ITS INFORMATION AND COMMUNICATIONS TECHNOLOGY Academy

NOME MODULO: ***JAVA***

UNITÀ DIDATTICA: ***JAVA.3 L5***

DOCENTE: Giuseppe Scalise

Biennio 2023-2025

Argomenti della lezione

- Unit test

Introduzione agli Unit Test

Quando si realizza un software, qualsiasi esso sia, la fase di testing svolge un ruolo fondamentale.

Effettuare i test su un software, consente di intercettare eventuali malfunzionamenti prima del suo utilizzo in ambienti di produzione.

Molto spesso la fase di test è trascurata perché:

- Non c'è abbastanza tempo per effettuare i test
- Non esiste una figura o un ruolo destinato a fare i test

Quando si scrivono migliaia di righe di codice è normale che ci siano degli errori.

Introduzione agli Unit Test

Uno dei modi per assicurarsi che il software sviluppato funzioni correttamente è quello di eseguire dei test **su ogni singolo componente** che compone il software.

I test si possono fare con:

- **Test manuali** – si crea un main che utilizza la classe da testare ed invoca tutti i metodi per verificarne la bontà. Ha un grosso limite di non poter essere ripetuta nel tempo se il software cambia (non si possono fare dei regression test)
- **Utilizzo di strumenti** – consiste nell'utilizzo di software o librerie che supportano lo sviluppatore nella definizione degli unit test (ad esempio Junit che è un framework per sviluppare unit test in Java)

Unit Test

Cos'è uno unit test ?

E' una metodologia che consente di verificare il corretto funzionamento di singole unità di codice sulla base di determinate condizioni.

In Java l'unità di codice è un metodo o un gruppo di metodi che implementano una singola funzionalità.

Esempio:

Supponiamo di aver implementato un metodo che verifica se una stringa è lunga almeno 8 caratteri.

Uno unit test consiste nel verificare che il metodo restituisca **false** se passiamo una stringa con meno di 8 caratteri.

Unit Test

Lo unit test di un software può essere organizzato in:

- **Test Case**: questo test consente di verificare una singola unità di codice
- **Test Suite**: è un gruppo di Test Case che consente di verificare diverse funzionalità correlate fra di loro.

Unit Test

Vantaggio nell'utilizzo degli unit test

- **Rendere più semplici le modifiche** – usando gli unit test è più semplice modificare il codice avendo la tranquillità che il componente modificato continuerà a funzionare correttamente. Se scrivete un test case per ogni metodo, se una modifica al codice produce un fallimento del test, è facile individuare la modifica responsabile del fallimento del test.
- **Rende più semplice l'integrazione tra i componenti** – gli unit test limitano i bug legati all'iterazione tra i componenti, semplificando i test di integrazione.
- **E' un supporto alla documentazione del software** – uno unit test, in fin dei conti, è un esempio di utilizzo delle funzionalità di un componente; questo rende uno unit test un prezioso supporto alla stesura della documentazione tecnica.

Unit Test

Limiti degli unit test

- Gli unit test non sono sempre in grado di identificare tutti gli errori in un software poiché, analizzando i singoli metodi, non è possibile rilevare gli errori legati all'integrazione tra i componenti (anche se sono di supporto ai test di integrazione). Non valutano ad esempio le performance.
- Gli unit test sono più efficaci quando vengono utilizzati insieme ad altre tecniche di testing del software.

Introduzione a JUnit

- Cos'è JUnit e come utilizzarlo per la creazione degli unit test.

Cos’è JUnit

- JUnit è un **framework open-source che consente di scrivere unit test**, supportando chi sviluppa software Java nell’attività di testing.
- JUnit consente di **scrivere ed eseguire automaticamente Test Case e Test Suite.**
- JUnit consente di scrivere **test di oggetti e classi Java.**
- JUnit **si integra con Eclipse** (ormai i jar che contengono API JUnit sono già incluse in Eclipse, quindi non è necessario importare nessuna libreria).

JUnit – Test Case e Test Suite

- Per ogni classe da testare occorre creare una classe **Test Case** al cui interno inserire i test da eseguire. Se abbiamo più Test Case correlati, dobbiamo creare una classe **Test Suite** che conterrà il riferimento alle classi Test Case.
- Generalmente le classi di test vanno create in una directory diversa da quella in cui è presente il codice (il codice nella directory **src**, le classi di testing nella directory **test**)
- Il nome della classe Test Case generalmente appartiene allo stesso package della classe da testare ed ha lo stesso nome seguito da Test.

JUnit – Test Case

Le annotazioni disponibili in JUnit da utilizzare per i metodi definiti nella classe Test Case sono:

@Test – indica che il metodo è un test

@Before – indica che il metodo deve essere eseguito prima di tutti i test

@After - indica che il metodo deve essere eseguito dopo di tutti i test

@BeforeClass - indica che il metodo deve essere eseguito prima di eseguire il primo test

@AfterClass - indica che il metodo deve essere eseguito dopo l'esecuzione dell'ultimo test

NOTA: la classe Test Case non deve definire un costruttore e tutti i metodi con annotation devono essere di tipo void, senza parametri in ingresso e devono effettuare il trows della classe Exception.

JUnit – Test Case

Quando effettuiamo un test, il risultato ottenuto dall'invocazione del metodo da testare, deve essere confrontato con quello atteso.

Se il confronto ha esito positivo, il test avrà successo, altrimenti il test fallisce.

Tale confronto si effettua usando i metodi statici della **classe Assert di JUnit**.

La classe Assert mette a disposizione dei metodi statici che possono essere utilizzati nel Test Case utilizzando il seguente metodo statico:

Import static org.junit.Assert.*;

JUnit – Test Case

Esempio

```
public class Calcolatrice {
public static void main(String[] args) {
Calcolatrice c = new Calcolatrice();
System.out.println("La somma è " + c.somma(3, 2));
}
public int somma(int a, int b) {
return a + b;
}
}

import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.Test;

class CalcolatriceTest {      //Classe che rappresenta il Test Case della Calcolatrice
@Test
void testSomma() {
Calcolatrice c=new Calcolatrice(); //Il test testSomma utilizza il metodo statico assertEquals
assertEquals(2, c.somma(2, 1) );   // per verificare che il metodo somma restituisca la somma
 }
}
```

JUnit – Test Case

Esiti del Test Case

L'esecuzione del test genera un risultato che può essere:

- **Successo** (colore verde): il risultato atteso coincide con il risultato ottenuto dal metodo testato. Il metodo **funzione** correttamente
- **Fallito** (colore rosso): il risultato non coincide con il risultato atteso dal metodo testato. Il metodo **non funzione** correttamente
- **Errore** (colore blu): il test case è andato in errore. Non sappiamo se il metodo funziona. Le cause dell'errore possono essere:
  - è stata generata un'eccezione durante l'esecuzione del test case (es. utilizzo di una variabile null …)
  - il test case non è configurato correttamente.

JUnit – Test Case

Esercizio1 – JUnitTestSomma.

Per inserire un Test Case :

New -> Other -> JUnit -> Junit Test Case -> Next ->

Inserire un nome per la classe di test e il nome della classe da testare -> Next

Selezionare i metodi da testare -> Finish

JUnit – Test Case

Esercizio 2 JUnit Calcolatrice
Costruire una classe calcolatrice con 4 metodi e eseguire i test con JUnit sui metodi

Esercizio 3 JUnit Stringhe
Costruire una classe con due metodi. Un metodo che data una frase e un carattere ritorna quanti dei caratteri dati ci sono all'interno della frase. Un metodo che riconosce se una parola è palindroma. Eseguire il test con JUnit.

JUnit – Test Suite

La Test Suite è una classe che **consente di eseguire un insieme di Test Case**.

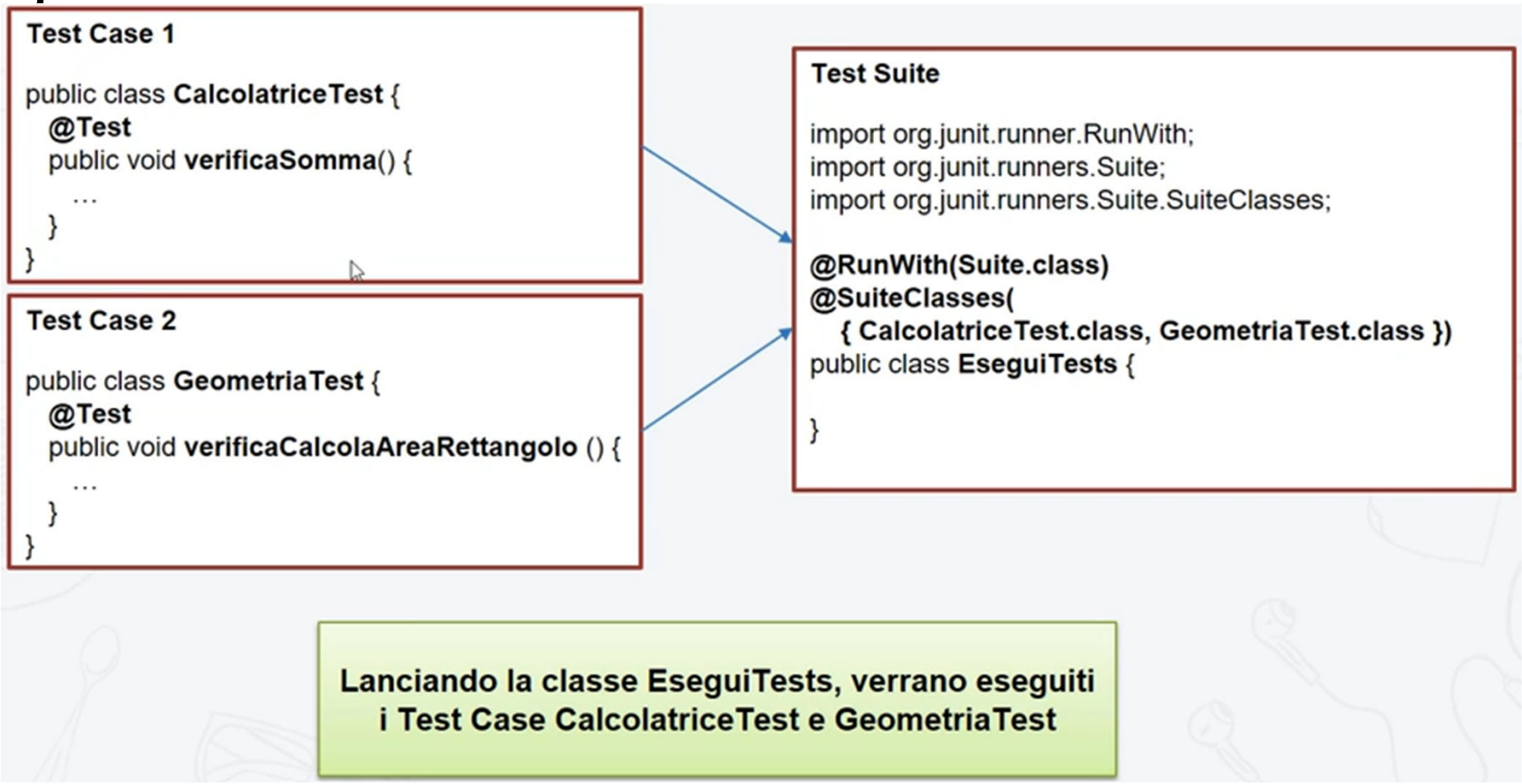
Lanciando la classe Test Suite, tutti i test associati ad essa verranno eseguiti, senza necessità di eseguirli uno per volta.

La classe Test Suite può avere qualsiasi nome, purchè (per naming convention) termini con Test.

Le Test Suite sono importantissime quando si lavora con strumenti di Continuous Integration (ad es. Jenkins, CircleCI …).

JUnit – Test Suite

Esempio



JUnit – Test Suite

Esercizio 4 – Esempio Test Suite.

JUnit – La classe Assert

La classe org.junit.Assert, contiene una serie di metodi statici che possiamo utilizzare per determinare se il metodo da testare funziona correttamente.

I metodi della classe Assert, permettono di confrontare il risultato restituito dal metodo da testare con l'output atteso.

Se un assert è vero, il test case continua l'esecuzione.

Se un assert è falso il test viene interrotto in quel punto e il risultato sarà «Fallito».

Se nessun assert fallisce durante l'esecuzione del test case, il risultato sara «Successo».

JUnit – La classe Assert

Alcuni metodi della classe Assert

- **fail(String msg)** consente di far fallire il test. La variabile **msg** deve contenere il testo del messaggio da visualizzare quando il test fallisce.
- **assertTrue(String msg, boolean cond)** verifica se la condizione «**cond**» è vera . «**msg**» contiene il testo visualizzato se la condizione è falsa e l'assert fallisce.
- **assertFalse(String msg, boolean cond)** verifica se la condizione «**cond**» è falsa . «**msg**» contiene il testo visualizzato se la condizione è vera e l'assert fallisce.
- **assertNull(String msg, Object obj)** verifica se l'oggetto «**obj**» è nullo. «**msg**» contiene il testo visualizzato se l'oggetto non è nullo e l'assert fallisce.
- **assertNotNull(String msg, Object obj)** verifica se l'oggetto «**obj**» non è nullo. «**msg**» contiene il testo visualizzato se l'oggetto è nullo e l'assert fallisce.

JUnit – La classe Assert

Alcuni metodi della classe Assert

- **assertEquals(String msg, Object expected, Object actual)** verifica se «actual» è uguale ad «expected», ovvero se actual.equals(expected) ritorna true. «**msg**» contiene il testo visualizzato se actual.equals(expected) ritorna false e l'assert fallisce.
- **assertNotEquals(String msg, Object expected, Object actual)** verifica se «actual» è diverso da «expected», ovvero se actual.equals(expected) ritorna false. «**msg**» contiene il testo visualizzato se actual.equals(expected) ritorna true e l'assert fallisce.
- **assertSame(String msg, Object expected, Object actual)** verifica se actual==expected è true, «**msg**» contiene il testo visualizzato se actual==expected è false e l'assert fallisce..

JUnit – La classe Assert

Alcuni metodi della classe Assert

- **assertArrayEquals(String msg, Object[] expected, Object[] actual)** verifica se «actuals» è uguale ad «expecteds», ovvero se :

    - gli array hanno la stessa lunghezza

    - per ogni «i», controlla se è vero uno dei seguenti assert:

        - assertEquals(expected[i],actual[i])

        - assertArrayEquals(expected[i],actual[i]).

JUnit – La classe Assert

Esercizio 5 – La classe Assert

JUnit – Test con eccezioni

Ci sono casi in cui i nostri metodi lanciano delle eccezioni in caso di errore.

Con JUnit, facendo fallire volutamente un test, possiamo verificare se viene lanciata l'eccezione corretta.

```
class JUnitException {
void testException() throws Exception {
throw new Exception("Eccezione JUnit");
}
public static void main(String[] args) throws Exception {
JUnitException ex = new JUnitException();
ex.testException();
}
}
```

```
import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.Test;

class JUnitExceptionTest {

@Test
void testTestException() {
JUnitException ex = new JUnitException();

Exception exception = assertThrows(Exception.class, () -> ex.testException());
assertEquals("Eccezione JUnit", exception.getMessage());
}

}
```

JUnit – Test con eccezioni

Esercizio 6 – Test con eccezioni

JUnit Test parametrizzati

Come si crea uno unit test parametrizzato.

JUnit Test parametrizzati

E’ possibile creare **test parametrizzati** che consentono di **effettuare lo stesso test più volte utilizzando diversi valori.**

Per creare un test parametrizzato è necessario:

Annotare la classe con l’annotation **@RunWith(Parameterized.class).** Per utilizzare l’annotation è necessario importare:

- org.junit.runner.RunWith
- org.junit.runner.Parameterized

```
import org.junit.Test;
import org.junit.runner.RunWith;
import org.junit.runners.Parameterized;

@RunWith(Parameterized.class)
public class MathUtilTest {
   …
}
```

JUnit Test parametrizzati

E’ necessario creare un metodo pubblico static annotato con @Parameters che ritorni una Collection contenente i valori da utilizzare come dataset per i test. Ogni elemento della Collection è un array bidimensionale, in cui il primo elemento è il valore da passare nel test, il secondo elemento è l’esito atteso (ad es. [1,false] 1 non è pari…)

```
…

@RunWith(Parameterized.class)
public class MathUtilTest {
  @Parameterized.Parameters
  public static Collection dataset() {
    return Arrays.asList(new Object[][] {
            {1, false},
            {2, true},
            …
            {10, true}
    });
  }
}
```

JUnit Test parametrizzati

Creare il costruttore della classe Test Case che prenda in ingresso i valori dell'i-esimo elemento della collection e due variabili di istanze private a cui assegnare tali valori.

```
…

@RunWith(Parameterized.class)
public class MathUtilTest {
   private int numeroTestato;
   private boolean risultatoAtteso;

   public MathUtilTest(int numeroTestato, boolean risultatoAtteso) {
      super();
      this.numeroTestato = numeroTestato;
      this.risultatoAtteso = risultatoAtteso;
   }
}
```

JUnit Test parametrizzati

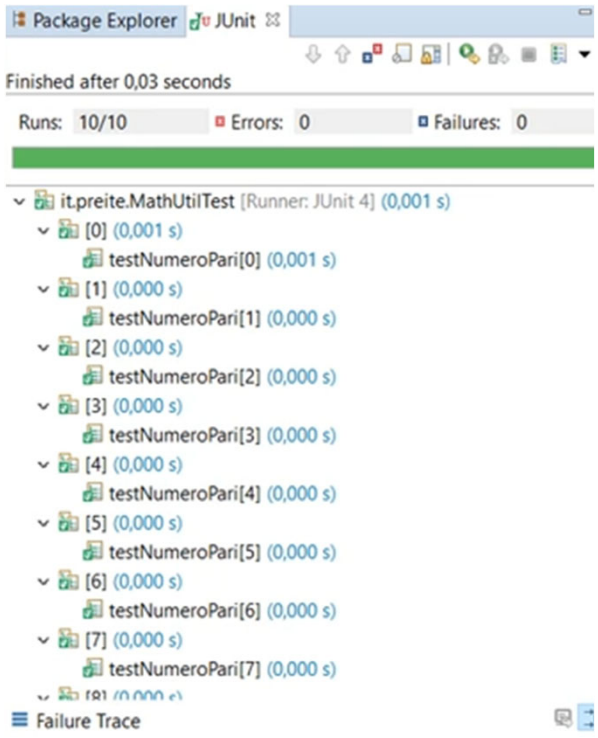
Creare il test da effettuare utilizzando le variabili di istanza nell'assert.

Lanciando la classe, verrà eseguito il test tante volte quanti sono gli elementi presenti nella collection contenuti nel dataset.

```
...

@RunWith(Parameterized.class)
public class MathUtilTest {
  @Test
  public void testNumeroPari() {
    System.out.println("... nel test è : " + numeroTestato);

    assertEquals(
      "Il numero " + numeroTestato + " è DISPARI!",
      risultatoAtteso,
      mathUtil.numeroPari(numeroTestato));
  }
}
```



JUnit - Esercizi

Esercizio 7 – JUnit Parametrizzato

Esercizio 8 – Test Studenti

Esercizio 9 – Test ordinamento array

Esercizio 10 – Test Suite Calcolatrice

Esercizio 11 – Test Calendario

RIFERIMENTI BIBLIOGRAFICI.

- Competitive Programmer's Handbook - Antti Laaksonen 2018.
- Effective Java – Joshua Bloch
- Java how to program – Paul Deitel Harvey Deitel
- www.wikipedia.org
- www.udemy.com
- www.digitalocean.com