

# Язык Java

## Глава VI. Обработка ошибок

Виталий Витальевич Перевощиков

Осенний семестр 2021

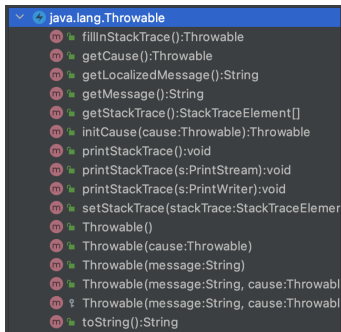
1. Исключения
2. Иерархия исключений
3. Обработка исключений
4. Логирование

# Исключения

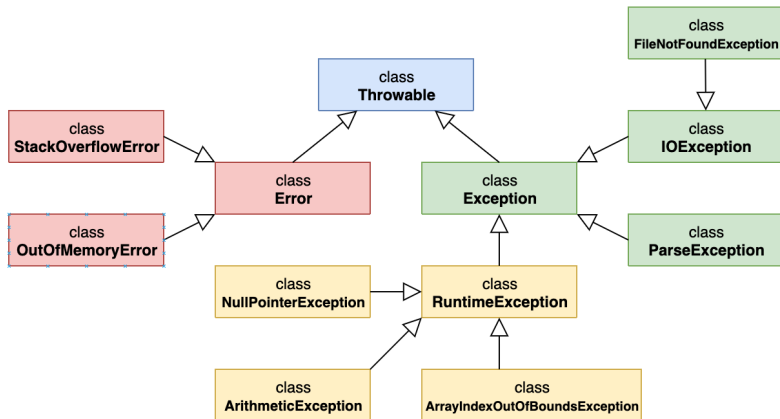
- Исключение - возникновение ошибок и непредвиденных ситуаций при выполнении программы, делающих дальнейшие вычисления невозможными или бессмысленными
- Причины исключений:
  - Ошибки программирования
  - Неправильные действия пользователя
  - Нехватка вычислительных ресурсов, доступных программе
  - Сетевые ошибки

# Исключения

- Объект исключения - объект, унаследованный от класса Throwable



# Иерархия исключений



Серьезные ошибки



Непроверяемые исключения (unchecked)



Проверяемые исключения (checked)

# Примеры критических ошибок

```
// Переполнения стека из-за бесконечной рекурсии (StackOverflowError)
static long calculateFactorial(int number) {
    return number * calculateFactorial(number - 1);
}

// Превышение лимита на динамическую память (OutOfMemoryError)
static int[] createLongArray() {
    return new int[Integer.MAX_VALUE];
}
```

# Примеры непроверяемых исключений

```
// Ссылка на пустое место (NullPointerException)
String string = null;
int length = string.length();

// Арифметическая ошибка (ArithmeticException)
int value = 1 / 0;

// Выход за границы массива (ArrayIndexOutOfBoundsException)
int[] array = { 1, 2, 3 };
int item = array[10];

// Выход за границы строки (StringIndexOutOfBoundsException)
String shortString = "hi";
char symbol = shortString.charAt(10);

// Ошибка при явном приведении типов (ClassCastException)
Object x = "hello";
Integer y = (Integer)x;
```

# Примеры проверяемых исключений

```
// файл с именем file.txt отсутствует в папке tmp (FileNotFoundException)  
Scanner scanner = new Scanner(new File("/tmp/file.txt"));
```

Unhandled exception: java.io.FileNotFoundException

Add exception to method signature

More actions...

```
// ошибка при преобразовании строки в дату (ParseException)  
new SimpleDateFormat().parse("что-то не похожее на дату");
```

Unhandled exception: java.text.ParseException

Add exception to method signature

More actions...



# Создание собственного исключения

```
class DatabaseClientException extends RuntimeException {  
  
    public DatabaseClientException(String message) {  
        super(message);  
    }  
}  
  
// клиент для подключения к базе данных  
public class DatabaseClient {  
  
    // строка с параметрами подключения к базе данных  
    private String connectionString;  
  
    public DatabaseClient(String connectionString) {  
        if (connectionString == null) {  
            throw new DatabaseClientException("connectionString is null");  
        }  
        this.connectionString = connectionString;  
    }  
}
```

```
class Test {  
    public static void main(String[] args) {  
        DatabaseClient client = new DatabaseClient(null);  
    }  
}
```

```
Exception in thread "main" ru.bfu.ipmit.DatabaseClientException Create breakpoint :  
connectionString is null  
at ru.bfu.ipmit.DatabaseClient.<init>(DatabaseClient.java:18)  
at ru.bfu.ipmit.Test.main(DatabaseClient.java:26)
```

# Обработка исключений

```
static void readFile() {  
    // файл с именем file.txt отсутствует в папке tmp (FileNotFoundException)  
    Scanner scanner = new Scanner(new File("/tmp/file.txt"));  
}
```

Unhandled exception: java.io.FileNotFoundException  
Add exception to method signature  
More actions...

- **Вариант 1:** делегирование обработки исключения потребителю метода

```
static void readFile() throws FileNotFoundException {  
    // файл с именем file.txt отсутствует в папке tmp (FileNotFoundException)  
    Scanner scanner = new Scanner(new File("/tmp/file.txt"));  
}
```

- **Вариант 2:** перехват исключения (try-catch) и самостоятельная обработка

```
static void readFile() {  
    try {  
        // файл с именем file.txt отсутствует в папке tmp (FileNotFoundException)  
        Scanner scanner = new Scanner(new File("/tmp/file.txt"));  
    } catch (FileNotFoundException e) {  
        e.printStackTrace();  
    }  
}
```

# Обработка исключений

- **Вариант 3:** перехват исключения (try-catch) и генерация нового исключения

```
static void readFile() {  
    try {  
        // файл с именем file.txt отсутствует в папке tmp (FileNotFoundException)  
        Scanner scanner = new Scanner(new File("/tmp/file.txt"));  
    } catch (FileNotFoundException e) {  
        throw new RuntimeException("Ошибка при открытии файла /tmp/file.txt", e);  
    }  
}
```

Причина исключения

```
Exception in thread "main" java.lang.RuntimeException Create breakpoint : Ошибка при открытии файла  
    at ru.bfu.ipmit.Main.readFile(Main.java:16)  
    at ru.bfu.ipmit.Main.main(Main.java:21)  
Caused by: java.io.FileNotFoundException Create breakpoint : /tmp/file.t (No such file or directory)  
    at java.base/java.io.FileInputStream.open0(Native Method)  
    at java.base/java.io.FileInputStream.open(FileInputStream.java:211)  
    at java.base/java.io.FileInputStream.<init>(FileInputStream.java:153)  
    at java.base/java.util.Scanner.<init>(Scanner.java:639)  
    at ru.bfu.ipmit.Main.readFile(Main.java:14)  
    ... 1 more
```

# Обработка нескольких исключений

```
// исключение для случая, если пароль не введен
class EmptyPasswordException extends Exception {
}

// исключение для случая, если пароль короткий
class ShortPasswordException extends Exception {
}

class User {
    private String password;

    User(String password) throws EmptyPasswordException, ShortPasswordException {
        if (password.equals("")) {
            throw new EmptyPasswordException();
        }
        if (password.length() < 6) {
            throw new ShortPasswordException();
        }
    }
}
```

```
try {
    User user = new User("");
} catch (EmptyPasswordException e) {
    System.out.println("Пустой пароль");
} catch (ShortPasswordException e) {
    System.out.println("Длина пароля должна быть как минимум 6 символов");
}
```

```
try {
    User user = new User("");
} catch (EmptyPasswordException | ShortPasswordException e) {
    System.out.println("Длина пароля должна быть как минимум 6 символов");
}
```

# finally

- try-catch-finally

```
OutputStream outputStream = new FileOutputStream("/tmp/file.txt");
try {
    outputStream.write("Hello".getBytes(StandardCharsets.UTF_8));
} catch (IOException e) {
    e.printStackTrace();
} finally {
    // освобождение ресурсов
    outputStream.close();
}
```

- try-finally

```
OutputStream outputStream = new FileOutputStream("/tmp/file.txt");
try {
    outputStream.write("Hello".getBytes(StandardCharsets.UTF_8));
} finally {
    // освобождение ресурсов
    outputStream.close();
}
```

- Альтернатива: try с ресурсами

```
try (OutputStream outputStream = new FileOutputStream("/tmp/file.txt")) {
    outputStream.write("Hello".getBytes(StandardCharsets.UTF_8));
}
```

# Антипаттерны

- “Проглатывание” исключения

```
static String readStringFromFile() {  
    String result = null;  
    try {  
        Scanner scanner = new Scanner(new File("/tmp/file.txt"));  
        result = scanner.next();  
    } catch (FileNotFoundException e) {} // "проглатывание" исключения  
    return result;  
}
```

- Возвращение значения в finally-блоке

```
static String readStringFromFile() {  
    String result = null;  
    try {  
        Scanner scanner = new Scanner(new File("/tmp/file.txt"));  
        result = scanner.next();  
    } finally {  
        return result; // "проглатывание" исключения  
    }  
}
```

# Антипаттерны

- Генерирование нового исключения в finally-блоке

```
static String readStringFromFile() throws Exception {  
    try {  
        Scanner scanner = new Scanner(new File("/tmp/file.txt"));  
        return scanner.next();  
    } finally {  
        // "проглатывание" исходного исключения (FileNotFoundException)  
        throw new Exception("Ой! Что-то пошло не так.");  
    }  
}
```

- Использование исключений как goto-оператор
  - Условный оператор if:

```
static int getLength(String string) {  
    try {  
        return string.length(); // if (string != null)  
    } catch (NullPointerException e) {  
        return 0; // else  
    }  
}
```

- Цикл for:

```
static int getMinimum(int[] array, int index) {  
    int min, current = 0;  
    try {  
        current = array[index];  
        min = getMinimum(array, index - 1);  
        int[] x = new int[current - min];  
    } catch (IndexOutOfBoundsException e) {  
        min = Integer.MAX_VALUE; // при index = -1  
    } catch (RuntimeException e) {  
        min = current; // при current < min  
    }  
    return min;  
}
```

# Логирование

- Лог - это журнал, в котором хранится информация состоянии выполнения программы
- Задачи логирования:
  - Помощь при отладке программы
  - Аудит
  - Интеграция с различными системами управления логами (Splunk, LogDNA, ...):
    - Централизованное хранение логов
    - Долгосрочное хранение
    - Ротация
    - Анализ информации
    - Поиск записей
    - Генерация отчетов
  - Интеграция с различными системами мониторинга (DataDog, Instana, ...)



# Фреймворки для логирования в Java

- Java Logging API (пакет `java.util.logging`)
- Log4J
- Logback
- SLF4J
- ...

# Java Logging API

- Основные компоненты:
  - **Logger'ы**: ответственны за создание сообщений:
    - Текст сообщения
    - Уровень логирования (отладка, предупреждение, ошибка, ...)
    - Дополнительные параметры
  - **Handler'ы**: определяют, куда будет записано сообщение:
    - Вывод в консоль
    - Запись в файл или базу данных
    - Отправление сообщения по сети
  - **Formatter'ы**: определяют формат вывода:
    - XML, JSON, ...
    - форматирование даты и времени

# Logger

```
package ru.bfu.ipmit;

import java.util.logging.Level;
import java.util.logging.Logger;

public class Main {

    private final static String LOGGER_NAME = Main.class.getName(); // "ru.bfu.ipmit.Main"
    private final static Logger LOGGER = Logger.getLogger(LOGGER_NAME);

    public static void main(String[] args) {
        // информация для отладки
        LOGGER.log(Level.INFO, "Начало выполнения программы");
        // предупреждение
        LOGGER.log(Level.WARNING, "Возраст пользователя должен был больше 18 лет. Отклоняем запрос.");
        // критическая ошибка
        LOGGER.log(Level.SEVERE, "Невозможно установить соединение с сервером");
    }
}
```

```
Oct 04, 2021 4:53:40 PM ru.bfu.ipmit.Main main
INFO: Начало выполнения программы
Oct 04, 2021 4:53:40 PM ru.bfu.ipmit.Main main
WARNING: Возраст пользователя должен был больше 18 лет. Отклоняем запрос.
Oct 04, 2021 4:53:40 PM ru.bfu.ipmit.Main main
SEVERE: Невозможно установить соединение с сервером
```

# Уровни логирования

```
// среда разработки, в которой запущена программа
enum Environment {
    DEVELOPMENT, // среда для разработки
    TEST,        // среда для тестировщиков
    UAT,         // среда для пользовательского тестирования
    PRODUCTION   // продуктивная среда
}

// класс для создания и конфигурации логгера
class LoggerFactory {
    private final Environment environment;

    LoggerFactory(Environment environment) { this.environment = environment; }

    public Logger getLogger(String loggerName) {
        Logger logger = Logger.getLogger(loggerName);
        // установка уровня логирования
        logger.setLevel(getLoggingLevel());
        return logger;
    }

    private Level getLoggingLevel() {
        if (Environment.PRODUCTION.equals(environment)) {
            return Level.SEVERE; // запись только критических ошибок в лог
        }
        if (Environment.UAT.equals(environment)) {
            return Level.WARNING; // запись предупреждений и критических ошибок в лог
        }

        return Level.INFO; // запись сообщений для отладки, предупреждений и критических ошибок в лог
    }
}
```

**Внимание!** Жесткое кодирование сред разработки в коде является плохим стилем программирования.

# Обработчик сообщения (Handler)

- Примеры:
  - `ConsoleHandler` - обработчик **по умолчанию**, записывает сообщения в поток `System.err`
  - `FileHandler` - записывает сообщения в файл
  - `SocketHandler` - отправляет сообщение по сети с помощью TCP-сокета
  - `MemoryHandler` - записывает сообщения в память

# FileHandler

```
package ru.bfu.ipmit;

import java.io.IOException;
import java.util.logging.FileHandler;
import java.util.logging.Level;
import java.util.logging.Logger;

public class Main {

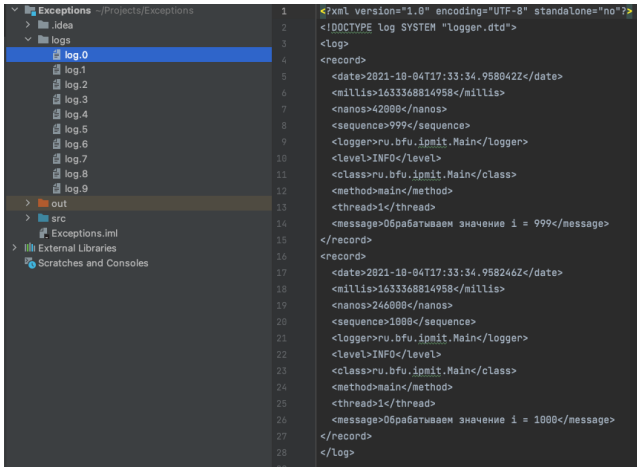
    private final static String LOGGER_NAME = Main.class.getName(); // "ru.bfu.ipmit.Main"
    private final static Logger LOGGER = Logger.getLogger(LOGGER_NAME);

    public static void main(String[] args) throws IOException {
        final String location = "logs/log"; // место нахождения файла относительно корневой директории проекта
        final int limit = 1000; // максимальный размер лог-файла в байтах
        final int count = 10; // максимальное количество лог-файлов
        final boolean isAppendMode = true; // режим добавления сообщений в существующие файлы

        final FileHandler fileHandler = new FileHandler(location, limit, count, isAppendMode);
        LOGGER.addHandler(fileHandler);

        for (int i = 1; i <= 1000; ++i) {
            LOGGER.log(Level.INFO, "Обрабатываем значение i = " + i);
        }
    }
}
```

# FileHandler

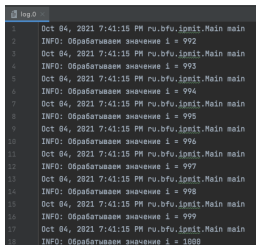


```
1 <?xml version="1.0" encoding="UTF-8" standalone="no"?>
2 <!DOCTYPE log SYSTEM "logger.dtd">
3 <log>
4 <record>
5   <date>2021-10-04T17:33:34.958042Z</date>
6   <millis>1633368814958</millis>
7   <nanos>42000</nanos>
8   <sequence>999</sequence>
9   <logger>ru.bfu.ipmit.Main</logger>
10  <level>INFO</level>
11  <class>ru.bfu.ipmit.Main</class>
12  <method>main</method>
13  <thread>1</thread>
14  <message>Обработка значения i = 999</message>
15 </record>
16 <record>
17   <date>2021-10-04T17:33:34.958246Z</date>
18   <millis>1633368814958</millis>
19   <nanos>246000</nanos>
20   <sequence>1000</sequence>
21   <logger>ru.bfu.ipmit.Main</logger>
22   <level>INFO</level>
23   <class>ru.bfu.ipmit.Main</class>
24   <method>main</method>
25   <thread>1</thread>
26   <message>Обработка значения i = 1000</message>
27 </record>
28 </log>
```

# Formatter

- SimpleFormatter (по умолчанию для ConsoleHandler)
- XMLFormatter (по умолчанию для FileHandler)

```
final FileHandler fileHandler = new FileHandler(location, limit, count, isAppendMode);
fileHandler.setFormatter(new SimpleFormatter());
LOGGER.addHandler(fileHandler);
```



```
1 Oct 04, 2021 7:41:15 PM ru.bfu.ipmit.Main main
2 INFO: Обрабатываем значение i = 992
3 Oct 04, 2021 7:41:15 PM ru.bfu.ipmit.Main main
4 INFO: Обрабатываем значение i = 993
5 Oct 04, 2021 7:41:15 PM ru.bfu.ipmit.Main main
6 INFO: Обрабатываем значение i = 994
7 Oct 04, 2021 7:41:15 PM ru.bfu.ipmit.Main main
8 INFO: Обрабатываем значение i = 995
9 Oct 04, 2021 7:41:15 PM ru.bfu.ipmit.Main main
10 INFO: Обрабатываем значение i = 996
11 Oct 04, 2021 7:41:15 PM ru.bfu.ipmit.Main main
12 INFO: Обрабатываем значение i = 997
13 Oct 04, 2021 7:41:15 PM ru.bfu.ipmit.Main main
14 INFO: Обрабатываем значение i = 998
15 Oct 04, 2021 7:41:15 PM ru.bfu.ipmit.Main main
16 INFO: Обрабатываем значение i = 999
17 Oct 04, 2021 7:41:15 PM ru.bfu.ipmit.Main main
18 INFO: Обрабатываем значение i = 1000
```