ИСКЛЮЧЕНИЯ

Исключения - это такая ситуация, когда дальнейшее выполнение программы либо невозможно, либо нецелесообразно.

Например, целочисленное деление на ноль NullPointerException, попытка получить 10-й элемент из массива размером в 5 элементов ArrayIndexOutOfBoundsException, попытка получить доступ к объекту по null ссылке NullPointerException, и т. д.

Исключения - объект наследник класса Throwable, который выброшен при помощи ключевого слова throw.

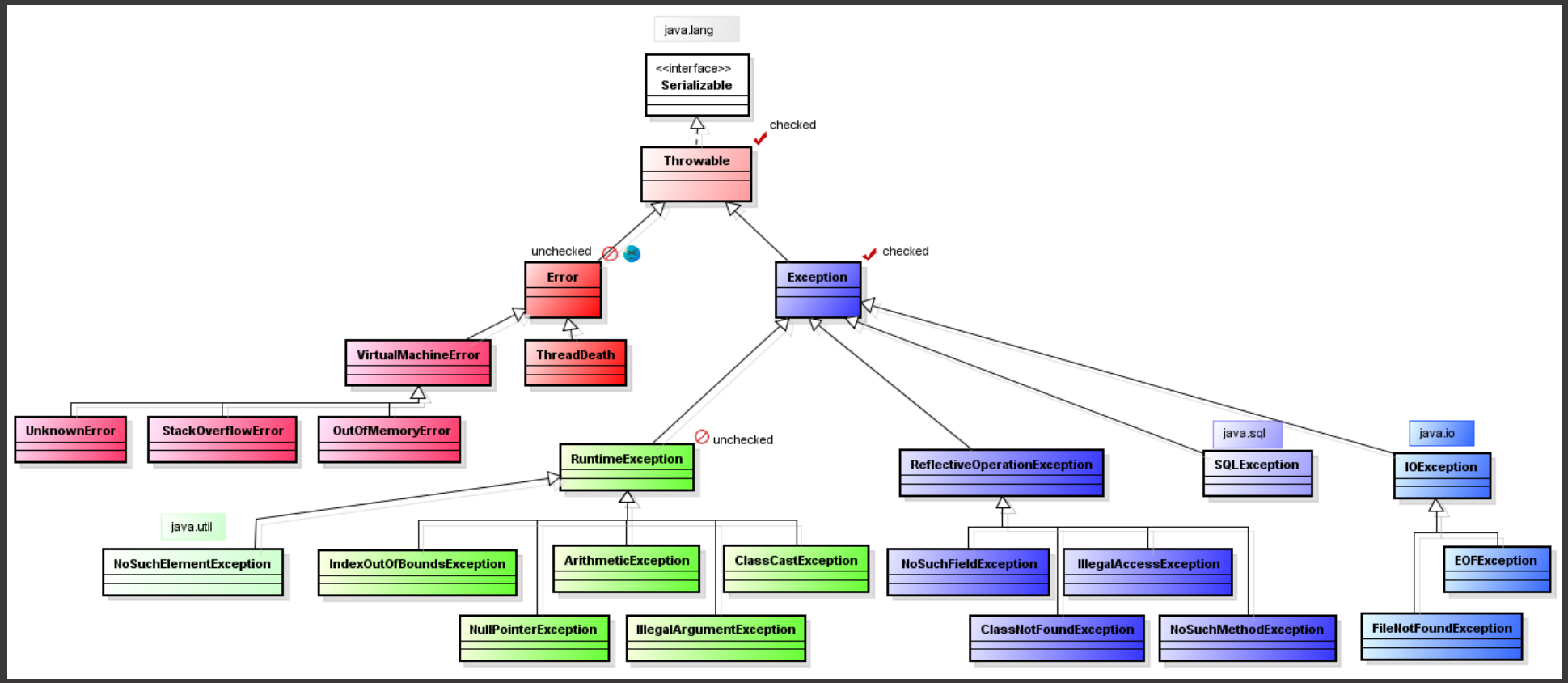
```
1 public void setAge(int age){
2     if (age < 18) {
3         throw new IllegalArgumentException("age не может быть меньше 1
4     }
5     this.age = age;
6 }
```

Stack trace - когда создается исключение, собирается стэк вызова. Развернутый стэк вызовов называется stack trace, он позволяет найти где в коде возникла проблема.

Методы исключений

- public String getMessage() возвращает сообщение о произошедшем исключении
- public Throwable getCause() возвращает причину исключения, ссылку на объект исключения, которое стало причиной текущего
- public void printStackTrace() выведение результата toString() совместно с трассировкой стека в System.err, поток вывода ошибок
- public StackTraceElement[] getStackTrace() возвращает массива, где каждый элемент в трассировке стека

Иерархия исключений



Иерархия исключений

- Error ошибки JVM, их обработка не имеет смысла
- Exception корневой класс для исключений, которые требуют обработки на этапе компиляции
- RuntimeException корневой класс для исключений, которые не требуют обработки на этапе компиляции

Обработка исключений / try-catch

В блок try помещается потенциально опасный код. Если происходит исключительная ситуация, то исполнение кода передается блоку catch. В этом случае блок catch перехватывает исключения, определенные в круглых скобках.

Обработка исключений / try-catch

```
 1 Object data = "42";
 2 try {
 3     /* потенциально опасный код приводит к ClassCastException */
 4     Integer integer = (Integer) data;/* управление переходит в catch
 5     /* если код порождает исключение,
 6        то остальной код блока try не выполнится */
 7 } catch (ClassCastException e) {
 8     /* ссылка на объект исключения передается в переменную e,
 9        блок способен перехватить только
10        ClassCastException и его наследников */
11     System.out.println("Обработка исключения: " + e.getMessage());
12 }
13 /* код после try catch продолжит выполнение */
```

Объединение блоков catch

Блоки catch могут идти друг за другом и обрабатывать разные исключения разными способами

```
1 try {
2     // потенциально опаный код
3 } catch (ClassCastException e) {
4     // Обработка исключения ClassCastException и его наследников
5 } catch (NullPointerException e) {
6     // Обработка исключения NullPointerException и его наследников
7 }
```

Объединение блоков catch

Обработку нескольких исключений можно объединить в одном блоке, тогда все они будут обработаны одним способом

```
1 try {
2     /* потенциально опаный код */
3 } catch (ClassCastException | NullPointerException e) {
4     /* Обработка ClassCastException, NullPointerException
5        и их наследников */
6 }
```

Объединение блоков catch

Можно перехватить исключение через общий тип данных. В таком случае все исключения данного типа будут обработаны, такой вариант не всегда может подойти

```
try {
    // потенциально опаный код
} catch (RuntimeException e) {
    // Обработка RuntimeException и его наследников
}
```

Необязательный блок finally

Используется, чтобы выполнить инструкции, вне зависимости от того было выброшено исключение или нет. Код выполнится даже если это исключение не перехватывается в блоке catch.

Обычно используется для высвобождения ресурсов, открытых в блоке try

```
1 try {
2     // потенциально опаный код
3 } catch (RuntimeException e) {
4     // Обработка RuntimeException и его наследников
5 } finally {
6     // закрытие ресурсов или любой другой код
7 }
```

Обрабатываемые / отслеживаемые времени компиляции / checked

К ним относятся Throwable и все его наследники, кроме ветвей Error и RuntimeException.

Такие исключения вынуждают программиста либо их обработать, либо пробросить на уровень выше в сигнатуре метода с помощью ключевого слова throws.

Обрабатываемые / отслеживаемые времени компиляции / checked

```java
class Reader {
    // исключение проброшено уровнемы выше
    public void read(File files) throws IOException {
        // код
        // было выброшено исключение времени компиляции
        throw new IOException("Чтение невозможно");
    }
    public void read(File[] files) throws IOException {
        for (File file : files) {
            // в методе выброшено исключение,
            // если оно не обрабатывается в блоке catch,
            // то снова пробрвсывается уровнем выше
            read(file);
        }
    }
}
```

Обрабатываемые / отслеживаемые времени компиляции / checked

```
1  class Application {
2      public static void main(String[] args) {
3          Reader reader = new Reader();
4          try {
5              // обрабатывается исключение, которые выброшено в методе
6              reader.read(new File[]{new File("one.txt"),
7                                     new File("two.txt")});
8          } catch (IOException e) {
9              e.printStackTrace();
10         }
11     }
12 }
```

Необрабатываемые / неотслеживаемые времени выполнения / unchecked

К ним относятся Throwable и все его наследники, кроме ветвей Error и Exception.

Такие исключения не вынуждают программиста их обрабатывать.

Собственные исключения

Чтобы написать свое исключение достаточно унаследовать класс Throwable или один из его потомков. Обычно это Exception или RuntimeException. Выбор из этих двух зависит исключительно от ситуации: хотите, чтобы программист обязательно обрабатывал исключение, тогда Exception, а если нет, то RuntimeException.