Projecte de Programació

Excepcions en Java

Prova de Programes: Java

Java admet un tractament d'errors i excepcions mitjançant les classes que hereten de la classe **`Throwable`**, que és la classe encarregada de gestionar les situacions excepcionals.

Hi ha dos tipus de situacions excepcionals:

- Errors
  (**`java.lang.Error`**, subclasse de **`java.lang.Throwable`**)

- Excepcions
  (**`java.lang.Exception`**, subclasse de **`java.lang.Throwable`**)

Prova de Programes: Java

- Errors
  (`java.lang.Error`, subclasse de `java.lang.Throwable`)

En Java, els errors es refereixen a situacions que no haurien de passar mai i que el programa ha d'abortar.

Exemples:

- La JVM no pot continuar per manca de recursos.
- Un `.class` té problemes en el seu contigut.
- Problemes d'incompatibilitats entre classes depenents.
- Problemes *seriosos* d'entrada/sortida.
- Un *thread* mor sense raó.
- Errors en l'AWT

Aquests errors són irrecuperables, no té sentit tractar-los. Normalment el programa ha d'aturar-se.

Prova de Programes: Java

- Excepcions
  (**`java.lang.Exception`**, subclasse de **`java.lang.Throwable`**)

En Java, les excepcions es refereixen a situacions extraordinàries que cal tractar de manera especial

Exemples:

- **`RunTimeException`**: Errors de programació que cal comprovar, p.ex. **`NullPointerException`**, **`IndexOutOfBoundsException`**, **`ArithmeticException`**, etc.
- **`IOException`**: Errors en operacions d'entrada/sortida
- **`NoSuchFieldException`**: No es troba un atribut
- **`NoSuchMethodException`**: No es troba un mètode
- N'hi ha moltíssims més...

Aquests errors té sentit tractar-los (*capturar-los*) i continuar el programa.

Prova de Programes: Java

- Excepcions

Una excepció en un programa Java ha de ser *considerada* d'alguna forma. De fet, el compilador ens obliga.

Imaginem que estem programant un mètode `m()` que invoca un altre mètode, i aquest executa una operació susceptible de generar una excepció de nom *NomExcepció*.

Aleshores el mètode que estem programant, anomenat `m()`, pot tractar, o no, l'excepció.

Prova de Programes: Java

- Excepcions

Suposem que `m()` **NO** tracta l'excepció *NomExcepció*.

- En aquest cas, a la capçalera de `m()` cal posar
  `throws NomExcepció`
  (o el nom de qualsevol de les superclasses de *NomExcepció*)

- *No tractar-la* significa *si es produeix una excepció amb aquest nom (o qualsevol de les seves subclasses), l'excepció es passa al mètode que l'ha cridat*

Cal considerar que si no es vol tractar CAP excepció n'hi ha prou amb posar la clausula `throws Exception` (la classe pare de totes les excepcions).

Si l'excepció arriba al `main` el programa aborta.

Prova de Programes: Java

- Excepcions

Suposem que `m()` **SÍ** tracta l'excepció *NomExcepció*.

- En aquest cas, no cal posar res a la capçalera de `m()`

- Cal posar les instruccions susceptibles de generar l'excepció dins Un bloc `try` amb un `catch` per tractar les excepcions. Eventualment es pot posar un bloc `finally`.

A grans trets: El codi dins el `try` està *vigilat*. Si es produeix una situació anormal i es genera una excepció, el control passa al bloc `catch` corresponent a l'excepció generada. Podem posar tants blocs `catch` com vulguem. Finalment, si hi és, s'executarà el bloc `finally`, tant si s'ha generat una excepció com si no (és a dir, *sempre*).

Prova de Programes: Java

- Excepcions

Exemple de NO tractament d'excepcions:

```
public void metodeExcepcio() throws Exception {
 // FileNotFoundException
 BufferedReader in = new
    BufferedReader(new FileReader("noexisteix"));
    ...
}
```

Prova de Programes: Java

- **Excepcions:** Exemple de tractament d'excepcions:

```
public void metodeExcepcio() {
    try {
       int[] a = new int[3];
       // IndexOutOfBoundsException
       //a[5]=10;
       // FileNotFoundException
       BufferedReader in = new BufferedReader(new FileReader("noexiste.txt"));
       ...
    }
    catch (IndexOutOfBoundsException e) {
        System.out.println(" *** IndexOutOfBoundsException... ");
    }
    catch (FileNotFoundException e) {
         System.out.println(" *** getMessage(): " + e.getMessage());
         System.out.println(" *** toString(): " + e.toString());
         e.printStackTrace();
   }
   finally {
         System.out.println(" *** finally... ");
    }
}
```

Prova de Programes: Java

- Excepcions. Algunes consideracions:

- No té gaire sentit tractar algunes **`RunTimeExceptions`**, com **`NullPointerException`** o **`IndexOutOfBoundsException`** cada cop que accedim a un objecte o treballem amb un *Array*.

- Hi ha alguns missatges que podem utilitzar en un bloc **`catch`**:

  - **`String getMessage()`**: Retorna el missatge associat a l'excepció

  - **`String toString()`**: Retorna la representació textual de l'excepció

  - **`printStackTrace()`**: Retorna la representació textual de la pila d'execució en el moment de l'excepció.

Prova de Programes: Java

- Excepcions. Creació d'excepcions pròpies.

Cal fer una classe que hereti d'**`Exception`**

Cal que hi hagi un constructor sense arguments, un constructor amb un **`String`** d'argument i cal que tots dos constructors invoquin el constructor de la classe pare.

Crearem un objecte de la classe **`MyException`** fent:

```
MyException me = new MyException("--Missatge d'error")
```

Si es vol generar l'excepció: **`throw me`**

En generar l'excepció, el mètode acaba immediatament sense retornar cap valor.

Prova de Programes: Java

- La instrucció **`assert`**

De vegades va bé utilitzar **`assert`** per depurar programes Java.

Cal inserir en el codi:

```
assert ExpressioBooleana;

assert ExpressioBooleana1 : Expressio2;
```

Quan s'executa la instrucció, s'avalua **`ExpressioBooleana`**. Si el resultat és fals, el sistema genera un **`AssertionError`** amb un missatge que contindrà el resultat d'haver avaluat **`Expressio2`** (si hi és).

Per defecte la JVM NO executa els **`assert`**, cal invocar-la amb l'opció **`-ea`** o bé **`--enableassertions`**