Java

Исключения в Java

Воспроизведём проблему

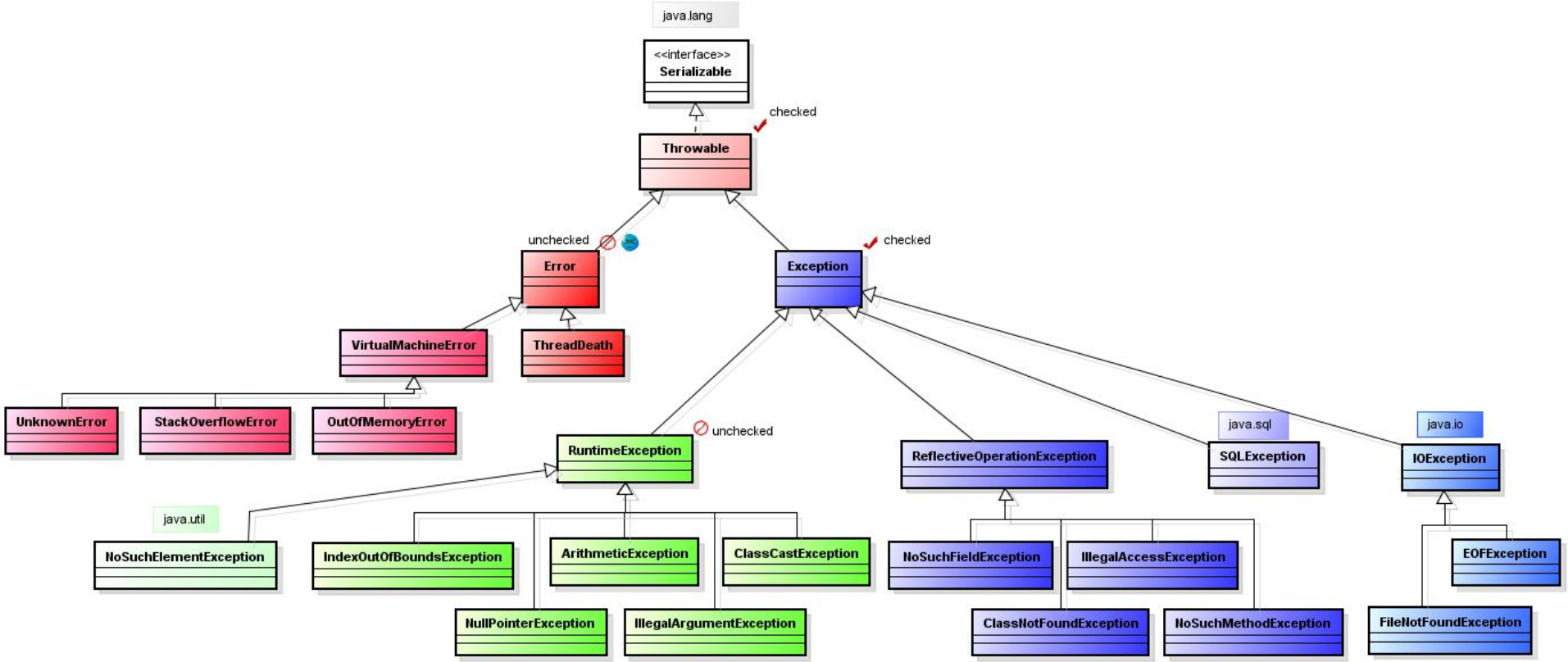
```
public static void main(String[] args) {
    hereWillBeTrouble(42, 0);
}
public static void hereWillBeTrouble(int a, int b) {
    int oops = a / b;
    System.out.println(oops);
}
```

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
    at Main.hereWillBeTrouble(Main.java:8)
    at Main.main(Main.java:3)

Process finished with exit code 1
```

Иерархия исключений и ошибки

[картинка]



Что такое исключения (Exceptions)

Эти исключительные ситуации возникают, если разработчик допустил невыполнимую операцию, не предусмотрел особые случаи в бизнес-логике программы (или сообщает о них с помощью исключений).

1. Невыполнимая операция - Java не знает, что делать дальше. Как раз из этого разряда деление на ноль. Другая частая ситуация — обращение к несуществующему элементу массива. Например, у нас в нём десять элементов, а мы пытаемся обратиться к одиннадцатому.

2. Особый случай в бизнес-логике программы - программируем задачу о перевозке волка, козы и капусты через реку: в лодке может быть только два пассажира, но волка с козой и козу с капустой нельзя оставлять на берегу вместе. Это и есть особый случай в бизнес-логике, который нельзя нарушать.

Что такое исключения (Exceptions)

Поскольку все классы исключений наследуются от класса Exception, то все они наследуют ряд его методов, которые позволяют получить информацию о характере исключения. Среди этих методов отметим наиболее важные:

- Метод getMessage() возвращает сообщение об исключении
- Метод getStackTrace() возвращает массив, содержащий трассировку стека исключения
- Метод printStackTrace() отображает трассировку стека

Что делать с исключениями

Простейший вариант — **ничего**; возникает исключение — программа просто прекращает работать.

Второе, что можно делать с исключениями, — это их обрабатывать.

Для этого нужно заключить кусок кода, который может вызвать исключение, в конструкцию try-catch.

Как это работает: если в блоке try возникает исключение, которое указано в блоке catch, то исполнение блока try прервётся и выполнится код из блока catch.

```
public static void main(String[] args) {
    hereWillBeTrouble(10, 0);
}

private static void hereWillBeTrouble(int a, int b) {
    int oops;
    try {
        System.out.println("Всё, что было до...");
        oops = a / b;
        System.out.println(oops);
        System.out.println("Всё, что будет после...");
    } catch (ArithmeticException e) {
        System.out.println("Говорили же не делить на ноль!");
        oops = 0;
    }
    System.out.println("Метод отработал");
}
```

Как читать исключение

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
    at Main.hereWillBeTrouble(Main.java:8)
    at Main.main(Main.java:3)

Process finished with exit code 1
```

`Exception in thread "main"` — это указание на поток, в котором произошло исключение. В нашей простой однопоточной программе это поток main.

`java.lang.ArithmeticException:` — какое исключение брошено. У нас это ArithmeticException. А java.lang.ArithmeticException — полное название класса вместе с пакетом, в котором он размещается.

`/ by zero` — весточка, которую принесло исключение. Дело в том, что одно и то же исключение нередко возникает по разным причинам. И тут мы видим стандартное пояснение «/ by zero» — из-за деления на ноль.

```
at Main.hereWillBeTrouble(Main.java:8)
at Main.main(Main.java:3)
```

— Стектрейс (Stack trace) — это упорядоченный список методов, сквозь которые исключение пронырнуло.

Как создать исключение

Раз исключение — это объект класса, то программисту всего-то и нужно, что создать объект с нужным классом исключения и бросить его с помощью оператора throw.

```
public static void main(String[] args) {
    hereWillBeTrouble(42, 0);
}
private static void hereWillBeTrouble(int a, int b) {
    if (b == 0) {
        throw new ArithmeticException("ты опять делишь на ноль?");
    }
    int oops = a / b;
    System.out.println(oops);
}
```

Проброс исключений

Три способа обойтись с исключением:

- Ничего не делать.
- Обработать в конструкции try-catch.
- **Пробросить наверх.**

Если метод **A** вызвал метод **B**, метод **B** вызвал метод **C**, а метод **C** пробросил исключение наверх, то оно всплывёт до метода **B** — и станет его проблемой.

Как пробросить исключение?

```
public static void main(String[] args) {
    hereWillBeTrouble();
}
private static void hereWillBeTrouble() throws ArithmeticException {
    System.out.println("Всё, что было до...");
    int oops = 42 / 0;
    System.out.println("Всё, что будет после...");
}
```

Что нам дал проброс исключения?

```
public static void main(String[] args) {
    try {
        hereWillBeTrouble();
    } catch (ArithmeticException ex) {
        System.out.println("Да, это случилось");
    }
}
private static void hereWillBeTrouble() throws ArithmeticException {
    System.out.println("Всё, что было до...");
    int oops = 42 / 0;
    System.out.println("Всё, что будет после...");
}
```

Проверяемые и непроверяемые исключения

Все исключения в Java делятся на две группы. Зовутся они checked и unchecked («проверяемые» и «непроверяемые»). Иногда их также называют «обрабатываемые» и «необрабатываемые».

Запомните! Проверяемые исключения обязательно нужно обрабатывать либо пробрасывать. Непроверяемые — по желанию.

Проверяемые и непроверяемые исключения

Без проблем компилируется и код, в котором кидают исключение явно:

```
private static void hereWillBeTrouble(int a, int b) {
    if (b == 0) {
        throw new ArithmeticException("Ты опять делишь на ноль?");
    }
    int oops = a / b;
    System.out.println(oops);
}
```

Варианты обработки проверяемого исключения

Первый вариант, try-catch:

```
public static void main(String[] args) {
    try {
        hereWillBeTrouble();
    } catch (CloneNotSupportedException e) {
        e.printStackTrace();
    }
}
```

Второй вариант, проброс выше:

```
public static void main(String[] args) throws CloneNotSupportedException {
    hereWillBeTrouble();
}
```

try-catch-finally

```
try{
    int[] numbers = new int[3];
    numbers[4]=45;
    System.out.println(numbers[4]);
}
catch(Exception ex){

    ex.printStackTrace();
}
finally{
    System.out.println("Блок finally");
}
System.out.println("Программа завершена");
```

Обработка нескольких исключений

```
int[] numbers = new int[3];
try{
    numbers[6]=45;
    numbers[6]=Integer.parseInt("gfd");
}
catch(ArrayIndexOutOfBoundsException ex){

    System.out.println("Выход за пределы массива");
}
catch(NumberFormatException ex){

    System.out.println("Ошибка преобразования из строки в число");
}
```

Создание своих классов исключений

Чтобы создать свой класс исключений, надо унаследовать его от класса Exception (или RuntimeException). Например, у нас есть класс, вычисляющий факториал, и нам надо выбрасывать специальное исключение, если число, передаваемое в метод, меньше 1:

```
class Factorial{
    public static int getFactorial(int num) throws
FactorialException{
        int result=1;
        if(num<1) throw new FactorialException("The
number is less than 1", num);
        for(int i=1; i<=num;i++){
            result*=i;
        }
        return result;
    }
}
```

```
class FactorialException extends Exception{
    private int number;
    public int getNumber(){return number;}
    public FactorialException(String message, int num){
        super(message);
        number=num;
    }
}
```

Источники

1. https://javastudy.ru/interview/exceptions/
2. https://skillbox.ru/media/base/isklyucheniya_v_java_chast_1/
3. https://skillbox.ru/media/base/isklyucheniya-v-java-chast-2/
4. https://metanit.com/java/tutorial/2.10.php
5. https://metanit.com/java/tutorial/4.1.php
6. https://highload.today/isklyucheniya-v-java/