Trattamento delle Eccezioni

Il trattamento degli errori..

Non tutti gli errori presenti in un programma possono essere rilevati in compilazione.

Quando si verifica una condizione anormale in una sequenza di codice al momento dell'esecuzione si dice che si è verificata una ***eccezione***.

Nei linguaggi di programmazione che non supportano la gestione delle eccezioni, gli errori devono essere controllati e gestiti 'manualmente' dal programmatore a costo di notevoli sforzi. Ciò avviene solitamente mediante l'utilizzo di codici di errore.

…Il trattamento degli errori.

Java supporta la gestione degli errori, rifacendosi al modello del C++ che a sua volta si basa sul modello di Ada.

Una condizione eccezionale è un problema che impedisce la regolare esecuzione del programma, ovvero l'elaborazione non può continuare perché non si hanno abbastanza informazioni per trattare il problema nel contesto corrente.

Per gestire una eccezione si potrebbe uscire dal contesto corrente relegando il tutto ad un contesto più alto. L'uscita dal contesto corrente corrisponde al ***sollevamento di un'eccezione*** (*throwing exception*).

Sollevamento di un'eccezione…

Quando si solleva un'eccezione vengono eseguite le seguenti operazioni:

1. viene creato l'***oggetto eccezione*** come un qualsiasi oggetto Java (nella memoria heap mediante l'operatore *new*);
2. il flusso di esecuzione corrente viene interrotto e il riferimento all'oggetto eccezione viene ***espulso*** dal contesto corrente;
3. il meccanismo di ***gestione delle eccezioni*** rileva la situazione e comincia a cercare il punto opportuno dove far proseguire l'esecuzione del programma.

…e sua gestione

Il punto opportuno da cui far riprendere l'esecuzione del programma si chiama **gestore dell'eccezione** (*exception handler*). Esso ha il compito di recuperare la situazione in modo tale che il programma possa o cercare qualche linea di esecuzione alternativa o semplicemente continuare.

La sezione di codice che potrebbe generare delle eccezioni e che è seguita dal codice per il loro trattamento, è chiamata *zona protetta* (**guarded zone**).

Modello a Terminazione o ripresa?

Supportata da Java e C++ : si presuppone che l'errore è talmente critico da non poter ritornare al punto in cui è stata sollevata l'eccezione.

Si riprende l'esecuzione dall'istruzione successiva al gestore della eccezione

Sollevare un'eccezione: l'istruzione *throw* ...

Per lanciare una eccezione in modo esplicito si utilizza l'istruzione *throw*. La sua sintassi generale è la seguente:

```
throw ThrowableInstance;
```

dove *ThrowableIstance* è istanza della classe *Throwable* o istanza di una sua sottoclasse.

Un semplice caso di sollevamento di eccezione potrebbe essere il passaggio di un riferimento ad un oggetto (ad esempio, `t`) non ancora inizializzato.

```
if(t == null) throw new NullPointerException();
```

dove `NullPointerException` è una classe predefinita di eccezioni.

…l'istruzione *throw*.

Esistono due costruttori per tutte le eccezioni standard:

- un costruttore di default
- costruttore che ha un argomento di tipo stringa in input.

```
if(t == null)
throw new
  NullPointerException("t==null");
```

Catturare una eccezione: Il blocco *try*

Se un metodo solleva un'eccezione, questa deve essere in qualche modo catturata (intercettata) e trattata.

Per catturare l'eccezione si può impostare un blocco speciale chiamato *try block* in cui si inserisce la sezione di codice che potrebbe generare l'eccezione.

La sintassi del *try block* è la seguente:

```
try {
// Code that might generate exceptions
}
```

Le parole chiave *try e catch*

Il gestore dell'eccezione segue il blocco *try* ed è introdotto dalla parola chiave *catch*.

Poiché in un blocco *try* si potrebbero sollevare più eccezioni di tipo differente, al blocco *try* è possibile far seguire diverse clausole catch, una per ogni tipo di eccezione che si vuole gestire.

I gestori delle eccezioni…

La sintassi (parziale) è la seguente

```
try {
// Code that might generate exceptions
} catch(Type1 id1) {// Handle
  exceptions of Type1
} catch(Type2 id2) {// Handle
  exceptions of Type2
}
```

…I gestori delle eccezioni…

Ogni proposizione ‘catch’ è come un piccolo metodo che ha come argomento un tipo particolare.

Gli identificatori (*id1*, *id2* e così via) possono essere usati all’interno del gestore come se fossero argomenti di metodi.

Anche se l’identificatore non viene utilizzato all’interno del blocco, deve essere sempre indicato.

I gestori devono essere inseriti subito dopo il *try block*.

…I gestori delle eccezioni…

Cosa accade a questo punto quando viene sollevata una eccezione ?

Sono eseguiti i seguenti passi:

1. Il meccanismo di gestione cerca il primo gestore il cui tipo di argomento corrisponde con il tipo dell'eccezione sollevata;
2. Quindi entra nel blocco del rispettivo 'catch' e l'eccezione viene gestita.
3. la ricerca dei gestori termina quando il blocco 'catch' è eseguito.

…I gestori delle eccezioni…

Dopo la gestione dell'eccezione si passa ad eseguire la prima istruzione dopo l'ultimo blocco *catch*.

Si noti che in questo caso non si ha bisogno di alcuna istruzione supplementare (come per esempio il *break* nello *switch*) per evitare che le restanti clausole *catch* siano eseguite.

Il gestore è unico, sebbene all'interno del *try block* diverse invocazioni di metodi potrebbero generare la stessa eccezione.

…I gestori delle eccezioni…

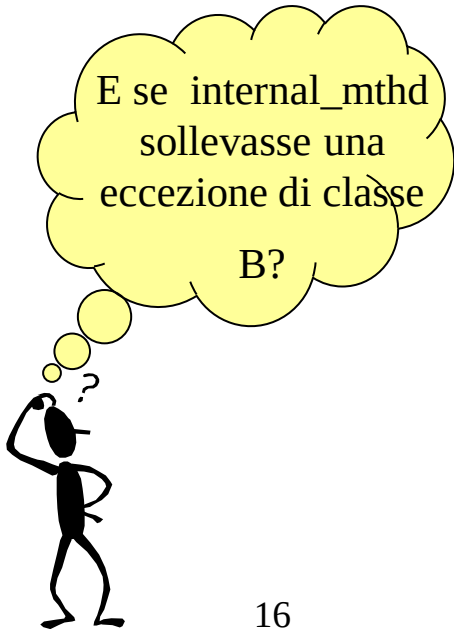
È possibile avere blocchi try annidati.

```
try {//istruzioni
      try  { //istruzioni
            } catch(A a1) { // gestore A}
// istruzioni
} catch(B b1) { // gestore B}
```

…I gestori delle eccezioni…

Più in generale i blocchi *try* potranno essere annidati ma in metodi differenti.

```
void external_mthd() { //istruzioni
try {// istruzioni
    internal_mthd()
    //istruzioni
} catch(B b1) { // gestore B}
}
void internal_mthd() { //istruzioni
try {// istruzioni
    } catch(A a1) { // gestore A}
}
```



Propagazione delle eccezioni

Se un'eccezione non viene intercettata e gestita quando si verifica, il controllo viene immediatamente restituito al metodo che ha invocato il metodo che l'ha prodotta.

Si può progettare il SW in modo che l'eccezione sia intercettata e gestita al livello più esterno, ma se non viene gestita neppure qui, il controllo passa al metodo che lo ha invocato e così via.

La propagazione delle eccezioni continua finché l'eccezione non viene intercettata e gestita, oppure fino a quando non si esce dal metodo main, nel qual caso termina il programma e si produce il messaggio d'errore relativo all'eccezione.

…I gestori delle eccezioni.

È possibile anche pensare a un gestore che catturi qualsiasi eccezione. Per fare ciò si cattura la classe base delle eccezioni *Exception*

```
catch(Exception e) {
System.err.println("Caught an exception");}
```

È consigliabile inserire questo gestore solo alla fine della lista degli altri gestori di eccezioni specifiche.

La parola chiave *finally*…

L'istruzione try può avere una clausola *finally* opzionale: essa definisce una sezione di codice che viene eseguita indipendentemente da come si sia usciti dal blocco *try.*

Spesso viene usata per garantire il rilascio della memoria o di altre risorse a prescindere da quello che accade nel blocco *try*.

Se non si verificano eccezioni, le istruzioni nella clausola *finally* vengono eseguite dopo che si è completato il blocco *try*.

Se si verifica un'eccezione nel blocco *try*, il controllo passa prima alla clausola *catch* appropriata e poi alla clausola *finally*.

…La parola chiave *finally*…

La clausola *finally*, se presente, dev'essere posta dopo tutte le clausole *catch*.

```
try { //istruzioni
} catch(A a1) { // gestore A
} catch(B b1) { // gestore B
} catch(C c1) { // gestore C
} finally {
// Activities that happen every time
}
```

Un blocco *try* non deve essere necessariamente seguito da una clausola *catch*: se non ci sono clausole *catch*, si può usare il solo blocco *finally*.

…La parola chiave *finally*.

Esempio

```
try
{
    System.out.println(Integer.parseInt(numString));
}
catch (NumberFormatException exception)
{
    System.out.println("Eccezione intercettata");
}
finally
{
    System.out.println("Fatto");
}
```

La clausola *throws*

Java fornisce anche una sintassi che permette di indicare quali sono le eccezioni sollevate da un metodo in modo tale che il programmatore possa gestirle.

Se un metodo può provocare una eccezione che non è in grado di gestire, deve generalmente specificare questo comportamento in modo tale che i metodi chiamanti si preparino a questa eccezione.

A tale scopo si utilizza la clausola *throws* nella dichiarazione del metodo che elenca i tipi di eccezione che un metodo dovrebbe sollevare.

…La clausola *throws*.

Questa sintassi viene chiamata *descrizione dell'eccezione* (**exception specification**) e fa parte della dichiarazione del metodo subito dopo la lista degli argomenti.

```
type    MethodName(argument    list)    throws
  exceptionList
{ //body
}
```

Java garantisce la verifica in compilazione della corretta gestione delle eccezioni.

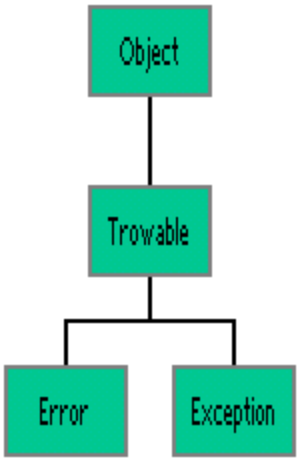
In questo differisce dal C++ che permette la cattura delle violazioni delle descrizioni delle eccezioni solo in esecuzione.

Eccezioni standard in Java…

Tutti i tipi di eccezione in Java sono derivati dalla classe *Throwable*.

Da *Throwable* sono derivate due sottoclassi che portano a due gerarchie di ereditarietà distinte.

Gerarchia di classi delle eccezioni



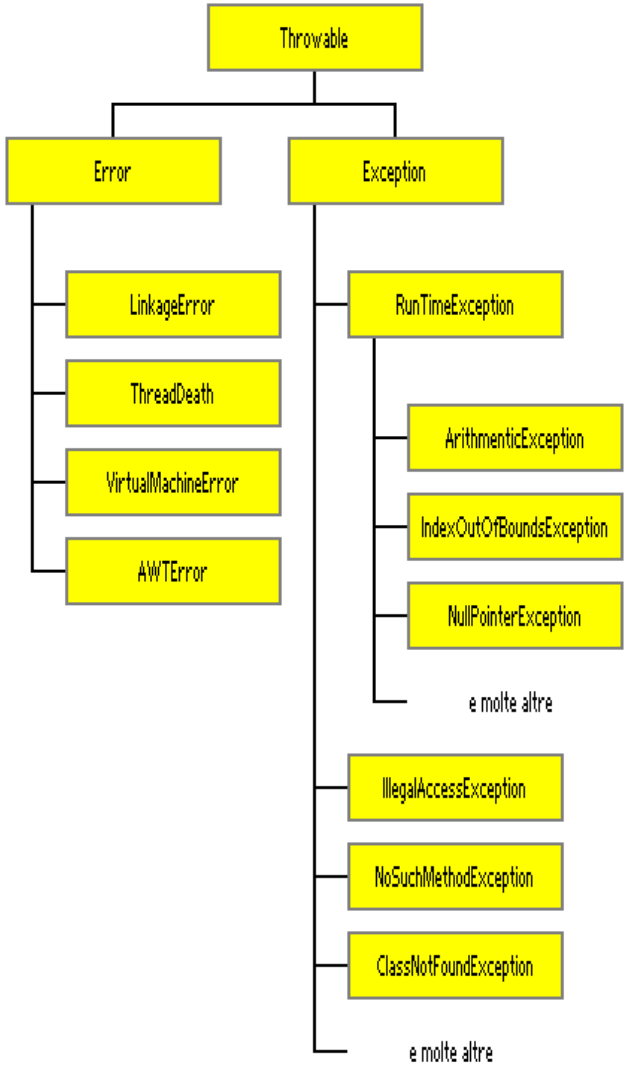
La prima fa capo alla classe *Exception* utilizzata per le condizioni eccezionali che i programmi utente dovrebbero catturare. Da questa classe saranno poi derivate tutte le sottoclassi relative alle eccezioni personalizzate.

…Eccezioni standard in Java…

L'altro ramo della gerarchia è capeggiata dalla classe *Error* che definisce le eccezioni da catturare in ***circostanze normali*** del programma. Questo tipo di eccezione viene utilizzata al run-time per indicare errori che interessano l'esecuzione del programma quali ad esempio il messaggio di overflow dello stack.

Java definisce diverse classi di eccezioni all'interno del package *java.lang* che viene implicitamente importato in tutti i programmi.

…Eccezioni standard in Java…



Le eccezioni più generali sono quelle derivate da *RuntimeException*. In Java queste eccezioni non devono essere inserite nell'elenco *throws* del metodo. Inoltre sono definite *eccezioni non controllate* perché il compilatore non controlla se un metodo gestisce o lancia queste eccezioni.

…Eccezioni standard in Java.

Il pacchetto *java.lang* contiene anche delle eccezioni che devono essere necessariamente nell'elenco *throws* del metodo se tale metodo è in grado di generare una di queste eccezioni e non la gestisce. Questo tipo di eccezioni sono chiamate *eccezioni controllate*.

In generale, è indispensabile elencare nella clausola *throws* le eccezioni sollevabili in un metodo e non gestite, escluse quelle di tipo *RuntimeException* e rispettive sottoclassi.

La creazione di eccezioni personalizzate…

Oltre alle eccezioni presenti nel package *java.lang* sono disponibili altri tipi di eccezioni correlate alle diverse librerie di classi.

Java permette di creare delle eccezioni personalizzate che servono a coprire i casi in cui non sono previste nelle librerie standard.

Per creare una nuova eccezione si cerca di derivarla da una già esistente che ha un significato analogo (dove è possibile farlo).

```
class MyException extends Exception {
public MyException() {}
public MyException(String msg) {super(msg);}
}
public class FullConstructors {
public static void f() throws MyException {
 System.out.println("Throwing MyException from f()");
 throw new MyException();
 }
public static void g() throws MyException {
 System.out.println("Throwing MyException from g()");
 throw new MyException("Originated in g()");
}
public static void main(String[] args) {
 try {
     f();
 } catch(MyException e) {
   e.printStackTrace(System.err);
 }
 try {g();}
 catch(MyException e) {  e.printStackTrace(System.err);}
 }
 }
```

Output:

```
Throwing MyException from f()
MyException
at FullConstructors.f(FullConstructors.java:16)
at FullConstructors.main(FullConstructors.java:24)
Throwing MyException from g()
MyException: Originated in g()
at FullConstructors.g(FullConstructors.java:20)
at FullConstructors.main(FullConstructors.java:29)
```