гр. София, пл. Славейков 11 ет.5 тел. : +359 897 91 93 96

office@progressbg.net www.progressbg.net

Компютърно образователен център Progress Палитра от възможности!

Лекция 8

Прихващане и обработка на изключения, вход-изход към файл – 1-ва част

- **Прихващане и обработка на изключения**
- **Приложение на try-catch**
- **Входни и изходни потоци към файл**
- **Примери**
- **Задачи**

гр. София, пл. Славейков 11 ет.5 тел. : +359 897 91 93 96

office@progressbg.net www.progressbg.net

Въведение

Изключение

Изключение (exception) в общия случай е уведомление за дадено събитие, нарушаващо нормалната работа на една програма. Изключенията дават възможност това необичайно събитие да бъде обработено и програмата да реагира по някакъв начин. Когато възникне изключение конкретното състояние на програмата се запазва и се търси **обработчик на изключението (exception handler)**.

Изключенията се предизвикват или **"хвърлят" (throw an exception)**.

Прихващане и обработка на изключения

Exception handling (**инфраструктура за обработка на изключенията**) е част от средата – механизъм, който позволява хвърлянето и прихващането на изключения. Част от тази инфраструктура са дефинираните езиковите конструкции за хвърляне и прихващане на изключения. Тя се грижи и затова изключението да стигне до кода, който може да го обработи.

Изключенията в Java

Изключение (exception) в Java представлява събитие, което уведомява програмиста, че е възникнало обстоятелство (грешка) непредвидено в нормалния ход на програмата. Това става като методът, в който е възникнала грешката изхвърля специален обект съдържащ информация за вида на грешката, мястото в програмата, където е възникнала, и състоянието на програмата в момента на възникване на грешката.

Синтаксис на try-catch :

```
try
  {
    //Тук се изпълнява някакъв код
  }
 catch(Exception e)
  {
    //Тук се поправят гпешките
  }
```

Пример за код, който хвърля изключения

Пример 1:

```
1  /*
2   * To change this template, choose Tools | Templates
3   * and open the template in the editor.
4   */
5
6  /**
7   *
8   * @author Vankov
9   */
10 public class MathException {
11
12     public static void main(String args[]) {
13
14         int a = 5;
15         int b = 0;
16
17         try {
18             int result = a / b;
19             System.out.println(result);
20
21         } catch (java.lang.ArithmeticException ex) {
22             System.out.println(ex.getMessage());
23             ex.printStackTrace();
24         }
25     }
26 }
27
```

Пример 2:

```
1
2
3   /*
4    * To change this template, choose Tools | Templates
5    * and open the template in the editor.
6    */
7   /**
8    *
9    * @author Vankov
10   */
11  public class NullPointerException {
12
13      public static void main(String args[]) {
14
15          String str = null;
16          try {
17              int a = str.length();
18
19          } catch (java.lang.NullPointerException ex) {
20              ex.printStackTrace();
21          }
22      }
23  }
24
```

Stack Trace

Информацията, която носи т. нар. **Stack trace**, съдържа подробно описание на естеството на изключението и за мястото в програмата, където то е възникнало. Stack trace се използва, за да се намерят причините за възникването на изключението и последващото им отстраняване (довеждане до нормалното изпълнение на програмата). Stack trace съдържа голямо количество информация и е предназначен за анализиране само от програмистите и администраторите, но не и от крайните потребители на програмата, които не са длъжни да са технически лица. Stack trace е стандартно средство за търсене и отстраняване (дебъгване) на проблеми.

Stack Trace – пример

Ето как изглежда stack trace на изключение за липсващ файл. Подали сме несъществуващ файл **`C:\missingFile.txt`** и вместо да изведем съобщението сме използвали метода **`e.printStackTrace()`**.

```
java.io.FileNotFoundException: C:\missingFile.txt (The system cannot find
the file specified)
      at java.io.FileInputStream.open(Native Method)
      at java.io.FileInputStream.<init>(Unknown Source)
      at java.io.FileInputStream.<init>(Unknown Source)
      at ReadFile.readFile(ReadFile.java:12)
      at ReadFile.main(ReadFile.java:35)
```

Системата не може да намери този файл и затова хвърля изключението **`FileNotFoundException`**.

Как да разчетем "Stack Trace"?

За да се ориентираме в един stack trace трябва да можем да го разчетем правилно и да знаем неговата структура.

Stack trace съдържа следната информация в себе си:

- пълното име на класа на изключението;
- съобщение – информация за естеството на грешката;
- информация за стека на извикване на методите.

От примера по-горе пълното име на изключението е **`java.io.FileNotFoundException`**. Следва съобщението за грешка. То донякъде повтаря името на самото изключение: "**`C:\missingFile.txt (The system cannot find the file specified)`**". Следва целият стек на извикване на методите. Стека най-често е най-голямата част от stack trace.

Всички методи от стека на извикванията са показани на отделен ред. Най-отгоре е методът, който първоначално е хвърлил изключение, а най-отдолу е **`main()`** методът. Всеки метод се дава заедно с класа, който го съдържа и в скоби реда от файла, където е хвърлено изключението, примерно **`ReadFile.readFile(ReadFile.java:12)`**. Редовете са налични само ако класът е компилиран с опция да включва дебъг информация (номерата на редовете и т.н.).

Ако методът е конструктор, то вместо името му се използва **`<init>`** **`java.io.FileInputStream.<init>(Unknown Source)`**. Ако липсва информация за номера на реда, където е възникнало изключението се изписва **`Unknown Source`**. Ако методът е **`native`** (външен за Java виртуалната машина), се изписва **`Native Method`**.

Това позволява бързо и лесно да се намери класа, метода и дори реда, където е възникнала грешката, да се анализира нейното естество и да се поправи.

Писане и четене от и към файл(файлови потоци)

Потоци

Потоците (streams) са важна част от всяка входно-изходна библиотека. Те намират своето приложение, когато програмата трябва да "прочете" или "запише" данни от или във външен източник на данни – файл, други компютри, сървъри и т.н.

Преди да продължим е важно да уточним, че терминът **вход (input)** се асоциира с четенето на информация, а терминът **изход (output)** – със записването на информация.

Потоци в Java

В Java класовете за работа с потоци се намират в пакета **`java.io`**. Сега ще се концентрираме върху тяхната йерархия и организация.

Можем да отличим два основни типа потоци – такива, които работят с двоични данни и такива, които работят с текстови данни. Ще се спрем на основните характеристики на тези два вида след малко.

Общото между тях е организацията и структурирането им. На върха на йерархията стоят абстрактни класове съответно за вход и изход. Те няма как да бъдат инстанцирани, но дефинират основната функционалност, която притежават всички останали потоци. Съществуват и буферирани потоци, които не добавят никаква допълнителна функционалност, но позволяват работата с буфер при четене и записване на информацията, което значително повишава бързодействието. Буферираните потоци няма да се разглеждат в тази глава, тъй като ние се концентрираме върху обработката на текстови файлове. Ако имате желание, може да се допитате до богатата документация, достъпна в Интернет, или към някой учебник за по-напреднали в програмирането.

Основните класове в пакета **`java.io`** са **`InputStream`**, **`OutputStream`**, **`BufferedInputStream`**, **`BufferedOutputStream`**, **`DataInputStream`**, **`DataOutputStream`**, **`Reader`**, **`Writer`**, **`BufferedReader`**, **`BufferedWriter`**, **`PrintWriter`** и **`PrintStream`**. Сега ще се спрем по-обстойно на тях, разделяйки ги по основния им признак – типа данни, с които работят.

Четене на текстов файл ред по ред – пример:

```
1
2   import java.io.BufferedReader;
3   import java.io.FileInputStream;
4   import java.io.IOException;
5   import java.io.InputStreamReader;
6
7   /**
8    *
9    * @author Vankov
10   */
11  public class ReadFileContentProgram {
12
13      public static void main(String args[]) {
14          FileInputStream fis = null;
15          try {
16              String fileName = "test.txt";
17              fis = new FileInputStream(fileName);//init a FileInputStream object
18              BufferedReader in = new BufferedReader(new InputStreamReader(fis));//creates BufferedReader which constructor requres InputStreamReader
19              String tmp = null;
20              while ((tmp = in.readLine()) != null) {//reads all lines from the file whie the result from readLine() is different from null
21                  System.out.println(tmp);
22              }
23              in.close();//closes FileInputStream
24              fis.close();//closes BufferedReader
25          } catch (java.io.IOException ex) {
26              ex.printStackTrace();//in case of IOException - print stack trace
27          } finally {
28              try {
29                  fis.close();
30              } catch (IOException ex) {
31                  ex.printStackTrace();
32              }
33          }
34      }
35  }
```

Писане на масив от низове във файл :

```
import java.io.BufferedWriter;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.OutputStreamWriter;
/**
 * @author Vankov
 */
public class WriteToFileProgram {

    public static void main(String args[]) {
        FileOutputStream fout = null;
        String arrFileContent[] = {"First line", "Second line", "Third line",
        "Our last line which we will write to the file.");
        try {
            String fileName = "test.txt";
            fout = new FileOutputStream(fileName);//init a FileOutputStream object
            BufferedWriter writer = new BufferedWriter(new OutputStreamWriter(fout));//creates BufferedWriter which constructor requres OutputStreamWriter

            for (int i = 0; i < arrFileContent.length; i++) {
                writer.write(arrFileContent[i]+"\n");//writes current String to the file with new line
            }
            writer.flush();//execute always before closing the stream
            fout.close();//closes FileOutputStream
            writer.close();//closes BufferedWriter
        } catch (java.io.IOException ex) {
            ex.printStackTrace();//in case of IOException - print stack trace
        } finally {
            try {
                fout.close();
            } catch (IOException ex) {
                ex.printStackTrace();
            }
        }
    }
```