

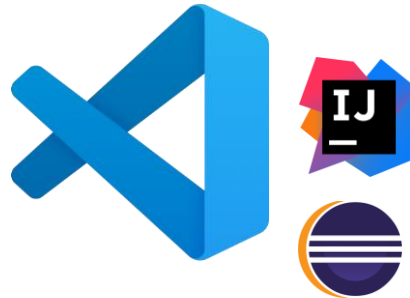


LABORATORIO 5 - TESTING IN JUNIT

Fondamenti di Ingegneria del Software 2024-2025



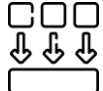
JUNIT

- E' un **framework di testing** per programmi **Java**
 - Unit testing
 - Framework → libreria di classi e convenzioni per usarle (es. @Test)
- Sviluppato da Erich Gamma* e Kent Beck+
 - * Anche Design Pattern ed Eclipse
 - + Anche Extreme Programming e TDD
- Integrato in vari IDE



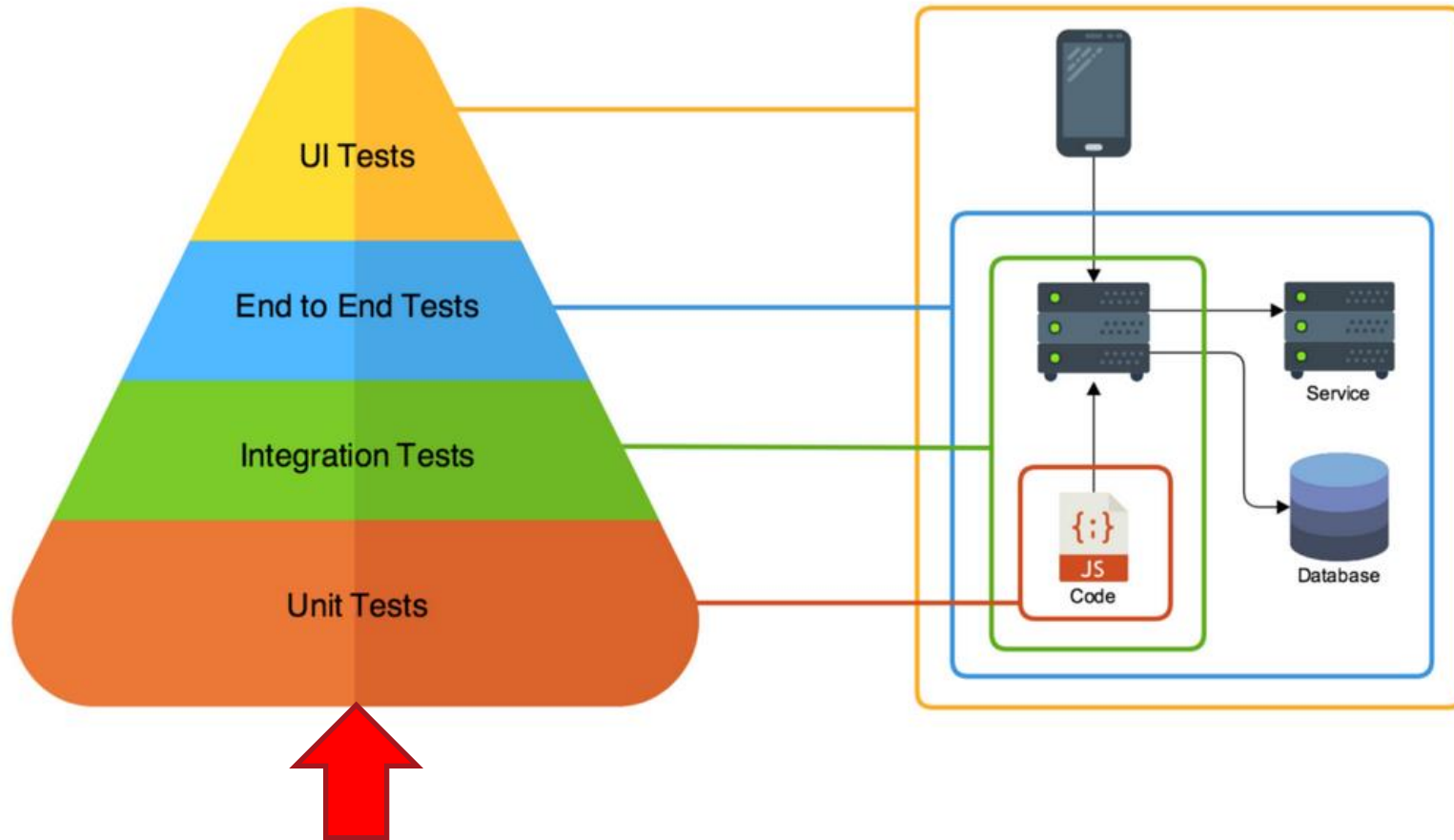
Latest Release: JUnit 5 (5.11.3)
Disponibile a: junit.org

JUNIT: USI

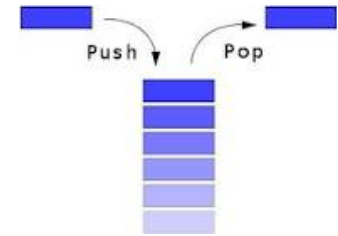
1. **Scrivere casi di test** 
 - Test Class > Test Methods
 - Sequenza di chiamate a metodi + input + expected values
2. **Eseguire casi di test** 
 - **Pass** / **Fail**
3. **Raggruppare casi di test in “Test Suite”** 

Buona pratica: Per ogni classe del progetto,
scrivere una corrispondente classe di test
Es. Stack → StackTest

UNIT VS INTEGRATION VS UI TESTING



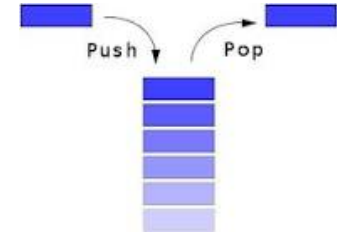
TESTING VIA MAIN()



I casi di test JUnit “sostituiscono” l’uso del **main()** per testare la correttezza di una classe

```
public class StackTest {  
    ...  
    public static void main (String[] args) {  
        Stack aStack = new Stack();  
        if (!aStack.isEmpty())  
            System.out.println ("Stack should be empty!");  
        aStack.push(10);  
        aStack.push(-4);  
        System.out.println("Last element: " + aStack.pop());  
        System.out.println("First element: " + aStack.pop());  
    }  
}
```

TESTING VIA MAIN()



I casi di test JUnit “sostituiscono” l’uso del **main()** per testare la correttezza di una classe

```
public class StackTest {  
    ...  
    public static void main (String[] args) {  
        Stack aStack = new Stack();  
        if (!aStack.isEmpty())  
            System.out.println ("Stack should be empty!");  
        aStack.push(10);  
        aStack.push(-4);  
        System.out.println("Last element: " + aStack.pop());  
        System.out.println("First element: " + aStack.pop());  
    }  
}
```

isEmpty()
pop()
push(...)

10

-4

Output atteso

TEST CLASS

```
public class Stack {  
    isEmpty()  
    pop()  
    push(...)  
}
```

```
import static org.junit.jupiter.api.Assertions.*;  
import org.junit.jupiter.api.Test;  
  
public class StackTest {  
    ...  
    @Test  
    public void test() {  
        fail("Not yet implemented");  
    }  
}
```

TEST CLASS

```
public class Stack {  
    isEmpty()  
    pop()  
    push(...)  
}
```

import di classi e annotazioni
del framework JUnit

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

```
public class StackTest {
```

```
    ...
```

```
    @Test
```

annotazione per marcare
i metodi di test

```
    public void test() {
```

```
        fail("Not yet implemented");
```

```
    }
```

```
}
```

corpo del metodo in cui inserire il caso di test.
In questo esempio, il test fallisce sempre

ASSERTZIONI

- In sostituzione delle print via `main()`, nei test si introducono asserzioni esplicite
- **Assertzione** = affermazione che può essere vera o falsa, automaticamente verificata

```
assertEquals(-4, aStack.pop());
```

- Se l'asserzione è:

expected

actual

- **vera**: il test è andato a buon fine
- **falsa**: il test è fallito e il codice non si comporta come atteso

TEST METHOD

```
public class Stack {  
    isEmpty()  
    pop()  
    push(...)  
}
```

```
...  
@Test  
public void testStack() {  
    Stack aStack = new Stack();  
    assertTrue(aStack.isEmpty(), "Stack should be empty!");  
    aStack.push(10);  
    assertTrue(!aStack.isEmpty());  
    aStack.push(-4);  
    assertEquals(-4, aStack.pop());  
    assertEquals(10, aStack.pop());  
}  
}
```

TEST METHOD

```
public class Stack {  
    isEmpty()  
    pop()  
    push(...)  
}
```

```
...  
@Test  
public void testStack() {  
    Stack aStack = new Stack();  
    assertTrue(aStack.isEmpty(), "Stack should be empty!");  
    aStack.push(10);  
    assertTrue(!aStack.isEmpty());  
    aStack.push(-4);  
    assertEquals(-4, aStack.pop());  
    assertEquals(10, aStack.pop());  
}  
}
```

assertTrue, con stampa su console

assertEquals

ESEGUIRE TEST JUNIT IN VS CODE

Cliccare alternativamente su



oppure su



*Esegue l'intera
Test Class*

*Esegue il singolo
Test Method*

```
9  class TestEuro {
10
11     private Euro euro1;
12     private Euro euro2;
13
14     @BeforeEach
15     public void setUp() {
16         euro1 = new Euro(v:530.5);
17         euro2 = new Euro(v:100);
18     }
19
20     @Test
21     void testSum() {
22         Euro e3 = euro2.sum(euro1);
23         assertEquals(expected:630.5, e3.getValue());
24         fail(message:"Not yet implemented");
25     }
26
```

TEST RUNNER VS CODE (1)

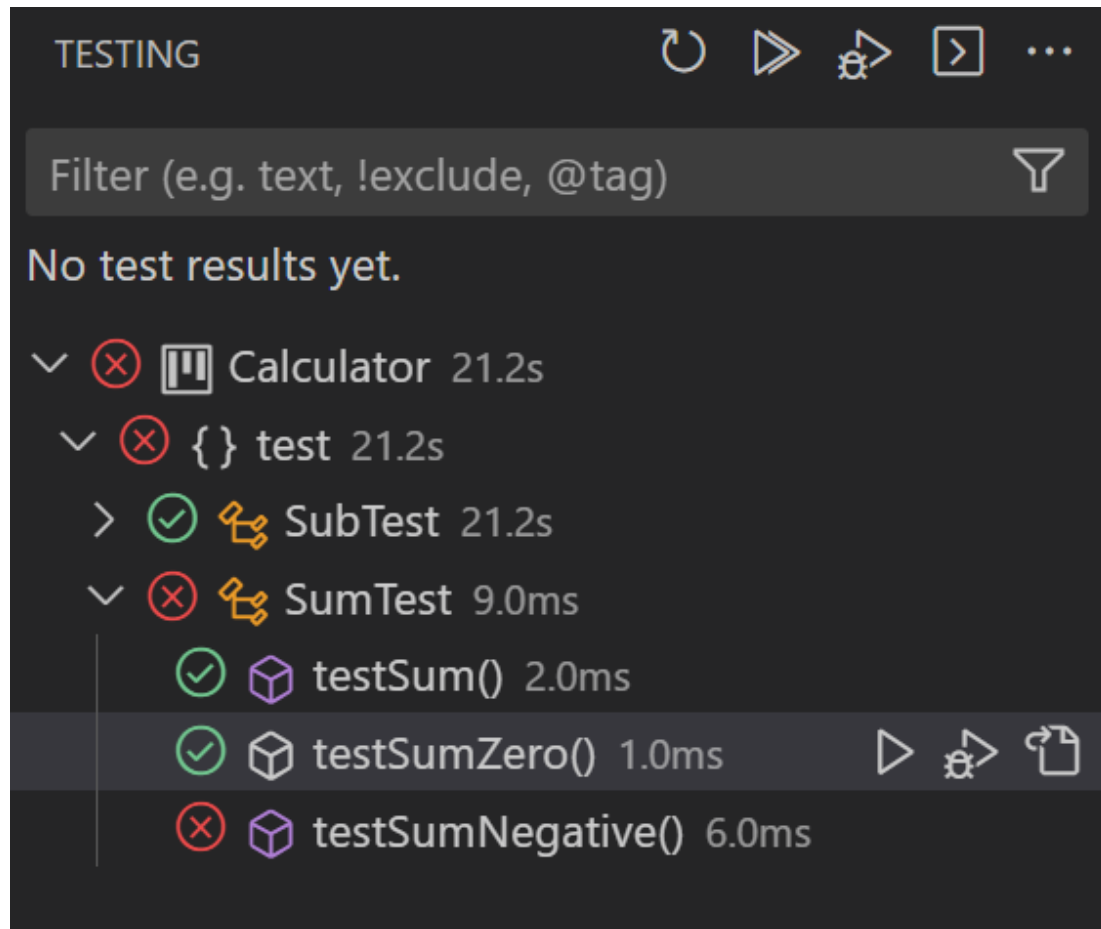
```
8  public class SumTest {
9      @Test
10     public void testSum() {
11         assertEquals(expected:8, Calculator.sum(a:5, b:3));
12     }
13
14     @Test
15     public void testSumZero() {
16         assertEquals(expected:0, Calculator.sum(a:0, b:0));
17     }
18
19     @Test
20     public void testSumNegative() {
21         assertEquals(-3, Calculator.sum(-1, -1)); Expected [-3] but was [-2]
```

Expected [-3] but was [-2] testSumNegative()

Expected	Actual
-3	-2

- Test run at 10/18/2023, 12:39:13 PM
- testSumNegative()
 - Expected [-3] but was [-2]
 - java.lang.AssertionError: expected:[-3] but was:[-2] at test.SumTest.test...
- testSum()

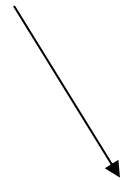
TEST RUNNER VS CODE (2)



FAILURE VS ERROR

- **Failure**: un'asserzione nel Test Method non soddisfatta

- `assertEquals(-3, aStack.pop());`



expected <-3> but was <-4>

- **Error**: errore a run-time del codice che stiamo testando

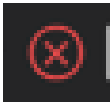
- `assertEquals(-3, saldo.getSaldo());`



`java.lang.ArrayIndexOutOfBoundsException`

FAILURE VS ERROR IN VS CODE

Failure



3/4 tests passed (75.0%)

- ✓ SumNumbers 8.0ms
- ✓ {} test 8.0ms
- ✓ SubTest 5.0ms
- testSub() 5.0ms
- > SumTest 3.0ms

```
6 import code.Calculator;  
7  
8 public class SubTest {  
9     @Test  
10    public void testSub() {  
11        ... assertEquals(expected: 1, Cal
```

java.lang.AssertionError: expected:[1] but was:[3] at test.SubTest.testSub(SubTest.java:11)

Error



3/4 tests passed (75.0%)

- ✓ SumNumbers 8.0ms
- ✓ {} test 8.0ms
- ✓ SubTest 6.0ms
- testSub() 6.0ms
- ✓ SumTest 2.0ms
 - ✓ testSum() 2.0ms
 - ✓ testSumZero() 0.0ms
 - ✓ testSumNegative() 0.0ms

```
6 import code.Calculator;  
7  
8 public class SubTest {  
9     @Test  
10    public void testSub() {  
11        ... assertEquals(expected: 1, Cal
```

java.lang.ArithmeticException: / by zero at code.Calculator

java.lang.ArithmeticException: / by zero
at code.Calculator.sub(Calculator.java:15)
at test.SubTest.testSub(SubTest.java:11)

PROBLEMA DI COESIONE NEI TEST

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

public class StackTest {

    @Test
    public void testStack() {
        Stack aStack = new Stack();
        assertTrue(aStack.isEmpty());
        aStack.push(10);
        assertTrue(!aStack.isEmpty());
        aStack.push(-4);
        assertEquals(-4, aStack.pop());
        assertEquals(10, aStack.pop());
    }
}
```

Vengono testate tutte le operazioni di Stack in un unico metodo!

VERSO LA COESIONE...

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

public class StackTest {

    @Test
    public void testStackEmpty() {
        Stack aStack = new Stack();
        assertTrue(aStack.isEmpty());
        aStack.push(10);
        assertTrue(!aStack.isEmpty());
    }

    @Test
    public void testStackOperations() {
        Stack aStack = new Stack();
        aStack.push(10);
        aStack.push(-4);
        assertEquals(-4, aStack.pop());
        assertEquals(10, aStack.pop());
    }
}
```

PROBLEMA DI CODICE DUPLICATO

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

```
public class StackTest {
```

```
    @Test
```

```
    public void testStackEmpty() {
        Stack aStack = new Stack();
        assertTrue(aStack.isEmpty());
        aStack.push(10);
        assertTrue(!aStack.isEmpty());
    }
```

```
    @Test
```

```
    public void testStackOperations() {
        Stack aStack = new Stack();
        aStack.push(10);
        aStack.push(-4);
        assertEquals(-4, aStack.pop());
        assertEquals(10, aStack.pop());
    }
}
```

Inizializzazione duplicata



BEFOREEACH E AFTEREACH

@BeforeEach **inizializza gli oggetti da testare**

- Metodo chiamato prima di ogni Test Method

@AfterEach **rilascia gli oggetti**

- Metodo chiamato dopo ogni Test Method

```
ShoppingCart cart;  
Book book1, book2;  
  
@BeforeEach  
protected void setUp() {  
    cart = new ShoppingCart();  
    book1 = new Book("JUnit", 29.95);  
    book2 = new Book("XP", 18.25);  
    cart.addItem(book1);  
    cart.addItem(book2);  
}
```

BEFOREEACH E AFTEREACH

@BeforeEach **inizializza gli oggetti da testare**

- Metodo chiamato prima di ogni Test Method

@AfterEach **rilascia gli oggetti**

- Metodo chiamato dopo ogni Test Method

```
public class StackTest {  
    @BeforeEach setUp() {...}  
    @AfterEach tearDown {...}  
  
    @Test testIsEmpty() {...}  
    @Test testPop() {...}  
    @Test testPush(...) {...}  
}
```



```
setUp()  
    testIsEmpty()  
tearDown()  
  
setUp()  
    testPop()  
tearDown()  
  
setUp()  
    testPush()  
tearDown()
```

ESEMPIO BEFOREEACH

```
public class StackTest {  
  
    @Test  
    public void testStackEmpty() {  
        Stack aStack = new Stack();  
        assertTrue(aStack.isEmpty());  
        ...  
    }  
  
    @Test  
    public void testStackOperations() {  
        Stack aStack = new Stack();  
        aStack.push(10);  
        ...  
    }  
}
```



```
public class StackTest {  
  
    Stack aStack;  
  
    @BeforeEach  
    public void setUp() {  
        aStack = new Stack();  
    }  
  
    @Test  
    public void testStackEmpty() {  
        Stack aStack = new Stack();  
        assertTrue(aStack.isEmpty());  
        ...  
    }  
  
    @Test  
    public void testStackOperations() {  
        Stack aStack = new Stack();  
        aStack.push(10);  
        ...  
    }  
}
```

ESEMPIO BEFOREEACH

Inizializzazione in unico
metodo!

```
public class StackTest {

    @Test
    public void testStackEmpty() {
        Stack aStack = new Stack();
        assertTrue(aStack.isEmpty());
        ...
    }

    @Test
    public void testStackOperations() {
        Stack aStack = new Stack();
        aStack.push(10);
        ...
    }

}
```



```
public class StackTest {

    Stack aStack;

    @BeforeEach
    public void setUp() {
        aStack = new Stack();
    }

    @Test
    public void testStackEmpty() {
Stack aStack = new Stack();
        assertTrue(aStack.isEmpty());
        ...
    }

    @Test
    public void testStackOperations() {
Stack aStack = new Stack();
        aStack.push(10);
        ...
    }

}
```

COSA ACCADE QUANDO CI SONO PIÙ ASSERZIONI?

```
public void testStackOperations() {  
    Stack aStack = new Stack();  
    aStack.push(10);  
    aStack.push(-4);  
    assertEquals(-4, aStack.pop(), "pop() error");  
    assertEquals(10, aStack.pop());  
}
```

- Se una **asserzione è falsa**:
 1. Il test fallisce immediatamente (**red**)
 2. La parte restante del metodo viene “saltata”
 3. Il messaggio (se presente) è stampato → **pop() error**
- Se una **asserzione è vera**:
 - l'esecuzione continua normalmente
- Se **tutte le asserzioni sono vere**:
 - Il test passa (**green**)

ASSERT*() E FAIL()

- Per condizioni booleane
 - `assertTrue(boolean condition, “message for fail”);`
- Per uguaglianza tra tipi object, int, string, ...
 - `assertEquals(expected, actual, “message for fail”);`
 - per tipi objects verrà usato il metodo `equals()` definito
 - per gli array, il metodo `equals()` non confronta il contenuto ma il riferimento all'array → usare `assertArrayEquals(expected, actual);`
 - `assertEquals(expected, actual, delta);`
 - Per i tipi double e float è definita una soglia di tolleranza
- Per controllare il riferimento allo stesso oggetto
 - `assertSame(expected, actual);`
 - `assertNotSame(expected, actual);`
- Per far fallire il test immediatamente
 - `fail(“message for fail”);`

JUNIT 5: FUNZIONALITÀ AVANZATE (1)

```
@Test
public void testIllegalArgException() {
    assertThrows(IllegalArgumentException.class,
        () -> { value.divideBy(0); })
}
```

Testare le eccezioni

```
@Test
public void doPaymentNotExceed15Seconds() {
    OrderService orderService = new OrderService();
    assertTimeout(ofSeconds(15),
        () -> { orderService.doPayment(); })
}
```

**Testare il
tempo di
esecuzione**

*Il metodo deve
completarsi in 15 secondi*

Riunire casi di test in Test Suite

```
@RunWith(JUnitPlatform.class)
@SelectClasses({TestStack4.class, ParametrizedStackIntTest.class})
public class TestAll { ... }
```

JUNIT 5: FUNZIONALITÀ AVANZATE (2)

Inizializzazione flessibile

- @BeforeEach
- @AfterEach
- @BeforeAll
- @AfterAll

```
setUp()  
    testMethod1()  
tearDown()  
  
setUp()  
    testMethod2()  
tearDown()  
  
setUp()  
    testMethod3()  
tearDown()
```

```
setUp()  
    testMethod1()  
    testMethod2()  
    testMethod3()  
tearDown()
```

JUNIT 5: FUNZIONALITÀ AVANZATE (3)

Test Parametrici

```
@ParameterizedTest
@ValueSource(strings = { "Hello", "JUnit" })
void testWithValueSource(String word) {
    assertNotNull(word);
}
```

▼	✓ Test Results	26ms
▼	✓ HelloParams	26ms
▼	✓ testWithValueSource(String)	26ms
	✓ [1] Hello	25ms
	✓ [2] JUnit	1ms