Частное учреждение образования
«Колледж бизнеса и права»

УТВЕРЖДАЮ
Ведущий
методист колледжа
_____Е.В. Паскал
«___»_____2021

Специальность: 2-40 01 01 «Программное обеспечение информационных технологий»	Учебная дисциплина: «Основы кроссплатформенного программирования»

Лабораторная работа № 7
Инструкционно-технологическая карта

Тема: «Обработка исключений»

Цель: Научиться работать с исключениями

Время выполнения: 4 часа

ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

1. Изучить теоретические сведения;
2. Ответить на контрольные вопросы;
3. Откомпилировать примеры программ из раздела «Теоретические сведения»;
4. Выполнить ИДЗ.

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ ДЛЯ ВЫПОЛНЕНИЯ РАБОТЫ

Исключение — некая исключительная, незапланированная ситуация, которая произошла при работе программы. Примеров исключений в Java может быть много. Например, ты написал код, который считывает текст из файла и выводит в консоль первую строку.

```
public class Main {

  public static void main(String[] args) throws IOException {
    BufferedReader reader = new BufferedReader(new FileReader("C:\\Users\\Username\\Desktop\\test.txt"));
    String firstString = reader.readLine();
    System.out.println(firstString);
  }
}
```

Но такого файла не существует! Результатом работы программы будет исключение — FileNotFoundException. Вывод:

Exception in thread "main" java.io.FileNotFoundException: C:\Users\Username\Desktop\test.txt (Системе не удается найти указанный путь)

Каждое исключение представлено в Java отдельным классом. Все классы исключений происходят от общего “предка” — родительского класса Throwable. Название класса-исключения обычно коротко отображает причину его возникновения:

- FileNotFoundException (файл не найден)
- ArithmeticException (исключение при выполнении математической операции)
- ArrayIndexOutOfBoundsException (указан номер ячейки массива за пределами его длины). Например, если попытаться вывести в консоль ячейку array[23] для массива array длиной 10.

Всего таких классов в Java почти 400 штук! Зачем так много? Именно для того, чтобы программистам было удобнее с ними работать. Представьте себе: вы написали программу, и она при работе выдает исключение, которое выглядит вот так:

Exception in thread "main"

Ничего не понятно. Что за ошибка, откуда взялась — неясно. Никакой полезной информации нет. А вот благодаря такому разнообразию классов программист получает для себя главное — тип ошибки и ее вероятную причину, которая заложена в названии класса. Ведь совсем другое дело увидеть в консоли:

Exception in thread "main" java.io.FileNotFoundException: C:\Users\Username\Desktop\test.txt (Системе не удается найти указанный путь)

Сразу становится понятно, в чем может быть дело и “в какую сторону копать” для решения проблемы! Исключения, как и любые экземпляры классов, являются объектами.

Кратко о ключевых словах try, catch, finally, throws

Обработка исключений в Java основана на использовании в программе следующих ключевых слов:

- try – определяет блок кода, в котором может произойти исключение;
- catch – определяет блок кода, в котором происходит обработка исключения;
- finally – определяет блок кода, который является необязательным, но при его наличии выполняется в любом случае независимо от результатов выполнения блока try.

Эти ключевые слова используются для создания в программном коде специальных обрабатывающих конструкций: try{}catch, try{}catch{}finally, try{}finally{}.

- throw – используется для возбуждения исключения;

- throws – используется в сигнатуре методов для предупреждения, о том что метод может выбросить исключение.

Пример использования ключевых слов в Java-программе:

```
//метод считывает строку с клавиатуры

public String input() throws MyException {//предупреждаем с помощью throws,
// что метод может выбросить исключение MyException
    BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));
   String s = null;
//в блок try заключаем код, в котором может произойти исключение, в данном
// случае компилятор нам подсказывает, что метод readLine() класса
// BufferedReader может выбросить исключение ввода/вывода
   try {
      s = reader.readLine();
// в блок catch заключаем код по обработке исключения IOException
   } catch (IOException e) {
      System.out.println(e.getMessage());
// в блоке finally закрываем поток чтения
   } finally {
// при закрытии потока тоже возможно исключение, например, если он не был открыт, поэтому “оборачиваем” код в блок try
      try {
         reader.close();
// пишем обработку исключения при закрытии потока чтения
      } catch (IOException e) {
         System.out.println(e.getMessage());
      }
   }

   if (s.equals("")) {
// мы решили, что пустая строка может нарушить в дальнейшем работу нашей программы, например, на результате этого метода нам надо вызывать метод substring(1,2), поэтому мы вынуждены прервать выполнение программы с генерацией своего типа исключения MyException с помощью throw
      throw new MyException("String can not be empty!");
   }
   return s;
}
```

Создание исключения

При исполнении программы исключение генерируется JVM или вручную, с помощью оператора throw. При этом в памяти создается объект исключения и выполнение основного кода программы прерывается, а обработчик исключений JVM пытается найти способ обработать исключение.

Обработка исключения

Создание блоков кода, для которых мы предусматриваем обработку исключений в Java, производится в программе с помощью конструкций try{}catch, try{}catch{}finally, try{}finally{}.

При возбуждении исключения в блоке try обработчик исключения ищется в следующем за ним блоке catch. Если в catch есть обработчик данного типа исключения – управление переходит к нему. Если нет, то JVM ищет обработчик этого типа исключения в цепочке вызовов методов до тех пор, пока не будет найден подходящий catch. После выполнения блока catch управление передается в необязательный блок finally. В случае, если подходящий блок catch не найден, JVM останавливает выполнение программы, и выводит стек вызовов методов – stack trace, выполнив перед этим код блока finally при его наличии. Пример обработки исключений:

```
public class Print {

    void print(String s) {
      if (s == null) {
         throw new NullPointerException("Exception: s is null!");
      }
      System.out.println("Inside method print: " + s);
    }

    public static void main(String[] args) {
      Print print = new Print();
      List list= Arrays.asList("first step", null, "second step");

      for (String s:list) {
         try {
            print.print(s);
         }
         catch (NullPointerException e) {
            System.out.println(e.getMessage());
            System.out.println("Exception was processed. Program continues");
         }
         finally {
            System.out.println("Inside block finally");
         }
         System.out.println("Go program....");
         System.out.println("-----------------");
      }

    }
    }
```

Результаты работы метода main:

```
Inside method print: first step
Inside block finally
Go program....
-----------------
Exception: s is null!
Exception was processed. Program continues
```

```
Inside block finally
Go program....
----------------
Inside method print: second step
Inside block finally
Go program....
----------------
```

Блок finally обычно используется для того, чтобы закрыть открытые в блоке try потоки или освободить ресурсы. Однако при написании программы не всегда возможно уследить за закрытием всех ресурсов. Для облегчения нашей жизни разработчики Java предложили нам конструкцию try-with-resources, которая автоматически закрывает ресурсы, открытые в блоке try. Наш первый пример можно переписать так с помощью try-with-resources:

```
public String input() throws MyException {
   String s = null;
   try(BufferedReader reader = new BufferedReader(new InputStreamReader(System.in))){
      s = reader.readLine();
   } catch (IOException e) {
      System.out.println(e.getMessage());
   }
   if (s.equals("")){
      throw new MyException ("String can not be empty!");
   }
   return s;
}
```

Благодаря возможностям Java, начиная с версии 7, мы также можем объединять перехват разнотипных исключений в одном блоке, делая код более компактным и читабельным. Например:

```
public String input() {
   String s = null;
   try (BufferedReader reader = new BufferedReader(new InputStreamReader(System.in))) {
      s = reader.readLine();
      if (s.equals("")) {
         throw new MyException("String can not be empty!");
      }
   } catch (IOException | MyException e) {
      System.out.println(e.getMessage());
   }
   return s;
}
```

Итоги

Использование исключений в Java позволяет повысить отказоустойчивость программы за счет использования «запасных» путей, отделить логику основного кода от кода обработки исключительных ситуаций за счет использования блоков catch, а также дает нам возможность

переложить обработку исключений на пользователя нашего кода с помощью throws.

Встроенные исключения Java

Существуют несколько готовых системных исключений. Большинство из них являются подклассами типа RuntimeException и их не нужно включать в список throws. Вот небольшой список непроверяемых исключений.

- ArithmeticException - арифметическая ошибка, например, деление на нуль
- ArrayIndexOutOfBoundsException - выход индекса за границу массива
- ArrayStoreException - присваивание элементу массива объекта несовместимого типа
- ClassCastException - неверное приведение
- EnumConstantNotPresentException - попытка использования неопределённого значения перечисления
- IllegalArgumentException - неверный аргумент при вызове метода
- IllegalMonitorStateException - неверная операция мониторинга
- IllegalStateException - некорректное состояние приложения
- IllegalThreadStateException - запрашиваемая операция несовместима с текущим потоком
- IndexOutofBoundsException - тип индекса вышел за допустимые пределы
- NegativeArraySizeException - создан массив отрицательного размера
- NullPointerException - неверное использование пустой ссылки
- NumberFormatException - неверное преобразование строки в числовой формат
- SecurityException - попытка нарушения безопасности
- StringIndexOutOfBounds - попытка использования индекса за пределами строки
- TypeNotPresentException - тип не найден
- UnsupportedOperationException - обнаружена неподдерживаемая операция

Список проверяемых системных исключений, которые можно включать в список throws.

- ClassNotFoundException - класс не найден
- CloneNotSupportedException - попытка клонировать объект, который не реализует интерфейс Cloneable
- IllegalAccessException - запрещен доступ к классу
- InstantiationException - попытка создать объект абстрактного класса или интерфейса
- InterruptedException - поток прерван другим потоком

- NoSuchFieldException - запрашиваемое поле не существует
- NoSuchMethodException - запрашиваемый метод не существует
- ReflectiveOperationException - исключение, связанное с рефлексией

Создание собственных классов исключений

Система не может предусмотреть все исключения, иногда вам придётся создать собственный тип исключения для вашего приложения. Вам нужно наследоваться от Exception (напомню, что этот класс наследуется от Trowable) и переопределить нужные методы класса Throwable. Либо вы можете наследоваться от уже существующего типа, который наиболее близок по логике с вашим исключением.

- final void addSuppressed(Throwable exception) - добавляет исключение в список подавляемых исключений (JDK 7)
- Throwable fillInStackTrace() - возвращает объект класса Throwable, содержащий полную трассировку стека.
- Throwable getCause() - возвращает исключение, лежащее под текущим исключение или null
- String getLocalizedMessage() - возвращает локализованное описание исключения
- String getMessage() - возвращает описание исключения
- StackTraceElement[] getStackTrace() - возвращает массив, содержащий трассировку стека и состояний из элементов класса StackTraceElement
- final Throwable[] getSuppressed() - получает подавленные исключения (JDK 7)
- Throwable initCause(Throwable exception) - ассоциирует исключение с вызывающим исключением. Возвращает ссылку на исключение.
- void printStackTrace() - отображает трассировку стека
- void printStackTrace(PrintStream stream) - посылает трассировку стека в заданный поток
- void printStackTrace(PrintWriter stream) - посылает трассировку стека в заданный поток
- void setStackTrace(StackTraceElement elements[]) - устанавливает трассировку стека для элементов (для специализированных приложений)
- String toString() - возвращает объект класса String, содержащий описание исключения.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что такое исключение?
2. Для чего используется try?
3. Для чего используется catch?
4. Для чего используется, throws?

5. Для чего используется finally?
6. Как происходит обработка исключений?
7. Перечислите встроенные исключения.

ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

Задание 1.

Для своего варианта лабораторной работы № 5 оптимизировать программу, включив в нее обработку исключительных ситуаций: ввода данных, операции деления и др.

Вариант 1. Задание 1
Вариант 2. Задание 2
Вариант 3. Задание 3

Задание 2.

Смоделировать структуру предприятия:

Классы	Свойства и методы
Фирма	название (get, set)
Отдел	название (get, set) количество сотрудников (get, set)
Сотрудник	фио (get, set) должность (get, set) оклад (get, set) рассчитать зарплату()
Штатный сотрудник	премия (get, set) рассчитать зарплату()
Сотрудник по контракту	рассчитать зарплату()

а) Обработать все Exception с помощью блока try…catch(Exception …) в методе «рассчитать зарплату» классов Штатный сотрудник и Сотрудник по контракту. При возникновении Exception выводить на экран сообщение об ошибке.

Описать собственный класс Exception PremiyaException. В методе «рассчитать зарплату» класса Штатный сотрудник выбрасывать собственное Exception типа PremiyaException при отрицательном значении свойства Премия. В этом же методе обработать PremiyaException в блоке catch. При возникновении Exception выводить сообщение об ошибке на экран.

В основной программе проверить работу блока обработки Exception метода «рассчитать зарплату».

б) Описать собственный класс Exception OkladException. В конструкторе реализовать проверку значения оклада, при отрицательном значении выбрасывать собственное Exception типа OkladException. Обработать Exception OkladException с помощью блока try…catch, в блоке обработки Exception вывести на экран сообщение «Невозможно создать сотрудника – указан отрицательный оклад: <оклад>» и повторно создать Exception.

В основной программе (main) обработать вызов конструктора класса Сотрудник и проверить работу обработчика Exception.

ДОМАШНЕЕ ЗАДАНИЕ

Индивидуальное задание

ЛИТЕРАТУРА

У. Савитч, язык Java. Курс программирования, Вильямс, 2017

Преподаватель А.С.Кибисова

Рассмотрено на заседании цикловой комиссии
программного обеспечения информационных технологий
Протокол № ____ от «___» _____ 2021
Председатель ЦК _____В.Ю.Михалевич