Lab: Skills - JUnit and EclEmma in Eclipse

Getting Ready: Before going any further, you should:

1. Setup your development environment.

2. Download the following files:
   GiftCard.java
   to an appropriate directory/folder. (In most browsers/OSs, the easiest way to do this is by right-clicking/control-clicking on each of the links above.)

3. If you don't already have one from earlier in the semester, create a project named eclipseskills.

4. Drag the file GiftCard.java into the default package (using the "Copy files" option).

5. Open GiftCard.java.

Part 1. JUnit Basics: JUnit is an open-source testing framework. It provides a way to write, organize, and run repeatable test. This part of the lab will help you become familiar with JUnit.

1. Create an empty JUnit test named GiftCardTest in the default package by clicking File-New-JUnit Test Case
   Use the most recent version of JUnit; if necessary, add JUnit to the build path when asked.

   Note: Normally, you should put tests in their own package(s). To keep things simple, we will break that rule.

2. Copy the following code into GiftCardTest.

```
import static org.junit.Assert.assertEquals;
import org.junit.Test;
import org.junit.rules.ExpectedException;


public class GiftCardTest
{
   @Test
   public void getIssuingStore()
   {
      double       balance;
      GiftCard     card;
      int          issuingStore;

      issuingStore = 1337;
      balance      = 100.00;
      card = new GiftCard(issuingStore, balance);

      assertEquals("getIssuingStore()",
                issuingStore, card.getIssuingStore());
   }
}
```

3. A JUnit test suite is a class, much like any other class. Tests are methods that are preceded with the annotation @Test. (Note: An annotation provides information about a program but is not part of the program. Annotations have no effect on the operation of the program. Instead, they are used to provide information to tools that might use the program as input.)
   How many tests are in the test suite GiftCardTest?

   There is only one test in the GiftCardTest suite.

4. JUnit has an Assertclass that has a static assertEquals()method with the following signature that is used to compare expected and actual results:
public static void assertEquals(String description, int expected, int actual)

where descriptionis a human-readable Stringdescribing the test, expectedis the expected result, and actualis the result of actually running the code being tested.

How would you call this method and pass it the String"getIssuingStore()", the intissuingStore, and the intreturned by the cardobject's getIssuingStore()method?

Assert.*assertEquals*("getIssuingStore()", issuingStore, card.getIssuingStore());

5. How is this method actually being called in GiftCardTest?

assertEquals("getIssuingStore()", issuingStore, card.getIssuingStore());

6. Why isn't the class name needed?

The assertequals method is imported with the static keyword, which means that you can leave out the class name if it's a static method you're calling.

7. Execute GiftCardTest. Why is it somewhat surprising that you can execute GiftCardTest without a main method?

It's surprising because we've always needed a main method somewhere to run a program.

8. What output was generated?

A window opened that says that the tests ran successfully and the time that it took for the test to run. There was no visible output aside from that.

9. To see what happens when a test fails, modify the getIssuingStore() method in the GiftCard class so that it returns issuingStore + 1, compile GiftCard.java, and re-run the test suite.
Now what happens?

Running the test suite results in the window showing that the test failed. It says that there were 0 errors and 1 failures.

10. In the "Failure Trace", interpret the line:
java.lang.AssertionError: getIssuingStore() expected:<1337> but was:<1338>
Note: You may have to scroll the "Failure Trace" window to the right to see the whole message.

It's saying that when getIssuingStore() was run, it was supposed to return 1337 but instead returned 1338.

11. What mechanism is JUnit using to indicate an abnormal return?

Junit throws an exception.

12. Before you forget, correct the fault in the getIssuingStore() method.

13. The Assert class in JUnit also has a static assertEquals() method with the following signature:
public static void assertEquals(String description, double expected, double actual, double tolerance)
where tolerance determines how close to double values have to be in order to be considered "approximately equal".

Add a test named getBalance() that includes a call to assertEquals() that can be used to test the getBalance() method in the card class (with a tolerance of 0.001).

```
@Test
public void getBalance()
{
    double balance;
    GiftCard card;
    int issuingStore;

    issuingStore = 1337;
    balance = 100.00;
    card = new GiftCard(issuingStore, balance);

    assertEquals("getBalance()",balance,card.getBalance(), 0.001);
}
```

14. How many tests are in your test suite now?

There are now two tests in my test suite.

15. Suppose you had put both calls to assertEquals() in one method (named, say, getIssuingStore()). How many tests would be in your test suite?

There would technically be two tests inside of one "test" method. Each call to assertEquals() is just testing different

values so it depends on what you consider a test.

16. Re-compile and re-execute the test suite. How many tests were run?

Two tests were run.

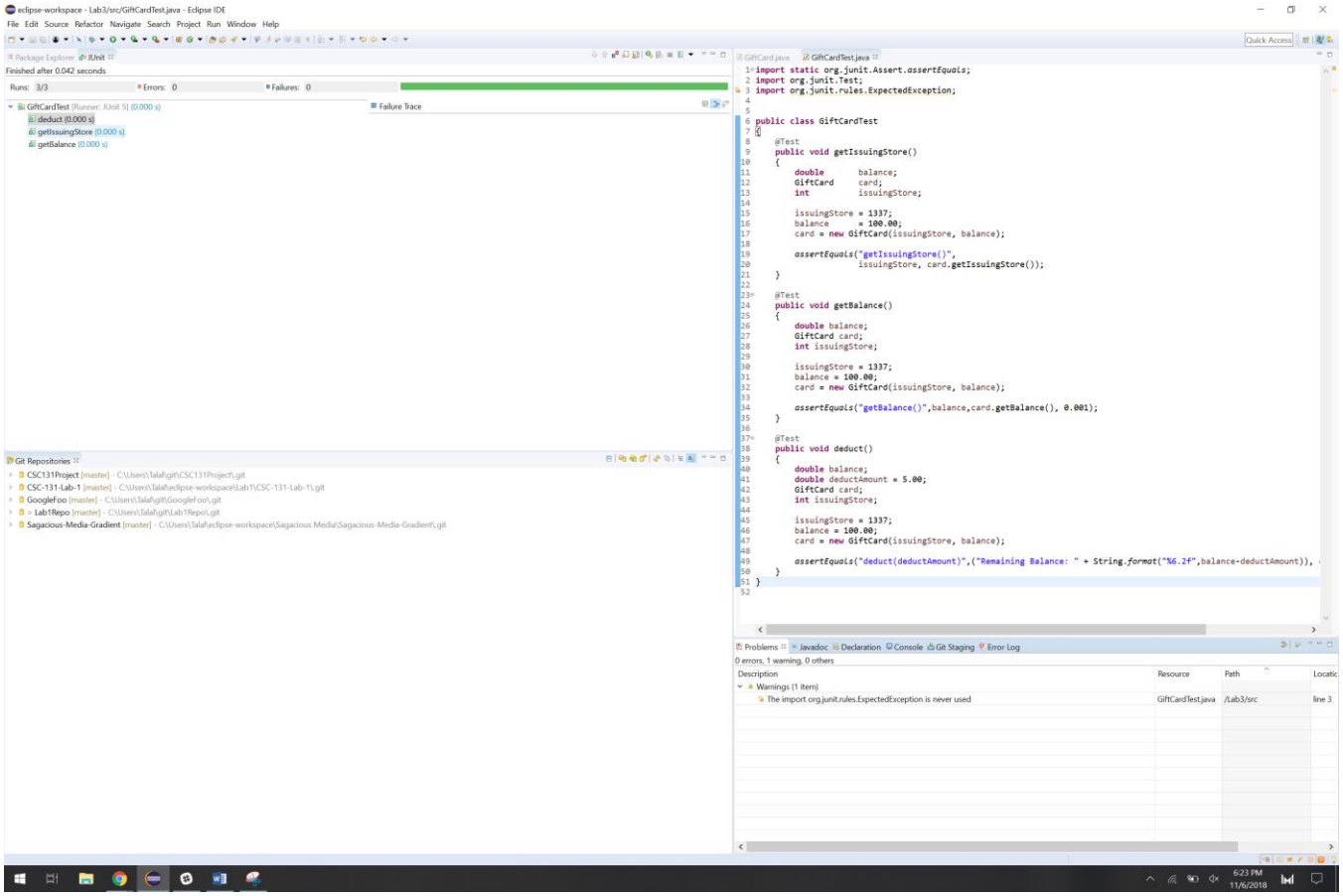
17. The Assert class in JUnit also has a static assertEquals() method with the following signature:
public static void assertEquals(String description, String expected, String actual)
Using JUnit terminology, add a test named deduct() to your test suite that can be used to test the deduct() method in the GiftCard class. Note: Be careful, the variables that are declared in the getIssuingStore() method are local.

```
    @Test
    public void deduct()
    {
        double balance;
        double deductAmount = 5.00;
        GiftCard card;
        int issuingStore;

        issuingStore = 1337;
        balance = 100.00;
        card = new GiftCard(issuingStore, balance);

        assertEquals("deduct(deductAmount)",("Remaining Balance: " + String.format("%6.2f",balance-deductAmount)), card.deduct(deductAmount));
    }
}
```

18. Execute the test suite.


eclipse-workspace - Lab3/src/GiftCardTest.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Finished after 0.042 seconds
Runs: 3/3
Errors: 0
Failures: 0
GiftCardTest (Runner: JUnit 5) (0.000 s)
deduct (0.000 s)
getIssuingStore (0.000 s)
getBalance (0.000 s)
Failure Trace
Git Repositories
CSC131Project [master] - C:\Users\Talal\git\CSC131Project\.git
CSC-131-Lab-1 [master] - C:\Users\Talal\eclipse-workspace\Lab1\CSC-131-Lab-1\.git
GoogleFoo [master] - C:\Users\Talal\git\GoogleFoo\.git
Lab1Repo [master] - C:\Users\Talal\git\Lab1Repo\.git
Sagacious-Media-Gradient [master] - C:\Users\Talal\eclipse-workspace\Sagacious Media\Sagacious-Media-Gradient\.git
GiftCard.java
GiftCardTest.java
1 import static org.junit.Assert.assertEquals;
2 import org.junit.Test;
3 import org.junit.rules.ExpectedException;
4
5
6 public class GiftCardTest
7 {
8 @Test
9 public void getIssuingStore()
10 {
11 double balance;
12 GiftCard card;
13 int issuingStore;
14
15 issuingStore = 1337;
16 balance = 100.00;
17 card = new GiftCard(issuingStore, balance);
18
19 assertEquals("getIssuingStore()",
20 issuingStore, card.getIssuingStore());
21 }
22
23 @Test
24 public void getBalance()
25 {
26 double balance;
27 GiftCard card;
28 int issuingStore;
29
30 issuingStore = 1337;
31 balance = 100.00;
32 card = new GiftCard(issuingStore, balance);
33
34 assertEquals("getBalance()",balance,card.getBalance(), 0.001);
35 }
36
37 @Test
38 public void deduct()
39 {
40 double balance;
41 double deductAmount = 5.00;
42 GiftCard card;
43 int issuingStore;
44
45 issuingStore = 1337;
46 balance = 100.00;
47 card = new GiftCard(issuingStore, balance);
48
49 assertEquals("deduct(deductAmount)",("Remaining Balance: " + String.format("%6.2f",balance-deductAmount)),
50 }
51 }
52
Problems
Javadoc
Declaration
Console
Git Staging
Error Log
0 errors, 1 warning, 0 others
Description
Resource
Path
Locatic
Warnings (1 item)
The import org.junit.rules.ExpectedException is never used
GiftCardTest.java
/Lab3/src
line 3
6:23 PM
11/6/2018

Part 2. Coverage: This part of the lab will help you understand coverage tools and coverage metrics.

1. Read Eclipse_Eclemma.pdf.
2. Run GiftCardTest using EclEmma, click on the "Coverage" tab and expand the directories/packages until you can see GiftCard.java and GiftCardTest.java.
   Why is the coverage of GiftCardTest.java 100% and why is this uninteresting/unimportant?

   The coverage of GiftCardTest.java being 100% means that everything was tested successfully in GiftCardTest.java. All the tests within GiftCardTest.java ran. This is not unsurprising as there were no errors in the tests that were written.

3. How many of the tests passed?

   All three tests passed

4. Does this mean that the GiftCard class is correct?

This doesn't ensure the GiftCard class is correct as not every possible input isn't being tested.

5. What is the statement coverage for GiftCard.java?

   The statement coverage for GiftCard.java is only 55.7%

6. Select the tab containing the source code for GiftCard.java. What is different about it?

   Different parts of the code are highlighted in different colors.

7. What do you think it means when a statement is highlighted in red?

I think it means that statement was not tested.

8. Hover your mouse over the icon to the left of the first if statement in the constructor. What information appears?

   It says 2 of 4 branches missed.

9. Add tests to your test suite so that it covers all of the statements and branches in the deduct() method in the GiftCard class.

```
    }

    @Test
    public void deductInvalidTransaction()
    {
        double balance;
        double deductAmount = 5.00;
        GiftCard card;
        int issuingStore;

        issuingStore = 1337;
        balance = 100.00;
        card = new GiftCard(issuingStore, balance);

        assertEquals("deducting -1 dollars","Invalid Transaction",card.deduct(-1.0));
    }

    @Test
    public void deductBalanceLessThanZero()
    {
        double balance;
        double deductAmount = 5.00;
        GiftCard card;
        int issuingStore;

        issuingStore = 1337;
        balance = 100.00;
        card = new GiftCard(issuingStore, balance);

        assertEquals("deducting $101 from $100",("Amount Due: " + String.format("%6.2f"),card.deduct(101.0));
    }
}
```

10. Your test suite still does not cover every statement in the GiftCard class. What is different about the statements that remain untested?

The only statements that remain untested are the ones that throw IllegalArgumentException

Part 3. Testing Methods that Throw Exceptions: This part of the lab will help you learn how to test methods that throw exceptions.

1. The easiest (though not the most flexible) way to test for exceptions is to use the optional expected parameter of the @Test annotation. For example, add the following test to your test suite.

```
@Test(expected = IllegalArgumentException.class)
public void constructor_IncorrectBalance() throws IllegalArgumentException
{
    new GiftCard(1, -100.00);
}
```

   Note: IllegalArgumentException is an unchecked exception. Hence, the code will compile even if it isn't re-thrown. If you are testing for a checked exception then the method must specify the exception.

2. Add a test to your test suite named constructor_IncorrectID_Low() that covers the case when the storeID is less than 0.

```
@Test(expected = IllegalArgumentException.class)
public void constructor_IncorrectID_Low()
{
    new GiftCard(-1, 100.00);
}
```

Part 4. Coverage and Completeness: This part of the lab will help you better understand code coverage and the completeness of test suites.

1. Run EclEmman on your current test suite. What is the statement coverage for GiftCard.java now?

   The statement coverage for GiftCard.java is 100%

2. What branch does the test suite fail to test?

   It fails to test constructor_IncorrectBalance() and constructor_IncorrectID_Low()

3. Add a test to your test suite named constructor_IncorrectID_High() that covers the other branch.

```
@Test(expected = IllegalArgumentException.class)
public void constructor_IncorrectID_High()
{
    new GiftCard(1000, 100.00);
}
```

4. Run EclEmma. What is the branch coverage now?

   The first branch is not covered. For some reason, my constructor_IncorrectID_Low() and constructor_IncorrectBalance() are highlighted as red.

5. From a "white box" testing perspective, is your test suite complete? Conversely, can you think of tests that should be added?

I think it's not complete since the two constructor tests did not work. This resulted in a whole branch being untestable.

6. From a "black box" testing perspective, is your test suite complete? Conversely, can you think of tests that could be added?

From a black box perspective, it is not complete as you can still attempt a number of different inputs to attempt to find out what causes a bug in the program.