**Mockito Backend Testing**

Mockito is a popular mocking framework for Java that is used for unit testing applications. It is designed to create and configure mock objects in a simple and intuitive way. Mockito allows developers to write clean, maintainable, and robust tests by providing a powerful API to simulate the behavior of dependencies in isolation.

**Getting Started with Mockito**

To get started with Mockito in a Maven or Gradle project, you need to add the Mockito dependency to your pom.xml or build.gradle file respectively.

**Maven:**

```
<dependency>
 <groupId>org.mockito</groupId>
 <artifactId>mockito-core</artifactId>
 <version>4.0.0</version>
 <scope>test</scope>
</dependency>
```

**Gradle:**

```
testImplementation 'org.mockito:mockito-core:4.0.0'
```

After adding the dependency, you can start creating mock objects using the mock() method and specify the behavior of mocks using the when() and thenReturn() methods.

**Some examples**

**1. Creating Mocks**

Function: mock(Class<T> classToMock)

Purpose: Creates a mock instance of the given class or interface.

Example:

```
List mockedList = mock(List.class);
```

**2. Stubbing Method Calls**
Function: when(T methodCall).thenReturn(T returnValue)

Purpose: Allows stubbing methods to return specific values when invoked.

Example:

```
// Stubbing a method call for the mocked list

when(mockedList.get(0)).thenReturn("First Element");

// Using the stubbed method

System.out.println(mockedList.get(0)); // Outputs: First Element

// For a method that is not stubbed, Mockito returns the default value (null, 0, false, etc.)

System.out.println(mockedList.get(99)); // Outputs: null
```

**3. Verifying Method Calls**
Function: verify(T mock).methodToVerify(arguments)

Purpose: Verifies that a specific method was called with the given arguments.

Example:

```
mockedList.add("One");
mockedList.clear();

// Verify that the add method was called with "One"
verify(mockedList).add("One");

// Verify that the clear method was called
verify(mockedList).clear();
```

**4. Argument Matchers**
Function: when(T methodCall).thenReturn(T returnValue)

Purpose: Allows using built-in argument matchers (like any(), eq(), etc.) to provide flexible interaction with the mock.

Example:

```
// Using an argument matcher to stub a method for any integer argument
when(mockedList.get(anyInt())).thenReturn("Element");

// Now, any call with an integer argument returns "Element"
System.out.println(mockedList.get(999)); // Outputs: Element
```

**5. Exception Throwing**
Function: when(T methodCall).thenThrow(Throwable...)

Purpose: Allows stubbing methods to throw exceptions when invoked.

Example:

```
// Stubbing to throw an exception
when(mockedList.get(anyInt())).thenThrow(new RuntimeException("Error"));

// This call throws the stubbed exception
try {
 mockedList.get(0);
} catch (RuntimeException e) {
 System.out.println(e.getMessage()); // Outputs: Error
}
```

**6. Verifying Number of Invocations**
Function: verify(T mock, VerificationMode mode).methodToVerify(arguments)

Purpose: Verifies that a method was called a specific number of times.

Example:

```
mockedList.add("Once");
mockedList.add("Twice");
mockedList.add("Twice");

// Verifying the number of invocations
verify(mockedList, times(1)).add("Once");
verify(mockedList, times(2)).add("Twice");
verify(mockedList, never()).add("Never happened");
```

**7. Argument Capturing**

Function: ArgumentCaptor<T>

Purpose: Captures arguments passed to methods for further assertions.

Example:

```
ArgumentCaptor<String> argumentCaptor = ArgumentCaptor.forClass(String.class);
mockedList.add("One");
verify(mockedList).add(argumentCaptor.capture());

assertEquals("One", argumentCaptor.getValue());
```

**8. Spying on Real Objects**
Function: spy(T object)

Purpose: Creates a spy to monitor real objects, allowing stubbing on the real methods.

Example:

```
List<String> list = new ArrayList<>();
List<String> spyList = spy(list);

// Use the spy to perform real operations
spyList.add("One");
spyList.add("Two");

// Stubbing a method on the spy
when(spyList.size()).thenReturn(100);

System.out.println(spyList.get(0)); // Outputs: One
System.out.println(spyList.size()); // Outputs: 100
```

**9. DoReturn/When for Stubbing Voids**
Function: doReturn(Object).when(T mock).methodToStub()

Purpose: Alternative syntax for stubbing, particularly useful for stubbing void methods with no return type.

Example:

```
doNothing().when(mockedList).clear();

mockedList.clear();
verify(mockedList).clear(); // Verifies that clear was called
```

**10. InOrder Verification**
Function: InOrder

Purpose: Ensures that methods are called in a specific order.

Example:

```
List firstMock = mock(List.class);
List secondMock = mock(List.class);

// Perform some actions
firstMock.add("was called first");
secondMock.add("was called second");

// Create inOrder verifier for the mocks
InOrder inOrder = inOrder(firstMock, secondMock);

// Verify that methods were called in order
inOrder.verify(firstMock).add("was called first");
inOrder.verify(secondMock).add("was called second");
```

**11. Timeout for Verification**
Function: verify(T mock, VerificationMode mode).methodToVerify(arguments)

Purpose: Verifies that a method is called within a specific time frame, useful for asynchronous testing.

Example:

```
// Assuming a mockedList object that processes async operations
verify(mockedList, timeout(100)).add("async call");
```

**12. Stubbing Consecutive Calls (Iterator-style Stubbing)**
Function: when(T methodCall).thenReturn(T value1).thenReturn(T value2)...

Purpose: Allows stubbing consecutive calls to a method to return different values or throw exceptions.

Example:

```
when(mockedList.get(0)).thenReturn("first call").thenReturn("second call");
```

```
System.out.println(mockedList.get(0)); // Outputs: first call
System.out.println(mockedList.get(0)); // Outputs: second call
```

## Mockito Imports

### Mockito Core

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

This static import includes the core Mockito methods such as mock(), when(), verify(), and others, enabling you to create mocks, stub method calls, and verify interactions without prefixing Mockito. before method names.

### Argument Captor

```
import org.mockito.ArgumentCaptor;
```

Used for capturing method arguments for further assertions. This is crucial when you want to test the specific values with which methods were called.

### JUnit Assert

```
import static org.junit.Assert.*;
```

While not part of Mockito, this import from JUnit is often used alongside Mockito for assertions (like assertEquals()) in your test cases to validate the expected outcomes.

### InOrder Verification

```
import org.mockito.InOrder;

import static org.mockito.Mockito.inOrder;
```

These imports are necessary for verifying the order of method calls. InOrder is the class used to create an order verifier, and inOrder() is the Mockito method to initialize it.

### Mockito Annotations

```
import org.mockito.Mock;

import org.mockito.InjectMocks;

import org.mockito.junit.MockitoJUnitRunner;
```

import org.junit.runner.RunWith;

When using annotations like @Mock or @InjectMocks for creating and injecting mocks respectively, these imports are required. MockitoJUnitRunner is used with the @RunWith annotation to initialize these annotated fields.

**Spy**

import static org.mockito.Mockito.spy;

Necessary when you are using Mockito spies to wrap real objects and monitor or stub their method calls.

**Verification with Timeout**

import static org.mockito.Mockito.timeout;

This import is used when verifying that a method call meets a specified timeout, useful in asynchronous testing scenarios.