 1. What is unit testing? Why is it important?

Answer:
Unit testing verifies that individual units (like methods or classes) work as expected in isolation. It ensures early detection of bugs, simplifies debugging, improves code quality, and makes refactoring safer.

---

 2. What are the characteristics of a good unit test?

Answer:
A good unit test should be:

Fast (quick execution)

Isolated (independent of external systems like DB)

Repeatable (gives same result every time)

Readable (easy to understand)

Specific (tests one logical unit at a time)

---

 3. Difference: Unit vs Integration vs Functional Testing?

Answer:

Unit Testing: Test individual methods/classes.

Integration Testing: Test how multiple components work together.

Functional Testing: Test the complete business flow from end-to-end.

---

 4. How do you write a simple unit test using JUnit?

java

CopyEdit

@Test

void testSum() {

    Calculator calc = new Calculator();

```
    int result = calc.sum(2, 3);
    assertEquals(5, result);
}
```

---

 5. Explain JUnit lifecycle annotations?

Answer:

@BeforeEach: Runs before each test method.

@AfterEach: Runs after each test method.

@BeforeAll: Runs once before all tests (static).

@AfterAll: Runs once after all tests (static).

---

 6. Difference: @BeforeEach vs @BeforeAll?

Answer:

@BeforeEach: Runs before every test.

@BeforeAll: Runs once before any test (must be static).

---

 7. How do you test exception handling in JUnit 5?

java

CopyEdit

```
@Test
void testException() {
    assertThrows(IllegalArgumentException.class, () -> {
        service.doSomething(null);
    });
}
```

---

 8. What are parameterized tests in JUnit 5?

java

CopyEdit

@ParameterizedTest

@ValueSource(strings = {"test", "TEST", "TeSt"})

void testToLowerCase(String input) {

    assertEquals("test", input.toLowerCase());

}

---

 9. What is mocking? Why Mockito?

Answer:
Mocking is simulating real dependencies using fake objects. Mockito allows us to isolate the unit under test without invoking real services, DBs, or APIs.

---

 10. How do you mock a dependency in a service?

java

CopyEdit

@Mock

UserRepository userRepository;

@InjectMocks

UserService userService;

---

 11. Explain: @Mock, @InjectMocks, @Spy?

@Mock: Creates a mock instance of a class.

@InjectMocks: Injects mocks into the actual class being tested.

@Spy: Creates a partial mock (real methods unless stubbed).

---

 12. Use of when(...).thenReturn(...)?

java

CopyEdit

when(repo.findById(1)).thenReturn(Optional.of(user));

---

 13. Difference: @Mock vs @Spy?

Answer:

@Mock: Everything is mocked.

@Spy: Real object, unless method is stubbed.

---

 14. How to verify method calls in Mockito?

java

CopyEdit

verify(service).sendEmail(anyString());

---

 15. How to simulate exceptions?

java

CopyEdit

when(service.callAPI()).thenThrow(new RuntimeException("Error"));

---

 16. Unit test for Spring @Service class?

java

CopyEdit

```
@Test
void testGetUser() {
   when(userRepository.findById(1)).thenReturn(Optional.of(user));
   User result = userService.getUser(1);
   assertEquals("John", result.getName());
}
```

---

 17. How to test Spring REST controllers?

Answer:
Use @WebMvcTest and MockMvc.

---

 18. How to test REST endpoints without starting server?

Answer:
Use MockMvc with @WebMvcTest:

java

CopyEdit

```
@Autowired
private MockMvc mockMvc;

@Test
void testGetUser() throws Exception {
   mockMvc.perform(get("/users/1"))
          .andExpect(status().isOk());
}
```

---

 19. What is MockMvc?

Answer:
MockMvc is used to test Spring MVC controllers by simulating HTTP requests/responses without deploying the app.

---

 20. How to mock repositories?

java

CopyEdit

@Mock

private UserRepository userRepository;

---

 21. Unit test for a utility or @Component class?

java

CopyEdit

@Test

void testUtilityMethod() {

    assertEquals("Hello", Utility.greet("Hello"));

}

---

 22. How to test classes with @Autowired dependencies?

Answer:
Use @InjectMocks to inject mocks into the class under test.

---

 23. Unit test a method using RestTemplate?

java

CopyEdit

```
@Mock
RestTemplate restTemplate;

@Test
void testApiCall() {
   when(restTemplate.getForObject(anyString(), eq(String.class)))
       .thenReturn("response");
}
```

---

 24. How to mock static methods or final classes?

Answer:
Use Mockito’s inline mocking or PowerMock:

```
java
CopyEdit
try (MockedStatic<UtilClass> mock = mockStatic(UtilClass.class)) {
   mock.when(() -> UtilClass.staticMethod()).thenReturn("Mocked");
}
```

---

 25. How to test classes using env variables or config?

Answer:
Use @TestPropertySource or mock Environment:

```
java
CopyEdit
@Mock
Environment env;
```

when(env.getProperty("app.mode")).thenReturn("dev");

---

 26. Common mistakes in unit testing?

Answer:

Testing multiple units in one test

Not isolating external dependencies

Not cleaning up test data

Ignoring edge cases

---

 27. Should we test private methods?

Answer:
Generally no. Private logic should be tested indirectly via public methods.

---

 28. What is code coverage? Ideal %?

Answer:
It shows how much code is tested. 70–80% is good, but focus on quality not just quantity.

---

 29. How unit testing helps with legacy code?

Answer:
Adding tests ensures that refactoring or adding features doesn’t break existing behavior.

---

 30. Tools for coverage like Jacoco or SonarQube?

Answer:

Jacoco plugin for Maven/Gradle generates coverage reports.

SonarQube analyzes quality and coverage as part of CI/CD.

---

 31. Unit test for discount logic?

java

CopyEdit

```
@Test
void testDiscount() {
   double result = service.calculateDiscount("PREMIUM", 500);
   assertEquals(50, result); // 10% discount
}
```

---

 32. Unit test for controller returning list?

java

CopyEdit

```
mockMvc.perform(get("/users"))
      .andExpect(status().isOk())
      .andExpect(jsonPath("$.length()").value(2));
```

---

 33. Test method with DB save and email send?

java

CopyEdit

```
@Test
void testSaveOnly() {
   when(repo.save(any(User.class))).thenReturn(user);
   service.registerUser(user);
   verify(repo).save(user);
   verify(emailService, never()).sendEmail(anyString());
```

}

---

 34. Test external API with RestTemplate?

java

CopyEdit

```
@Test
void testExternalCall() {
    when(restTemplate.getForObject("https://api.com", String.class))
        .thenReturn("success");
    String result = service.callApi();
    assertEquals("success", result);
}
```