Università degli Studi di Modena e Reggio Emilia

Facoltà di Ingegneria – Reggio Emilia

CORSO DI

RETI DI CALCOLATORI
Linguaggio Java: Eccezioni

Prof. Franco Zambonelli

Lucidi realizzati in collaborazione con
Ing. Enrico Denti - Univ. Bologna

ECCEZIONI

Spesso i programmi contengono istruzioni "critiche", che potrebbero portare a errori.

Tipicamente, in quei casi si inseriscono controlli (`if... then..`) per intercettare le situazioni critiche e cercare di gestirle in modo appropriato.

Tuttavia, questo modo di procedere è spesso insoddisfacente:

- non è facile prevedere tutte le situazioni che potrebbero produrre l'errore
- "gestire" l'errore spesso significa solo stampare a video qualche messaggio.

Java adotta anche in questo ambito un approccio innovativo, introducendo il concetto di *eccezione:*

- anziché tentare di prevedere i casi che possono portare a errore, ***si prova a eseguire l'operazione <u>in un blocco "controllato"</u>***
- **se si prduce un errore, l'operazione *solleva un'eccezione***
- **l'eccezione viene *catturata* dal blocco entro cui l'operazione era eseguita, e può essere *gestita* nel modo più appropriato.**

```
try {
   // operazione critica che può sollevare eccez.
}
catch (Exception e) {
   // gestione dell'eccezione
}
```

Se l'operazione può sollevare diversi tipi di eccezione (corrispondenti a diversi tipi di errore), possono essere previsti *più blocchi* `catch` di seguito allo stesso blocco `try`.

ESEMPIO

Una tipica operazione “critica” è la lettura da input.

In Java, il dispositivo di input standard è la variabile (`static`) `System.in`, di classe `InputStream` (una classe astratta, di cui `System.in` è una istanza “anomala” predefinita).

Poiché `InputStream` fornisce solo un metodo `read()` che legge singoli byte, si usa incapsulare `System.in` in un oggetto dotato di maggiori funzionalità, come ad esempio un `BufferedReader`, che fornisce anche un metodo `readLine()`:

```
import java.io.*;
class EsempioIn {
  public static void main(String args[]){
    int a = 0, b = 0;
    BufferedReader in = new BufferedReader(
            new InputStreamReader(System.in));
    try {
      System.out.print("Primo valore: ");
      a = Integer.parseInt(in.readLine());
      System.out.print("Secondo valore:");
      b = Integer.parseInt(in.readLine());
    }
    catch (IOException e) {
      System.out.println("Errore in input");
    }
    System.out.println("La somma vale " + (a+b));
  }}
```

Qui, `readLine()` solleva una `IOException` in caso di errore in fase di input, mentre `Integer.parseInt()` solleva una `NumberFormatException` (non catturata) se la stringa restituita da `readLine()` non corrisponde alla sintassi di un numero intero.

Una eccezione non catturata *si propaga verso l’esterno*, di blocco in blocco: se raggiunge il main, provoca l’aborto del programma.

COS'È UN'ECCEZIONE

Una eccezione è un oggetto, istanza di `java.lang.Throwable` o di una qualche sua sottoclasse.

In particolare, le due sottoclassi più comuni sono

- `java.lang.Exception`
- `java.lang.Error`

La parola "eccezione" è però usualmente riferita a entrambe.

Un `Error` indica problemi relativi al caricamento della classi o al funzionamento della macchina virtuale Java (es. not enough memory), e va considerato *irrecuperabile*: perciò *non* è da catturare.

Una `Exception`, invece, indica di solito situazioni *recuperabili* (es: fine file, indice di un array oltre i limiti, errori di input, etc.).

Poiché un'eccezione è un oggetto, può contenere dati o definire metodi:

- tutte le eccezioni definiscono un metodo `getMessage()` che restituisce il messaggio d'errore associato all'eccezione
- alcune eccezioni definiscono dei campi, come `bytesTransferred` in `InterruptedIOException`, che forniscono ulteriori informazioni utili per meglio gestire la situazione.

RILANCIARE ECCEZIONI

Java richiede che un metodo entro cui si può generare un'eccezione *o gestisca l'eccezione*, con un costrutto `try`/`catch`, ***oppure dichiari di rilanciarla all'esterno*** **del metodo stesso, con la clausola `throws`**:

```
public int readInteger(BufferedReader in)
    throws IOException, NumberFormatException {
  return Integer.parseInt(in.readLine());
}
```

DEFINIRE E GENERARE NUOVE ECCEZIONI

Essendo un'eccezione nulla più che un normale oggetto Java (istanza di una classe opportuna), è possibile

- definire nuovi tipi di eccezione definendo *nuove classi*
- generare eccezioni dall'interno di propri metodi.

Per definire un nuovo tipo di eccezione basta definire una nuova classe che estenda la classe `Exception` o una delle sue sottoclassi. Ad esempio:

```
class NumberTooBigException
    extends IllegalArgumentException {
 public NumberTooBigException() { super(); }
 public NumberTooBigException(String s){super(s);}
}
```

Solitamente non è necessario definire particolari campi dati, in quanto l'informazione fondamentale (il tipo dell'eccezione) è già veicolata dalla classe stessa (e dal messaggio `s`).

Per lanciare un'eccezione dall'interno di un metodo occorre *creare* l'oggetto eccezione, e *poi "lanciarlo"* con l'istruzione `throw`:

```
public int readInteger(BufferedReader in)
    throws IOException, NumberFormatException,
           NumberTooBigException {
    int x = Integer.parseInt(in.readLine());
    if (x>100) throw new NumberTooBigException();
    return x;
}
```

- **non confondere** `throw` (che è un'istruzione che lancia un'eccezione) con la dichiarazione `throws` (usata nella dichiarazione di una classe)
- la creazione dell'oggetto `NumberTooBigException` può avvenire - e spesso avviene - entro l'istruzione `throw` stessa.