Programmazione Orientata agli Oggetti in Linguaggio Java

Test e Correzione: Tecniche di Test

versione 1.1

G. Mecca – Università della Basilicata – mecca@unibas.it

Sommario

- Tecniche di Test
- Test di Unità
  - ⇨Classi di Test
- Test di Regressione

Tecniche di Test

- Dopo la scrittura del codice
  - ⇨è necessario condurre le verifiche (test)
  - ⇨ed eliminare gli eventuali errori logici
- Il metodo tipico di test
  - ⇨test funzionali manuali e interattivi
  - ⇨il programmatore esegue una serie di test interagendo interattivamente col sistema
  - ⇨e verifica che il comportamento sia quello atteso

Tecniche di Test

- Test funzionali
  - ⇨test del sistema nella sua interezza e “dall’esterno” (dal punto di vista dell’utente)
  - ⇨vengono eseguiti i diversi casi d’uso per verificare che il comportamento sia corretto
  - ⇨il sistema è visto come una “scatola nera”
  - ⇨e ne viene verificato solo il funzionamento esterno senza entrare nel merito dei componenti interni

Tecniche di Test

- Piano dei test
  - ⇨è necessario mettere a punto una serie di casi di test (“test case”)
  - ⇨che coprano i vari casi d’uso dell’applicazione
  - ⇨in tutte le condizioni significative di utilizzo
  - ⇨tipicamente: vari casi di test per ciascun caso d’uso

Tecniche di Test

- Esempio: la morra cinese
  - ⇨due casi d’uso, vari casi di test
- Caso d’uso: “Utente gioca partita”
  - ⇨test n. 1: partita intera vinta dall’utente
  - ⇨test n. 2: partita intera vinta dal computer
  - ⇨test n. 3: partita interrotta dall’utente
- Caso d’uso: “Utente visualizza punteggi”
  - ⇨un test dopo ogni partita

Tecniche di Test

- Che fare nel caso di errori logici ?
  - ⇨test in cui il codice si comporta in modo diverso da quello atteso
- In questi casi
  - ⇨è necessario capire la causa dell'errore
  - ⇨correggere il codice
  - ⇨ripetere integralmente l'esecuzione del piano dei test

Tecniche di Test

- Svantaggi di questo approccio
  - ⇨ce ne sono vari molto gravi
- Primo svantaggio: solo test funzionali
  - ⇨eseguire SOLO test funzionali non fornisce indicazioni immediate sulle cause reali di un comportamento scorretto
  - ⇨bisogna andare a studiare il codice per risalire dal comportamento scorretto al componente responsabile

Tecniche di Test

- Secondo svantaggio: test manuali
  - ⇨i test sono eseguiti interattivamente
  - ⇨il programmatore è costretto a ripeterli tutte le volte che effettua modifiche
  - ⇨processo noioso e aperto agli errori
- In concreto
  - ⇨i programmatori raramente eseguono in forma sistematica test manuali

Tecniche di Test

- Le soluzione al primo problema
  - ⇨scrivere test sui singoli componenti e non solo test funzionali su tutta l'applicazione
  - ⇨test di unità ("unit test")
- La soluzione al secondo problema
  - ⇨automatizzare l'esecuzione dei test
  - ⇨in modo che siano ripetibili con poco sforzo dopo ciascuna modifica
  - ⇨test di regressione ("regression tests")

Test di Unità

- Nella programmazione a oggetti
  - ⇨l'organizzazione del codice in componenti suggerisce una modalità più raffinata di test
  - ⇨oltre ai test funzionali (indispensabili) è consigliabile anche verificare i singoli componenti
  - ⇨in condizioni isolate gli uni rispetto agli altri
  - ⇨in modo da verificare se si comportano correttamente da soli

Test di Unità

- Test di Unità
  - ⇨test effettuato su una singola classe di componenti (una classe e tutti i suoi oggetti)
  - ⇨per quanto possibile in modo isolato rispetto agli altri componenti
  - ⇨tipicamente: una serie di test per ciascun metodo dell'interfaccia del componente
- Differenze rispetto ai test funzionali
  - ⇨non vengono verificati interi casi d'uso

Test di Unità

- Scrivere test di unità
  - ⇨un metodo tipico: le classi di test
- Classe di test (Classe di Collaudo)
  - ⇨idea: per ciascuna classe da verificare, ne scrivo un'altra di collaudo
  - ⇨con un metodo main che chiama i metodi della classe ed effettua stampe dei risultati
  - ⇨vantaggio: non devo implementare schermi o casi d'uso complessi

Esempio: Test della Classe Punto

ATTENZIONE: la classe di test non implementa un caso d'uso dell'applicazione; il suo obiettivo è quello di verificare la correttezza dei metodi di una classe specifica

```
package collaudo;

import segmenti.Punto;

public class TestPunto {

   public static void main(String[] args) {
      Punto p = new Punto();
      p.setAscissa(10);
      p.setOrdinata(20);
      System.out.println("Punto: " + p.toString());
      System.out.println("Quadrante: " + p.getQuadrante());
   }
}
```

Test di Unità

- Per eseguire il test
  - ⇨il programmatore avvia la classe di collaudo
  - ⇨e verifica che i risultati delle stampe siano corretti
  - ⇨si tratta di un passo avanti, ma anche in questo caso il processo è noioso e ripetitivo
- Soluzione
  - ⇨automatizzare completamente i test

Test di Regressione

- Test di regressione
  - ⇨test la cui esecuzione è automatizzata
  - ⇨eseguibile rapidamente, ripetutamente, senza intervento interattivo
  - ⇨“eseguire i test deve costare quanto schiacciare un tasto”
- Piano di test automatizzato
  - ⇨“batteria” di test di regressione

Test di Regressione

- Vantaggio di questo approccio
  - ⇨è possibile effettuare più facilmente modifiche al codice
  - ⇨scoprendo presto eventuali “regressioni” (passi indietro), ovvero errori introdotti dalle modifiche
  - ⇨in modo da poterli eliminare concentrandosi solo sulle porzioni di codice scorrette
  - ⇨i test diventano un “paracadute”

Test di Regressione

- Come sono organizzati i test
  - ⇨per ogni classe dell’applicazione, una classe di test
  - ⇨contenente vari metodi di test
- Metodo di test
  - ⇨metodo che utilizza uno o più metodi del componente da verificare
  - ⇨sulla base di dati stabiliti dal programmatore
  - ⇨e facendo asserzioni sul risultato atteso

Test di Regressione

- Asserzione
  - ⇨espressione a valori booleani
  - ⇨in questo caso viene utilizzata per verificare il risultato fornito dal metodo del componente verificato
  - ⇨il test è superato se l'asserzione assume valore vero, altrimenti viene tipicamente lanciata un'eccezione

Test di Regressione

- Esempio: il metodo getQuadrante()
  - ⇨Punto p = new Punto();
  - ⇨p.setAscissa(10);
  - ⇨p.setOrdinata(20);
- Dopo queste istruzioni
  - ⇨posso asserire che
  - ⇨p.getQuadrante() == 1

Test di Regressione

- Un esempio di metodo per le asserzioni
  - ⇨riceve una stringa (descrizione dell'asserzione) ed un'espressione e verifica che l'espressione sia vera

```
public void assertTrue(String s, boolean asserzione) {
    if (asserzione) {
        System.out.print(".");
    } else {
        throw new AssertionFailedException(s);
    }
}
```

NOTA: il metodo si chiama assertTrue perchè assert è una parola chiave di Java

Un Esempio di Classe che Effettua Test di Regressione su Punto

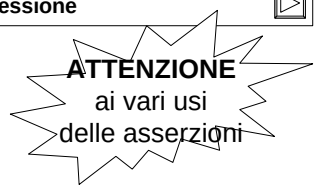
```
package varie;
import segmenti.Punto;
public class TestPunto {
    public void assertTrue(String s, boolean asserzione) {
        if (asserzione) {
            System.out.print(".");
        } else {
            throw new AssertionFailedException(s);
        }
    }
    public void testPunto1() {
        Punto p = new Punto();
        p.setAscissa(10); p.setOrdinata(2);
        assertTrue("quadrante 1", p.getQuadrante() == 1);
    }
    public void testPunto2() {
        Punto p = new Punto();
        p.setAscissa(-1); p.setOrdinata(2);
        assertTrue("quadrante 2", p.getQuadrante() == 2);
    }
    public static void main(String[] args) {
        TestPunto test = new TestPunto();
        test.testPunto1();
        test.testPunto2();
    }
}
```

Test di Regressione

- In altri termini
  - ⇨i test di regressione non vengono eseguiti in modo interattivo
  - ⇨ma “esercitando” i metodi del componente o dei componenti da verificare
  - ⇨ed effettuando asserzioni sui risultati
  - ⇨al termine viene prodotto un rapporto dei test “passati” e degli eventuali test “falliti”

Test di Regressione

- In questo modo
  - ⇨è molto più facile individuare dove nascono i problemi
- In Java
  - ⇨esistono framework molto utilizzati per lo sviluppo di test di regressione
  - ⇨in particolare: JUnit (http://www.junit.org) >>



Test di Regressione

- Uso delle asserzioni
  - ⇨l'idea generale è quella di aggiungere istruzioni con cui il codice si autoverifica
  - ⇨nel caso delle guardie, la verifica è una verifica "a priori"; l'asserzione è "utile" quando fallisce -> segnala un errore allo sviluppatore
  - ⇨nel caso dei test di regressione, la verifica è "a posteriori"; l'asserzione è utile quando è verificata -> segnala che il metodo è corretto

Test di Regressione

- In altri termini
  - ⇨le asserzioni utilizzate per le guardie servono a verificare "precondizioni" per l'esecuzione corretta dei metodi
  - ⇨le assezioni utilizzate nei test servono a verificare "postcondizioni" dopo l'esecuzione dei metodi

Riassumendo

- Tecniche di Test
- Test di Unità
  - ⇨Classi di Test
- Test di Regressione

Termini della Licenza