Homework 6: JUnit

Name : _____

Student Number : _____

Laboratory Time : _____

Objectives

- Create JUnit Test Cases in Eclipse
- Create JUnit Test Suites in Eclipse
- Run JUnit Test Cases and Suites in Eclipse

Preamble

JUnit is a unit-testing framework for Java. It provides a common and reusable structure that is required for developing automate and repeatable unit tests for Java classes. JUnit provides a base class called TestCase that can be extended to create series of tests for your classes, an assertion library that can be used to evaluate the results of the tests, and test drivers, both command line and GUI based, called TestRunner to run the test cases you create.

The recent version of the Eclipse JDT already has JUnit Plug-in built in to make creating and running test cases more convenient. The plug-in includes a wizard for assisting in creating testing a test case and test suite, and an environment for running them.

In this lab, you will learn how to set up a project for creating JUnit tests. Then you will create test cases and a test suite, and run them.

Grading Checklist

By the end of the laboratory session, you need to demonstrate to the TA that you can do the following tasks. The TA will check off the items below that you have completed and collect this cover page from you.

- ☐ JUnit library is in the project's build path
- ☐ Test case for GUIEnvironment class has been created and asserts added to init method body
- ☐ The test case for GUIEnvironment runs successfully (may find errors in GUIEnvironment)
- ☐ Test case for Processor class has been created and assert added to calculate method body
- ☐ The test case for Processor runs successfully (may find errors in Processor)
- ☐ Test suite created and works

TA Initials: ____

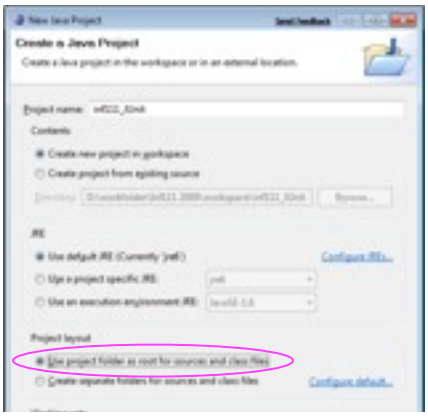
Instructions for the Laboratory

Task 1: Set up a new project and create JUnit Test Cases for the GUIEnvironment class

For this task, you will set up a new project and include the provided classes in the project. Then you will create a test case to test the GUIEnvironment class. In JUnit convention, a test class is created for every application class, and every non-trivial method is tested.

a) Download and uncompress LunarLanderHW7.zip, which contains GUIEnvironment.java and other Java files.

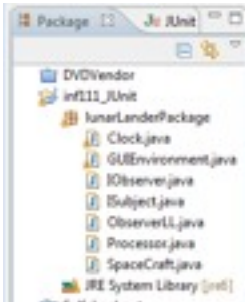
b) Create a new Java project in Eclipse called inf111_JUnit. In a New Java Project diaglog, select the option to "Use project folders as root for sources and class files"



1. Import GUIEnvironment.java and other Java files in the zip file into a default (unnamed) package. You can do this by right clicking on the project and then selecting Import → General → File System. You need to select the directory where you have Java files residing. You can use Eclipse's file filtering capabilities here.

2. Create a new package and name it lunarLanderPackage (note the lowercase 'l')

3.Use the refactoring feature to move the Java files from the default package into the lunarLanderPackage that you just created.

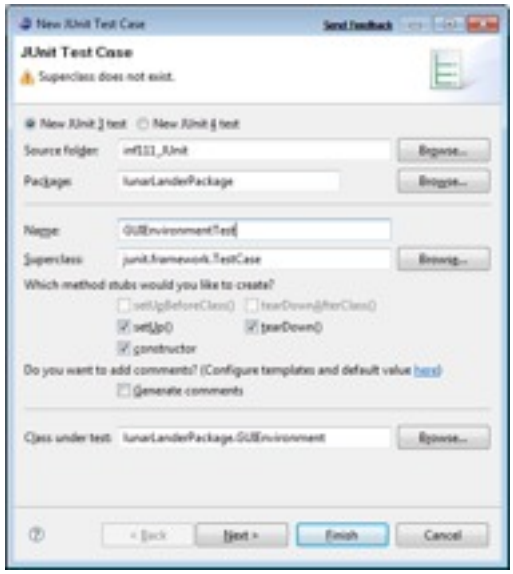


c) Create a JUnit Test Case Class.

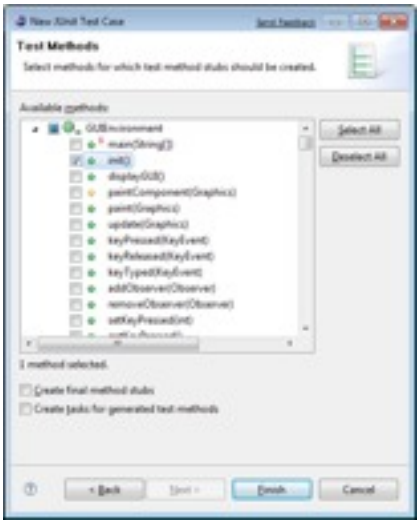
1.First, to bring up the New JUnit Test Case Wizard, select the lunarLanderPackage package. Then, select File → New → JUnit Test Case.

2.Name the class by putting the name GUIEnvironmentTest in the name test box.

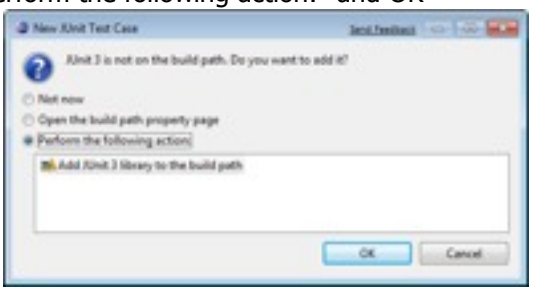
3.Select setUp() and tearDown() boxes to automatically generate skeleton of these methods. The setUp() and tearDown() methods are run before and after each test case is run.



4.For "Class under test", enter GUIEnvironment. Eclipse should find GUIEnvironment class in the lunarLanderPackage and should select it (Browse..). Then press Next.



5.Now you can select methods for which test method stubs will be created. Select init().
6.Press Finish.
7.A dialog should pop up to ask whether JUnit library should be added to the build path. Select "Perform the following action:" and OK



8.A new class, GUIEnvironmentTest, that extends junit.framework.TestCase is created with generated method stubs.

d) Implement a test case function in GUIEnvironmentTest to test the GUIEnvironment class.
1.In GUIEnvironmentTest, create a new class variable, lunarLanderEnvironment as a private variable of the type GUIEnvironment.

```
GUIEnvironment lunarLanderEnvironment;
```

2.Implement the setUp() method to initialize variables.

```
lunarLanderEnvironment = new GUIEnvironment();
```

3.Implement the tearDown() method to clear the variable.

```
lunarLanderEnvironment = null;
```

4.Implement the testInit() to check that GUIEnvironment initialization method is working correctly.

```
lunarLanderEnvironment.init();
assertFalse(lunarLanderEnvironment.getHit());
assertNotNull(lunarLanderEnvironment.getSpaceCraft());
```

5.Add in testInit() two assertions to validate that Clock and Sensor (properties of

GUIEnvironment) are not null.

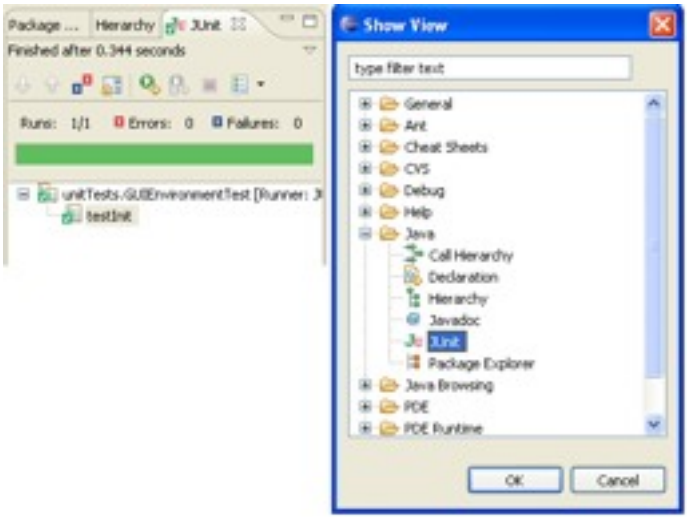
If you get a compilation error, be sure to import the required packages, including junit.framework.Assert package.

You can find more information on the assert API in the JUnit JavaDocs at http://www.junit.org/junit/javadoc/3.8.1/index.htm

Task 2: Running GUIEnvironmentTest as a JUnit test case

In this task, you will run the test case in GUIEnvironmentTest as a JUnit Test Case.

a) Right click at GUIEnvironmentTest, and select Run → JUnit Test.
b) You should see the result of your test case in JUnit view. If the view does not appear, show the JUnit view by selecting Window → Show View → Other → Java → JUnit.



Task 3: Create JUnit Test Case for the Processor class
(You may want to skip this task if you are running behind on time)

a) Now repeat Task 1, step d) to create a test case for the calculate method in the Processor class. Implement a test case function in ProcessorTest to test the Processor class.
   1. Create three new class variables, lunarLanderEnvironment as a private variable of the type GUIEnvironment, spaceCraft as a private variable of the type SpaceCraft and processor a private variable of the type Processor.

```
GUIEnvironment lunarLanderEnvironment;
SpaceCraft spaceCraft;
Processor processor;
```

   2. Implement the setUp() method to initialize variables.

```
processor = new Processor();
lunarLanderEnvironment = new GUIEnvironment();
lunarLanderEnvironment.init();
spaceCraft = lunarLanderEnvironment.getSpaceCraft();
```

```
JFrame app = new JFrame();
app.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
app.getContentPane().add (lunarLanderEnvironment);
app.setSize (800,800);
app.setBackground (Color.WHITE);
app.setLocation (0,0);
app.addKeyListener (lunarLanderEnvironment);
app.setVisible (true);
```

3.Implement the tearDown() method to clear the variable.

```
lunarLanderEnvironment = null;
spaceCraft = null;
processor = null;
```

4.Implement the testCalculate() to check that the calculate method is working correctly.

```
int rotation;
rotation = spaceCraft.getRotation();
//simulates right key pressed
lunarLanderEnvironment.setKeyPressed(KeyEvent.VK_RIGHT
);
processor.calculate(spaceCraft,
lunarLanderEnvironment);
assertEquals(rotation + spaceCraft.getRotationRate(),
spaceCraft.getRotation());
```

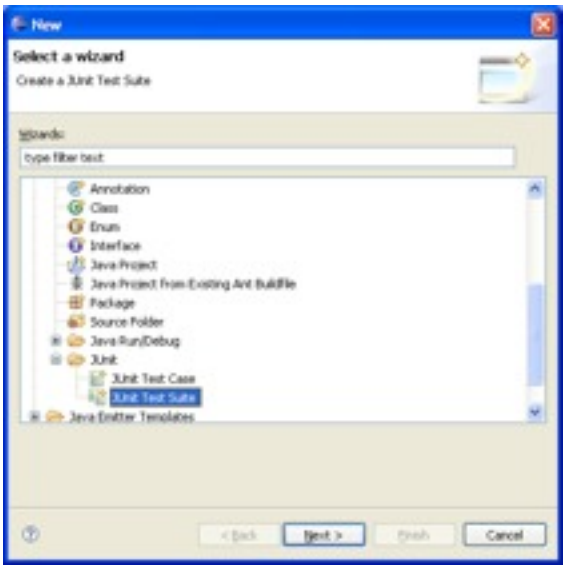
5.Run ProcessorTest as a JUnit Test Case (See Task 2).

If you get a compilation error, be sure to import the required packages, including junit.framework.Assert package. Other classes that are needed to be imported are `java.awt.event.KeyEvent`, `java.awt.Color and javax.swing.JFrame` class.

Task 4: Creating a Test Case Suite

In this task, you will create a test case suite for the test cases created in Task 1 and Task3. A test case suite allows more convenient test case execution and will allow you to run all your test cases at once.

a) Create a Test Suite by selecting the lunarLanderPackage package, and then selecting File→ New → Other → Java → JUnit-> JUnit Test Suite.

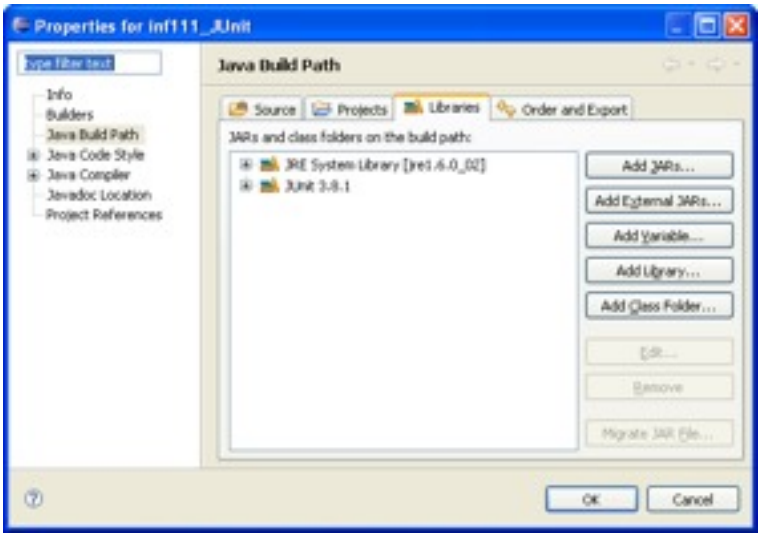


b) In the next screen, name the Test Suite "LunarLanderAllTests", and include your test cases created in the Task 1 and Task 3 (GUIEnvironmentTest and also ProcessorTest if you had time) into the test suite by selecting the check boxes in front of the appropriate classes. Then click Finish.
c) Run the test suite by selecting the Test Suite class, and then Right Click → Run → JUnit Test

Appendix

You can also manually add the JUnit library, junit.jar, to the project's build path using the following steps.

a) Right click the project and select Properties. The properties dialog box as shown below should appear.



b) Select Java Build Path on the left panel and bring the Libraries tab forward.
c) Click on "Add External JARs…" button.
d) In the JAR Selection dialog box, find the plugins directory under Eclipse's installation directory (C:\Opt\eclipse). Then locate the JUnit folder. The current version of JUnit should be org.junit_3.8.1. From that directory, select junit.jar, as in the figure below.

Testing (90 points)

In this part of the homework, you will be finishing off the test suite for the Self-Check Out code. Some test cases have been provided for you. There are three use cases at the end of this document that specify the expected behavior of the software. You should cover all the sub-variations in your integration test cases.

1. Unit Tests. (20 points) Create JUnit tests for the following classes:

- UPC.java
- PackagedProduct.java

Be sure to test with a variety of valid and invalid UPC codes. As well, these tests should be in the same package as the application code. You can use the tests classes BulkProductTest.java and BICTest.java as examples to implement your own unit tests.

2. Integration Tests. (60 points) Create the following integration tests for the system:

- Adding packaged products
- Sequences of adding and bagging events
- Adding combinations of bulk and packaged items
- Purchasing large numbers of items

The first test can be added to the SelfCheckOutTest.java class in the IntegrationTest package. The remaining tests should be put in the same package using as many or as few additional classes as you desire.

-

3. Test Suite. (5 points) Create a test suite so that all the tests will run automatically. You should include all the test cases that were provided and the unit and integration tests that you created.

4. Commenting the Code. (5 points) Document your tests cases using informative, high-quality comments in the code. You should explain what you are testing for and how you are going about doing so. Where appropriate, you should include Javadoc comments.

Deliverables

Your deliverables are your JUnit test cases and a written report describing your changes. Be sure to include a list of classes and methods that were changed or added, so the TA can examine your work.

Handing In Your Assignment

Your assignment must be submitted electronically to checkmate.ics.uci.edu. You will submit **two** files.

1. Your report in Report.doc or Report.pdf. The report should describe your changes to the code and any other information that you want the TA to know about. In other words, if you want credit for your work, you should describe it here.
2. A zip file containing all of the application and test code for the Self-Check Out system.

Do not zip these two files into one big .zip file.

USE CASE 1	Self-Check Out	
Goal in Context	Buy packaged and bulk items using the Self-Check Out system	
Preconditions	User has chosen the packaged and bulk item that will buy	
Success End Condition	User entered all the items, bagged them and paid for them	
Failed End Condition	User could not enter all the items in the system and could not pay for them	
Primary, secondary Actors	User	
Trigger	User starts the transaction	
DESCRIPTION	**Step**	**Action**
	1	Per each item:
	1.1	User adds item
	1.2	User bags item
	2	User pays for items
	3	The system prints receipt
EXTENSIONS	**Step**	**Branching Action**
	1.1a	User adds a packaged item: Extend use case: 2. Add Packaged Item
	1.1b	User adds a bulk item: Extend use case: 3. Add Bulk Item
SUB-VARIATIONS		**Branching Action**
	1.1	If the previous item has not been bagged, the system will not allow to add the item and will show an exception
	1.1	If the user just paid for items, the system will not allow the user to add an item and will show an exception. The user should start a new transaction
	1.2	If the new item has already been bagged, the system will not allow to bag the item again and will show an exception
	2	If no items have been added, the system will not allow to pay and will show an exception
	2	If there is any problem with the payment, the system will show an exception
	3	If there is any problem while printing the receipt, the system will show an exception

USE CASE 2	Add Packaged Item	
Goal in Context	Add a packaged item to the transaction	
Preconditions	User started the transaction	
Success End Condition	Packaged item is added to the transaction	
Failed End Condition	Packaged item is not added to the transaction	
Primary, secondary Actors	User	
Trigger	User adds a packaged item	
DESCRIPTION	**Step**	**Action**
	1	The system requests the Universal Product Code
	2	User enters the Universal Product Code
	3	The system validates the Universal Product Code
	4	The system looks for the Universal Product Code in the Database of products
	5	The system adds the item to the transaction
	6	Add cost to total cost and weight to total weight
EXTENSIONS	**Step**	**Branching Action**
SUB-VARIATIONS		**Branching Action**
	3	If the value of UPC is null, the system will show an exception
	3	If the length of UPC is different than 12 digits, the system will show an exception
	3	If the checksum of UPC is not valid, the system will show an exception
	4	If the product is not found in the Database, the system will show an exception

USE CASE 3	Add Bulk Item	
Goal in Context	Add a bulk item to the transaction	
Preconditions	User started the transaction	
Success End Condition	Bulk item is added to the transaction	
Failed End Condition	Bulk item is not added to the transaction	
Primary, secondary Actors	User	
Trigger	User adds a bulk item	
DESCRIPTION	**Step**	**Action**
	1	The system requests the Bulk Item Code
	2	User enters the Bulk Item Code
	3	The system requests the weight for the bulk item
	4	User enters the weight for the bulk item
	5	The system validates the Bulk Item Code
	6	The system looks for the Bulk Item in the Database of products
	7	The system validates the weight for the bulk item
	8	The system adds the item to the transaction
	9	Calculate partial cost for the bulk item. Multiply the price by weight
	10	Add partial cost to total cost and weight to total weight
EXTENSIONS	**Step**	**Branching Action**
SUB-VARIATIONS		**Branching Action**
	5	If the value of BIC is null, the system will show an exception
	5	If the length of BIC is different than 5 digits, the system will show an exception
	6	If the product is not found in the Database, the system will show an exception
	7	If the value of weight is null, the system will show an exception
	7	If the value of weight is not a double number properly formatted, the system will show an exception