Creating Mocks

In this section:

Introduction

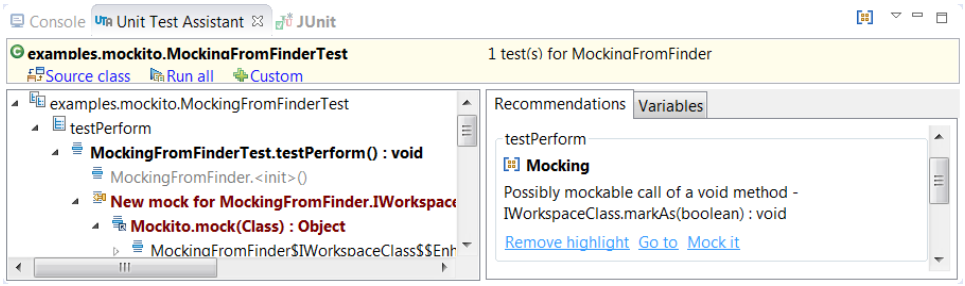
Unit Test Assistant facilitates creating mock objects that isolate a test from its dependencies. When a test is generated, UTA can automatically create mocks, which simulate the behavior of real complex objects, such as external interfaces, that cannot be incorporated into a test. This allows you to run your tests in isolation - without having to add excessive code to your test.

Creating Unit Tests with Mocks

1. Ensure that the options for creating mocks have been configured (see Configuring Mocking Options).
2. Create a new unit test with UTA (see Creating a Basic Unit Test, Creating Multiple Unit Tests or Creating a Parameterized Unit Test). By analyzing source code, UTA will detect complex objects that cannot be incorporated and add a mock object to the test.

```
// When
IWorkspaceClass item = Mockito.mock(IWorkspaceClass.class);
int getValueResult = 0; // UTA: default value
Mockito.when(item.getValue()).thenReturn(getValueResult);
```

3. Execute the test and view the results in the UTA interface (see Executing Unit Tests with Unit Test Assistant).

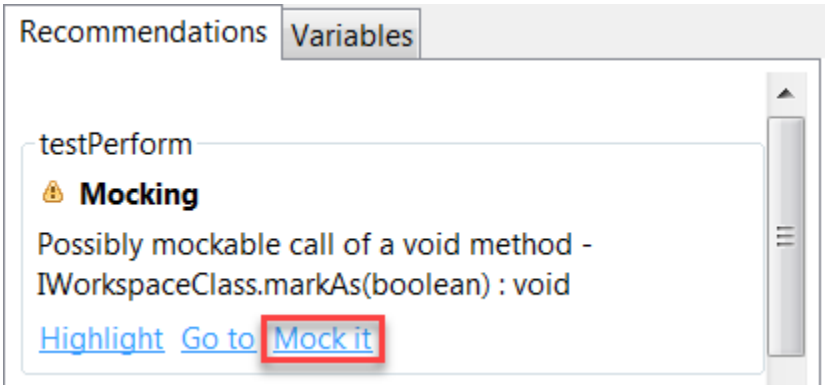


The **Recommendations** section displays information about interactions with mock objects. Clicking the **Highlight** link helps you view these interactions in execution flow displayed in the left column. The nodes marked in **brown** indicate the mock object. The nodes in **bold** indicate interactions with the mock object. Double-clicking the nodes navigates the test code.

Customizing Mocks

Mocks created with UTA include default values you may want to modify.

1. Click the **Mock it** link in the recommendation field to extend the test with elements required for mock modification.



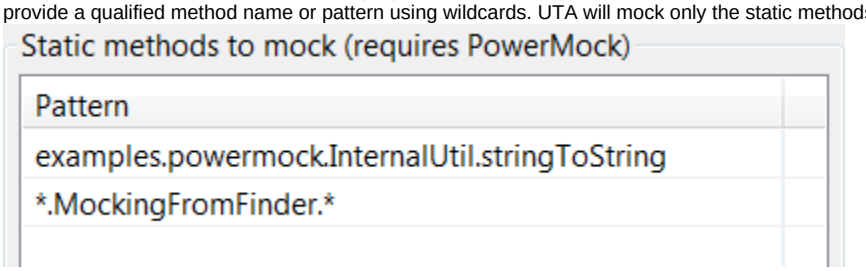
2. Modify the mock with values that meet your testing strategy.

```
// When
IWorkspaceClass item = Mockito.mock(IWorkspaceClass.class);
boolean isPositiveResult = false; // UTA: default value
Mockito.when(item.isPositive()).thenReturn(isPositiveResult);
Mockito.doAnswer(new Answer<Void>() {
    public Void answer(InvocationOnMock invocation) {
        boolean param0 = (Boolean) invocation.getArguments()[0];
        // TODO Auto-generated answer method
        return null;
    }
}).when(item).markAs(Mockito.anyBoolean());
int getValueResult = 0; // UTA: default value
Mockito.when(item.getValue()).thenReturn(getValueResult);
```

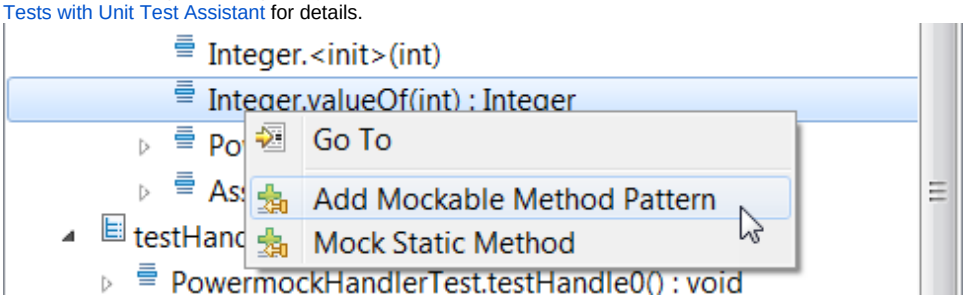
Mocking Static Methods

You can configure UTA to create mocks for specific static methods.

1. Go to **Parasoft> Preferences> Unit Test Assistant** in your IDE menu bar.
2. Enable the **Mockable static invocations (requires PowerMock)** option in the Recommendations section (see Configuring Mocking Options for details).
3. Go to **Parasoft> Preferences> Unit Test Assistant>Mocking**.
4. Select the **Enable** check box to enable creating unit tests with mocks.
5. Select the **PowerMock with Mockito** framework.
6. Complete the **Static methods to mock (requires PowerMock)** table with a list of static methods that you want to be mocked. Click **New** and provide a qualified method name or pattern using wildcards. UTA will mock only the static methods that are specified in the table.



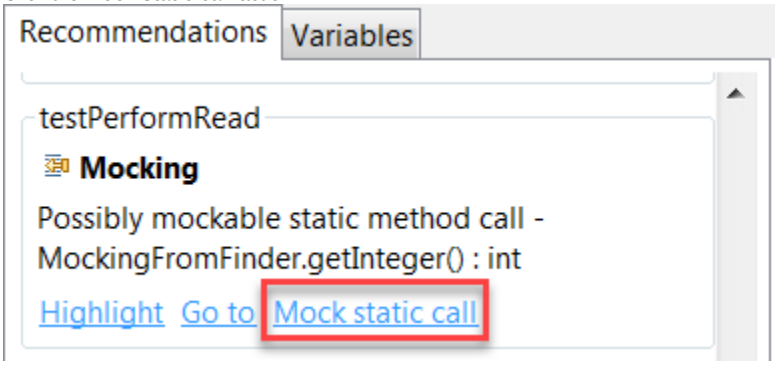
Alternatively, you can add static methods to the list after you execute your tests (see Executing Unit Tests with Unit Test Assistant for details). Right-click a node that represents a static method in the execution flow and choose the **Add Mockabe Method Pattern** option. Selecting the **Mock Static Method** option will update your code to mock the selected method call, but the method will not be added to the table; see Executing Unit Tests with Unit Test Assistant for details.



If UTA detects invocations of the specified static methods in the tested code during test creation, mocks will be included in the generated test templates (see Creating a Basic Unit Test).

7. Run your unit tests with UTA (see Executing Unit Tests with Unit Test Assistant). UTA will detect calls to the specified static methods and display the Recommendations that allow you automatically create mocks.

8. Click the **Mock static call** action link.



The specified static method will be mocked with the with the *mockStatic* method.

```
121    @Test
122    public void testPerformRead5() throws Throwable {
123        // Given
124        mockStatic(MockingFromFinder.class);
125
```