**✅ 1. What is JUnit? Why is it used?**

**Answer:**  
JUnit is a **Java unit testing framework** used to write and run repeatable automated tests. It helps in **test-driven development (TDD)** and ensures the correctness of code by testing individual units (methods).

**✅ 2. Write a simple test case using JUnit 5**

**Code under test:**

public class Calculator {

public int add(int a, int b) {

return a + b;

}

}

**Test case:**

import org.junit.jupiter.api.Test;

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

public class CalculatorTest {

@Test

void testAdd() {

Calculator calculator = new Calculator();

assertEquals(5, calculator.add(2, 3));

}

}

**✅ 3. How to test an exception in JUnit?**

public class AgeValidator {

public void validate(int age) {

if (age < 0) {

throw new IllegalArgumentException("Invalid age");

}

}

}

**Test case**

import org.junit.jupiter.api.Test;

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

public class AgeValidatorTest {

@Test

void testValidateNegativeAge() {

AgeValidator validator = new AgeValidator();

assertThrows(IllegalArgumentException.class, () -> validator.validate(-1));

}

}

**✅ 4. What is @BeforeEach and @AfterEach?**

**Answer:**

* @BeforeEach: Runs before each test method.
* @AfterEach: Runs after each test method.

**Example:**

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

public class SampleTest {

@BeforeEach

void setUp() {

System.out.println("Before Each Test");

}

@AfterEach

void tearDown() {

System.out.println("After Each Test");

}

@Test

void testExample() {

System.out.println("Test is running...");

}

}

**✅ 5. How do you write a parameterized test in JUnit 5?**

**Example:**

import org.junit.jupiter.params.ParameterizedTest;

import org.junit.jupiter.params.provider.ValueSource;

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

public class ParamTest {

@ParameterizedTest

@ValueSource(strings = {"madam", "racecar", "level"})

void testIsPalindrome(String input) {

assertTrue(new StringBuilder(input).reverse().toString().equals(input));

}

}

**✅ 6. How do you mock a dependency using Mockito in JUnit?**

We use **Mockito** in **JUnit** tests to create **mock objects** and simulate the behavior of complex dependencies like databases, services, or APIs — so we can test a class **in isolation**.

**Code to test:**

public class UserService {

private UserRepository repo;

public UserService(UserRepository repo) {

this.repo = repo;

}

public String getUserName(int id) {

return repo.findById(id);

}

}

interface UserRepository {

String findById(int id);

}

**Test case with mock:**

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

public class UserServiceTest {

@Test

void testGetUserName() {

UserRepository mockRepo = mock(UserRepository.class);

when(mockRepo.findById(1)).thenReturn("Vishal");

UserService service = new UserService(mockRepo);

assertEquals("Vishal", service.getUserName(1));

}

}

**✅ 7. What is the difference between assertEquals and assertSame?**

* assertEquals(a, b) → checks **value equality** (e.g., contents).
* assertSame(a, b) → checks if **both refer to same object**.

**✅ 8. How to disable a test temporarily?**

import org.junit.jupiter.api.Disabled;

import org.junit.jupiter.api.Test;

public class DisabledTest {

@Disabled("Not ready yet")

@Test

void testWillBeSkipped() {

// will not run

}

}

**✅ 9. Can you test private methods using JUnit?**

**Answer:**  
JUnit cannot test private methods directly. It's recommended to test via public methods. However, if needed, you can use **reflection**, though it's not a best practice.

**✅ 10. JUnit Annotations Summary**

| **Annotation** | **Description** |
| --- | --- |
| @Test | Marks a test method |
| @BeforeEach | Runs before each test |
| @AfterEach | Runs after each test |
| @BeforeAll | Runs once before all tests |
| @AfterAll | Runs once after all tests |
| @Disabled | Disables the test |
| @ParameterizedTest | Runs test multiple times with different input |