Laborator 7:

Tratarea excepţiilor în Java

Întocmit de: Adina Neculai

Îndrumător: Asist. Drd. Gabriel Danciu

13 noiembrie 2011

I. NOŢIUNI TEORETICE

A. Ce este o excepţie?

O *excepţie* este un eveniment care apare în execuţia unui program şi care întrerupe evoluţia normală a acestuia.

Încercarea de a soluţiona aceste excepţii folosind metode clasice duce la creşterea semnificativă a complexităţii codului, ceea ce afectează, în mod indirect, corectitudinea codului şi claritatea acestuia. De aceea, cei care au creat limbajul Java s-au gândit la un sistem de tratare a excepţiilor ce permite programatorului:

- tratarea excepţiilor la un nivel superior celui în care apar;
- propagarea excepţiilor la nivelele superioare în mod ierarhic;
- tratarea unitară a excepţiilor de acelaşi tip.

B. Ierarhia excepţiilor

Pentru a crea un obiect excepţie, Java pune la dispoziţia programatorului o ierarhie de clase, aflată în pachetul *java.lang*, ierarhie ce poate fi observată în imaginea de mai jos.

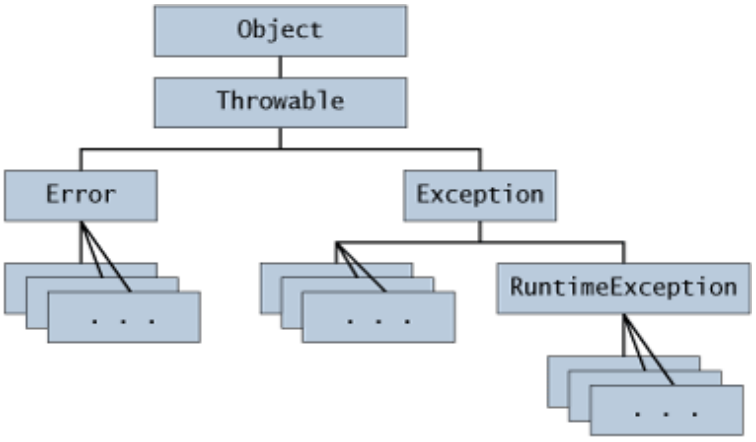

Figura 1: Ierarhia simplificată a claselor de tip excepţie din pachetul java.lang

După cum se poate observa din figura de mai sus, clasa *Throwable* are doi descendenţi: clasa *Error* şi clasa *Exception*. Nici una nu adaugă metode suplimentare, dar au fost introduse în acest fel pentru a delimita două tipuri fundamentale de excepţii ce pot apărea într-o aplicaţie Java.

Clasa *Error* corespunde excepţiilor ce nu mai pot fi recuperate de către programator. Apariţia unei astfel de excepţii înseamnă că a apărut o eroare deosebit de gravă şi aceasta determină terminarea programului. Deşi este datoria programatorului să arunce şi să trateze excepţiile apărute, excepţiile de tipul *Error* nu trebuie tratate în acest fel. Ele sunt utilizate de maşina virtuală Java (JVM).

Clasa *Exception* este, de fapt, cea utilizată efectiv de programator în procesul de tratare a excepţiilor. Atât această clasă cât şi descendenţii ei se ocupă de excepţii ce pot fi rezolvate de către program, fără oprirea acestuia.

Există o mare varietate de clase derivate din *Exception* care pot fi utilizate. Lista completă se află aici.

C. Aruncarea unei excepţii

Aruncarea unei excepţii se face cu ajutorul cuvântului cheie *throw* în felul următor:

throw <obiectExceptie>

sau

throw new <clasaExceptie>("Mesaj")

unde *<obiectExceptie>* şi *new <clasaExceptie>("Mesaj")* sunt instanţe ale clasei *Throwable* sau ale unei clase derivate din aceasta.

O metodă poate arunca mai multe excepţii, însă prin aruncarea unei excepţii se iese din metodă fără a mai executa instrucţiunile care urmau. În cazul în care o metodă aruncă o excepţie, fie prin *throw*, fie prin *apelul unei alte metode*, fără a avea o secvenţă *try - catch* de prindere, atunci această metodă trebuie să specifice clar acest lucru.

public void <numeMetoda> *throws* <classException1>, <classException2>, . . . {

. . .

***throw* <obExceptie1>**

. . .

***throw* <obExceptie2>**

...

}

D. Prinderea unei excepţii

O dată ce o excepţie a fost aruncată, ea trebuie prinsă. Acest lucru se poate realiza cu ajutorul unui bloc *try-catch*, a cărui formă generală este prezentată mai jos:

```
try{
   //cod ce poate arunca o exceptie
}
catch(<classExcept1> <idExcept1>){
   //handler exceptie de tip <classExcept1>
}
catch(<classExcept2> <idExcept2>){
   //handler exceptie de tip <classExcept2>
}
...
finally{
   //secventa de cod executata oricum
}
```

După cum se poate observa, structura de prinderere a excepţiilor poate fi delimitată în trei blocuri:

- *try*

  Secvenţa de cod din acest bloc poate arunca, în anumite condiţii, excepţii. În cazul în care se aruncă o exepţie, execuţia secvenţei din blocul try se întrerupe şi se declanşează procedura de tratare a excepţiei. În caz contrar, secvenţa de cod din interiorul blocului se execută în întregime, controlul fiind predat primei instrucţiuni de dupa blocul *try-catch*.

- *catch*

  Numite şi *handlere de excepţii*, blocurile catch tratează excepţiile. În momentul în care apare o excepţie în blocul try, se parcurge lista blocurilor catch în ordinea în care apar în codul sursă. În cazul în care excepţia corespunde unui bloc catch, se execută secvenţa de

cod care corespunde blocului respectiv şi căutarea ia sfârşit, considerându-se că excepţia a fost rezolvată. După cum se poate observa, pot exista mai multe blocuri catch, acest lucru subliniind faptul că se pot trata excepţii de mai multe tipuri. Important este ca acestea să fie introduse în ordinea *copil - părinte* pentru că blocurile catch se parcurg secvenţial.

- *finally*
  Blocul finally cuprinde secvenţa de cod care se execută în final, indiferent dacă a apărut o situaţie de excepţie sau nu.

Este obligatoriu ca blocul *try* să fie urmat de cel puţin un bloc *catch* sau *finally*.

E. Crearea propriilor excepţii

În dezvoltarea unei aplicaţii Java pot exista cazuri în care excepţiile apărute să nu poată fi tratate de clasele de excepţie deja existente în pachetele Java (java.util, java.io, java.net). Astfel, dezvoltatorii de aplicaţii sunt nevoiţi să-şi creeze propriile clase de excepţie. Acest lucru se poate realiza foarte uşor şi anume, creând clase care extind clasa *Throwable* sau clase descendente ale acesteia. Se recomandă, dintre subclasele lui *Throwable*, să se extindă clasa *Exception*. Clasa *Error* este specifică erorilor grave, care duc la terminarea execuţiei programului, şi este puţin probabil să avem nevoie de ea.

II. PREZENTAREA LUCRĂRII DE LABORATOR

A. Exemple de programe ce demonstrează lucrul cu excepţiile

1. Programul următor conţine funcţia *main()* care apelează metoda *metoda1()* având ca parametru un număr întreg. Funcţia *metoda1()* va arunca o excepţie în cazul în care parametru trimis este diferit de 0, altfel funcţia se va executa până la capăt. Observaţi ce se afişează pentru i=0 şi pentru i=1; când se execută funcţia *metoda1()* până la capăt şi când nu. De asemenea, observaţi şi prezenţa blocului finally.

```
1 public class TestExceptions1 {
2    public static void main(String[] args) {
3       for (int i = 0; i <= 1; i++) {
4          try {
5             System.out.println("\n Caz "+i);
```

```
            //aparitia exceptiei se realizeaza prin apelul metodei metoda1
            metoda1(i);
            System.out.println("\n Sfarsit caz "+i);
         } catch (Exception ex) {
            //tratarea exceptiei intalnite in metoda1
            //apelarea functiei getMessage() a clasei Exception
            System.out.println("A aparut o exceptie. Mesajul ei este: "+ex.getMessage());
         } finally {
            System.out.println("Se executa blocul finally");
         }
      }
   }

   //metoda1 specifica prin intermediul cuvantului cheie "throws" ce tipuri de exceptii arunca
   private static void metoda1(int i) throws Exception {
      System.out.println("Am intrat in metoda1");
      if (i != 0){
         //aruncarea exceptiei de tipul Exception
         throw new Exception("exceptie din metoda1");
      }
      System.out.println("Am iesit din metoda1");
   }
}
```

2. Acest program conţine funcţia *main()* care apelează funcţia *metoda1()*, care, la rândul ei apelează funcţia *metoda2()* ce poate arunca o excepţie. Observaţi că ambele funcţii (metoda1() şi metoda2()) specifică ce tip de excepţie aruncă (Throwable). Deşi blocul try-catch are mai multe blocuri catch, excepţia apărută este tratată de blocul corespunzător acesteia, adică Throwable. Acestă clasă conţine diverse metode care pot ajuta programatorul în localizarea excepţiei. Despre acestea puteţi citi aici.

```
public class TestExceptions2 {
   public static void main(String[] args) {
      for (int i = 0; i <= 1; i++) {
         try {
            System.out.println("\n Caz " + i);
            metoda1(i);
            System.out.println("\n Sfarsit caz " + i);
         } catch (Exception ex) {
            System.out
                  .println("A aparut o exceptie Exception. Mesajul ei este: "
                        + ex.getMessage());
         } catch (Throwable exTh) {
            System.out.println("A aparut o exceptie Throwable in main");
            /*apelarea metodei printStackTrace() a clasei Throwable care afiseaza, conform
                 principiului unei stive LIFO,
              informatii despre locatiile parcurse de exceptie*/
            exTh.printStackTrace(System.out);
         } finally {
            System.out.println("Se executa blocul finally din main");
         }
      }
   }
```

```
22
23    private static void metoda1(int i) throws Throwable {
24       System.out.println("Am intrat in metoda1");
25       metoda2(i);
26       System.out.println("Am iesit din metoda1");
27    }
28
29    private static void metoda2(int i) throws Throwable {
30       System.out.println("Am intrat in metoda2");
31       if (i != 0) {
32          throw new Throwable("exceptie din metoda2");
33       }
34       System.out.println("Am iesit din metoda2");
35    }
36 }
```

3. Programul următor este unul didactic întrucât aruncă clase de excepţie descendente ale clasei *RuntimeException*. Nu este obligatoriu ca acest gen de excepţii să fie ”prinse” (tratate), dar exemplul a fost introdus pentru o mai bună înţelegere a mecanismului de aruncare/prindere a excepţiilor.

   După cum se poate observa la o primă rulare a programului, va fi prinsă prima excepţie, *NumberFormatException*. Comentaţi, respectiv decomentaţi, secvenţe de cod din program, astfel încât să obţineţi, pe rând, toate excepţiile tratate.

   De asemenea, programul conţine o metodă *callException()* ce primeşte ca parametru un tip de dată excepţie. Aceasta apelează câteva dintre metodele implementate de clasele de tip excepţie. Metodele respective au rolul de a oferi mai multe informaţii despre locaţia şi tipul de excepţie apărut.

```
1  public class TestExceptions3 {
2     public static void main(String[] args) {
3        try {
4           System.out.println("\n Number Format Exception");
5           String str1 = "123r";
6           Integer n = Integer.parseInt(str1);
7
8           System.out.println("\n ArrayIndexOutOfBoundsException");
9           int sir[] = { 1, 2, 3, 4 };
10          sir[6] = 3;
11
12          System.out.println("\n StringIndexOutOfBoundsException");
13          String str2 = "abcde";
14          char c1 = str2.charAt(-1);
15
16          System.out.println("\n NullPointerException");
17          String str3 = null;
18          System.out.println(str3.length());
19
20          System.out.println("\n ArithmeticException");
```

```
21            int n2 = 12/0;
22
23        } catch (NumberFormatException nfex) {
24            System.out.println("Tratare exceptie NumberFormatException...");
25            callException(nfex);
26        } catch (ArrayIndexOutOfBoundsException arrayEx) {
27            System.out.println("Tratare exceptie ArrayIndexOutOfBoundsException...");
28            callException(arrayEx);
29        } catch (StringIndexOutOfBoundsException strEx) {
30            System.out.println("Tratare exceptie StringIndexOutOfBoundsException...");
31            callException(strEx);;
32        } catch (NullPointerException nullEx) {
33            System.out.println("Tratare exceptie NullPointerException...");
34            callException(nullEx);
35        }catch(ArithmeticException aEx){
36            System.out.println("Tratare exceptie ArithmeticException...");
37            callException(aEx);
38        }
39    }
40
41    private static void callException(RuntimeException rtex){
42        System.out.println("Mesajul standard al exceptiei: ");
43        System.out.println(rtex);
44        System.out.println("Mesajul exceptiei: ");
45        System.out.println(rtex.getMessage());
46        System.out.println("Mesajul local: ");
47        System.out.println(rtex.getLocalizedMessage());
48        System.out.println("\nStack trace: ");
49        rtex.printStackTrace(System.out);
50    }
51 }
```

B. Crearea propriilor excepţii

Crearea propriilor excepţii se face prind crearea unei clase obişnuite, dar care moşteneşte clasa *Exception*. Clasa următoare face exact acest lucru. *IncompatibleMatrixException* conţine doi constructori, unul fără parametri şi unul cu un parametru şi care apelează constructorul clasei de bază.

```
1 public class IncompatibleMatrixException extends Exception{
2
3     public IncompatibleMatrixException() {
4     }
5
6     public IncompatibleMatrixException(String message) {
7         super(message);
8     }
9 }
```

Clasa prezentata mai jos, declară două matrici de dimensiuni diferite şi se încearcă înmulţirea acestora. Metoda *multiplyMatr()* este apelată în cadrul unui bloc try şi verifică dacă două matrici

sunt compatibile. În cazul în care numărul de coloane ale primei matrici este diferit de numărul de linii ale celei de-a doua, atunci se aruncă excepţia creată anterior, *IncompatibleMatrixException*.

```
public class TestMatrixException {
   public static void main(String[] args) {
      int[][] matr1, matr2;
      matr1 = new int[2][3];
      matr2 = new int[4][4];

      try{
         multiplyMatr(matr1, matr2);
      }catch(IncompatibleMatrixException imEx){
         System.out.println("Mesajul este: "+imEx.getMessage());
         imEx.printStackTrace(System.out);
      }
   }

   private static void multiplyMatr(int[][] matr1, int[][] matr2) throws IncompatibleMatrixException{
      if (matr1[0].length != matr2.length){
         throw new IncompatibleMatrixException("matrici incompatibile pt inmultire");
      }
      //cod destinat pt inmultirea matricilor
   }

}
```

III. TEMĂ

1. Rulaţi toate programele prezentate ca exemplu în secţiunea II.

2. Realizaţi un program în care să se testeze captarea şi tratarea excepţiilor generate la calcularea a cel puţin 10 funcţii matematice din clasa *Math* aflată în pachetul *java.lang*. Folosiţi blocuri de catch specifice pentru fiecare funcţie în parte.