JUnit Rules

In the above examples, we have used the JUnit **runner (MockitoJUnitRunner)**. It makes the test dependent on that particular runner.

We cannot use multiple runners in the same test. To overcome this problem, we should follow **JUnit rules** that makes the test more flexible. It allows us to use multiple rules in the same test.

A JUnit rule is defined as a component that is used to obstruct the test method calls and allows us to perform something before and after the test method is invoked. The JUnit provides the following rules to:

* Create directories/files that are deleted after a test method has been run.
* Fail a test, if the described timeout has exceeded before a test method is invoked.
* Establish an external resource like a socket or a database connection before a test method is invoked.
* Free the configured external resource after a test method is invoked.

To use the JUnit rules, we need to add the **@Rule** annotation in the test.

**@Rule:** It annotates the fields. It refer to the rules or methods that returns a rule. The annotated fields must be public, non-static, and subtypes of the **TestRule** or **MethodRule.**

1. @Rule
2. **public** MockitoRule mockitorule = MockitoJUnit.rule();

In the above code snippet, we have used the **MockitoRule** class. You can use any JUnit rule as per your requirement.

Example of JUnit Rule (MockitoRule)

Here, we are going to create an example using the JUnit rule. In this example, we are using the **MockitoRule,** and we can use any JUnit rule as per your requirement.

**Step 1:** Create an interface named **ToDoService** that contains two unimplemented methods.

**ToDoService.java**

1. **import** java.util.List;
3. **public** **interface** ToDoService {
5. **public** List<String> getTodos(String user);
6. **public** **void** deleteTodos(String doString);
7. }

**Step 2:** Create an implementation class named **ToDoBusiness.**

**ToDoBusiness.java**

1. **import** java.util.ArrayList;
2. **import** java.util.List;
4. // ToDoBusiness is a SUT (system under test)
5. // ToDoService is a Dependency (as ToDoBusiness is dependent on it)
7. **public** **class** ToDoBusiness {
9. **public** ToDoService doService;
11. **public** ToDoBusiness(ToDoService doService) {
12. **this**.doService = doService;
13. }
15. **public** **void** deleteTodosNotRelatedToHibernate(String user) {
17. List<String> Combinedlist = doService.getTodos(user);
19. **for**(String todos:Combinedlist) {
20. **if**(!todos.contains("Hibernate"))
21. {
22. doService.deleteTodos(todos);
23. }
24. }
25. }
26. }

**Step 3:** Create a test class named **ToDoBusinessMock** in which business logic is defined.

**ToDoBusinessMock.java**

1. **import** **static** org.hamcrest.CoreMatchers.is;
2. **import** **static** org.junit.Assert.assertThat;
3. **import** **static** org.mockito.BDDMockito.given;
4. **import** **static** org.mockito.BDDMockito.then;
5. **import** **static** org.mockito.Mockito.never;
6. **import** **static** org.mockito.Mockito.times;
7. **import** **static** org.mockito.Mockito.verify;
8. **import** java.util.Arrays;
9. **import** java.util.List;
11. **import** org.junit.Rule;
12. **import** org.junit.Test;
13. **import** org.mockito.ArgumentCaptor;
14. **import** org.mockito.Captor;
15. **import** org.mockito.InjectMocks;
16. **import** org.mockito.Mock;
17. **import** org.mockito.junit.MockitoJUnit;
18. **import** org.mockito.junit.MockitoRule;
20. **public** **class** ToDoBusinessMock {
22. @Rule
23. **public** MockitoRule mockitorule = MockitoJUnit.rule();
25. @Mock
26. ToDoService servicemock;
28. @InjectMocks
29. ToDoBusiness business;
31. @Captor
32. ArgumentCaptor<String> argumentCaptor;
34. @Test
35. **public** **void** deleteTodosusing\_BDD() {
37. // Given
38. List<String> combinedlist = Arrays.asList("Use Hibernate Java",
39. "Use Hibernate Core", "Use Hibernate", "Use Spring MVC");
41. given(servicemock.getTodos("dummy")).willReturn(combinedlist);
43. // When
44. business.deleteTodosNotRelatedToHibernate("dummy");
46. // Then
47. verify(servicemock, times(1)).deleteTodos("Use Spring MVC");
48. verify(servicemock, never()).deleteTodos("Use Hibernate Java");
49. verify(servicemock, never()).deleteTodos("Use Hibernate");
50. System.out.println("test is working..");
51. }
53. @Test
54. **public** **void** deleteTodosusing\_BDD\_usingArgumentCaptor() {
56. //Given
57. List<String> combinedlist = Arrays.asList("Use Hibernate Java",
58. "Use Hibernate Core", "Use Hibernate", "Use Spring MVC");
60. given(servicemock.getTodos("dummy")).willReturn(combinedlist);
62. //When
63. business.deleteTodosNotRelatedToHibernate("dummy");
65. //Then
66. then(servicemock).should().deleteTodos(argumentCaptor.capture());
67. assertThat(argumentCaptor.getValue(),is("Use Spring MVC"));
68. System.out.println("test is working..");
69. }
70. }

**Output**

The following output shows that the test is successfully running using the JUnit rule **(MockitoRule).**

