Практическая работа № 18. Исключения и работа с ними в Джава

Цель данной практической работы являются получение практических навыков разработки программ, изучение синтаксиса языка Java, освоение основных конструкций языка Java (циклы, условия, создание переменных и массивов, создание методов, вызов методов), а также научиться осуществлять стандартный ввод/вывод данных.

Ключевые слова: try, catch, finally, throw, throws

Теоретические сведения

Механизм исключительных ситуаций в Java поддерживается пятью ключевыми словами:

- try
- catch
- finally
- throw
- throws

В языке Джава Java всего около 50 ключевых слов, и пять из них связано как раз с исключениями, это- try, catch, finally, throw, throws.

Из них catch, throw и throws применяются к экземплярам класса, причём работают они только с Throwable и его наследниками.

На рис. 18.1 представлена иерархия классов исключений,

используемая в Java

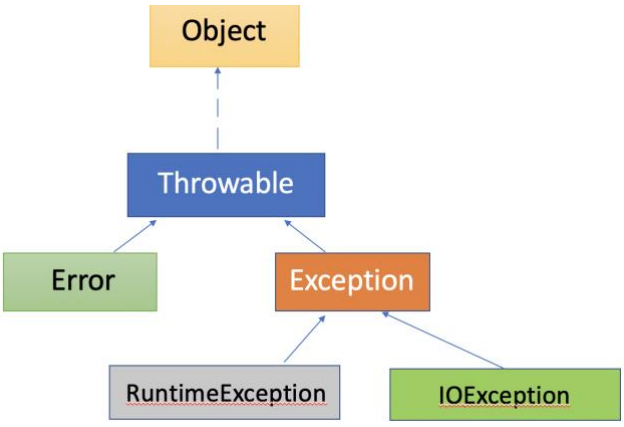

Рисунок 18.1. Иерархия классов исключений в Джава

Наиболее популярные исключений в Java представлены в таблице 1.

Таблица 1. Классы исключений в Java

№ пп	Класс исключения	Класс предок/тип
1.	ArithmeticException	RuntimeException
2.	NegativeArraySizeException	RuntimeException
3.	ArrayIndexOutOfBoundsException	RuntimeException
4.	NoSuchElementException	RuntimeException
5.	ArrayStoreException	RuntimeException
6.	NotSerializableException	Exception
7.	AssertionError	Error
8.	NullPointerException	RuntimeException
9.	ClassCastException	RuntimeException
10.	NumberFormatException	RuntimeException
11.	ClassNotFoundException	Exception
12.	OutOfMemoryError	Error
13.	CloneNotSupportedException	Exception
14.	SecurityException	RuntimeException
15.	ConcurrentModificationException	RuntimeException
16.	StackOverflowError	Error
17.	EOFException	Exception
18.	StringIndexOutOfBoundsException	RuntimeException
19.	FileNotFoundException	Exception
20.	ThreadDeath	Error
21.	IllegalArgumentException	RuntimeException
22.	UnsupportedEncodingException	Exception
23.	InterruptedException	Exception
24.	UnsupportedOperationException	RuntimeException

То, что исключения являются объектами важно по двум причинам:

1) они образуют иерархию с корнем java.lang.Throwable (java.lang.Object — предок java.lang.Throwable, но Object —это не исключение!)
2) они могут иметь поля и методы

По первому пункту: catch — полиморфная конструкция, т.е. catch по типу класса родителя перехватывает исключения для экземпляров объектов как родительского класса, так или его наследников(т.е. экземпляры непосредственно самого родительского класса или любого его потомка).

Листинг 18.1 Пример обработки исключения

```
public class App {
    public static void main(String[] args) {
        try {
```

```
                System.err.print(" 0");
                if       (true)       {throw       new
RuntimeException();}
                System.err.print(" 1");
            } catch (Exception e) {   // catch по
Exception ПЕРЕХВАТЫВАЕТ RuntimeException
                  System.err.print(" 2");
               }
            System.err.println(" 3");
         }// end main
    }
```

Результат работы программы: представленной на листинге 18.1:

```
>> 0 2 3
```

Задания на практическую работу № 18 (Основы Try-Catch- Finally)

Задание № 1

Шаг 1. Выполните следующую программу и посмотрите, что происходит:

Листинг 18.2 Пример обработки деления на ноль

```
public class Exception1 {
    public void exceptionDemo() {
      System.out.println( 2 / 0 );
   }
}
```

Описание работы

Вам необходимо инстанцировать класс и выполнить exceptionDemo(). Что произойдет?

Ответ: программа даст сбой, и вы получите следующее сообщение

java.lang.ArithmeticException: / by zero at Exception1.exceptionDemo(Exception1.java:12)

Это говорит нам о том, что программа пытается выполнить деление на ноль, который он не в состоянии выполнить.

Объясните поведение программы.

Шаг 2. Измените программу следующим образом.

Замените 2/0 на 2,0 / 0,0 и повторно вызовите метод. Что произойдет?

Теперь измените код в классе Exception1 и включите блок try-catch следующим образом:

Листинг 18.3 Пример обработки исключения

```
public class Exception1 {
    public void exceptionDemo() {
       try{
           System.out.println( 2/0 );
           } catch ( ArithmeticException e ) {
System.out.println("Attempted division by zero");
           }
     }
}
```

Шаг 3. Запустите программу и обратите внимание на новое поведение.

Объясните поведение программы.

Задание № 2

Шаг 1.Измените код в листинге 18.3 на следующий:

Листинг 18.4 Пример программы

```
public class Exception2 {
public void exceptionDemo() {
Scanner myScanner = new Scanner( System.in);
System.out.print( "Enter an integer ");
String intString = myScanner.next();
int i = Integer.parseInt(intString);
System.out.println( 2/i );
       }
}
```

Шаг 2. Запустите эту программу со следующими выводами: Qwerty 0 1.2 1. Посмотрите на вывод.

Объясните какие исключения выбрасываются?

Шаг 3. Измените код, добавив блоки try – catch, чтобы иметь дело с определяемыми исключениями.

Объясните поведение программы

Задание № 3

С помощью перехватывания исключений можно оказывать влияние на поведение программы. В вашем решении в предыдущем упражнении вы можете добавить новый пункт - catch в начале списка пунктов catch.

Шаг 1. Выполите это действие, чтобы поймать общее исключение класса Exception.

Шаг 2. Перезапустите программу с приведенными выше данными и обратите внимание на ее поведение.

Объясните новое поведение программы

Задание № 4

Шаг 1. Добавьте блок finally к решению Задания №2.

Шаг 2. Повторно запустите программу, чтобы проверить ее поведение. Объясните новое поведение программы

Генерация собственных исключений

На предыдущем шаге при выполнении заданий мы рассмотрели, как Java работает с предопределенными исключениями, теперь перейдем к тому, как генерируется исключение.

Все исключения в рассмотренных ранее примерах и заданиях были определены заранее. В этом разделе практичнской работы вы будете создавать и пробрасывать свои собственные исключения (exceptions).

Задание № 5

Самый простой способ генерации исключения показан в следующем примере кода:

Листинг 18.5. Класс ThrowsDemo

```
public class ThrowsDemo {
     public void getDetails(String key) {
           if(key == null) {
throw   newNullPointerException("null   key   in
getDetails" );
               }
            // do something with the key
          }
}
```

Шаг 1. Когда определяется условие ошибки, то мы выбрасываем исключение с определенным именем. Сообщение может быть связано с исключением. Откомпилируйте этот класс, создайте его экземпляр и выполните метод getDetails() с нулем в качестве значения параметра.

Вы можете получить следующийвывод:

java.lang.NullPointerException: null key in getDetails at ThrowsDemo.getDetails(ThrowsDemo.java:13)

Шаг 2. Добавьте блок try-catch таким образом, чтобы перехватить исключение и рассмотреть его внутри метода.

Подумайте, является ли этот способ подходящим, чтобы иметь дело с этим исключением?

Объясните поведение программы.

Ответ. Причиной ошибки, может является, например неправильное значение для параметра. Может было бы лучше, если бы метод вызывал getDetails() и там решалась бы эта проблема.

Обратите внимание на следующее:

Листинг 18.6 Пример видоизмененной программы ThrowsDemo

```
public class ThrowsDemo {
     public void printMessage(String key) {
        String message = getDetails(key);
System.out.println( message );
      }
public String getDetails(String key) {
    if(key == null) {
throw new NullPointerException( "null key in
getDetails" );
      }
       return "data for" + key; }
}
```

Задание № 6

Шаг 1. Откомпилируйте и запустите эту программу с правильным значением для ключа и с нулем в качестве значения. При выполнении с нулевым значением вы увидите некоторый вывод.

Шаг 2. Обобщите все вышесказанное и выполните эту программу с правильным значением для ключа и с нулем в ключе.

java.lang.NullPointerException: null key in getDetails

at ThrowsDemo.getDetails(ThrowsDemo.java:21) at ThrowsDemo.printMessage(ThrowsDemo.java:13)

Шаг 3. Теперь добавьте блоки try-catch, чтобы использовать для вывода сообщений метод printMessage(), таким образом, чтобы исключения обрабатывались и программа не "ломалась".

Объясните ее поведение.

Задание № 7

Теперь мы расширим наш пример для демонстрации прохождения исключения через цепочку вызовов.

Листинг 18.7 класс ThrowsDemo

```
public class ThrowsDemo {
public void getKey() {
      Scanner myScanner = new Scanner( System.in
);
      String key = myScanner.next();
      printDetails( key );
   }
   public void printDetails(String key) {
      try {  String message = getDetails(key);
               System.out.println( message );
              }catch ( Exception e){
                 throw e;
              }
    }
   private String getDetails(String key) {
     if(key          ==          "")          {
                     throw new Exception( "Key set
to empty string" );
      }
   return "data for " + key; }
}
```

Шаг 1. Создайте следующий класс (листинг 18.7) и попытайтесь его скомпилировать.

При попытке компиляции вы получите следующий синтаксис ошибки:

Исключение Unreported java.lang.Exception

В результате успешного пробрасывания исключение должен быть поймано или объявлено. Объясните причину.

Ответ. Причиной полученной ошибки является выражение ***throw e.***

Пояснение. В данном случае метод printDetails () решил, что он не может иметь дело с исключением и проходит все дерево его вызовов. Поскольку метод getKey() не имеет блока try-catch для обработки исключений, то Java становится перед выбором, что в таком случае делать.

Проблему можно решить несколькими способами:

1) Добавьте соответствующие try-catch блоки таким образом, чтобы в конечном итоге один из них обрабатывал исключение;
2) Удалите блоки try-catch для всех методов, кроме одного, который обрабатывает исключение. Добавьте throws, котрый бросает исключение методу, который проходит исключение без обработки.

Задание №8

Шаг 1. Измените код из предыдущего примера следующим образом:

1. Удалите throws Exception из метода getKey().
2. Измените метод getKey(), добавив try-catch блок для обработки исключений.
3. Добавьте цикл к getKey() таким образом, чтобы пользователь получил еще один шанс на ввод значение ключа

Замечания: Инструкция throw очень аналогична инструкции return – она прекращает выполнение метода, дальше мы не идем. Если мы нигде не ставим catch, то у нас выброс Exception очень похож на System.exit() – система завершает процесс. Но мы в любом месте можем поставить catch и, таким образом, предотвратить поломку кода.

Выводы*:*

Фактически при работе с исключениями весь материал делится на две части: синтаксис (ответ на вопрос, что компилятор пропустит, а что нет) и семантика (вопрос, как лучше делать) исключений. В отличие от вариантов с for, while, switch, использование исключений – более сложный механизм. Но он сложен не синтаксически, а семантически, по своему подходу. То есть при генерации исключений нужно думать о том - не как правильно его использовать, и с каким умыслом его использовать. То есть вопрос стоит так, в каких ситуациях стоит ли ломать систему и где, а в каких ситуациях ее восстанавливать.

В хорошей инженерной системе каждый любой модуль всегда проверяет все входные данные.

Можно еще сказать что существует иерархия различных способов прекращения выполнения некоего действия (или ряда действий) в виде участка кода и эта иерархия классифицирует возможные действия по мощности используемого оператора: continue, break, return, throw.

- **continue** прекращает выполнение данной итерации в цикле;

- **break** прекращает выполнение данного блока (например цикла);
- **return** – это инструкция выхода из данного метода (например прекращение выполнения функции);
- **throw** – еще более сильная инструкция, она прекращает выполнение данного метода и метода, который его вызвал. Так-как исключения вообще-то позволяют “сломать” весь стек работы программы.