JUnit

Alcuni esempi sono tratti da:

Martin Robillard, Introduction to Software Design with Java, 2019, Springer, https://github.com/prmr/DesignBook

Lars Vogel, JUnit 5 tutorial - Learn how to write unit tests, https://www.vogella.com/tutorials/JUnit/article.html

JUnit

- Testing framework open source in Java
  - usato per scrivere ed eseguire unit test *ripetibili*
- Creato a fine ‘90 da Kent Beck & Erich Gamma
  - evoluzione di SUnit per SmallTalk
- E’ uno dei framework XUnit esistenti per lo unit test
  - CUnit, PyUnit, PHPUnit, …

JUnit 5

- Ultima versione
- Introduce supporto a Java 8
  - Annotations
  - Lambda expressions
  - Tag & Filtering
- https://junit.org/junit5/
  - User Guide
  - Javadoc
  - GitHub Repository

Test in JUnit

- Un **caso di test** in Junit è un metodo annotato con **@Test** e contenuto in una **classe di test** usata solo per il testing
  - Per convenzione tali classi contengono la parola *Test* o *Tests* in coda come suffisso
- Le **asserzioni** sono interpretate da JUnit e mostrano messaggi nel caso che il test fallisca

```
package com.vogella.junit.first;

import static org.junit.jupiter.api.Assertions.assertTrue;

import org.junit.jupiter.api.Test;

class AClassWithOneJUnitTest {

    @Test
    void demoTestMethod() {
        assertTrue(true);
    }
}
```

Esempio

Funzione (UUT)

```
Math.abs(5) == 5;
```

Classe di test

```
public class AbsTest
{
   @Test
   public void testAbsPositive()
   {
      assertEquals(5,Math.abs(5));
   }

   @Test
   public void testAbsNegative()
   {
      assertEquals(5,Math.abs(-5));
   }

   @Test
   public void testAbsMax()
   {
      assertEquals(Integer.MAX_VALUE,Math.abs(Integer.MIN_VALUE));
   }
}
```

UUT = unit under test

Esempio

Classe (UUT)

```java
package com.vogella.junit5;

public class Calculator {

    public int multiply(int a, int b) {
        return a * b;
    }
}
```

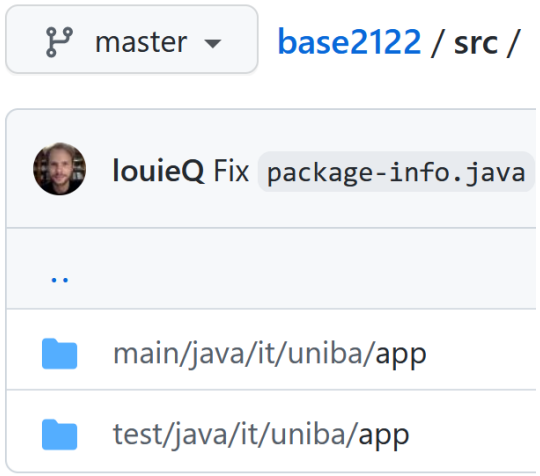
Classe di test

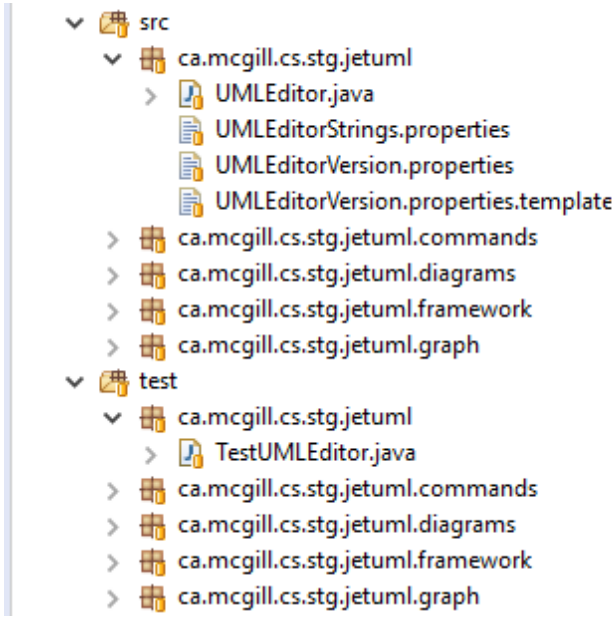
```java
package com.vogella.junit5;

import static org.junit.jupiter.api.Assertions.assertEquals;

import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.RepeatedTest;
import org.junit.jupiter.api.Test;

class CalculatorTest {

    Calculator calculator;

    @BeforeEach                                                    (1)
    void setUp() {
        calculator = new Calculator();
    }

    @Test                                                          (2)
    @DisplayName("Simple multiplication should work")              (3)
    void testMultiply() {
        assertEquals(20, calculator.multiply(4, 5),                (4)
                "Regular multiplication should work");             (5)
    }

    @RepeatedTest(5)                                               (6)
    @DisplayName("Ensure correct handling of zero")
    void testMultiplyWithZero() {
        assertEquals(0, calculator.multiply(0, 5), "Multiple with
zero should be zero");
        assertEquals(0, calculator.multiply(5, 0), "Multiple with
zero should be zero");
    }
}
```

UUT = unit under test

Organizzazione della test suite

- Una classe di test per ogni classe Java del progetto
  - Fanno eccezione le classi UI *boundary* se si è applicato il principio di presentazione separata
- Tutte le classi di test sono in una cartella separata dal resto del codice
  - src/main/java – per classi Java
  - src/test/java - per classi di test
- La struttura della cartella delle classi di test rispecchia la struttura della cartella delle classi Java





Organizzazione di una classe di test

- Creare i **casi di test** come metodi annotati con **`@Test`**
  - Per convenzione tali metodi contengono la parola ***test*** come prefisso, in testa al nome del metodo
- I casi di test devono poter essere eseguiti in qualsiasi ordine e quindi devono essere indipendenti l'uno dall'altro
  - Possibile necessità di creare oggetti prima dell'esecuzione di un test (**@Before**)
  - Possibile necessità di rimuovere oggetti prima dell'esecuzione di un test (**@After**)

Esempio

```
public class StandardTestCase {
        @BeforeAll
        static void setUpAll() {
        }
        @BeforeEach
        void setUp() {
        }
        @Test
        void testSearchRecord() {
        }
        @Test
        void testDeleteRecord() {
        }
        @AfterEach
        void tearDown() {
        }
        @AfterAll
        static void tearDownAll() {
        }
}
```

Es. Apro connessione al database

Es. Aggiungo alla tabella X righe da cercare e cancellare

Es. Cerco righe nella tabella X

Es. Cancello righe dalla tabella X

Es. Ripristino lo stato del db dopo l'esecuzione di ogni metodo di test

Es. Chiudo connessione al database

Asserzioni

org.junit.jupiter.api.Assertions.* package.

```
assertTrue
assertFalse
assertEquals
assertNotEquals
assertNull
assertNotNull
```

```
assertSame
assertNotSame
assertArrayEquals
assertAll
assertThrows
fail
```

assertEquals != assertSame

- `assertEquals()` confronta *i valori* di due oggetti tramite metodo `equals()` della classe

```
assertEquals(4, sum(2, 2)); // OK
// equivale a
i1 = new Integer(4);
i2 = new Integer(sum(2, 2));
assertTrue(i1.equals(i2));
```

- `assertSame()` confronta se due oggetti sono uguali in base ai loro *riferimenti* in memoria

```
assertSame(i1, i2);  // fallisce
assertNotSame(i1, i2);  // OK
```

Asserzioni su array

```
@Test
void testMarks() {
	int[] marks = new int[]{18, 22, 30};
	assertArrayEquals(marks, getMarks("Jim"));
}
```

Raggruppare asserzioni

```
@Test
void testUserCredentials() {
 // In a grouped assertion all assertions are executed,
 // any failure will be reported
 assertAll("Check person credentials with lambdas", () -> {
                    assertNotNull(person.getFirstName());
                    assertEquals("John", person.getFirstName());
                     assertNotNull(person.getLastName());
                      assertEquals("Doe", person.getLastName());
            });
}
```

Testare le eccezioni

```
@Test
void testDivisionByZero() {
	// remember to use a lambda expression here
	assertThrows(ArithmeticException.class, () -> {
		div(7, 0);
	});
}
```

Terminare un test con errore

```
@Test
void testLogin() {
	Login login = new Login (username, password);
	if(login == null)
		fail("Login failed");
}
```

Assunzioni

- `assumeTrue() / assumeFalse()`
- Usate per verificare condizioni assolutamente necessarie al prosieguo del test di unità

```
@BeforeAll
static void setUpAll() {
    connection = new DBConnection("localhost", 46000);
    assumeTrue(connection.isConnected());
}
@Test
void testLogin() {
    Login login = new Login (username, password);
    assumeTrue(login != null, "Login failed");
}
```

Annotazioni

- Usate per configurare i casi di test
- Formato
  - `@annotazione`
- Già viste
  - `@Test`, `@BeforeAll`, `@BeforeEach`, …
- Altre utili
  - `@Disabled`
  - `@DisplayName`
  - `@Tag`

Disabilitare test

```
import org.junit.jupiter.api.Disabled;
import org.junit.jupiter.api.Test;

public class DisabledTestsDemo {
	@Disabled
	@Test
	public void testWillBeSkipped() {
		fail("Not implemented.");
	}

	@Test
	public void testWillBeExecuted() {
	}
}
```

Nomi dei casi di test

- Di default il nome coincide con quello del metodo di test
- Per modificarlo

```
@Test
@DisplayName("Testing sum of integers")
void testAddition() {
    …
}
```

- Persino gli emoji ☺

```
@Test
@DisplayName("A failing test! 😱") // non ne abusate…
void testFailing() {
    …
}
```

Tag & Filtering

È possibile definire custom tag per raggruppare casi di test

```
@Tag("db")
public class TaggingDemoTest {
        @Test
        @Tag("net")
        @Tag("fast")
        void testAddToRemoteDb() {
        }

        @Test
        @Tag("slow")
        void testMassUpdateDb() {
        }
}
```

Anti-pattern

Casi di test con più di un assert

```
// FIX: one test case, one assert!!
@Test
void standardAssertions() {
	assertEquals(4, sum(2, 2));
	assertNotEquals(5, sum(2, 2));
	assertTrue(0 < div(3, 4));
	assertFalse(0 == div(3, 4));
	assertNotNull(buildList('a', 'b', 'c', 'd'));
	assertNull(buildList(null));
}
```

JaCoCo

- JAva COde COverage
- Libreria per calcolare la percentuale di codice sorgente eseguito (“coperto”) dai casi di test
- Code coverage in Eclipse

Coveralls.io

scacchi CI/CD passing coverage 58%

- Servizio web per mostrare un *badge* con la percentuale di coverage di un progetto
  - GitHub, BitBucket, GitLab, …
  - Gratis per progetti open source
- Report di JaCoCo reso accessibile via web
- Mantiene storia del coverage dopo ciascuna build