Java exceptions

Exceptions

Исключение — 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.

Exceptions

Исключение в рамках языка Java — это объект наследник класса Throwable, который выброшен при помощи ключевого слова throw.

```
throw new NumberFormatException("NumberException");
```

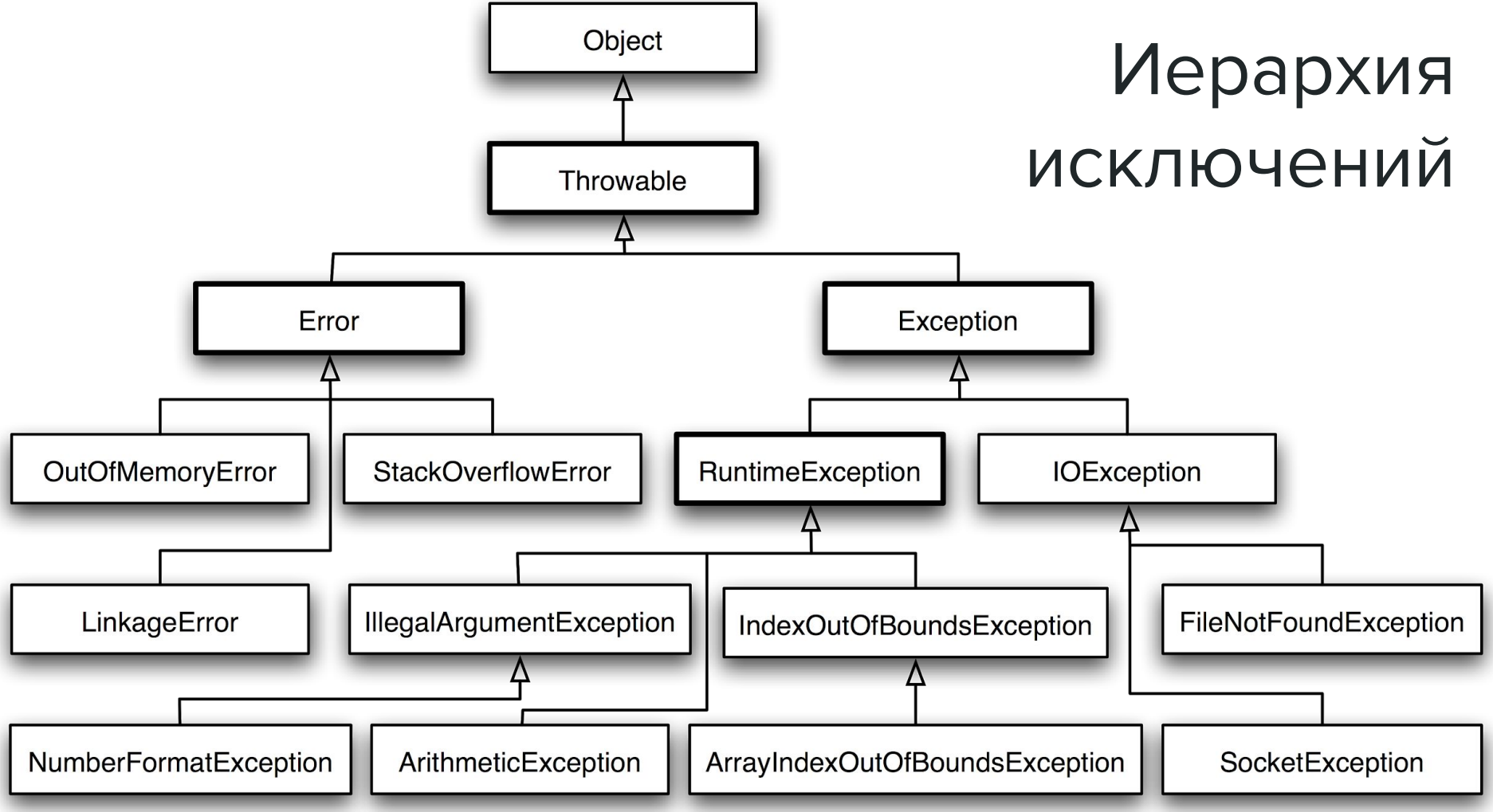
Stack trace

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

Методы исключений

- public String **getMessage()** возврат подробного сообщения о произошедшем исключении
- public Throwable **getCause()** возврат причины исключения
- public void **printStackTrace()** выведение результата toString() совместно с трассировкой стека в System.err, поток вывода ошибок
- public StackTraceElement [] **getStackTrace()** возврат массива, содержащего каждый элемент в трассировке стека
- public Throwable **fillInStackTrace()** заполняет трассировку стека данного объекта Throwable текущей трассировкой стека, дополняя какую-либо предшествующую информацию в трассировке стека

Иерархия исключений



Stack trace

Error — ошибки JVM, их обработка не имеет смысла;

Exception — корневой класс для всех пользовательских и библиотечных исключений;

RuntimeException — корневой класс для всех пользовательских исключений, которые не требуют обработки на этапе компиляции.

try-catch finally

Обработка исключений осуществляется при помощи структуры **try-catch**

```
Object data = "42";
try {
   // потенциально опасный код
   Integer integer = (Integer) data; // ClassCastException
   System.out.println("Hello after disaster!"); // код не выполнится
} catch (ClassCastException e) {
   System.out.println("We caught exception here: " + e.getMessage());
}
System.out.println("Here program finishes"); // код выполнится
```

Когда **выбросится исключение**, то **исполнение кода передается блоку catch**.
В этом случае блок catch ловит исключение ClassCastException и всех его наследников

try-catch finally

Несколько блоков catch | объединение в одном блоке catch

Блоки catch могут идти друг за другом и обрабатывать разные исключения

```
catch (ClassCastException e) {}
```

```
catch (ArrayIndexOutOfBoundsException e) {}
```

Можно отловить любое исключение через RuntimeException

```
catch (RuntimeException e) {}
```

Либо объединить их в одном блоке (Java 7)

```
catch (ClassCastException | ArrayIndexOutOfBoundsException e){}
```

try-catch finally

Для того, чтобы выполнить какую-то операцию, независимо от того было выброшено исключение или нет, используется блок finally:

```
try { }
catch (RuntimeException e) {
   e.printStackTrace();
}
finally {
   // Этот блок выполнится всегда.
   System.out.println("I will be invoked always");
}
```

Обычно используется для высвобождения ресурсов, открытых в блоке try

Checked exceptions (обрабатываемые исключения)

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.

Checked exceptions (обрабатываемые исключения)

Вариант решения 1 — обработать в блоке try-catch:

```
try {
   throw new Exception("Useless exception");
} catch (Exception e) {
   // Исключение обработано, все хорошо.
   e.printStackTrace();
}
```

Checked exceptions (обрабатываемые исключения)

Вариант решения 2 — пробросить на уровень выше:

```
public static void main(String[] args) throws Exception {
   throw new Exception("Useless exception");
}
```

Checked exceptions (обрабатываемые исключения)

Вариант решения 2 — пробросить на уровень выше:

```
public static void main(String[] args) throws Exception {
   throw new Exception("Useless exception");
}
```

Если вызывается метод, который пробрасывает checked exception, то необходимо снова провести обработку исключения:

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 вы, чтобы программист обязательно обрабатывал ваше исключение, тогда Exception,
а если нет, то RuntimeException.

Собственные исключения

```
public class MyCoolUncheckedException extends RuntimeException {
    public MyCoolUncheckedException(String message) {
        super(message);
    }
    public MyCoolUncheckedException(String message, Throwable cause) {
        super(message, cause);
    }
}
public class MyCoolCheckedException extends Exception {
    public MyCoolCheckedException(String message) {
        super(message);
    }
    public MyCoolCheckedException(String message, Throwable cause) {
        super(message, cause);
    }
}
```

Когда мы заворачиваем одно исключение в другое, то при выводе трейса первое будет в части caused by

Собственные исключения

```
// Все хорошо, нет обрабатываемых исключений.
public static void main(String[] args) {
  try {
    // Вынуждены обработать.
    throwsChecked();
  }
  catch (MyCoolCheckedException e) {
    // Заворачиваем словленное исключение в необрабатываемое и выбрасываем.
    // Теперь компилятор спокоен.
    throw new MyCoolUncheckedException("Wrapped", e);
  }
}
// Все хорошо, исключение будет обработано уровнем выше.
private static void throwsChecked() throws MyCoolCheckedException {
  throw new MyCoolCheckedException("Useless exception");
}
```