

Lab: Gaining Experience with GitLab, Junit and Eclemma

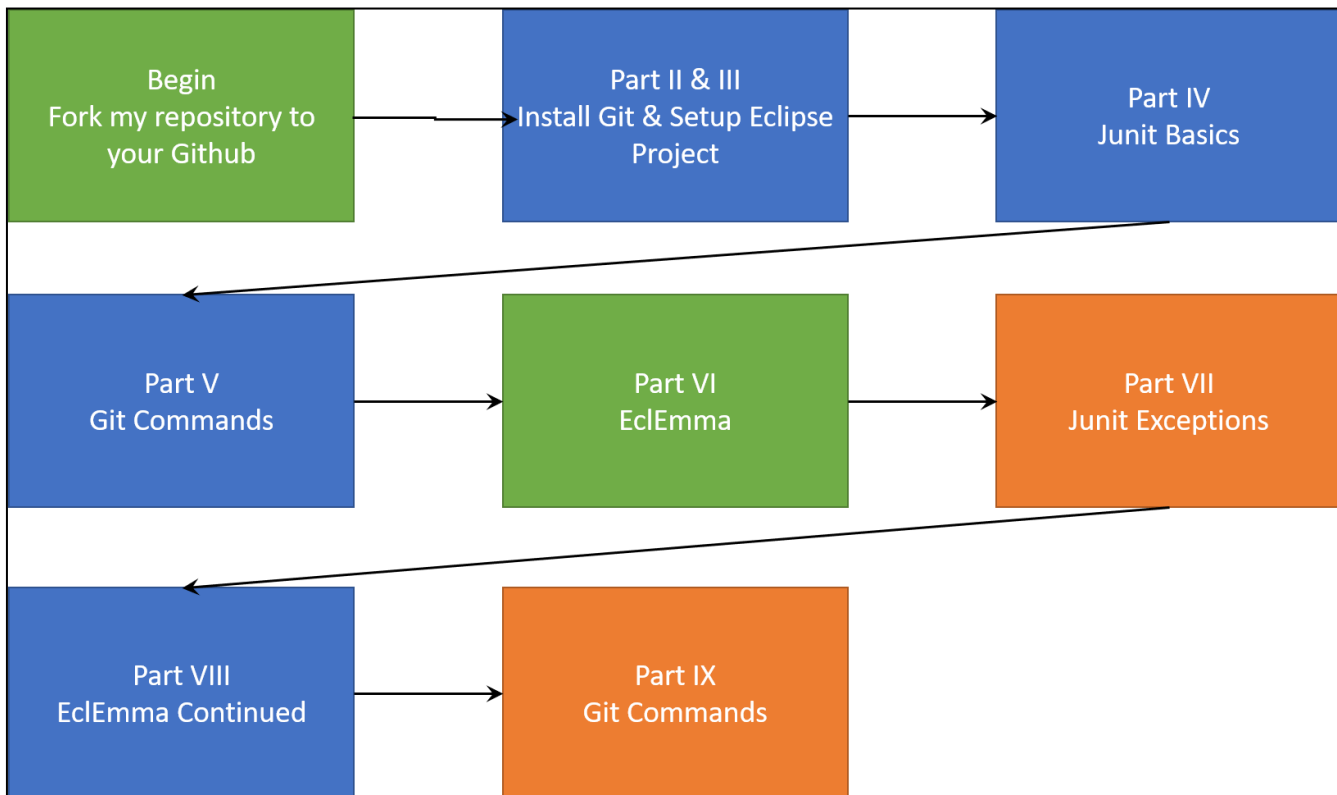
Required:

Internet Connection

Eclipse IDE for **Enterprise Java Developers** (can download here: <https://www.eclipse.org/downloads/packages/>)

See the accompanying Lab video.

This is a multi-part lab that you can complete in sections, over time. Each section is a predecessor to the next. Fill in the answers to questions posed throughout the lab.



Overview of this Lab

Instructions:

Part I: Fork the Git Repository

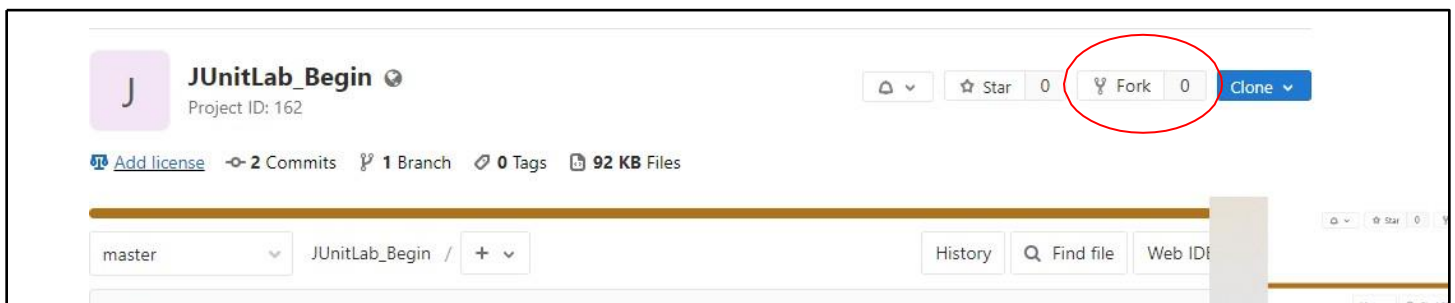
For this lab, you will use github.com. CSUS also has gitlab.ecs.csus.edu.

If you do not have an account, create one on github.com

1. Login to your account and goto:

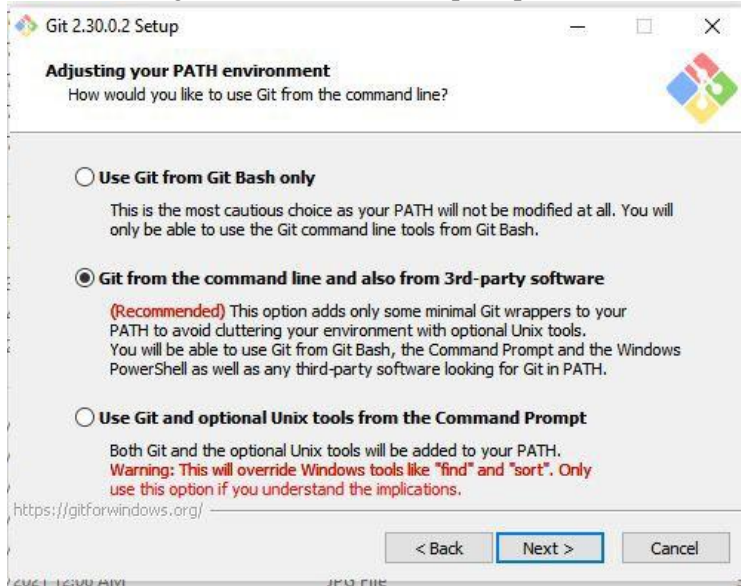
Github.com: https://github.com/azizijonespenn2/JUnitLab_Begin.git

2. Fork the repository: This will create a copy of it in your own account location.



Part II: Introduction to Git

1. Download Git from <https://git-scm.com/downloads>
2. Install Git on your local machine. Accept the defaults. Make sure you add Git to your path so that you will be able to execute git commands from the prompt.



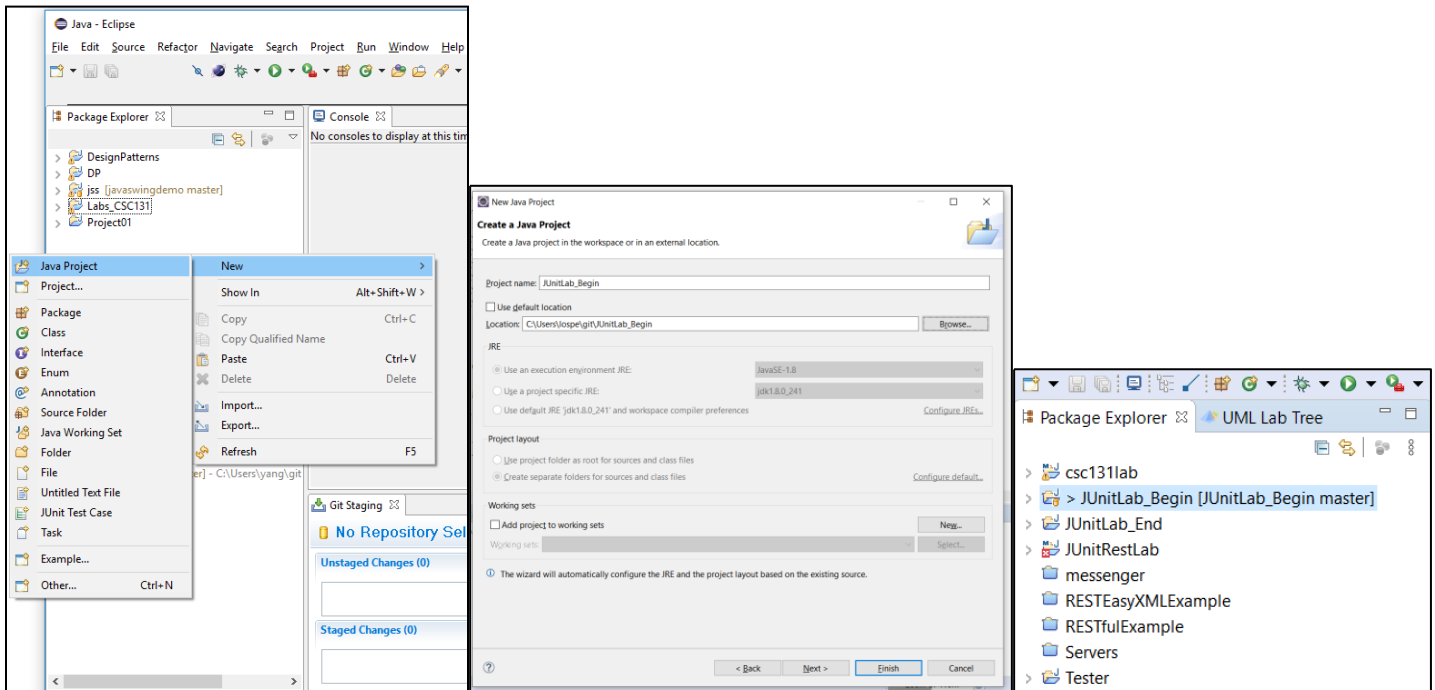
3. Open Windows CMD prompt or Windows Powershell.
4. Choose or create a directory to house your local repository. Navigate to this directory.
5. Clone **your** remote repository. In this example, the student is Aliyah Penn (aliyhpenn)

```
c:\>mkdir gitbase  
  
c:\>cd gitbase  
  
c:\gitbase>git clone https://github.com/aliyhpenn/JUnitLab_Begin.git
```

6. Now that you have cloned your remote repo to your local repo, proceed with the Junit Test portion of the lab.

Part III: Creating a Java Project using the Cloned Repository

5. In *Package Explorer*, create a *New* → *Java Project*. Name the new project in the next screen. Uncheck the option “*Use default location*” since you will use the location of your local Git repository. Then, (click) *Browse* and locate the folder that houses your local repository e.g. *git* → *JUnitLab_Begin*. Click *OK* → Click *finish*. You will see a new java project that appears in the *Package Explorer*, named *JUnitLab_Begin*.



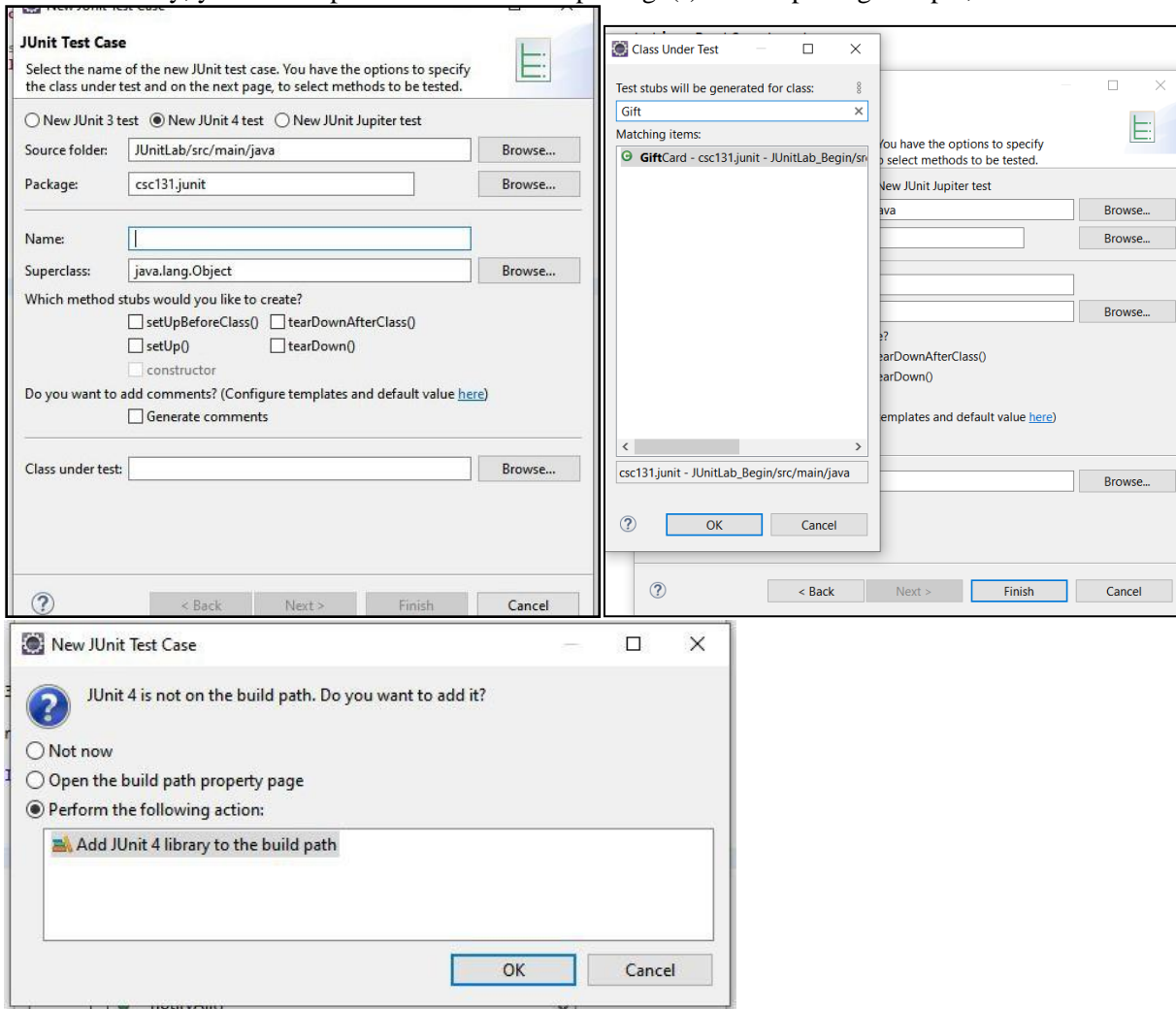
1. New Java Project, 2. Use the local Git repository as the project location 3. Project appears in your package explorer.

Part IV: JUnit Basics.

JUnit Basics: JUnit is an open-source testing framework. It provides a way to write, organize, and run repeatable tests. This part of the lab will help you become familiar with JUnit. [If you do not already have the code from the Git lab exercises, clone the repository located at: https://github.com/azizijonespenn2/JUnitLab_Begin.git to get the source code for this lab]

1. Create an empty JUnit test named GiftCardTest in the csc131.junit package by clicking on:
 - File->New->Other->Java->JUnit->JUnitTestCase.
2. Use the Junit4 version.
3. Click Browse for the Class under test: field and start typing in the GiftCard.java. When the class is found, select it and click OK.;Then click Finish. 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.



4. Type 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());
    }
}
```

5. 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`?

One test, `getIssuingStore()`

6. JUnit has an `Assert` class 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 `description` is a human-readable `String` describing the test, `expected` is the expected result, and `actual` is the result of actually running the code being tested.

How would you call this method and pass it the `String` `"getIssuingStore()"`, the `int` `issuingStore`, and the `int` returned by the `card` object's `getIssuingStore()` method?

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

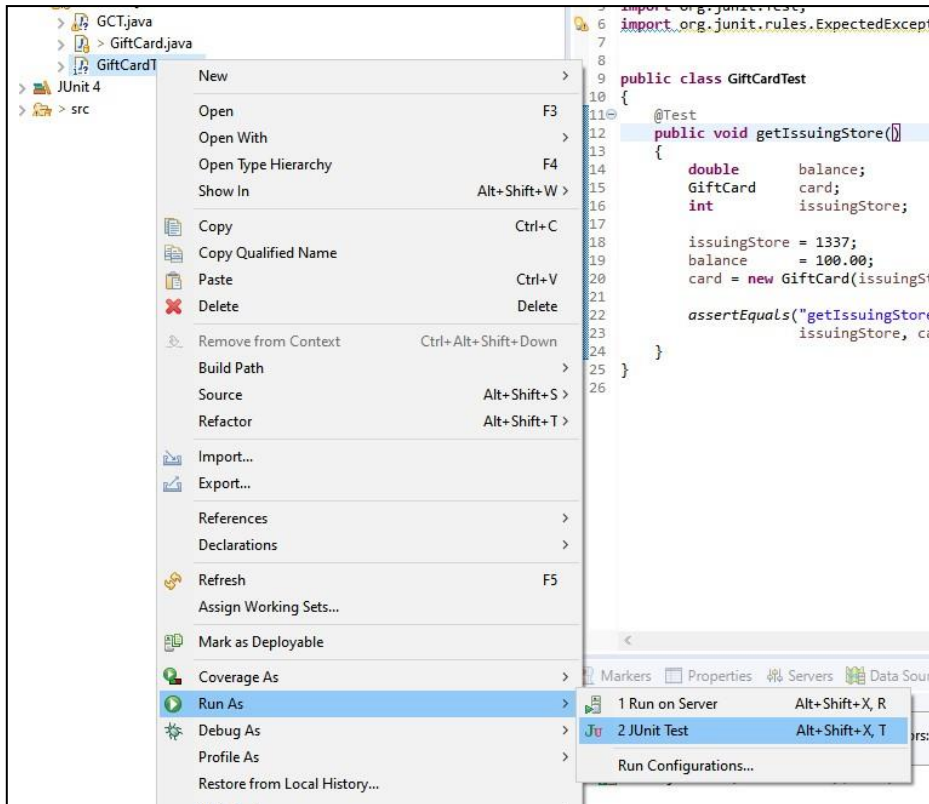
7. How is this method actually being called in `GiftCardTest`?

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

8. Why isn't the class name needed?

Because we import "import static org.junit.Assert.assertEquals;", it's static modifier and it can seem to be a class we declared. Then, the compiler can understand that the name class can be reduced when we declared static method before.

9. Execute GiftCardTest. (Right Click on the File in Project Explorer and select Run As->JUnit Test



10. What output was generated?

Nothing, but we can see the test results in a new window. It shows us a test suite and if our code passed or not.

11. 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?

The test fails

12. 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.

The test "getIssuingStore()" was run, but the expected value is 1337 not 1338

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

It's throwing an exception

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

15. 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).

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

2

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

Only one test. Suppose we put both calls to `assertEquals()`, we just check different cases in one method

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

2, both of them passed

19. 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_RemainingBalance()` 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. Copy and paste the code for the `deduct` test here:

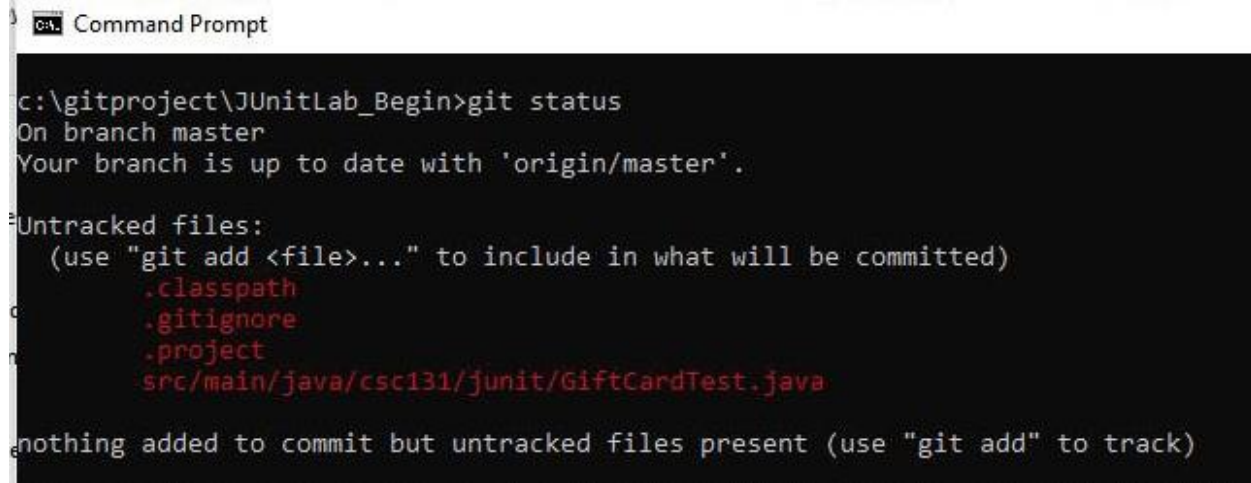
```
@Test
public void deduct_RemainingBalance()
{
    double balance;
    GiftCard card;
    int issuingStore;
    String s;

    issuingStore = 1337;
    balance = 100.00;
    card = new GiftCard(issuingStore, balance);
    s = "Remaining Balance: " + String.format("%.2f", 90.0);

    assertEquals("deduct(10.00)", s, card.deduct(10.0));
}
```

Part V: Git Commit, Push,

1. You now have a new file, GiftCardTest.java to commit to your repository. You need to mark it to be staged to your local repository.
2. Open a Command Prompt or Powershell and navigate to your project's directory.
3. Type “git status” to see what state the files are in. You should see GiftCardTest.java listed as an untracked file.



```
Git - Command Prompt

c:\gitproject\JUnitLab_Begin>git status
On branch master
Your branch is up to date with 'origin/master'.

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        .classpath
        .gitignore
        .project
        src/main/java/csc131/junit/GiftCardTest.java

nothing added to commit but untracked files present (use "git add" to track)
```

4. Execute “git add <filename>” to stage your file(s). Then run “git status” again to see what happened.


```
c:\gitproject\JUnitLab_Begin>git add src/main/java/csc131/junit/GiftCardTest.java

c:\gitproject\JUnitLab_Begin>git status
On branch master
Your branch is up to date with 'origin/master'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:   src/main/java/csc131/junit/GiftCardTest.java

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    .classpath
    .gitignore
    .project
```

5. Now Commit the File to your local repository by typing “git commit”. What happens when you execute this command?
*** Please tell me who you are.

Run

```
git config --global user.email "you@example.com"
git config --global user.name "Your Name"
```

to set your account's default identity.

Omit --global to set the identity only in this repository.

fatal: unable to auto-detect email address (got 'ptanh@MSI.(none)')

Add a commit message to describe your changes. Depending on your OS, Git may utilize a VIM type editor for you to enter your commit message.

6. Push the changes to your remote repository

7. You probably still have a few files that are marked as untracked. Some are local files specific to your Eclipse installation. These files should not be pushed to a shared repository since other developers will have different settings. Since you don't wish to commit these files AND don't want them to continue to show up as untracked, what do you need to do to have Git ignore them?

I need to use “.gitignore” command to ignore files I don't need.

Part VI: EclEmma

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.

GiftCardTest (Mar 3, 2021 8:52:09 PM)				
Element	Coverage	Covered Instructions	Uncovered Instructions	Total Instructions
> JUnitLab	73.4 %	130	47	177

Test are executed

3. How many of the tests passed?

3

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

Yes

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

73.4%. I think 3 tests were passed. It should be 100%.

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

Even though tests were executed, some statements were not covered.

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

1 of 2 branches missed.

8. 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 deduct_AmountDue()
{
    double balance;
    GiftCard card;
    int issuingStore;
    String s;

    issuingStore = 1337;
    balance = 100.00;
    card = new GiftCard(issuingStore, balance);
    s = "Amount Due: " + String.format("%6.2f", 10.0);

    assertEquals("deduct 110 from 100", s, card.deduct(110.0));
}
```

```
@Test
public void deduct_InvalidTransaction()
{
    double balance;
    GiftCard card;
    int issuingStore;
    String s;

    issuingStore = 1337;
    balance = 100.00;
    card = new GiftCard(issuingStore, balance);
    s = "Invalid Transaction";

    assertEquals("deduct -10.00 from 100", s, card.deduct(-10.0));
}
```

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

Those statements are in exception.

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

JUnit has an `Assertions` class that has a static `assertThrows()` method with the following signature that is used to test expected exceptions. The signature is:

```
public static <T extends Throwable> T assertThrows(Class<T> expectedType, Executable executable)
```

1. Create another `@Test` annotated method and add the following code to it's body:

```
assertThrows(IllegalArgumentException.class, () -> {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.

```
@Test
```

```
public void constructor_IncorrectID_Low()
```

```
{
```

```
    assertThrows(IllegalArgumentException.class, () -> {new GiftCard(-1, 9999)});
```

```
}
```

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

```
@Test
```

```
public void constructor_IncorrectID_Low()
```

```
{
```

```
    assertThrows(IllegalArgumentException.class, () -> {new GiftCard(-1, 9999)});
```

```
}
```

Part VIII: EclEmma Continued

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

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

100%

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

The test suite is covering $\text{StoreID} < 0$ and $\text{StoreID} > \text{MAX_ID}$

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

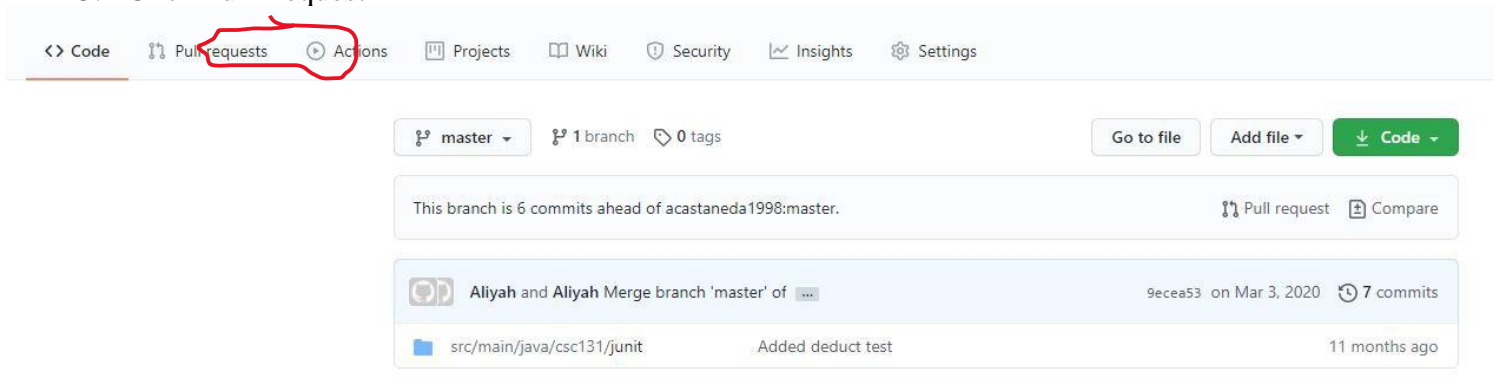
```
@Test
public void constructor_IncorrectID_High()
{
    // ...
    assertThrows(IllegalArgumentException.class, () -> {new GiftCard(100000, 100.00);});
}
```

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

All branches are covered now

Part IX: Git

1. Commit your changes to your local and push to your remote repository using the process you've already done.
2. Now, create a pull request. Go to your Github account.
3. Click Pull Request



4. Create a new pull request

