Utilisation de Mockito dans les tests Java

Pour utiliser Mockito dans les tests Java, suivez ces étapes pour créer des objets mock, stubber des méthodes et vérifier les interactions :

1. Ajouter les dépendances

Incluez Mockito et JUnit dans votre `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. Configurer la classe de test

Utilisez JUnit 5 avec l'extension Mockito :

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

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

3. Créer des mocks et injecter les dépendances

- Utilisez `@Mock` pour les dépendances que vous souhaitez mocker.
- Utilisez `@InjectMocks` pour injecter les mocks dans la classe à tester.

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

```java
@InjectMocks
private UserService userService; // UserService dépend de UserRepository
```

4. Stubber les appels de méthode

Utilisez `when().thenReturn()` pour définir le comportement des mocks :

```java
@Test
void getUserById_ReturnsUser_WhenUserExists() {
    User mockUser = new User(1L, "Alice");
    // Stubber la méthode du repository
    when(userRepository.findById(1L)).thenReturn(mockUser);

    User result = userService.getUserById(1L);

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

5. Vérifier les interactions

Vérifiez si une méthode mock a été appelée comme prévu :

```java
@Test
void getUserById_CallsRepository() {
    userService.getUserById(1L);
    // Vérifier que la méthode du repository a été appelée une fois avec l'ID 1
    verify(userRepository, times(1)).findById(1L);
}
```

Cas d’utilisation courants

Stubber les exceptions

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

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

Argument Matchers Utilisez `any()`, `eq()`, etc., pour correspondre aux arguments de manière flexible :

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

Style BDD (Behavior-Driven Development) Utilisez `given().willReturn()` pour une meilleure lisibilité :

```
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 Capturez les arguments pour effectuer des assertions détaillées :

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

    userService.saveUser("Charlie");

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

Notes clés

- **Initialisation** : Utilisez `@ExtendWith(MockitoExtension.class)` (JUnit 5) ou `MockitoAnnotations.openMocks(this)` dans `@BeforeEach`.
- **Mocks statiques** : Utilisez la dépendance `mockito-inline` pour mocker les méthodes statiques/constructeurs.
- **Spies** : Utilisez `@Spy` pour envelopper des objets réels (mocks partiels).

Dépannage

- **Mocks nuls** : Assurez-vous que les mocks sont initialisés (utilisez `@ExtendWith` ou `openMocks()`).
- **Erreurs de stubbing** : Stubber les méthodes avant qu'elles ne soient appelées dans le test.

Pour une utilisation avancée, référez-vous à la documentation Mockito.