PROGRAMARE ORIENTATĂ PE OBIECTE

GENGE BÉLA

Capitolul 6
Excepții. Fire de execuție

Excepții

- Reprezintă mecanismul de tratare a erorilor apărute în timpul rulării.
- Dacă o excepție nu este tratată, mașina virtuală întrerupe execuția aplicației.
- Exemplu de excepție:
  - Alocarea unui tablou și accesarea unui index din afara limitelor.

Sursa excepțiilor

- Evenimente neașteptate (fișier șters/redenumit, împărțirea cu 0, referință null, etc.).

- Eroare detectată de algoritm, de unde poate rezulta crearea unei excepții și **aruncarea** acesteia (ștergerea unui element căutat dintr-o listă – elementul nu există).

Tratarea excepțiilor

- Excepțiile trebuie **tratate**:
  - De către aplicație.
  - De către JVM – în cazul în care nu sunt tratate de aplicație se întrerupe execuția.
- La apariția unei excepții:
  - Aplicația intră într-un mod de tratare special.
  - JVM suspendă execuția normală.
  - Excepția tratată poate fi propagată (aruncată) dintr-o metodă în alta.
  - Dacă aplicația nu tratează excepția, aceasta ajunge la JVM, care operște execuția.

Tratarea excepțiilor

- Generarea unei excepții: throw.
- Tratarea: try-catch-finally.
- Obiectele excepție se crează ca orice alt obiect.
- Pentru crearea unei excepții “personalizate” se moștenește clasa Throwable.
  - Metodele din această clasă asigură funcționalitatea de bază specifică excepțiilor (ex: crearea unei imagini a stivei care să fie afișată).
  - Această abordare permite administarea unitară a JVM a tuturor obiectelor de tip excepție.
- Bună practică: denumirea unei excepții să conțină cuvântul cheie Exception.
  - Ex: StackOverflowException

Clasa Exception

- Reprezintă prima clasă din ierarhie care moștenește clasa Throwable.
- Asigură o funcționalitate de bază pentru toate excepțiile (o excepție generică).
- Unul din constructorii clasei Exception preia un String – o descriere detaliată a erorii.

- Aruncarea unei excepții Exception:
  - throw new Exception(“Eroare de algoritm!”).
  - Metoda în cadrul căreia se aruncă excepția trebuie să specifice:
    - throws Exception.

Clasa Exception

- Tratarea excepțiilor:
  - try{
  - // instructiuni generatoare de exceptie
  - }
  - catch(Exception1 ex) {//tratare}
  - catch(Exception2 ex) {//tratare}
  - finally {//}
- Finally e opțional, de regulă utilizat pentru instrucțiuni care se doresc a fi executate în toate scenariile (ex: închiderea unui fișier).
- Ordinea tipului excepțiilor trebuie să urmeze ierarhia excepțiilor: mai întâi cele derivate, după care cele de bază.
- Metoda printStackTrace().

Excepții verificate/neverificate

- Excepțiile verificate (la compilare): trebuie tratate OBLIGATORIU.
  - Ex: InterruptedException.
  - **Se moștenesc din clasa Exception.**
- Excepțiile neverificate (la compilare): NU trebuie tratate neapărat.
  - Ex: ArrayIndexOutOfBoundsException.
  - Se moștenesc din clasa RuntimeException (aceasta moștenește direct clasa Exception).

Excepții verificate/neverificate

- Metodele care nu tratează excepțiile verificate trebuie să includă în semnătură cuvântul cheie **throws**.
- Mai multe excepții aruncate de o metodă pot fi separate prin virgulă.
- Exemplu (sursa: https://www.tutorialspoint.com/java/java_exceptions.htm):

```
import java.io.*;
public class className {

   public void deposit(double amount) throws RemoteException {
      // Method implementation
      throw new RemoteException();
   }
   // Remainder of class definition
}
```

Crearea propriilor clase excepție

- Exemplu StackException.

Fire de execuție

- Permit execuția mai multor secvențe de cod în paralel în cadrul aceluiași **proces**.
- Firele de execuție partajează memoria.
- Procesele rulează în spații de memorie diferite.
- În Java: clasa Thread. Main-ul este rulat de un fir creat automat de JVM.
- Noțiunea de multi-threading și paralelism poate fi "simulată" prin execuția și deplanificarea succesivă rapidă.
  - Paralelismul real depinde de JVM, S.O. Și de resursele hardware.

Fire de execuție

- Ce este propriu fiecărui fir:
  - Stiva.
  - Stack pointer.
- Toate firele din același proces au acces la variabile globale (heap).
- Același cod poate fi rulat de mai multe fire în paralel:
  - Variabilele locale (declarate pe stivă) vor fi diferite de la un fir la altul.

De ce fire de execuție?

- Asigură o soluție elegantă la problema accesului la resurse cu blocare:
  - Citire/scriere pe rețea.
  - Citire/scriere într-o bază de date.
  - **Atenție! Soluția vine cu o serie de probleme: administrarea timpului de viață și comunicarea cu firele de execuție.**
- Exemple de probleme ce pot fi paralelizate:
  - Adunarea/înmulțirea matricilor.
  - Prelucrarea imaginilor.
  - Aplicațiile Server (chat, file).
- Exemple de probleme ce NU pot fi paralelizate:
  - Calculare funcție HASH criptografică (înlănțuire de operații, fiecare depinde de blocul anterior).

Crearea firelor de execuție

- Moștenirea clasei Thread.
- Implementarea interfeței Runnable.
- În ambele cazuri codul firului se implementează în metoda:
  - **public void run();**

Stările unui fir de execuție

- Inițializare.
- Planificare.
- Deplanificare.
- Oprire.

Moștenirea clasei Thread

- Suprascrierea metodei run().

```
public class MyThread extends Thread{

    public void run() {
        try{
            sleep(1000);
        }
        catch(InterruptedException ex){
        }
    }
}

....

MyThread th = new MyThread();
th.start();
```

Moștenirea clasei Thread

- Exemplu:
  - Crearea a două fire de execuție, care afișează numere întregi din intervalul a,b, câte un număr la fiecare secundă.

Implementarea interfeței Runnable

- Implementarea metodei run() în clasa MyThread.
- Instanțierea clasei Thread și a clasei Mythread.
- Constructorul clasei Thread primește referința către o clasă ce implementează interfața Runnable.

Implementarea interfeței Runnable

```
public class MyThread implements Runnable{

    public void run() {
        try{
            sleep(1000);
        }
        catch(InterruptedException ex){
        }
    }
}

....
MyThread mth = new MyThread();
Thread th = new Thread(mth);
th.start();
```