Usando Mockito en Pruebas de Java

Para usar Mockito en pruebas de Java, sigue estos pasos para crear objetos simulados, definir métodos simulados y verificar interacciones:

1. Agregar Dependencias

Incluye Mockito y JUnit en tu `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. Configurar la Clase de Prueba

Usa JUnit 5 con la extensión de Mockito:

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

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

3. Crear Simulaciones e Inyectar Dependencias

- Usa `@Mock` para las dependencias que deseas simular.
- Usa `@InjectMocks` para inyectar simulaciones en la clase bajo prueba.

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

```
@InjectMocks
private UserService userService; // UserService depende de UserRepository
```

4. Definir Llamadas de Métodos Simulados

Usa `when().thenReturn()` para definir el comportamiento de la simulación:

```
@Test
void getUserById_ReturnsUser_WhenUserExists() {
    User mockUser = new User(1L, "Alice");
    // Simular el método del repositorio
    when(userRepository.findById(1L)).thenReturn(mockUser);

    User result = userService.getUserById(1L);

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

5. Verificar Interacciones

Verifica si un método simulado fue llamado como se esperaba:

```
@Test
void getUserById_CallsRepository() {
    userService.getUserById(1L);
    // Verificar que el método del repositorio fue llamado una vez con ID 1
    verify(userRepository, times(1)).findById(1L);
}
```

Casos de Uso Comunes

Simular Excepciones

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

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

Coincidencias de Argumentos Usa `any()`, `eq()`, etc., para coincidir con argumentos de manera flexible:

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

Estilo BDD (Desarrollo Orientado a Comportamiento) Usa `given().willReturn()` para mejorar la legibilidad:

```
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());
}
```

Captor de Argumentos Captura argumentos para realizar aserciones detalladas:

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

    userService.saveUser("Charlie");

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

Notas Clave

- **Inicialización**: Usa `@ExtendWith(MockitoExtension.class)` (JUnit 5) o `MockitoAnnotations.openMocks(this)` en `@BeforeEach`.
- **Simulaciones Estáticas**: Usa la dependencia `mockito-inline` para simular métodos/constructores estáticos.
- **Espías**: Usa `@Spy` para envolver objetos reales (simulaciones parciales).

Solución de Problemas

- **Simulaciones Nulas**: Asegúrate de que las simulaciones se inicialicen (usa `@ExtendWith` o `openMocks()`).
- **Errores de Simulación**: Simula métodos antes de que se llamen en la prueba.

Para un uso avanzado, consulta la Documentación de Mockito.