Using Mockito in Java Tests

To use Mockito in Java tests, follow these steps to create mock objects, stub methods, and verify interactions:

1. Add Dependencies

Include Mockito and JUnit in your `pom.xml` (Maven):

```
<dependency>
    <groupId>org.mockito</groupId>
    <artifactId>mockito-core</artifactId>
    <version>5.3.1</version>
    <scope>test</scope>
</dependency>
<dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter</artifactId>
    <version>5.9.2</version>
    <scope>test</scope>
</dependency>
```

2. Set Up the Test Class

Use JUnit 5 with Mockito's extension:

```
import org.junit.jupiter.api.extension.ExtendWith;
import org.mockito.junit.jupiter.MockitoExtension;

@ExtendWith(MockitoExtension.class)
public class UserServiceTest {
    // ...
}
```

3. Create Mocks and Inject Dependencies

- Use `@Mock` for dependencies you want to mock.
- Use `@InjectMocks` to inject mocks into the class under test.

```
@Mock
private UserRepository userRepository;
```

```java
@InjectMocks
private UserService userService; // UserService depends on UserRepository
```

4. Stub Method Calls

Use `when().thenReturn()` to define mock behavior:

```java
@Test
void getUserById_ReturnsUser_WhenUserExists() {
    User mockUser = new User(1L, "Alice");
    // Stub the repository's method
    when(userRepository.findById(1L)).thenReturn(mockUser);

    User result = userService.getUserById(1L);

    assertEquals("Alice", result.getName());
}
```

5. Verify Interactions

Check if a mock method was called as expected:

```java
@Test
void getUserById_CallsRepository() {
    userService.getUserById(1L);
    // Verify the repository method was called once with ID 1
    verify(userRepository, times(1)).findById(1L);
}
```

Common Use Cases

Stub Exceptions

```java
@Test
void getUserById_ThrowsException_WhenRepositoryFails() {
    when(userRepository.findById(anyLong())).thenThrow(new RuntimeException("DB Error"));

    assertThrows(RuntimeException.class, () -> userService.getUserById(1L));
}
```

Argument Matchers Use `any()`, `eq()`, etc., to match arguments flexibly:

```
when(userRepository.findById(anyLong())).thenReturn(new User(1L, "Bob"));
```

BDD (Behavior-Driven Development) Style Use `given().willReturn()` for readability:

```
import static org.mockito.BDDMockito.*;

@Test
void bddStyleExample() {
    given(userRepository.findById(1L)).willReturn(new User(1L, "Alice"));

    User result = userService.getUserById(1L);

    then(userRepository).should().findById(1L);
    assertEquals("Alice", result.getName());
}
```

Argument Captor Capture arguments to perform detailed assertions:

```
@Test
void saveUser_CapturesArgument() {
    ArgumentCaptor<User> userCaptor = ArgumentCaptor.forClass(User.class);

    userService.saveUser("Charlie");

    verify(userRepository).save(userCaptor.capture());
    assertEquals("Charlie", userCaptor.getValue().getName());
}
```

Key Notes

- **Initialization**: Use `@ExtendWith(MockitoExtension.class)` (JUnit 5) or `MockitoAnnotations.openMocks(this)` in `@BeforeEach`.
- **Static Mocks**: Use `mockito-inline` dependency for mocking static methods/constructors.
- **Spies**: Use `@Spy` to wrap real objects (partial mocks).

Troubleshooting

- **Null Mocks**: Ensure mocks are initialized (use `@ExtendWith` or `openMocks()`).
- **Stubbing Errors**: Stub methods before they're called in the test.

For advanced usage, refer to the Mockito Documentation.