Asserzioni in Java

fondamenti

Cosa è un'asserzione?

- Una "assertion" è una espressione booleana che deve essere "true" se e solo se il codice sta funzionando correttamente.
- Se l'asserzione risulta falsa, viene segnalato l'errore e bisogna correggere il codice.
- Le asserzioni permettono di inserire un meccanismo di "sanity check".
- Questi controlli sono utilizzati nella fase di test e sviluppo (**non** di produzione come le eccezioni).
- Il controllo delle asserzioni può essere attivato o disattivato a "runtime".
- Per default il controllo delle asserzioni è disabilitato.

Uso ‘appropriato’

1. Le asserzioni possono essere utilizzate come *precondition* nei metodi non pubblici: i metodi pubblici devono fare un controllo esplicito e sollevare un eccezione.
2. Le asserzioni non vanno usate per controllare l’input dell’utente, i parametri a linea di comando, etc.
3. Le asserzioni possono sempre essere utilizzate come *postconditions* e *loop invariants*.

Sintassi

```
assert condizione [: messaggio];
```

Se la condizione è falsa, solleva un errore contenente il valore del messaggio (che è opzionale).

In messaggio è possibile specificare una stringa ma anche la chiamata di un metodo che non ritorni void. Questo verrà invocato solo se l’asserzione è false

La seconda espressione è utilizzata come un messaggio che dettaglia l’AssertionError sollevato

Semplice esempio (postcondition)

```
public class EsempioAssertion {

        public void mioMetodo(int valore) {
                Object foo = null;

                // … elaboro l'oggetto foo

                // Controllo che l'oggetto sia valorizzato (postcondition)
                assert foo != null: "Impossibile ottenere foo con valore " + valore;
        }

        public static void main(String[] args) {
                EsempioAssertion mioOggetto = new EsempioAssertion();
                mioOggetto.mioMetodo(10);
        }
}
```

Attenzione allo statement "assert"

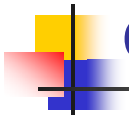
- Lo scopo dello statement `assert` è quello di fornire un modo per intercettare errori "in anticipo"
- Lo statement `assert` è presente da Java 1.4
- La nuova parola del linguaggio potrebbe generare errori in programmi che utilizzano assert come identificativo di proprietà.
  - C'è quindi un modo per scongiurare questo pericolo
  - Infatti sarebbe un vero e proprio errore di sintassi
- C'è la possibilità di "turn off" la parola chiave "assert"
  - Quando è "turn off", java semplicemente ignora lo statement.
  - Sfortunatamente questo significa che gli `assert` potrebbero essere "spenti" quando invece si desidera che facciano il loro lavoro

AssertionError

- `AssertionError` è un Error, non un Exception
  - Non c'è bisogno di metterlo in un try/catch
  - Quindi l'uso di assert non necessita di altro "lavoro extra" da parte del programmatore
- La seconda espressione non è sempre necessaria
  - Se c'è un AssertionError, java automaticamente fornisce uno stacktrace completo di numero di riga
  - Utilizzare il secondo argomento solo se è utile aggiungere ulteriori informazioni al messaggio di errore

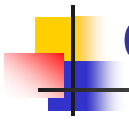
Asserzioni vs Eccezioni

- Quando usare asserzioni invece di eccezioni?
  - Entrambi intercettano problemi del programma, ma...
  - Il loro scopo è molto diverso!
- L'eccezione dice all'utente del programma che qualcosa è andato storto.
- L'asserzione documenta qualcosa relativamente al programma
  - Quando fallisce infatti segnala che c'è un bug
- Si creano eccezioni quando si ha a che fare con problemi che possono insorgere.
- Si scrivono asserzioni per "salvaguardare" cose che si conoscono (o si dovrebbero conoscere) relativamente al comportamento del programma



Quando usare eccezioni

- Usa eccezioni quando:
  - Serve verificare se i parametri di metodo/costruttore pubblico sono validi
    - Se sono public infatti altre persone oltre allo sviluppatore possono invocarli
- In breve,
  - Pensa a te stesso come l'autore della classe in oggetto
  - Qualunque cosa che potrebbe "andare storto" sulla quale l'autore non ha controllo all'interno della stessa determina un'eccezione



Quando usare asserzioni

- Frequentemente!
- Intendi le asserzioni come "cheap to write"
  - Inseriscile nel codice ogni volta ne senti il bisogno
  - Esempio: `assert eta >= 0;` è molto semplice e veloce
- Non devono essere intese come uno strumento di debug (sebbene possano essere usate anche per questo...)
- Piuttosto dovrebbero essere usate per specificare cose che devono essere vere in vari punti del codice
  - Forniscono documentazione, non solo controllo sugli errori
  - Sono di fatto documentazione "automatica" che può essere verificata dal programma stesso!

Invariants

Internal invariant sono fatti che si crede debbano essere sempre veri ad un certo punto di esecuzione di un programma

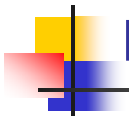
Control-flow invariant è l'assunzione che il programma sarà eseguito in un certo ordine. Ad uno step ne seguirà un altro. Certe righe saranno eseguite ed altre no

- NB: E' un errore inserire codice dove java sa che non potrà mai raggiungere (es. Immediatamente dopo un return)

Preconditions, Postconditions fatti che devono essere sempre veri rispettivamente quando un metodo è invocato e quando la sua esecuzione è terminata con successo

- NB: E' un errore utilizzare le assertions come preconditions in metodi pubblici

Class invariant è la condizione che un oggetto deve sempre soddisfare per essere un membro valido della classe, eccetto durante la transizione da uno stato ad un altro



Esempi (internal invariants)

```
if (x < 0) {
    . . .
}
else if (x == 0) {
    . . .
}
else {
    assert x > 0;
    . . .
}
```

```
switch(suit) {
   case Suit.CLUBS:
      …
      break;
   case Suit.DIAMONDS:
      …
      break;
   case Suit.HEARTS:
      …
      break;
   case Suit.SPADES:
      …
      break;
   default:
      assert false: suit;
}
```

Esempi (control flow invariants)

```
void foo() {
    for (...) {
        if (...)
            return;
    }
    assert false; // non dovrebbe mai finire qui
}
```

Dove non usare le asserzioni

- Non usare le asserzioni per fare lavoro “necessario”
  - Le asserzioni possono essere disabilitate (“turned off”)
  - Non è una buona idea avere un programma che funziona correttamente solo con le asserzioni abilitate (“turned on”)
- Non usare le asserzioni per controllare gli argomenti di metodi pubblici
  - Il controllo degli argomenti è parte del “contratto”, deve funzionare anche senza le asserzioni
  - Ci sono le eccezioni per questo: IllegalArgumentException, NullPointerException, e IndexOutOfBoundsException, ecc…

Asserzioni possono essere “illegali” (*disallowed)*

- Vecchie versioni di Java non le implementano
  - E’ possibile che nel codice `assert` sia stato usato per nome di variabile o metodo
  - Dobbiamo consentire l’uso di vecchi programmi
  - Quindi potremmo aver bisogno di rimuovere assert come keyword del linguaggio
- Non ci sono problemi con vecchi compilati (.class) che usano assert come un nome definito dall’utente
- Il problema potrebbe sorgere quando ricompiliamo il vecchio sorgente
  - Nelle JDK 1.4 di default javac non usa però la nuova modalità
  - Nota bene: il compilatore javac è cambiato, *non* l’interprete java

Compilazione Sorgenti

Per compilare dei sorgenti utilizzando le assertion è necessario utilizzare la sintassi.

```
javac –source 1.4 NomeClasse.java
```

Se l’opzione –source 1.4 viene dimenticata, la parola chiave “assert” verrà interpretata come un identificatore.

Asserzioni On/Off in pratica

Per abilitare/disabilitare le asserzioni si utilizza l'opzione –ea e –da

Es:

- `java –ea NomeClasse`
- `java –da NomeClasse`
- `java -ea:UnaClasse NomeClasse`
- `java -da:package... NomeClasse`
- `java -ea:package...`
  `-da:package.Subpkg0... NomeClasse`

Per indicare un package si usano i "..."

Assicurarsi che siano abilitate

- Il seguente codice, posto in testa alla vostra classe, controlla se sono abilitate

```
public static boolean isAssertionEnabled() {
   boolean assertionEnabled = false;
   assert (assertionEnabled == true);
   return assertionEnabled;
}
```