Лабораторная работа № 18
Основы программирования на Java. Исключительные ситуации в коде

Цели и задачи:

1. Изучить исключительные ситуации и их применение
2. Изучить встроенные исключения (`Exception`, `IndexOutOfBoundsException`, `NullPointerException`)
3. Разработать приложение, которое обработает исключительную ситуацию

Начало работы

Убедитесь в работоспособности вашего ПК. При обнаружении неполадок, сообщите преподавателю.

Для работы вам потребуется:

1. Среда разработки Eclipse (возможно так же использование другой IDE)
2. Java Development Kit 8 и выше.
3. Тетрадь и ручка для записи важных моментов

Теоретическая часть

Исключение - это нештатная ситуация, ошибка во время выполнения программы. Самый простой пример - деление на ноль. Можно вручную отслеживать возникновение подобных ошибок, а можно воспользоваться специальным механизмом исключений, который упрощает создание больших надёжных программ, уменьшает объём необходимого кода и повышает уверенность в том, что в приложении не будет необработанной ошибки.

Существует пять ключевых слов, используемых в исключениях: `try`, `catch`, `throw`, `throws`, `finally`. Порядок обработки исключений следующий.

Операторы программы, которые вы хотите отслеживать, помещаются в блок try. Если исключение произошло, то оно создаётся и передаётся дальше. Ваш код может перехватить исключение при помощи блока `catch` и обработать его. Системные исключения автоматически передаются самой системой. Чтобы передать исключение вручную, используется `throw`. Любое исключение, созданное и передаваемое внутри метода, должно быть указано в его интерфейсе ключевым словом `throws`. Любой код, который следует выполнить обязательно после завершения блока `try`, помещается в блок `finally`

Схематически код выглядит так:

```
try {
    // блок кода, где отслеживаются ошибки
}
catch (тип_исключения_1 exceptionObject) {
    // обрабатываем ошибку
}
catch (тип_исключения_2 exceptionObject) {
    // обрабатываем ошибку
}
finally {
    // код, который нужно выполнить после завершения блока try
}
```

Существует специальный класс для исключений `Trowable`. В него входят два класса `Exception` и Error.

Класс `Exception` используется для обработки исключений вашей программой. Вы можете наследоваться от него для создания собственных типов исключений. Для распространённых ошибок уже

существует класс **`RuntimeException`**, который может обрабатывать деление на ноль или определять ошибочную индексацию массива.

Класс **`Error`** служит для обработки ошибок в самом языке Java и на практике вам не придётся иметь с ним дело.

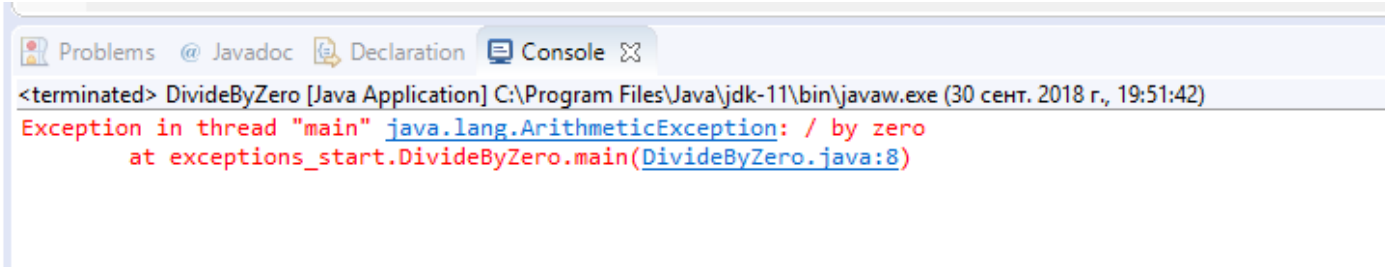
Прежде чем научиться обрабатывать исключения, нам (как и нормальному любопытному коту) хочется посмотреть, а что происходит, если ошибку не обработать. Давайте разделим число котов в вашей квартире на ноль, хотя мы и знаем, что котов на ноль делить нельзя!

Для начала поделим на ноль…

Давайте создадим простой проект. Создадим в нем простой файл и назовем его, например, **`DivideByZero.java`** и напишем следующий код:

```
public class DivideByZero {
	public static void main(String[] args) {
		int a = 90;
		int b = 0;
		System.out.println(a / b);
	}
}
```

И попробуем запустить:



Что произошло? Вот тут-то и начинается некая «магия». Java поняла, что мы сделали гадость – разделили на ноль. В математике деление на ноль запрещено. Как вы могли заметить, когда делали задачу №1, при делении на ноль вы как раз-таки получали исключительную ситуацию. Механизм исключений в Java устроен так, что сама JVM отслеживает неправильный код. Воспользуемся полученными знаниями из теории, и обернем эту операцию в **`try-catch`**!

```
try {
	int a = 90;
	int b = 0;
	System.out.println(a / b);
}
catch (Exception ex) {
	System.out.println("Деление на ноль запрещено!");
}
```

В таком случае мы не получим сообщение об ошибке от JVM, а просто обработаем такую ситуацию таким образом, словно ничего страшного не произошло. Ведь в случае не обработанного исключения программа завершает свою работу. Согласитесь, для пользователя гораздо проще понять написанное вами сообщение об ошибке. Запустите программу и убедитесь, что все в порядке.

Научились «ловить» исключения. Давайте научимся их «выбрасывать».

Данное действие нужно тогда, когда работа программы невозможна и необходимо остановить поток. Чтобы «выбросить» (от англ. «throw» – бросить) исключение, воспользуемся следующим кодом:

```
try {
	int a = 90;
	int b = 0;
	System.out.println(a / b);
}
catch (Exception ex) {
	throw new Exception("Ай, что-то не так!");
}
```

Таким образом, мы точно выйдем из потока выполнения программы, но будем знать, что произошло. В качестве конструктора класс **`Exception`** принимает аргумент, который необходимо вывести в качестве сообщения об ошибке. Мы так же можем вывести сообщение, которое сгенерировала сама JVM при помощи метода **`getMessage()`** класса **`Exception`**:

```
try {
	int a = 90;
	int b = 0;
	System.out.println(a / b);
}
catch (Exception ex) {
	throw new Exception(ex.getMessage());
}
```

Потренируйтесь с выбросом исключений. Помните, что класс Exception является основным. Если вы не знаете, какое исключение «ловить», используйте его в блоке **`catch`**.

Ловим конкретное исключение

Рассмотрим ситуацию, в которой мы можем сделать 3 непоправимые во время выполнения вещи:

1. Выход за пределы массива (**`IndexOutOfBoundsException`**)
2. Деление на ноль (**`ArithmeticException`**)
3. Обращение к элементу, который не существует (**`NullPointerException`**)

Напишем такой код:

```
public static void main(String[] args) {
	try {
		int a = 90;
		int b = 3;
		System.out.println(a / b);
		printSum(23, 234);
		int[] abc = { 1, 2 };
		abc[3] = 9;
	} catch (ArithmeticException ex) {
		System.out.println("Деление на ноль запрещено!");
	} catch (NullPointerException ex) {
		System.out.println("Указатель не может указывать на null!");
	} catch (IndexOutOfBoundsException ex) {
		System.out.println("Массив выходит за пределы своего размера!");
	}
}

public static void printSum(Integer a, Integer b) {
	System.out.println(a + b);
}
```

Попробуйте, не запуская код догадаться, какое исключение «выбросит» JVM?

Почему **Integer**? **NullPointerException** работает только со ссылками (*pointer*), а ссылки возможно создавать только у классов. В Java существует тип **int**, и это ТИП. В то же самое время, существует КЛАСС **Integer**, который содержит кучу разных полезных методов и констант для работы с целочисленными значениями.

Если мы напишем так:

```
int x; // x = 0;
x = 23; // x = 23;
```

То x в данном конкретном случае будет сразу же инициализирована со значением 0. Это тип. В то же время **Integer**:

```
Integer x; // x = null;
System.out.println(x); // The local variable x may not have been initialized
```

Вызовет **ошибку**, так как x НЕ ИМЕЕТ значения, и является null. Она ОБЯЗАНА быть инициализирована.

Исключения уровня методов

Если метод способен возбуждать исключения, которые он сам не об-рабатывает, он должен объявить о таком поведении, чтобы вызывающие методы могли защитить себя от этих исключений. Для задания списка исключений, которые могут возбуждаться методом, используется ключе-вое слово throws. Если метод в явном виде (т.е. с помощью оператора throw) возбуждает исключе-ние соответствующего класса, тип класса исключений должен быть ука-зан в операторе throws в объявлении этого метода. С учетом этого наш прежний синтаксис определения метода должен быть расширен следую-щим образом:

```
тип имя_метода(список аргументов) throws список_исключений {}
```

Уведомим другие методы, что **printSum** может вызвать исключение **NullPointerException**:

```
public static void printSum(Integer a, Integer b) throws NullPointerException {
	if(a == null || b == null) throw new NullPointerException("Oops!");
	System.out.println(a + b);
}
```

Так же мы воспользовались **throw** для выброса исключений.

Самостоятельная работа

Задача №1

Разработайте программу, которая выбросит **Exception**, когда пользователь вводит пустую строку. Попробуйте так же в действии блок **finally**. Покажите результат преподавателю.

Сформируйте и сдайте отчет по работе!