

Mockowanie



- Mockowanie daje przewidywalny wynik niezależny od np. połączenia z innymi systemami
- Mockowanie pozwala pozbyć się zależności
- Dzięki mockowaniu łatwo uniknąć sytuacji, gdy testy dają inne wyniki w zależności na jakim systemie zostały wywołane (litera „R” w F.I.R.S.T)
- Mocki pozwalają w łatwy sposób zmieniać zachowanie danego obiektu w różnych test caseach

```
testCompile group: 'org.mockito', name: 'mockito-core', version:
'3.3.3'
```

```
<dependency>
  <groupId>org.mockito</groupId>
  <artifactId>mockito-core</artifactId>
  <version>3.3.3</version>
  <scope>test</scope>
</dependency>
```

Stubbing



```
Controller mockedController = Mockito.mock(Controller.class);

// Stubbing
Mockito.when(mockedController.process("GET /home")).thenReturn(404);

// Will print: 404
System.out.println(mockedController.process("GET /home"));
```

```
//creating a mock
LinkedList mockedList = mock(LinkedList.class);

//stubbing
when(mockedList.get(0)).thenReturn("first item");
when(mockedList.get(1)).thenThrow(new RuntimeException());

//following call prints "first item"
System.out.println(mockedList.get(0));

//following call throws runtime exception
System.out.println(mockedList.get(1));
```

```
// Stub void method to throw Exception  
doThrow(new RuntimeException()).when(mockedList).clear();  
  
//following throws RuntimeException:  
mockedList.clear();
```

```
// Call real method
when(mockedList.add("test")).thenReturn(true);

mockedList.add("test");
// This assertion will pass
assertEquals(1, mockedList.size());

// Calling real void method
doCallRealMethod().when(mockedList).clear();
```



```
ArgumentMatchers.anyInt();  
ArgumentMatchers.any();  
ArgumentMatchers.anyBoolean();  
ArgumentMatchers.anyDouble();  
ArgumentMatchers.anyList();  
ArgumentMatchers.anyString();  
ArgumentMatchers.startsWith("ABC");  
ArgumentMatchers.eq("Hi There");  
ArgumentMatchers.contains("xyz");
```

```
class CustomMatcher implements ArgumentMatcher<Movie> {  
  
    @Override  
    public boolean matches(Movie movie) {  
        if (movie.getTitle().startsWith("Fast and Furious")) {  
            return false;  
        }  
        return true;  
    }  
}
```

- Jeśli użyjemy Matchera w jednym argumencie, wszystkie argumenty muszą być Matcherami

```
// this is correct - any is also a matcher  
when(mock.process(anyString(), anyInt(),  
eq("third"))).thenReturn("finish");
```

```
// this is incorrect - exception will be thrown  
when(mock.process(anyString(), anyInt(),  
"third")).thenReturn("finish");
```



Weryfikacja

- Co jeśli metoda, którą testujemy nie zwraca wartości (`void`)
- Dzięki weryfikacji możemy sprawdzić, czy dana metoda została wywołana (też z konkretnymi argumentami)
- Można zweryfikować ilość wywołań
- Można zweryfikować, czy na mocku zostały wywołane metody (tzw. interakcja)

```
LinkedList mockedList = mock(LinkedList.class);
mockedList.size();
// Check if method was invoked once
// Without above line exception will be thrown
verify(mockedList).size();
```

```
mockedList.add("once");

mockedList.add("twice");
mockedList.add("twice");

mockedList.add("three times");
mockedList.add("three times");
mockedList.add("three times");

//following two verifications work exactly the same - times(1) is used by default
verify(mockedList).add("once");
verify(mockedList, times(1)).add("once");

//exact number of invocations verification
verify(mockedList, times(2)).add("twice");
verify(mockedList, times(3)).add("three times");
```

```
// Verify that nothing was invoked on mock  
verifyZeroInteractions(mockedList);
```

```
// Let's verify the order of operations  
mockedList.size();  
mockedList.add("a parameter");  
mockedList.clear();
```

```
InOrder inOrder = inOrder(mockedList);  
inOrder.verify(mockedList).size();  
inOrder.verify(mockedList).add("a parameter");  
inOrder.verify(mockedList).clear();
```

```
@Test
public void verifyInteractionTimes() throws Exception {
    User user = new User(mockWebService, USER_ID, PASSWORD);

    user.logout();

    verify(mockWebService, times(1)      ).logout();
                           atLeast(1)
                           atLeastOnce()
                           atMost(1)
                           only()
                           never()
}
```



```
@Test
public void captureArguments() throws Exception {
    User user = new User(mockWebService, USER_ID, PASSWORD);
    user.login(mockLoginInterface);
    verify(mockWebService).login(responseArgumentCaptor.capture());
    Response response = responseArgumentCaptor.getValue();

    response.onRequestCompleted(true);

    verify(mockLoginInterface).onLoginSuccess();
}
```

```
@Test
public void stubMethod() throws Exception {
    User user = new User(mockWebService, USER_ID, PASSWORD);
    when(mockWebService.isOffline()).then(new Answer<Boolean>() {
        int index = 0;

        @Override
        public Boolean answer(InvocationOnMock in) throws ... {
            return index++ % 2 > 0;
        }
    });

    user.login(mockLoginInterface);

    verify(mockWebService, never()).login();
}
```

- ▶ Normal syntax

```
when(mockWebService.isOffline()).thenReturn(true);
```

- ▶ Alternative syntax

```
doReturn(true).when(mockWebService).isOffline();
```

- ▶ BDD syntax

```
given(mockWebService.isOffline()).willReturn(true);
```

```
mockedList.clear();  
mockedList.clear();  
mockedList.clear();  
  
verify(mockedList, atLeast(1)).clear();  
verify(mockedList, atMost(10)).clear();  
  
// Verify with exact argument  
mockedList.add("test");  
verify(mockedList).add("test");
```

- Mock automatycznie zastępuje wszystkie metody stubbami
- Co jeśli chcemy zamockować wywołanie tylko 1 metody?
 - Ściana `thenCallRealMethod()` ?
- Spy
 - Tworzony z prawdziwego obiektu (od wersji 1.10.12 można tworzyć spy na klasach abstrakcyjnych bez potrzeby podawania obiektu)
 - Wywoływane są prawdziwe metody – jeśli nie zostaną zastubbowane
 - Tworzona jest kopia prawdziwego obiektu

```
List list = new LinkedList();  
List spy = spy(list);  
  
//optionally, you can stub out some methods:  
when(spy.size()).thenReturn(100);  
  
//using the spy calls *real* methods  
spy.add("one");  
spy.add("two");  
  
//prints "one" - the first element of a list  
System.out.println(spy.get(0));  
  
//size() method was stubbed - 100 is printed  
System.out.println(spy.size());
```

```
List list = new LinkedList();  
List spy = spy(list);  
  
//Impossible: real method is called so spy.get(0)  
// throws IndexOutOfBoundsException (the list is yet  
empty)  
when(spy.get(0)).thenReturn("foo");  
  
//You have to use doReturn() for stubbing  
doReturn("foo").when(spy).get(0);
```

```
public class ArticleManagerTest {
    @Mock
    private ArticleCalculator calculator;
    @Mock(name = "database")
    private ArticleDatabase dbMock; // note the mock name attribute
    @Spy
    private UserProvider userProvider = new ConsumerUserProvider();

    @InjectMocks
    private ArticleManager manager;

    // test go here
}

public class ArticleManager {
    ArticleManager(ArticleCalculator calculator, ArticleDatabase database) {
        // parameterized constructor
    }
}
```


<https://github.com/michalgorecki/mockito-training> - repozytorium z ćwiczeniami

- <https://site.mockito.org/> - dokumentacja do Mockito
- <https://www.baeldung.com/mockito-series> – dobry tutorial do Mockito
- <https://mvnrepository.com/> - repozytorium z zależnościami



www.tt.psc.pl/en
www.25lat.tt.com.pl/en
www.iot.tt.com.pl/en

Looking forward to build longterm partnership with You • Imię Nazwisko imie.nazwisko@ttpsc.pl