🧩 UNIT TESTING – Interview Q&A

1 What is Unit Testing?

Answer:
Unit testing is the process of testing individual units or components of code—such as methods or classes—to ensure that each one functions as expected.
It's usually written by developers during development using frameworks like JUnit.

Example:
Testing a `calculateDiscount()` method in isolation without running the full application.

---

2 Why is Unit Testing important?

Answer:

- Detects bugs early in development
- Makes code easier to refactor
- Improves code quality and reliability
- Helps achieve continuous integration (CI/CD)
- Acts as documentation for code behavior

---

3 What are the characteristics of a good unit test?

Answer:
A good unit test should be:

- **Fast** – runs quickly
- **Isolated** – doesn't depend on external systems (like DBs)
- **Repeatable** – same result every time
- **Self-checking** – uses assertions to verify results
- **Readable & Maintainable** – easy to understand and update

---

4 What is the difference between Unit, Integration, and System Testing?

Type	Scope	Purpose
Unit Testing	Tests single function/class	Verify correctness of individual components
Integration Testing	Tests combined modules	Ensure components work together

Type	Scope	Purpose
System Testing	Tests entire system	Validate complete functionality end-to-end

---

⚙️ JUNIT – Interview Q&A

5 What is JUnit?

Answer:
JUnit is a popular open-source testing framework in Java used to write and run repeatable unit tests. It provides annotations, assertions, and lifecycle methods for testing Java code efficiently.

---

6 What are some important JUnit annotations?

Answer:

Annotation	Purpose
`@Test`	Marks a method as a test case
`@BeforeEach`	Runs before each test method
`@AfterEach`	Runs after each test method
`@BeforeAll`	Runs once before all test methods (static)
`@AfterAll`	Runs once after all test methods (static)
`@Disabled`	Skips a test
`@DisplayName`	Gives a readable name to a test case

---

7 What are Assertions in JUnit?

Answer:
Assertions are methods used to verify the expected result of a test.
Common assertions:

```
assertEquals(expected, actual);
assertTrue(condition);
assertFalse(condition);
assertNotNull(object);
assertThrows(Exception.class, () -> method());
```

---

8 How do you group or run multiple tests together in JUnit?

Answer:
JUnit 5 supports **test suites** using `@Suite` annotation:

```
@Suite
@SelectClasses({ CalculatorTest.class, UserServiceTest.class })
public class AllTests { }
```

9️⃣ What is the difference between JUnit 4 and JUnit 5?

Feature	JUnit 4	JUnit 5
Annotation prefix	`org.junit`	`org.junit.jupiter`
Lifecycle methods	`@Before`, `@After`	`@BeforeEach`, `@AfterEach`
Dependency	Monolithic JAR	Modular (Jupiter, Platform, Vintage)
Parameterized tests	Limited	Built-in support

🔟 Example JUnit Test

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

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

🧠 MOCKITO – Interview Q&A

1️⃣1️⃣ What is Mockito?

Answer:
Mockito is a Java mocking framework used for unit testing. It lets you **mock dependencies** so you can test classes in isolation without using actual implementations like databases or APIs.

1️⃣2️⃣ What is Mocking?

Answer:
Mocking means creating a **dummy object** that simulates the behavior of real dependencies. This allows testing a class independently of its external components.

1️⃣3️⃣ How do you create a mock object using Mockito?

Answer:

```
@Mock
UserRepository userRepo;

@InjectMocks
UserService userService;
```

or

```
UserRepository mockRepo = Mockito.mock(UserRepository.class);
```

14 What are common Mockito methods?

Method	Description
`when(...).thenReturn(...)`	Defines mock behavior
`verify(mock).method()`	Verifies if method was called
`any()`	Argument matcher
`times(n)`	Ensures method was called *n* times
`doThrow()`, `doNothing()`, `doReturn()`	Define behaviors for void methods

15 Example Mockito Test

```
import static org.mockito.Mockito.*;
import org.junit.jupiter.api.Test;
import org.mockito.InjectMocks;
import org.mockito.Mock;
import org.mockito.MockitoAnnotations;

class UserServiceTest {

    @Mock
    private UserRepository userRepo;

    @InjectMocks
    private UserService userService;

    public UserServiceTest() {
        MockitoAnnotations.openMocks(this);
    }

    @Test
    void testGetUserById() {
        User mockUser = new User(1, "John");
        when(userRepo.findById(1)).thenReturn(mockUser);

        User result = userService.getUserById(1);

        assertEquals("John", result.getName());
        verify(userRepo, times(1)).findById(1);
    }
}
```

16 What is the difference between @Mock and @InjectMocks?

Annotation	Description
`@Mock`	Creates a mock object of a dependency
`@InjectMocks`	Injects mock objects into the tested class automatically

17 Why use Mockito instead of writing fake classes manually?

Answer:
Mockito reduces boilerplate, automatically handles mock creation, supports verifications, and integrates smoothly with JUnit — making tests faster, cleaner, and more maintainable.

---

18 What is Stubbing in Mockito?

Answer:
Stubbing means defining what a mock should return when a method is called.
Example:

```
when(userRepo.findById(1)).thenReturn(new User(1, "John"));
```

---

19 What is Verification in Mockito?

Answer:
Verification checks whether certain methods were called during test execution.

```
verify(userRepo, times(1)).findById(1);
```

---

20 How do you mock void methods in Mockito?

Answer:

```
doNothing().when(mockObject).methodName();
doThrow(new Exception()).when(mockObject).methodName();
```