**Знакомство с исключениями**

[Java Syntax](https://javarush.ru/quests/QUEST_JAVA_SYNTAX)

[Уровень 9](https://javarush.ru/quests/lectures?quest=QUEST_JAVA_SYNTAX&level=9), Лекция 3

— Привет, Амиго! Сегодня будет очень интересный урок. Сегодня я расскажу тебе об исключениях. Исключения – это специальный механизм для контроля над ошибками в программе. Вот примеры ошибок, которые могут возникнуть в программе:

1. Программа пытается записать файл на заполненный диск.

2. Программа пытается вызвать метод у переменной, которая хранит ссылку – null.

3. Программа пытается разделить число на 0.

Все эти действия приводят к возникновению ошибки. Обычно это приводит к закрытию программы — продолжать выполнять дальше код не имеет смысла.

— Почему?

— А есть ли смысл крутить руль, если машина слетела с трассы и падает с обрыва?

— Программа что, должна завершиться?

— Да. Раньше так и было. Любая ошибка приводила к завершению программы.

— Это очень разумный подход.

— А разве не лучше было бы попробовать работать дальше?

— Ага. Ты набрал большущий текст в Word’е, сохранил его, он не сохранился, но программа говорит тебе, что все в порядке. И ты продолжаешь набирать его дальше. Глупо, да?

— Ага.

— Потом разработчики придумали интересный ход: каждая функция возвращала статус своей работы. 0 означал, что она отработала как надо, любое другое значение – что произошла ошибка: это самое значение и было кодом ошибки.

— Но был у такого подхода и минус. После каждого(!) вызова функции нужно было проверять код (число), который она вернула. Во-первых, это было неудобно: код по обработке ошибок исполнялся редко, но писать его нужно было всегда. Во-вторых, функции часто сами возвращают различные значения – что делать с ними?

— Ага. Я тоже об этом подумал.

— Но потом наступило светлое будущее — появились исключения и механизм обработки ошибок. Вот как это работает:

1. Когда возникает ошибка, Java-машина создаёт специальный объект – exception – исключение, в который записывается вся информация об ошибке. Для разных ошибок есть разные исключения.

2. Затем это «исключение» приводит к тому, что программа тут же выходит из текущей функции, затем выходит из следующей функции, и так пока не выйдет из метода main. Затем программа завершается. Еще говорят, что Java-машина «раскручивает назад стек вызовов».

— Но ты же сказала, что теперь программа не обязательно завершается.

— Верно, потому что есть способ перехватить исключение. В нужном месте, для нужных нам исключений мы можем написать специальный код, который будет перехватывать эти исключения и что-то делать. Важное.

— Для этого есть специальная конструкция **try-catch**. Вот как это работает:

|  |
| --- |
| **Вот пример программы, которая перехватывает исключение – деление на 0. И продолжает работать.** |
| public class ExceptionExample2  {  public static void main(String[] args)  {  System.out.println("Program starts");  try  {  System.out.println("Before method1 calling");  method1();  System.out.println("After method1 calling. Never will be shown");  }  **catch (Exception e)**  {  System.out.println("Exception has been caught");  }  System.out.println("Program is still running");  }  public static void method1()  {  int a = 100;  int b = 0;  System.out.println(a / b);  }  } |
| **Вот что будет выведено на экран:** |
| «Program starts»  «Before method1 calling»  «Exception has been caught»  «Program is still running» |

— А почему не будет выведено «After method1 calling. Never will be shown»?

— Рада, что ты спросил. В строчке 25 у нас было деление на ноль. Это привело к возникновению ошибки – исключения. Java-машина создала объект *ArithmeticException* с информацией об ошибке. Этот объект является исключением.

— Внутри метода method1 возникло исключение. И это привело к немедленному завершению этого метода. Оно привело бы и к завершению метода **main**, если бы не было блока **try-catch.**

— Если внутри блока **try** возникает исключение то, оно захватывается в блоке **catch**. Остаток кода в блоке try, не будет исполнен, а сразу начнётся исполнение блока **catch**.

— Как-то не очень понятно.

— Другими словами этот код работает так:

1. Если внутри блока **try** возникло исключение, то код перестаёт исполняться, и начинает исполняться блок **catch**.

2. Если исключение не возникло, то блок try исполняется до конца, а **catch** никогда так и не начнёт исполняться.

— Гм?

— Представь, что после вызова каждого метода мы проверяем: завершился ли только что вызванный метод сам по себе или в результате исключения. Если исключение было, тогда мы переходим на исполнение блока catch, если он есть, и захватываем исключение. Если блока catch нет, то завершаем и текущий метод. Тогда такая же проверка начинается в том методе, который вызвал нас.

— Теперь вроде понятно.

— Вот и отлично.

— А что значит Exception внутри catch?

— Все исключения – это классы, унаследованные от класса Exception. Мы можем перехватить любое из них, указав в блоке **catch** его класс, или все сразу, указав общий родительский класс — Exception. Затем из переменной e (эта переменная хранит ссылку на объект исключения), можно получить всю необходимую информацию о возникшей ошибке.

— Круто! А если в моем методе могут возникнуть разные исключения, можно обрабатывать их по-разному?

— Не можно, а нужно. Сделать это можно вот так:

Пример:

public class ExceptionExample2

{

public static void main(String[] args)

{

System.out.println("Program starts");

try

{

System.out.println("Before method1 calling");

method1();

System.out.println("After method1 calling. Never will be shown ");

}

catch (NullPointerException **e**)

{

System.out.println("Reference is null. Exception has been caught");

}

catch (ArithmeticException e)

{

System.out.println("Division by zero. Exception has been caught");

}

catch (Exception **e**)

{

System.out.println("Any other errors. Exception has been caught");

}

System.out.println("Program is still running");

}

public static void method1()

{

int a = 100;

int b = 0;

System.out.println(a / b);

}

}

— Блок try может содержать несколько блоков catch, каждый из которых будет захватывать исключения своего типа.

— Гм. Ну, вроде понятно. Сам такого не напишу, конечно, но если в коде встречу – пугаться не буду.