Eccezioni ed asserzioni

I concetti relativi ad eccezioni, errori ed asserzioni e le relative gestioni, permettono allo sviluppatore di scrivere del software robusto, ovvero che riesca a funzionare correttamente, anche in presenza di situazioni impreviste.

In JAVA è possibile definire un'eccezione come un situazione imprevista che il flusso di un'applicazione può incontrare, esse vengono gestite utilizzando le seguenti parole chiave:

- try
- catch
- finally
- throw
- throws.

Attraverso esse sarà possibile creare eccezioni personalizzate e decidere non solo come, ma anche in quale parte del codice gestirle, e questo è implementato nella libreria Java mediante la classe Exception e le sue sottoclassi.

Un esempio di eccezione è la divisione tra due variabili numeriche nella quale la variabile divisore ha valore 0.

Possiamo definire un errore come una situazione imprevista non dipendente da un errore commesso dallo sviluppatore ma essi a differenza delle eccezioni non sono gestibili, questo concetto è implementato nella libreria Java mediante la classe Error e le sue sottoclassi.

Un esempio di errore che potrebbe causare un programma è quello relativo alla terminazione delle risorse di memoria che ovviamente, non è gestibile.

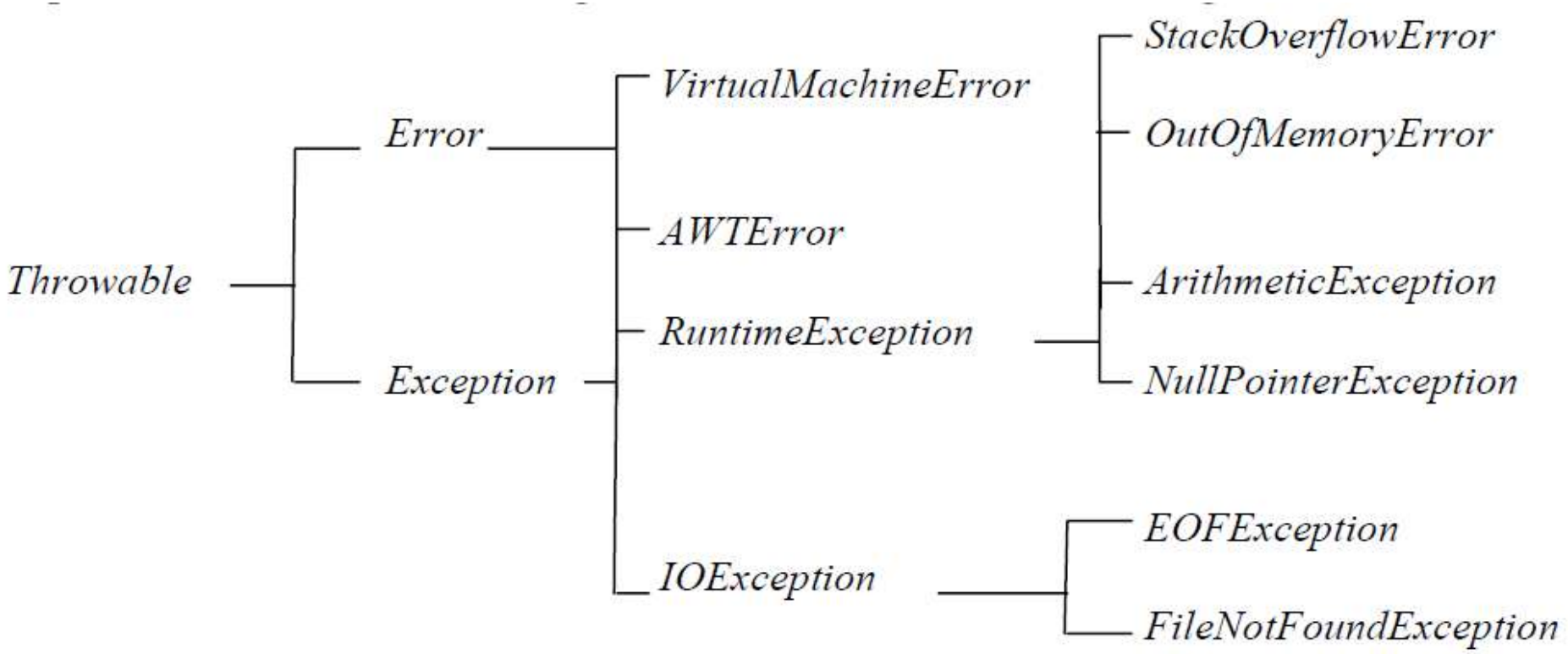
Infine è possibile definire un'asserzione come una condizione che deve essere verificata affinché lo sviluppatore consideri corretta una parte di codice.

A differenza delle eccezioni e degli errori, le asserzioni rappresentano uno strumento da abilitare per testare la robustezza del software, ed eventualmente disabilitare in fase di rilascio.

Questo concetto è implementato tramite la parola chiave assert.

Un esempio un'asserzione potrebbe essere quello di asserire che la variabile che deve fare da divisore in una divisione, deve essere diversa da 0, se questa condizione non dovesse verificarsi allora l'esecuzione del codice sarà interrotta.

Nella libreria standard di Java, esiste una gerarchia di classi che mette in relazione, la classe Exception e la classe Error essendo entrambe estensioni della superclasse Throwable.



La JVM reagisce ad un eccezione-errore nel seguente modo:

1. Se il programma genera un'eccezione, la JVM istanzia un oggetto dalla classe eccezione relativa al problema, e "lancia" l'eccezione appena istanziata (tramite la parola chiave throw).
2. Se il nostro codice non "cattura" (tramite la parola chiave catch) l'eccezione, il gestore automatico della JVM interromperà il programma generando in output informazioni dettagliate su ciò che è accaduto.

Per esempio, se si prova ad eseguire una divisione per zero tra interi,la JVM istanzierà un oggetto di tipo ArithmeticException e lo lancerà , è come se la JVM eseguisse le seguenti righe di codice:

```
ArithmeticException exc = new ArithmeticException();
throw exc;
```

Questo avviene in maniera totalmente trasparente.

Se bisogna sviluppare una parte di codice che potenzialmente può scatenare un'eccezione è possibile circondarlo con un blocco try seguito da uno o più blocchi catch. Per esempio:

```
public class Ecc1 {
public static void main(String args[]) {
int a = 10;
int b = 0;
int c = a/b;
System.out.println(c);
}}
```

Questa classe genererà un'eccezione durante la sua esecuzione, dovuta all'impossibilità di eseguire una divisione per zero.

In tal caso, la JVM dopo aver interrotto il programma produrrà il seguente output:

```
Exception in thread "main" java.lang.ArithmeticException: / by zero at
Ecc1.main(Ecc1.java:6)
```

Il messaggio evidenzia:

- il tipo di eccezione (java.lang.ArithmeticException)
- un messaggio descrittivo (/ by zero)
- il metodo in cui è stata lanciata l'eccezione (at Ecc1.main)
- il file in cui è stata lanciata l'eccezione (Ecc1.java)
- la riga in cui è stata lanciata l'eccezione (:6)

L'unico problema è che il programma è terminato prematuramente,ma utilizzando le parole chiave try e catch è possibile gestire l'eccezione in maniera personalizzata:

```
public class Ecc2 {
```

```
public static void main(String args[]) {
int a = 10;
int b = 0;
try {
int c = a/b;
System.out.println(c);}
catch (ArithmeticException  exc) {
System.out.println("Divisione per zero...");}
}}
```

Quando la JVM eseguirà tale codice incontrerà la divisione per zero della prima riga del blocco try, lancerà l'eccezione ArithmeticException che verrà catturata nel blocco catch seguente e quindi non sarà eseguita la stampa di c, bensì la stringa "Divisione per zero...", ed il programma di terminare in maniera naturale.

Si noti che il blocco catch deve dichiarare un parametro del tipo dell'eccezione che deve essere catturata.

Nell'esempio il reference exc, puntava all'eccezione che la JVM istanzia e lancia, e tramite esso è possibile reperire informazioni proprio sull'eccezione stessa invocando il metodo printStackTrace()

```
int a = 10;
int b = 0;
try {
int c = a/b;
System.out.println(c);}
catch (ArithmeticException exc) {
exc.printStackTrace();}
}
```

Il metodo printStackTrace() produrrà in output i messaggi informativi che il programma avrebbe prodotto se l'eccezione non fosse stata gestita, ma senza interrompere il programma stesso.

Come per i metodi, anche per i blocchi catch i parametri possono essere polimorfi, come avviene nel seguente frammento di codice:

```
int a = 10;
int b = 0;
try {
int c = a/b;
System.out.println(c);}
catch (Exception exc) {
exc.printStackTrace();}}
```

in cui il blocco catch gestirebbe qualsiasi tipo di eccezione, essendo Exception, la superclasse da cui discende ogni altra eccezione, ed il reference exc, è in questo esempio, un parametro polimorfo.

È anche possibile far seguire ad un blocco try, più blocchi catch, come nel seguente esempio:

```
int a = 10;
int b = 0;
try {
int c = a/b;
System.out.println(c);}
catch (ArithmeticException exc) {
System.out.println("Divisione per zero...");}
catch (NullPointerException exc) {
System.out.println("Reference nullo...");}
catch (Exception exc) {
exc.printStackTrace();}}
```

Il blocco `catch (Exception exc) {exc.printStackTrace();}` deve essere posizionata in coda agl'altri blocchi catch.

È anche possibile far seguire ad un blocco try, oltre a blocchi catch, un altro blocco finally, tutto ciò che è definito in esso viene eseguito in qualsiasi caso, sia se viene o non viene lanciata l'eccezione , ad esempio:

```
public void insertInDB() {
try {cmd.executeUpdate("INSERT INTO…");}
catch (SQLException exc) {exc.printStackTrace();}
finally {
try{connection.close();}
catch (SQLException exc) {
exc.printStackTrace();}}
```

Questo codice tenta di eseguire una "INSERT" in un database, tramite le interfacce JDBC del package java.sql, cmd è un oggetto Statement e connection è un oggetto Connection, il comando executeUpdate() ha come parametro il codice SQL e se ci sono errori la JVM lancerà una SQLException, che verrà catturata nel relativo blocco catch. In ogni caso la connessione al database deve essere chiusa.

Le eccezioni più frequenti sono Java:

- NullPointerException : probabilmente la più frequente tra le eccezioni. Viene lanciata dalla JVM, quando viene chiamato un metodo su di un reference che invece punta a null.
- ArrayIndexOutOfBoundsException : questa eccezione viene lanciata quando si prova ad accedere ad un indice di un array troppo alto.
- ClassCastException : eccezione particolarmente insidiosa. Viene lanciata al runtime quando si prova ad effettuare un cast ad un tipo di classe sbagliato.
- IOException e le sottoclassi FileNotFoundException, EOFException, etc…. del package java.io
- SQLException del package java.sql
- ConnectException, il package java.net

È comunque possibile definire nuovi tipi di eccezioni.

Ad esempio un programma che deve gestire in maniera automatica le prenotazioni per un teatro, potrebbe voler lanciare un'eccezione nel momento in cui si tenti di prenotare un posto non più disponibile.

In tal caso la soluzione è estendere la classe Exception, ed eventualmente aggiungere membri e fare override di metodi come toString().

```
public class PrenotazioneException extends Exception {
public PrenotazioneException() {
// Il costruttore di Exception chiamato inizializza la
// variabile privata message
super("Problema con la prenotazione");
}
public String toString() {
return getMessage() + ": posti esauriti!";
}
}
```

La JVM sa quando lanciare una ArithmeticException ma non sa quando lanciare una PrenotazioneException, e in questo caso si utilizza la parola chiave throw che lancia un'eccezione tramite la seguente sintassi:

```
PrenotazioneException exc = new PrenotazioneException();
throw exc;
//o equivalentemente
throw new PrenotazioneException();
```

Un codice di esempio

```
try {
//controllo sulla disponibilità dei posti
if (postiDisponibili == 0) {
throw new PrenotazioneException();}
//istruzione eseguita se non viene lanciata l'eccezione
postiDisponibili--;}
catch (PrenotazioneException exc){
System.out.println(exc.toString());}
```

Le asserzioni

Un'asserzione è un'istruzione che permette di testare eventuali comportamenti che un'applicazione deve avere.

Ogni asserzione richiede che sia verificata un'espressione booleana se questa non è verificata, allora si deve parlare di bug, quindi le asserzioni rappresentare un'utile strumento per accertarsi che il codice scritto si comporti così come ci si aspetta.

Esistono due tipi di sintassi per le asserzioni:

1. assert espressione_booleana;
2. assert espressione_booleana: espressione_stampabile;

Con la sintassi 1) quando l'applicazione esegue l'asserzione valuta il valore dell'espressione_booleana, se questo è true, il programma prosegue normalmente, ma se il valore è false viene lanciato l'errore AssertionError.

Per esempio l'istruzione:

```
assert b > 0;
```

è semanticamente equivalente a:

```
if (!(b>0)) {
throw new AssertionError();
}
```

La sintassi 2) permette di specificare anche un messaggio esplicativo tramite l'espressione_stampabile.

```
assert b > 0; "il valore di b è " + b;
```

Se la condizione che viene asserita dal programmatore è falsa, l'applicazione terminerà immediatamente mostrando le ragioni tramite uno stacktrace (metodo printStackTrace() di cui sopra).

È comunque possibile disabilitare la lettura delle asserzioni da parte della JVM, una volta rilasciato il proprio prodotto, al fine di non rallentarne l'esecuzione.