Exercise 4: Unit Testing for Isolated Objects

This exercise shows the use of testing patterns such as Test Stub, Self Shunt and Mock. These patterns are used to isolate the tested objects by replacing dependencies. Once the dependencies from the tested object to other objects have been cut, the tested object can be controlled via the test case.

4.1 ObservableStubTest (4 points)

As example for a class to be tested we use *java.util.Observable,* from the Java 2 Platform API; see http://docs.oracle.com/javase/7/docs/api/java/util/Observable.html. The class is part of the Java implementation of the design pattern *Observer* and collaborates with other classes that implement the interface *java.util.Observer*.

Use JUnit to write a test *testNotifyObservers()* that should be put in the test class *at.fhhagenberg.sqe.exercise4.ObservableStubTest*. The test should assert that the *Observer* receives an *update(...)* including a notification message when *notifyObservers()* of the *Observable* is called. Make sure that the notification (1) is received **exactly once** and (2) passes the **correct parameters**.

Please note:

- The test pattern *Dummy* should be used. Develop a class *DummyObserver* that implements the interface *Observer* and registers with the *Observable*.
- Notifications are only sent when the *Observable* has been changed. The *Observable* can be marked as changed via *setChanged()*. However, this method is *protected*. Therefore *java.util.Observable* has to be sub-classed for the purpose of testing and the public method *change()* should be added to call *setChanged()*. Name the subclass *MyObservable*.
- Name the Eclipse project SQE04-*Lastname*_ObservableTests; replace *Lastname* by your last name.

4.2 ObservableShuntTest (2 points)

Create the same test as in the previous example *ObservableStubTest,* but now use the *Self Shunt* test pattern. Name the new test class *ObservableShuntTest*. Again, make sure that the notification (1) is received exactly once and (2) passes the correct parameters.

Please note:

- When using the *Self Shunt* test pattern, the test class itself acts as dummy. Therefore the test class implements the interface *Observer* and by setting the reference *this* the test class is registered as *Observer* with the *Observable*.
- This example is based on the previous example. Follow the instructions from the previous if applicable, especially the naming of the Eclipse project.

4.3 ObservableMockTest (3 points)

Create the same test as in the previous example *ObservableStubTest,* but now use the mock framework *EasyMock* (*http://www.easymock.org*) to implement the test case. Name the new test class *ObservableMockTest*. Again, make sure that the notification (1) is received exactly once and (2) passes the correct parameters.

Please note:

- The *Mock* test pattern uses stubs generated by the mock framework. It is not necessary to implement the stubs manually in form of dummies or shunts anymore.
- Add the jar file *easymock-3.4.jar* from *easymock-3.4-bundle.zip* (to be downloaded from the elearning platform or directly from the EasyMock web site) to the project's classpath (right click on the jar: *Build Path > Add to Build Path*.). Note: Avoid the use absolute paths in referencing the libraries.
- Create the mock object for the interface *Observer* by using the framework EasyMock. Detailed instructions can be found in the documentation available online at *http://www.easymock.org/user-guide.html* describing a similar notification example.
- Follow the instructions from the previous if applicable, especially the naming of the Eclipse project.

5.1 Design for Testability (1 point)

When writing a unit test is technically hard or impossible, this is often an indicator that the design of the code you want to test is not testable and, thus, should be refactored. Read the chapter on testable design from *Lasse Koskela: Effective Unit Testing (Manning Publications, 2013).* The chapter explains typical testability issues that block unit testing and provides guidelines for testable design.

Read the entire chapter. Select one of the guidelines for testable design (7.3), which you think is the most useful for you. "Promise to yourself" that you will always develop according to this guideline in future!

Which guideline have you selected? Why? Answer these questions in the text file *Exercise5-Q&A.txt*.

Submission Instructions

- Add a comment at the beginning of every source code file containing your full name and your student ID number.
- Submit the complete Eclipse project including all relevant additional files as specified via packing in a Zip archive. Adhere to the naming convention for projects: SQE##-*Lastname_Title* (## = exercise number, *Lastname* = your name, *Title* = exercise title).
- Upload the archive at the specified submission link on the elearning platform http://elearning.fh-hagenberg.at/ until the specified date and time before the next lecture.