Ingegneria Del Software

Università di Roma Tor Vergata

JUnit

Open-source Testing Framework

Ivan Bruno

Roadmap

JUnit basics

JUnit cooding

Ingegneria del Software

Disciplina il cui obiettivo è lo sviluppo di sistemi software di alta qualità senza sprechi.

Standardizza alcune fasi della creazione del software identificandone un **ciclo di vita**.

1. Specifiche del Software
2. Progettazione
3. Implementazione
4. **Convalida**
5. Evoluzione

Convalida del Software

Generalmente indicata con l'acronimo **V&V** (Verification and Validation)

Obiettivi:

Mostrare che il sistema è conforme alle sue specifiche

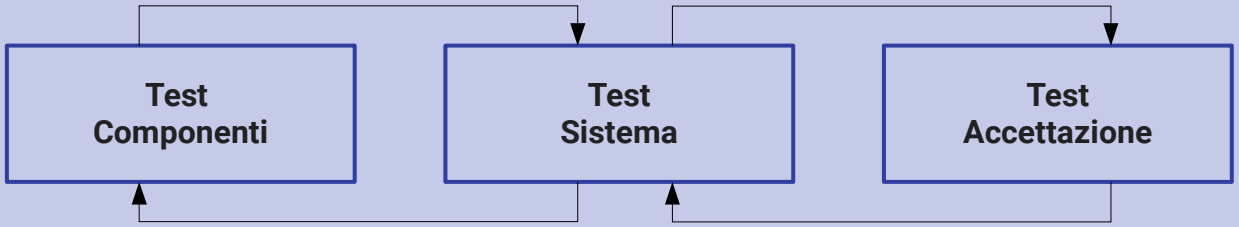
Soddisfa le aspettative del cliente

Testing

Attività che stabilisce l'esistenza di difetti

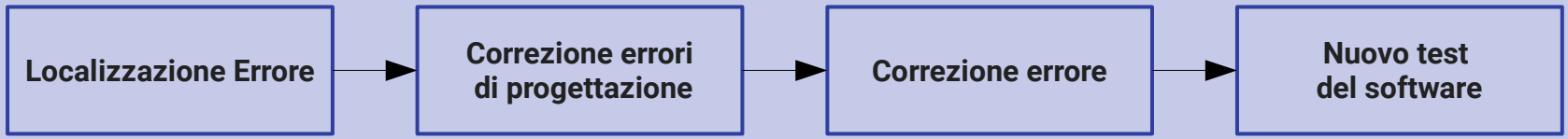
Gli stadi del processo di test sono 3:

1 – Test dei componenti o di **unità**
2 – Test di sistema o di **integrazione**
3 – Test di Accettazione o **collaudo**



Debugging

Attività che localizza il difetto e cerca di correggerlo



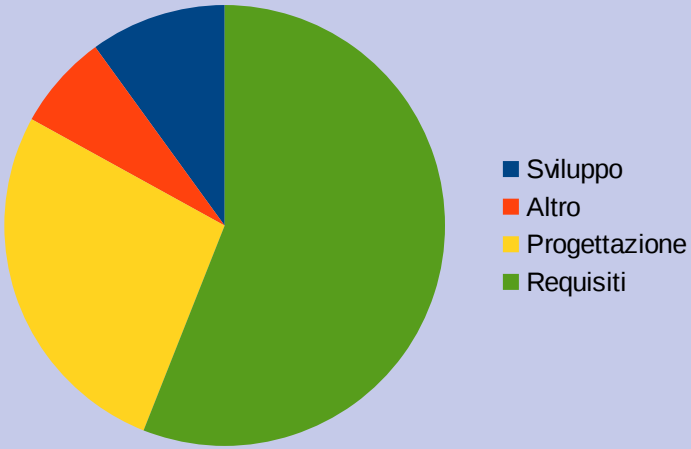
Generare ipotesi sul comportamento visibile del programma

Verificare tali ipotesi nella speranza di trovare il difetto riscontrato tramite:

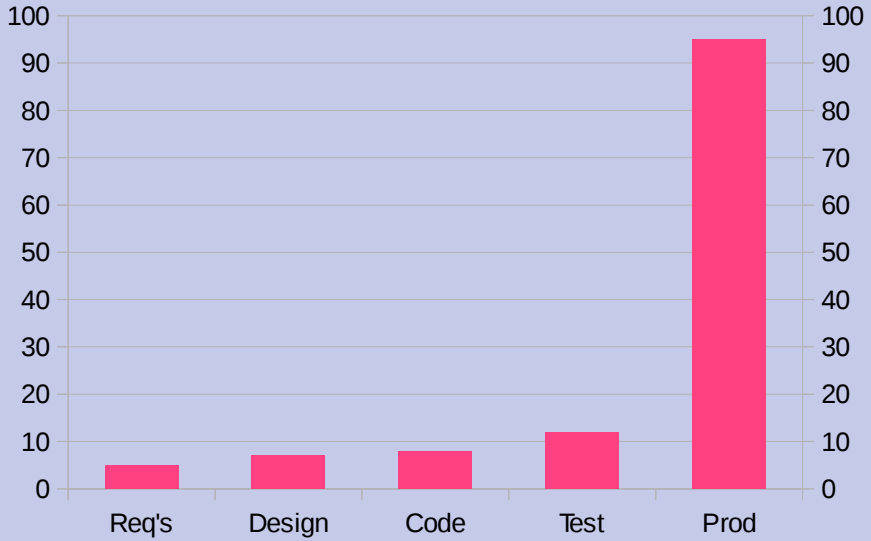
- **Tracciamento manuale** del codice
- Stesura di appositi **test case** per localizzare il problema

Il processo di debug può essere aiutato da **strumenti interattivi** che mostrano i valori intermedi delle variabili e tracciano le istruzioni eseguite

Importanza del Testing



Effort del testing



Costo relativo al fix di un difetto

JUnit Framework

Junit è un framework per la stesura di test in maniera ripetibile

Sviluppato da **Erich Gamma** e **Kent Beck** a supporto delle metodologie agili

Promuove la cosiddetta **TDD** Test Driven Development, una metodologia di sviluppo nella quale il testing ha un ruolo fondamentale infatti ha come idea di fondo:

"First testing then coding"

Caratteristiche principali:

- Fornisce annotazioni per identificare i metodi di test
- Fornisce asserzioni per testare i risultati attesi
- Fornisce apposite classi per eseguire i test
- Fornisce un feedback immediato dei test
- Test possono essere eseguiti in modo automatico
- Test possono essere organizzati in Suite contenenti a loro volta altre Suite e test

Unit Test

Un test di unità è una parte di codice atta ad eseguirne un'altra al fine di verificare il funzionamento di quest'ultima

Formalmente un test di unità è caratterizzato da 2 elementi:

1 - un valore in input noto
2 - un valore di output atteso

Per ogni **funzionalità/requisito** da testare:

1- Un test il cui esito atteso è **positivo**
2- Un test il cui esito atteso è **negativo**

Università di Roma
Tor Vergata

JUnit cooding

Java Setup

Scaricare ed installare Java Development Kit dal seguente URL:

http://www.oracle.com/technetwork/java/javase/downloads/

Dopo aver effettuato l'installazione, bisogna controllare che sia **Java** sia il **Java Compiler** siano stati aggiunti correttamente alle **variabili d'ambiente** relative al sistema operativo che si sta utilizzando.

In caso non lo siano, bisogna aggiungerli manualmente.

Per verificare se Java è stato riconosciuto correttamente, basta aprire una shell e digitare il seguente comando:

java ***-version***

```
[ivan18@localhost ~]$ java -version
java version "1.8.0_77"
Java(TM) SE Runtime Environment (build 1.8.0_77-b03)
Java HotSpot(TM) 64-Bit Server VM (build 25.77-b03, mixed mode)
```

Per verificare se il Java Compiler è stato riconosciuto correttamente, basta aprire una shell e digitare il seguente comando:

javac **-version**

```
[ivan18@localhost ~]$ javac -version
javac 1.8.0_77
```

Junit download

Prima di iniziare abbiamo bisogno di scaricare i package necessari per il funzionamento di Junit dal seguente link:

https://github.com/junit-team/junit4/wiki/Download-and-Install

Abbiamo bisogno di 2 tipi di package:

- **junit.jar** framework JUnit
- **hamcrest-core.jar** framework di supporto per la scrittura di test in linguaggio Java

Eclipse Java IDE

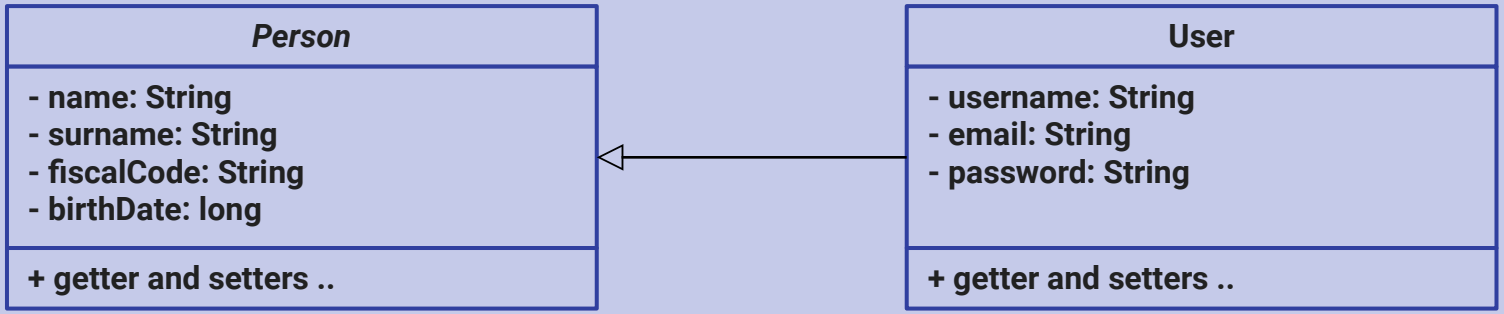
Ricordate che il codice relativo ad un qualsiasi linguaggio di programmazione può essere scritto tramite l'utilizzo di un qualsiasi **editor di testo**

Una volta ottenuti i file con il codice sorgente, questi vanno **compilati** ed **eseguiti** manualmente tramite shell

Per facilitare e velocizzare la scrittura di software Java utilizzeremo **Eclipse**, un apposito **IDE** (Integrated development environment) scaricabile dal seguente link:

https://www.eclipse.org/ide/

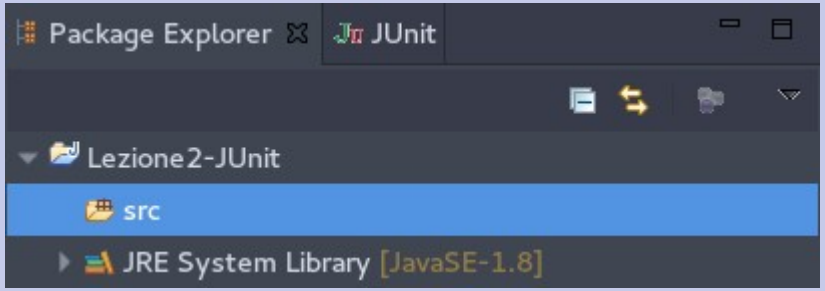
Caso di studio



Apriamo eclipse e procediamo con la creazione di un nuovo progetto java:

Clicchiamo sul menù **file -> new -> Java Project**

Inseriamo il nome del progetto: **Lezione2-JUnit**



Nell'area **PackageExplorer** possiamo visualizzare quanto ottenuto

Prima di procedere con l'implementazione del caso di studio è utile attuare piccole accortezze sullo **scaffolding** dell'applicazione

Andiamo ad eliminare il package principale **src** dal **Build Path** di eclipse:

(TastoDx del mouse su)**src -> Build Path -> Remove from Build Path**

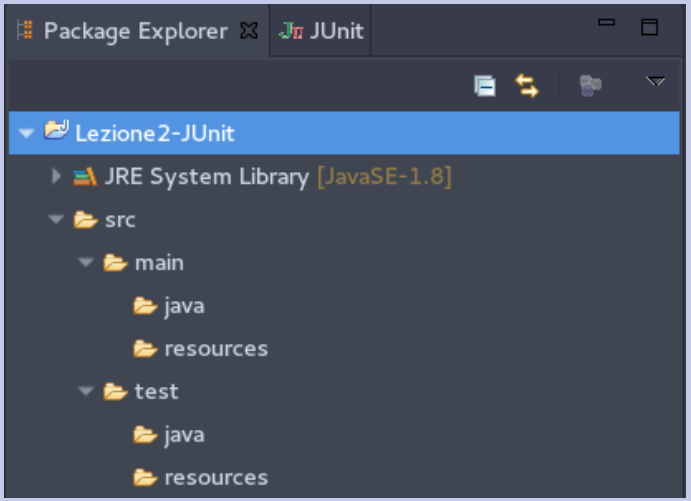
Nell'area **PackageExplorer** notiamo graficamente la corretta rimozione del package dal Build Path

Ricordando che è buona norma mantenere **diviso** il codice sorgente dal codice di test, procediamo alla creazione di 2 sottopackage del package principale (src):

1 – **main** in cui andremo ad inserire i sorgenti del progetto
2 – **test** in cui andremo ad inserire i nostri test

Per convenzione creiamo nei package appena descritti 2 ulteriori sottopackage:

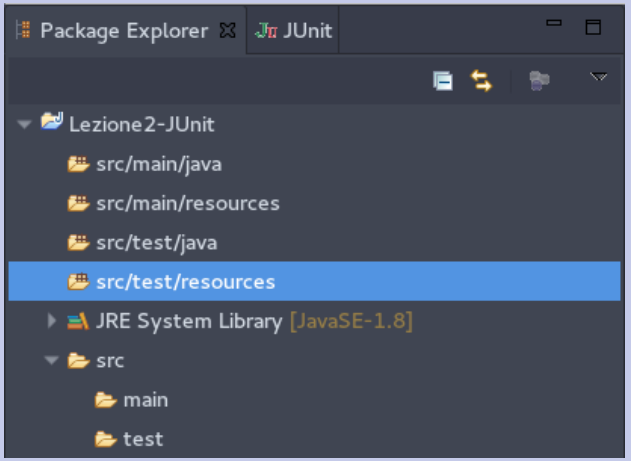
1 – **java** conterrà il codice prodotto
2 – **resources** conterrà tutti gli eventuali assets per il progetto



A questo punto possiamo **inserire** i nuovi package nel Build Path come Source Folder

Su ogni package clicchiamo:
tastoDx > Build Path -> Use as source folder

Dopo le operazioni precedenti dovremmo aver ottenuto un risultato simile al seguente:



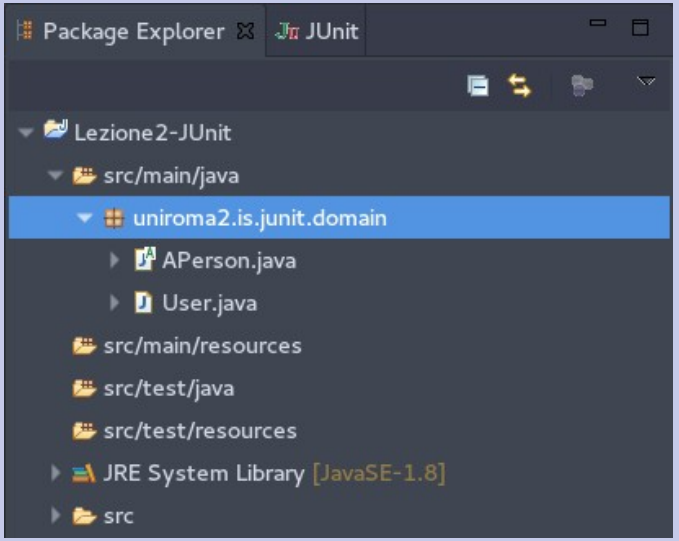
Implementazione caso di studio

Procediamo con l'implementazione del diagramma delle classi mostrato in precedenza

Creiamo ne package src/main/java un nuovo package che andrà a contenere le classi di dominio individuate nel caso di studio:

1 – APerson (classe astratta)

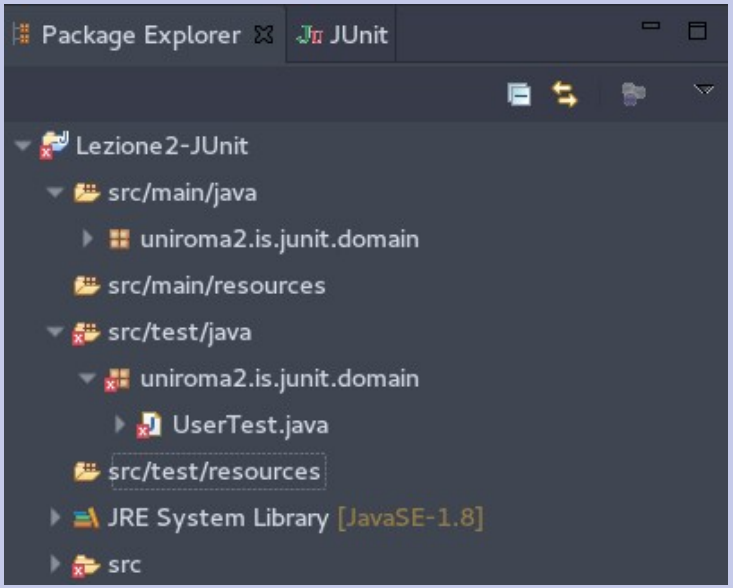
2 – User (classe concreta)



Primo caso di test

Dopo aver implementato le classi individuate nel caso di studio, procediamo con la definizione di alcuni semplici test

Andiamo quindi a creare un **unit test** atto ad introdurre il framework JUnit



```java
package uniroma2.is.junit.domain;

public class UserTest {

    @Test
    public void checkUsername() {
        User u = new User("ivan", "bruno", "1234567891234567",123123 ,"ivan18", "ivan18@gmail.com", "ok");
        String username = u.getUsername();
        System.out.println(username);
    }

}
```

A questo punto, eclipse lancerà un errore in quanto non riconoscerà l'annotazione @Test

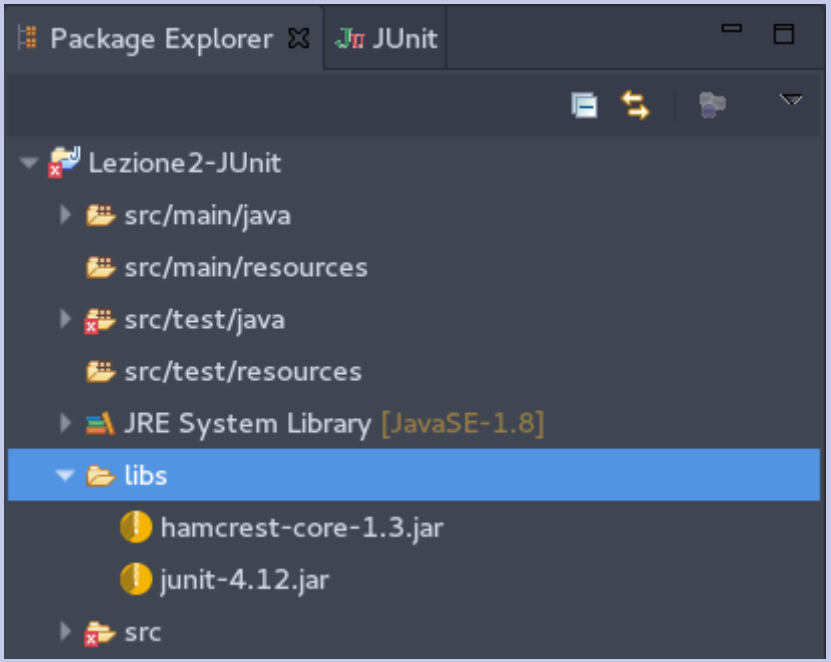
Questo perchè al progetto non sono state aggiunte le librerie necessarie al corretto funzionamento di Junit

Andiamo quindi ad inserire le librerie Junit (scaricate in precedenza)

Aggiunta librerie JUnit

Come prima cosa, creiamo una directory per contenere le nostre librerie

Al suo interno, inseriamo le librerie necessarie per il corretto funzionamento di Junit

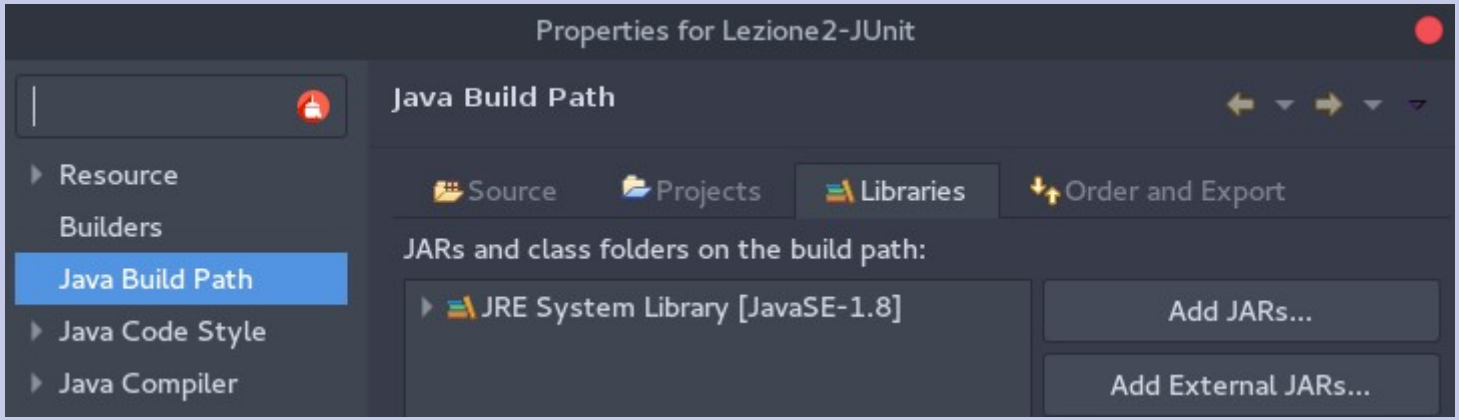


Collegamento librerie

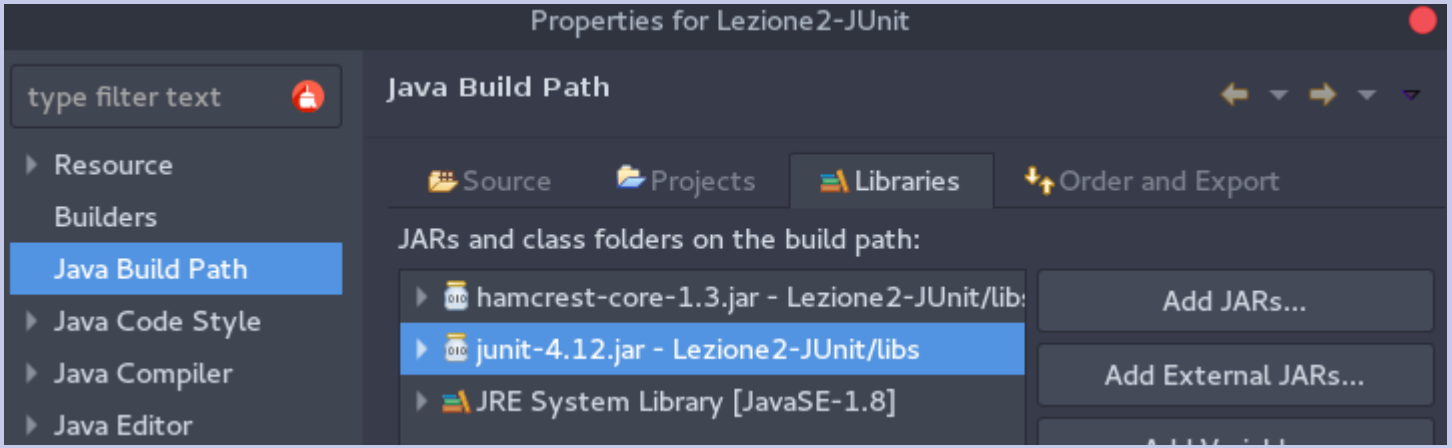
Per collegare le librerie al progetto occorre:

1 – Accediamo alle proprietà del progetto tramite il Package Explorer

2 – Spostiamoci nel menù **Java Build Path** e selezioniamo la sezione **Libraries**



Clicchiamo su **Add JARs..** e dopo aver selezionato le nostre librerie confermiamo.



A questo punto, i precedenti errori di compilazione dovrebbero essere risolti

Asserzioni

JUnit fornisce dei metodi per effettuare asserzioni su tutti i tipi di dato **primitivi** e su **array**

Questo tipo di metodi si aspettano in ingresso 2 parametri:

1 – il **valore atteso**
2 – il **valore effettivo** da valutare

Opzionalmente è possibile inserire come primo valore una stringa da visualizzare in caso di fallimento del test.

Per approfondimenti consultare:

https://github.com/junit-team/junit4/wiki/Assertions

Esempi

Runners

Per eseguire il test è necessario utilizzare un'apposita classe

Gli strumenti che JUnit mette a disposizione per eseguire i test vengono detti **Runners**

Tipologie

1 – Console based Runner

2 – Code Runners

3 – Third Party Runners

Per approfondimenti consultare:

https://github.com/junit-team/junit4/wiki/Test-runners

Esempi

Università di Roma
Tor Vergata

Suites

Alcune volte, vi è l'esigenza di raggruppare un insieme di test per eseguirli in blocco

A tal proposito Junit fornisce un opportuno Runner tramite la classe **Suite**

Il Suite runner **ingloba** test di altre classi e li esegue rispettandone tutte le caratteristiche

Per approfondimenti consultare:
http://junit.org/javadoc/latest/org/junit/runners/Suite.html

Esempi

Ordering

Di default Junit **non** specifica l'ordine con il quale eseguirà di test

Per cambiare l'ordine di esecuzione dei test si utilizza l'annotazione **@FixMethodOrder**

Bisogna inoltre specificare il metodo di sorting che si vuole utilizzare:

1 – MethodSorters.**DEFAULT**

2 – MethodSorters.**JVM**

3 – MethodSorters.**NAME_ASCENDING**

Esempi

Università di Roma
Tor Vergata

Timeout

E' possibile inoltre specificare un tempo di esecuzione **massimo** per un test

Allo scadere di questo lasso di tempo il test verrà considerato **fallito**

JUnit mette a disposizione il **parametro** Timeout per l'annotazione @Test

Tale parametro specifica il tempo in **millisecondi** oltre il quale il test andrà fallito

Esempio:

```
@Test(timeout=1000)       //Tempo di timeout 1 secondo
Public void methodToTest(){ }
```

Esempi

Fixtures

Hanno lo scopo di preparare l'ambiente per il caso di test da eseguire

Differenza tra per risorse **costose** (vere risorse) e risorse **leggere**

Esempi di risorse poco costose: **mock** e **stub**

Codice sviluppato senza alcune parti

- Stub/Mock
- Software completo (accortezza a disponibilità ed integrità delle risorse)

Scelta risorse: cordinate con il repository

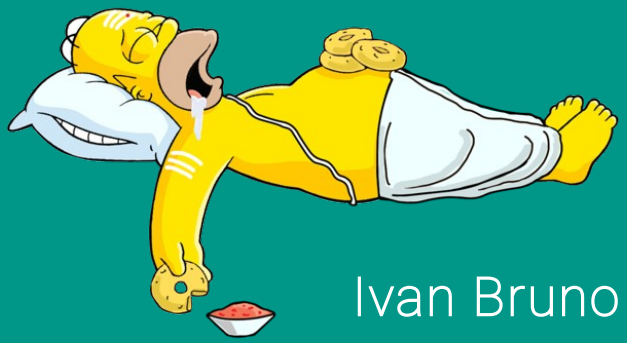
Per approfondimenti consultare:

https://github.com/junit-team/junit4/wiki/Test-fixtures

Università di Roma
Tor Vergata

Esempi

Grazie per l'attenzione



Ivan Bruno