Summary

This guide will run through how to use Mock objects for running unit test cases.

Description

Mock objects

Mock object [http://en.wikipedia.org/wiki/Mock_object] is an object used for imitating the object class under scrutiny for helping unit tests run, and it increases independency of the unit tests.

If you want to test an object in a program, the object is likely to be inter-related with other classes. However, if the developer wants to test solely the object only, and would prefer not to (or unable to) use other "collaborator" objects, instead he can create Mock objects that imitate collaborator objects (i.e. MockCollaborator1~3), and define the main object methods that he wants to test, he can use mock objects.

An easier example is, Mock objects can be created in order to eliminate compile errors for object in test. Or, if web containers or DB environment is required but is hard to replicate it in the test setting, a Mock object can be created to imitate those elements. All these cases can be covered by Mock objects.

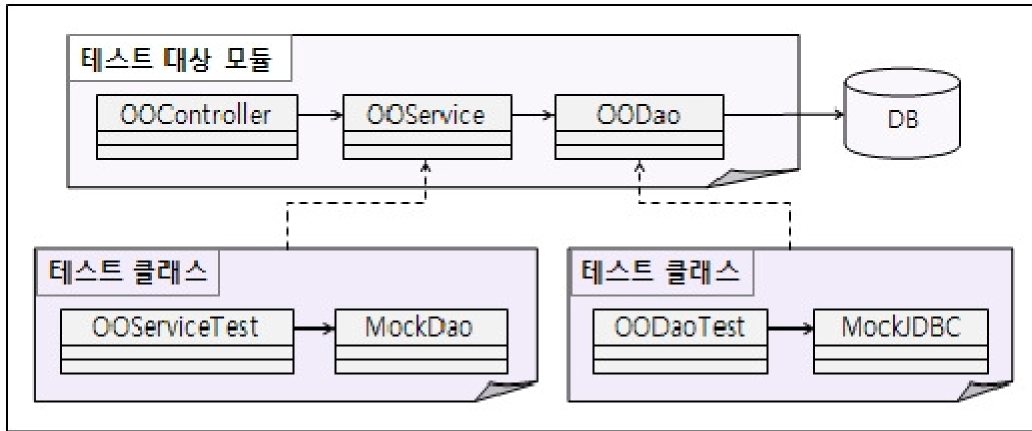
To sum up, following are the scenarios where Mock objects can be used.

- Actual object for test cnnot be created or set up.
- Actual behavior by the object is difficult to trigger.
- Actual object is slow in responses.
- User interfaces the ones that need testing.
- Test needs to ask the actual object on how to run.
- Test object is not available yet. (i.e. collaborating with other teams, or working on new hardware platform)

How to create mock objects for test cases

When using runtime environments, developers are likely to develop code in tiered manner for Controller, Service, DAO and so on, plus would need to use databases. In that case, using mock objects so that test cases can be independently used to test those modules, or, in other words, mock objects (MockDao, MockJDBC) can be created for test code per test target (OOServiceTest, OODaoTest) and per collaborator (OODao, DB) for testing purposes.

You can also create mock objects separately, or in the same source, to focus on testing your current source code. Also, if you want to connect with a real server to test, refer to the DB Support; and rarely if you wish to test JDBC-related codes, you should use mocks instead of testing the DBs themselves, and focus on only the DAOs.



Mock object types

There are three types of Mocks developers can use.

1. Pre-existing Mocks
   Web containers, DBs, mail servers under JEE environment are commonly-used, but difficult to test. There are some Mock environments or objects that are already implemented as open-source, or commercial, softwares. In our case, Spring already provides a variety of Mock objects and utilities to help you implement your test classes.
   ex) Spring Test (web, JNDI), Mockrunner [http://mockrunner.sourceforge.net/]
   ✓ Refer to separate instructions on how to use them.
2. Mock libraries
   If there are relatively simple classes or interfaces, you can quickly create mock objects inside your test cases for them so that you can easily reduce dependencies or prevent compilation errors, such as open-source [http://www.mockobjects.com/]. Our IDE uses EasyMock [http://www.easymock.org/].
   ex) http://www.mockobjects.com/ [http://www.mockobjects.com/], EasyMock [http://www.easymock.org/], JMock [http://www.jmock.org/], Mockito [http://code.google.com/p/mockito/] etc.
   ✓ Refer to Mock library guidelines for further details.
3. Implementing collaborators for using as mocks

   You can create your own class or test environment for mock purposes.

   ex) Commons-mail Mock, SMS Mock etc.
   ✓ Refer to Creating Collaborators for Mock purposes guildeline.

Environmental settings

Identical to setting unit test environment.

Manual

Sample code that shows you how to use Mock objects is included in the guide program. The following is the explanations of how each mock scenario can be described in each of the samples.

Pre-existing Mocks

This includes JDBC API Mocks, and Servlet API Mocks. This section will explain those mocks provided by the Spring Test. For further details, refer to the Spring Reference - 8.Testing [http://static.springframework.org/spring/docs/2.5.6/reference/testing.html] and Spring 2.5.6 API [http://static.springframework.org/spring/docs/2.5.6/api/] manuals.

Servlet API Mock
Lets you focus on unit tests by providing mocks for Servlet APIs such as web container's HTTP request/response, or HTTP Session.

JNDI Mock
Provides JNDI registration/binding/lookup/release and other features so that you can use JNDI services without the containers such as WAS, if you are testing classes that use JNDI lookup.

Using Servlet API Mock

Web containers are more or less required for testing controllers. For this, Spring Test provides Servlet API Mock objects. The entire sample is accessible in the HttpRequestMockTest source.

Suppose that you are testing a servlet or controller class such as below.

```
public class Servlet extends HttpServlet {

    public void service(HttpServletRequest request, HttpServletResponse response)
            throws ServletException {
        request.setAttribute("res", this.getClass().getSimpleName());
    }
}
```

You cannot prepare the classes such as HttpServletRequest or HttpServletResponse for testing, if you do not have web containers available. This requires a rather tedious preparation of setting up the web containers and loading them. But fortunately mock classes that already have the interfaces implemented are available for such common development scenarios.

Therefore, you can use such mock classes to write test codes for servlets.

```
private MockHttpServletRequest request;
private MockHttpServletResponse response;
```

Declare the target class as a test fixture too.

```
private Servlet servlet;
```

Use mock classes instead of using the real HttpServletRequest and HttpServletResponse classes for actual testing.

```
@Before
public void setUp() {
    servlet     = new Servlet();

    request     = new MockHttpServletRequest();
    response = new MockHttpServletResponse();
}
```

Now that the mock classes are ready, you just need to write the test codes.

```
    @Test
    public void testCallMyServlet() throws Exception {
        servlet.service(request, response);
        assertEquals(servlet.getClass().getSimpleName(), (String) request.getAttribute("res"));
```

```
}
```

Using JNDI Mock

When you use web services, or retrieve DB connection info from JNDI info, you need the WAS environment that the JNDI is registered under. For this purpose, Spring Test provides a mock object for JNDI. The sample code can be found in the JNDIMockTest source.

Suppose that you will be testing a class that retrieves DB connection info by using the following JNDI.

```
/** name */
private static final String name = "java:comp/env/jdbc/myDS";

public DataSource createDataSource() {
	DriverManagerDataSource dataSource = new DriverManagerDataSource();
	dataSource.setDriverClassName("org.hsqldb.jdbcDriver");
	dataSource.setUrl("jdbc:hsqldb:sampledb");
	dataSource.setUsername("sa");

	return dataSource;
}
<code>

(Following is the sample test code.)
<code:java>
	/**
	 * Binding Test initialization method that sets up binding with the desired object
	 */
	@BeforeClass
	public static void setUp() throws Exception {
		Object objectToBind = createDataSource();

		SimpleNamingContextBuilder.emptyActivatedContextBuilder().bind(name, objectToBind);
	}

	/**
	 * Binding 한 객 체 에 대 한 lookup 이 제 대 로 실 행 되 는 지 확 인 하 는 테 스 트
	 *
	 * @throws Exception
	 */
	@Test
	public void testBindedObjectIsNull() throws Exception {
		DataSource dataSource;

		dataSource = (DataSource) new InitialContext().lookup(name);
		Assert.assertNotNull("Binded object is dataSource.", dataSource);
	}
```

You can add more flesh to the test code once you write up the JNDI portion as above.

Using mock libraries

One of the easy ways to use readily available mock classes instead of writing them yourself just to test few of the features that you are interested in, is to use mock frameworks such as EasyMock. In this IDE, EasyMock works well along with Unitils. Refer to the EasyMock Documentation [http://www.easymock.org/Documentation.html], or other online documentations.

How to use EasyMock

This will explain how to use EasyMock with Unitils. Check the entire source of EasyMockTest source code. For mock support for Unitils, refer to Unitils Tutorial - Testing with Mock objects [http://unitils.sourceforge.net/tutorial.html#Testing_with_mock_objects], and EasyMock Support [http://unitils.sourceforge.net/tutorial.html#EasyMock_support].

The following sample is a test code for EasyMockService that uses CollaboratorDao. CollaboratorDao is the interface, and you can come up with any implementation using it.

EasyMockService will look like below. It has a selectList method and an insert method, and each will be calling CollaboratorDao's selectList and insert methods respectively.

```
@Service("easyMockService")
public class EasyMockService {

    @Resource(name="collaboratorDao")
    private CollaboratorDao collaboratorDao;

    public List<BoardVO> selectList() throws Exception {
        List<BoardVO> result =      collaboratorDao.selectList();
        return result;
    }

    public void insert(BoardVO vo) throws Exception {
        collaboratorDao.insert(vo);
    }

}
```

CollaboratorDao interface looks like below.

```
public interface CollaboratorDao {
    List<BoardVO> selectList();
    void insert(BoardVO vo);
}
```

You can write the test code as follows.

1. Declare that you will be using Unitils.

```
@RunWith(UnitilsJUnit4TestClassRunner.class)
public class EasyMockTest {
  . . .
}
```

2. Declare the Mock object.
   - **@Mock** : declare as Mock
   - **@InjectIntoByType** : CollaboratorDao injected into Service class. [3)]

```
@Mock
@InjectIntoByType
private CollaboratorDao mockDao;
```

3. Declare the test class.
   - **@TestedObject** : Test target class to be injected with mock object.

```
@TestedObject
private EasyMockService service;
```

4. Define the method to be mocked.
   Define the behaviour of the test target method's signature, and also return values if needed, and use those data to run tests.

   Following is when the mock target method does not have a return value.

```
mockDao.insert(board);
EasyMockUnitils.replay();
```

   Following is when the mock target method does have a return value.

```
expect(mockDao.selectList()).andReturn(Arrays.asList(new BoardVO(101), new BoardVO(102)));
EasyMockUnitils.replay();
```

5. Write code for running the test target method and confirming the results.

```
// Run the test target method
List<BoardVO> selectList = service.selectList();


// Check results
assertNotNull("Check test target was created", selectList);
assertPropertyLenientEquals("id", Arrays.asList(101, 102), selectList);
```

Implementing collaborator for mock purposes

The guide program has been describing how to create common mock classes and their sample codes, such as EmailMockTest and JDBCMockTest. Instead of describing every detail on how to create the mock classes, we will just list the three high-level, general steps, and the rest is identical to writing any other code.

1. Create a class that inherits the interface to be mocked.
2. Implement the methods that are to be tested out of all the interface methods.
   The extent of the implementation can be just returning the desired data, or minimal logic.

References

Mock Objects : http://www.mockobjects.com [http://www.mockobjects.com]
EasyMock : http://www.easymock.org [http://www.easymock.org]
EasyMock Documentation : http://www.easymock.org/Documentation.html [http://www.easymock.org/Documentation.html]
Spring 2.5.6 Reference - 8.Testing : http://static.springframework.org/spring/docs/2.5.6/reference/testing.html [http://static.springframework.org/spring/docs/2.5.6/reference/testing.html]
Spring 2.5.6 API : http://static.springframework.org/spring/docs/2.5.6/api/ [http://static.springframework.org/spring/docs/2.5.6/api/]
Unitils Tutorial - Testing with Mock objects : http://unitils.sourceforge.net/tutorial.html#Testing_with_mock_objects [http://unitils.sourceforge.net/tutorial.html#Testing_with_mock_objects]
Unitils Tutorial - EasyMock Support : http://unitils.sourceforge.net/tutorial.html#EasyMock_support

- [http://unitils.sourceforge.net/tutorial.html#EasyMock_support]
  Unitils Guildelines http://unitils.sourceforge.net/guidelines.html
  [http://unitils.sourceforge.net/guidelines.html]

---

1) For example, there are open-source frameworks such as Cactus that emulates JEE server environment, or Mockrunner that provides multiple mock environments.
2) If using commercial or specific S/W, implementing the right mock interface class, and using it to write test classes can remove dependencies towards them.
3) If you are not using this annotation, the Service class must contain a Dao setter method. This annotation is a Unitils annotation that lets you inject without a setter method. (Unitils Tutorial - Testing with Mock objects)

Refer to [http://unitils.sourceforge.net/tutorial.html#Testing_with_mock_objects]'s Mock injection.