Java Pro 3.2

TEL-RAN
by Starta Institute

# Исключения

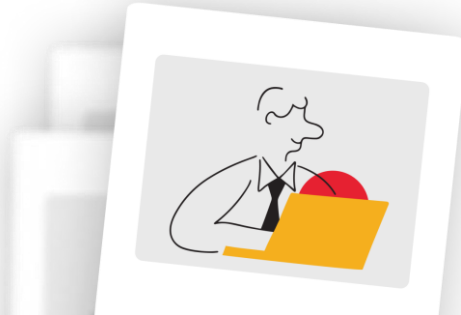
ПРЕПОДАВАТЕЛЬ

TEL-RAN
by Starta Institute

# Юрий Костяной

**Java/Kotlin backend-разработчик**

- 3+ года опыта в коммерческой разработке
- 2+ года опыта в преподавании
- Проекты по интеграции сторонних платформ, CRM
- Проблемно-ориентированный подход в преподавании



# ВАЖНО:

- Камера должна быть включена на протяжении всего занятия.
- Если у Вас возник вопрос в процессе занятия, пожалуйста, поднимите руку и дождитесь, пока преподаватель закончит мысль и спросит Вас, также можно задать вопрос в чате или когда преподаватель скажет, что начался блок вопросов.
- Организационные вопросы по обучению решаются с кураторами, а не на тематических занятиях.
- Вести себя уважительно и этично по отношению к остальным участникам занятия.
- Во время занятия будут интерактивные задания, будьте готовы включить камеру или демонстрацию экрана по просьбе преподавателя.

TEL-RAN
by Starta Institute

# ПЛАН ЗАНЯТИЯ

1. Основной блок
2. Вопросы по основному блоку
3. Домашняя работа

TEL-RAN
by Starta Institute

TEL-RAN
by Starta Institute

# 1 ПОВТОРЕНИЕ ИЗУЧЕННОГО

# Повторение

TEL-RAN
by Starta Institute

1 Что такое обобщённые типы?

2 Что такое стирание типов?

3 Если компилятор стирает все параметризированные типы в байт-коде, то для чего вообще использовать параметризованные типы?

# Повторение

TEL-RAN
by Starta Institute

1 Что такое обобщённые типы?

2 Что такое стирание типов?

3 Если компилятор стирает все параметризированные типы в байт-коде, то для чего вообще использовать параметризованные типы?

1 Обобщённые типы – это типы данных, которые применяются при создании классов и методов, при этом сам тип данных является переменным значением, вместо которого могут быть подставлены конкретные ссылочные типы данных (параметризация).

2 При выполнении программы параметры обобщённых типов заменяются на конкретные типы данных. А сами параметризованные типы полностью теряют информацию о типе параметра.

3 Для того, чтобы уже на стадии компиляции предотвращать появление ClassCastException и постоянные приведения типов.

# Повторение

Скомпилируется ли код ниже. Если нет, то почему?

```
public final class Algorithm {
    public static <T> T max(T x, T y) {
        return x > y ? x : y;
    }
}
```

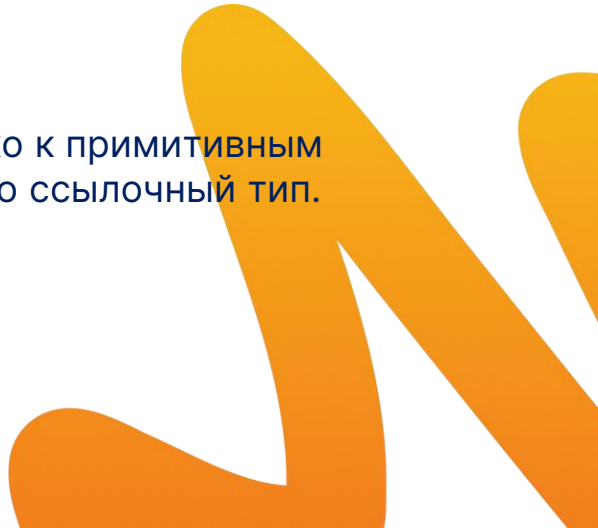
# Повторение

TEL-RAN
by Starta Institute

Скомпилируется ли код ниже. Если нет, то почему?

```
public final class Algorithm {
    public static <T> T max(T x, T y) {
        return x > y ? x : y;
    }
}
```

Код не скомпилируется, потому что оператор > применим только к примитивным числовым типам, а вместо обобщённого типа может быть только ссылочный тип.



# Повторение

TEL-RAN
by Starta Institute

На какие типы будут заменены K и V после стирания типов?

```
public class Pair<K, V> {

    public Pair(K key, V value) {
        this.key = key;
        this.value = value;
    }

    public K getKey() { return key; }
    public V getValue() { return value; }

    public void setKey(K key)     { this.key = key; }
    public void setValue(V value) { this.value = value; }

    private K key;
    private V value;
}
```

# Повторение

TEL-RAN
by Starta Institute

На какие типы будут заменены K и V после стирания типов?

На Object

```
public class Pair<K, V> {

    public Pair(K key, V value) {
        this.key = key;
        this.value = value;
    }

    public K getKey() { return key; }
    public V getValue() { return value; }

    public void setKey(K key)     { this.key = key; }
    public void setValue(V value) { this.value = value; }

    private K key;
    private V value;
}
```

# Повторение

TEL-RAN
by Starta Institute

Как класс будет выглядеть после стирания типов?

```
public class TypeErasureWithBound<T extends Iterable<T>> {
    private final T element;
    private final TypeErasureWithBound<T> previous;

    public TypeErasureWithBound(T element,
TypeErasureWithBound<T> previous) {
        this.element = element;
        this.previous = previous;
    }

    public T getElement() {
        return element;
    }
}
```

# Повторение

TEL-RAN
by Starta Institute

Как класс будет выглядеть после стирания типов?

```
public class TypeErasureWithBound<T extends Iterable<T>> {
    private final T element;
    private final TypeErasureWithBound<T> previous;

    public TypeErasureWithBound(T element,
TypeErasureWithBound<T> previous) {
        this.element = element;
        this.previous = previous;
    }

    public T getElement() {
        return element;
    }
}
```

```
public class TypeErasureWithBound {
    private final Iterable element;
    private final TypeErasureWithBound previous;

    public TypeErasureWithBound(Iterable element,
TypeErasureWithBound previous) {
        this.element = element;
        this.previous = previous;
    }

    public Iterable getElement() {
        return element;
    }
}
```

# Повторение

TEL-RAN
by Starta Institute

Скомпилируется ли код ниже. Если нет, то почему?

```
public static void print(List<? extends Number> list) {
    for (Number n : list)
        System.out.print(n + " ");
    System.out.println();
}
```

# Повторение

TEL-RAN
by Starta Institute

Скомпилируется ли код ниже. Если нет, то почему?

```
public static void print(List<? extends Number> list) {
    for (Number n : list)
        System.out.print(n + " ");
    System.out.println();
}
```

Да, скомпилируется

# Повторение

TEL-RAN
by Starta Institute

Скомпилируется ли код ниже. Если нет, то почему?

```
public static class Singleton<T> {

    public static T getInstance() {
        if (instance == null)
            instance = new Singleton<T>();

        return instance;
    }

    private static T instance = null;
}
```

# Повторение

TEL-RAN
by Starta Institute

Скомпилируется ли код ниже. Если нет, то почему?

```
public static class Singleton<T> {

    public static T getInstance() {
        if (instance == null)
            instance = new Singleton<T>();

        return instance;
    }

    private static T instance = null;
}
```

Не скомпилируется, потому что нельзя создавать статическое типизированное поле.

# Повторение

TEL-RAN
by Starta Institute

Даны следующие классы:

*class Shape {...}*

*class Circle extends Shape { ... }*

*class Rectangle extends Shape { ... }*

*class Node<T> { ... }*

Скомпилируется ли код ниже. Если нет, то почему?

*Node<Circle> nc = new Node<>();*

*Node<Shape>  ns = nc;*

# Повторение

TEL-RAN
by Starta Institute

Даны следующие классы:

*class Shape {...}*

*class Circle extends Shape { ... }*

*class Rectangle extends Shape { ... }*

*class Node<T> { ... }*

Скомпилируется ли код ниже. Если нет, то почему?

*Node<Circle> nc = new Node<>();*

*Node<Shape>  ns = nc;*

Не скомпилируется, потому что Node<Circle> не является подтипом Node<Shape> (типы инвариантны).

# Повторение

TEL-RAN
by Starta Institute

Дан класс:

```
class Node<T> implements Comparable<T> {
    public int compareTo(T obj) { ...}
    // ...
}
```

Скомпилируется ли код ниже. Если нет, то почему?

```
Node<String> node = new Node<>();
Comparable<String> comp = node;
```

# Повторение

TEL-RAN
by Starta Institute

Дан класс:

```
class Node<T> implements Comparable<T> {
    public int compareTo(T obj) { ...}
    // ...
}
```

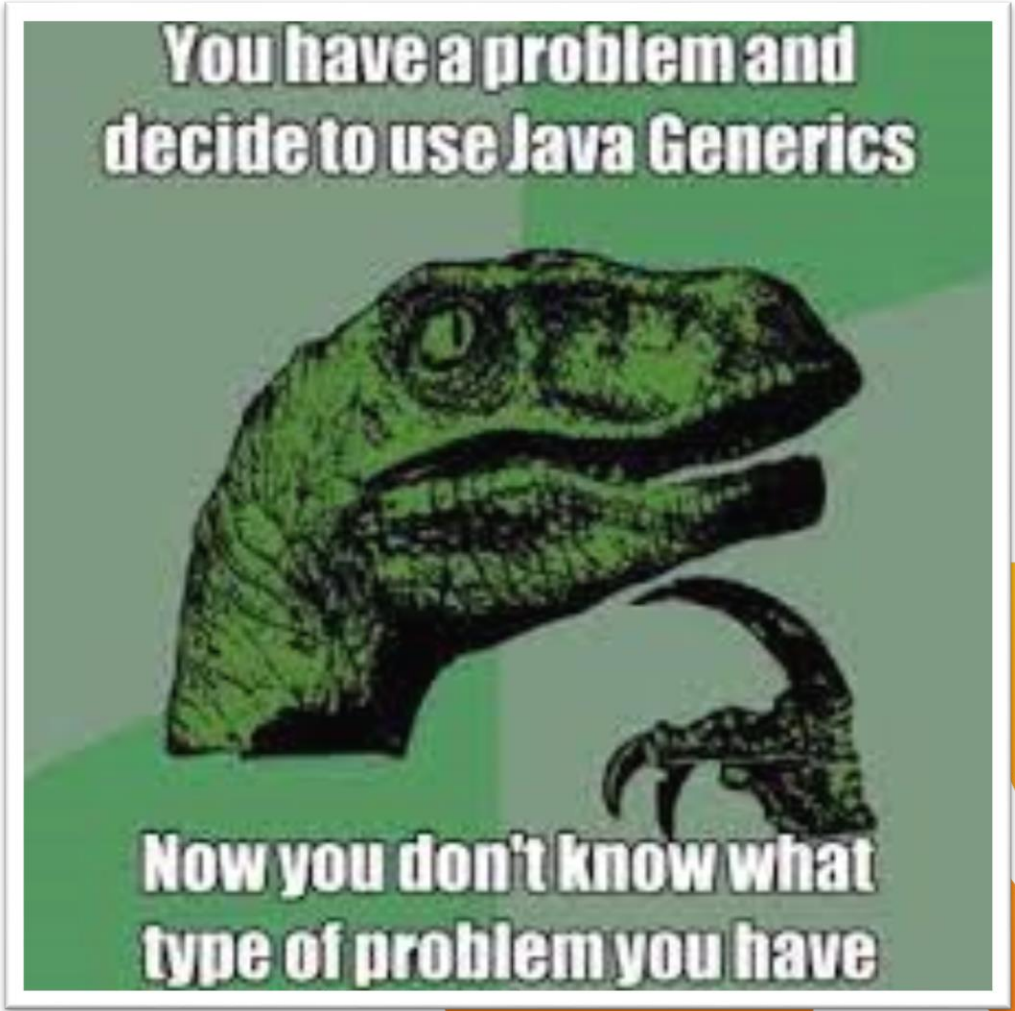
Скомпилируется ли код ниже. Если нет, то почему?

```
Node<String> node = new Node<>();
Comparable<String> comp = node;
```

Скомпилируется (типы ковариантны).

# Повторение

В чём прикол мема?



TEL-RAN
by Starta Institute

# 2 ОСНОВНОЙ БЛОК

# Введение

TEL-RAN
by Starta Institute

- Исключительные обстоятельства
- Династия бросаемых
- Поймать и обезвредить

# Проблема

TEL-RAN
by Starta Institute

Хорошо написанная программа сама по себе никогда не приводит к ошибкам. Но в реальном мире ошибки могут возникнуть в результате:

- неправильных действий пользователя
- отсутствии необходимого ресурса на диске или памяти
- потери соединения с сервером по сети
- неправильного использования API.

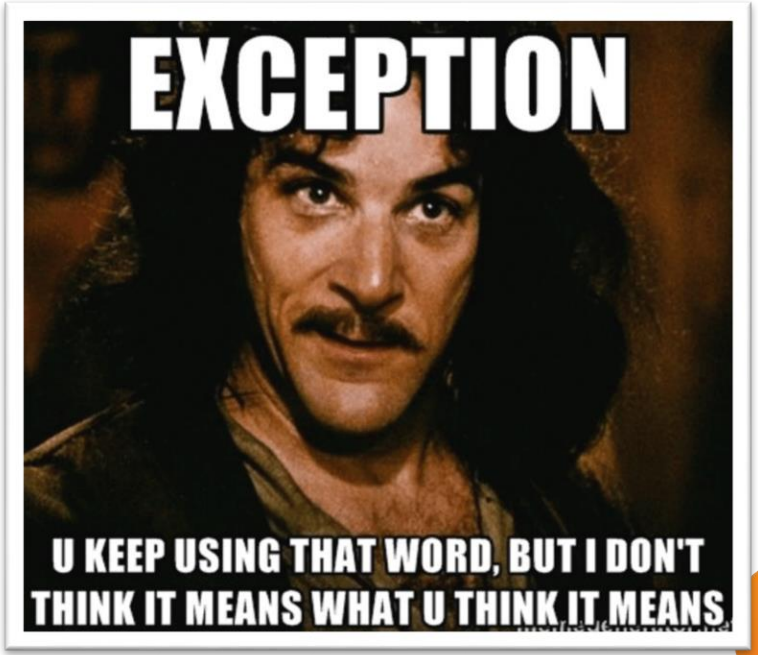
*Всё это исключительные ситуации. Когда программист понимает, что такая ситуация возможна, то необходимо предпринять какие-то действия в случае её возникновения.*

Исключительные обстоятельства

# Исключения

TEL-RAN
by Starta Institute

**Исключение (exception)** – возникновение ошибок и непредвиденных ситуаций при выполнении программы.



# Исключительные обстоятельства

# Пример исключения

TEL-RAN
by Starta Institute

```
public class Main {
    public static void main(String[] args) {
        System.out.println("First: " + divide( number: 15,  divisor: 3));
        System.out.println("Before second");
        System.out.println("Second: " + divide( number: 15,  divisor: 0));
        System.out.println("After second");
    }

    2 usages
    private static int divide(int number, int divisor) {
        return number/divisor;
    }
}
```

```
Main
"C:\Program Files\Java\jdk-17.0.5\bin\java.exe" "-javaagent:C:\Program Files\JetBr
First: 5
Before second
Exception in thread "main" java.lang.ArithmeticException Create breakpoint : / by zero
    at Main.divide(Main.java:10)
    at Main.main(Main.java:5)

Process finished with exit code 1
```

Выполнение прекратилось на этой строке, когда был вызван метод деления и в нём в качестве делителя выступал 0. Т.к. результатом выполнения метода должен быть int, а на 0 делить нельзя, то результатом выполнения стало выброшенное исключение ArithmeticException.
При выбрасывании исключения выполнения текущего метода/потока прерывается , поэтому последующий код программы не был выполнен

IDE показывает нам StackTrace, т.е. последовательность вызовов методов, для того чтобы мы увидели цепочку вызовов методов до места возникновения исключения. Цепочка вызовов записана в обратном порядке.

# Исключительные обстоятельства

# Задание

TEL-RAN
by Starta Institute

Напишите программу, чтобы узнать, какое исключение будет брошено, если:

1 Распознать строку “12c” в число.

2 Создать массив из отрицательного количества элементов.

3 Обратиться к элементу массива, которого нет в массиве.

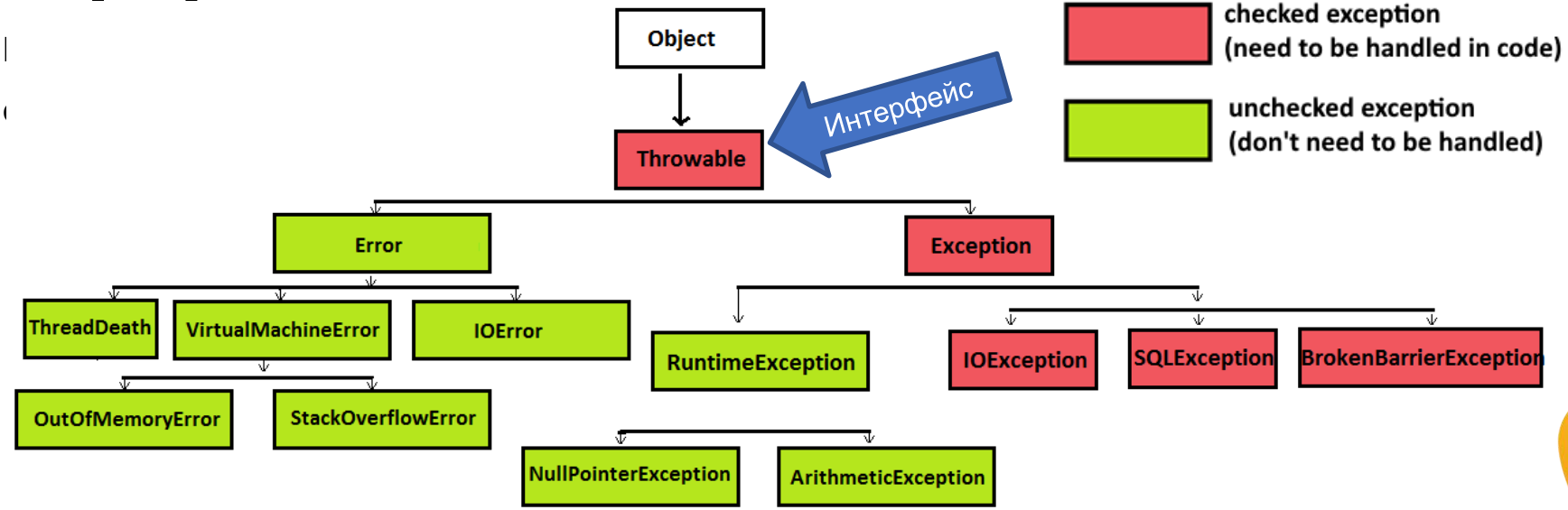
4 Использовать метод valueOf() у одного из Enum, передав в него пустую строку.

5 Создать строковую переменную, не присваивая ей значения, и вызвать у неё метод получения длины.

6 В отдельном методе создать Scanner и указать в нём путь к файлу вместо System.in.

Династия бросаемых

# Иерархия исключений




1 **Checked** – компилятор требует указания таких исключений в коде (наследники *Exception*).

2 **Unchecked** – как и *Checked*, бросаются во время выполнения программы, но компилятор не требует их указывать в коде (наследники *RuntimeException*).

3 **Error** – всё плохо. Ошибки появляются из-за проблем JVM или железа (например, не хватает оперативной памяти). Обработать/перехватить *Error* нельзя.

Династия бросаемых

# Иерархия исключений

TEL-RAN
by Starta Institute

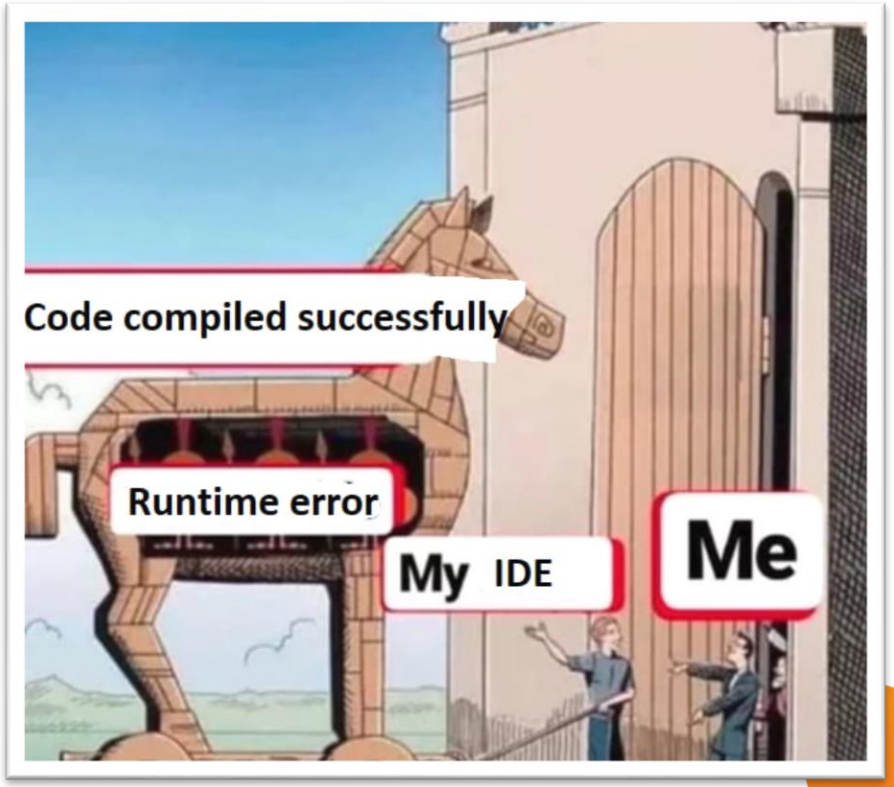
Ошибки выполнения более опасны для выполнения программы, т.к. чаще всего их трудно предсказать.

Династия бросаемых

# Иерархия исключений

TEL-RAN
by Starta Institute

Ошибки выполнения более опасны для выполнения программы, т.к. чаще всего их трудно предсказать.



Династия бросаемых

# Unchecked исключения

TEL-RAN
by Starta Institute

**ArithmeticException** – ошибка при выполнении арифметического вычисления.

**ArrayIndexOutOfBoundsException** – обращение к элементу массива, которого в массиве нет.

**ArrayStoreException** – ошибка сохранения в массиве объекта недопустимого типа.

**ClassCastException** – ошибка приведения типов.

**ConcurrentModificationException** – ошибка изменения объекта конкурирующим потоком вычислений (*thread*) или ошибка изменения коллекции во время прохода по коллекции.

**EmptyStackException** – ошибка извлечения объекта из пустого стека.

**IllegalArgumentException** – методу передано неверное значение

**IllegalMonitorStateException** – ошибка многопоточной программы при обращении к методу *wait*, *notifyAll* или *notify* объекта, когда текущий поток вычислений не обладает блокировкой (*lock*) этого объекта.

Династия бросаемых

# Unchecked исключения

TEL-RAN
by Starta Institute

**IllegalStateException** - ошибка выполнения операции в то время, когда объект не находится в соответствующем состоянии.

**IllegalThreadStateException** – ошибка выполнения операции, когда объект потока вычислений не находится в соответствующем состоянии (например, вызван метод *start* для потока, который уже приступил к работе).

**MissingResourceException** – не найден требуемый ресурс или пакет ресурсов.

**NegativeArraySizeException** – ошибка создания массива с отрицательным размером.

**NoSuchElementException** – ошибка поиска элемента в объекте одного из контейнерных классов (например, бросает итератор).

**NumberFormatException** – ошибка распознавания строки в число.

**SecurityException** – ошибка выполнения операции, запрещенной системой обеспечения безопасности в соответствии с действующей политикой безопасности.

Династия бросаемых

# Unchecked исключения

TEL-RAN
by Starta Institute

**StringIndexOutOfBoundsException** – ошибка обращения к символу строки по индексу, которого нет в строке.

**UnsupportedOperationException** – ошибка выполнения операции над объектом, который ее не поддерживает (например, модификация объекта, обозначенного признаком “только для чтения”).

**NullPointerException** – ошибка при попытке обращения к полю, методу или объекту по ссылке, равной *null*.

Всегда выполняйте проверку параметров методов ссылочного типа на null.

Династия бросаемых

# Checked исключения

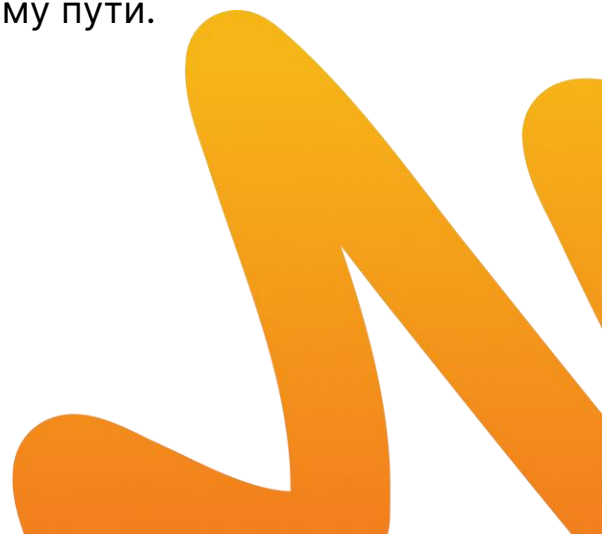
TEL-RAN
by Starta Institute

**ClassNotFoundException** – JVM не нашла класс при запуске программы.

**InterruptedException** – ошибка прерывания работы занятого потока.

**IOException** – ошибки потоков ввода-вывода. Например, когда при чтении файла теряется доступ к жёсткому диску.

**FileNotFoundException** – не найден файл для чтения по указанному пути.



Династия бросаемых

# Собственные исключения

TEL-RAN
by Starta Institute

Чтобы сигнализировать именно о той проблеме, которая важна для написанного Вами класса, лучше создавать собственные исключения.

Чтобы создать свой класс проверяемых исключений, достаточно расширить свой класс от класса *Exception* или одного из его потомков.
Для создания непроверяемого исключения, в своём классе нужно расширять класс *RuntimeException*.

ExceptionExaple.zip

# Задание

TEL-RAN
by Starta Institute

Создайте проект.

1 Создайте класс BankAccount для банковской программы, который хранит данные о балансе счёта и методы для снятия и пополнения счёта.

2 Создайте пакет exception. В пакете создайте два класса исключений:

- checked-исключение InsufficientFundsException (потомок Exception или его потомка).
- unchecked-исключение IncorrectAccountOperationException (потомок RuntimeException или его потомка).

3 Метод снятия бросает InsufficientFundsException, когда снимаемая сумма больше остатка по счёту. Метод пополнения бросает IncorrectAccountOperationException, когда переданная сумма меньше или равна 0.

Поймать и обезвредить

# Маркер исключения

TEL-RAN
by Starta Institute

Исключения можно *бросить*, *поймать* и *обработать*.

Если какой-то метод может бросить checked-исключения, то в его шапке добавляется ключевое слово **throws,** после которого указываются типы выбрасываемых исключений (типов может быть несколько). Так мы предупреждаем класс, который вызовет метод, что возможно исключение.

Метод бросает исключение, когда

1. Один из используемых в нём методов бросает исключение;
2. В коде метода явно бросается исключение с помощью ключевого слова **throw**. Например,

```
IllegalArgumentException e = new IllegalArgumentException();
throw e;
```

# Поймать и обезвредить
# **Бросаем и ловим**

```java
public class Main {
    public static void main(String[] args) throws Exception {
        System.out.println("First: " + divide( number: 15,  divisor: 3));
        System.out.println("Before second");
        System.out.println("Second: " + divide( number: 15,  divisor: 0));
        System.out.println("After second");
    }

    2 usages
    private static int divide(int number, int divisor) throws Exception {
        if (divisor == 0) {
            throw new Exception("Делитель не может быть равен 0");
        }
        return number/divisor;
    }
}
```

```
Main
"C:\Program Files\Java\jdk-17.0.5\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\Intel
First: 5
Before second
Exception in thread "main" java.lang.Exception Create breakpoint : Делитель не может быть равен 0
    at Main.divide(Main.java:12)
    at Main.main(Main.java:6)

Process finished with exit code 1
```

Если внутри метода вызывается другой метод, который может бросить *проверяемое* исключение, то его либо нужно поймать в текущем методе, либо указать в throws вызывающего метода.

Если в методе может быть брошено *проверяемое* исключение, то его либо нужно поймать, либо указать с помощью throws, что метод выбрасывает такой вид исключений в место вызова метода.

Здесь явно проверяется, что делитель не равен 0. В случае, если равен, то с помощью оператора *throw* мы бросаем исключение. Исключение – это класс, поэтому его объекты создаются с помощью оператора new.

Поймать и обезвредить

# Ловим и обрабатываем

TEL-RAN
by Starta Institute

Чтобы поймать исключение, используют конструкцию *try-catch-finally*.

Обработка исключений:

**try** – определяет блок кода, в котором может произойти исключение;

**catch** – определяет блок кода, в котором происходит обработка исключения. Необязательный блок;

**finally** – определяет блок кода, который выполняется в любом случае, независимо от результатов выполнения блока *try*. Необязательный блок.

Поймать и обезвредить

# Ловим и обрабатываем

TEL-RAN
by Starta Institute

Эти ключевые слова используются для создания в программном коде специальных обрабатывающих конструкций:

*try{} catch{}* – поймать и обработать или проигнорировать исключение

*try{} catch{} finally{}* – поймать и обработать исключение, кроме того вне зависимости от появления исключения выполнить ряд действий, например, закрыть доступ к занятым ресурсам (файлам, потокам).

*try{} finally{}* - поймать исключение, проигнорировать обработку и выполнить обязательные шаги после.

# Поймать и обезвредить

# **Ловим и обрабатываем**

TEL-RAN
by Starta Institute

```java
public class Main {
    public static void main(String[] args) throws Exception {
        System.out.println("First: " + divide( number: 15,  divisor: 3));
        System.out.println("Before second");
        try {
            System.out.println("Second: " + divide( number: 15,  divisor: 0));
        } catch (Exception e) {
            System.out.printf("We have an exception: type = %s, message = %s\n", e.getCause(), e.getLocalizedMessage());
        }
        System.out.println("After second");
    }

    2 usages
    private static int divide(int number, int divisor) throws Exception {
        if (divisor == 0) {
            throw new Exception("Делитель не может быть равен 0");
        }
        return number/divisor;
    }
}
```

```
Main
"C:\Program Files\Java\jdk-17.0.5\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition
First: 5
Before second
We have an exception: type = null, message = Делитель не может быть равен 0
After second

Process finished with exit code 0
```

Код, в котором может возникнуть исключение, помещается в try.

Т.к. исключение было поймано и обработано в catch, программа продолжила своё выполнение, а не упала с ошибкой

# Поймать и обезвредить
# **Finally**

TEL-RAN
by Starta Institute

```
public class Main {
    public static void main(String[] args) {
        String path = "notes.txt";
        System.out.println("First time: " + readStringFromFile(path));
        System.out.println("Second time: " +
readStringFromFile_tryWithResources(path));
    }

    private static String readStringFromFile(String filePath) {
        FileReader fr = null;
        BufferedReader br = null;
        try {
            fr = new FileReader(filePath);
            br = new BufferedReader(fr);
            return br.readLine();
        } catch (IOException e) {
            throw new RuntimeException(e);
        } finally {
            closeSoftly(fr);
            closeSoftly(br);
        }
    }
```

```
    private static void closeSoftly(Closeable stream) {
        if (stream == null) {
            return;
        }
        try {
            stream.close();
        } catch (IOException e) {
            System.err.println("Невозможно закрыть поток
чтения");
        }
    }

    private static String
readStringFromFile_tryWithResources(String filePath) {
        try (FileReader fr = new FileReader(filePath);
             BufferedReader br = new BufferedReader(fr)) {
            return br.readLine();
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
    }
}
```

# Поймать и обезвредить
# **Finally**

TEL-RAN
by Starta Institute

```
public class Main {
    public static void main(String[] args) {
        String path = "notes.txt";
        System.out.println("First time: " + readStringFromFile(path));
        System.out.println("Second time: " +
readStringFromFile_tryWithResources(path));
    }

    private static String readStringFromFile(String filePath) {
        FileReader fr = null;
        BufferedReader br = null;
        try {
            fr = new FileReader(filePath);
            br = new BufferedReader(fr);
            return br.readLine();
        } catch (IOException e) {
            throw new RuntimeException(e);
        } finally {
            closeSoftly(fr);
            closeSoftly(br);
        }
    }
```

В блоке finally закрываются потоки чтения данных из файла вне зависимости от того, возникло ли исключение.

```
    private static void closeSoftly(Closeable stream) {
        if (stream == null) {
            return;
        }
        try {
            stream.close();
        } catch (IOException e) {
            System.err.println("Невозможно закрыть поток
чтения");
        }
    }

    private static String
readStringFromFile_tryWithResources(String filePath) {
        try (FileReader fr = new FileReader(filePath);
             BufferedReader br = new BufferedReader(fr)) {
            return br.readLine();
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
    }
}
```

Try-catch-finally можно вкладывать друг в друга. Вложенные структуры лучше выносить в отдельный метод.

Метод аналогичен readStringFromFile(), но использует конструкцию **try-with-resources**

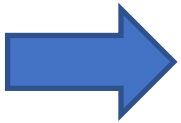
Поймать и обезвредить

# Инструкция return в блоке finally

TEL-RAN by Starta Institute

Вызов оператора *return* в блоке *finally* замещает аналогичный вызов в *catch*, а также скрывает возникшее исключение.

```
public static void main(String[] args) {
    System.out.print(getNumber());
}
private static int getNumber() {
    try {
        return 3;
    } finally {
        return 5;
    }
}
```



Результат:

5

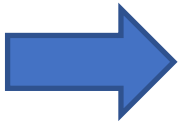
Поймать и обезвредить

# Инструкция return в блоке finally

TEL-RAN
by Starta Institute

Вызов оператора *return* в блоке *finally* замещает аналогичный вызов в *catch*, а также скрывает возникшее исключение.

```
public static void main(String[] args) {
    System.out.print(getNumber());
}
private static int getNumber() {
    try {
        throw new Error();
    } finally {
        return 5;
    }
}
```



Результат:

5

Поймать и обезвредить

# Порядок catch при исключении

TEL-RAN by Starta Institute

Иерархия наследования исключений с помощью полиморфизма позволяет задавать порядок обработки выброшенных исключений.

Исключения должны отлавливаться в порядке от наследника к предку!

```
import java.io.*;

public class Main {
    public static void main(String[] args) {
        String path = "notes.txt";
        System.out.println(readStringFromFile_tryWithResources(path));
    }

    private static String readStringFromFile_tryWithResources(String filePath) {
        try (FileReader fr = new FileReader(filePath);
             BufferedReader br = new BufferedReader(fr)) {
            return br.readLine();
        } catch (FileNotFoundException e) {
            System.err.println("Файл не найден: " + e.getLocalizedMessage());
        } catch (IOException e) {
            System.err.println("Ошибка чтения файла: " + e.getLocalizedMessage());
        } catch (Exception e) {
            System.err.println("Исключение: " + e.getLocalizedMessage());
        }
        return null;
    }
}
```

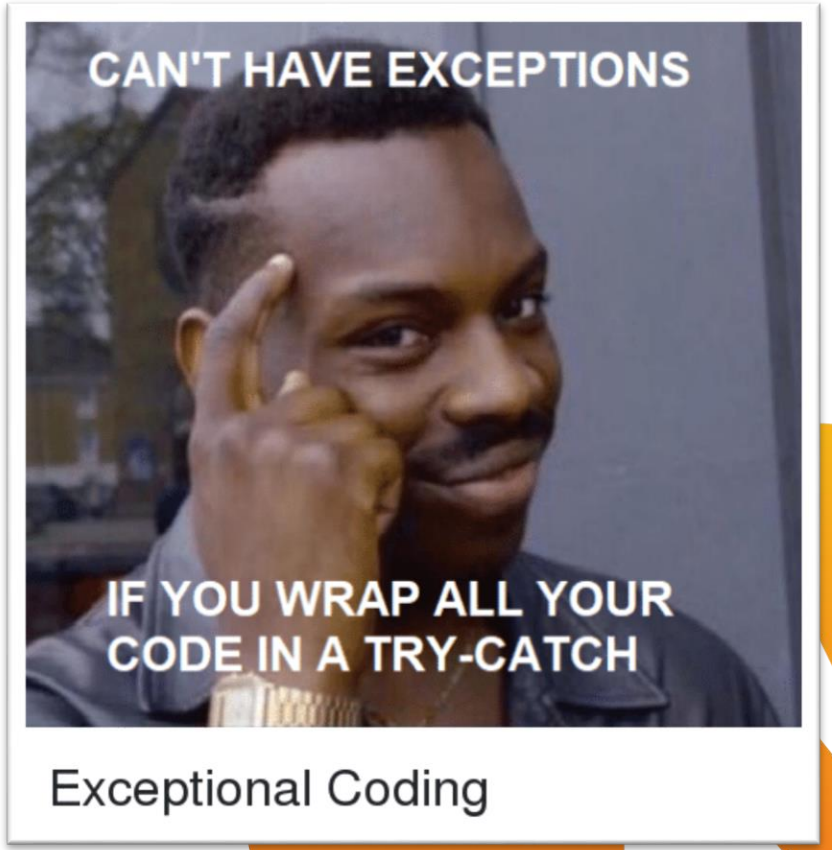
Поймать и обезвредить

# Не злоупотребляй исключениями

TEL-RAN
by Starta Institute

Часто исключения можно избежать обычной проверкой в коде.

Исключения должны отлавливаться как можно ближе к тому месту, где они выброшены, т.к. процедура сбора *StackTrace* является дорогостоящей с точки зрения ресурсов, потому что данные собираются со всех фреймов стека вызовов.



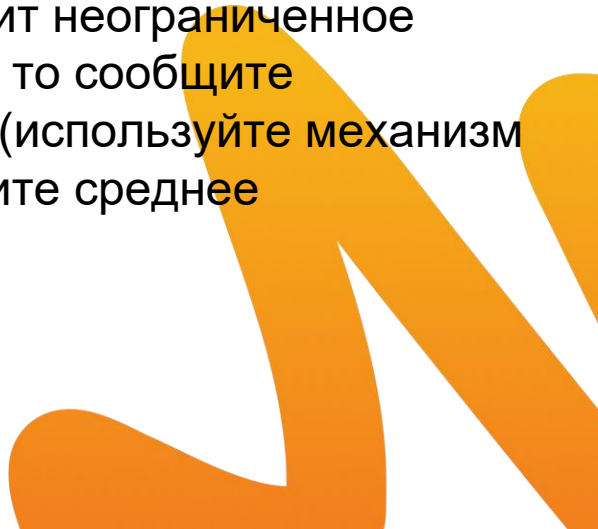
Кто крайний?

# Задание

TEL-RAN
by Starta Institute

1 В созданном ранее проекте вызовите методы класса BankAccount. Отловите исключения в едином блоке try-catch. В блоке catch выведите тип и сообщение пойманного исключения. В блоке finally выведите сообщение о закрытии транзакции.

2 Создайте программу, в которой пользователь вводит неограниченное количество чисел. Если пользователь ввёл не число, то сообщите пользователю, что значение будет проигнорировано (используйте механизм исключений). Когда пользователь введёт quit, выведите среднее арифметическое введённых чисел.



TEL-RAN
by Starta Institute

# 4 Домашнее задание

# Домашнее задание

TEL-RAN
by Starta Institute

1 Напишите класс CommandLineParser. Парсер должен принять аргументы командной строки, переданных в программу и превратить их в экземпляр класса настройки некой сортировки по следующим правилам:

a. режим сортировки (-a или -d), необязательный, по умолчанию сортируем по возрастанию;

b. тип данных (-s или -i), обязательный;

c. имя выходного файла, обязательное;

d. остальные параметры – имена входных файлов, не менее одного.

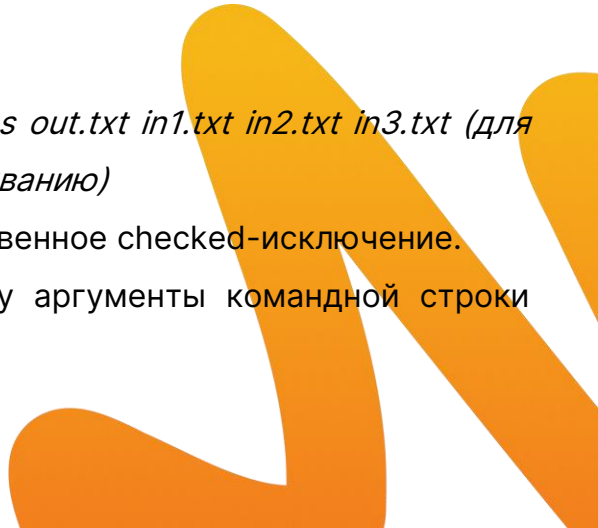
Примеры запуска из командной строки для Windows:

*sort-it.exe -i -a out.txt in.txt (для целых чисел по возрастанию) sort-it.exe -s out.txt in1.txt in2.txt in3.txt (для строк по возрастанию) sort-it.exe -d -s out.txt in1.txt in2.txt (для строк по убыванию)*

2 Если переданы неправильные данные, то парсер должен выбросить собственное checked-исключение.

3 В основной программе создайте экземпляр парсера и передайте ему аргументы командной строки запуска программы. Обработайте необходимые исключения.

4 Сохраните программу – в будущем она нам ещё пригодится.



# Полезные ссылки

TEL-RAN
by Starta Institute

Lesson: Exceptions (The Java™ Tutorials > Essential Java Classes) (oracle.com)

https://docs.oracle.com/javase/7/docs/api/java/lang/Exception.html

# ЗАКЛЮЧЕНИЕ

TEL-RAN
by Starta Institute