Mockito in Java-Tests verwenden

Um Mockito in Java-Tests zu verwenden, befolgen Sie diese Schritte, um Mock-Objekte zu erstellen, Methoden zu stubben und Interaktionen zu überprüfen:

1. Abhängigkeiten hinzufügen

Fügen Sie Mockito und JUnit in Ihrer `pom.xml` (Maven) hinzu:

```
<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. Testklasse einrichten

Verwenden Sie JUnit 5 mit der Mockito-Erweiterung:

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

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

3. Mocks erstellen und Abhängigkeiten injizieren

- Verwenden Sie `@Mock` für Abhängigkeiten, die Sie mocken möchten.
- Verwenden Sie `@InjectMocks`, um Mocks in die zu testende Klasse zu injizieren.

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

```
@InjectMocks
private UserService userService; // UserService hängt von UserRepository ab
```

4. Methodenaufrufe stubben

Verwenden Sie `when().thenReturn()`, um das Mock-Verhalten zu definieren:

```
@Test
void getUserById_ReturnsUser_WhenUserExists() {
    User mockUser = new User(1L, "Alice");
    // Stubben Sie die Repository-Methode
    when(userRepository.findById(1L)).thenReturn(mockUser);

    User result = userService.getUserById(1L);

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

5. Interaktionen überprüfen

Überprüfen Sie, ob eine Mock-Methode wie erwartet aufgerufen wurde:

```
@Test
void getUserById_CallsRepository() {
    userService.getUserById(1L);
    // Überprüfen Sie, dass die Repository-Methode einmal mit der ID 1 aufgerufen wurde
    verify(userRepository, times(1)).findById(1L);
}
```

Häufige Anwendungsfälle

Exceptions stubben

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

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

Argument Matcher Verwenden Sie `any()`, `eq()`, usw., um Argumente flexibel zu vergleichen:

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

BDD (Behavior-Driven Development) Stil Verwenden Sie `given().willReturn()` für bessere Lesbarkeit:

```
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 Fangen Sie Argumente ab, um detaillierte Assertions durchzuführen:

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

    userService.saveUser("Charlie");

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

Wichtige Hinweise

- **Initialisierung**: Verwenden Sie `@ExtendWith(MockitoExtension.class)` (JUnit 5) oder `MockitoAnnotations.openMoc` in `@BeforeEach`.
- **Statische Mocks**: Verwenden Sie die `mockito-inline`-Abhängigkeit zum Mocken statischer Methoden/Konstruktoren.
- **Spies**: Verwenden Sie `@Spy`, um echte Objekte zu umhüllen (teilweise Mocks).

Fehlerbehebung

- **Null-Mocks**: Stellen Sie sicher, dass Mocks initialisiert sind (verwenden Sie `@ExtendWith` oder `openMocks()`).
- **Stubbing-Fehler**: Stubben Sie Methoden, bevor sie im Test aufgerufen werden.

Für fortgeschrittene Nutzung verweisen Sie auf die Mockito-Dokumentation.