JUnit Mockito :

In Java, while building an app, we will have **multiple classes** and methods

Imagine a method is not working after deploying the app 💀

That's why, we need to test first
We test each methods, those are units
So JUnit is used to test 👍

Unit Test → Smallest testable part of an application (method/class).

1. Create testcase
2. Test it, it would fail 😭
3. Because no methods, direct test case
4. Then fix it 😎

Basically, create new problems and find solutions 😂

Example :

We have add(a, b) method in Calculator

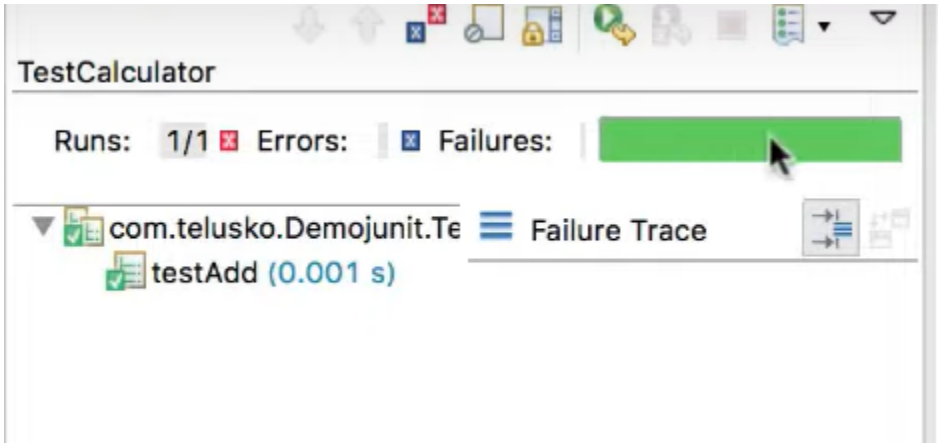
We need to create a Junit test java class, to test all the methods we wrote

Simply we create **own testcases,**
Like for this input = this output 👍

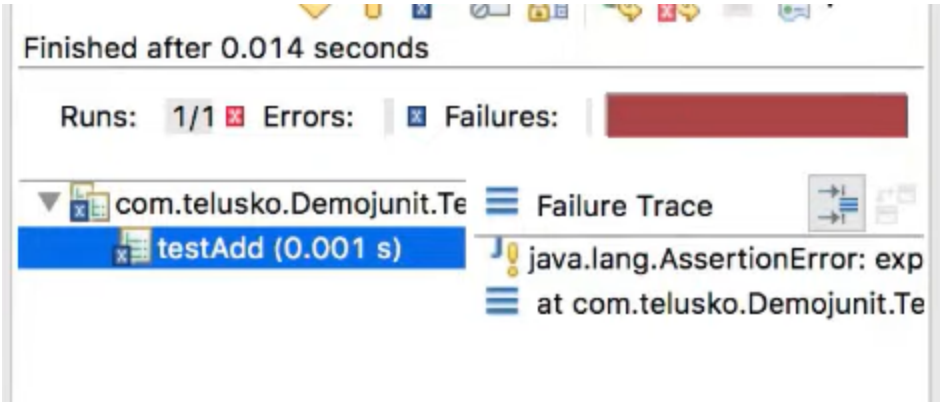
```java
public class TestCalculator
{
    Calculator c = new Calculator();

    @Test
    public void testAdd()
    {
        assertEquals(5,c.add(2, 3));
    }
}
```

Before the test method, we need to mention **@ Test**

And inside, **assertEquals(expected, actual);**



Green = passed 🔥



If test case failed, it shows red

Red = failure 😞

Basic Annotations :

`@Test`: Marks a method as a test case.

`@BeforeEach`: Runs a setup method **before every** test. Good for resetting variables.

`@AfterEach`: Runs a cleanup method **after every** test.

`@BeforeAll`: Runs **once before all** tests in a class.

`@AfterAll`: Runs **once after all** tests in a class.

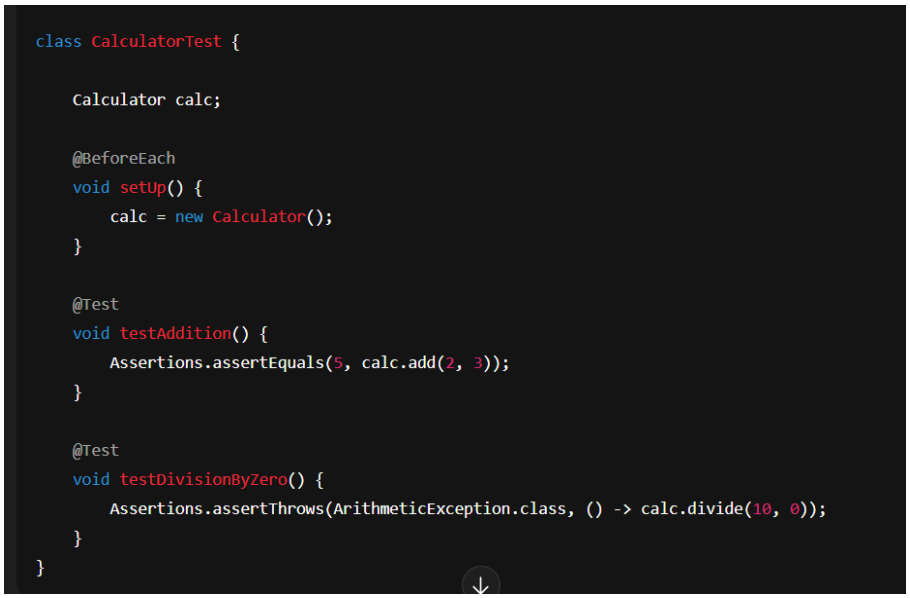
Assertions :

`assertEquals(expected, actual)`: Checks if two values are the same.

`assertTrue(condition)`: Checks if a condition is true.

`assertNotNull(object)`: Checks if an object isn't null.

Some Examples :



```
class CalculatorTest {

    Calculator calc;

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

    @Test
    void testAddition() {
        Assertions.assertEquals(5, calc.add(2, 3));
    }

    @Test
    void testDivisionByZero() {
        Assertions.assertThrows(ArithmeticException.class, () -> calc.divide(10, 0));
    }
}
```

We can also send **values as a parameter** 😱

Like we wrote palindrome method, we need string inputs

```java
@ParameterizedTest
@ValueSource(strings = {"racecar", "madam", "level"})
void testPalindrome(String word) {
    assertTrue(isPalindrome(word));
}
```

Mockito :

A mocking framework → creates fake objects to test code without real DB/API calls.

Mockito creates **mock objects**. These are fake versions of real objects.

Why???? 😭

So you can test your code without relying on complex, slow, or external dependencies
(like a database or an API call).

Example :

Imagine you are testing a class that class another class,

That another class gets data from database 😮

Then it would take 10 minutes to test 😔
Solution ? Mockito 😀

-> KEY ANNOTATIONS AND METHODS:

- **`@Mock`:** Creates a mock object. You put this on a field.

- **`@InjectMocks`:** Injects the mocks you created into the class you want to test. This is **clutch**.

- **`when(mockObject.method()).thenReturn(value)`:** This is the **MAGIC !!!**.

You're telling the mock object,

Yooo, when someone calls this method, just return this specific value.

Key Mockito Methods

java

```
when(mock.method()).thenReturn(value); // mock return value
verify(mock).method();                 // check method was called
doNothing().when(mock).voidMethod();   // mock void method
```

Key trap : void methods must use **doNothing()** method, not when()

Example with JUnit with Mockito :

```
@SpringBootTest
class ExpenseServiceMockitoTest {

    @Mock
    ExpenseRepository repo;

    @InjectMocks
    ExpenseService service;

    @Test
    void testSaveExpense() {
        Expense exp = new Expense("Dinner", 300);
        when(repo.save(exp)).thenReturn(exp);

        Expense saved = service.addExpense(exp);

        assertEquals("Dinner", saved.getName());
        verify(repo, times(1)).save(exp);
    }
}
```

Here, **repo is a dummy one** (mock) 👍

Think repo in spring boot, JPA Repository
findAll()
save()
And many methods??? 🫠

Those methods can be tested here using mock,

Here
Service injects mock, so saves in repo
Gets from service

Then we check using assertions 🔥👍