



HIGHER SCHOOL OF ECONOMICS
NATIONAL RESEARCH UNIVERSITY

Лекция 9

Обработка исключений

Программирование на языке Java

Роман Гуров

ВШЭ БИ 2021





Пример иерархии

Рассмотрим пример иерархии классов, реализующей геометрические фигуры

```
public class Point {  
    public Point(double x, double y) {  
        this.x = x;  
        this.y = y;  
    }  
  
    public double getX() { return x; }  
  
    public double getY() { return y; }  
  
    @Override  
    public String toString() { return "(" + x + ", " + y + ")"; }  
  
    private final double x;  
    private final double y;  
}
```

Пример иерархии

```
public enum Color {  
    BLACK,  
    WHITE,  
    RED,  
    GREEN,  
    BLUE  
}
```

```
public abstract class Shape {  
    public Shape(Color color) { this.color = color; }  
  
    public Color getColor() { return color; }  
  
    public abstract double getArea();  
  
    private final Color color;  
}
```

```
public class Circle extends Shape {  
    private final Point center;  
    private final double radius;  
  
    public Circle(Point center, double radius, Color color) {  
        super(color);  
        this.center = center;  
        this.radius = radius;  
    }  
  
    public Point getCenter() { return center; }  
  
    public double getRadius() { return radius; }  
  
    @Override  
    public double getArea() { return radius * radius * Math.PI; }  
}
```

```
public class Triangle extends Shape {  
    private final Point a;  
    private final Point b;  
    private final Point c;  
  
    public Triangle(Point a, Point b, Point c, Color color) {  
        super(color);  
        this.a = a;  
        this.b = b;  
        this.c = c;  
    }  
  
    @Override  
    public double getArea() {  
        return Math.abs((a.getX() - c.getX()) * (b.getY() - c.getY())  
            - (b.getX() - c.getX()) * (a.getY() - c.getY())) / 2;  
    }  
}
```

```
public class Square extends Shape {  
    private final Point center;  
    private final double size;  
  
    public Square(Point corner, double size, Color color) {  
        super(color);  
        this.center = corner;  
        this.size = size;  
    }  
  
    public Point getCenter() { return center; }  
  
    public double getSize() { return size; }  
  
    @Override  
    public double getArea() { return size * size; }  
}
```

Пример иерархии

```
public static void main(String[] args) {
    Circle circle = new Circle(
        new Point(0, 0), 1, Color.BLACK);

    Triangle triangle = new Triangle(
        new Point(0, 0), new Point(1, 0), new Point(0, 1), Color.RED);

    Square square = new Square(
        new Point(5, 5), 2, Color.BLUE);

    Shape shape = triangle;
    Object object = triangle;
    triangle = (Triangle) object;

    Shape[] shapes = {circle, triangle, square};

    Shape maxShape = findShapeWithMaxArea(shapes);
    System.out.println("Shape with max area: " + maxShape);
}

private static Shape findShapeWithMaxArea(Shape[] shapes) {
    Shape maxShape = null;
    double maxArea = Double.NEGATIVE_INFINITY;
    for (Shape shape: shapes) {
        double area = shape.getArea();
        if (area > maxArea) {
            maxArea = area;
            maxShape = shape;
        }
    }
    return maxShape;
}
```

Как обрабатывать ошибки?

Рассмотрим пример: нужно прочитать текст из файла и вывести его на экран

Условно, решение может выглядеть вот так:

```
public static void main(String[] args) {  
    String file_text = readFile("texts/testfile.txt");  
    System.out.println(file_text);  
}
```

А что делать, если файла с таким именем не существует?

Что в таком случае должна делать функция readFile?

```
public static String readFile(String file_path) {  
    if (!fileExists(file_path)) {  
        // Ошибка! А что делать то?!  
    }  
    // ...  
    return file.read();  
}
```



Как обрабатывать ошибки?

Попытаемся разрулить ситуацию самостоятельно

Давайте возвращать пустой результат при ошибке:

```
public static String readFile(String file_path) {  
    if (!fileExists(file_path)) {  
        return "";  
    }  
    // ...  
    return file.read();  
}
```

Снаружи будет невозможно отличить, ошибка это, или просто пустой файл

Тогда, воспользуемся особенностью ссылочных переменных и вернём null

Уже лучше, но что если файл существует и мы столкнулись с ошибкой доступа?
Если тоже вернём null, то снаружи будет невозможно определить причину ошибки

Как обрабатывать ошибки?

Поступим как в языке C, возвращая из функции код ошибки, а результат будем принимать с помощью аргументов

```
public static int readFile(String file_path, StringBuilder result) {  
    if (!fileExists(file_path)) {  
        return 1;  
    }  
    // ...???  
    result.append(file.read());  
    return 0;  
}
```

Использование функции теперь выглядит как-то так:

```
public static void main(String[] args) {  
    StringBuilder file_text = new StringBuilder();  
    int error_code = readFile1("texts/testfile.txt", file_text);  
  
    if (error_code == 1) {  
        System.out.println("File not found");  
    }  
  
    System.out.println(file_text);  
}
```

А если в аргумент result передали null?

Как обрабатывать ошибки?

А если в аргумент `result` передали `null`?

Наш придуманный механизм обработки ошибок теперь тоже требует обработки ошибок:

```
public static int readFile(String file_path, StringBuilder result) {
    if (!fileExists(file_path)) {
        return 1;
    }
    if (result == null) {
        return 2;
    }
    // ...???
    result.append(file.read());
    return 0;
}
```

```
public static void main(String[] args) {
    StringBuilder file_text = new StringBuilder();
    int error_code = readFile1("texts/testfile.txt", file_text);

    if (error_code == 1) {
        System.out.println("File not found");
    } else if (error_code == 2) {
        System.out.println("null pointer error"); // ???
    } else if (error_code != 0) {
        System.out.println("Some unknown error with code " + error_code);
    }

    System.out.println(file_text);
}
```


Как обрабатывать ошибки?

Можем ограничить и конкретизировать виды ошибок с помощью enum:

```
enum ReadFileStatus {  
    Good,  
    FileNotFoundError,  
    NullPointerError  
}
```

```
public static ReadFileStatus readFile(String file_path, StringBuilder result) {  
    if (!fileExists(file_path)) {  
        return ReadFileStatus.FileNotFoundError;  
    }  
    if (result == null) {  
        return ReadFileStatus.NullPointerError;  
    }  
    // ...???  
    result.append(file.read());  
    return ReadFileStatus.Good;  
}
```

```
public static void main(String[] args) {  
    StringBuilder file_text = new StringBuilder();  
    ReadFileStatus error_code = readFile1("texts/testfile.txt", file_text);  
  
    if (error_code == ReadFileStatus.FileNotFoundError) {  
        System.out.println("File not found");  
    } else if (error_code == ReadFileStatus.NullPointerError) {  
        System.out.println("null pointer error"); // ???  
    }  
  
    System.out.println(file_text);  
}
```

Но это всё ужасно и неудобно.

Как правильно обрабатывать ошибки?

Изобретательство велосипедов не привело к хорошему результату
Никак не обойтись без нового волшебного механизма от самого языка программирования

В Java для этого существует механизм *исключений* (англ. *exception*)

```
public static void main(String[] args) {  
    StringBuilder builder = null;  
    builder.append("data");  
}
```

Исключение – событие, прерывающее стандартный ход исполнения программы
Его можно обработать и вернуть программу обратно в штатный режим исполнения

Инициация такого события называется *бросанием исключения*

Если исключение никак не обработано, то, по-умолчанию, его текст будет выведен в System.out, а сама программа экстренно завершит исполнение:

```
Exception in thread "main" java.lang.NullPointerException Create breakpoint : Cannot invoke "java.lang.StringBuilder.append(String)" because "builder" is null  
at ru.hse.lecture9.Mainer.main(Mainer.java:6)
```



Примеры исключений, встроенных в Java

`NullPointerException` – исключение, возникающее, когда значение `null` оказалось в неподходящем месте
Например, при обращении к методу или полю объекта

```
public static void main(String[] args) {  
    StringBuilder builder = null;  
    builder.append("data");  
}
```

```
Exception in thread "main" java.lang.NullPointerException: Cannot invoke "java.lang.StringBuilder.append(String)" because "builder" is null  
at ru.hse.lecture9.Mainer.main(Mainer.java:6)
```

Примеры исключений, встроенных в Java

`ArrayIndexOutOfBoundsException` – обращение за пределы границ массива

```
public static void main(String[] args) {  
    int[] array = new int[100];  
    array[1000] = 30;  
}
```

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Index -1000 out of bounds for length 100  
    at ru.hse.lecture9.Mainer.main(Mainer.java:6)
```

```
public static void main(String[] args) {  
    ArrayList<Integer> arr = new ArrayList<>();  
    arr.get(0);  
}
```

```
Exception in thread "main" java.lang.IndexOutOfBoundsException: Index 0 out of bounds for length 0  
    at java.base/java.util.Objects.checkIndex(Objects.java:359)  
    at java.base/java.util.ArrayList.get(ArrayList.java:427)  
    at ru.hse.lecture9.Mainer.main(Mainer.java:8)
```



Примеры исключений, встроенных в Java

`StringIndexOutOfBoundsException` – обращение за пределы границ строки

```
public static void main(String[] args) {  
    String str = "Short";  
    str.charAt(-100);  
}
```

```
Exception in thread "main" java.lang.StringIndexOutOfBoundsException: String index out of range: -100  
    at java.base/java.lang.StringLatin1.charAt(StringLatin1.java:48)  
    at java.base/java.lang.String.charAt(String.java:1512)  
    at ru.hse.lecture9.Mainer.main(Mainer.java:6)
```

Вместе с исключением также выводится и полный стек вызовов до места его появления

Примеры исключений, встроенных в Java

`NoSuchFileException` – указанный файл не найден

```
public static void main(String[] args) throws java.io.IOException {  
    Path file_path = Paths.get("texts/testfile.txt");  
    byte[] file_text_bytes = Files.readAllBytes(file_path);  
    String file_text = new String(file_text_bytes);  
    System.out.println(file_text);  
}
```

```
Exception in thread "main" java.nio.file.NoSuchFileException Create breakpoint : texts\testfile.txt  
    at java.base/sun.nio.fs.WindowsException.translateToIOException(WindowsException.java:85)  
    at java.base/sun.nio.fs.WindowsException.rethrowAsIOException(WindowsException.java:103)  
    at java.base/sun.nio.fs.WindowsException.rethrowAsIOException(WindowsException.java:108)  
    at java.base/sun.nio.fs.WindowsFileSystemsProvider.newByteChannel(WindowsFileSystemsProvider.java:236)  
    at java.base/java.nio.file.Files.newByteChannel(Files.java:380)  
    at java.base/java.nio.file.Files.newByteChannel(Files.java:432)  
    at java.base/java.nio.file.Files.readAllBytes(Files.java:3288)  
    at ru.hse.lecture9.Mainer.main(Mainer.java:11)
```

Примеры исключений, встроенных в Java

Исключения существуют и для ошибок, произошедших в JVM

`OutOfMemoryError` – виртуальная машина Java израсходовала всю выделенную её память

```
public static void main(String[] args) {  
    ArrayList<Double> arr = new ArrayList<>();  
    while (true) {  
        arr.add(3.14);  
    }  
}
```

```
Exception in thread "main" java.lang.OutOfMemoryError: Create breakpoint : Java heap space  
at java.base/java.util.Arrays.copyOf(Arrays.java:3512)  
at java.base/java.util.Arrays.copyOf(Arrays.java:3481)  
at java.base/java.util.ArrayList.grow(ArrayList.java:237)  
at java.base/java.util.ArrayList.grow(ArrayList.java:244)  
at java.base/java.util.ArrayList.add(ArrayList.java:454)  
at java.base/java.util.ArrayList.add(ArrayList.java:467)  
at ru.hse.lecture9.Mainer.main(Mainer.java:12)
```

Примеры исключений, встроенных в Java

Исключения существуют и для ошибок, связанных с JVM

`NoClassDefFoundError` – виртуальная машина Java не смогла найти запрашиваемый класс

Вспомним старый пример с библиотекой и забудем передать её при запуске:

```
import org.apache.commons.lang3.StringUtils;

public class Main {
    public static void main(String[] args) {
        System.out.println("Hello, world!");
        System.out.println(StringUtils.equals("java", "java")); // true
    }
}
```

```
C:\Users\user\IdeaProjects\external>javac -classpath commons-lang3-3.12.0.jar Main.java
```

```
C:\Users\user\IdeaProjects\external>java Main
```

```
Hello, world!
```

```
Exception in thread "main" java.lang.NoClassDefFoundError: org/apache/commons/lang3/StringUtils
    at Main.main(Main.java:6)
```

```
Caused by: java.lang.ClassNotFoundException: org.apache.commons.lang3.StringUtils
    at java.base/jdk.internal.loader.BuiltinClassLoader.loadClass(BuiltinClassLoader.java:641)
    at java.base/jdk.internal.loader.ClassLoaders$AppClassLoader.loadClass(ClassLoaders.java:188)
    at java.base/java.lang.ClassLoader.loadClass(ClassLoader.java:520)
    ... 1 more
```




java.lang.Throwable

Как и почти всё в языке Java, брошенные исключения являются объектами соответствующих классов

Все исключения обязательно наследуются от класса Throwable

Ключевое свойство экземпляров Throwable и его подклассов – возможность быть брошенными

Бросить исключение можно при помощи ключевого слова throw

```
public static void main(String[] args) {  
    throw new RuntimeException(  
        "This program is supposed to die."  
    );  
}
```

```
Exception in thread "main" java.lang.RuntimeException: Create breakpoint : This program is supposed to die.  
    at ru.hse.lecture9.Mainer.main(Mainer.java:10)
```

Важно заметить, что стек вызовов запоминается в момент создания объекта исключения, а не в момент его бросания

java.lang.Throwable

Класс Throwable обладает множеством полезных методов:

```
package java.lang;

public class Throwable {

    public String getMessage() { /* ... */ }

    public void printStackTrace() { /* ... */ }

    public StackTraceElement[] getStackTrace() { /* ... */ }

    public Throwable getCause() { /* ... */ }

    public Throwable[] getSuppressed() { /* ... */ }

    // ...
}
```

- `getMessage()` – возвращает то самое сообщение, которое было передано в конструктор исключения и используется для описания подробностей возникшей проблемы
- `printStackTrace()` – печатает на экран отформатированный стек вызовов для этого исключения
- `getStackTrace()` – возвращает стек вызовов в формате массива элементов, по которым можно проитерироваться и вручную обработать в коде



java.lang.Throwable

```
public Throwable getCause() { /* ... */ }  
public Throwable[] getSuppressed() { /* ... */ }
```

- `getCause()` – возвращает исключение, являющееся реальной причиной создания текущего исключения

Обычно причина задаётся при преобразовании одного исключения в другое, более собирательное

Например, для баз данных в Java принято бросать свой специальный подтип исключений – `SQLException`

Если база данных использует локальные файлы и словила при работе с ними `IOException`, то она все равно бросит наружу `SQLException`, указав объект реального исключения как `cause`

- `getSuppressed()` – возвращает массив исключений, которые были заглушены при броске данного

Допустим, мы открыли файл на флешке, чтобы записать в него данные, но во время записи флешку вытащили

Метод `write()` бросил `IOException`, в обработчике этого исключения мы пытаемся закрыть файл, чтобы благополучно завершить работу с ним

Но объекту открытого файла настолько плохо, что при попытке закрытия он бросает ещё одно исключение

В такой ситуации логично было бы считать первородное исключение главным, а остальные добавлять в его массив `suppressed` как заглушённые им



Виды исключений

Есть три основных группы исключений в Java:

- `java.lang.Error` – исключительные ситуации в JVM

```
public class Error extends Throwable {
```

Пытаться как-то обработать эти исключения обычно бесполезно, после них программа вряд ли способна нормально продолжать работу

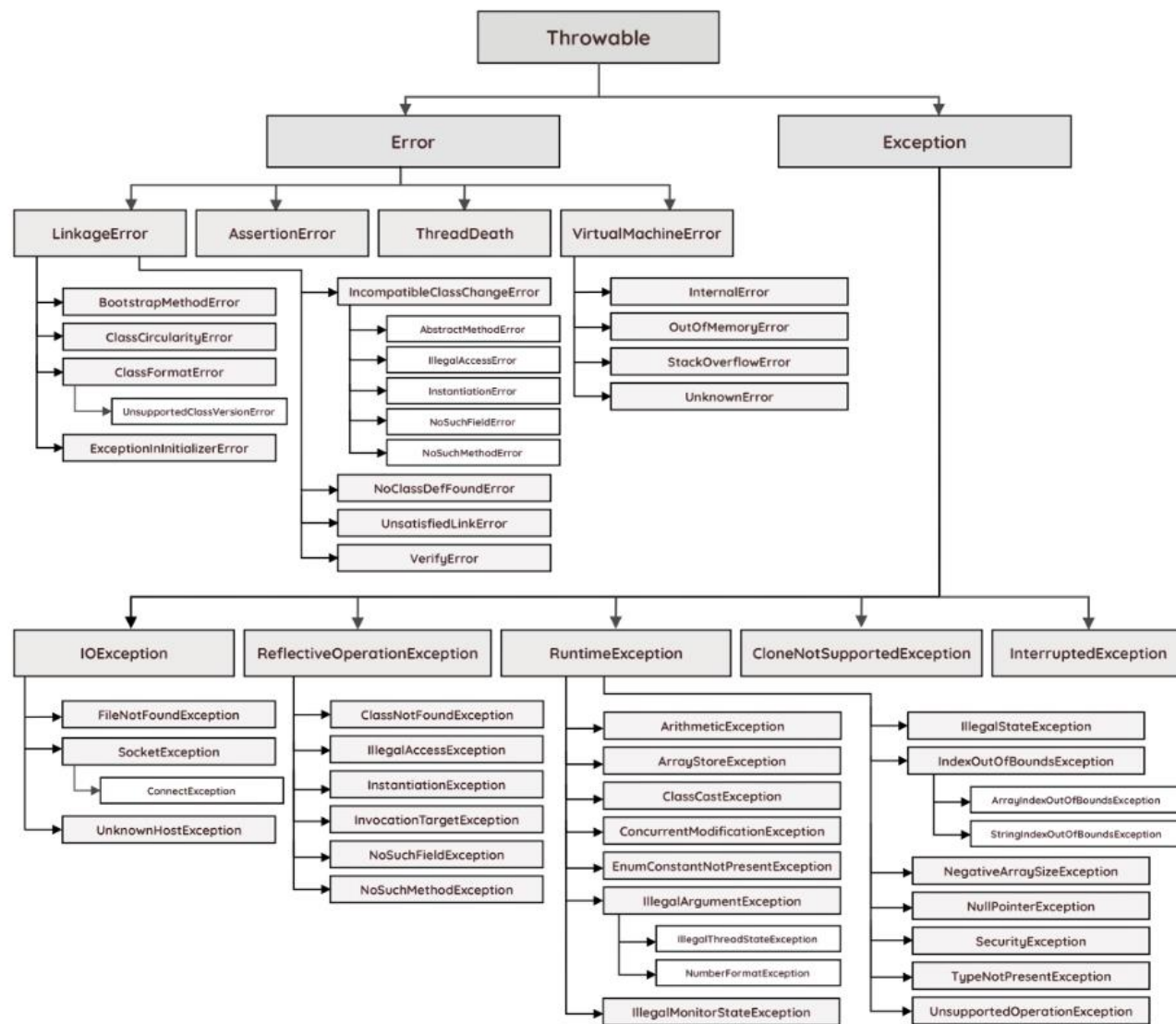
- Исключительные ситуации в пользовательском коде:
 - `java.lang.Exception` – проверяемые (checked)
 - `java.lang.RuntimeException` – непроверяемые (unchecked)

```
public class Exception extends Throwable {
```

```
public class RuntimeException extends Exception {
```

Обработка этих исключений уже, чаще всего, осмысленна

Виды исключений



Проверяемые исключения

Исключения-наследники `RuntimeException` являются *непроверяемыми* – их можно спокойно бросать отовсюду

Все остальные наследники `Exception` являются *проверяемыми* – компилятор внимательно следит за тем, чтобы эти исключения не остались незамеченными

```
public static void someFunction() {  
    throw new FileNotFoundException("I throw eheheheh!");  
}
```

При попытке бросить такое исключение, мы столкнёмся с ошибкой компиляции:

```
java: unreported exception java.io.FileNotFoundException; must be caught or declared to be thrown
```

Проверяемое исключение обязательно нужно либо обработать, либо явно указать ключевым словом `throws`, что наша функция умеет его бросать

```
public static void someFunction() throws IOException {  
    throw new FileNotFoundException("I throw eheheheh!");  
}
```

Точно такое же правило относится и к функциям, внутри которых вызываются бросающие функции:

```
public static void main(String[] args) throws IOException {  
    someFunction();  
}
```

Собственные исключения

Научимся создавать свои типы исключений

```
class Cat {  
    private float weight;  
    public static final float MAX_WEIGHT = 10.f;  
  
    public Cat(float weight) {  
        this.weight = weight;  
    }  
}
```

При создании кота может возникнуть ошибка

Обычно, в такой ситуации мы могли бы обойтись стандартными исключениями из Java, но создание своего собственного позволит явно отличать его от других

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

```
public Cat(float weight) {  
    if (weight > MAX_WEIGHT) {  
        throw new CatCreationException("The cat is too fat with weight: " + weight + "!");  
    } else if (weight < 0) {  
        throw new CatCreationException("The cat has negative weight!");  
    }  
    this.weight = weight;  
}
```

Обработка исключений

Как же всё-таки не крашить программу при бросании исключения?

Для обработки исключения в языке есть конструкция try-catch

```
public static void dangerousFunction() {  
    throw new IllegalArgumentException("I'll crash your program!");  
}  
public static void main(String[] args) {  
    try {  
        dangerousFunction();  
    } catch (RuntimeException ex) {  
        System.out.println("Dangerous function thrown an exception with text: " + ex.getMessage());  
    }  
}
```

В блоке try пишется код, способный бросить исключение

При броске, исключение летит наружу по стеку вызовов (*раскручивает стек вызовов*)

Если его никто не поймает, то оно вылетит за пределы точки входа и вызовет экстренное завершение

Чтобы поймать исключение, к блоку try добавляется блок catch, в котором указывается ловимый тип (или общий предок)

Внутри блока catch реализуется обработка пойманного исключения, также тут доступен его объект

Обработка исключений

Обработчиков может быть несколько:

```
public static void main(String[] args) {
    try {
        dangerousFunction();
    } catch (IllegalArgumentException ex) {
        System.out.println("I knew it! The function threw IllegalArgumentException");
    } catch (RuntimeException ex) {
        System.out.println("That's weird, it threw some other kind of RuntimeException");
    } catch (Exception ex) {
        System.out.println("That shouldn't happen, dangerousFunction doesn't specify checked exceptions using throws keyword");
    } catch (Throwable ex) {
        System.out.println("What is going on inside that function?! Does it try to break the JVM somehow?");
    }
    System.out.println("Congratulations! We survived the exception attack! But at what cost?");
}
```

В таком случае будет вызван *только первый* подходящий

Если исключение поймано, то исполнение кода продолжается после конструкции try-catch

Несколько типов в одном catch

Иногда для разных типов исключений нужна одинаковая обработка

```
try {
    dangerousFunction();
} catch (IllegalArgumentException ex) {
    System.out.println("Exception caught: " + ex.getMessage());
} catch (IllegalStateException ex) {
    System.out.println("Exception caught: " + ex.getMessage());
}
```

Вместо того, чтобы ловить общего предка вместе со всеми другими наследниками, можно просто перечислить оба типа в одном блоке catch

```
try {
    dangerousFunction();
} catch (IllegalArgumentException | IllegalStateException ex) {
    System.out.println("Exception caught: " + ex.getMessage());
}
```

Тогда будут пойманы только они, а переменная ex будет иметь тип их ближайшего общего предка

Повторное бросание

Из блока catch можно бросить новое исключение

```
try {
    try {
        throw new NullPointerException("inner");
    } catch (NullPointerException ex) {
        throw new IllegalStateException("outer", ex);
    }
} catch (RuntimeException ex) {
    ex.printStackTrace();
}
```

```
java.lang.IllegalStateException Create breakpoint : outer
    at ru.hse.lecture9.Mainer.main(Mainer.java:37)
Caused by: java.lang.NullPointerException Create breakpoint : inner
    at ru.hse.lecture9.Mainer.main(Mainer.java:35)
```

Также, можно повторно бросить и само пойманное исключение

```
try {
    try {
        throw new NullPointerException("inner");
    } catch (NullPointerException ex) {
        System.out.println("NullPointerException caught but rethrown");
        throw ex;
    }
} catch (RuntimeException ex) {
    ex.printStackTrace();
}
```

```
NullPointerException caught but rethrown
java.lang.NullPointerException Create breakpoint : inner
    at ru.hse.lecture9.Mainer.main(Mainer.java:35)
```



finally

Существует ещё один блок – finally

```
InputStream is = new FileInputStream("text.txt");
try {
    readTextFromStream(is);
} finally {
    is.close();
}
```

Он будет вызван последним *всегда*, несмотря на то, было исключение или нет, поймано оно или не поймано

Используется он для того, чтобы освободить взятый ресурс: открытый файл, интернет-соединение

Сам же блок не ловит исключений, необработанные исключения после выполнения этого блока полетят дальше

Проблема с finally


```
InputStream is = new FileInputStream("text.txt");
try {
    readTextFromStream(is);
} finally {
    is.close();
}
```

А если close сам бросит исключение? Тогда изначальное исключение “потеряется”

Можно попытаться поймать и заигнорить все побочные исключения

```
InputStream is = new FileInputStream("text.txt");
try {
    readTextFromStream(is);
} finally {
    try {
        is.close();
    } catch (IOException ex) {
        // do nothing
    }
}
```

Терять информацию – неправильно



try-with-resources

Для правильного закрытия ресурсов есть специальная конструкция – try-with-resources

```
try (InputStream is = new FileInputStream("text.txt")) {  
    readTextFromStream(is);  
} catch (IOException ex) {  
  
}
```

После слова try в скобках перечисляются через точку с запятой все ресурсы, которые используются в этом блоке

Гарантируется, что в после выполнения блока все ресурсы будут освобождены вызовом их метода close()

Если при брошенном исключении при освобождении ресурсов вылетело ещё одно, то оно будет проигнорировано и добавлено в массив suppressed у изначального

AutoCloseable

Для использования ресурса с конструкцией try-with-resources, он должен реализовывать интерфейс AutoCloseable:

```
public interface AutoCloseable {  
    void close() throws Exception;  
}
```

Например:

```
static class Resource implements AutoCloseable {  
    public void die() throws FileNotFoundException {  
        throw new FileNotFoundException();  
    }  
    @Override  
    public void close() throws IOException {  
        throw new IOException("closing failed");  
    }  
}  
  
public static void main(String[] args) throws IOException {  
    try (Resource res = new Resource()) {  
        res.die();  
    } catch (FileNotFoundException ex) {  
        ex.printStackTrace();  
    }  
}
```

```
java.io.FileNotFoundException Create breakpoint  
at ru.hse.lecture9.Resource.die(Mainer.java:35)  
at ru.hse.lecture9.Mainer.main(Mainer.java:48)  
Suppressed: java.io.IOException: closing failed  
at ru.hse.lecture9.Resource.close(Mainer.java:39)  
at ru.hse.lecture9.Mainer.main(Mainer.java:47)
```



Гарантии исключений

При реализации метода класса стоит задуматься о том, какие гарантии безопасности исключений он даёт

Сильные гарантии – при выбросе исключения из метода, объект останется в том же корректном состоянии, как если бы этот метод не был вызван вообще

Слабые гарантии – при выбросе исключения из метода, объект остаться в изменённом, но обязательно корректном состоянии

Если таких гарантий дать не удастся и объект всегда становится невалидным при бросании исключения, то можно гарантировать хотя бы отсутствие утечек ресурсов

В худшем случае, может не быть вообще никаких гарантий