

Специализация: ООП и исключения














Специализация: ООП и исключения



На прошлом уроке

-  Классы;
-  Объекты;
-  Статика;
-  Стек и куча;
-  Сборщик мусора;
-  Конструкторы;
-  Инкапсуляция;
-  Наследование;
-  Полиморфизм.





Что будет на уроке сегодня

- 📌 Перечисления;
- 📌 Вложенные и внутренние (Nested) классы;
- 📌 Статические вложенные классы;
- 📌 Механизм исключительных ситуаций;
- 📌 Введение в многопоточность.





Перечисления



Перечисление

Перечисление - это упоминание объектов, объединённых по какому-либо признаку





Сезоны

```
enum Season { WINTER, SPRING, SUMMER, AUTUMN }
```

[]

Java



Один сезон

```
Season current = Season.AUTUMN;  
System.out.println(current);
```

✓ 0.3s

Java

AUTUMN



Перечислить

```
Season[] seasons = Season.values();  
for (Season s : seasons) {  
    System.out.printf("%s ", s);  
}
```

✓ 0.3s

Java

WINTER SPRING SUMMER AUTUMN



Порядковый номер

```
System.out.println(current.ordinal())
```

✓ 0.1s

Java

3

```
System.out.println(Seasons.ordinal())
```

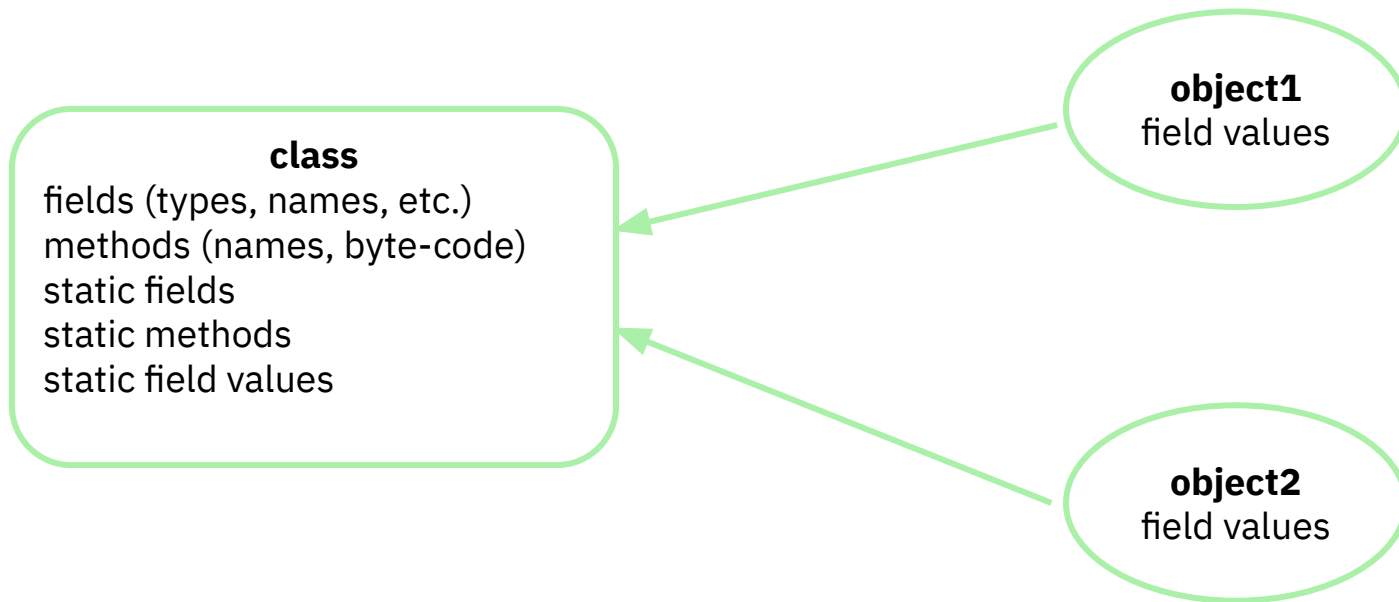
⊗ 0.2s

Java

```
| System.out.println(Seasons.ordinal())  
cannot find symbol  
symbol:   variable Seasons
```



Статические поля





Перечисление цвет

```
enum Color {  
    RED("#FF0000"), GREEN("#00FF00"), BLUE("#0000FF");  
    String code;  
    Color (String code) {  
        this.code = code;  
    }  
    String getCode() {  
        return code;  
    }  
}
```

✓ 0.5s

Java



Перечисление с полем

```
for (Color c : Color.values()) {  
    System.out.printf("%s(%s) ", c, c.getCode());  
}
```

✓ 0.1s

Java

RED(#FF0000) GREEN(#00FF00) BLUE(#0000FF)

Ответьте на вопросы сообщением в чат

Вопросы:

1. Перечисления нужны, чтобы вести учёт:
 - a. созданных в программе объектов;
 - b. классов в программе;
 - c. схожих по смыслу явлений в программе.
2. Перечисление - это:
 - a. массив;
 - b. класс;
 - c. объект.
3. Каждый объект в перечислении - это:
 - a. статическое поле;
 - b. статический метод;
 - c. статический объект.

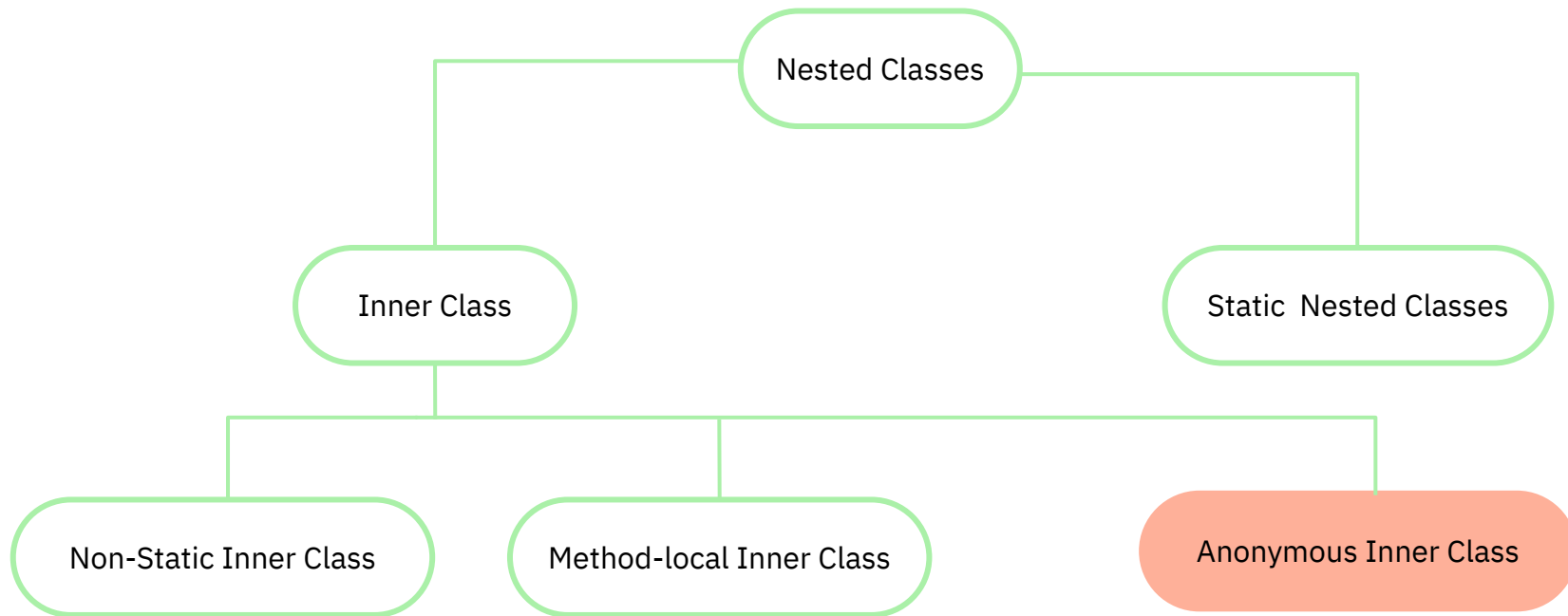




Вложенные и внутренние (Nested) классы



Вложенные классы





Класс апельсин

```
1  public class Orange {  
2      public void squeezeJuice() {  
3          System.out.println("Squeeze juice ...");  
4      }  
5      class Juice {  
6          public void flow() {  
7              System.out.println("Juice dripped ...");  
8          }  
9      }  
10 }  
11
```

✓ 1.4s

Java



Использование апельсина

```
1  Orange orange = new Orange();  
2  Orange.Juice juice = orange.new Juice();  
3  orange.squeezeJuice();  
4  juice.flow();
```

✓ 0.6s

Java

Squeeze juice ...

Juice dripped ...



Технологичный апельсин

```
1 public class Orange {
2     private Juice juice;
3     public Orange() {
4         this.juice = new Juice();
5     }
6     public void squeezeJuice(){
7         System.out.println("Squeeze juice ...");
8         juice.flow();
9     }
10    private class Juice {
11        public void flow() {
12            System.out.println("Juice dripped ...");
13        }
14    }
15 }
```



Апельсиновый сок

```
1  Orange orange = new Orange();  
2  orange.squeezeJuice();
```

✓ 0.3s

Java

Squeeze juice ...

Juice dripped ...



Схема работы внутреннего класса





Апельсин и технологичный апельсин

```
1 public class Orange {
2     public void squeezeJuice() {
3         System.out.println("Squeeze juice ...");
4     }
5     class Juice {
6         public void flow() {
7             System.out.println("Juice dripped ...");
8         }
9     }
10 }
11
```

✓ 1.4s

Java

```
1 public class Orange {
2     private Juice juice;
3     public Orange() {
4         this.juice = new Juice();
5     }
6     public void squeezeJuice(){
7         System.out.println("Squeeze juice ...");
8         juice.flow();
9     }
10    private class Juice {
11        public void flow() {
12            System.out.println("Juice dripped ...");
13        }
14    }
15 }
```

Java





Особенности внутренних классов



Внутренний объект не существует без внешнего;






Особенности внутренних классов

-  Внутренний объект не существует без внешнего;
-  Внутренний имеет доступ ко всему внешнему;







Особенности внутренних классов

-  Внутренний объект не существует без внешнего;
-  Внутренний имеет доступ ко всему внешнему;
-  Внешний не имеет доступа ко внутреннему без создания объекта;








Особенности внутренних классов

-  Внутренний объект не существует без внешнего;
-  Внутренний имеет доступ ко всему внешнему;
-  Внешний не имеет доступа ко внутреннему без создания объекта;
-  У внутренних классов есть модификаторы доступа;









Особенности внутренних классов

-  Внутренний объект не существует без внешнего;
-  Внутренний имеет доступ ко всему внешнему;
-  Внешний не имеет доступа ко внутреннему без создания объекта;
-  У внутренних классов есть модификаторы доступа;
-  Внутренний класс не может называться как внешний;










Особенности внутренних классов

-  Внутренний объект не существует без внешнего;
-  Внутренний имеет доступ ко всему внешнему;
-  Внешний не имеет доступа ко внутреннему без создания объекта;
-  У внутренних классов есть модификаторы доступа;
-  Внутренний класс не может называться как внешний;
-  Во внутреннем классе нельзя иметь не-final статические поля;











Особенности внутренних классов

-  Внутренний объект не существует без внешнего;
-  Внутренний имеет доступ ко всему внешнему;
-  Внешний не имеет доступа ко внутреннему без создания объекта;
-  У внутренних классов есть модификаторы доступа;
-  Внутренний класс не может называться как внешний;
-  Во внутреннем классе нельзя иметь не-final статические поля;
-  Объект внутреннего класса нельзя создать в статическом методе «внешнего» класса



Особенности внутренних классов

-  Внутренний объект не существует без внешнего;
-  Внутренний имеет доступ ко всему внешнему;
-  Внешний не имеет доступа ко внутреннему без создания объекта;
-  У внутренних классов есть модификаторы доступа;
-  Внутренний класс не может называться как внешний;
-  Во внутреннем классе нельзя иметь не-final статические поля;
-  Объект внутреннего класса нельзя создать в статическом методе «внешнего» класса
-  Со внутренними классами работает наследование и полиморфизм.

Ответьте на вопросы сообщением в чат

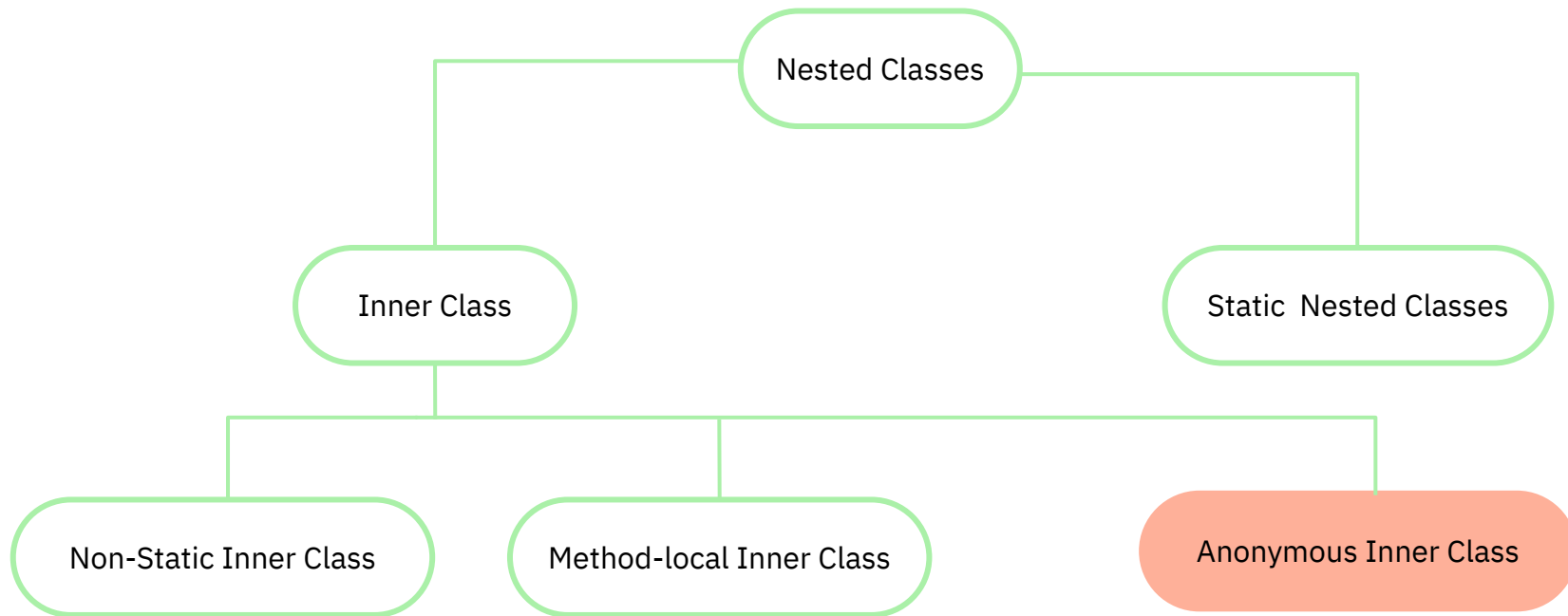
Вопросы:

1. Внутренний класс:
 - a. реализует композицию;
 - b. это служебный класс;
 - c. не требует объекта внешнего класса.
2. Инкапсуляция с использованием внутренних классов:
 - a. остаётся неизменной;
 - b. увеличивается;
 - c. уменьшается.
3. Статические поля внутренних классов:
 - a. могут существовать;
 - b. могут существовать только константными;
 - c. не могут существовать.





Вложенные классы





Локальный внутренний класс

```
1 public class Animal {  
2  
3     void performBehavior(boolean state) {  
4         class Brain {  
5             void sleep() {  
6                 if(state) {  
7                     System.out.println("Sleeping");  
8                 } else {  
9                     System.out.println("Not sleeping");  
10                }  
11            }  
12        }  
13        Brain brain = new Brain();  
14        brain.sleep();  
15    }  
16 }  
17
```



Вызов с локальным классом

```
1 Animal animal = new Animal();  
2 animal.performBehavior(true);
```

✓ 0.2s

Java

Sleeping



Особенности локальных классов



Сохраняет доступ ко всем полям и методам внешнего класса;



Особенности локальных классов






Сохраняет доступ ко всем полям и методам внешнего класса;



Должен иметь свои внутренние копии всех локальных переменных;



Особенности локальных классов

-  Сохраняет доступ ко всем полям и методам внешнего класса;
-  Должен иметь свои внутренние копии всех локальных переменных;
-  Имеют ссылку на окружающий экземпляр.



Статические вложенные классы

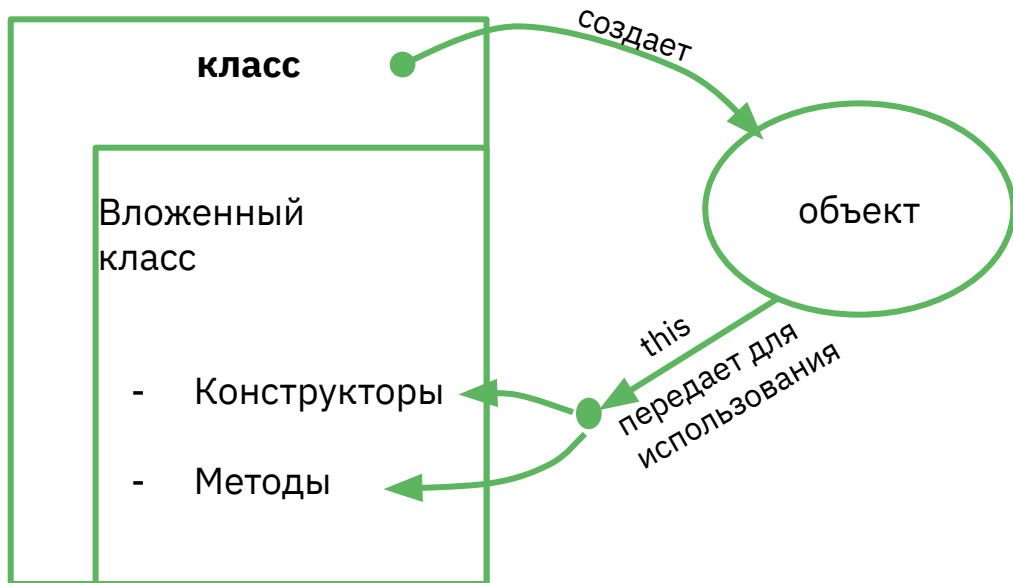
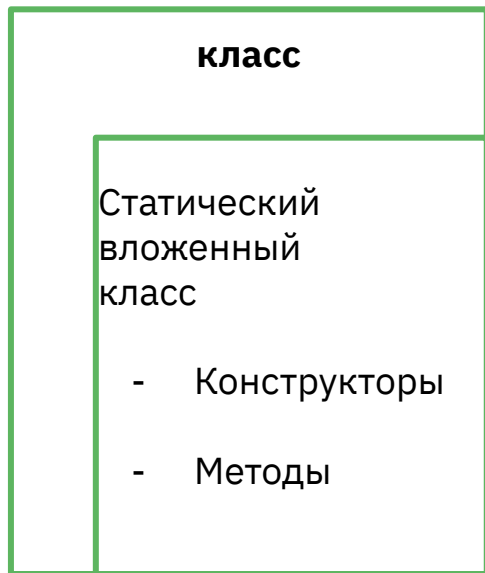


Статический класс

```
1 public class Cat {
2     private String name, color;
3     private int age;
4
5     public Cat() { }
6     public Cat(String name, String color, int age) {
7         this.name = name;
8         this.color = color;
9         this.age = age;
10    }
11
12    static class Voice {
13        private final int volume;
14        public Voice(int volume) { this.volume = volume; }
15        public void sayMur() {
16            System.out.printf(
17                "A cat purrs with volume %d\n", volume);
18        }
19    }
```



Отличие static и не static





Использование статического класса

```
1  Cat.Voice voice = new Cat.Voice(100);  
2  voice.sayMur();
```

✓ 0.3s

Java

A cat purrs with volume 100



Последнее о статике

```
1  for (int i = 0; i < 4; i++) {  
2      Cat.Voice voice = new Cat.Voice(100 + i);  
3      voice.sayMur();  
4  }
```

✓ 0.3s

Java

A cat purrs with volume 100

A cat purrs with volume 101

A cat purrs with volume 102

A cat purrs with volume 103

Ответьте на вопросы сообщением в чат

Вопросы:

1. Вложенный класс:
 - a. реализует композицию;
 - b. это локальный класс;
 - c. всегда публичный.
2. Статический вложенный класс обладает теми же свойствами, что:
 - a. константный метод;
 - b. внутренний класс;
 - c. статическое поле.





Механизм исключительных ситуаций



Исключение






Исключение - это отступление от общего правила, несоответствие обычному порядку вещей



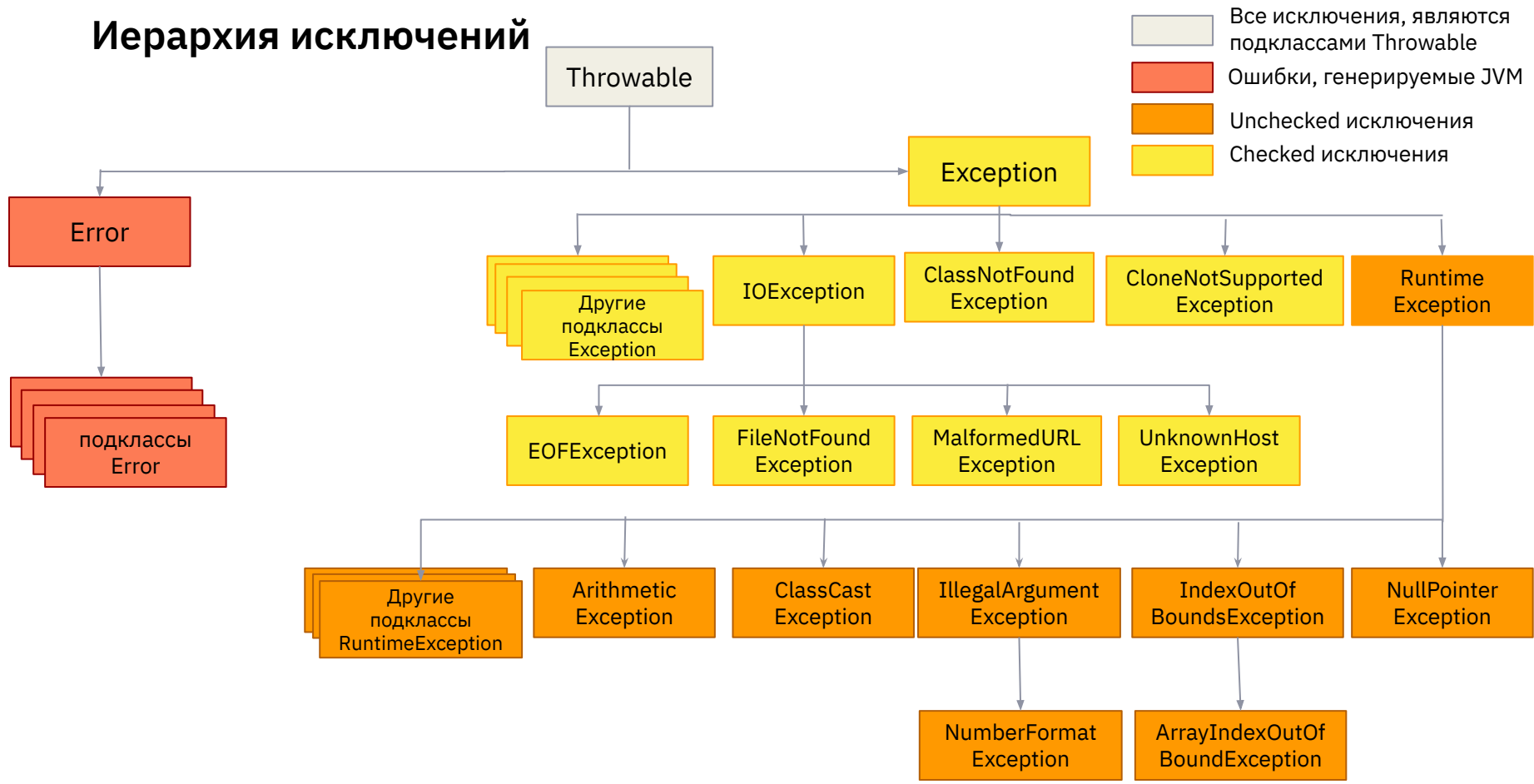


В общем случае, исключение - это ошибка. Но где?

-  Ошибка в коде программы;
-  Ошибка в действиях пользователя;
-  Ошибка в аппаратной части компьютера.



Иерархия исключений





Матрешка методов

```
1 private static int div0() {  
2     return 1 / 0;  
3 }  
4 private static int div1() {  
5     return div0();  
6 }  
7 private static int div2() {  
8     return div1();  
9 }
```

Java





Метод деления

```
1 private static int div0(int a, int b) {  
2     |     return a / b;  
3 }
```

✓ 0.3s

Java



Смысл исключения

```
1 private static int div0(int a, int b) {  
2     if (b != 0) {  
3         return a / b;  
4     }  
5     return /* ??? */;  
6 }
```

Java

```
1 private static int div0(int a, int b) {  
2     if (b == 0) {  
3         throw new RuntimeException("parameter error");  
4     }  
5     return a / b;  
6 }
```

Java



Исключение

```
1 System.out.println(div0(1,2));  
2 System.out.println(div0(1,0));
```

⊗ 0.7s

Java

0

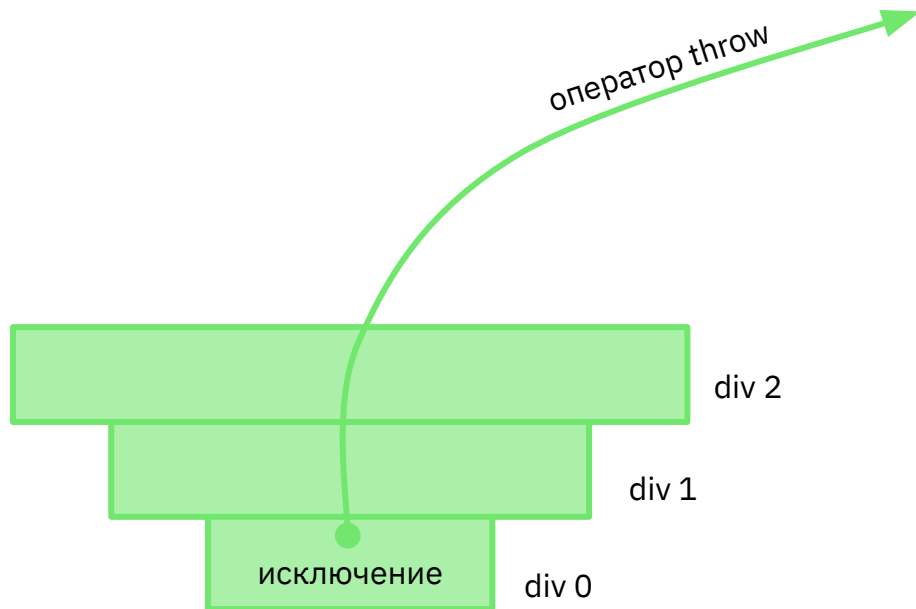
```
-----  
-----  
java.lang.RuntimeException: parameter error  
    at .div0(#22:3)  
    at .(#24:1)
```



Введение в многопоточность



Ключевое слово throw (бросить)

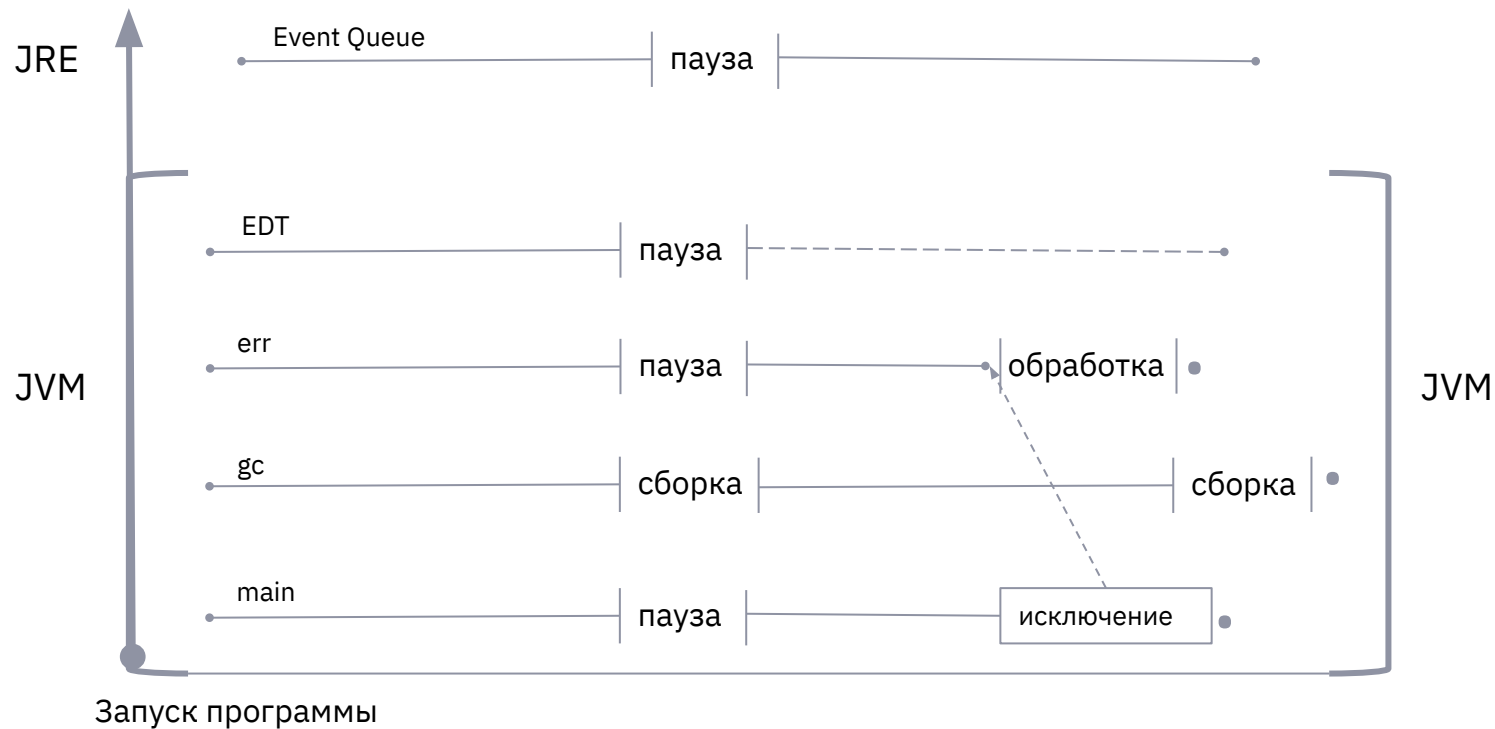


обработчик исключений





Передача объекта исключения потоку error





Стектрейс

```
Exception in thread "main" java.lang.RuntimeException Create breakpoint : parameter error
    at ru.gb.jcore.Main.div0(Main.java:5)
    at ru.gb.jcore.Main.div1(Main.java:9)
    at ru.gb.jcore.Main.div2(Main.java:12)
    at ru.gb.jcore.Main.main(Main.java:15)
```



Простой пример исключения

```
1  int[] arr = {1};  
2  System.out.println(arr[2])
```

⊗ 0.4s

Java

```
-----  
-----  
java.lang.ArrayIndexOutOfBoundsException: Index 2 out of bounds for  
length 1  
    at .(#17:1)
```




Объект исключения

```
1 RuntimeException e = new RuntimeException();
```

✓ 0.1s

Java

```
1 throw e;
```

⊗ 0.2s

Java

```
-----  
-----  
java.lang.RuntimeException: null  
    at .(#12:1)
```



Рантайм выводы

```
1 public static void methodA() {  
2     RuntimeException e = new RuntimeException();  
3     throw e;  
4 }
```

✓ 0.2s

Java

```
1 methodA();
```

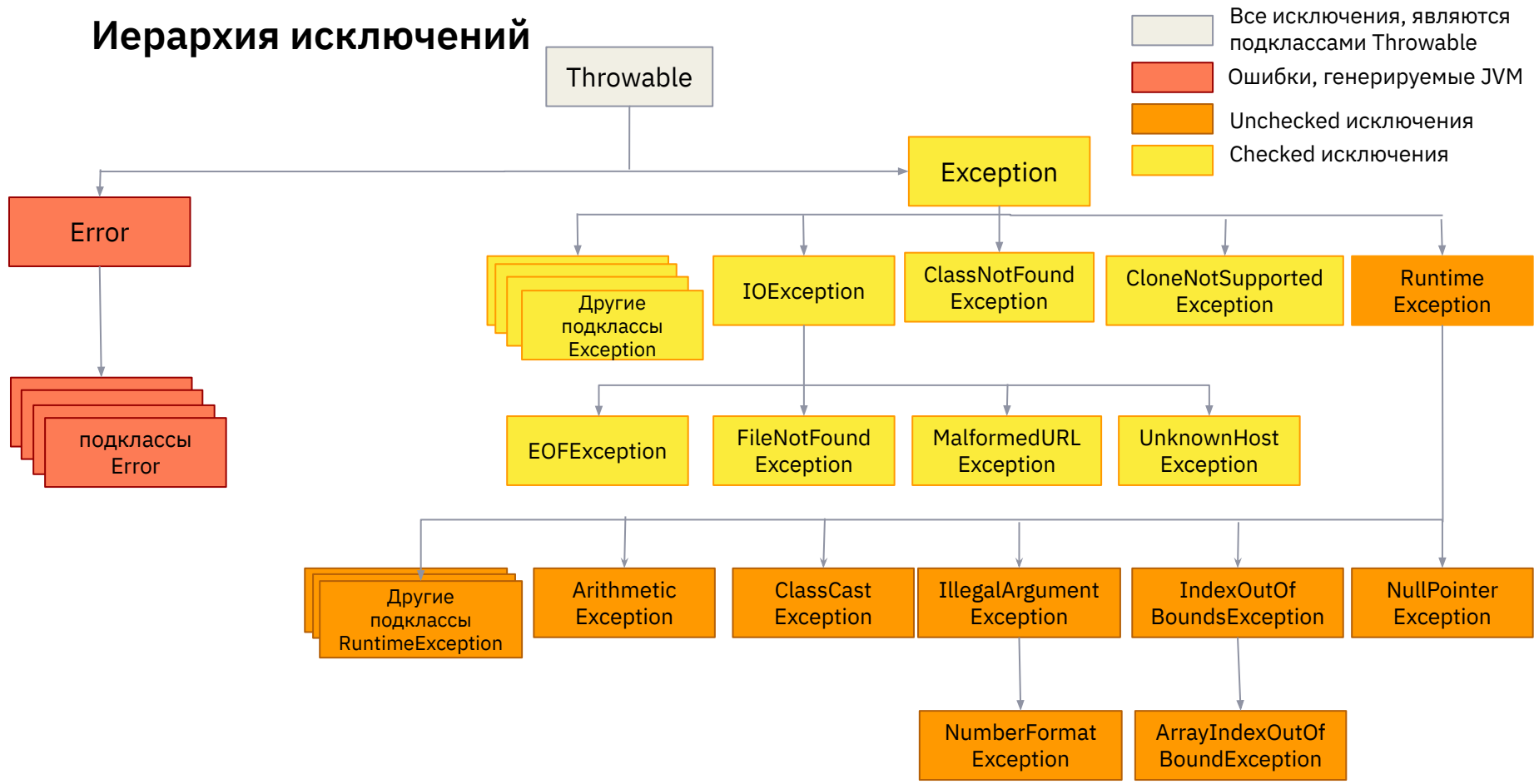
⊗ 0.1s

Java

```
-----  
-----  
java.lang.RuntimeException: null  
    at .methodA(#14:2)  
    at .(#15:1)
```



Иерархия исключений





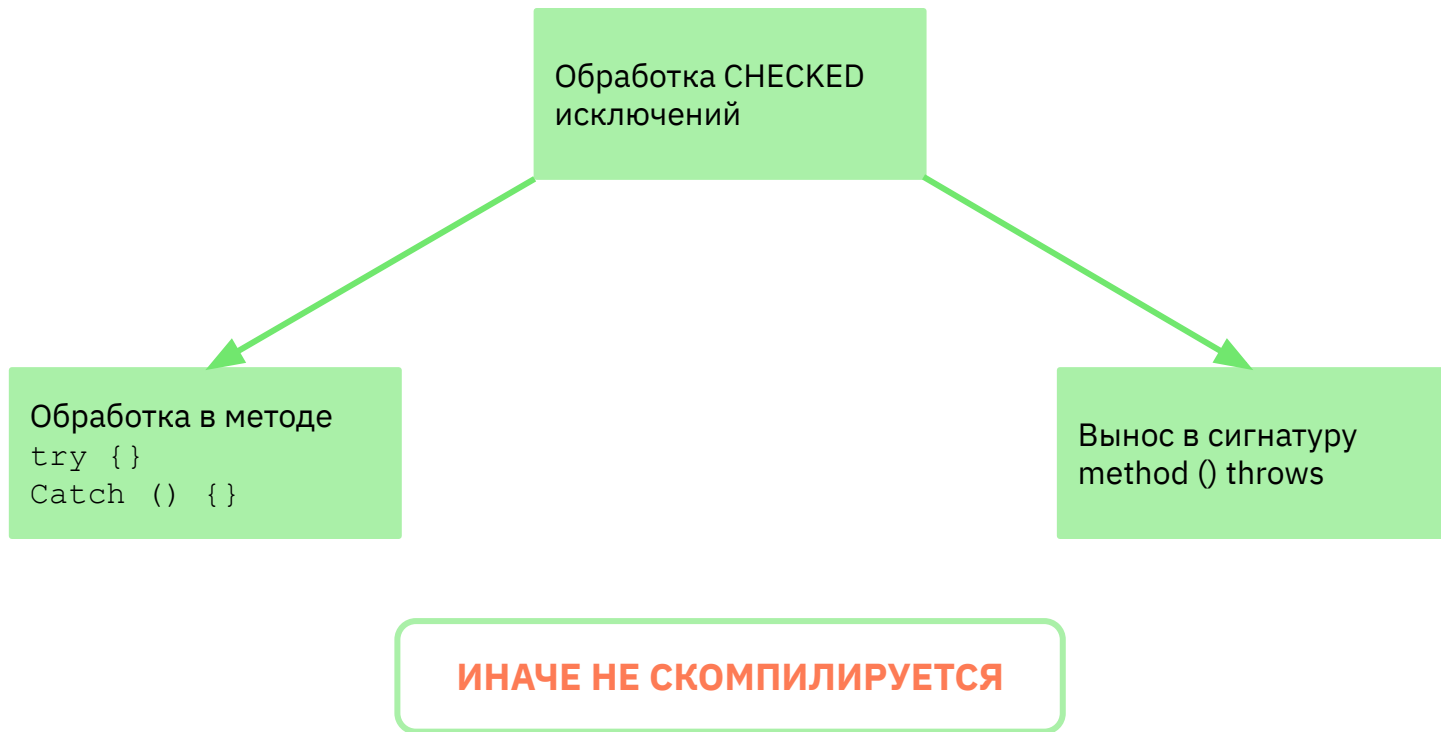
Пробуем ловить

```
try {  
    // что-то исключительное?  
} catch (Exception e) {  
    // случилось исключение или его наследники  
}  
// будто и не случилось ничего необычного
```

здесь может быть
любое количество
секций catch
(Exception)

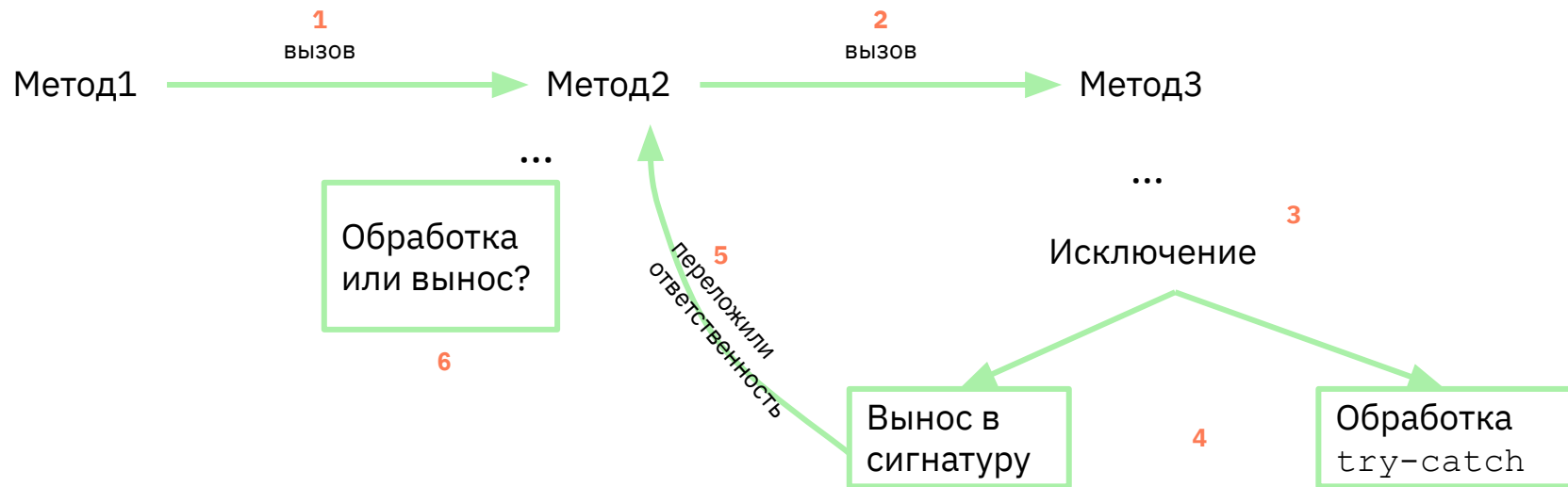


Варианты обработки checked исключений





Ответственность





Обработка вариант 1

```
1 public void methodB() {  
2     throw new IOException();  
3 }
```

⊗ 0.1s

Java

`throw new IOException();`
unreported exception java.io.IOException; must be caught or declared to be thrown

```
1 public void methodB() {  
2     try {  
3         throw new IOException();  
4     } catch (IOException e) {  
5         System.out.println(e.getMessage());  
6     }  
7 }
```

✓ 0.1s

Java



Обработка вариант 2

```
1 public void methodB() throws IOException {  
2     throw new IOException();  
3 }
```

✓ 0.2s

Java

```
1 public void methodA() {  
2     RuntimeException e = new RuntimeException();  
3     //throw e;  
4     methodB();  
5 }
```

⊗ 0.1s

Java

methodB()
unreported exception java.io.IOException; must be caught or declared to be thrown

```
1 class TestStream {  
2     TestStream() {  
3  
4     }  
5     int read() throws FileNotFoundException {  
6         new FileInputStream("file.txt");  
7         return 1;  
8     }  
9 }
```

✓ 0.1s

Java

```
1 TestStream stream = new TestStream();  
2 int i = stream.read();
```

⊗ 0.2s

Java



Не оставляйте исключения без внимания или с простым логированием





ключевое слово finally

```
1 class TestStream {  
2     TestStream() {  
3         System.out.println("construct OK");  
4     }  
5     int read() throws FileNotFoundException {  
6         new FileInputStream("file.txt");  
7         System.out.println("read OK");  
8         return 1;  
9     }  
10    void close() throws IOException {  
11        System.out.println("close OK");  
12        throw new IOException();  
13    }  
14 }
```

✓ 0.4s

Java

```
1 TestStream stream = null;  
2 try {  
3     stream = new TestStream();  
4     int i = stream.read();  
5 } catch (FileNotFoundException fnfe) {  
6     System.out.println("read NOT OK");  
7 } catch (IOException e) {  
8     System.out.println("close NOT OK");  
9 } finally {  
10    stream.close();  
11 }
```

✓ 0.2s

Java

construct OK

read NOT OK

close OK



Проблема

```
1 class TestStream {
2     TestStream() throws Exception {
3         // System.out.println("construct OK");
4         throw new Exception();
5     }
6     int read() throws FileNotFoundException {
7         new FileInputStream("file.txt");
8         System.out.println("read OK");
9         return 1;
10    }
11    void close() throws IOException {
12        System.out.println("close OK");
13        //throw new IOException();
14    }
15 }
```

✓ 0.2s

Java

```
1 TestStream stream = null;
2 try {
3     stream = new TestStream();
4     int i = stream.read();
5 } catch (FileNotFoundException fnfe) {
6     System.out.println("read NOT OK");
7 } catch (IOException e) {
8     System.out.println("close NOT OK");
9 } catch (Exception e) {
10    System.out.println("construct NOT OK");
11 } finally {
12    stream.close();
13 }
```

⊗ 0.2s

Java

construct NOT OK

java.lang.NullPointerException: null
at .(#24:1)



try с ресурсами

```
1 class TestStream implements Closeable {
2     TestStream() throws Exception {
3         // System.out.println("construct OK");
4         throw new Exception();
5     }
6     int read() throws FileNotFoundException {
7         new FileInputStream("file.txt");
8         System.out.println("read OK");
9         return 1;
10    }
11    public void close() throws IOException {
12        System.out.println("close OK");
13        throw new IOException();
14    }
15 }
```

✓ 0.2s

Java

```
1 try (TestStream stream = new TestStream()) {
2     int a = stream.read();
3 } catch (IOException e) {
4     new RuntimeException(e);
5 }
6
```

✗ 0.2s

Java



Исключение

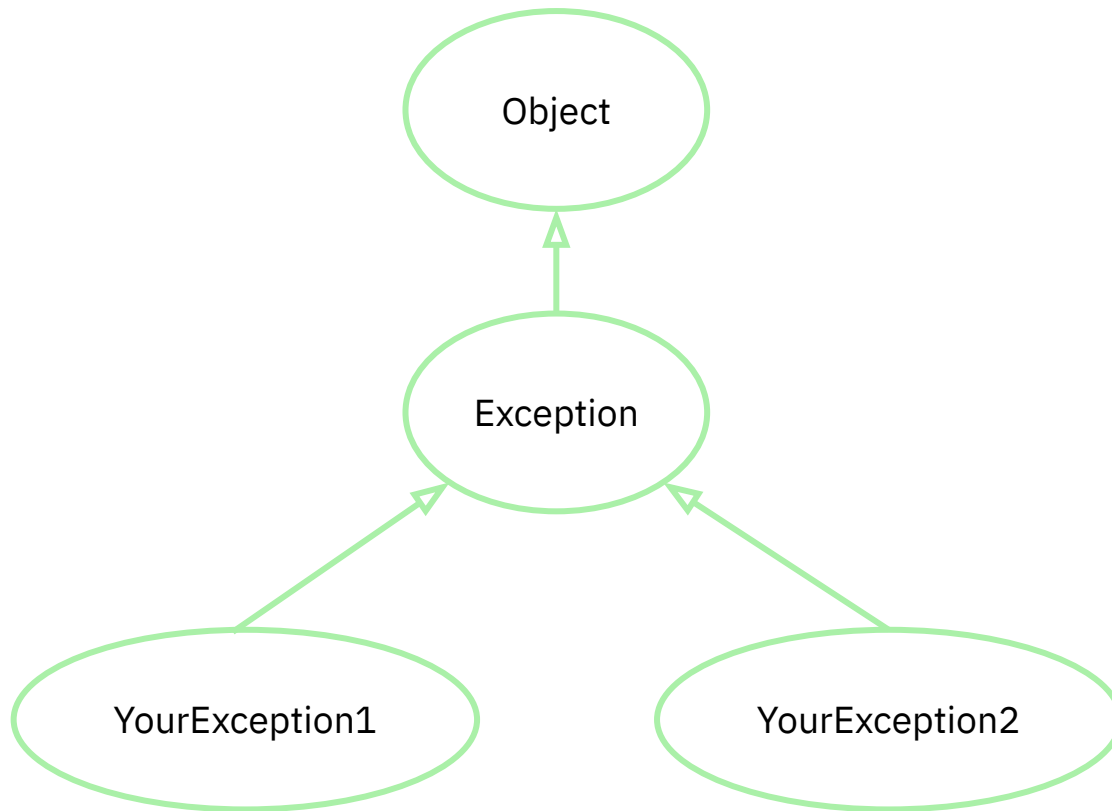


Исключение - это отступление от общего правила, несоответствие обычному порядку вещей











Наследование и полиморфизм в исключениях





На этом уроке

-  Перечисления;
-  Внутренние классы;
-  Вложенные классы;
-  Внутренние статические классы;
-  Механизм исключительных ситуаций;
-  Введение в многопоточность;





Практическое задание

- 📌 напишите два наследника класса Exception: ошибка преобразования строки и ошибка преобразования столбца;
- 📌 разработайте исключения-наследники так, чтобы они информировали пользователя в формате ожидание/реальность;
- 📌 для проверки напишите программу, преобразующую квадратный массив целых чисел 5x5 в сумму чисел в этом массиве, при этом, программа должна выбросить исключение, если строк или столбцов в исходном массиве окажется не 5.





Не стыдись учиться в зрелом возрасте:
лучше научиться поздно, чем никогда.

Эзоп