Mockito

Mockito è un framework di mocking per il linguaggio di programmazione Java. La sua principale utilità è quella di aiutare nello sviluppo di test unitari creando oggetti simulati (chiamati mock) che possono sostituire oggetti reali durante i test. Questo è particolarmente utile quando si desidera isolare il codice in fase di test dall'implementazione effettiva di altre classi o componenti.

Ecco alcuni concetti chiave relativi a Mockito:

1. **Mockito.mock():**
    - Il metodo principale di Mockito per creare un mock oggetto è `Mockito.mock(ClassToMock.class)`. Questo metodo restituisce un'istanza di una classe che implementa l'interfaccia specificata, agendo come un "mock" di quella classe o interfaccia.

```
MyClass myMock = mock(MyClass.class);
```

2. **when() e thenReturn():**
    - Il metodo `when()` viene utilizzato per dichiarare il comportamento desiderato del mock quando viene chiamato un determinato metodo. Il metodo `thenReturn()` specifica cosa deve essere restituito quando il metodo è chiamato.

```
when(myMock.someMethod()).thenReturn("Hello");
```

3. **Verifica delle Chiamate:**
    - Mockito consente di verificare se i metodi di un mock sono stati chiamati, quanti sono stati chiamati e con quali parametri. Ad esempio, `verify(myMock).someMethod()` verifica se il metodo `someMethod()` è stato chiamato almeno una volta sul mock.

```
verify(myMock, times(1)).someMethod();
```

4. **Argument Matchers:**
    - Mockito supporta "argument matchers", che consentono di dichiarare condizioni più flessibili quando si specifica il comportamento del mock o quando si verificano le chiamate. Ad esempio, `anyString()` può essere utilizzato per corrispondere a qualsiasi stringa.

```
when(myMock.methodWithArgument(anyString())).thenReturn("Result");
```

5. **Spying:**
    - Mockito consente anche di creare "spy" oggetti, che sono oggetti reali con alcune delle loro chiamate rese monitorabili. A differenza di un mock, uno spy consente di mantenere la logica reale del metodo chiamato, a meno che non sia specificato diversamente.

```
MyClass mySpy = spy(new MyClass());
```

6. **Reset e Clear Invocations:**
    - Puoi resettare uno spy o un mock e cancellare le invocazioni precedenti utilizzando `reset()` e `clearInvocations()`.

```
reset(myMock);
clearInvocations(myMock);
```

7. **ArgumentCaptor:**
    - Mockito fornisce `ArgumentCaptor`, che consente di catturare gli argomenti passati a un metodo durante una chiamata di verifica. Questo può essere utile quando si desidera verificare argomenti specifici durante i test.

```
ArgumentCaptor<String> argumentCaptor = ArgumentCaptor.forClass(String.class);
verify(myMock).someMethod(argumentCaptor.capture());
String capturedValue = argumentCaptor.getValue();
```

8. **JUnit Integration:**
    - Mockito si integra bene con JUnit, consentendo l'uso di annotazioni come `@RunWith(MockitoJUnitRunner.class)` per semplificare la gestione dei mock. Puoi anche utilizzare i metodi `MockitoAnnotations.initMocks(this)` o l'annotazione `@Mock` per inizializzare i mock.

```
@RunWith(MockitoJUnitRunner.class)
public class MyTest {
    @Mock
    private MyClass myMock;
}
```

In generale, Mockito è uno strumento potente per isolare il codice in fase di test, migliorando la modularità e facilitando la scrittura di test unitari. Tuttavia, è importante utilizzare Mockito con cautela e capire quando è opportuno utilizzare un vero oggetto o un mock durante i test.