



# Alma Mater Studiorum-Università di Bologna

## Scuola di Ingegneria

---

### Fondamenti di Informatica T2

### **Introduzione a JUnit**

*Corso di Laurea in Ingegneria Informatica*

*Anno accademico 2021/2022*

**Prof. ROBERTA CALEGARI**

**Prof. AMBRA MOLESINI**

*Dipartimento di Informatica – Scienza e Ingegneria (DISI)*



# Oltre le **assert**

- Le asserzioni sono un grosso passo avanti rispetto alle print, MA **scrivere molti test basati su `assert` è laborioso e scomodo**
- Per usarle efficacemente occorre **strumentarle**
- Serve un'**infrastruttura** che garantisca:
  - la **notifica** dei problemi in un formato amichevole e analizzabile
  - l'**esecuzione di tutti i test** *anche nel caso in cui alcuni falliscano*
    - adesso, la prima **assert** che fallisce causa l'*abort* del programma
  - un **report leggibile** su quali test siano falliti e quali passati
    - da sola, **assert** non spiega perché sia fallita (il suo argomento è solo un **boolean**, senza messaggi..)
  - l'azzeramento della replicazione del codice (stampe e quant'altro)

## Cos'è?

- Un **framework di test** scritto esso stesso in Java: una serie di classi gestiscono il lavoro ripetitivo, come
  - far girare i test in batch
  - contare e riportare gli errori e i test falliti, ...

## Cosa fa?

- **Strumenta efficacemente il processo di Unit Testing** (da cui il nome)
- IDEA: verificare il funzionamento di **piccole porzioni (unità) di codice** (un **metodo**, una **classe**, max un **componente**).

## Dov'è?

- È integrato all'interno dell'ambiente di sviluppo
- ma funziona anche a linea di comando

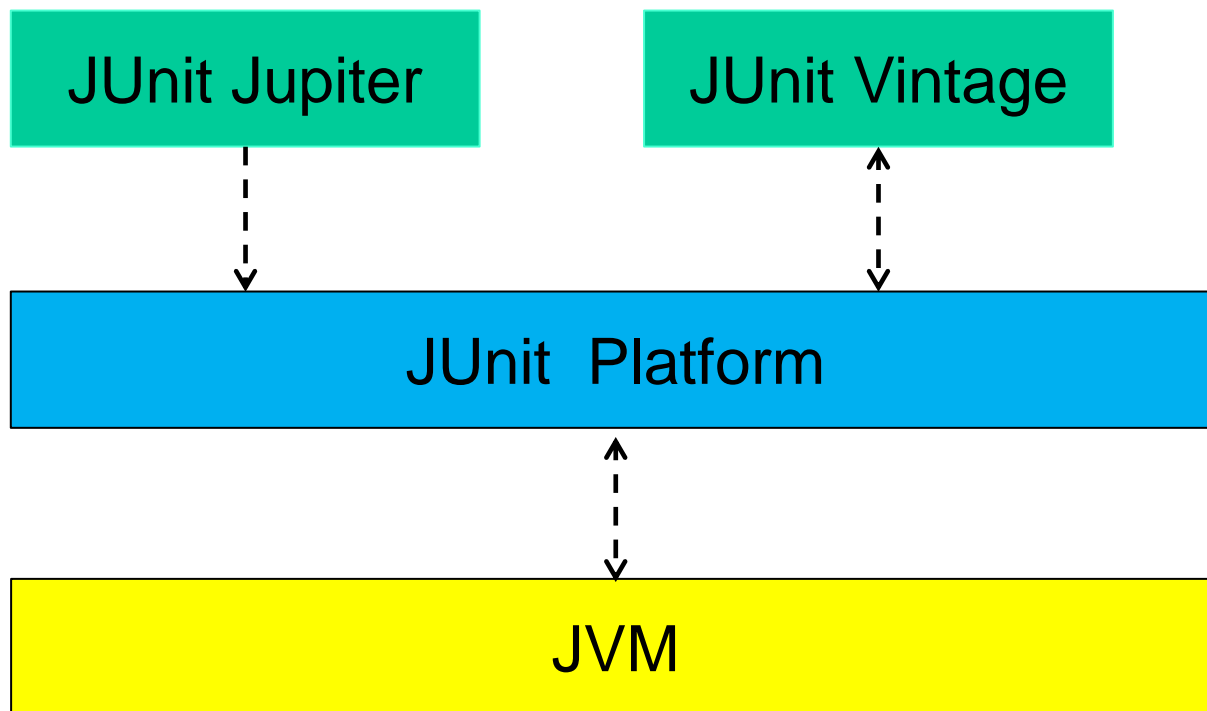
# JUnit 5

---

- Attualmente il framework JUnit è arrivato alla versione 5
- L'architettura del framework ha subito un netto cambiamento rispetto alle versioni precedenti che prevedevano uno *stile architetturale monolitico*
- Il framework ora è composto da 3 sotto-parti inter-dipendenti tra loro, ciascuna formata da diversi moduli

JUnit5 = ***JUnit Platform + JUnit Jupiter + JUnit Vintage***

# JUnit 5

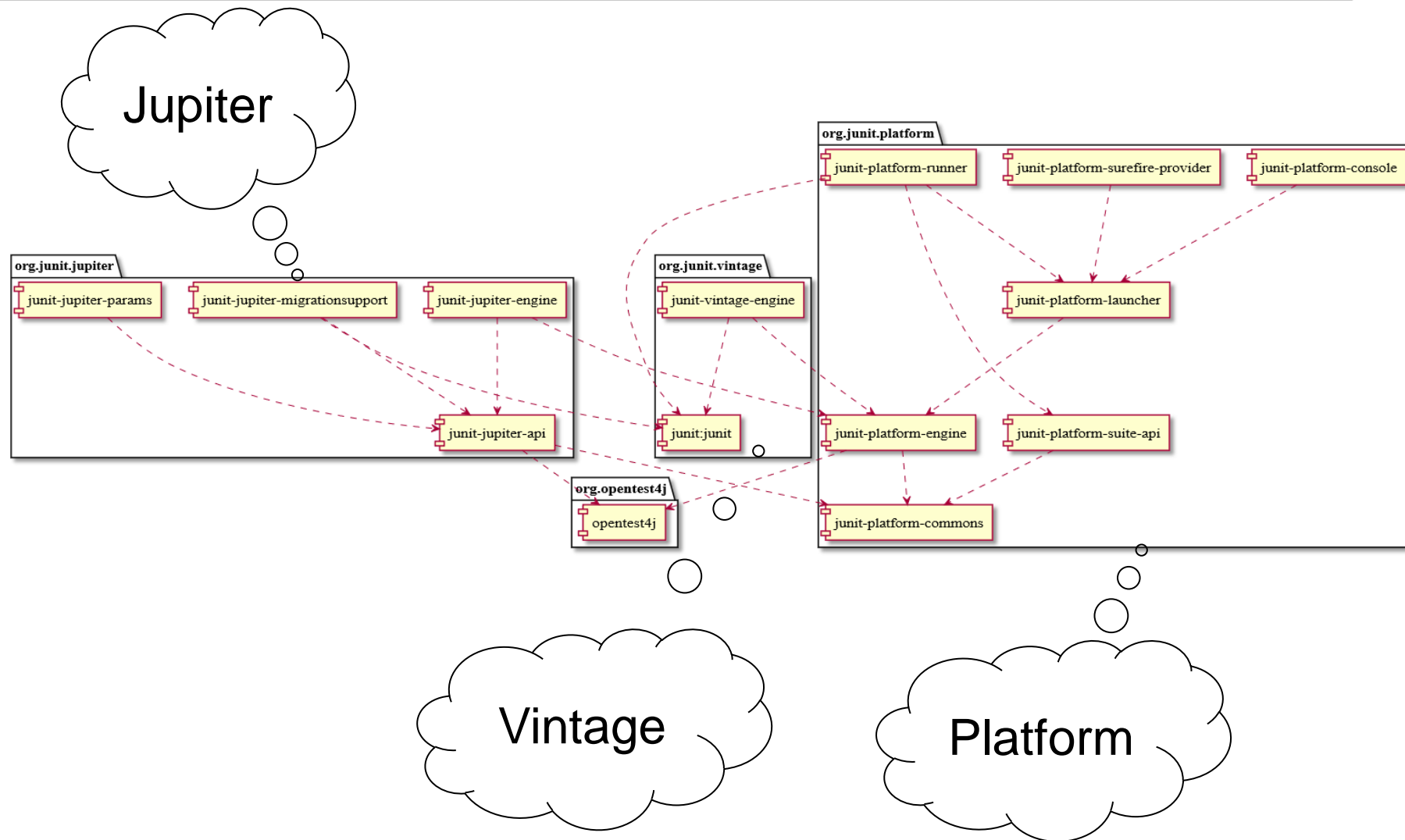


- A runtime è richiesto Java 8 o superiore
- In Eclipse basta importare la libreria JUnit5, è tutto già compreso

# JUnit 5

- *JUnit Platform* fornisce le basi per lanciare il framework di test
  - definisce le TestEngine API per lo sviluppo di un framework di test
  - fornisce un Launcher per Console per lanciare la piattaforma anche dalla riga di comando
  - fornisce un Runner basato su JUnit 4 per eseguire qualsiasi TestEngine sulla piattaforma
- *JUnit Jupiter* è il nuovo modello di programmazione per la scrittura di test ed estensioni
- *JUnit Vintage* fornisce le TestEngine API per eseguire test basati su JUnit 3 e JUnit 4

# Architettura JUnit 5



# Terminologia

- **Test Fixture:** una **classe** i cui metodi realizzano i casi di test di un'entità che si vuole raggruppare
  - ATTENZIONE: non tutti i metodi di una *Fixture* realizzano test: possono esserci *altri* metodi con *altri* scopi (è pur sempre una normalissima classe..)
- **Come distinguere i metodi di test dagli altri?**
  - **NON DAL NOME:** voglio poter usare i nomi che voglio
  - **MA** tramite un opportuno "tag": **@Test**
  - *@Test* è un esempio di *annotation*



# Test Runner

- Il **Test Runner** è il motore che fa girare i test
  - esegue TUTTE le test fixtures
  - esegue TUTTI i metodi di test di CIASCUNA test fixture
- **L'ordine di esecuzione è volutamente non deterministico**
  - i test devono essere *indipendenti* gli uni dagli altri, ovvero **il risultato di ogni test non deve dipendere dall'ordine di esecuzione**
    - se succede, sono progettati male
  - per evitare influenze reciproche, **un buon test non deve dare per scontato che *altri metodi funzionino correttamente***
    - ad esempio, il test che verifica l'elaborazione di informazioni lette da un file *non dovrebbe dare per scontato che la lettura da file funzioni*
  - necessità di **mock** per simulare altri pezzi del sistema



# JUnit – Primo Esempio

```
public class MyCalendarTest {  
    MyCalendar myCal;  
  
    @Test  
    public void testAdd() {  
        LocalDateTime from = LocalDateTime.of(2019, Month.MARCH, 10,  
        12, 30, 0);  
        LocalDateTime to = LocalDateTime.of(2019, Month.MARCH, 15,  
        15, 30, 0);  
        Appointment app = new Appointment("Colloquio", from, to);  
        myCal.add(app);  
        int expected = 1;  
        org.junit.jupiter.api.Assertions.assertEquals(expected,  
        myCal.getAllAppointments().size());  
    }  
}
```

Metodo di test

**assertEquals** verifica  
che il **valore** sia quello atteso

**assertEquals** è una funzione statica della classe **Assertions**,  
che fa parte del pacchetto software "org.junit.jupiter.api"



# JUnit – Primo Esempio

- Opportuna una **import** per non dover ripetere ogni volta **"org.junit.jupiter.api.Assertions"**
- Ricorda, ci sono due direttive distinte:
  - **import**  
importa i nomi delle classi di un *intero pacchetto software* permettendo di usare i nomi delle classi importate senza il prefisso del package
  - **import static**  
importa *solo i membri statici* (*campi e metodi*) di una *singola classe*
- Esempi
  - **import** **org.junit.jupiter.\*;**  
importa i nomi delle classi dell' intero pacchetto software
  - **import static** **org.junit.jupiter.api.Assertions.\*;**  
importa i nomi dei *metodi statici* della sola classe ***Assertions***

# JUnit – Primo Esempio

```
import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.*;
public class MyCalendarTest {
    MyCalendar myCal;
    @Test
    public void testAdd() {
        LocalDateTime from = LocalDateTime.of(2019, Month.MARCH, 10, 12,
        30, 0);
        LocalDateTime to = LocalDateTime.of(2019, Month.MARCH, 10, 15,
        30, 0);
        Appointment app = new Appointment("Compleanno", from, to);
        myCal.add(app);
        assertEquals(1, myCal.getAllAppointments().size());
    }
```

Importa *nomi pubblici*  
definiti dalla classe Assertions

Importa le *classi pubbliche*  
del package junit.org.jupiter.api

Non occorre più la dizione completa  
**org.junit.jupiter.api.Assertions.assertEquals:**  
basta la forma breve

# DA `assert` A `assertXXX`

In luogo della *singola istruzione* `assert`, JUnit offre una famiglia di metodi **`assertXXX`** specializzati:

Metodo	Descrizione
<code>assertEquals(expected, current)</code>	Verifica che i due <i>valori</i> siano <i>uguali</i> → equals
<code>assertArrayEquals(expected, current)</code>	Verifica che i due array siano uguali → equals
<code>assertTrue(condition)</code>	Verifica che la condizione sia vera
<code>assertFalse(condition)</code>	Verifica che la condizione sia false
<code>assertNull(value)</code>	Verifica che il valore sia nullo
<code>assertNotNull(value)</code>	Verifica che il valore non sia nullo
<code>assertSame(expected, current)</code>	Verifica che i <i>referimenti</i> puntino allo <i>stesso oggetto</i>
<code>assertNotSame(expected, current)</code>	Verifica che i <i>referimenti</i> puntino a <i>oggetti distinti</i>
<code>fail</code>	Forza il fallimento del test
....	



# La famiglia **assertXXX**

- Occhio a **assertEquals** e **assertSame**
  - **assertSame** si basa sulla coincidenza dei riferimenti
  - **assertEquals** si basa invece su **equals** ..

Metodo	Descrizione
<b>assertEquals(expected, current)</b>	Verifica che i due <i>valori</i> siano <i>uguali</i> → equals
assertArrayEquals(expected, current)	Verifica che i due array siano uguali → equals
assertTrue(condition)	Verifica che la condizione sia vera
assertFalse(condition)	Verifica che la condizione sia false
assertNull(value)	Verifica che il valore sia nullo
assertNotNull(value)	Verifica che il valore non sia nullo
<b>assertSame(expected, current)</b>	Verifica che i <i>riferimenti</i> puntino allo <i>stesso oggetto</i>
<b>assertNotSame(expected, current)</b>	Verifica che i <i>riferimenti</i> puntino a <i>oggetti distinti</i>
fail	Forza il fallimento del test



# La famiglia **assertXXX**

- Occhio a **assertEquals** e **assertSame**

- **assertSame** si basa sulla coincidenza dei riferimenti
- **assertEquals** si basa invece su **equals** ..

*ma la nostra equals non funziona!!*

```
c1 = new MyCalendar();
```

```
c2 = new MyCalendar();
```

```
assertEquals(c1, c2);
```

**FALSA ??? Perché?**

- Scopriremo fra qualche lezione il vero motivo
  - la nostra **equals** attuale ha un *difetto di fondo* che la rende inusabile da un *framework* come JUnit
  - si può ovviare con **assertTrue(c1.equals(c2))** ;



# Confrontare valori *floating point*

- A causa della rappresentazione dei numeri reali, l'uguaglianza fra valori reali è un concetto *approssimato*
- Di conseguenza, **assertEquals** per valori reali è diversa dalle altre, perché è vera *a meno di un "delta" da specificare*  
→ *3 argomenti anziché 2*
  - il valore atteso (expected)
  - il valore corrente (actual)
  - **il delta massimo (delta)**ovvero *l'intorno* entro cui i due valori sono considerati “uguali”

```
double d1 = 0.1 + 0.1 + 0.1;  
double d2 = 0.3;  
double d3 = 1.0 - 0.7;
```

SEMBRANO tutti uguali...  
...ma non lo sono



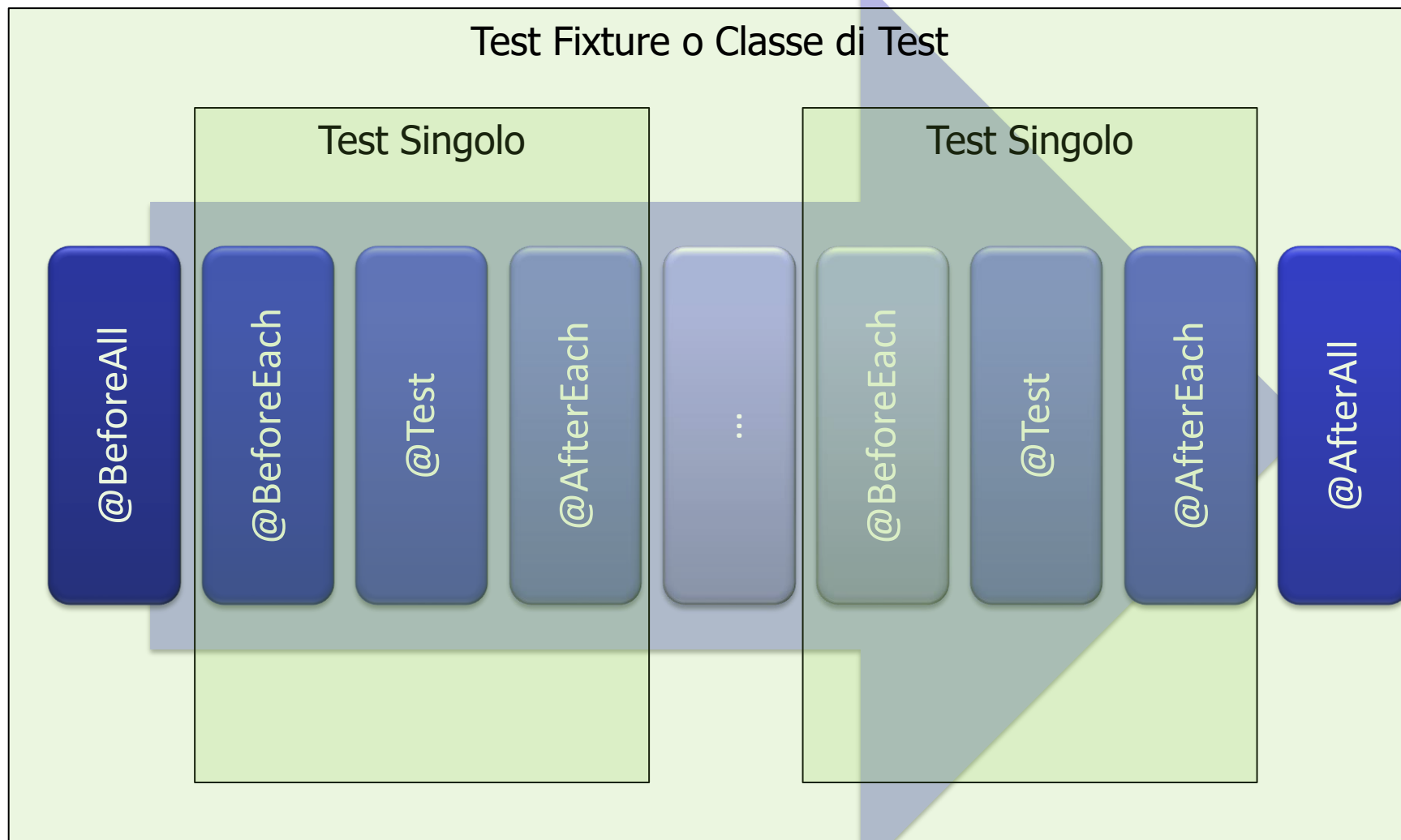
# Strutturazione dei test

- **È frequente che più test richiedano operazioni comuni:**
  - prima e dopo l'esecuzione di **tutti i test di una intera classe**
  - prima e dopo l'esecuzione di un **singolo test**
- **A tal fine conviene:**
  - incapsulare in opportuni metodi quelle operazioni comuni
  - etichettare tali metodi per farli riconoscere al motore Test Runner
- **Annotation disponibili:**
  - **@BeforeAll / @AfterAll:** marca una singola funzione statica eseguita PRIMA/DOPO una **intera classe di test**
  - **@BeforeEach / @AfterEach:** marca un metodo non statico eseguito PRIMA/DOPO **ogni singolo metodo di test**

**Statica** perché riguarda l'intera classe,  
**non il singolo oggetto-test**

**Non statico** perché  
riguarda il **singolo**  
**oggetto-test**

# Sequenza di esecuzione





# Test di MyCalendar (1/3)

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

public class MyCalendarTest {
    MyCalendar myCal;

    @Test
    public void testAdd() {
        myCal = new MyCalendar();
        LocalDateTime from = LocalDateTime.of(2019, Month.MARCH, 10, 12, 30,
0);
        LocalDateTime to = LocalDateTime.of(2019, Month.MARCH, 10, 15, 30,
0);
        Appointment app = new Appointment("Compleanno", from, to);
        myCal.add(app);
        assertEquals(1, myCal.getAllAppointments().size());
    }
    ...
}
```



# Test di MyCalendar(2/3)

...

```
@Test
public void testGetListAppointmentDay() {
    myCal = new MyCalendar();
    LocalDateTime from1 = LocalDateTime.of(2019, Month.MARCH, 10, 12, 30,
0);
    LocalDateTime to1 = LocalDateTime.of(2019, Month.MARCH, 10, 15, 30, 0);
    Appointment app1 = new Appointment("Compleanno", from1, to1);
    myCal.add(app1);
    LocalDateTime from3 = LocalDateTime.of(2019, Month.MARCH, 10, 20, 30,
0);
    LocalDateTime to3 = LocalDateTime.of(2019, Month.MARCH, 10, 22, 30, 0);
    Appointment app3 = new Appointment("Cena Lavoro", from3, to3);
    myCal.add(app3);
    LocalDate date = LocalDate.of(2019, Month.MARCH, 10);
    AppointmentCollection coll = myCal.getDayAppointments(date);
    assertEquals(2, coll.size());
}
```

...



# Test di MyCalendar (3/3)

@Test

```
public void testRemove() {  
    myCal = new MyCalendar();
```

```
    LocalDateTime from1 = LocalDateTime.of(2019, Month.MARCH, 10, 12, 30, 0);  
    LocalDateTime to1 = LocalDateTime.of(2019, Month.MARCH, 10, 15, 30, 0);  
    Appointment app1 = new Appointment("Compleanno", from1, to1);  
    myCal.add(app1);  
  
    LocalDateTime from3 = LocalDateTime.of(2019, Month.MARCH, 10, 20, 30, 0);  
    LocalDateTime to3 = LocalDateTime.of(2019, Month.MARCH, 10, 22, 30, 0);  
    Appointment app3 = new Appointment("Cena Lavoro", from3, to3);  
    myCal.add(app3);  
  
    boolean removed = myCal.remove(to3);  
    assertTrue("Appointment removed", removed);  
    LocalDateTime date = LocalDateTime.of(2019, Month.MARCH, 10, 20, 30, 0);  
    assertEquals("Appointment not removed", myCal.isAppointment(date), true);  
}
```

**Osserviamo che la frase:**  
**`myCal = new MyCalendar();`**  
**è presente e replicata in tutti i tre test.**  
***Ha senso farla una sola volta all'inizio.***



# Test di MyCalendar - revised (1/3)

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

```
import org.junit.jupiter.api.*;
```

```
public class MyCalendarTest {
```

```
MyCalendar myCal;
```

```
    @BeforeEach
```

```
    public void setUp() {
```

```
        myCal = new MyCalendar();
```

```
    }
```

```
    @Test
```

```
    public void testAdd() {
```

```
        LocalDateTime from = LocalDateTime.of(2019, Month.MARCH, 10, 12, 30, 0);
```

```
        LocalDateTime to = LocalDateTime.of(2019, Month.MARCH, 10, 15, 30, 0);
```

```
        Appointment app = new Appointment("Compleanno", from, to);
```

```
        myCal.add(app);
```

```
        assertEquals(1, myCal.getAllAppointments().size());
```

```
    }
```

```
...
```

Viene eseguito **prima** di ogni test  
→ utile per catturare  
*codice comune a tutti i test*

Non c'è più la riga con la creazione  
di MyCalendar, perché è implicita



# Test di MyCalendar – revised (2/3)

...

`@Test`

```
public void testGetListAppointmentDay() {
```

```
    LocalDateTime from1 = LocalDateTime.of(2019, Month.MARCH, 10, 12, 30, 0);
```

```
    LocalDateTime to1 = LocalDateTime.of(2019, Month.MARCH, 10, 15, 30, 0);
```

```
    Appointment app1 = new Appointment("Compleanno", from1, to1);
```

```
    myCal.add(app1);
```

```
    LocalDateTime from3 = LocalDateTime.of(2019, Month.MARCH, 10, 20, 30, 0);
```

```
    LocalDateTime to3 = LocalDateTime.of(2019, Month.MARCH, 10, 22, 30, 0);
```

```
    Appointment app3 = new Appointment("Cena Lavoro", from3, to3);
```

```
    myCal.add(app3);
```

```
    LocalDate date = LocalDate.of(2019, Month.MARCH, 10);
```

```
    AppointmentCollection coll = myCal.getDayAppointments(date);
```

```
assertEquals(2, coll.size());
```

```
}
```

...

Non c'è più la riga con la creazione di MyCalendar



# Test di MyCalendar - revised (3/3)

Non c'è più la riga con la creazione di MyCalendar

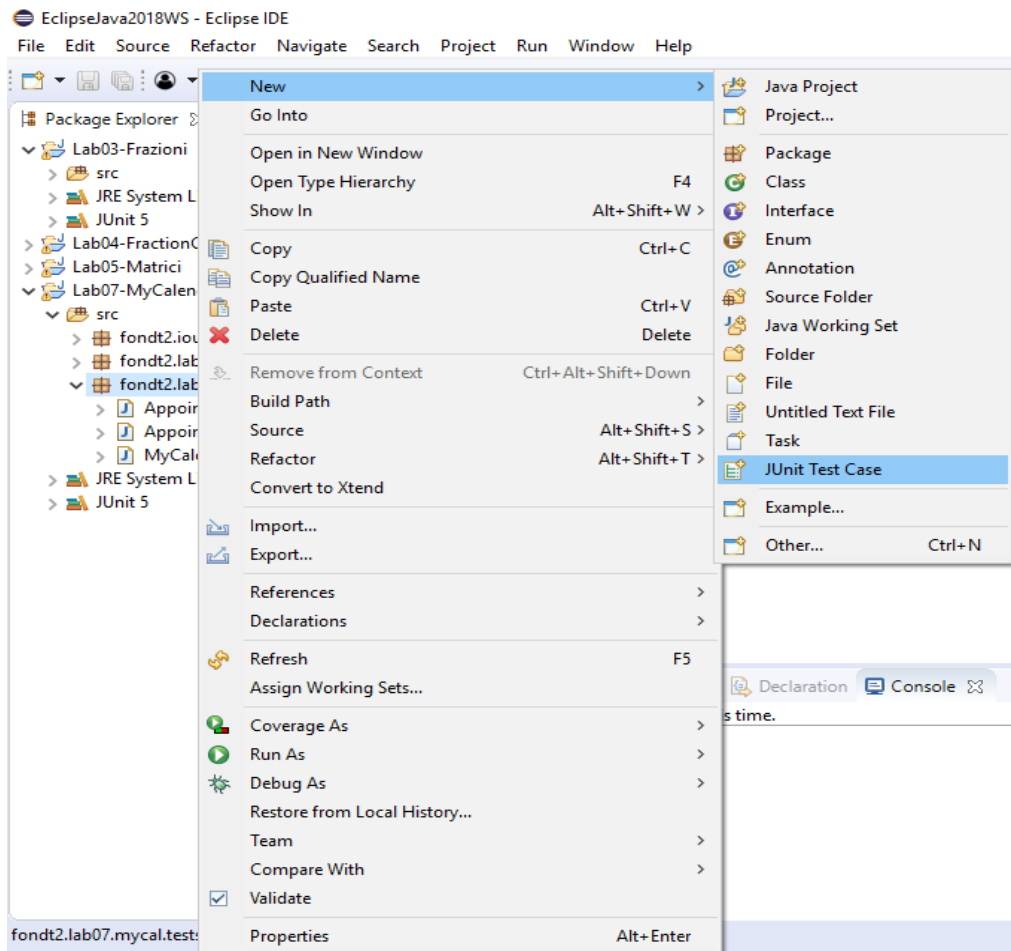
@Test

```
public void testRemove() {
    LocalDateTime from1 = LocalDateTime.of(2012, Month.MARCH, 10, 12, 30, 0);
    LocalDateTime to1 = LocalDateTime.of(2012, Month.MARCH, 10, 15, 30, 0);
    Appointment app1 = new Appointment("Compleanno", from1, to1);
    myCal.add(app1);
    LocalDateTime from3 = LocalDateTime.of(2012, Month.MARCH, 10, 20, 30, 0);
    LocalDateTime to3 = LocalDateTime.of(2012, Month.MARCH, 10, 22, 30, 0);
    Appointment app3 = new Appointment("Cena Lavoro", from3, to3);
    myCal.add(app3);
    boolean removed = myCal.remove(app3);
    assertTrue(removed);
    LocalDate date = LocalDate.of(2012, Month.MARCH, 10);
    assertEquals(1, myCal.getDayAppointments(date).size());
}
}
```



# Integrazione con Eclipse (1)

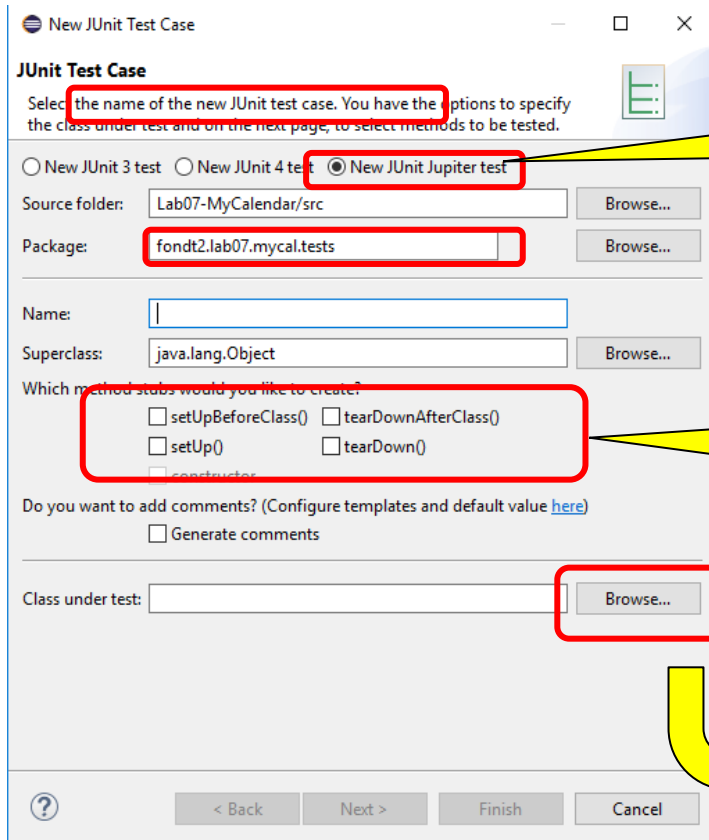
Eclipse facilita la creazione dei test tramite *wizard*



Tasto destro sulla classe che  
si intende collaudare  
New > JUnit Test Case

# Integrazione con Eclipse (2)

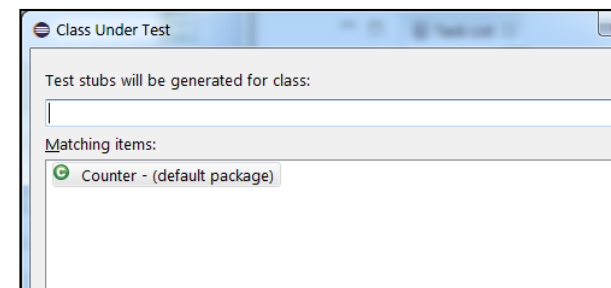
Il wizard permette di *scegliere i metodi da collaudare e crea lo scheletro della Fixture di test.*



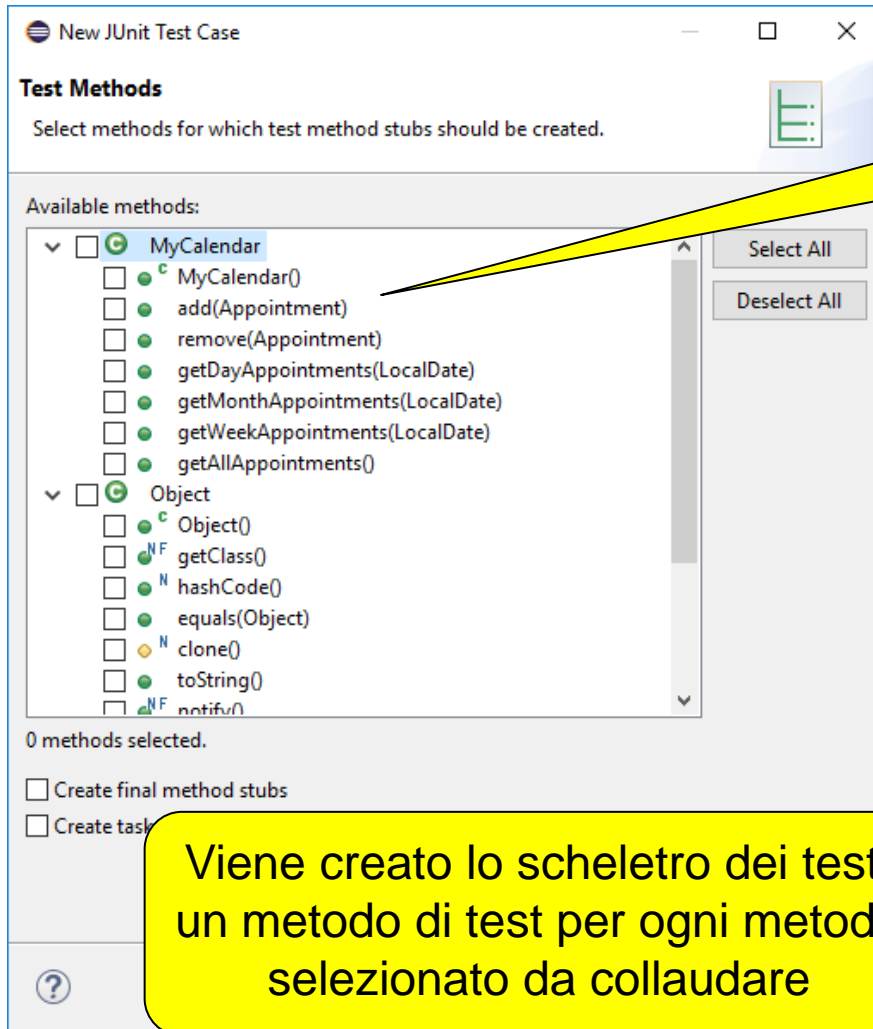
**Importante impostare JUnit 5**

- la sintassi di JUnit 4 era diversa

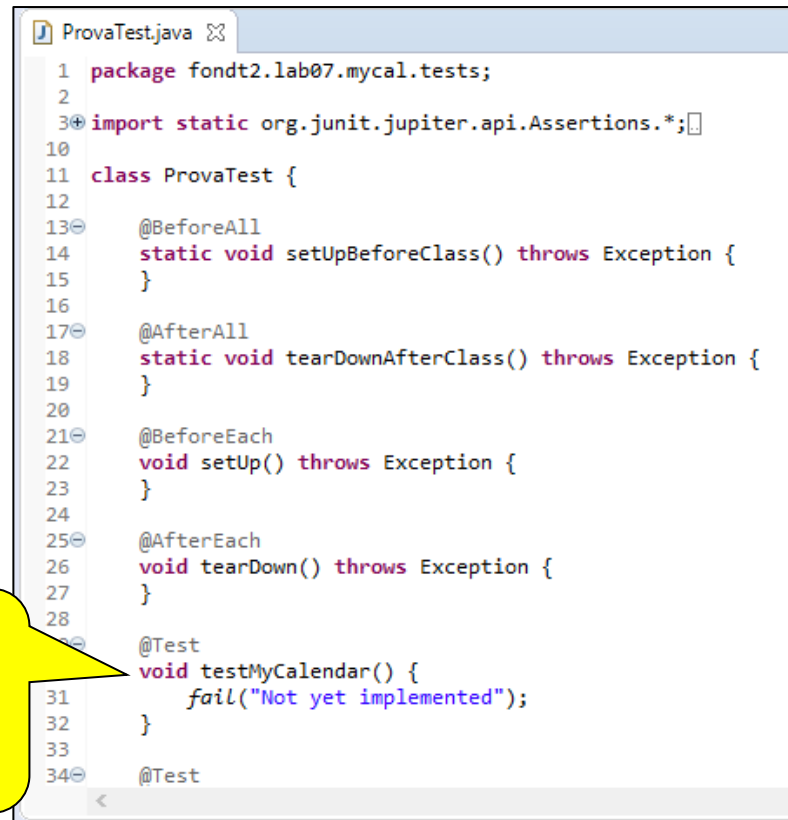
Genera lo scheletro dei due metodi **@AfterEach**, **@BeforeEach** e delle due funzioni statiche **@AfterAll**, **@BeforeAll**



# Integrazione con Eclipse (3)



Selezione dei metodi di cui si vogliono effettuare i test

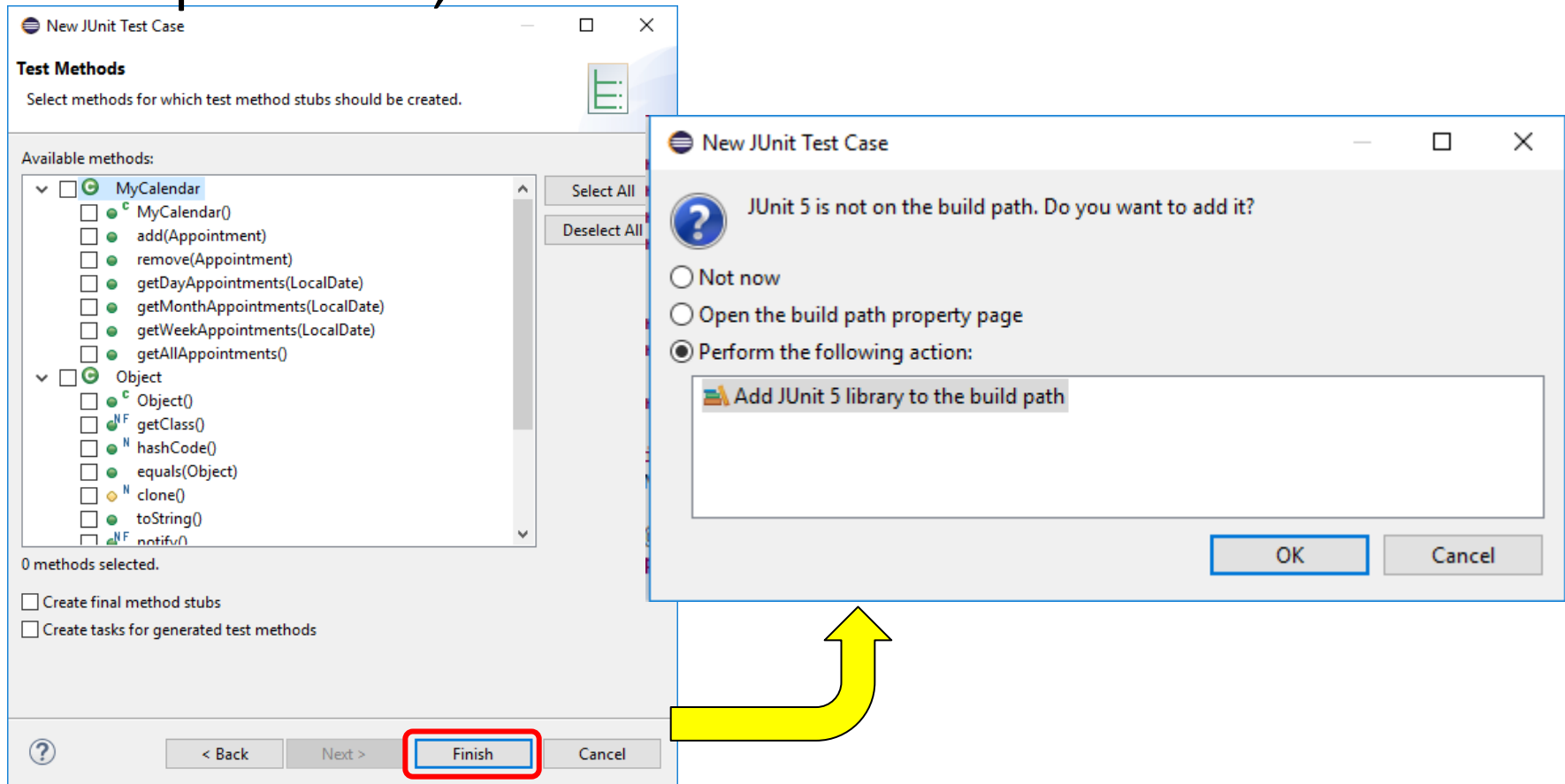


```
1 package fondt2.lab07.mycal.tests;
2
3 import static org.junit.jupiter.api.Assertions.*;
4
5
6
7
8
9
10
11 class ProvaTest {
12
13     @BeforeAll
14     static void setUpBeforeClass() throws Exception {
15     }
16
17     @AfterAll
18     static void tearDownAfterClass() throws Exception {
19     }
20
21     @BeforeEach
22     void setUp() throws Exception {
23     }
24
25     @AfterEach
26     void tearDown() throws Exception {
27     }
28
29     @Test
30     void testMyCalendar() {
31         fail("Not yet implemented");
32     }
33
34     @Test
```

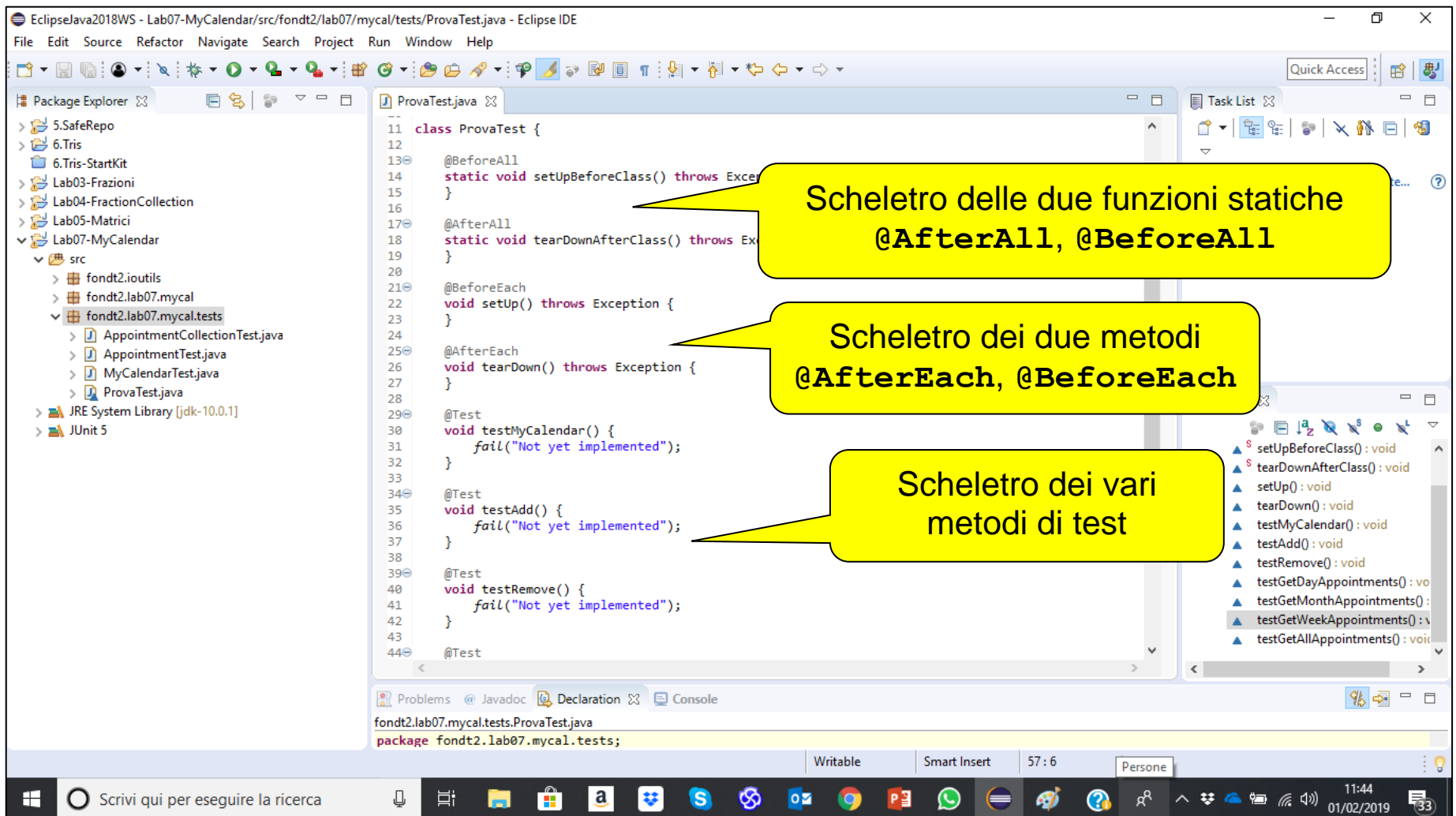
Viene creato lo scheletro dei test:  
un metodo di test per ogni metodo  
selezionato da collaudare

# Integrazione con Eclipse (4)

Se non già presente, viene chiesto di aggiungere JUnit al build path → sì, certo!



# Integrazione con Eclipse (5)



The screenshot shows the Eclipse IDE interface with the `ProvaTest.java` file open. The Package Explorer on the left shows the project structure, including the `src` folder and the `fondt2.lab07.mycal.tests` package. The main editor displays the code for `ProvaTest`, which includes static methods for setup and teardown, and several test methods. Three yellow callout boxes highlight specific parts of the code:

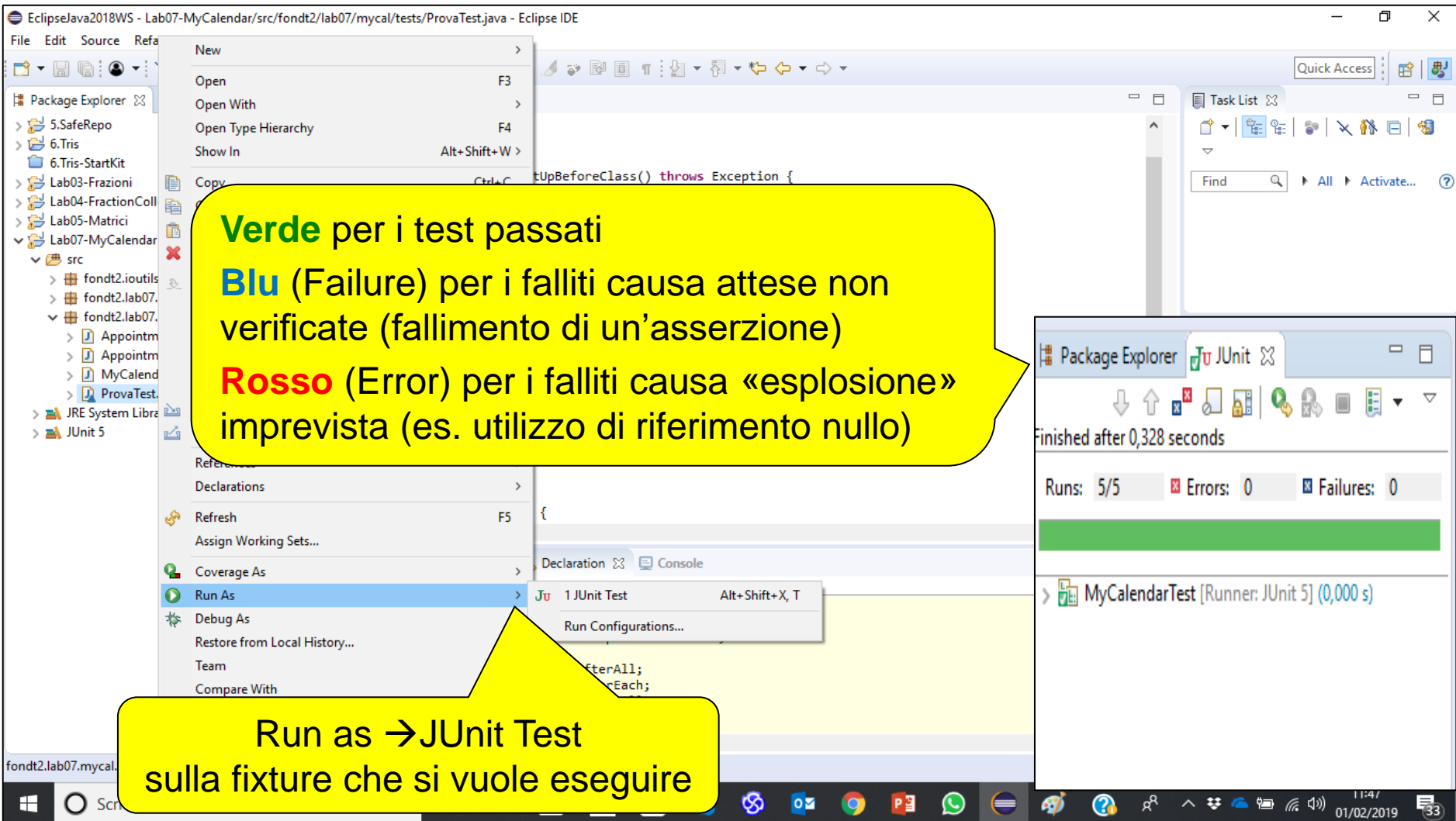
- Scheletro delle due funzioni statiche @AfterAll, @BeforeAll**: Points to the `setUpBeforeClass()` and `tearDownAfterClass()` methods.
- Scheletro dei due metodi @AfterEach, @BeforeEach**: Points to the `setUp()` and `tearDown()` methods.
- Scheletro dei vari metodi di test**: Points to the `testMyCalendar()`, `testAdd()`, `testRemove()`, and `testRemove()` methods.

The code in `ProvaTest.java` is as follows:

```
11 class ProvaTest {
12
13     @BeforeAll
14     static void setUpBeforeClass() throws Exception {
15     }
16
17     @AfterAll
18     static void tearDownAfterClass() throws Exception {
19     }
20
21     @BeforeEach
22     void setUp() throws Exception {
23     }
24
25     @AfterEach
26     void tearDown() throws Exception {
27     }
28
29     @Test
30     void testMyCalendar() {
31         fail("Not yet implemented");
32     }
33
34     @Test
35     void testAdd() {
36         fail("Not yet implemented");
37     }
38
39     @Test
40     void testRemove() {
41         fail("Not yet implemented");
42     }
43
44     @Test
```

The right-hand side of the IDE shows the Task List and the Declaration view, which lists the methods defined in the class, including `setUpBeforeClass()`, `tearDownAfterClass()`, `setUp()`, `tearDown()`, `testMyCalendar()`, `testAdd()`, `testRemove()`, `testGetDayAppointments()`, `testGetMonthAppointments()`, `testGetWeekAppointments()`, and `testGetAllAppointments()`.

# Esecuzione



**Verde** per i test passati

**Blu** (Failure) per i falliti causa attese non verificate (fallimento di un'asserzione)

**Rosso** (Error) per i falliti causa «esplosione» imprevista (es. utilizzo di riferimento nullo)

Run as → JUnit Test  
sulla fixture che si vuole eseguire

Finished after 0,328 seconds

Runs: 5/5   Errors: 0   Failures: 0

MyCalendarTest [Runner: JUnit 5] (0,000 s)

# Esecuzione

- Se i test falliscono non è sufficiente «prenderne atto»
- **Occorre capire *perché* falliscono**
- JUnit dice esattamente cosa sia accaduto:
  - **BLU**: asserzione fallita → indica quale e perché
  - **ROSSO**: errore ("esplosione") nell'esecuzione del test  
→ riporta il messaggio d'errore e lo *stack trace*  
(elenco dei record di attivazione)

# JUnit & Eclipse

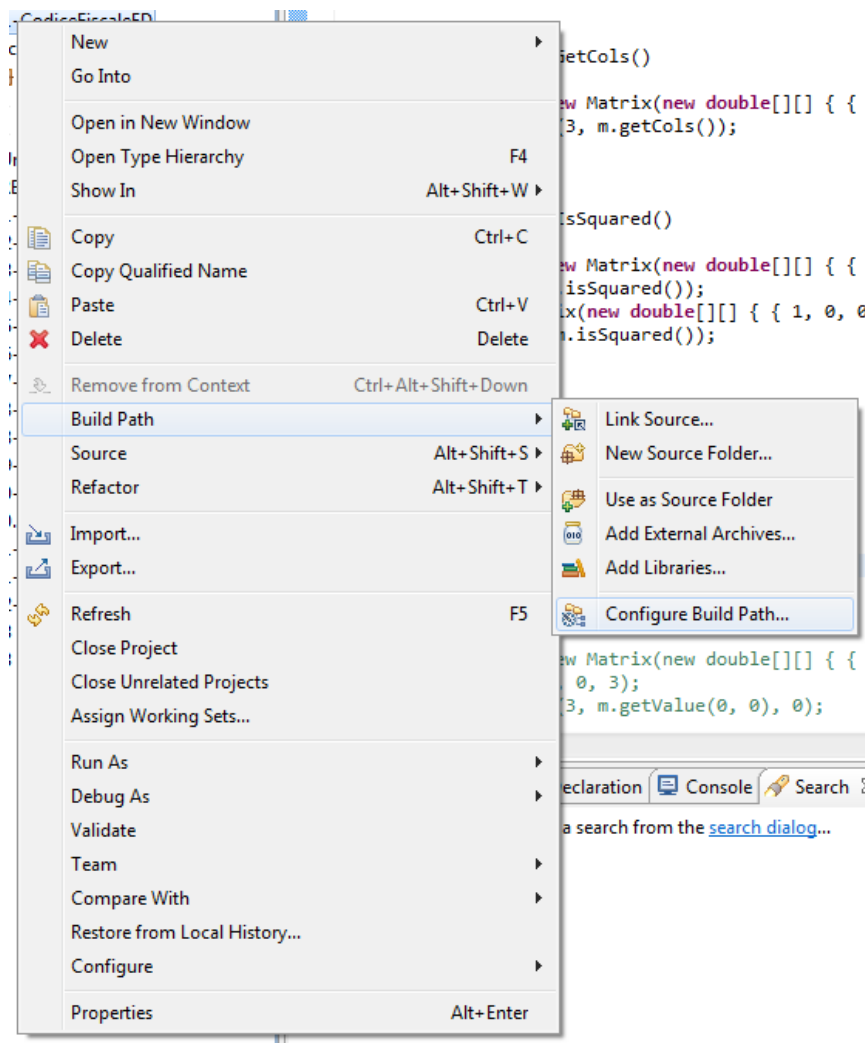
- Tecnicamente, JUnit è una normale libreria (JAR)  
→ il corrispondente JAR dev'essere nel progetto
- Come abbiamo visto, di norma creando un nuovo *JUnit Test case*, il JAR di JUnit viene norma aggiunto al build path automaticamente
- Tuttavia, qualche volta l'automatismo non scatta  
→ occorre aggiungere JUnit al progetto *manualmente*



# Aggiungere JUnit al progetto (1)

## PROCEDURA

1. Selezionare il progetto col tasto destro
2. Selezionare ora la voce *Build Path*
3. Selezionare quindi il menù *Configure Build Path...*



# Aggiungere JUnit al progetto (2)

**Nella finestra che appare:**

1. Andare nella scheda *Libraries*
2. Premere il pulsante *Add Library*
3. Scegliere *JUnit 5*

