

Исключения в программировании
и их обработка

Урок 1

Работа с ошибками в программировании





Оглавление

На этом уроке	3
Введение	3
Термины, используемые в лекции	4
Источники ошибок в коде	4
Сценарии взаимодействия с ошибками	5
Сценарий 1	5
Сценарий 2	6
Как программа оповещает разработчика об ошибках	6
Первый подход	7
Второй подход	8
Недостатки кодов ошибок	9
Исключения	10
Подробнее про исключения	11
А можем ли мы сами бросать исключения?	15
Заключение	17
Контрольные вопросы	17



На этом уроке

1. Обсудим, откуда при написании программ берутся ошибки.
2. Узнаем, как программа оповещает разработчика об ошибках: используя коды ошибок или исключения.
3. Перечислим плюсы и минусы этих подходов.
4. Выясним, что означает обработка ошибок и как она выполняется.

Введение

Добро пожаловать на курс «Исключения в программировании и их обработка». Тема обработки исключений и ошибок считается одной из основополагающих в разработке программного обеспечения, и разработчик обязан учитывать, что при возможных ошибках в коде приложение должно работать стабильно. Следовательно, надо чётко понимать, какие ошибки и по каким причинам могут возникать при выполнении кода, и как написать код, который минимизирует негативное воздействие этих ошибок.

В рамках этого курса мы познакомимся с причинами возникновения ошибок, способами, которыми приложение о них оповещает разработчика, и подходами к перехвату и обработке ошибочных ситуаций. В курсе вы найдёте ответы на такие вопросы:

1. Что может быть причинами возникновения ошибок в процессе выполнения программы?
2. Как приложение говорит, какая именно ошибка возникла и с чем она связана?
3. Что такое коды ошибок, исключения, и чем они отличаются?
4. Почему важно помнить про места в коде, которые могут приводить к потенциальным ошибкам?
5. Как выполнять обработку ошибок или исключений для повышения отказоустойчивости приложения?
6. Что такое исключения в языке Java и как они устроены?



Работа с ошибками — одна из важнейших тем в программировании, которая позволяет писать отказоустойчивые приложения.



Отказоустойчивые приложения — приложения, которые при возникновении некритичных ошибок могут продолжать свою работу.

Термины, используемые в лекции

Исключения — это специальный тип объектов, которые создаются при возникновении ошибочной ситуации и выбрасываются Java-машиной.

Код ошибки — целочисленный код, указывающий на причину ошибки. Для разных приложений — разные коды ошибок.

Источники ошибок в коде

Узнаем, откуда берутся ошибки в коде, и приведём примеры.

1. Один из наиболее распространённых случаев — когда разработчик что-то не учёл в своём коде, и определённый набор входных данных привёл к возникновению ошибки.

Самый простой пример: написание консольного калькулятора. Разработчик добавил функцию деления числа и позволил пользователю вводить абсолютно любой делитель, но забыл, что делить на ноль нельзя.

Пользователь может спокойно в качестве делителя указать ноль, что при дальнейшем исполнении приведёт к возникновению ошибки и падению приложения. Если бы разработчик поставил условие, что при указании 0 в качестве делителя не надо выполнять операцию деления, а просто показать пользователю сообщение с предупреждением, программа продолжила свою работу.

Есть ещё один подход: мы можем указать программе, что при возникновении ошибок, связанных с исполнением арифметических операций, надо показывать пользователю только сообщение о некорректности введённых данных и не закрывать приложение целиком. Это и есть обработка ошибок, но об этом чуть позже.



2. Второй возможный сценарий заключается в том, что разработчик абсолютно корректно написал своё приложение, но некая внешняя среда, с которой его приложение взаимодействует, привела к возникновению ошибочной ситуации.

Звучит сложно, но есть простой пример. Вы написали программу, пишущую некий файл отчёта на флешку, по пути, который указал пользователь. Возможна ситуация, когда пользователь запустил такую операцию, и она выполняется довольно длительное время, например, пять минут. И в это время пользователь, забыв о работающем приложении, выдёргивает флешку из системного блока. Получается, с одной стороны, приложение должно дописать этот файл, а с другой — сделать это невозможно, потому что записывать некуда. В таком случае системная функция записи данных в файл выбросит ошибку. Это снова приведёт к падению приложения, если не перехватить и не обработать ошибку.

3. И третий вариант: как правило, вы можете использовать сторонние библиотеки, код которых написан не вами. В них тоже могут возникать различного рода ошибки, которые будут приводить в том числе к падению вашего приложения, если не обрабатывать эти ошибки.

Сценарии взаимодействия с ошибками

Сценарий 1

Программа не может корректно работать после возникновения ошибки, соответственно, она просто закроется. Это плохой вариант, разберём почему. Берём калькулятор и пытаемся делить на 0. Из-за возникновения ошибки калькулятор просто бы закрылся, ничего никому не сказав. Согласитесь, вариант не очень хороший.

Рассмотрим второй пример. Вы, используя какое-то приложение, копируете очень большой файл на флеш-накопитель, и в какой-то момент в программе возникает ошибка, потому что на диске закончилось свободное место. При рассматриваемом сценарии программа порождает ошибку и просто закрывается, вас же при этом никак не оповещая об ошибке. В итоге вам кажется, что операция завершена успешно, и вы, не глядя, забираете флешку с битыми данными.



Сценарий 2

При возникновении ошибки программа перехватывает ошибку и корректно на неё реагирует. Например, показывая пользователю сообщение о её возникновении. В таком случае пользователь сможет понять:

- что на 0 делить нельзя;
- что на диске недостаточно места, и надо как-то это поправить.






Цель нашего курса — как раз научиться корректно реализовывать второй сценарий, то есть перехватывать возможные ошибки и максимально корректно их обрабатывать.

Как программа оповещает разработчика об ошибках

Если в приложении возникает ошибка, было бы неплохо узнать, откуда она там взялась.

Например, если заполняете на странице интернет-сайта какую-то форму и указали случайно некорректный email (без @), то страница подсветит вам это поле и выдаст ошибку, что email некорректный. В таком случае вы сразу понимаете, что сделали не так, и всё исправляете.

Вход через соцсети

или Email

Email

Введите адрес электронной почты.

Пароль

☐ Запомнить меня [Войти по смс](#)

[Войти](#)

[Забыли пароль?](#) [Не получили письмо для активации?](#)



Первый подход

Первый подход — это использование кодов ошибок. Его можно встретить, например, в языке Си.



Код ошибки — целочисленный код, указывающий на причину ошибки.

Для функции или метода выбирается определённое значение, которое не входит в область значений этой функции. И если это значение будет возвращено, значит, при исполнении возникла ошибка. Формулировка немного сложная, поэтому разберёмся на примере.

Представим, что в Java вместо исключений использовались бы коды ошибок. В таком случае метод получения длины файла имел бы вид как на слайде. В качестве аргумента методу передаётся путь к файлу в виде строки. По этому пути мы делаем проверку существования файла. Если файл не существует, возвращаем значение -1, которое фактически сообщает об ошибке — нельзя запросить размер несуществующего файла. А если файл существует — вернётся его длина в байтах.

```
public static long getFileLength(String path) {  
    File file = new File(path);  
    if (!file.exists()) {  
        return -1;  
    }  
    return file.length();  
}
```

В этом примере мы точно видим, что -1 — это ошибка, поскольку файлов с отрицательным размером существовать не может. Если получили неотрицательное значение, значит, файл существует, и мы его измерили. Здесь нам повезло — значение ошибки не входит в область возможных возвращаемых функцией значений.



Второй подход

Разберём более сложный пример с кодами ошибок и создадим метод деления двух чисел. Как мы ранее говорили, в нашем калькуляторе делить на 0 нельзя, то есть надо продумать момент, где при попытке деления на 0 возвращается ошибка.

```
public static int div(int a, int b) {  
    if (b == 0) {  
        return -1;  
    }  
    return a / b;  
}
```

На изображении мы видим код метода, который повторяет подход к обработке ошибок из метода получения размера файла. То есть в случае ошибки — вернуть -1.

Теперь у нас появляется довольно большая проблема:

почему метод вернул -1?

1. Мы попытались поделить на 0, так делать нельзя, и мы получили ошибку?
2. Или мы выполнили корректное деление, и на самом деле получили -1, например, поделив 5 на -5 или 10 — на -10.

В результате пользователь этого метода не сможет понять, корректно у него сработал метод или нет. И мы в принципе не можем выбрать никакого специального кода ошибки, поскольку при делении нескольких чисел можем получить совершенно любое целое число. Здесь коды ошибок, конечно же, использовать крайне сложно.

Недостатки кодов ошибок

Разберём недостатки использования кодов ошибок.



1. Зачастую код ошибки невозможно отличить от того результата, который действительно может вернуть метод.

В качестве примера вспомним деление на 0, где -1 — это и возможный результат, и возможный код ошибки.

2. Разработчик, использующий метод, может забыть, что метод возвращает какой-то список кодов ошибок, и что на них надо реагировать.

Этот список ещё где-то надо найти (в документации), что тоже крайне неудобно.

3. Есть некоторые ошибки, которые обязательно надо обрабатывать.

Допустим, вы подключаетесь к базе данных, и в процессе работы возникает ошибка, приводящая к завершению работы приложения. В таком случае надо обязательно закрывать соединение с сервером базы данных, чтобы не тратить её ресурсы. Желательно, чтобы проверка наличия обработчика ошибки происходила на этапе компиляции, что невозможно сделать с кодами ошибок.

4. При возникновении ошибки мы видим только код, но у нас нет никакой служебной информации, почему это ошибка возникла. То есть нельзя как-то детализировать, какие входные данные привели к появлению этой ошибки.

Если мы говорим конкретно про язык Java, то метод не может возвращать два разных типа данных в зависимости от результата работы: в одном случае, при корректной работе вернуть объект, а в случае возникновения ошибки — целочисленное значение.

Исключения

В языке Java для обработки ошибок используются не коды ошибок, а исключения.



Исключения — это специальный тип объектов, которые создаются при возникновении ошибочной ситуации и выбрасываются Java-машиной.

Рассмотрим пример деления на ноль.



```
1 public class MainClass {  
2     public static void main(String[] args) {  
3         int a = 0;  
4         int b = 10 / a;  
5     }  
6 }
```

Java не поддерживает целочисленное деление на ноль, поэтому при выполнении кода, показанного на изображении выше, произойдёт следующее: программа завершит свою работу с ошибкой, ведь мы не обработали исключение, и в консоль выводится сообщение, как на изображении ниже.

```
Exception in thread "main" java.lang.ArithmeticException: / by zero  
at MainClass.main(MainClass.java:4)
```

Из сообщения можно понять, что возникло исключение, в связи с чем программа не может дальше корректно выполняться и завершает работу. В сообщении при этом довольно много деталей, которые могут помочь в исправлении кода:

- ошибка возникла в потоке main;
- было брошено исключение, имеющее тип `java.lang.ArithmeticException`, который указывает на ошибку при выполнении арифметической операции;
- есть уточняющее сообщение, о делении на 0, и далее прописано имя класса и метода, а также строка, где возникла эта ошибка.

То есть посредством исключения мы поняли не только тип ошибки, но и причину и место её возникновения, что значительно облегчит исправление кода.



Подробнее про исключения

Рассмотрим подробнее, как работает механизм обработки ошибок в языке Java. Возьмём пример того же кода.

```
1 public class MainClass {  
2     public static void main(String[] args) {  
3         int a = 0;  
4         int b = 10 / a;  
5     }  
6 }
```

1. Итак, в строке 4 возникает попытка деления на ноль.
2. В этот момент исполняющая среда Java приостанавливает работу программы, поскольку целочисленное деление на ноль недопустимо.
3. Затем JVM формирует объект типа `ArithmeticException`, который служит исключением, и выполняет так называемый бросок этого объекта, чтобы в случае необходимости мы могли его перехватить и обработать.
4. В нашем простом коде никакой обработки нет, поэтому объект летит и долетает до стандартного обработчика исключений Java-машины.

Если исключение перехватывает не вашим кодом, а стандартным обработчиком, то стандартный обработчик останавливает работу приложения целиком и выводит в консоль сообщение об исключении.

```
Exception in thread "main" java.lang.ArithmeticException: / by zero  
    at MainClass.main(MainClass.java:4)
```

В этом сообщении есть вся необходимая детализация:



1. исключение возникло в потоке `main` — в основном потоке исполнения Java-приложения;
2. `java.lang.ArithmeticException` — тип исключения, указывающий, что ошибка связана с исполнением арифметических операций;
3. исключение возникло в классе `MainClass` в четвёртой строке файла `MainClass.java`.

Это упрощает дальнейшую работу с кодом.

Рассмотрим более сложный пример возникновения исключения, который демонстрирует новый тип исключения.

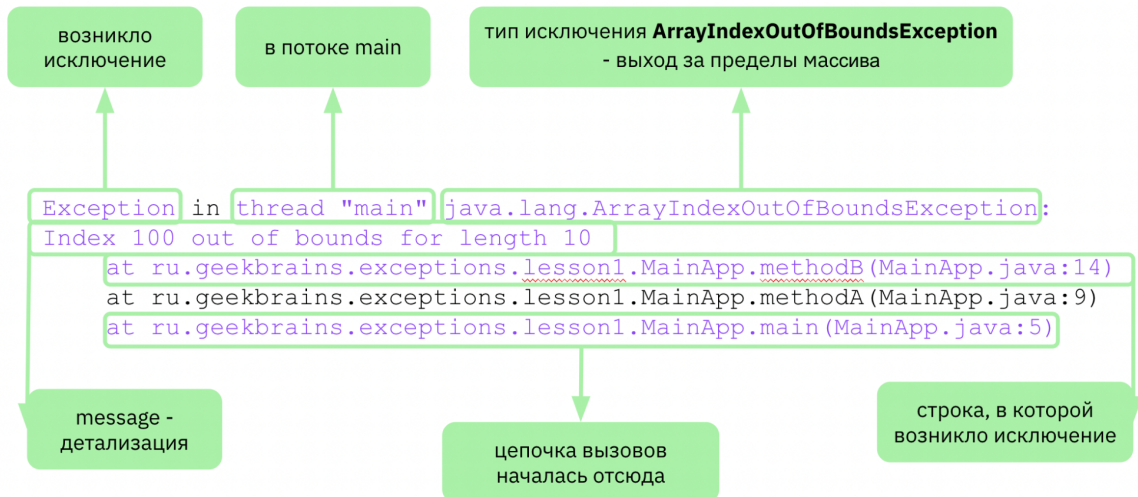
```
1 package ru.geekbrains.exceptions.lesson1;
2
3 public class MainApp {
4     public static void main(String[] args) {
5         methodA();
6     }
7
8     public static void methodA() {
9         methodB();
10    }
11
12    public static void methodB() {
13        int[] arr = new int[10];
14        System.out.println(arr[100]);
15    }
16 }
```

1. Из кода выше мы видим, что в приложении есть метод `main()`, вызывающий `methodA()`.
2. `methodA()`, в свою очередь, вызывает `methodB()`, где создаётся целочисленный массив длиной 10.
3. Затем программа отпечатывает в консоль сотый элемент массива.

Но такую операцию невозможно выполнить, так как мы не можем взять сотый элемент, если у нас их всего 10. Поскольку такая операция



невыполнима, Java-машина сгенерирует и бросит исключение, указывающее на недопустимость операции. Ниже приводится пример выполнения этого кода.



Разберём по порядку вывод в консоль.

Во-первых, исключение возникло в потоке `main`, то есть в основном потоке исполнения нашего кода.

Во-вторых, мы получили новый тип исключения, а именно: `ArrayIndexOutOfBoundsException` — больше типов исключений разберём на следующем занятии. Оно говорит, что мы вышли за пределы массива — попытались обратиться к несуществующему в массиве элементу. Далее идёт уточняющее сообщение: «Индекс 100 выходит за пределы длины массива в 10 элементов».

В-третьих, видим три строки, которые указывают на часть кода, приведшего к возникновению ошибки. Мы можем идти по ним как сверху вниз, так и снизу вверх — это зависит от того, что хотим получить. Самая верхняя строка указывает на класс/метод/строку кода, где возникло исключение: в нашем случае — в классе `ru.geekbrains.exceptions.lesson1.MainApp`, методе `methodB()`, строка 14 в файле `MainApp.java`.

Это правило будет сохраняться всегда. Если в консоли вы увидели распечатку стека вызовов, то, глядя на верхнюю строку стека, сразу сможете найти причину ошибки. Но часто надо понять, как мы попали в указанную строку и для этого можем пойти по стеку снизу вверх: начал работать метод



main(), который вызвал метод methodA(), вызвавший methodB(), где исполнилась строка с некорректным обращением к массиву.

Итак, если идём по стеку сверху вниз, то пройдем от места ошибки к началу всей цепочки вызовов методов. А если снизу вверх, то увидим, как программа выполнялась по порядку до возникновения ошибки.

А можем ли мы сами бросать исключения?

До сих пор мы видели только, как JVM бросает исключение, если не может корректно исполнить какую-то операцию. Это касается и целочисленного деления на ноль, и выхода за пределы массива.

А если мы пишем свой код и тоже хотим бросить исключение, например, в собственном методе деления?

С чего мы начали:

```
1 public class MainClass {  
2     public static void main(String[] args) {  
3         int a = 0;  
4         int b = 10 / a;  
5     }  
6 }
```

А теперь мы хотим обернуть это в собственный метод и самостоятельно бросить исключение, если увидели в делителе 0:



```
1 public static int div(int a , int b) {  
2     if (b == 0) {  
3         throw new RuntimeException("Нельзя делить на 0");  
4     }  
5     return a / b;  
6 }
```

Чтобы бросить исключение, используется ключевое слово **throw** («бросить»), после которого идёт создание объекта типа `RuntimeException`. Почему именно `RuntimeException`, и какие ещё бывают типы исключений — на следующем занятии.



При исполнении этой строки будет брошено исключение, а метод прекратит свою работу и не сможет вернуть некорректное значение.

Посмотрим на последний пример и ответим на вопрос.

Какое исключение будет брошено в этом случае: одно или два?

```
1 public class MainApp {  
2     public static void main(String[] args) {  
3         int[] array = {1, 2, 3, 4, 5};  
4         int x = 10 / 0;  
5         System.out.println(array[1000]);  
6     }  
7 }
```

Будет брошено только одно исключение — `ArithmeticException`.

Что касается `ArrayIndexOutOfBoundsException`, то до строки 5 дело не дойдёт, так как при возникновении исключения оно будет брошено, не обработано, и долетит до стандартного обработчика исключений JVM, на чём работа приложения и завершится. Поэтому у вас никогда не возникнет ситуация с выбрасыванием одновременно нескольких исключений.



Заключение

Подведём итог и поговорим о том, что сегодня узнали:

1. Откуда берутся ошибки в приложениях.
2. Как приложение может оповещать об ошибках: посредством кодов ошибок и исключений.
3. Какие плюсы и минусы есть у каждого варианта.
4. Что собой представляют и как работают исключения.
5. Как бросать исключения в своём коде.

Следующий важнейший вопрос: «Как обрабатывать исключения», — разберём на следующей лекции.

Контрольные вопросы

1. Зачем для разных видов ошибок приложение возвращает разные коды ошибок или разные исключения?
2. В чём заключается неудобство использования кодов ошибок?
3. В чём смысл бросания исключений?
4. Что даёт распечатка стека вызовов, когда исключение долетает до стандартного обработчика?
5. Какие типы исключений вы знаете, и в каких случаях они возникают?