Paradigmi di programmazione
Sperimentazioni
2000-01

Alberto Martelli

Eccezioni

Eccezioni

Eccezioni: meccanismo per trattare condizioni eccezionali

- divisione per zero
- indice di array fuori dai limiti
- errori di input da file
- errore di stampa
- classe non trovata
- .....

L'esecuzione si blocca e viene sollevata (*lanciata*) una eccezione.

Eccezioni

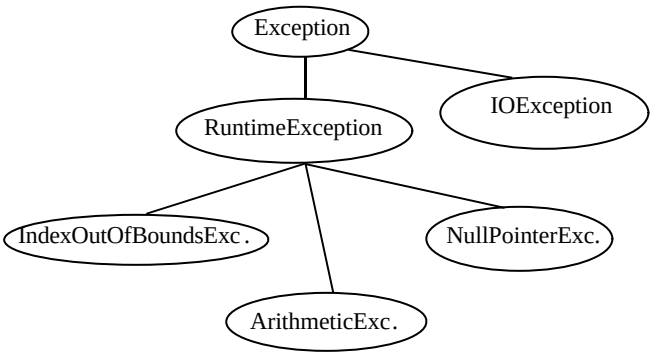
```
class Ecc {
   static int[] a = new int[2];
   static void p() {a[4] = 5;}
   public static void main (String[] args) {
       p();}
}
```

Eseguendo il *main* viene fornito il seguente messaggio di errore a run-time

```
Exception in "main"
  java.lang.ArrayIndexOutOfBoundsException
      at Ecc.p (Ecc.java: 3)
      at Ecc.main (Ecc.java: 5)
```


Classi eccezioni

Una eccezione è un oggetto della classe **Exception** o di una sua sottoclasse.



Una **RuntimeException** si verifica perché è stato commesso un errore di programmazione (*se è una **RuntimeException** è colpa del programmatore*):

- accesso ad array fuori dai limiti
- cast sbagliato
- puntatore nullo

Le eccezioni che non derivano da **RuntimeException** possono essere:

- tentativo di leggere oltre l'EOF
- tentativo di aprire un URL errato

throw

Le eccezioni possono essere *lanciate* con: **throw** <eccezione>

```
class Tabella {
   int n; int[] a;
   void inserisci(int x) {
      if (n >= a.length - 1)
         throw new RuntimeException("tabella piena");
      .........
   public static void main (String[] args) {
      Tabella t = new Tabella(2);
       t.inserisci(5); t.inserisci(8);
   ......
```

```
Runtime exception: tabella piena
      at Tabella.inserisci ...
      at Tabella.main .....
```

Catturare le eccezioni

Chi lancia una eccezione non ha informazioni sufficienti per decidere cosa fare.

Invece di bloccare comunque l'esecuzione, l'eccezione può essere catturata ad un livello più alto da un *exception handler*, che può decidere come continuare l'esecuzione del programma dopo aver preso le misure opportune.

Eccezione come terminazione eccezionale di un metodo.

Il lancio di una eccezione termina l'esecuzione di un metodo.

L'eccezione viene passata al chiamante che può:

- catturarla e gestirla
- passarla a sua volta al suo chiamante

Catturare le eccezioni

```
class Tabella {
   int n; int[] a;
   void inserisci(int x) {
      if (n >= a.length - 1)
         throw new RuntimeException("tabella piena");
      .........
   public static void main (String[] args) {
      Tabella t = new Tabella(2);
      try {
           t.inserisci(5);
           t.inserisci(8);
      } catch(RuntimeException e) {
        System.out.println(e.getMessage());
   }
```

try - catch

In generale un *exception handler* ha la forma:

```
try {
   codice che può generare una eccezione
} catch(Tipo1 id1) {
               gestisce eccez. di Tipo1
} catch(Tipo2 id2) {
               gestisce eccez. di Tipo2
}........
```

Tipo1, Tipo2, ... sono tipi di eccezioni.

Come catturare le eccezioni

Quando viene lanciata una eccezione, si interrompe l'esecuzione e si cerca **dinamicamente**, percorrendo all'indietro lo stack delle chiamate, il primo *exception handler* che ha una *catch* per il tipo dell'eccezione lanciata, e l'esecuzione riprende da quel punto.

Definizione di eccezioni

E' possibile definirsi una propria eccezione derivata da **Exception** o da una sua sottoclasse.

Eccezioni definite

```
class TabellaPiena extends RuntimeException {
   public int i;
   public TabellaPiena(int n) {i=n;}
}
class Tabella {.....
   void inserisci(int x) {
      if (n >= a.length - 1)
         throw new TabellaPiena(x);
      .........
   public static void main (String[] args) {
      Tabella t = new Tabella(2);
      try {
            ..........
      } catch(TabellaPiena tp) {
            System.out.println("La tabella è piena." +
               tp.i + " non è stato inserito");
   }
```


throws

Se un metodo può lanciare un'eccezione e non la cattura, deve segnalarlo con **throws** (tranne che per RuntimeException).

```
class Generica {
   void m(String s) throws ClassNotFoundException {
      Class c = Class.forName(s);
      ...........
```

Il metodo **`read`** può lanciare una **`IOException`**.

Due modi di gestirla in un metodo **`readLine`**:

1. Catturare l'eccezione

```
public static String readLine() {
   int ch;
   String r="";
   boolean done=false;
   while (!done) {
      try {ch=System.in.read();
           ...
           r=r+(char)ch;
          }
          catch (IOException e)
             {done=true;}
   }
   return r;
}
```


2. Passare l'eccezione senza gestirla

```
public static String readLine() throws IOException {
   int ch;
   String r="";
   boolean done=false;
   while (!done) {
      ch=System.in.read();
       ...
      r=r+(char)ch;
   }
   return r;
}
```

Exception handler generale

```
try {
   codice che può generare una eccezione
} catch(Tipo1 id1) {
              gestisce eccez. di Tipo1
} catch(Tipo2 id2) {
              gestisce eccez. di Tipo2
}........
} finally {....}
```

Il blocco di **finally** è eseguito sempre, anche se non sono sollevate eccezioni