



Mocking

Metodologias e Desenvolvimento de Software

Pedro Salgueiro

pds@di.uevora.pt

CLV-256



Testes Unitários

- Testar unidades
 - Métodos, funções, procedimentos, objetos, etc...
- Testar de forma isolada
 - Não depender de comportamentos de outras unidades
- Problema
 - Unidades dependem de outras unidades
 - Difícil de isolar unidades
- Solução
 - Simular unidades
 - Fazer “Mocking” de unidades



Fazer mocking

1) Criar objetos mock

- Objectos simulados

2) Especificar o comportamento dos objetos mock

- Comportamento específico
- Definido para cada teste

3) Comportamento típicos

- Quando “X” for invocado, devolve “Y”
- Quando “X” for invocado com argumentos “Y” e “Z”, devolve “W”



Criar objetos mock

- Manualmente
 - Através de código
 - Criação de classes “dummy”
 - Implementam as classes originais
 - Fazem override dos métodos usados nos testes
 - Comportamento “controlado”
 - Durante o teste são usadas as classes “dummy”
 - Não recomendado
 - Ou mesmo proibido
 - Má prática de programação



Criar objetos mock

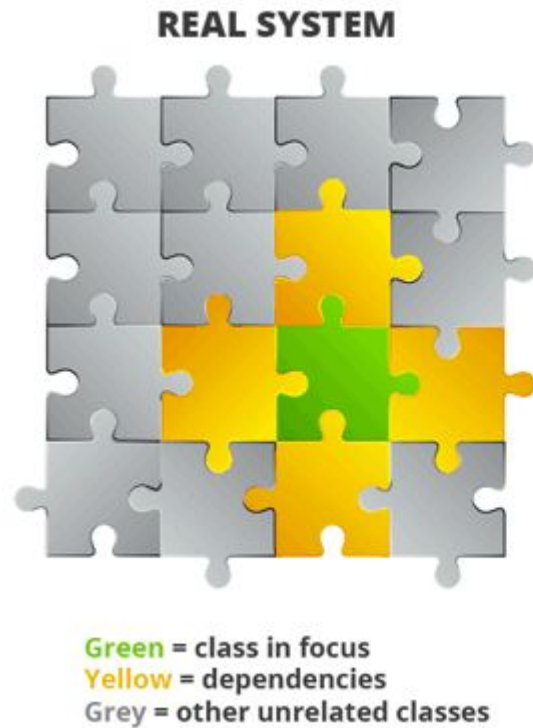
- Forma automática
 - Frameworks de Mocking
 - Forma correta de fazer Mocking de objetos
- Frameworks de Mocking
 - Criar objetos “falsos” a partir de classes/interfaces reais
 - Especificar o comportamento dos objetos “falsos”
 - Para dados específicos
 - Para cada método
 - Verificar a execução de métodos nos objetos “falsos”
 - Verificar argumentos passados aos métodos dos objetos falsos
 - Lançar exceções para algumas invocações



Frameworks de Mocking

- Usados para criar objetos simulados
- Para serem usados em testes unitários
- Substituírem os objetos reais durante os testes
- “Enganarem” um objeto
 - Pensar que está a interagir com um objeto real
- “Peça de teatro”
 - Planeada de forma muito cuidada
- Replica o “processo manual”
 - De forma cuidada e segura
 - De forma muito simples
- Alguns frameworks de Mocking para Java
 - **Mockito**, **PowerMock**, EasyMock

Frameworks de Mocking



CLASS IN UNIT TEST





Mocking

- Quando usar
 - Interação com métodos que não tenham um comportamento determinístico
 - Interação com métodos que tenham efeitos secundários que só façam sentido em ambientes de produção
 - Invocar operações externas
 - Forçar erros “estranhos” e difíceis de verificar



Mocking

- Uso típico
 - 1) Fazer Mock das dependências da classe a ser testada
 - 2) Executar o código da classe a ser testada
 - 3) Verificar se o código foi executado da forma esperada



Mockito

- Criar objetos de Mocks
 - Anotação @Mock
- Especificar valor de retorno
 - “when thenReturn”
 - “when thenThrow”
 - Invocações não especificadas devolvem valores nulos de acordo com o tipo
 - null para objetos
 - 0 para números
 - false para booleanos
 - colecções vazias para colecções
 - ...



Mockito

Exemplo típico

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

public class MockitoTest {

    @Mock
    MyDatabase databaseMock;

    @Rule public MockitoRule mockitoRule = MockitoJUnit.rule();

    @Test
    public void testQuery() {
        ClassToTest t = new ClassToTest(databaseMock);
        boolean check = t.query("* from t");
        assertTrue(check);
        verify(databaseMock).query("* from t");
    }
}
```



Mockito

- Especificar comportamientos
 - Valores de retorno
 - Diferentes para cada método
 - Dependendo dos argumentos
 - Possível ignorar os argumentos
 - API “fluente”
 - `when(...).thenReturn(...)`
 - `doReturn(...).when(...).methodCall`



Mockito

when(...).thenReturn(...)

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

@Test
public void test1() {
    // create mock
    MyClass test = mock(MyClass.class);

    // define return value for method getUniqueId()
    when(test.getUniqueId()).thenReturn(43);

    // use mock in test....
    assertEquals(test.getUniqueId(), 43);
}
```



Mockito

when(...).thenReturn(...)

```
// demonstrates the return of multiple values
@Test
public void testMoreThanOneReturnValue() {
    Iterator<String> i= mock(Iterator.class);
    when(i.next()).thenReturn("Mockito").thenReturn("rocks");
    String result= i.next()+" "+i.next();
    //assert
    assertEquals("Mockito rocks", result);
}

// this test demonstrates how to return values based on the input
@Test
public void testReturnValueDependentOnMethodParameter() {
    Comparable<String> c= mock(Comparable.class);
    when(c.compareTo("Mockito")).thenReturn(1);
    when(c.compareTo("Eclipse")).thenReturn(2);
    //assert
    assertEquals(1, c.compareTo("Mockito"));
}
```



Mockito

when(...).thenReturn(...)

```
// this test demonstrates how to return values independent of the input value
@Test
public void testReturnValueIndependentOnMethodParameter() {
    Comparable<Integer> c= mock(Comparable.class);
    when(c.compareTo(anyInt())).thenReturn(-1);
    //assert
    assertEquals(-1, c.compareTo(9));
}

// return a value based on the type of the provide parameter
@Test
public void testReturnValueIndependentOnMethodParameter2() {
    Comparable<Todo> c= mock(Comparable.class);
    when(c.compareTo(isA(Todo.class))).thenReturn(0);
    //assert
    assertEquals(0, c.compareTo(new Todo(1)));
}
```



Mockito

when(...).thenThrow(...)

```
Properties properties = mock(Properties.class);

when(properties.get("Anddroid")).thenThrow(new
IllegalArgumentException(...));

try {
    properties.get("Anddroid");
    fail("Anddroid is misspelled");
} catch (IllegalArgumentException ex) {
    // good!
}
```




Mockito

`doReturn(...).when(...).methodCall`

- Util quando usado juntamente com Spy
 - Wrap de objetos reais
 - Especificar valores de retorno

```
Properties properties = new Properties();  
Properties spyProperties = spy(properties);  
doReturn("42").when(spyProperties).get("shoeSize");  
String value = spyProperties.get("shoeSize");  
assertEquals("42", value);
```



Mockito

Spy

- Wrap de objetos reais
- Todas as invocações são delegadas para o objeto real
 - Excepto se especificado valor de retorno específico
- `when(...).thenReturn(...)`
 - Pode dar origem a erros

```
@Test
public void testLinkedListSpyWrong() {
    // Lets mock a LinkedList
    List<String> list = new LinkedList<>();
    List<String> spy = spy(list);

    // this does not work
    // real method is called so spy.get(0)
    // throws IndexOutOfBoundsException (list is still empty)
    when(spy.get(0)).thenReturn("foo");

    assertEquals("foo", spy.get(0));
}
```



Mockito

Spy

- `doReturn(...).when(...).get(...)`
 - Forma correcta

```
@Test
public void testLinkedListSpyCorrect() {
    // Lets mock a LinkedList
    List<String> list = new LinkedList<>();
    List<String> spy = spy(list);

    // You have to use doReturn() for stubbing
    doReturn("foo").when(spy).get(0);

    assertEquals("foo", spy.get(0));
}
```



Mockito

Verificar invocação de métodos

- Behavior testing
 - Verificam-se comportamentos
 - Não se verificam resultados
- Verificar se condições especificadas foram cumpridas
 - Se um método foi invocado de acordo com parâmetros específicos
 - N° de invocações
 - Valores de parâmetros
 - ...



Mockito

Verificar invocação de métodos

```
@Test
public void testVerify() {
    // create and configure mock
    MyClass test = Mockito.mock(MyClass.class);
    when(test.getId().thenReturn(43);

    // call method testing on the mock with parameter 12
    test.testing(12);
    test.getId();
    test.getId();

    // now check if method testing was called with the parameter 12
    verify(test).testing(ArgumentMatchers.eq(12));

    // was the method called twice?
    verify(test, times(2)).getId();

    // other alternatives for verifying the number of method calls for a method
    verify(test, never()).someMethod("never called");
    verify(test, atLeastOnce()).someMethod("called at least once");
    verify(test, atLeast(2)).someMethod("called at least twice");
    verify(test, times(5)).someMethod("called five times");
    verify(test, atMost(3)).someMethod("called at most 3 times");
    // This let's you check that no other methods were called on this object.
    // You call it after you have verified the expected method calls.
    verifyNoMoreInteractions(test);
}
```



Mockito

Injeção de Mocks

- Injeção de
 - Construtores, métodos ou atributos
 - Com base no tipo
 - Em objetos reais
 - De forma automática



Mockito

Injeção de Mocks

- Considerando a seguinte classe:

```
public class ArticleManager {  
    private User user;  
    private ArticleDatabase database;  
  
    public ArticleManager(User user, ArticleDatabase database) {  
        super();  
        this.user = user;  
        this.database = database;  
    }  
  
    public void initialize() {  
        database.addListener(new ArticleListener());  
    }  
}
```



Mockito

Injeção de Mocks

```
@RunWith(MockitoJUnitRunner.class)
public class ArticleManagerTest {

    @Mock ArticleCalculator calculator;
    @Mock ArticleDatabase database;
    @Mock User user;

    @Spy private UserProvider userProvider = new ConsumerUserProvider();

    @InjectMocks private ArticleManager manager;

    @Test public void shouldDoSomething() {
        // calls addListener with an instance of ArticleListener
        manager.initialize();

        // validate that addListener was called
        verify(database).addListener(any(ArticleListener.class));
    }
}
```




Mockito

Mocking de classes “Final”

- Possível a partir do Mockito v2
 - Ainda está em fase de teste
 - Necessário ativar esta funcionalidade
 - Criar o ficheiro
 - `src/test/resources/mockito-extensions/org.mockito.plugins.MockMaker` **OU**
 - `src/mockito-extensions/org.mockito.plugins.MockMaker`
 - E adicionar a seguinte linha
 - `mock-maker-inline`



Mockito

Mocking de classes “Final”

```
final class FinalClass {  
    public final String finalMethod() { return "something"; }  
}  
  
@Test  
public final void mockFinalClassTest() {  
    FinalClass instance = new FinalClass();  
  
    FinalClass mock = mock(FinalClass.class);  
    when(mock.finalMethod()).thenReturn("that other thing");  
  
    assertEquals(mock.finalMethod(), instance.finalMethod());  
}
```



Mockito

Testar métodos estáticos

- Mockito não permite
- É necessário recorrer a outros frameworks
 - Powermock
- Powermock
 - Disponibiliza classe para ser usada no Mockito: “PowerMockito”
 - Permite usar todas as operações do Mockito



Mockito

Testar métodos estáticos

Considerando a seguinte classe:

```
import java.net.InetAddress;
import java.net.UnknownHostException;

public final class NetworkReader {
    public static String getLocalHostname() {
        String hostname = "";
        try {
            InetAddress addr = InetAddress.getLocalHost();
            // Get hostname
            hostname = addr.getHostName();
        } catch ( UnknownHostException e ) {
        }
        return hostname;
    }
}
```



Mockito

Testar métodos estáticos

- Fazer mock da classe NetworkReader

```
import org.junit.runner.RunWith;
import org.powermock.core.classloader.annotations.PrepareForTest;

@RunWith( PowerMockRunner.class )
@PrepareForTest( NetworkReader.class )
public class MyTest {

    // Find the tests here

    @Test
    public void testSomething() {
        mockStatic( NetworkReader.class );
        when( NetworkReader.getLocalHostname() ).andReturn( "localhost" );

        // now test the class which uses NetworkReader
    }
}
```



Referências

- <http://site.mockito.org/>
- <http://static.javadoc.io/org.mockito/mockito-core/2.18.3/org/mockito/Mockito.html>
- <https://martinfowler.com/articles/mocksArentStubs.html>