Junit Interview Questions

1) What is JUnit and why is it used in Java development?

- → Junit is a widely-used unit testing framework for Java that allows developers to write and run repeatable automated tests.
- → It is essential for ensuring code quality and reliability by detecting bugs early in the development process.

2) How do you create a simple test case in Junit?

→ "To create a simple test case in JUnit, you need to define a method annotated with @Test. Inside this method, use assertions like assertEquals to verify the expected outcomes."

3) What are annotations in JUnit? Name a few commonly used ones.

- → Annotations in JUnit are metadata that provide information about the code and are used to configure and control test execution.
- → Commonly used annotations include @Test, @Before, and @After."

4) How do you run tests in JUnit? Describe the different ways to execute tests.

- → To run tests in JUnit, you can execute them directly from an IDE like IntelliJ or Eclipse, which provides built-in support for running JUnit tests.
- → Alternatively, you can use build tools like Maven or Gradle to automate test execution as part of your build process."
- → There is an option of running tests from the command line using the JUnit platform console launcher.

5) Write a JUnit test case that tests a method throwing an exception.

→ "To write a JUnit test case that tests a method throwing an exception, you can use the @Test(expected = Exception.class) annotation. This ensures that the test will pass only if the specified exception is thrown."

```
I. assertThrows() (JUnit 5 - most recommended)

Best practice - clean and expressive.

java

@Test
void testExceptionUsingAssertThrows() {
    Exception exception = assertThrows(
        IllegalArgumentException.class,
        () -> someMethodThatThrows(),
        "Expected IllegalArgumentException"
    );
    assertEquals("Expected message", exception.getMessage());
}
```

```
Java

@Test(expected = Exception.class) (JUnit 4 only)

Concise, but you can't assert on the message or content of the exception.

java

@Test(expected = IllegalArgumentException.class)
public void testExceptionAnnotation() {
    someMethodThatThrows();
}
```


5. Using AssertJ (fluent API for assertions)

More expressive and chainable, great for readability.

```
import static org.assertj.core.api.Assertions.*;

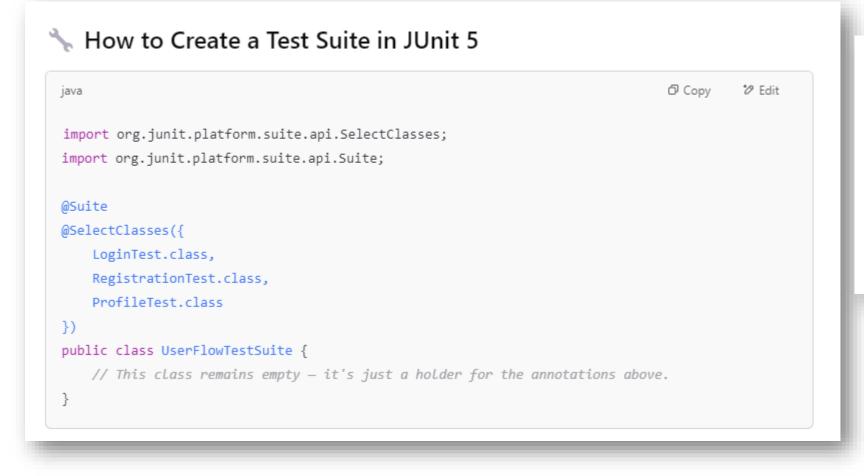
@Test
void testWithAssertJ() {
    assertThatThrownBy(() -> someMethodThatThrows())
        .isInstanceOf(IllegalArgumentException.class)
        .hasMessage("Expected message");
}
```

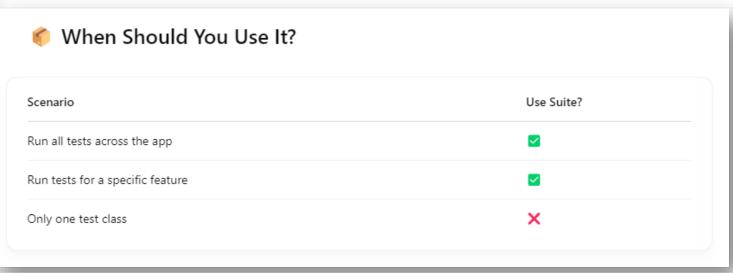
Ques) Explain the concept of test suites in JUnit.

- → A test suite is like a playlist
- → Instead of playing one song (test) at a time, you create a playlist of tests and run them all together!

Why Use a Test Suite?

- Run multiple test classes together
- Group related tests (e.g., all login-related tests)
- Save time & organize better
- Useful for integration testing



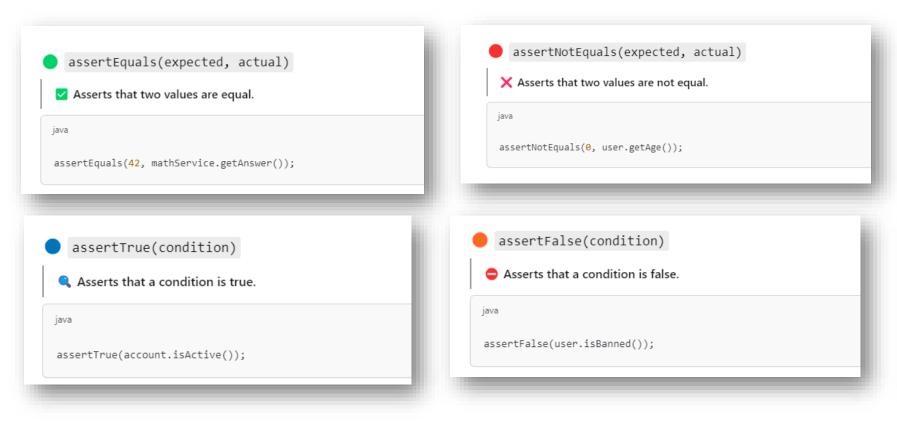


Ques) What are Assertions in JUnit?

- → Assertions are like checkpoints in your tests.
- → They verify that your code is working as expected.

If an assertion fails, the test fails — simple!

☑ Commonly Used JUnit Assertion Methods





Ques) Write a JUnit test case for a method that checks if a string is a palindrome.

```
public class StringUtils {

   public boolean isPalindrome(String input) {
      if (input == null) return false;
      String clean = input.replaceAll("[^a-zA-Z0-9]", "").toLowerCase();
      return new StringBuilder(clean).reverse().toString().equals(clean);
   }
}
```

```
@ExtendWith(MockitoExtension.class)
class PalindromeControllerTest {
   @InjectMocks
   PalindromeController controller;
   @Mock
   PalindromeService service;
   @Test
   void testPalindromeTrue() {
       Mockito.when(service.isPalindrome("madam")).thenReturn(true);
       var response = controller.checkPalindrome("madam");
       assertEquals(true, response.getBody());
   @Test
   void testPalindromeFalse() {
       Mockito.when(service.isPalindrome("hello")).thenReturn(false);
       var response = controller.checkPalindrome("hello");
       assertEquals(false, response.getBody());
```

Ques) What is the purpose of the @Before and @After annotations?

- → In JUnit, the @Before and @After annotations are used to define setup and teardown methods that run before and after each test method, respectively.
- → Runs before each test method.
- → @BeforeEach, Useful for setting up test data, initializing objects, or preparing test environment.
- → @AfterEach, Typically used for cleanup, like closing connections or resetting state.

```
gava
@BeforeEach
void setUp() {
    // Code here runs before every @Test
    service = new PalindromeService();
}
```

```
java

@AfterEach
void tearDown() {
    // Code here runs after every @Test
    service = null;
}
```

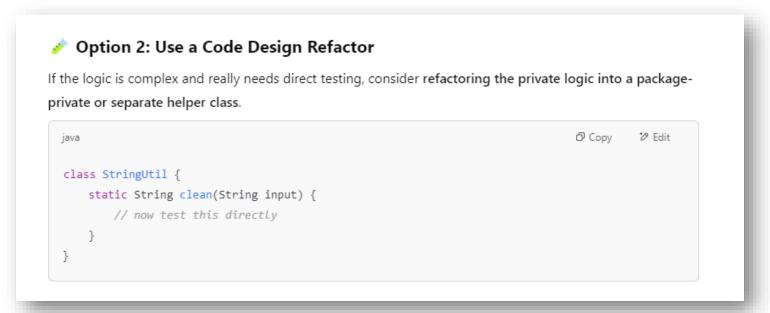
Ques) How do you test private methods in JUnit?

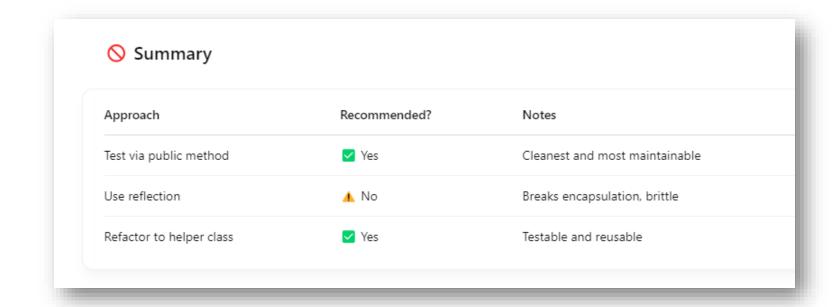
Testing private methods directly in JUnit is generally not recommended because:

→ Private methods are implementation details — they should be tested indirectly via public methods.









Ques) Write a JUnit test case that verifies the behavior of a method that interacts with a database.

```
✓ JUnit Test with H2 In-Memory DB (Recommended Way)
                                                                             ☐ Copy 况 Edit
 java
 @SpringBootTest
 @AutoConfigureTestDatabase(replace = AutoConfigureTestDatabase.Replace.ANY)
 @Transactional
 class UserRepositoryTest {
     @Autowired
     private UserRepository userRepository;
     @Test
     void testSaveAndFindUser() {
         User user = new User();
         user.setName("Alice");
         userRepository.save(user);
         Optional<User> found = userRepository.findByName("Alice");
         assertTrue(found.isPresent());
         assertEquals("Alice", found.get().getName());
```

Ques) How do you integrate JUnit with build tools like Maven or Gradle?

→Integrating JUnit with build tools like Maven or Gradle is straightforward — both tools have built-in support for JUnit.



```
3. Run Tests:

bash

mvn test
```

Ques) How do we test void methods

- **☑** 1. Use Mockito to Verify Behavior
 - → If the void method interacts with other components (e.g., calls a repo, logs, sends an email), you verify those interactions using **Mockito.verify()**.





Strategy	Use Case	
Mockito.verify()	Checks that the method did what it's supposed to	
ArgumentCaptor	Captures arguments passed to the void method	
doThrow()	Simulate exceptions from void methods	
Test side effects	Validate changes in state or object	