Capitolo 9

Assert in Java: tecniche e filosofia d'uso

Andrea Gini

Una delle novità più importanti del JDK 1.4 rispetto alle precedenti versioni è l'introduzione di un servizio di assert. Si tratta della più significativa modifica introdotta nel linguaggio dal 1997, data in cui vennero aggiunte le classi interne. Questa innovazione, tuttavia, si distingue dalla precedente per almeno due ragioni: per prima cosa, l'introduzione delle assert ha comportato significative modifiche alla JVM, cosa che rende il bytecode prodotto dal nuovo compilatore incompatibile con le JVM precedenti; in secondo luogo, con le assert viene introdotto per la prima volta nel linguaggio Java un costrutto di meta programmazione tipico della programmazione logica. Invece di ordinare al computer "cosa deve fare" in un determinato momento, una assert indica piuttosto "una condizione che non si dovrebbe dovrebbe mai verificare" in un certo punto del programma nel corso dell'esecuzione. L'esistenza di un simile costrutto permette di mettere in pratica una forma semplificata di programmazione per contratto, una tecnica di sviluppo che aiuta a realizzare programmi più robusti imponendo la dichiarazione di precondizioni, postcondizioni e invarianti.

Cosa sono le assert

Le assert sono istruzioni che controllano la verità di una condizione booleana, e provocano la chiusura del programma (mediante il lancio di un AssertionError) nel caso in cui tale condizione risulti falsa. Per esempio l'istruzione:

```
assert a + b > 0;
```

provocherà la chiusura del programma qualora la somma dei valori `a` e `b` sia uguale o inferiore a zero. In prima istanza, la semantica dell'assert si riconduce alla forma compatta di un'espressioni del tipo:

```
if( !(a+b>0))
  throw new AssertException();
```

Ma al di là dell'eleganza di un costrutto compatto, esistono sostanziali differenze tra i due casi, sia sul piano tecnico sia su quello filosofico. Da un punto di vista tecnico, il costrutto delle assert prevede la possibilità di disabilitare in blocco il controllo delle condizioni: come verrà mostrato più avanti, le assert vengono usate essenzialmente in fase di test e debugging: durante la normale esecuzione è possibile disabilitarle, eliminando in tal modo l'overhead legato alla loro gestione. Ma esiste anche una differenza assai sottile sul piano filosofico, che rende l'assert qualcosa di completamente diverso da qualsiasi altro costrutto presente in Java. Contrariamente a quanto avviene con i costrutti standard dei linguaggi imperativi, una assert non rappresenta un ordine ma una ipotesi: l'ipotesi che una certa condizione booleana sia vera in una determinata fase dell'esecuzione di un programma. La violazione di una assert causa la chiusura del programma, dal momento che si è verificato qualcosa che il programmatore non aveva previsto. Mai e in nessun caso una assert dovrà contenere direttive che influenzino la normale esecuzione del programma.

L'utilizzo delle assert permette al programmatore di verificare la consistenza interna di un programma al fine di renderlo più stabile. Nel contempo, le assert aggiungono espressività al codice, poiché formalizzano alcune ipotesi del programmatore circa lo stato del programma durante l'esecuzione, ipotesi che possono rivelarsi false a causa di un bug.

Per comprendere a fondo la filosofia di utilizzo delle assert, può essere utile una breve introduzione.

Sherlock Holmes e la filosofia delle Assert

Nel racconto "L'avventura degli omini danzanti" Sherlock Holmes, il più famoso detective letterario, si trova a dover affrontare un caso di omicidio, per il quale viene accusata la persona sbagliata. L'apparenza a volte inganna, ma la logica, se usata nel giusto modo, può aiutare a rimettere le cose a posto:

Lo studio era un locale non molto grande, coperto su tre pareti dai libri, con uno scrittoio di fronte ad una finestra che dava sul giardino. Per prima cosa, dedicammo la nostra attenzione al corpo del povero gentiluomo, la cui massiccia figura giaceva in mezzo alla stanza. Le vesti in disordine indicavano che era stato bruscamente risvegliato dal sonno. Il proiettile, sparato dal davanti, era rimasto nel corpo dopo avere attraversato il cuore. Non c'erano tracce di polvere da sparo, né sulla vestaglia né sulle mani. Secondo il medico, la signora invece mostrava tracce di polvere sul viso ma non sulle mani.

"L'assenza di tracce di polvere sulle mani non significa nulla; avrebbe avuto molto significato, invece, la loro presenza", disse Holmes. "Se il proiettile non è difettoso e la polvere non schizza

indietro, si possono sparare molti colpi senza che ne rimanga traccia. Ora suggerirei di rimuovere il corpo del signor Cubitt. Immagino, dottore, che lei non abbia recuperato il proiettile che ha ferito la signora?"

"Prima di poterlo recuperare occorre un complicato intervento chirurgico. Ma nella pistola ci sono ancora quattro proiettili. Due sono stati sparati, provocando due ferite, quindi il conto dei proiettili torna."

"Così sembrerebbe", convenne Holmes. "Forse può anche dirmi che fine ha fatto il proiettile che ha ovviamente colpito il bordo della finestra?"

Si era improvvisamente girato indicando, col lungo indice sottile, un foro che attraversava l'estremità inferiore del telaio della finestra, circa un pollice sopra il bordo.

"Per Giove!", esclamò l'ispettore. "Come diamine ha fatto a vederlo?"

"L'ho visto perché lo stavo cercando."

"Fantastico!", disse il dottore. "Lei ha senz'altro ragione, signore; e allora, ci deve essere stata una terza persona. Ma chi poteva essere, e come ha fatto ad andarsene?"

"È questo il problema che dobbiamo risolvere", disse Holmes. "Ispettore Martin, lei ricorderà che le domestiche hanno dichiarato di aver sentito odore di polvere da sparo uscendo dalle loro stanze, e che le ho detto che si trattava di un elemento di estrema importanza?"

"Sì, lo ricordo; ma confesso di non aver capito il motivo della sua raccomandazione."

"Ci porta a desumere che, al momento dello sparo, tanto la finestra che la porta della stanza erano aperte. Altrimenti, il fumo dell'esplosione non si sarebbe potuto diffondere così rapidamente nella casa. Per questo, bisognava che nella stanza ci fosse corrente. Porta e finestra, però, sono rimaste aperte solo per pochi minuti."

"Come può provarlo?"

"Perché la candela non aveva sgocciolato."

"Magnifico!", esclamò l'ispettore. "Magnifico!"

"Essendo certo che, al momento della tragedia, la finestra era aperta, ho pensato che nella faccenda poteva essere coinvolta una terza persona, che aveva sparato dall'esterno. Un proiettile diretto contro questa persona avrebbe potuto colpire il telaio della finestra. Ho cercato e, voilà, c'era il segno del proiettile."

La tecnica investigativa di Holmes è basata sulla deduzione: la sua massima più celebre è "Se escludi l'impossibile, ciò che rimane, per quanto improbabile, non può che essere la verità". Tuttavia in informatica non è permesso escludere l'impossibile: non esiste la possibilità di realizzare programmi che dimostrino la correttezza logica di un generico altro programma.

Quando Holmes afferma "L'assenza di tracce di polvere sulle mani non significa nulla; avrebbe avuto molto significato, invece, la loro presenza", egli enuncia una verità profonda: la logica deduttiva funziona soltanto in presenza di fatti di cui sia nota la condizione di verità. In assenza di fatti non è possibile dire con certezza se una determinata ipotesi sia vera o falsa. Ciò si accorda perfettamente con il pensiero del celebre scienziato informatico E. W. Dijkstra, che era solito sostenere che "il collaudo permette di dimostrare la presenza di errori in un programma, non la loro assenza".

Se è vero che non è possibile garantire l'assenza di errori, è comunque possibile seminare alcune "trappole" nei punti critici del codice, in modo da ottenere il maggior numero possibile

di indizi qualora si verifichi un errore inaspettato. Per semplificare questo lavoro è possibile ricorrere alle assert. Fondamentalmente, le assert vengono usate per controllare tre tipi di condizione: precondizioni, postcondizioni ed invarianti. Le precondizioni sono clausole che devono risultare vere prima che venga eseguita una determinata sequenza di operazioni; le postcondizioni al contrario devono essere vere alla fine della sequenza; gli invarianti infine sono condizioni che devono sempre risultare vere.

Si osservi come Holmes riesca a falsificare un'ipotesi dapprima verificando la violazione di una condizione invariante (il numero dei proiettili sparati è superiore a quello dei proiettili di una sola pistola), quindi controllando la verità di una precondizione (la finestra aperta) unitamente alla falsità di una postcondizione (l'assenza di sgocciolamento della candela). Grazie alle assert il programmatore, al pari di Holmes, può raccogliere indizi importanti, indizi che possono aiutare a verificare le proprie ipotesi grazie alla deduzione logica.

Sintassi delle assert

L'istruzione assert prevede due costrutti:

```
assert booleanExpression;
assert booleanExpression : message;
```

Il primo permette di specificare la condizione da controllare; il secondo contiene anche un messaggio da visualizzare in caso di violazione. Tale messaggio può contenere anche informazioni dettagliate sul caso che ha provocato il fallimento del programma, per esempio:

```
assert a + b > c : "La somma di " + a + " con " + b + " ha dato un risultato minore o uguale a " + c;
```

Compilazione ed esecuzione di codice con assert

L'uso delle assert richiede opzioni speciali in fase di compilazione e di esecuzione. Un esempio passo-passo dovrebbe aiutare a chiarire tutto quello che è necessario sapere. Si suggerisce quindi di copiare il seguente programma in un file dal nome AssertTest.java:

```
public class AssertTest  {
  public static void main(String args[]) {
    byte b = 0;
    for ( int i = 0; i <= 64; i++) {
      assert i >= 0;                                  // precondizione
      b = (byte)(i * 2);                              // assegnamento
      assert b >= 0 : "Valore inaspettato: b = " + b;  // postcondizione
      System.out.println("b = " + b);
    }
  }
}
```

La assert in quinta riga è un esempio di precondizione: essa attesta che la variabile i deve avere un valore positivo prima che venga eseguita l'operazione presente nella riga successiva. Dopo l'assegnamento in sesta riga, si può osservare invece un esempio di postcondizione, la quale asserisce che al termine dell'operazione il valore di b deve essere a sua volta positivo, dal momento che tale variabile contiene il valore di un numero positivo moltiplicato per 2. D'altra parte, il ciclo for in quarta riga contiene un errore logico: la condizione i<=64 farà in modo che l'operazione di casting presente nell'assegnamento produca, nel corso dell'ultima iterazione, un valore negativo, dal momento che una variabile di tipo byte non è in grado di contenere un valore superiore a 127. Si noti come un simile difetto logico possa nascere da un semplice errore di battitura.

Per fare in modo che il compilatore accetti il codice contenente assert, è necessario utilizzare lo speciale flag -source 1.4 da riga di comando. Tale soluzione è resa necessaria dall'esigenza di garantire la compatibilità con il codice precedente al JDK 1.4, che permetteva di usare la parola "assert" come un normale identificatore per nomi di variabile o di metodo:

```
javac -source 1.4 AssertTest.java
```

Per default le assert sono disabilitate; pertanto, se si prova a lanciare il programma con il comando:

```
java AssertTest
```

si ottiene il seguente output, che presenta come ultimo valore un numero negativo, ossia un errore dovuto all'overflow indotto dal ciclo for:

```
....
b = 122
b = 124
b = 126
b = -128
```

Per abilitare il controllo delle assert in fase di esecuzione, è necessario utilizzare il flag -ea:

```
java -ea AssertTest
```

In questo caso il programma terminerà prima di raggiungere l'overflow, e segnalerà la violazione della postcondizione:

```
....
b = 122
b = 124
b = 126
java.lang.AssertionError: Valore inaspettato: b = -128
        at AssertTest.main(AssertTest.java:7)
Exception in thread "main"
```

Come si può vedere, non è difficile utilizzare il costrutto assert nei propri programmi. I flag di attivazione dispongono comunque di una serie di opzioni avanzate, che verranno descritte nel prossimo paragrafo.

Abilitazione e disabilitazione selettiva

Oltre al flag -ea (Enable Assertion) è disponibile un flag complementare -da (Disable Assertion). Per entrambi è possibile specificare un parametro che può assumere i seguenti valori:

- *Nessun valore*: le assert vengono abilitate o disabilitate in tutte le classi (escluse le classi di sistema, disabilitate per default).
- *NomeDiPackage*: le assert vengono abilitate o disabilitate nel package indicato e in tutti i suoi sotto package.
- ... : abilita o disabilita le assert nel package di default.
- *NomeDiClasse*: Abilita o disabilita le assert nella classe specificata.

Grazie a questi flag è possibile specificare in modo preciso e dettagliato la modalità di esecuzione. Per esempio, la seguente riga di comando esegue la classe AssertionTest, dopo aver abilitato le assert nel package it.mokabyte.provaAssertion e nei suoi eventuali sotto package:

```
java -ea:it.mokabyte.provaAssertion... AssertionTest
```

È possibile replicare ciascuno di questi flag, in modo da ottenere il risultato desiderato. Il seguente comando lancia la classe AssertionTest dopo aver abilitato le assert nel package it.mokabyte.provaAssertion e nei suoi eventuali sotto package, con l'esclusione del sottopackage subPackage1 e della classe Class1A:

```
java -ea:it.mokabyte.provaAssertion... -da:it.mokabyte.provaAssertion.subPackage1...
  -da:it.mokabyte.provaAssertion.Class1A AssertionTest
```

I flag -ea e -da permettono di abilitare o disabilitare le assert su qualsiasi package, compresi i package di sistema. Tuttavia, quando si usano i flag senza parametro, le assert sono disabilitate sulle classi di sistema. Per gestire in modo esplicito l'abilitazione o la disabilitazione delle assert nelle classi di sistema, è possibile ricorrere ai flag -esa (Enable System Assertions) e -dsa (Disable System Assertions).