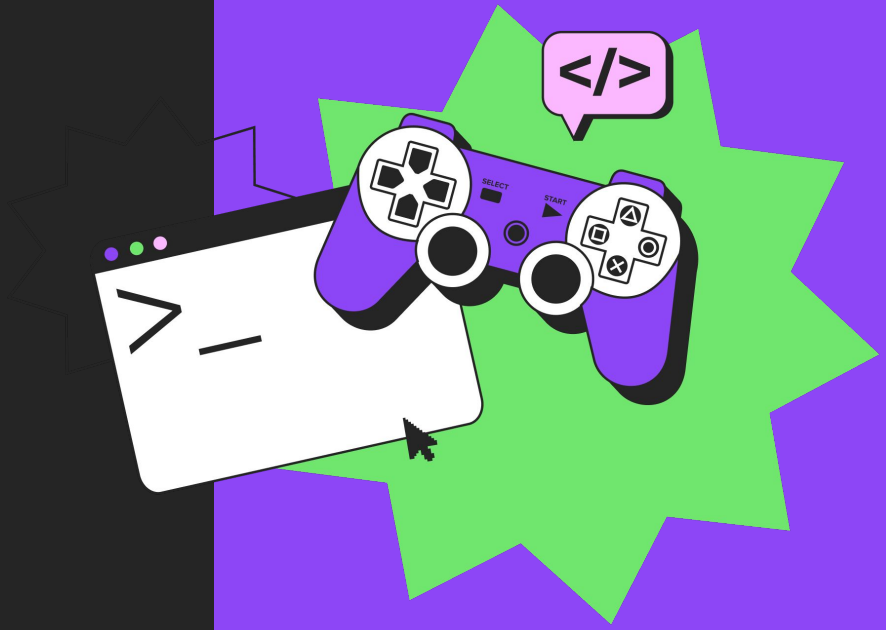
GeekBrains

Исключения и их обработка в Java

Урок 2

Типы и иерархия исключений в Java, исключения и ошибки и их обработка

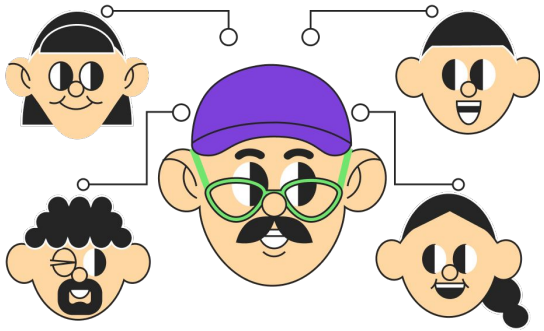


План курса

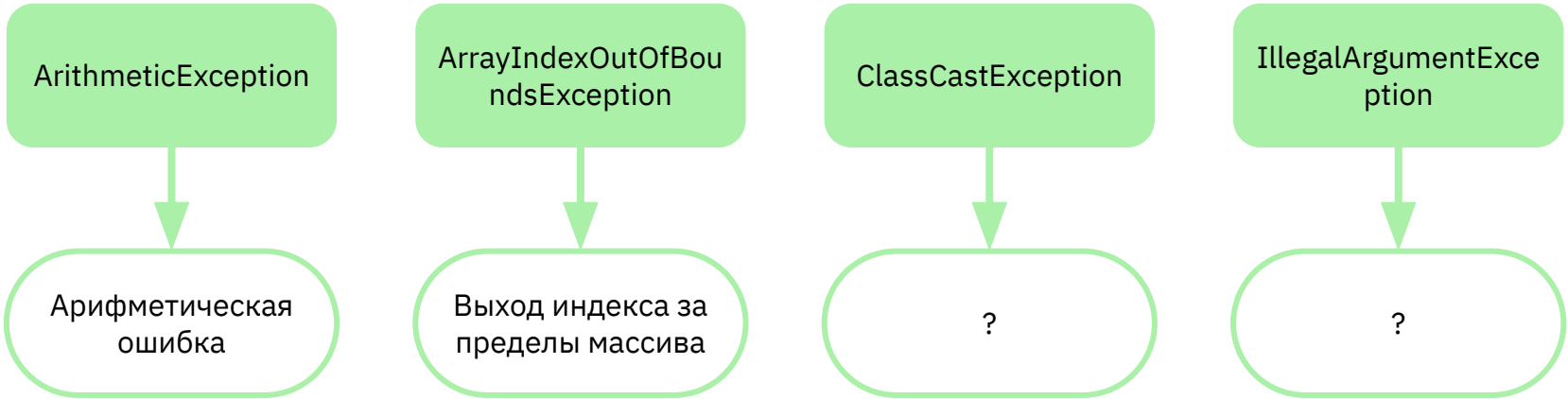
1. Обработка ошибок в программировании
2. Исключения и их обработка в Java
3. Продвинутая работа с исключениями в Java

План урока

- 📌 Разберём виды исключений в Java, какую иерархию они образуют и в чём её смысл
- 📌 Узнаем, чем исключения отличаются от ошибок
- 📌 Поймём, как не дать исключениям «ронять» приложение
- 📌 Узнаем, почему некоторые исключения компилятор заставляет обрабатывать, а некоторые нет



Типы исключений и их описание



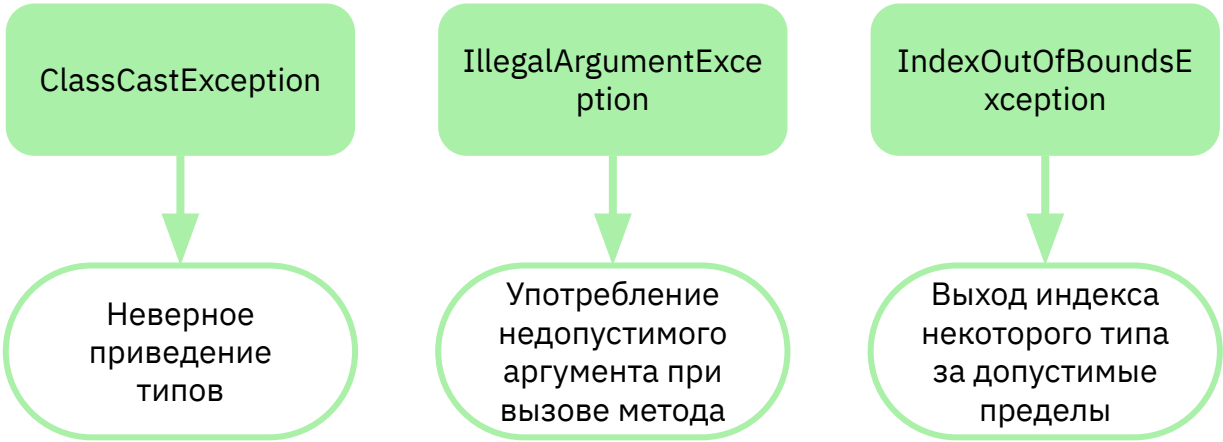
Попробуйте угадать, на какие ошибки могут указывать следующие исключения

Напишите свои ответы в обсуждениях под лекцией.
Время на размышление — 1 минута.

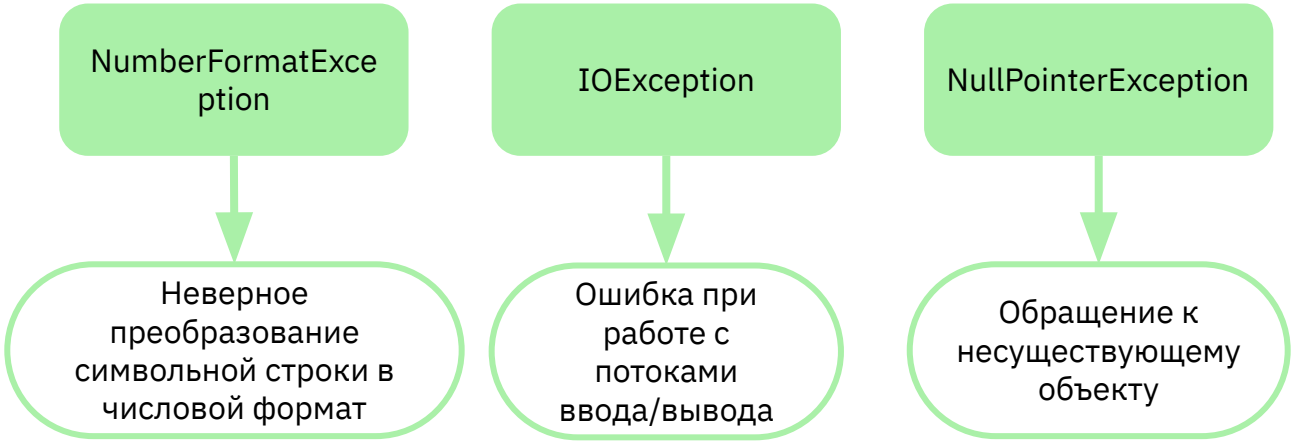
На какие ошибки могут указывать следующие исключения?

Тип исключения
ClassCastException
IllegalArgumentException
IndexOutOfBoundsException
NullPointerException
NumberFormatException
IOException
FileNotFoundException
ClassNotFoundException
UnsupportedOperationException

Типы исключений и их описание



Типы исключений и их описание



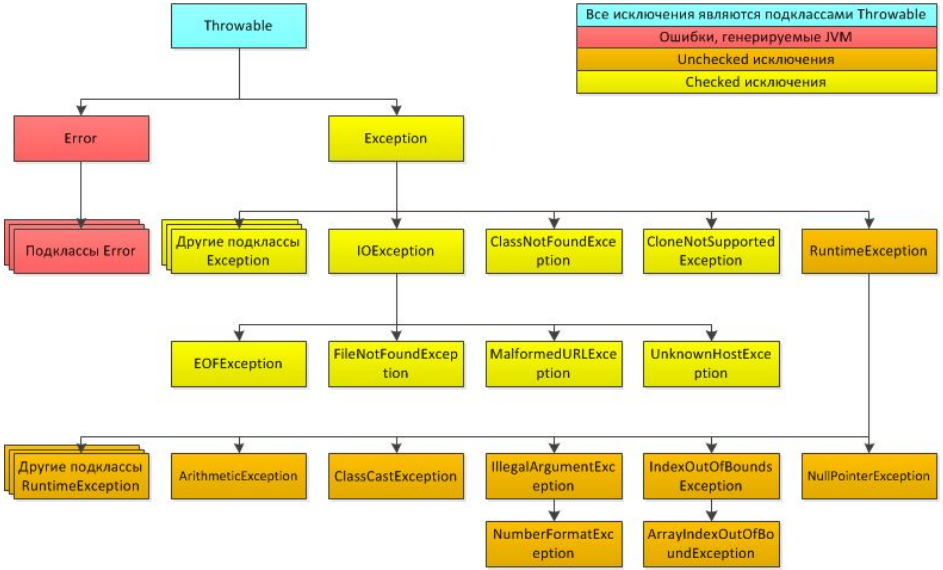
Типы исключений и их описание

Попробуем допустить ошибки в коде

Попробуем допустить ошибки в коде

```
// ArithmeticException, такой пример уже смотрели
  int a = 0;
  int b = 10;
  int c = b / a;
-------------------------------------------------------
// NullPointerException
  String str = null;
  System.out.println(str.length());
-------------------------------------------------------
// ArrayIndexOutOfBoundsException
  int[] array = new int[10];
  array[100] = 5;
-------------------------------------------------------
// NumberFormatException
  int value = Integer.parseInt("100a0");
-------------------------------------------------------
// ClassCastException
  Animal animal = new Cat();
  Dog dog = (Dog)animal;
```

Иерархия исключений

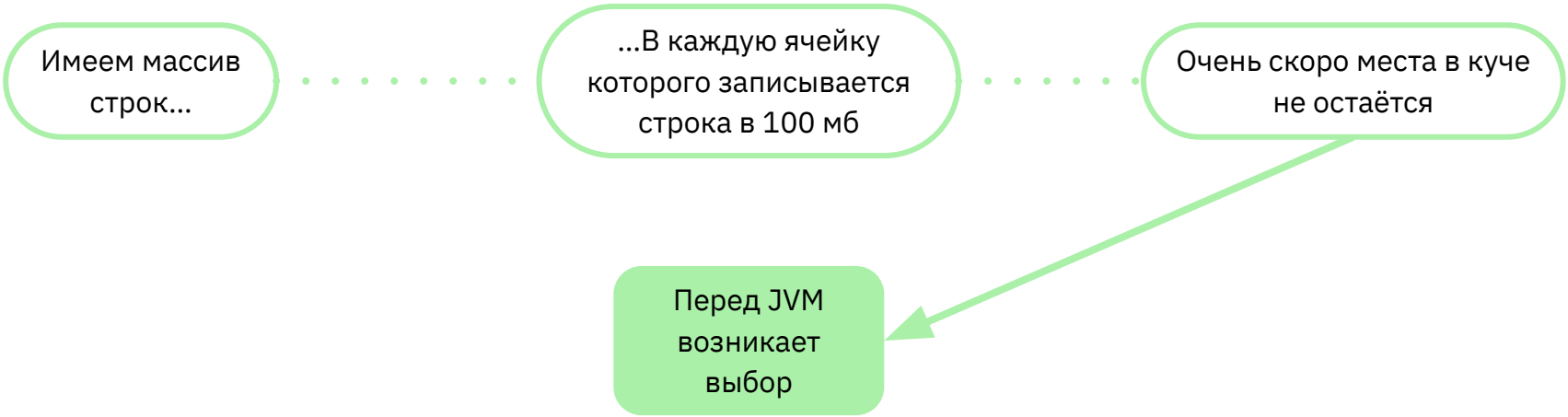


Checked. *Класс Exception и его подклассы* — исключения, которые вы **обязаны** обработать в вашем коде. Если этого не сделать, будет ошибка на этапе компиляции.

Unchecked. *Класс RuntimeException и его подклассы* — исключения, охватывающие такие ситуации, как деление на ноль или ошибочная индексация массивов.

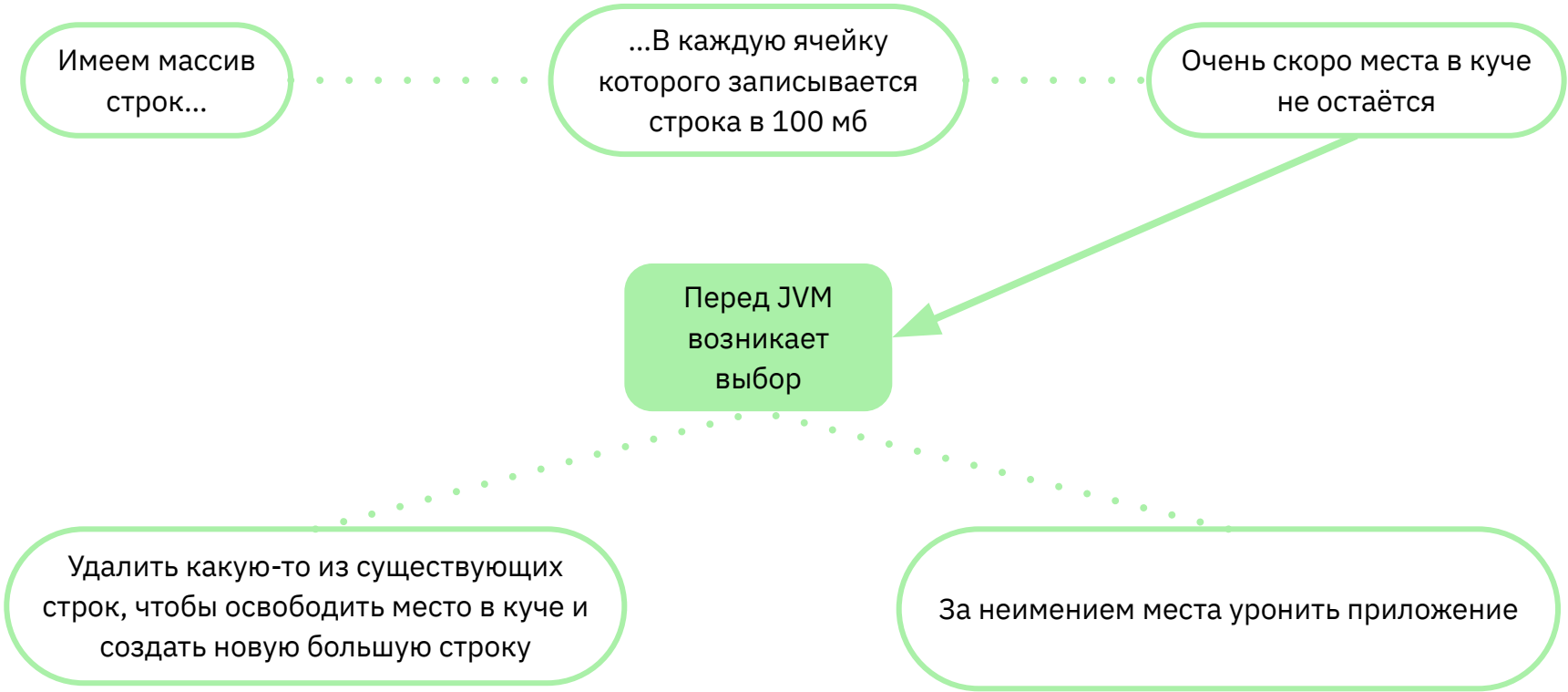
- ✔ Их можно обрабатывать, если есть вероятность возникновения.
- ✔ А можно и не обрабатывать, поскольку предполагается, что при правильном поведении программы такие исключения вовсе не должны возникать.

Что такое Error?



Что такое Error?

Что такое Error?



Что такое Error?

Имеем массив строк...

...В каждую ячейку которого записывается строка в 100 мб

Очень скоро места в куче не остаётся

Как поступить JVM?

Удалить какую-то из существующих строк, чтобы освободить место в куче и создать новую большую строку

За неимением места уронить приложение

Как поступить JVM?

Если JVM будет просто удалять объекты по своему желанию,

вы никогда не будете уверены, что ваше приложение работает как задумано: можно получить NullPointerException, где его быть никак не должно.

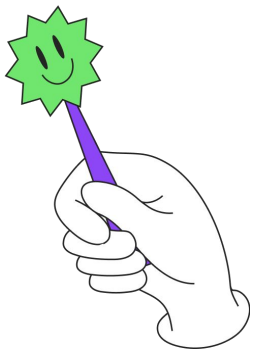
Итак, в результате будет выброшена ошибка OutOfMemoryError, **обозначающая, что в куче больше нет места и его никак не высвободить**.

Как же бороться с OutOfMemoryError?

Как бороться с OutOfMemoryError

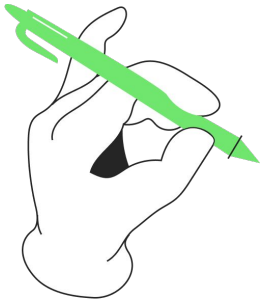
Продумывать код так, чтобы:

- объекты не могли заполнить кучу,
- либо заранее увеличить её максимальный размер.



Либо использовать слабые ссылки на объекты, чтобы JVM в случае проблем с памятью могла спокойно удалять объекты по этим ссылкам.

Начинаем обрабатывать исключения

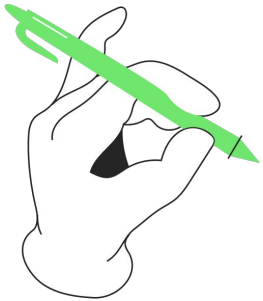


Обработать исключение можно одним из **двух способов**

Поместить код, бросающий исключение, в блок **try-catch**

Пробросить с помощью **throws** исключение методу на уровень выше, то есть методу, вызывающему текущий.

Начинаем обрабатывать исключения



Обработать исключение можно одним из **двух способов**

Поместить код, бросающий исключение, в блок **try-catch**

Пробросить с помощью **throws** исключение методу на уровень выше, то есть методу, вызывающему текущий

[третий плохой путь в никуда]

Вообще не обрабатывать исключение. Но в таком случае ваш код либо не скомпилируется, либо будет работать крайне нестабильно

Применяем try-catch

Применяем try-catch

```
public static void main(String[] args) {
    try {
        int a = 0;
        int b = 10 / a;
        System.out.println("Это сообщение не будет выведено в консоль");
    } catch (ArithmeticException e) {
        System.out.println("Деление на ноль");
    }
    System.out.println("Завершение работы");
}

Результат в консоли:
Деление на ноль
Завершение работы
```

- **try** — попытка выполнить код, в котором потенциально может возникнуть исключение
- **catch** — перехват исключения указанного типа (или его наследника) с целью обработать возникшую ошибочную ситуацию

Применяем try-catch

```
public static void main(String args[]) {
    System.out.println("Начало");
    try {
        int a = 0;
        int b = 42 / a;
    } catch (ArithmeticException e) {
        e.printStackTrace();
    }
    System.out.println("Конец");
}
```

Результат в консоли:

```
Начало
java.lang.ArithmeticException: / by zero
     at MainClass.main(MainClass.java:7)
Конец
```

Несколько операторов catch

Несколько операторов catch

```
public static void main(String args[]) {
    try {
        int a = 10;
        a -= 10;
        int b = 42 / a;
        int[] c = {1, 2, 3};
        c[42] = 99;
    } catch (ArithmeticException e) {
        System.out.println("Деление на ноль: " + e);
    } catch (ArrayIndexOutOfBoundsException e){
        System.out.println("Ошибка индексации массива: " + e);
    }
    System.out.println("После блока операторов try/catch");
}
```

Блоки catch проверяются сверху вниз

Сколько блоков catch сработает?

Важен ли порядок catch?

Важен ли порядок catch

```
public static void main(String args[]) {
    try {
        int a = 0;
        int b = 42 / a;
    } catch (Exception e) {
        System.out.println("Exception");
    } catch (ArithmeticException e) { // ошибка компиляции: недостижимый код !
        System.out.println("Этот код недостижим");
    }
}
```

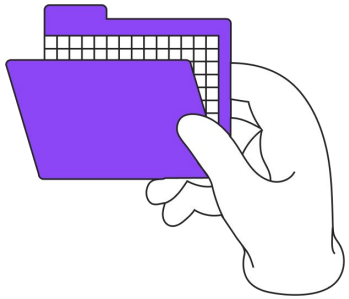
Так как блок catch с Exception стоит первым, он будет всегда первым перехватывать ***все*** исключения, и до остальных блоков выполнение кода никогда не дойдёт. Отсюда — ошибка недостижимости кода.

Если требуется обработать несколько типов, обработка исключений должна идти от наследника к родительскому классу.

Зачем вообще нужно так много типов исключения?

Почему бы просто не обрабатывать всегда корневой Exception?

Объяснение



Исключения разного типа требуют разной реакции

Если вы хотите в своей программе передать файл по сети, могут возникнуть два или больше исключений:

1. **FileNotFoundException** — файл по указанному пути не найден.
2. **IOException** — сетевое соединение разорвалось.

Объяснение

Файл по указанному пути не найден (FileNotFoundException)

Запросите у пользователя новый путь к файлу

Сетевое соединение разорвалось (IOException)

Попытайтесь переподключиться к серверу, куда отправляете файл

Оператор throws

Оператор throws

```
public static void main(String args[]) {
    try {
        createReport();
    } catch (IOException e) {
        e.printStackTrace();
    }
}


public static void createReport() throws IOException {
    PrintWriter writer = new PrintWriter("report.txt");
    writer.close();
}
```

За счёт throws в методе createReport() мы не обрабатываем Checked IOException, а пробрасываем его наверх.

В данном случае — в метод main().

Блок finally

Блок finally

```
try {
   // блок кода, в котором отслеживаются исключения
} catch (ТипИсключения1 e) {
   // обработчик исключения тип_исключения_1
} catch (ТипИсключения2 e) {
   // обработчик исключения тип_исключения_2
} finally {
   // блок кода, который обязательно выполнится по завершении блока try
}
```

Как правило, finally используется для обязательного закрытия системных ресурсов.

Многократный перехват исключения

Многократный перехват исключения

Если на несколько видов исключений у вас одна и та же реакция, их можно объединить в блоке catch через оператор ИЛИ.

```
try {
    …
} catch (ArithmeticException | ArrayIndexOutOfBoundsException e) {
    …
}
```

Спасибо за внимание!