TOP 20 INTERVIEW QUESTIONS ON UNIT TESTING

1. What is unit testing?

- Unit testing involves testing individual units or components of an application to ensure they work as expected.

2. What is Spring Boot's `@SpringBootTest` annotation?

- `@SpringBootTest` is used to bootstrap the entire Spring application context for integration tests, but it can also be overkill for unit tests, which typically don't require the full context.

3. What is the difference between unit tests and integration tests?

- Unit tests test individual components in isolation, while integration tests check how different components work together in a Spring application.

4. How do you mock dependencies in unit tests in Spring?

- Dependencies are typically mocked using mocking frameworks like Mockito or Spring's `@MockBean` and `@Mock` annotations.

5. What is the purpose of the `@MockBean` annotation in Spring?

- `@MockBean` is used to add a mock instance of a bean to the Spring application context, replacing any existing bean of the same type for testing purposes.

6. What is `@Mock` in Mockito?

- `@Mock` is used to create mock objects in a unit test that simulates the behavior of real objects.

7. What is the use of `@InjectMocks` in Mockito?

- `@InjectMocks` injects the mocks created with `@Mock` or `@MockBean` into the class you want to test, simulating dependency injection.

8. How does the `@BeforeEach` annotation work in unit tests?

- `@BeforeEach` is used to specify that a method should be executed before each test method, often used for setting up test data or mocks.

9. What is the difference between `@BeforeEach` and `@BeforeAll`?

- `@BeforeEach` runs before each test method, while `@BeforeAll` runs once before any test methods in the test class.

10. How do you test Spring repositories using unit tests?

- Spring repositories can be tested by mocking the repository interface using `@MockBean` and simulating the behavior with Mockito.

11. What is `Mockito.when()` used for?

- `Mockito.when()` is used to define the behavior of a mocked object when a specific method is called on it.

12. How do you verify method invocations using Mockito?

- You can use `Mockito.verify()` to check whether a particular method was called on a mock object during the test.

13. What is the purpose of `@Captor` in Mockito?

- `@Captor` is used to capture argument values passed to a method during test execution, allowing for detailed assertions on the arguments.

14. What is the role of `@WebMvcTest` in Spring?

- `@WebMvcTest` is used to test Spring MVC controllers, loading only the web layer (without full application context), which allows testing of controllers in isolation.

15. How do you test Spring services?

- Spring services are typically tested using mocks for their dependencies, along with the `@InjectMocks` annotation, to ensure the service logic behaves as expected.

16. How can you test exceptions in Spring services?

- You can use `Mockito.when()` to throw exceptions from mock objects and use JUnit's `assertThrows()` to verify that the expected exception is thrown.

17. How do you test Spring MVC controllers with MockMvc?

- MockMvc allows testing of Spring MVC controllers by performing HTTP requests and checking the status, headers, and response body without starting a full server.

18. How does the `@TestConfiguration` annotation help in unit testing?

- `@TestConfiguration` allows you to define custom configurations for your tests, isolating the test context from the main application context.

19. What is `ArgumentCaptor` in Mockito?

- `ArgumentCaptor` is used to capture arguments passed to methods on mock objects, allowing you to assert the correctness of the method's input.

20. How do you handle database interactions in unit tests?

- For unit tests, it’s common to mock database interactions using tools like Mockito or to use in-memory databases like H2 for testing purposes, along with libraries like `@DataJpaTest` for repository testing.