Lab. Programmazione (CdL Informatica) a.a. 2022-23

Monica Nesi

Università degli Studi dell'Aquila

10 Gennaio 2023

Eccezioni in Java: breve introduzione

In un linguaggio di programmazione le *eccezioni* hanno a che fare con situazioni di *errore*.

Ma *non* gli errori a tempo di compilazione.

Le eccezioni permettono di segnalare errori a tempo di esecuzione.

Ipotesi: codice sintatticamente corretto (i.e., compilazione ok).

Gli errori possono denotare:

- situazioni *normali* (e.g., la fine di un file) che vengono segnalate al programmatore che può gestirle o meno;
- situazioni *anormali* (e.g., accesso al di fuori di un array) che portano alla terminazione immediata del programma.

Riconsideriamo il metodo per il fattoriale...

Un esempio: il fattoriale

La specifica data in precedenza per un metodo per la funzione fattoriale è la seguente.

Scrivere un metodo che calcola il fattoriale di un numero naturale `n`: $0! = 1$ ed $n! = n \times (n-1) \times \ldots \times 2 \times 1$ per $n \geq 1$.
Se al metodo viene passato un numero intero negativo, il metodo restituisce `-1`.

Tramite il meccanismo delle eccezioni, non diremo più che in caso di parametro negativo il metodo per il fattoriale deve restituire un particolare valore, ma che in tal caso il metodo deve *sollevare un'opportuna eccezione*, ovvero deve segnalare l'errore al metodo che ha invocato il fattoriale.

Il metodo chiamante potrà decidere se *gestire l'eccezione sollevata* oppure *non gestirla e propagare l'eccezione*.

Un metodo per il fattoriale

```
public static int fattoriale (int n) {
  if (n<0) return -1;  //No, sollevare eccezione
  int f = 1;
  for (int i=1; i<=n; i++)
    f = f*i;
  return f;
}
```

Dato questo metodo per il fattoriale, non è assicurato che il metodo chiamante applichi un test sul risultato della chiamata del fattoriale e agisca di conseguenza (e.g., stampando un messaggio di errore se il risultato è -1).

Tramite il meccanismo delle eccezioni, si può imporre al metodo che invoca un metodo (che può sollevare eccezione) di gestire tale eccezione.

Eccezioni in Java

Un obiettivo di Java è l'essere un linguaggio portabile e robusto.

Per questo si ha una gestione uniforme/omogenea di tipi diversi di errore (i.e., errori di sistema ed errori di programmazione) + l'integrazione con la metodologia ad oggetti.

Le eccezioni in Java sono oggetti.

Definire un'eccezione significa creare un oggetto della classe `Throwable` (o sue classi derivate) che contiene informazioni sul tipo di problema.

In Java esiste un'estesa *gerarchia di classi* per le eccezioni a partire dalla classe `Throwable`.

Eccezioni in Java (cont.)

Tipicamente noi utilizzeremo la classe `Exception` o, meglio, *opportune ridenominazioni di tale classe*, a cui daremo un nome che ci dia informazioni sull'errore individuato.

Con le eccezioni ci sono due aspetti da considerare:

- individuazione della situazione di errore:
  occorre gettare/lanciare/sollevare un'eccezione (*throw*);
- ripristino (se possibile):
  l'eccezione sollevata viene catturata (*catch*), ovvero gestita da un gestore di eccezioni.

Lanciare un'eccezione: comando `throw`

Una situazione di errore viene segnalata lanciando un'eccezione, ovvero un oggetto di una classe eccezione opportuna, tramite il comando

```
throw <OggettoEccezione>;
```

Ad esempio, nel metodo per il fattoriale potremmo scrivere:

```
if (n<0) throw new Exception();
```

dove `new Exception()` è un oggetto della classe `Exception` creato invocando il costruttore di default, oppure

```
if (n<0) throw new IllegalArgumentException();
```

utilizzando invece la classe eccezione `IllegalArgumentException`.

Lanciare un'eccezione: comando `throw` (cont.)

Possiamo dare un nome opportuno alla classe eccezione per capire di quale errore si tratta.

Si può usare il costrutto `extends` per dare un nome diverso ad una classe, e.g. `Exception`:

```
class NegNumberException extends Exception {}
```

e poi usiamo la classe `NegNumberException` per sollevare l'eccezione nel metodo `fattoriale`:

```
if (n<0) throw new NegNumberException();
```

Qui il comando `throw` è seguito da un oggetto `new NegNumberException()` della classe eccezione `NegNumberException`.

Lanciare un'eccezione: marcatore `throws`

Si può dichiarare che un metodo può sollevare eccezioni aggiungendo (alla fine della sua intestazione) il marcatore `throws` seguito da una lista di classi eccezione:

```
throws <ClasseEccezione>,...,<ClasseEccezione>
```

Ad esempio, l'intestazione del metodo per il fattoriale viene modificata come segue:

```
public static int fattoriale (int n) throws
NegNumberException
```

In tal modo si dichiara che il metodo `fattoriale` può sollevare un'eccezione della classe `NegNumberException`.

L'uso di `throws` implica che il compilatore esegue dei controlli sui metodi chiamanti (i.e., verifica se tali metodi gestiscono l'eccezione che potrebbe essere sollevata) rendendo tale eccezione *controllata*.

Catturare/gestire un'eccezione

Quando viene sollevata un'eccezione, l'esecuzione non procede e passa al *gestore* dell'eccezione, fornendo un *meccanismo flessibile* per trasferire il controllo dal punto in cui viene individuato l'errore al gestore competente per il ripristino.

Le eccezioni sono gestite tramite blocchi `try-catch`:

```
try {
  <sequenza istruzioni>
}
catch (<ClasseEccezione> <OggettoEccezione>) {
  <sequenza istruzioni>
}
...
catch (<ClasseEccezione> <OggettoEccezione>) {
  <sequenza istruzioni>
}
```

Blocco `try-catch`: semantica

Viene eseguita la sequenza di istruzioni nel blocco `try` che può sollevare eccezioni.
Se si verifica un'eccezione di una data classe, l'esecuzione del blocco `try` si arresta e si passa ad eseguire la clausola `catch` corrispondente a tale classe.
Se non si hanno eccezioni nella sequenza di istruzioni del blocco `try`, le clausole `catch` vengono saltate.

N.B. In questo corso non consideriamo la clausola `finally`.

In `<OggettoEccezione>` è memorizzato l'oggetto eccezione sollevato, se di tipo corrispondente a <ClasseEccezione>.

Gestione di eccezioni: osservazioni

In un programma si definisce una clausola `catch` per ogni eccezione che si vuole gestire.

Il programmatore deve essere consapevole delle eccezioni e trattarle esplicitamente.

Ad essere rigorosi, le eccezioni dovrebbero essere tutte controllate, ma in genere ciò non è possibile, perché il codice diventa pesante ed illeggibile.

Esempio: almeno k elementi in un array con una data proprietà

Es.11. (lezione 9 novembre 2022) + eccezione
Scrivere un metodo che, dati un array di interi `a` ed un intero `k`, restituisce `true` se in `a` compaiono *almeno* `k` numeri strettamente positivi, `false` altrimenti.
Il metodo deve sollevare un'opportuna eccezione se `k` non è strettamente positivo.

N.B. Ora non si assume `k>0`. Si richiede invece di controllare il suo valore e di lanciare un'eccezione se la condizione non è soddisfatta.

Definiamo una classe eccezione `NonPosNumberException`:

```
class NonPosNumberException extends Exception {}
```

Esempio: almeno k elementi in un array con una data proprietà (cont.)

Una versione del metodo con eccezione è la seguente:

```
public static boolean almenokPos(int[] a,
int k) throws NonPosNumberException {
  if (k<=0) throw new NonPosNumberException();
  int i=0, cont=0;
  while (i<a.length && cont<k) {
    if (a[i]>0)
      cont++;
    i++;
  }
  return (cont>=k);
}
```

Versione ricorsiva di metodi con eccezioni

Come abbiamo visto nel corso, la *versione ricorsiva* di un metodo è stata implementata definendo due metodi con lo stesso nome (sfruttando l'overloading), di cui uno (quello iniziale) non ricorsivo e l'altro ricorsivo (con vari parametri in più).

Se la versione ricorsiva di un metodo deve sollevare eccezione, le situazioni di errore sono controllate e le eccezioni sollevate *solo* nel metodo iniziale e *non* in quello ricorsivo.

Quindi, la definizione del primo metodo si arricchisce con `throws` e `throw`, mentre il metodo ricorsivo rimane inalterato.

Almeno k elementi in un array con una data proprietà: versione ricorsiva

Ad esempio, assumendo di utilizzare ancora la classe eccezione `NonPosNumberException`, la seguente è la versione ricorsiva per il metodo che verifica se esistono almeno `k` elementi in un array con una data proprietà, in cui è richiesto di sollevare un'opportuna eccezione se `k` non è strettamente positivo:

Metodo iniziale + eccezione:

```
public static boolean almenokPosR(int[] a,
int k) throws NonPosNumberException {
  if (k<=0) throw new NonPosNumberException();
  return almenokPosR(a,k,0,0);
}
```

Almeno k elementi in un array con una data proprietà: versione ricorsiva (cont.)

Il metodo ricorsivo non cambia:

```
public static boolean almenokPosR(int[] a,
int k, int i, int cont) {
  if (cont >= k)
    return true;
  if (i == a.length)
    return false;
  if (a[i] > 0)
    cont++;
  return almenokPosR(a,k,i+1,cont);
}
```