JUNIT

Unit testing alla XP

JUnit è un framework per l'automazione del testing di unità di programmi Java che si ispira al eXtreme Programming

Ogni test contiene le asserzioni che controllano che il programma non contiene difetti

Nota: XP è molto di più che JUnit, è proprio un modo nuovo di organizzare la fase implementativa (e il modo di lavorare) che coinvolge anche la fase di testing

eXtreme Programming

Coding

cliente sempre disponibile, seguire gli standard, pair programming, proprietà collettiva del codice, nessun straordinario

Testing

Tutto il codice deve avere test di unità, quando un bug è trovato, vengono creati i test, è completamente automatizzato

Planning

scrivere casi d'uso degli utenti, frequenti release, incontri frequenti, iterazioni continue

Designing

semplicità, limitare le funzionalità, refactor, usa metafora del sistema

Sviluppo guidato dai test

Lo sviluppo guidato dai test "Test-Driven Development Cycle" prevede i seguenti passi:

2. **scrivi i casi di test**
   - dalle specifiche (casi d'uso e storie dell'utente) scrivi un possibile scenario di chiamata di metodi e/o classi
3. **esegui i test (che falliscono)**
4. **scrivi il codice fino a quando tutti i casi di test passano**
5. **ricomincia da 1**

- Se trovi un difetto ricomincia da 1: scrivi i casi di test che falliscono per colpa del difetto e poi modifica il codice

Principi dello sviluppo guidato dai test

Il progetto e il codice evolvono sotto la guida di test e scenari reali

Anche se ti rallentano inizialmente, alla fine portano più vantaggi che svantaggi

- il codice è doppio (applicazione + test)

I test fanno parte della documentazione dell'applicazione

L'unit testing deve essere fatto automaticamente

- il giudizio che un metodo/classe sia corretto deve essere fatto dal codice stesso

Alcuni tools

Ci sono diversi tools per Unit testing nei diversi linguaggi

Per C++:

- CUnit/ CPPUnit

Per Java il più diffuso è JUnit

- http://junit.sourceforge.net/
- scritto da Kent Beck (autore di XP) e Erich Gamma (autore dei design pattern)

Junit

Usa la riflessione di Java (i programmi Java possono esaminare il loro stesso codice)

supporta i programmatori nel:

- definire ed eseguire test e test set
- formalizzare in codice i requisiti sulle unità
- scrivere e debuggare il codice
- integrare il codice e tenerlo sempre funzionante

Integrato con molti IDE (eclipse, netbeans, …)

L'ultima versione 4 sfrutta le annotation di Java

- useremo questa versione, codice molto più semplice

Mini Esempio con JUnit 4

- Un caso di test consiste in una classe ausiliaria
  - con alcuni metodi annotati `@Test`
  - in cui si controlla la corretta esecuzione del codice da testare mediante istruzioni come `assertEquals`

Esempio per testare l'operazione *:

```
...
public class HelloWorld {
@Test public void testMultiplication() {
     //Testing if 2*2=4:
     assertEquals("Mult", 4, 2*2);
}}
```

Unit test di una classe

Si voglia testare una classe Counter che rappresenta un contatore

le operazioni di Counter sono:

- **il costruttore che crea un contatore e lo setta a 0**
- **il metodo inc che incrementa il contatore e restituisce il nuovo valore incrementato**
- **il metodo dec che decrementa il contatore e restituisce il nuovo valore**

Schema della classe

Dovremmo scrivere prima il caso di test e poi la classe Counter, per chiarezza scriviamo prima lo scheletro di Counter:

```
public class Counter {
      public Counter(){}
      public int inc() {}
      public int dec() {}
}
```

Test di una classe

Per testare una classe X si crea una classe ausiliaria (in genere con nome XTest)

XTest contiene dei metodi annotati con @Test che rappresentano i casi di test

Testare Counter

Per testare l'unità Counter, in particolare i metodi inc e dec, creiamo una nuova classe CounterTest (o qualsiasi nome) così:

```
import static org.junit.Assert.*;
import org.junit.Test;

public class CounterTest {
   @Test public void testInc() {}
   @Test public void testDec() {}
}
```

import delle istruzioni di assert

annotazioni per dire che sono metodi che testano altri metodi

Test di un metodo

Nel singolo metodo di test testiamo ogni metodo della classe sotto test

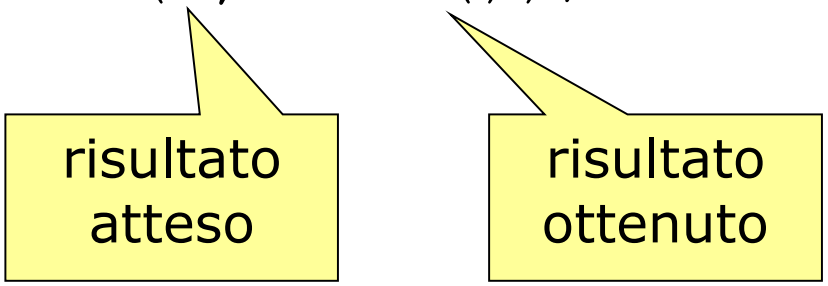
Dobbiamo:

3. creare eventuali oggetti delle classe sottotest
4. chiamare il metodo da testare e ottenere il risultato
5. confrontare il risultato ottenuto con quello atteso
   - **per far questo usiamo dei metodi assert di Junit che permettono di eseguire dei controlli**
   - **se un assert fallisce, JUnit cattura il fallimento e lo comunica al tester**
6. ripetiamo da 1

Test di inc()

In `testInc`() creiamo un'istanza di Counter, lo incrementiamo con il metodo `inc()` e controlliamo (mediante l'istruzione `assertEquals`) che inc() funzioni correttamente:

```
@Test
public void testInc() {
      Counter c = new Counter();
      assertEquals(1,c.inc());
}
```



Metodi assert

Ci sono molti metodi assert:

- **assertEquals**(*expected*, *actual*)
- **assertEquals**(String ***message***, *exp*, *act*)
  - **per controllare l'uguaglianza (con equals) di exp e act**
- **assertTrue**(expression)
  - per controllare che expression sia vera
- **assertNull**(Object ***object***)
- **fail**() e **fail**(String ***message***)
  - per terminare con un fallimento
- **assertSame**
  - per usare == invece che equals per il confronto

JUnit in Eclipse (1)

E' consigliabile usare un IDE come eclipse che assiste nella scrittura ed esecuzione dei casi di test

Per scrivere una caso di test:

3. scrivi la tua classe al solito (almeno lo scheletro)
4. seleziona la classe per cui vuoi creare i casi di test con il tasto destro -> new -> JUnit Test Case
5. appare un dialogo: selezione JUnit 4 e deseleziona tearDown, setUp ... - non sono necessari per piccoli esercizi

JUnit in Eclipse (2)

1. fai next -> seleziona i metodi per cui vuoi creare i casi di test
2. riempi i metodi di test con il codice che faccia i controlli opportuni

Per eseguire un caso di test in eclipse:

- tasto destro sulla classe di test -> run As -> Junit Test
- appare un pannello con una barra che rimane verde se tutto va bene

In sintesi

- Abbiamo visto che JUnit:
  - fa parte della metodologia eXtreme Programming;
  - adotta lo sviluppo guidato dai test;
  - è completamente automatico;
- Ricordate che in JUnit:
  - per testare una classe utilizzo un'altra classe
  - per testare i metodi uso dei metodi annotati @Test;
  - nei metodi di test eseguo i metodi da testare
  - e controllo la corretta esecuzione con gli assert
- Infine:
  - eclipse supporta JUnit 4