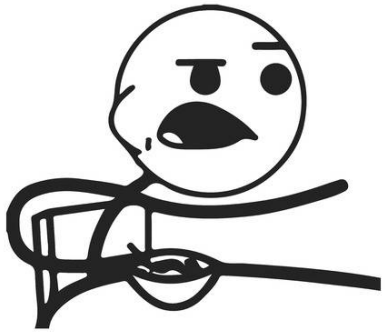


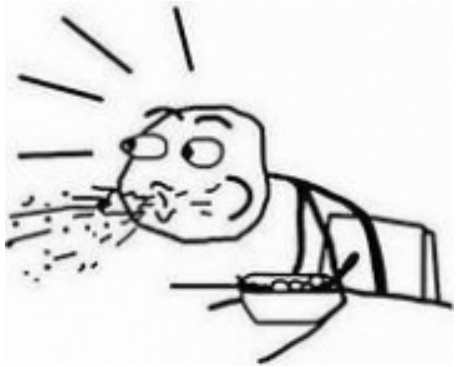
Исключения (exceptions)

```
public static void main(String[] s) {

    try {

        int x = …. // читаем из файла
        int y = 5 / x;

    } catch (ArithmeticException e) {
        // обработка
    }

}
```

Java предлагает порождать объекты определенных типов данных, которые будут содержать состояние и поведение возникшей ошибки.

А еще в языке имеется ряд ключевых слов для того, чтобы оперировать такими объектами, обрабатывать их генерацию и т.д.





Исключения (exceptions)

```
public static void main(String[] s) {

    try {
        int x = …. // читаем из файла
        int y = 5 / x;
    } catch (ArithmeticException e) {
        // обработка
    } finally {
        // обработка
    }

}
```

Ключевое слово, предваряющее блок кода

Блок кода, который необходимо выполнить

Если в блоке возникнет ошибка, то управление будет передано блоку *catch*

Блок finally выполняется в любом случае: была ошибка или нет. Он не является обязательным

Исключения (exceptions)

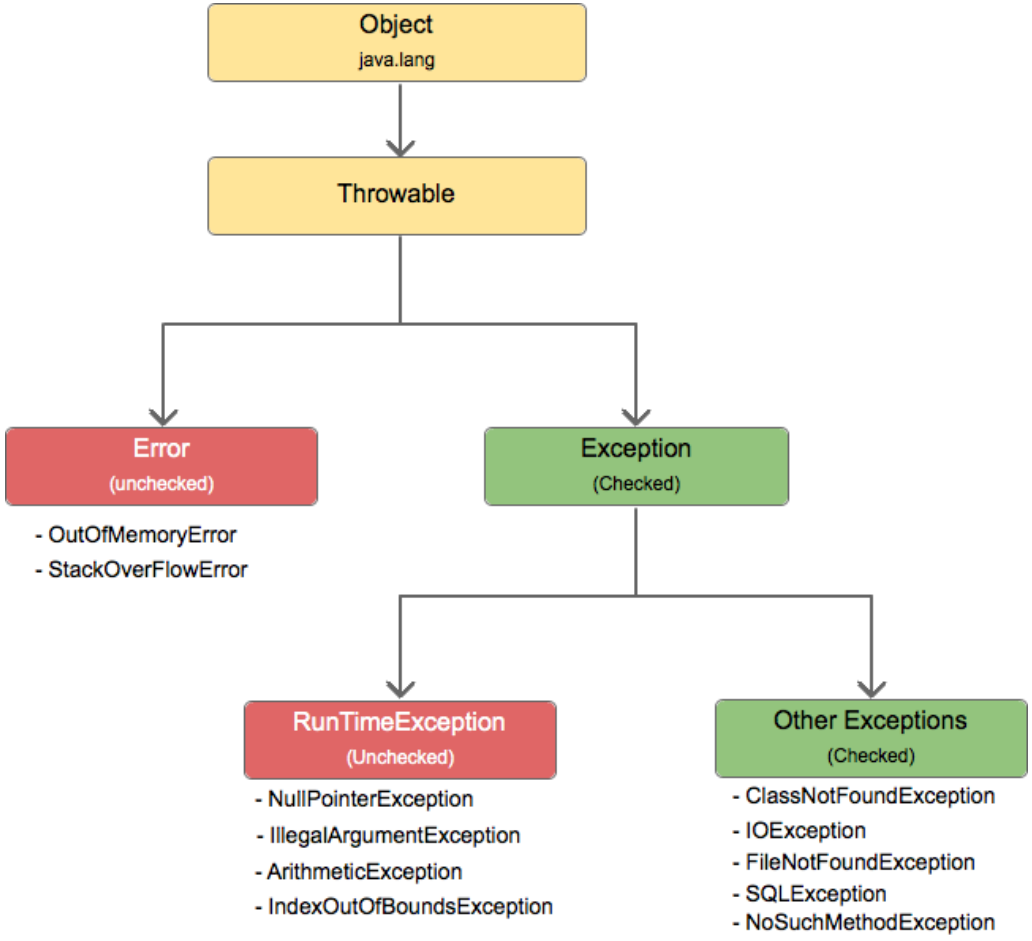
```
public static void main(String[] s) {

    try {

        int x = …. // читаем из файла
        int y = 5 / x;

    } catch (ArithmeticException e) {
        // обработка
    } finally {
        // обработка
    }

}
```

Каждая такая исключительная ситуация в программе генерирует объект своего типа

Если посмотреть внимательно на то, как устроен блок catch, то можно заметить, что он похож на объявление метода с именем *catch* и одним аргументом с именем *e*

Как Java понимает какие типы данных являются исключениями, а какие нет?

Исключения и их классификация



Собственные исключения

```
public class TheBestException extends Exception {

    // можно переопределить ранее объявленные методы
    // можно добавить свои поля, методы, конструкторы

}
```

Определяем класс, который будет наследником класса Throwable

Затем используем его в нужном месте оператором throw

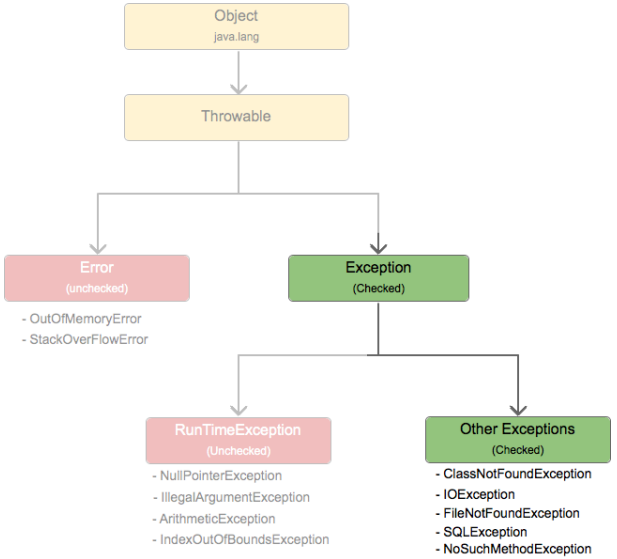
```
public void someMethod() {

    TheBestException tbe =
                new TheBestException();
     throw tbe;

}
```

Исключения (exceptions)

```
public static void main(String[] s) {

    try {

        someMethod();

    } catch (TheBestException e) {
        // обработка
    } finally {
        // обработка
    }

}
```

Классификация в действии (checked)

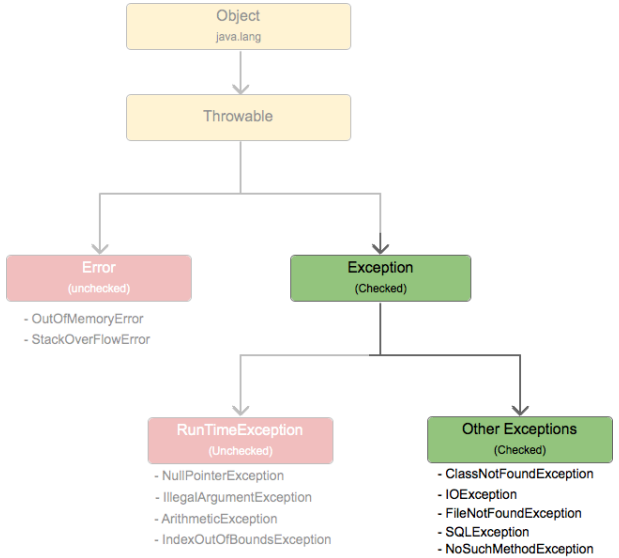
```
public class ExceptionExample {

    public void someMethod() throws Exception {

        Exception ex = new Exception();
        throw ex;

    }
}
public static void main(String[] s) {
    ExeptionExample ee = new ExceptionExample();
    try {
        someMethod();
    } catch (Exception e) {
        // обработка
    }

}
```



Коварианты инструкции throws

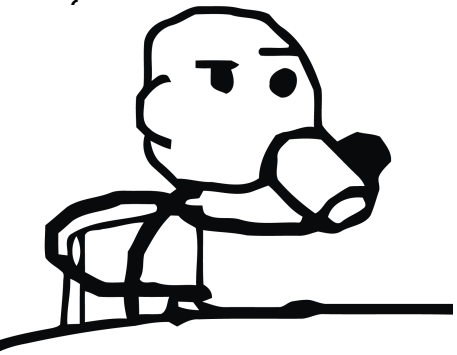
```
public class MyException extends Exception {   }
```

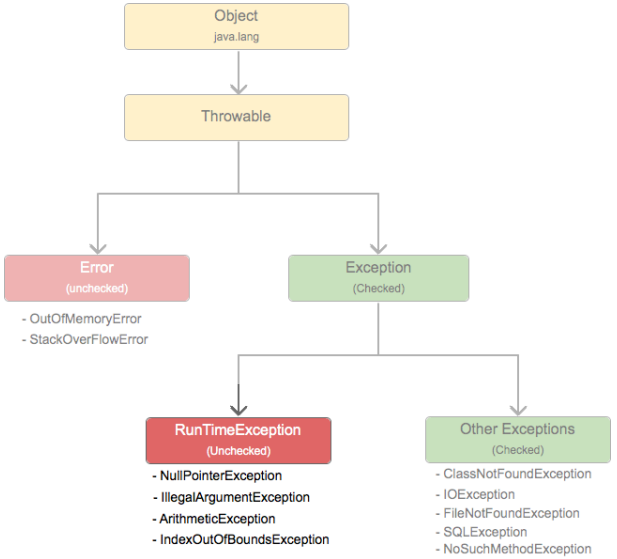
```
public class ExceptionGen extends ExceptionExample {

    public void someMethod() throws MyException {
        MyException ex = new MyException();
        throw ex;
    }
}
public static void main(String[] s) {
    ExeptionExample ee = new ExceptionGen();
    try {
        someMethod();
    } catch (Exception e) {
        // обработка
    }
}
```



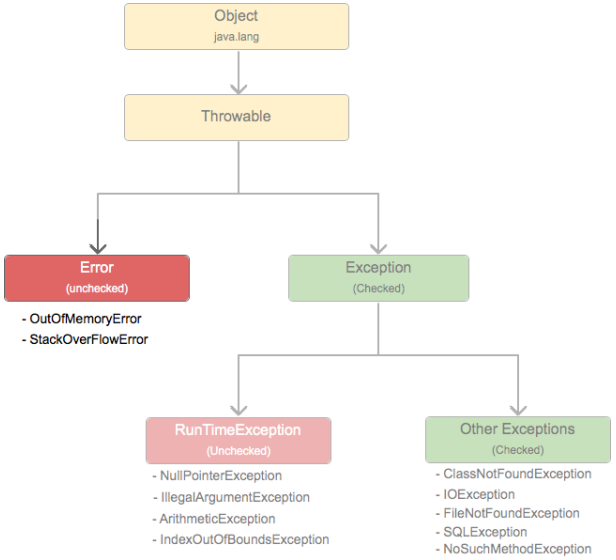
Классификация в действии (unchecked)

```
public class MyException extends RuntimeException {   }


public static void main(String[] s) {
    ExeptionExample ee = new ExceptionGen();
    try {
        someMethod();
    } catch (Exception e) {
        // обработка
    }
}
```





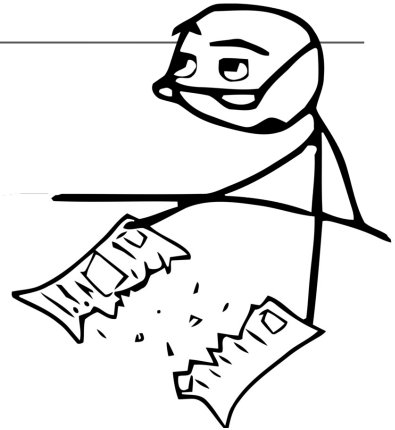
Классификация в действии (error)



Multiple catch

```
public class ExceptionExample {

    public void test() {
        try {
            new ThrowsExample().method();

        } catch (Exception1 | Exception2 ex) {
            // do nothing
        }
    }
}
```

```
public class ThrowsExample {

    public void method() throws Exception1, Exception2 {
        method();
    }
}
```



Catched

```
public class Example{

    public void test() {
        try {
            method();

        } catch (Throwable e) {
            // WAT ?
        }
    }
}
```

```
String getMessage()

Throwable getCause()

printStackTrace();

etc
```

try-with-resources

```
public class Example{

    public void test() {
        try {
            var x = new FileInputStream();

        } catch (Throwable e) {

        } finally {
                x.close();
        }
    }
}
```

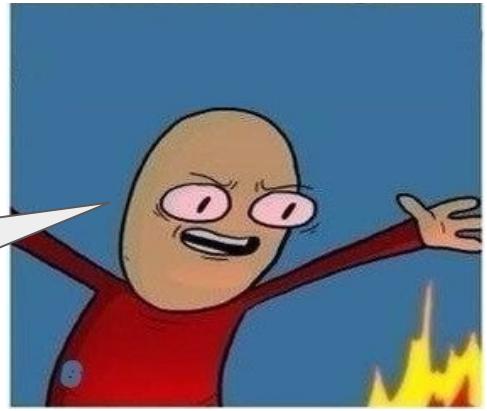
```
public class Example{

    public void test() {
        try (var x = new FileInputStream()) {

            var x = new FileInputStream();

        }
    }
}
```

?

Функциональное программирование в Java

Это было 18 марта 2014 г.

Java приобрела элементы функционального синтаксиса



Они же еще почти школьники!

Небольшая предыстория...

```
public class Person {

    private int age;
    private String name;

    public int getAge() {
        return age;
    }

    public String getName() {
        return name;
    }

    // остальные методы и конструкторы

}
```

```
public interface Checker {

    boolean check(Person person);

}
```

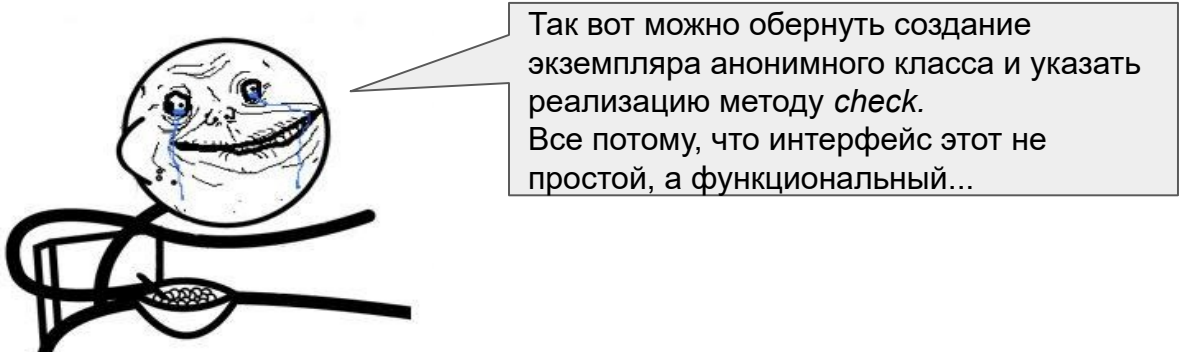
```
void printPerson(Person p, Checker c) {
    if(c.check(p)) { // if person is good :)
        // print
    }
}
```

Небольшая предыстория...

```
public void test(Person[] persons) {

    Checker checker = new Checker() {
            @Override
            public boolean check(Person person) {
                return   person.getAge() > 17   &&   person.getName().startsWith("Я");
            }
    };

    for(Person p : persons) {
        printPerson(p, checker);
    }
}
```

Лямбда-выражения

```
public void test(Person[] persons) {

    for(Person p : persons) {
        printPerson(p, person -> person.getAge() > 17 && person.getName().startsWith("Я"));
    }
}
```



Лямбда-выражения

```
public void test(Person[] persons) {

    for(Person p : persons) {
        printPerson(p, person -> person.getAge() > 17 && person.getName().startsWith("Я"));
    }
}
```

Лямбда-выражения

```
Checker checker = new Checker() {
        @Override
        public boolean check(Person person) {
            return   person.getAge() > 17   &&   person.getName().startsWith("Я");
        }
    };
```

```
(person) -> person.getAge() > 17 && person.getName().startsWith("Я")
```

Функциональные интерфейсы

```
@FunctionalInterface
public interface Checker {

    boolean check(Person person);

    boolean anotherCheck(Person person);

}
```

Functions

```
Function<Person, Boolean> function = person ->
                                        person.getAge() > 17 && person.getName().startsWith("Я");


Boolean result = function.apply( new Person() );
```

Ссылки на методы

```
public static boolean checkMethod(Person p) {
    return p.getAge() > 17 && p.getName().startsWith("Я");
}
```

```
public interface Checker {

    boolean check(Person person);

}
```

```
public void test(Person[] persons) {

    for(Person p : persons) {
        printPerson(p, __checker__);
    }
}
```

Ссылки на методы

```
public static boolean checkMethod(Person p) {
    return p.getAge() > 17 && p.getName().startsWith("Я");
}
```

```
public void test(Person[] persons) {

    for(Person p : persons) {
        printPerson(p, Main::checkMethod);
    }
}
```

```
public interface Checker {

    boolean check(Person person);

}
```

Ссылки на методы

```
public void test(Person[] persons) {

    for(Person p : persons) {
        printPerson(p, object::checkMethod);
    }
}
```

Ссылки на ~~методы~~

```
public class Person {

    private int age;
    private String name;

    public int getAge() {
        return age;
    }

    public String getName() {
        return name;
    }

    Person(int age, String name) {...}

}
```

```
interface PersonFactory {

        Person create(int age, String name);
}
```

Ссылки на конструкторы

```
public class Person {

    private int age;
    private String name;

    public int getAge() {
        return age;
    }

    public String getName() {
        return name;
    }

    Person(int age, String name) {...}

}
```

```
interface PersonFactory {

        Person create(int age, String name);
}
```

```
PersonFactory pf = Person::new ;

pf.create(5, “вот те на”);
```

?

Wrapper classes (Обёртки)

- Примитивные типы
  - byte, short, int, long, float, double, char, boolean
- Обёртки
  - Byte, Short, Integer, Long, Float, Double, Character, Boolean
    - Integer.MIN_VALUE / Integer.MAX_VALUE
  - Number, BigInteger, BigDecimal
  - Void

Boxing / Unboxing

- `Integer i1 = new Integer(5);`
- `Integer i2 = Integer.valueOf(10);`
- `int i3 = i1.intValue();`
- `int i4 = i2.intValue();`

Autoboxing / Autounboxing

- `Integer i1 = 5;`
- `Integer i2 = 10;`
- `int i3 = i1;`
- `int i4 = i2;`

- `i1 += i2;`

Autoboxing / Autounboxing

- `Integer i1 = 5;`
- `Integer i2 = 10;` `// Integer i2 = Integer.valueOf(10);`
- `int i3 = i1;`
- `int i4 = i2;` `// int i4 = i2.intValue()`

- `i1 += i2;`

```
// i1 = Integer.valueOf ( i1.intValue() + i2.intValue() );
```

Autoboxing / Autounboxing

- `Integer i1 = 5;`
- `Integer i2 = 10;` `// Integer i2 = Integer.valueOf(10);`
- `int i3 = i1;`
- `int i4 = i2;` `// int i4 = i2.intValue()`

```
for (var x = 0; x < 1_000_000; x++) {
```

- `  i1 += i2;` `// i3 += i4;`

```
// i1 = Integer.valueOf ( i1.intValue() + i2.intValue() );
}
```

Number

- java.lang.Number
  - Byte
  - Short
  - Integer
  - Long
  - Float
  - Double
  - java.math.BigInteger
  - java.math.BigDecimal

BigInteger & BigDecimal

- BigInteger - целое число произвольной длины
  - var bigInt = new BigInteger("99999999999999999999");
- BigDecimal - точное вещественное число
  - var bigDec = new BigDecimal("99.99");

  15.31 + 15.32 // 30.630000000000003
- Методы
  - .add(), .subtract(), .multiply(), .divide(), .remainder()

Array & ArrayList

- Массив
  - любые типы данных
    - примитивные, массивы, классы, интерфейсы, ...
  - фиксированный размер
    - задается при создании и больше не меняется
- java.util.ArrayList
  - только ссылочные типы данных (НЕ ПРИМИТИВНЫЕ)
  - динамический размер (может увеличиваться)

Array & ArrayList

```java
String[] array =
         new String[3];

array[0] = "A";

System.out.println(array[0]);

array[2] = "D";

System.out.println(array.length);
System.out.println(array);
```

```java
var list =
    new ArrayList<String>();

list.add("A");

System.out.println(list.get(0));

list.set(2, "D");

list.add("B"); list.add(0,"C");
list.add("D"); list.add(1,"E");
list.remove(0);
System.out.println(list.size());
System.out.println(list);
```

Array & ArrayList

```java
int[] array = {1,2,3};

array[3] = 4;
```

```java
var list = List.of(1,2,3);

list.add(4);
```

Array & ArrayList

```java
int[] array = {1,2,3};

array[3] = 4;
```

```java
var list = List.of(1,2,3);

list.add(4);

var list2 = new ArrayList(
                List.of(1,2,3));

list2.add(4);
```