Хекслет

Ошибки и исключения

Наставник:
Полковников Дмитрий

Введение

Ошибка может произойти в любой момент работы программы, как по вине пользователя, так и по вине программиста или окружения.

Глобально ошибки можно разделить на два типа: ошибки компиляции и ошибки работы программы.

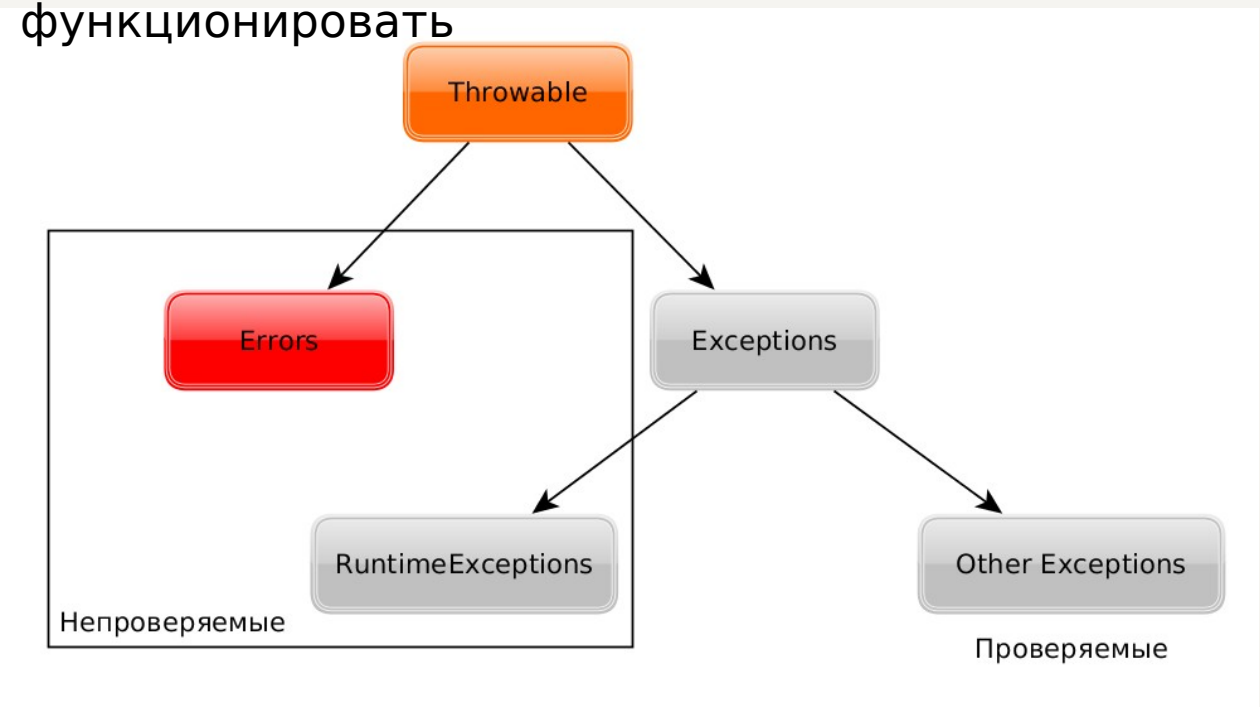
Большую часть ошибок можно избежать или обработать.

План

- Определение и иерархия.
- Способы обработки исключений. Синтаксис:
  - try-catch-finally
  - try with resources
  - throws
- Генерация исключений
- Реализация собственных исключений
- Демо
- Итог

Определение и иерархия

Исключительная ситуация — ситуация в которой программа не может продолжать правильно функционировать



Способы обработки исключений

- Конструкция try-catch-finally
- Ключевое слово throws

try-catch-finally

```
try {
    // Испольняемый код,
    // который может сгенерировать исключение
} catch (Exception exception) {
    // Обработка исключения
} finally {
    // Общий код, который выполнится в любом случае
}


// Пример обработки разных типов исключений
try {...}
catch (ArithmeticException | IndexOutOfBoundsException exception) {...}
catch (Exception exception) {...}
finally {...}
```

try with resources

В случаях, когда требуется работать объектами типа реализующими интерфейс Closeable , удобно использовать конструкцию try with resources

```
try (FileWriter fileWriter = new FileWriter("filename.txt")) {
    fileWriter.append("some string");
} catch (IOException e) {
    e.printStackTrace();
}
```

Ключевое слово throws

В случаях, когда не получается обработать checked исключение там же, где оно было сгенерировано, можно пробросить его на уровень выше с помощью ключевого слова throws

```
public static void method() throws IOException {
    throw new IOException();
}
```

Генерация исключений

Часто в процессе разработки приходится генерировать (выбрасывать) исключения вручную. Например при валидации входных данных пользователя.

Для генерации исключения используется ключевое слово throw

```
public void validateAge(int age) {
    if (age < 6) {
        throw new RuntimeException("Пользователь должен быть старше 6 лет");
    }
    if (age > 18) {
        throw new RuntimeException("Пользователь должен быть младше 18 лет");
    }
}
```

Реализация собственных исключений

Исключения - классы, наследующие класс Throwable

Для реализации собственного исключения необходимо создать класс, унаследованный от классов RuntimeException (unchecked) или Exception (checked)

```
public class ValidationException extends RuntimeException {
    public String fieldName;

    public ValidationException(String message, String fieldName) {
        super(message);
        this.fieldName = fieldName;
    }
}
```

Демо

- Среда разработки: Intellij Idea 2022.3
- SDK: OpenJDK 17.0.5

Итог

- Рассмотрели какие могут возникать проблемы в процессе работы программы
- Узнали какие бывают исключительные ситуации, их структуру и как их можно обрабатывать:
- Изучили несколько новых конструкций языка Java:
  - try-catch-finally
  - try with resources
  - throws
  - Выброс исключений (throw)
  - Реализация собственных исключений

Спасибо за внимание