Fondamenti di informatica
Oggetti e Java

Luca Cabibbo

Uso di JUnit

ottobre 2012

JUnit

JUnit è uno strumento per assistere il programmatore Java nel testing

- JUnit consente di scrivere test di oggetti e classi Java
- i test sono automatizzati – ovvero, è possibile eseguire in modo automatico un gran numero di test – ed ottenere una risposta sintetica dell'esito del test
  - barra verde: test passato
  - barra rossa: test fallito – è poi possibile vedere in dettaglio in quali casi è fallito il test
- per test case si intende una classe di test – composta da uno o più metodi di test
- attenzione, l'uso di JUnit è basato su alcune caratteristiche di Java che non saranno spiegate in dettaglio in questo corso
  - tuttavia, è possibile usare Junit anche senza comprendere tutto in dettaglio

Metodi di test

Useremo JUnit per effettuare il test di un metodo

- per semplicità, di un metodo parametrico che calcola e restituisce un valore (niente effetti collaterali)
- il test di un metodo richiede di eseguire il metodo per ciascun dataset scelto per la verifica del metodo
- intuitivamente, va definito un metodo di test per ciascun dataset scelto

Ciascun singolo metodo di test

- ha lo scopo di verificare il funzionamento del nostro codice in un singolo caso – con riferimento a uno specifico dataset
- contiene un'invocazione del metodo in esame
- ha lo scopo di confrontare il comportamento atteso del metodo con il comportamento effettivo – ovvero, di confrontare il risultato atteso con il risultato effettivamente calcolato dal metodo
- questo confronto viene realizzato mediante un'asserzione

Metodi di test – esempio

Supponiamo di voler verificare un metodo **int fattoriale(int n)** di una classe **Fattoriale** che calcola il fattoriale di un numero naturale **n**

- quali dataset per verificare questo metodo?
  - è necessario verificare il funzionamento del metodo almeno per i seguenti valori di **n**: 0, 1 e 4
- quali asserzioni per verificare questo metodo?
  - il fattoriale di 0 vale 1
  - il fattoriale di 1 vale 1
  - il fattoriale di 4 vale 24

Come vedremo, questi ragionamenti sono necessari alla predisposizione di un test

- JUnit ci fornisce uno strumento per realizzare un test di questo tipo in modo semplice

JUnit, in pratica

Per prima cosa, è necessario avviare Eclipse e creare un nuovo progetto Java o aprire un progetto Java esistente

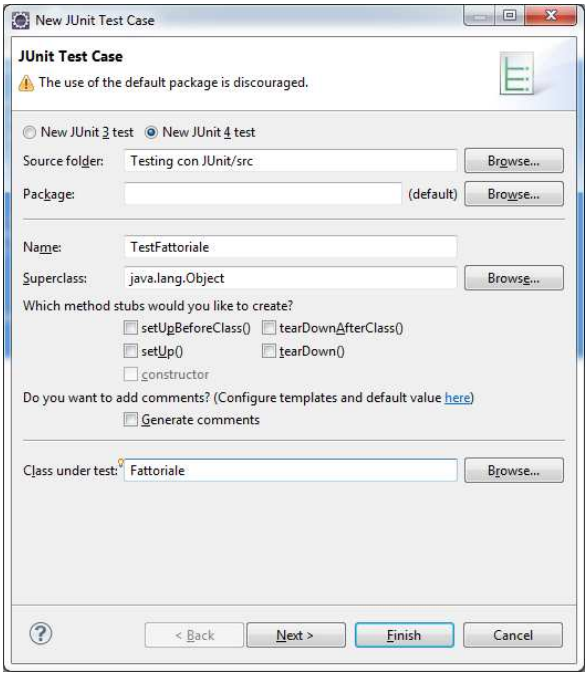
- nell'ambito di questo progetto possiamo poi definire la classe che vogliamo verificare – ad esempio la classe **Fattoriale**

```
public class Fattoriale {

   /* Calcola il fattoriale di n. */
   public static int fattoriale(int n) {
        // pre: n>=0
        int fatt;   // il fattoriale di n
        int i;      // per iterare tra 1 e n

        /* calcola il fattoriale di n */
        fatt = 1;
        for (i=1; i<n; i++) {
                fatt *= i;
        }
        return fatt;
   }

}
```


JUnit, in pratica

A questo punto possiamo creare una classe di test per la classe **Fattoriale**

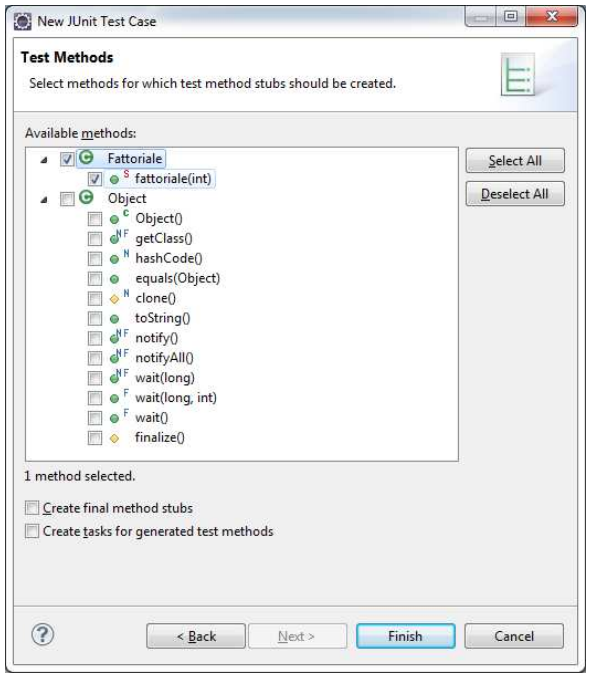
- da Eclipse, selezioniamo New -> JUnit Test Case
- come nome ("name") del test possiamo usare **TestFattoriale**
- come classe sotto test ("Class under test") indichiamo **Fattoriale**
- è importante che in alto sia selezionata la voce "New JUnit 4 test" – tutte le altre caselle possono rimanere non selezionate
- poi premiamo Finish



JUnit, in pratica

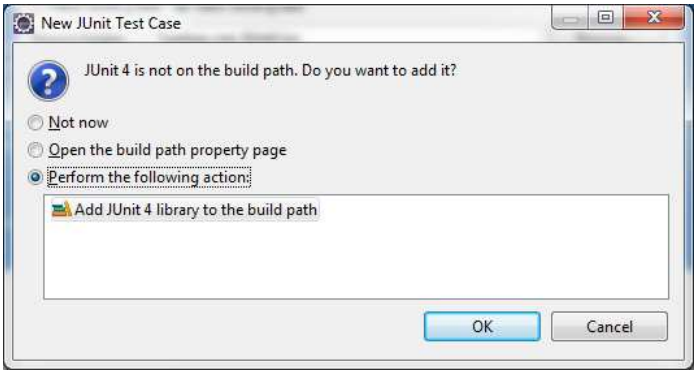
Con le versioni precedenti di Eclipse, può essere però utile premere Next (anziché Finish), e selezionare il metodo che si vuole testare

- nel nostro esempio, il metodo **fattoriale** della classe **Fattoriale**
- poi premiamo Finish



JUnit, in pratica

La prima volta che eseguiamo questa operazione in un progetto ci viene proposta la seguente schermata



- rispondiamo OK

JUnit, in pratica

Questo è il codice generato automaticamente da Eclipse

- è la classe di test **TestFattoriale**
- che contiene un solo metodo di test di nome **test**

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

import org.junit.Test;

public class TestFattoriale {

    @Test
    public void test() {
        fail("Not yet implemented");
    }

}
```

- **@Test** è un'annotazione che indica a JUnit che quello che segue è, appunto, un metodo di test
- **fail(...)** è invece un'asserzione – che fallisce sempre

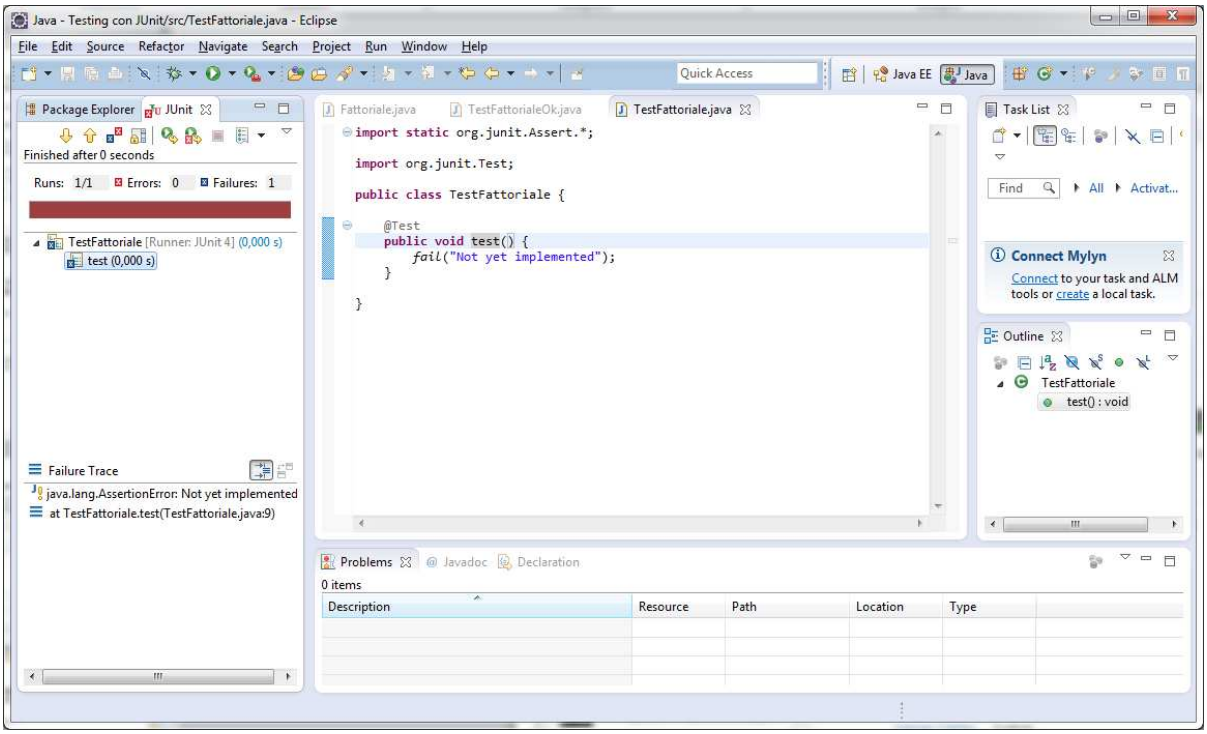
JUnit, in pratica

A questo punto è già possibile eseguire il test – anche se già sappiamo che fallirà

- con il tasto DESTRO del mouse – non il sinistro – clicchiamo sul nome del file per la classe di test – **TestFattoriale.java**
- selezioniamo la voce Run As -> JUnit Test

JUnit, in pratica

Ecco il risultato dell'esecuzione del test – barra rossa



JUnit, in pratica

Ora possiamo modificare il test andando a prendere in considerazione la prima asserzione che avevamo scelto

- il fattoriale di 0 vale 1

- modifichiamo il metodo di test **test**
- per prima cosa lo chiamiamo **testFattorialeZero**
- dobbiamo usare l'asserzione **assertEquals(*valoreAtteso*, *valoreEffettivo*)**
  - questa asserzione ha successo se le espressioni ***valoreAtteso*** e ***valoreEffettivo*** hanno lo stesso valore – e fallisce altrimenti
- le espressioni di cui vogliamo confrontare i valori sono
  - **1** – il valore atteso di **fattoriale(0)**
  - **Fattoriale.fattoriale(0)** – il risultato fornito dall'effettiva invocazione del metodo

JUnit, in pratica

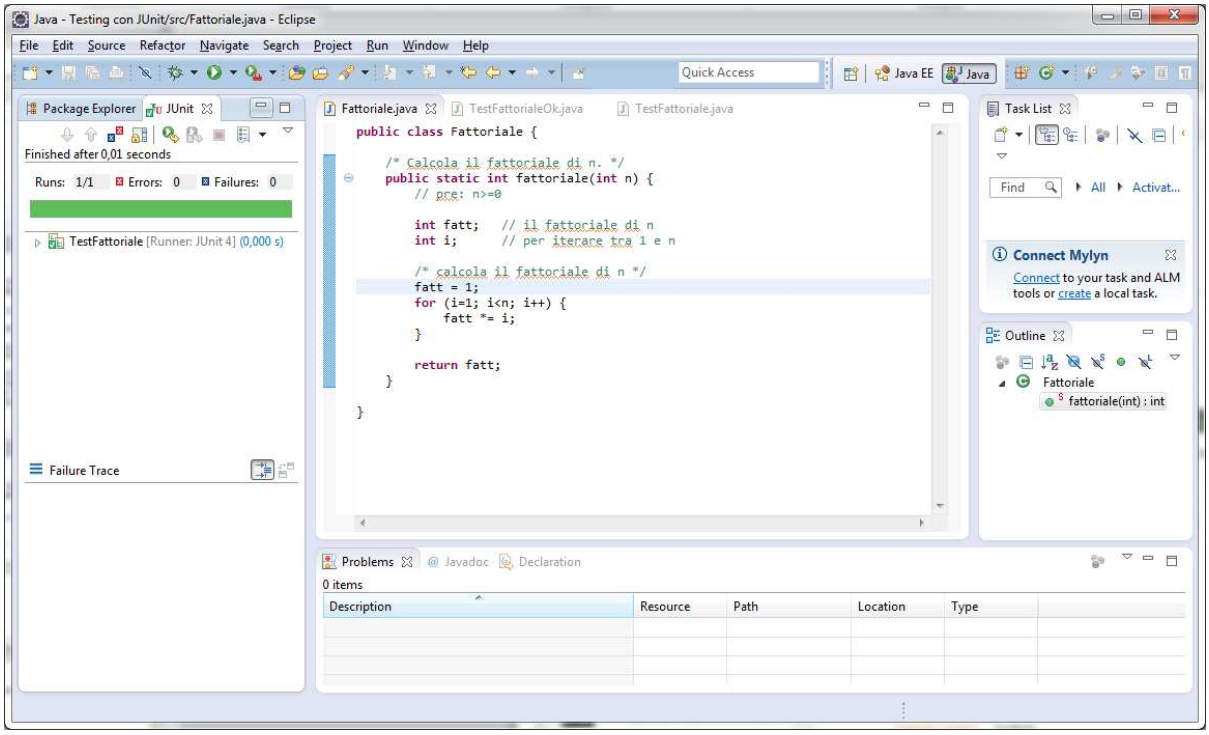
Ora possiamo modificare il test andando a prendere in considerazione la prima asserzione che avevamo scelto

- il fattoriale di 0 vale 1

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

public class TestFattoriale {

    @Test
    public void testFattorialeZero() {
        assertEquals(1, Fattoriale.fattoriale(0));
    }

}
```


JUnit, in pratica

Ecco il risultato dell'esecuzione del test – barra verde



JUnit, in pratica

Possiamo modificare il test aggiungendo la seconda asserzione scelta (copia-incolla-modifica il test già definito)

- il fattoriale di 1 vale 1

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

public class TestFattoriale {

    @Test
    public void testFattorialeZero() {
        assertEquals(1, Fattoriale.fattoriale(0));
    }
    @Test
    public void testFattorialeUno() {
        assertEquals(1, Fattoriale.fattoriale(1));
    }

}
```

- anche in questo caso il test ha successo – barra verde

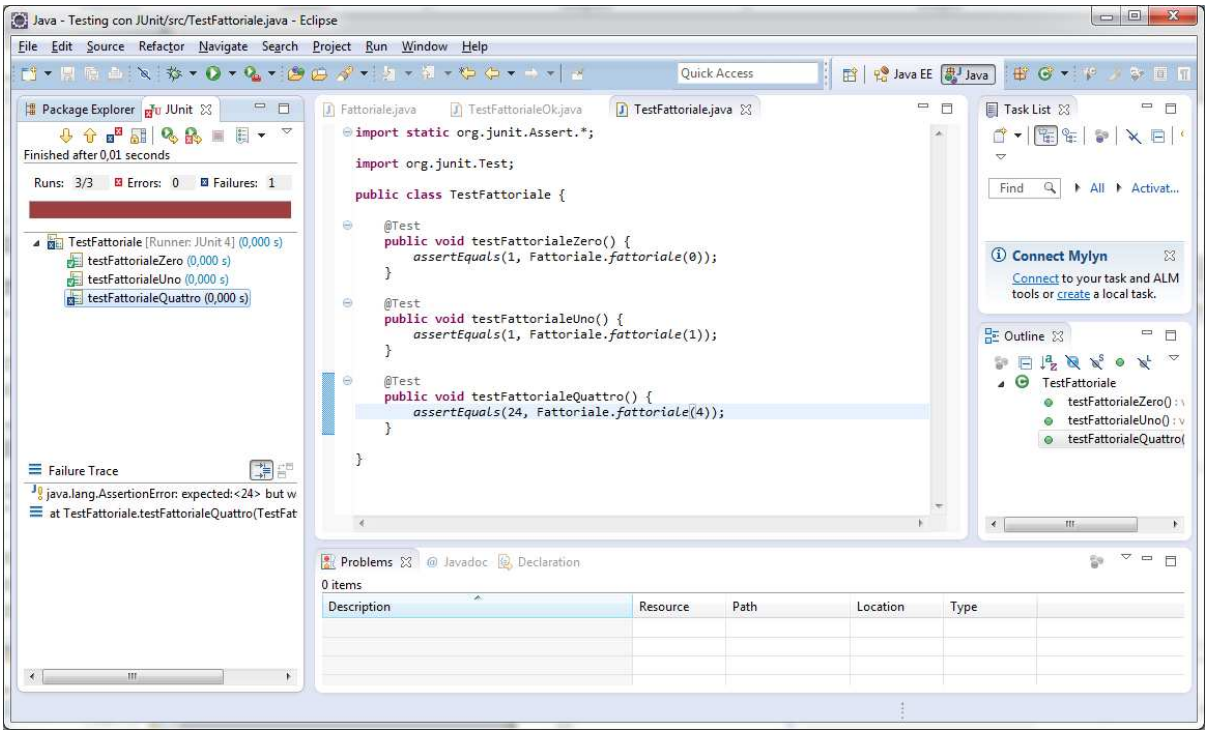
JUnit, in pratica

Infine, aggiungiamo anche il metodo di test per la terza asserzione – il fattoriale di 4 vale 24

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

public class TestFattoriale {

    @Test
    public void testFattorialeZero() {
        assertEquals(1, Fattoriale.fattoriale(0));
    }
    @Test
    public void testFattorialeUno() {
        assertEquals(1, Fattoriale.fattoriale(1));
    }
    @Test
    public void testFattorialeQuattro() {
        assertEquals(24, Fattoriale.fattoriale(4));
    }

}
```

JUnit, in pratica

In questo caso il test fallisce – barra rossa



JUnit, in pratica

L'interfaccia di JUnit riporta delle indicazioni utili sul fallimento del test

- l'esecuzione di **testFattorialeZero** e **testFattorialeUno** è avvenuta con successo (spunta verde)
- tuttavia, l'esecuzione di **testFattorialeZero** è fallita (crocetta blu)
- la segnalazione del fallimento è
  - *java.lang.AssertionError: expected:<24> but was:<6>*
- poiché il valore calcolato 6 è uguale a 3! anziché a 4!, potremmo aver commesso un errore di uno

JUnit, in pratica

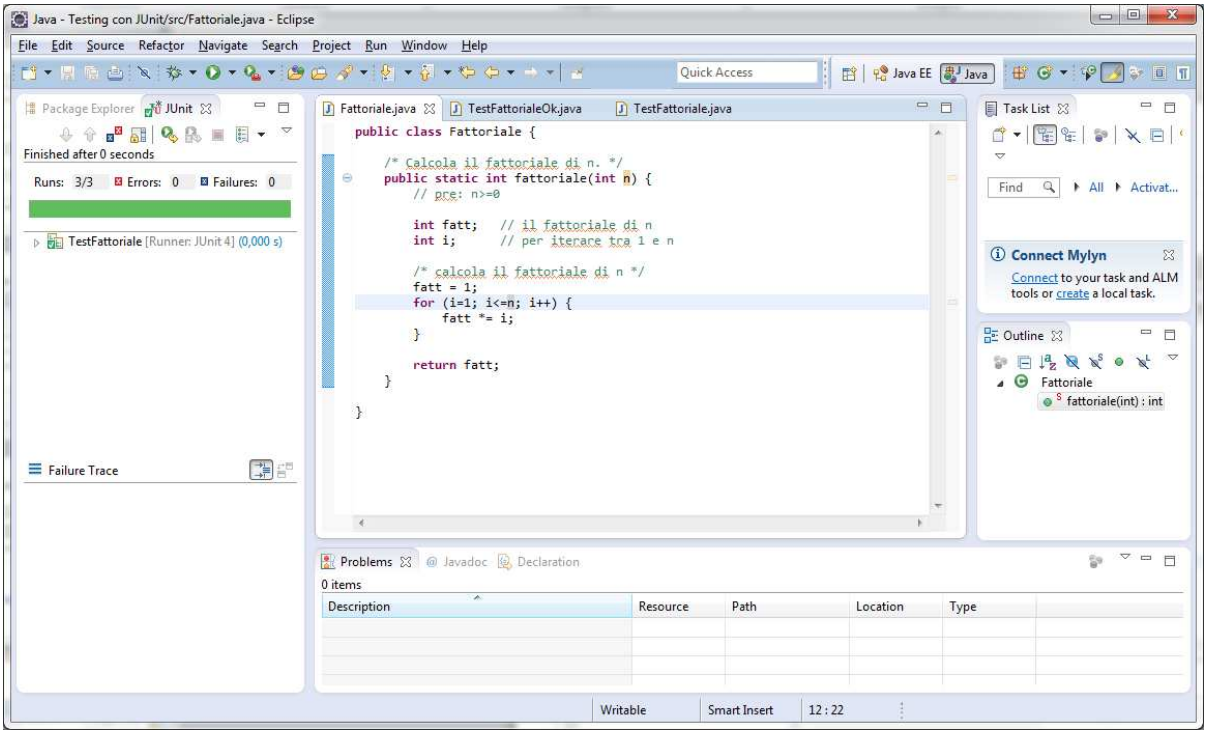
In effetti, nel metodo **fattoriale** abbiamo commesso un errore di uno

```
/* Calcola il fattoriale di n. */
public static int fattoriale(int n) {
     // pre: n>=0
     int fatt;   // il fattoriale di n
     int i;      // per iterare tra 1 e n

     /* calcola il fattoriale di n */
     fatt = 1;
     for (i=1; i<n; i++) {
            fatt *= i;
     }
     return fatt;
}
```

- possiamo correggere l'errore e ripetere l'esecuzione del test

JUnit, in pratica

Barra verde – ora il test ha successo



Riassumendo

Riassumendo

- ogni volta che definiamo una classe C è bene definire una classe di test per la classe C
  - in alcuni casi è utile definire una classe di test per ciascun metodo M di C
- per testare un metodo M
  - scegliamo i dataset per il metodo M
  - per ciascun dataset D per M definiamo un metodo di test nella classe di test
- per realizzare un metodo di test
  - utilizziamo un'asserzione che confronta il comportamento atteso del metodo con il comportamento effettivo del metodo
  - il risultato atteso di un metodo M va indicato con un letterale, il risultato effettivo dell'esecuzione di M va indicato con un'invocazione del metodo

Asserzioni

Ciascuna asserzione è un'affermazione che è vera (se non ci sono errori) – ma che invece risulta falsa in caso di errori nel nostro metodo o nella nostra classe

- se tutte le asserzioni (in tutti i metodi di test) sono vere
  - il test ha successo – barra verde
- se almeno un'asserzione (in un metodo di test) è falsa
  - il test fallisce – barra rossa
- di solito va scritta una (e una sola) asserzione per ciascun metodo di test

Asserzioni

JUnit offre diversi tipi di asserzioni

- la più comune è **assertEquals(valoreAtteso, valoreEffettivo)**
  - ha successo se **valoreAtteso** e **valoreEffettivo** sono uguali
  - JUnit usa **==** per confrontare valori primitivi e **equals** per confrontare oggetti
- in alcuni casi l'asserzione è una condizione che non è relativa ad un'uguaglianza – si possono usare le seguenti asserzioni
  - **assertTrue(condizione)** – ha successo se la condizione **condizione** è vera – e fallisce altrimenti
  - **assertFalse(condizione)** – ha successo se la condizione **condizione** è falsa – e fallisce altrimenti
- un'altra asserzione utile è **assertArrayEquals(arrayAtteso, arrayEffettivo)**
  - ha successo se gli array **arrayAtteso** e **arrayEffettivo** sono uguali

Esecuzione di una classe di test

Come viene eseguito un test case (una classe di test)?

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

public class TestFattoriale {
    @Test
    public void testFattorialeZero() {
        assertEquals(1, Fattoriale.fattoriale(0));
    }
    @Test
    public void testFattorialeUno() {
        assertEquals(1, Fattoriale.fattoriale(1));
    }
}
```

Esecuzione di una classe di test

Come viene eseguito un test case (una classe di test)?

- intuitivamente, JUnit esegue ordinatamente tutti i metodi di test – ovvero quelli che hanno l'annotazione **@Test**
- ogni asserzione è in effetti un'invocazione di metodo – la cui esecuzione richiede che prima vengano valutati i suoi parametri – e dunque calcolato il risultato atteso ed eseguito il metodo sotto test e calcolato il risultato effettivo
- JUnit tiene traccia dell'esito del test – che viene aggiornato durante l'esecuzione dei diversi metodi di test – e in particolare quando ne viene valutata l'asserzione
- dopo che sono stati eseguiti tutti i metodi di test, allora JUnit mostra un rapporto sull'esito del test
  - sintetico, mediante la barra verde o rossa
  - dettagliato, sotto la barra, dicendo quali metodi di test hanno avuto successo e quali sono falliti
  - per i metodi falliti, viene mostrata una descrizione del fallimento – ad es., *il risultato atteso era 24 ma è stato 6*