

Complementi di Ingegneria del Software

JUNIT = implementazione della suite XUnit per
Java: due librerie che eseguono codice
per noi: non progettano test ma lo eseguono.

a.a. 2019-2020

Docente: Prof. Antonio Esposito

Perchè creare una Suite di Test?

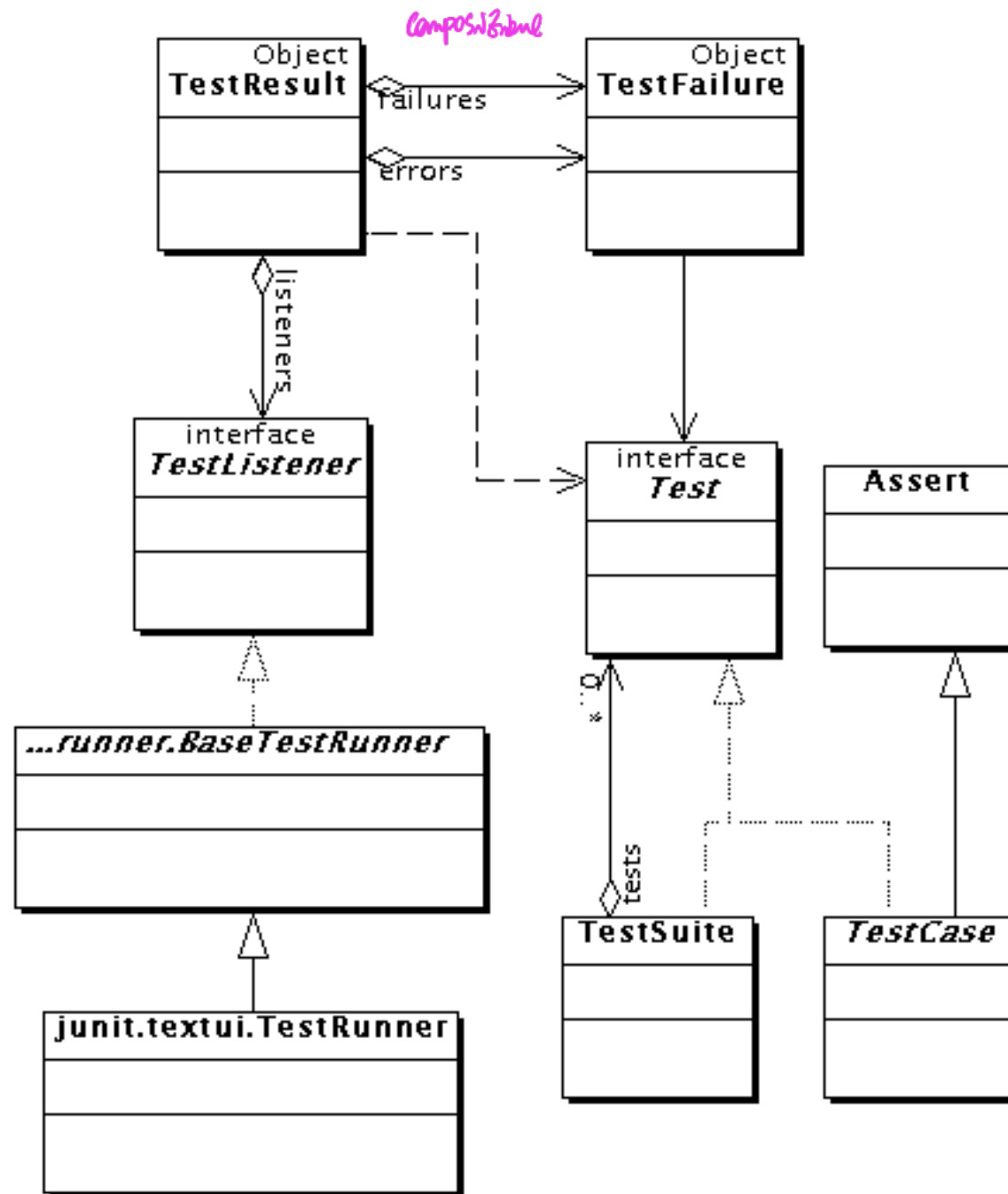
- Ovviamente per testare il codice
 - È possibile creare del test ad-hoc (da lanciare ogni volta che ce n'è bisogno)
 - È possibile creare delle test suite predefinite che eseguono una serie di test in momenti specifici
- Svantaggi
 - Occorre scrivere molto più codice *Scrivere codice che fa il test*
 - Richiede molto più tempo per la progettazione
- Vantaggi:
 - Tempo di debugging molto ridotto
 - Il codice testato evolve nel tempo nel prodotto finale \Rightarrow *Test driven development*
 - I bug nel codice finale risultano ridotti
 - Il codice è più manutenibile *(cioè scrivo il codice in modo che sia facilmente testabile.)*

oggetto che esegue i test

- Il framework di test di JUnit è un package di classi che permettono di scrivere test diversi per ogni metodo e funzionalità da testare
- **TestRunner** esegue i test e restituisce i risultati in un oggetto **TestResults**
- I Test vanno eseguiti estendendo la classe astratta **TestCase**
- I controlli sui test vanno eseguiti usando la classe **Assert**

Assert è al core: variabile che per esempio è nulla

il test fallisce se scopre che l'assert non è rispettato.



Test Case

L'elemento base del framework è il **TestCase**

- Il programmatore deve scrivere i TestCase. Ognuno di essi è una classe che contiene dei metodi di test, scritti dal programmatore
- La classe deve estendere **junit.framework.TestCase**
 - Approccio più moderno -> Uso delle Test Fixtures (@Test)
- La prassi vuole che ci sia un TestCase per ogni classe da testare
- Il nome della classe è importante: deve contenere la parola Test all'inizio o alla fine
 - In questo modo TestRunner può trovarla

*Libreria di Sistema (SWT)
mi dice che tutto il codice che sono è
un test case.*

Esempio:

- **public class TestShoppingCart extends TestCase**

↓ Documentazione

Test Case

- Per ogni funzionalità (metodo) da testare, è bene realizzare un metodo di test

- La prassi prevede la convenzione testXXX() *non obbligatorio ma alcune implementazioni possono dare errori*

```
public class TestShoppingCart extends TestCase{
```

```
    ...
```

```
    public void testClearCart(){...}
```

```
    public void testAddToCart(){...}
```

```
    public void testRemoveFromCart(){...}
```

```
    ...
```

```
}
```

Tante quanti sono i metodi della classe originale

AAA Pattern

Ogni test organizzato seguendo schema per identificare il problema

Arrange, Act, Assert

1. Imposta le precondizioni. Arrange
2. Esecui le funzionalità da testare. Act
3. Verifica le post condizioni. Assert

↑ affinché funzioni codice. ES: 1° metto qualcosa nel carrello per testare acquisto

chiamo funzione che svuota

→ Verifico se ottengo la cosa giusta

Esempio

```
public void testEmptyList() {
```

```
    Bowl emptyBowl = new Bowl();
```

```
    assertEquals("Size of an empty list should be zero.", 0, emptyList.size());
```

```
    assertTrue("An empty bowl should report empty.", emptyBowl.isEmpty()); }
```

Verifico se ottengo Bowl vuota

* ↑ assert ↑ act

"Voglio che il risultato finale sia 0."

Se non ho che è 0, errore con scritto la stringa.

* Descrizione test che sto facendo.

Δ Un oggetto vuoto deve restituire True se è vuoto, assert True: lo comparo con True.

Asserzioni

Per controllare il comportamento delle classi oggetto del test, si usano dei metodi speciali della classe **Assert**

- **TestCase estende junit.framework.Assert** oppure dico **@Assert**, lui lo introduce all'interno del codice
 - Lo deve importare se si usano le Test Fixtures

Da non confondere con le assertion introdotte in Java 1.4

- Quelle di JUnit sono semplici metodi e hanno senso all'interno del framework
- Quelle Java hanno valenza a livello generale
 - Non adatte per il testing, ma complementari alle Eccezioni

Ogni metodo di Assert ha dei parametri del tipo:

- **Messaggio**
 - Il Messaggio è **opzionale** (grazie all'overloading dei metodi di JUnit), ma è sempre meglio metterlo
 - I messaggi aiutano a documentare il test e a capire meglio dove avviene il fallimento
- **Valore atteso** *non sempre presente*
- **Valore reale**

Le asserzioni che lavorano su numeri in virgola mobile hanno anche un parametro aggiuntivo, la tolleranza.

Controllo di Uguaglianza

AssertEquals

Verifica l'uguaglianza di due valori, o due oggetti

- Se valori, viene effettuato un test ==
- Se oggetti, vengono chiamati i rispettivi equals()

```
assertEquals(2,2);
```

```
assertEquals(23, Hour.MAX);
```

```
assertEquals("John", person.getName());
```

```
assertEquals(expectedObj, actualObj);
```

Test di Nullità

AssertNull / AssertNotNull

Verificano se un oggetto ha o meno il valore null

```
assertNull(null);
```

```
assertNull(new Object());
```

non è mai null

```
assertNotNull(new Object());
```

```
assertNotNull(myObject);
```

```
assertNotNull(5); //che succede?
```

↑ Usa le Wrapper: 5 = new Integer(5)

Test Booleano

AssertFalse / AssertTrue

Verificano se una condizione è vera o meno.

```
assertTrue(true);
```

```
assertFalse(true);
```

```
assertTrue(Hour.MAX == 23);
```

```
assertTrue(user.isAuthenticated());
```

Test di Riferimento

AssertSame

Verifica se due oggetti sono gli stessi (cioè, se le due reference puntano alla stessa istanza)

→ Fallisce sempre

```
assertSame(new Object(), new Object());
```

```
Object obj1 = new Object();
```

```
Object obj2 = obj1;
```

```
assertSame(obj1, obj2);
```

✓ Va sempre a buon fine.

TestSuite

↑ *organizzazione*

E' bene raggruppare gruppi di TestCase omogenei in TestSuite

Due ragioni:

- Eseguendo la suite, JUnit esegue automaticamente tutti i test in essa contenuti
- Aiuta a mantenere il codice di test più ordinato

Quando c'è solo un TestCase, JUnit crea dietro le quinte una TestSuite:

```
public static Test suite(){
```

→ gli passo nome di una classe e lui esegue tutti i test

```
    return new TestSuite(MyTestCase.class);
```

```
}
```

↳ implementazione di questo metodo ed esegue

oppure

```
public static Test suite(){
```

```
    TestSuite s = new TestSuite();
```

↑ istanza specifica della classe

```
    s.addTest(new MyTestCase("testXyz"));
```

```
    return s;
```

```
}
```

E con le fixtures?

```
@RunWith(Suite.class)
```

↑ qui devo specificare questo

```
@SuiteClasses({TestJUnit1.class, TestJUnit2.class})
```

TestAll

Prima test suite kulla mykone

Suite di test che prima o poi, tutti creano...

```
public class TestAll extends TestSuite{  
    public static Test suite(){  
        TestSuite suite = new TestSuite("Tutto!");  
        suite.addTestSuite(MyTestCase.class);  
        suite.addTestSuite(MyTestSuite1.class);  
        return suite;  
    }  
    public static void main(String args[]){  
    }  
}
```

Test

Sia **TestCase** che **TestSuite** implementano l'interfaccia **junit.framework.Test**

```
public interface Test{  
    public int countTestCases();  
    public void run(TestResult result);  
}
```

TestResult

L'esecuzione di un test può portare tre risultati:

- **Successo (pass)** *All assert a buon fine*
- **Fallimento (failure)** *Almeno 1 degli assert non ha funzionato. Lancia AssertionError.*
- **Errore (error)** *La test suite non è riuscita a completare. Problema nel codice del test stesso.*

I front end grafici di JUnit mostrano una barra indicante l'esito dei test. La barra è verde solo se il 100% dei test ha successo

In caso di fallimento, viene lanciata una

JUnit.framework.AssertionFailedError

Per eseguire i test JUnit dispone di due classi apposite

`junit.textui.TestRunner` (su console)

`junit.swingui.TestRunner` (grafico)

Failure ed Error

Failure

- E' una condizione "normale" in JUnit. Vuol dire l'oggetto del test non ha prodotto il risultato atteso, ed il test l'ha rilevato
- Il "TDD taliban" è contento

Error

- E' una condizione imprevista
- Qualcosa è andato storto nell'oggetto del test oppure nel test stesso
 - (es: la solita, maledetta NullPointerException ...)

Test Fixtures


V_melketo test

- Possono essere usati per organizzare i Test e ridurre la necessità di estendere le classi di JUnit
- `@Test` indica un singolo test all'interno di una classe
- `Result result = JUnitCore.runClasses(TestJunit.class)`
 - Per invocare il test o la suite Test

TestResult



Classe contenente i test
`@Test`



Test Fixtures

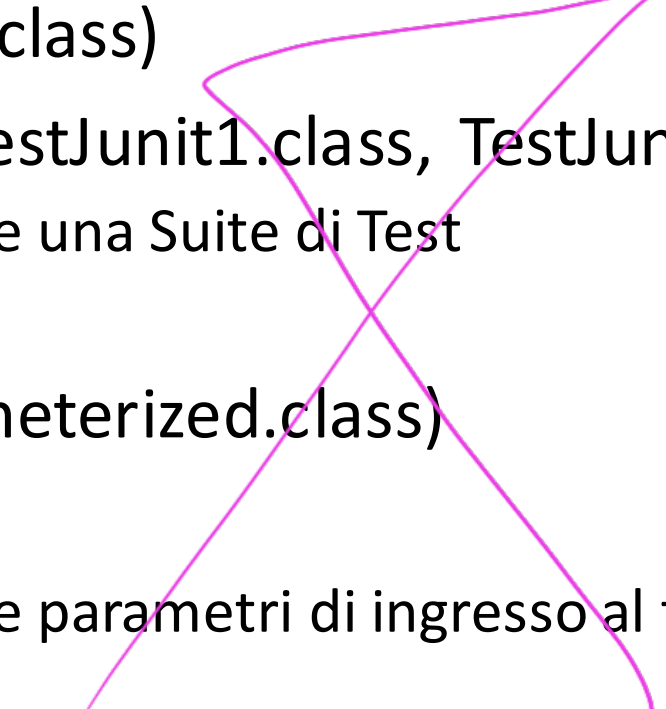
→ Sempre nella classe di test

speciali. Per codice che non è test, per altre cose certe operazioni devono essere fatte prima di tutti i test / dopo

- Da Anteporre\Posporre alla batteria di test o al test stesso
- Modificano l'ordine di esecuzione o impongono alcune elaborazioni pre e post test
- **@BeforeClass/@BeforeAll**
 - Esegui il codice UNA sola volta PRIMA di tutti i Test
- **@AfterClass/@AfterAll**
 - Esegue il codice UNA sola volta DOPO tutti i Test
- **@Before/@BeforeEach**
 - Esegue il codice prima di ogni test per ogni test
- **@After/@AfterEach**
 - Esegue il codice dopo ogni test per ogni test

es. Testi sempre su una lista vuota

Test Fixtures

- `@RunWith(Suite.class)`
 - `@SuiteClasses({TestJUnit1.class, TestJUnit2.class})`
 - Usati per definire una Suite di Test
 - `@RunWith(Parameterized.class)`
 - `@Parameters`
 - Usati per definire parametri di ingresso al test (e gestire lo stesso test con ingressi diversi)
- 

Assert: Riepilogo

1	void assertEquals(boolean expected, boolean actual) Checks that two primitives/objects are equal.
2	void assertTrue(boolean condition) Checks that a condition is true.
3	void assertFalse(boolean condition) Checks that a condition is false.
4	void assertNotNull(Object object) Checks that an object isn't null.
5	void assertNull(Object object) Checks that an object is null.
6	void assertSame(object1, object2) The assertEquals() method tests if two object references point to the same object.
7	void assertNotSame(object1, object2) The assertEquals() method tests if two object references do not point to the same object.
8	void assertEquals(expectedArray, resultArray); The assertEquals() method will test whether two arrays are equal to each other.

Metodi di TestCase

1	int countTestCases() Counts the number of test cases executed by run(<code>TestResult result</code>).
2	TestResult createResult() Creates a default <code>TestResult</code> object.
3	String getName() Gets the name of a <code>TestCase</code> .
4	TestResult run() A convenience method to run this test, collecting the results with a default <code>TestResult</code> object.
5	void run(TestResult result) Runs the test case and collects the results in <code>TestResult</code> .
6	void setName(String name) Sets the name of a <code>TestCase</code> .
7	void setUp() Sets up the fixture, for example, open a network connection.
8	void tearDown() Tears down the fixture, for example, close a network connection.
9	String toString() Returns a string representation of the test case.

Metodi di TestResult

1	void addError(Test test, Throwable t) Adds an error to the list of errors.
2	void addFailure(Test test, AssertionFailedError t) Adds a failure to the list of failures.
3	void endTest(Test test) Informs the result that a test was completed.
4	int errorCount() Gets the number of detected errors.
5	Enumeration<TestFailure> errors() Returns an Enumeration for the errors.
6	int failureCount() Gets the number of detected failures.
7	void run(TestCase test) Runs a TestCase.
8	int runCount() Gets the number of run tests.
9	void startTest(Test test) Informs the result that a test will be started.
10	void stop() Marks that the test run should stop.

Metodi di TestSuite

1	void addTest(Test test) Adds a test to the suite.
2	void addTestSuite(Class<? extends TestCase> testClass) Adds the tests from the given class to the suite.
3	int countTestCases() Counts the number of test cases that will be run by this test.
4	String getName() Returns the name of the suite.
5	void run(TestResult result) Runs the tests and collects their result in a TestResult.
6	void setName(String name) Sets the name of the suite.
7	Test testAt(int index) Returns the test at the given index.
8	int testCount() Returns the number of tests in this suite.
9	static Test warning(String message) Returns a test which will fail and log a warning message.