✅ JUnit & Mockito (Java Testing Frameworks)

🔹 What is JUnit?

JUnit is the most widely used **unit testing framework** for Java.
It allows you to write tests for **individual classes and methods** to ensure they behave as expected.

🔹 What is Mockito?

Mockito is a **mocking framework** for Java used **with JUnit**.
It allows you to create **mock (fake)** versions of dependencies (e.g., DAO, services) so you can test only the logic you want.

🧱 Why Testing is Important?

Reason	Why It Matters
Bug-Free Code	Catch issues early
Code Confidence	Know changes don't break old features
Cleaner Design	Encourages better architecture
Automation	Tests run with build (CI/CD)

🔧 JUnit: Basic Concepts

Concept	Explanation
@Test	Marks a method as a test case
@BeforeEach	Runs before each test
@AfterEach	Runs after each test
@BeforeAll	Runs once before all tests
@AfterAll	Runs once after all tests
Assertions	Used to check output (e.g., assertEquals())

📘 Simple JUnit Example

java

CopyEdit

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

public class CalculatorTest {

  @Test
  public void testAdd() {
    Calculator calc = new Calculator();
    int result = calc.add(2, 3);
    assertEquals(5, result);
  }
}
```

🔧 Mockito: Basic Concepts

Concept	Explanation
@Mock	Creates a mock object
@InjectMocks	Injects mock objects into your class
when(...).thenReturn(...)	Mocks a method return
verify()	Verifies if a method was called

📘 Mockito Example with JUnit

java

CopyEdit

```
@RunWith(MockitoJUnitRunner.class)
public class StudentServiceTest {

  @Mock
  private StudentRepository studentRepository;

  @InjectMocks
  private StudentService studentService;

  @Test
  public void testGetStudentById() {
    Student mockStudent = new Student(1, "Amit");

    when(studentRepository.findById(1)).thenReturn(Optional.of(mockStudent));

    Student result = studentService.getStudentById(1);
    assertEquals("Amit", result.getName());
  }
}
```

🧠 **Real-Life Example**

Imagine you have a Spring Boot app with a StudentService that depends on StudentRepository.
Instead of hitting the **real DB**, Mockito lets you **mock** that repository and just test the service logic.

✅ FASTER
✅ NO DB REQUIRED
✅ UNIT TEST ONLY THE BUSINESS LOGIC

✅ Benefits

JUnit	Mockito
Standard Java testing tool	Easy mocking of dependencies
Integrates with Maven, Gradle, IDEs	Removes DB, API dependencies from tests
Supports annotations and lifecycle methods	Tests only the method under test
Works with TestNG, Spring Boot	Can simulate success/failure scenarios easily

---

❌ Drawbacks

JUnit	Mockito
Limited in mocking capabilities	Doesn't test real integration
Can be verbose with setup	Misuse can lead to false positives
Only for Java	Not useful for full-stack tests

🔁 Common Interview Questions

🔸 JUnit

1. What is JUnit?
2. How do you write a test case in JUnit?
3. What is the difference between @BeforeEach and @BeforeAll?
4. How do you assert that an exception is thrown?

🔸 Mockito

1. What is Mockito used for?
2. What is the use of @Mock and @InjectMocks?
3. How do you mock a method call using Mockito?
4. What is the difference between mock() and spy()?
5. How do you verify method calls?

🧱 Maven Dependencies

xml

CopyEdit

```
<!-- JUnit 5 -->
<dependency>
 <groupId>org.junit.jupiter</groupId>
 <artifactId>junit-jupiter</artifactId>
 <version>5.10.0</version>
 <scope>test</scope>
</dependency>

<!-- Mockito -->
<dependency>
 <groupId>org.mockito</groupId>
 <artifactId>mockito-core</artifactId>
 <version>5.11.0</version>
 <scope>test</scope>
</dependency>
```

🧪 BONUS: How to Run Tests?

- In IntelliJ or Eclipse → Right click on test class → "Run"
- In Maven:

bash

CopyEdit

```
mvn test
```

- In Gradle:

bash

CopyEdit

gradle test