Язык программирования Java

ШАГ компьютерная АКАДЕМИЯ

Урок №8

Исключения

Содержание

1. Теория исключений

1.1 Что такое исключительная ситуация?

Исключение – это ошибка, которая возникает во время выполнения приложения.

В языке *Java* исключения являются объектами, которые являются наследниками *Throwable*. Существует ряд стандартных исключительных ситуаций, которые предопределены. В данном случае объекты исключений создаются автоматически, при возникновении исключительной ситуации. Также мы можем создавать собственные исключения, если наследуем базовый класс *Exception* или *RuntimeException* (исключим ошибки *Error*).

Исключения могут возникать во многих случаях, например: передача неправильного аргумента, обращение к объекту, который равен *null*, выход за пределы массива, ошибка при приведении типов (неверный формат) и т.д.

Обработка исключительных ситуаций (*exception handling*) – это механизм языков программирования, предназначенный для описания реакции программы на ошибки времени выполнения и другие возможные проблемы (*называемые исключениями*), которые могут возникнуть при выполнении программы и приводят к невозможности дальнейшей отработки программой ее базового алгоритма.

1.2 Типы исключений

Рассмотрим иерархию классов исключений:

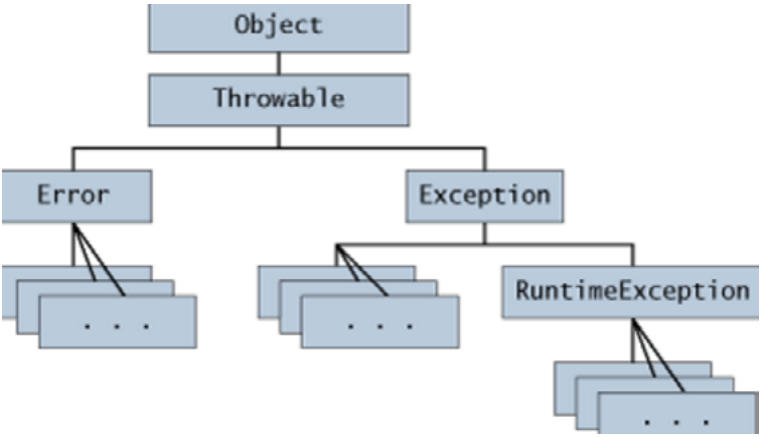

Рис. 1 Иерархия исключений

В языке *Java* присутствуют два типа исключений: *checked* и *unchecked*, а также – ошибки (*Error*), которые следует рассматривать отдельно.

Checked исключения обязательно должны обрабатываться блоком *catch* или описываться в сигнатуре метода (рассмотрим дальше), в отличие от *unchecked*, которые обрабатывать не обязательно, их можно избежать проверками. C*hecked* исключения являются наследниками класса *Exception* (исключая ветку *RuntimeException*).

Unchecked – это необязательные для обработки исключения, которые наследованные от *RuntimeException*.

Примеры исключений:

(*unchecked*) *NullPointerException*,

(*checked*) *IOException*.

NullPointerException возникает при обращении к неинициализированной переменной. Это исключение

можно предупредить, сделав проверку на *null*, при обращении к свойствам этого объекта. Исключение *IOException* возникает, например при открытии файла. Даже если файл существует, может произойти ряд причин, которые нельзя предугадать, поэтому это исключение обязательно необходимо обработать.

Если во время работы приложения возникли ошибки (*Error*), это означает наличие серьезных проблем. Большинство из этих ошибок влечет за собой ненормальный ход выполнения программы. Ошибки не рекомендуется отмечать методах посредством *throws*-объявления, поэтому они также очень часто называются не проверяемые (*unchecked*).

1.3 Блоки try и catch

Ключевое слово ***try*** используется для задания блока программного кода, который может спровоцировать исключительную ситуацию. Сразу после блока *try* должен располагаться блок ***catch***, задающий тип исключения, которого необходимо обрабатывать.

Синтаксис:

```
try{
  <блок кода, с возможной исключительной
  ситуацией>
}
catch (<тип и объект исключения, которое мы
отлавливаем>)
  { <блок кода действия, при возникновении
  исключения>
}
```

Catch можно разделить на несколько блоков для того, чтобы на каждое отдельное исключение, которые может произойти в блоке *try*, выполнять разные действия.

Пример исключительных ситуаций:

```
public class Main {
  static Object object;

  public static void foo() {
    System.out.println(object.toString());
    System.out.println(1 / 0);
  }

  public static void main(String[] args) {
    foo();
  }
}
```

В методе *foo*() написан код, который вызывает два исключения:

- *NullPointerException* – обращение к неинициализированному объекту;
- *ArithmeticException*: / *by zero* – деление на ноль.

Обработаем исключения:

```
public static void foo() {
  try {
    System.out.println(object.toString());
    System.out.println(1 / 0);
  } catch (Exception e) {
    System.out.println("Exception");
  }
}
```

Класс *Exception* является родителем для *NullPointerException* и *ArithmeticException*, поэтому независимо от того, где возникнет ошибка, мы попадем в блок *catch*. Следует обратить внимание, что если мы получим исключение в первой строке блока *try* , то дальше мы не пойдем, поэтому мы отловим только одно исключение.

```
public static void foo() {
  try {
    System.out.println(object.toString());
    System.out.println(1 / 0);
  } catch (NullPointerException e) {
    System.out.println("Объект равен null");
  } catch (ArithmeticException e) {
    System.out.println("На ноль делить нельзя");
  }
}
```

Записав блок *catch* в таком виде, мы можем определить различные действия, при том или ином исключении, но опять же, мы сможем обработать только одно исключение; если мы хотим однозначно обработать обе строки, тогда их следует вынести в разные блоки *try*:

```
public static void foo() {
  try {
    System.out.println(object.toString());
  } catch (NullPointerException e) {
    System.out.println("Объект равен null");
  }
  try {
    System.out.println(1 / 0);
```

```
  }catch (ArithmeticException e) {
    System.out.println("На ноль делить нельзя");
  }
}
```

1.4 Ключевое слово finally

Для объявления участка кода, который будет гарантированно выполняться, независимо от того, какие исключения были возбуждены и перехвачены, необходимо использовать блок *finally*.

Этот блок отработает при успешном выполнении блока *try* или при обработке исключения; в обоих случаях блок *finally* будет выполнен после *try-catch* и до того, как управление перейдет к операторам, следующим после *try*. Блок *finally* очень удобен для закрытия файлов и освобождения любых ресурсов, захваченных для временного использования в начале выполнения метода.

1.5 Ключевое слово throw

Оператор *throw* используется для того, чтобы программно возбудить исключение. Для того чтобы сделать это, нужно иметь объект исключения (который можно получить в блоке *catch*, либо создать с помощью оператора *new*).

Пример:

```
throw new NullPointerException();
```

При достижении этого оператора выполнение кода прекращается, поэтому следующий за ним оператор не выполнится. Вы можете возбуждать исключения в соб-

ственных методах, если, к примеру, входные данные не проходят валидацию и метод может работать с ними некорректно.

1.6 Ключевое слово throws

Ключевое слово *throws* в методе явно задает те исключения, которые необходимо обработать при вызове этого метода. Его следует применять, если метод способен возбуждать исключения, которые он сам не обрабатывает.

Для определения списка исключений, которые могут возбуждаться методом, используется ключевое слово *throws* и список исключений через запятую, пример:

```
public class Main {
  public static char getCharFromString(String str, int
  index) throws IllegalArgumentException,
  IndexOutOfBoundsException {
    if (index < 0)
      throw new IllegalArgumentException();
    return str.toCharArray()[index];
  }

  public static void main(String[] args) {
    try {
      System.out.println(getCharFromString
      ("hello", 20));
      } catch (IllegalArgumentException e) {
      System.out.println("Индекс не может
      быть меньше нуля");
    } catch (IndexOutOfBoundsException e) {
      System.out.println("Вы вышли за пределы массива");
    }
  }
}
```

Урок № 8
Исключения