Wentworth Institute of Technology
COMP1050 – Computer Science II
Fall 2017, Wiseman

Programming Assignment 1

This lab is intended to familiarize you with the software and tools we will use in this class, as well as how you are to obtain, test, document, and submit assignments.

1. Install Software

First, download and install the following software.

1.1. Java SE Development Kit 8

The Java Standard Edition (SE) Development Kit (or JDK for short) provides the ability to run programs written in Java (including applets in a browser), as well as compile new programs. By contrast, the Java Runtime Environment (JRE) can run programs, but not to make new ones.

We will use the latest version of the Oracle JDK, version 8 (you will see JDK8 as well as 1.8, which mean the same thing). Use the search term "jdk8" to find Oracle's download page. Then, download and install the latest "update" (e.g. 8u144) for your platform (by default use 64-bit).

Java is widely used, and thus it is common for vulnerabilities to be discovered/exploited in the JDK. *You should keep your version up-to-date to limit your exposure.*

Special Notes: Mac OS
Some Java-based programs may require you to install Apple's JDK6 in parallel with JDK8. Given the security warning above, avoid this situation if possible, and see Apple's website for instructions and a download link.

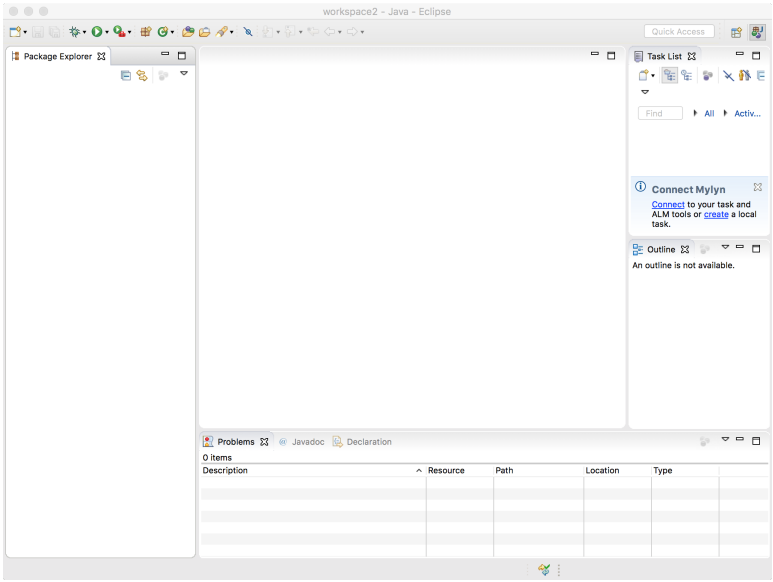
Special Notes: Linux
OpenJDK is an open-source implementation of Java, and is what may be the default option for Java on common Linux distributions. It is possible to install Oracle's JDK, but may require adding a non-free repository and/or a manual installation process. If you choose to use OpenJDK for this class, recognize that we will grade using the Oracle JDK, and thus you are taking a risk with respect to graded work.

1.2. Eclipse IDE for Java Developers

There are several good integrated-development environments (IDEs) for Java (e.g. NetBeans, IntelliJ) – in this class we will use Eclipse. Eclipse has a highly extensible plug-in system, supports many languages/platforms (e.g. Android), and is widely used in industry.

First, navigate to the Eclipse Downloads page (search term "eclipse java download") – click "Download Packages" from there. Next, download "Eclipse IDE for Java Developers" for your platform (match 32/64-bit to the JDK version above). Eclipse typically comes as a compressed folder – you can extract the contents wherever is convenient and simply run the main eclipse executable (e.g. eclipse.exe on Windows, Eclipse.app on Mac OS). There is a new installer option, which still seems to be buggy/slow.

When you first run Eclipse, it will ask you to indicate a "workspace" – this is a folder that will house all of your projects. You should choose a folder that does not currently exist, as Eclipse will directly manage its contents. After this, click the "x" icon, closing the "Welcome" tab (see the screenshot below) – you are now ready to use Eclipse.



2. GitLab

In this class, we are going to use GitLab[1] to distribute starter code, as well as for you to submit lab and programming assignments. GitLab is one of several popular web-based platforms (e.g. GitHub, BitBucket) that provide *version control*[2] services and software. While it is common to use a cloud-based version, in this class we'll use a WIT-based installation.

We won't talk about version control much in this class, but it's basic function is to manage *changes* to documents (in this case, the source code for your assignments) – it is crucial for most software projects, especially those involving teams, and is a fundamental skill for software development in industry. FYI: there are many software packages used for revision control (e.g. CVS, Subversion, Mercurial) – GitLab, as

[1] https://eagle.cs.wit.edu
[2] https://en.wikipedia.org/wiki/Revision_control

the name implies, uses git[3], a free and open-source system that is useful and popular for personal and industrial software development.

2.1. Login

All students in this class have a GitLab account. To login, visit the page and click the "Forgot your password?" link – type in your WIT e-mail address, click the "Reset password" button, and wait for an e-mail with instructions. Once you login, feel free to customize your profile/avatar as you wish, and set privacy/account settings as you see fit.

Important: Use Your WIT E-mail as Primary

In this class we will be automating the process of downloading and grading your programming/lab. This process depends upon you using your WIT e-mail (whatever@wit.edu) as your primary GitLab e-mail address. Thus, on the profile page[4], make sure your WIT e-mail appears in the Email field.

3. Complete the Assignment

The steps up till now you only need to do once. By contrast, you will follow the remaining steps for each assignment for the rest of the semester. Before you begin, here is the big picture of the process you will complete for each assignment:

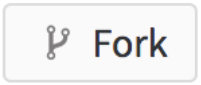
1. Visit the assignment **repository**
2. **Fork** the repository (i.e. make a *private* copy that is owned by your account).
3. Set appropriate **access** to your forked repository
4. Import the repository as a **project** in Eclipse
5. Write your code, saving your work frequently (**commit** and **push**)
6. Test your code using **JUnit**
7. Document your code using **Javadoc**
8. Confirm that your final submission is accessible to the instructor

3.1. Visit the Assignment Repository

In most version control systems there is the concept of a *repository* (or "repo" for short), which is a collection of one or more files and folders. In this class, each assignment will have its own repository. In GitLab, and comparable services, each repository has its own corresponding website. So to begin, visit the website for this assignment's repository (a link will be listed under "Your Projects" when you visit the GitLab main dashboard – accessible by clicking the Wentworth logo at the top-middle of any page).

3.2. Fork the Assignment Repository

On the repository website, click the "Fork" button – this will create a copy of the repository that your account will own (this will be where you will submit your work).

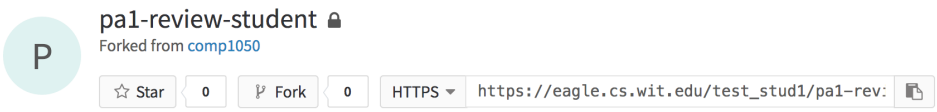


The resulting page will ask which user/project should own the forked repository – click the box that represents your user.

---

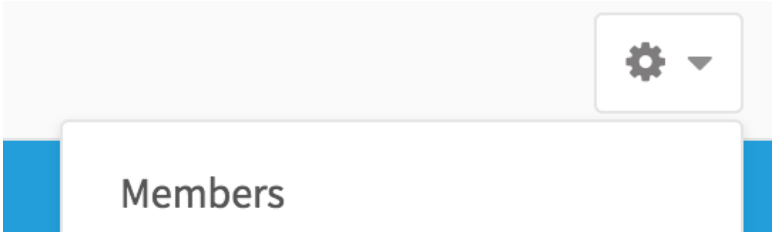
[3] https://git-scm.com
[4] https://eagle.cs.wit.edu/profile

If successful, you will be brought to the website of your forked repository (note that it will include a new button showing the repository from which it was forked).



3.3. Set Repository Access

The lock icon next to the name of the repository means that no one can access your repository unless you provide explicit access. To grant access, click the "Members" link from the "gears" (i.e. settings) menu:



On the resulting page, find your instructor in the **People** field, set **Project Access** to Developer, and click the "Add users to project" button. If this succeeds, you should see your user and the instructor, tagged as a Developer.

Remember

- If your repository is shared with any additional users, we will assume that you were cheating by letting other students see your work.
- If your instructor does not have Developer access, your work cannot be graded and you will receive a 0.

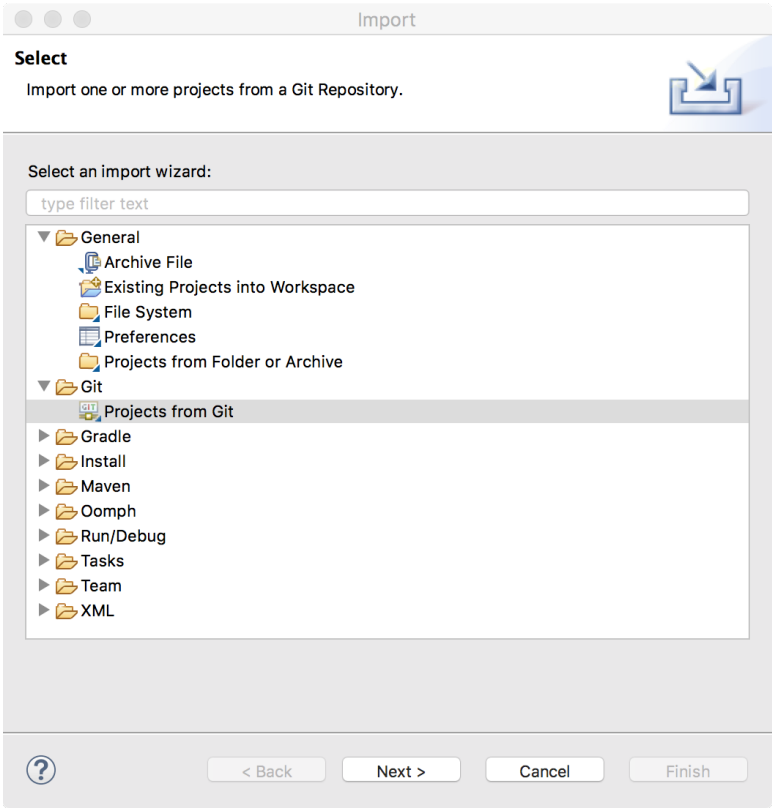
3.4. Import into Eclipse

You now have a private copy of the assignment in your repository on GitLab's servers – it's time to bring this to your computer, into Eclipse, so you can write some Java code!

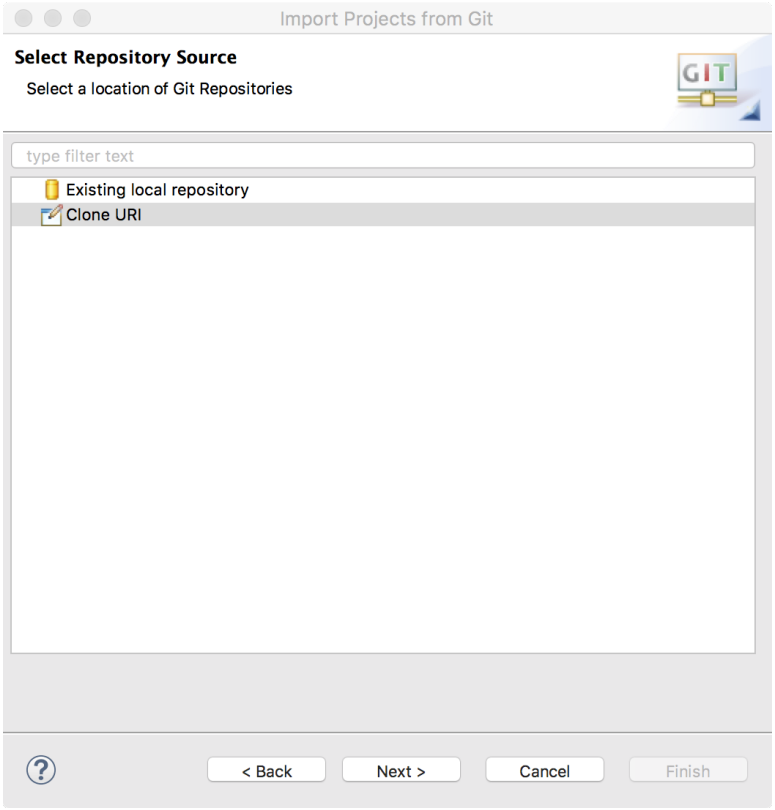
To begin, go to the website for your project (via the "Project" link at the top) and click the copy/paste button next to HTTPS – this will reveal copy the URL that git can use to interact with your repository:

Note that this URL should begin with https (if not – choose HTTPS from the dropdown to the left). Also, the URL should have your username in its second part – if not, navigate back to YOUR project website (otherwise you'll be trying to access code that your account does not own).

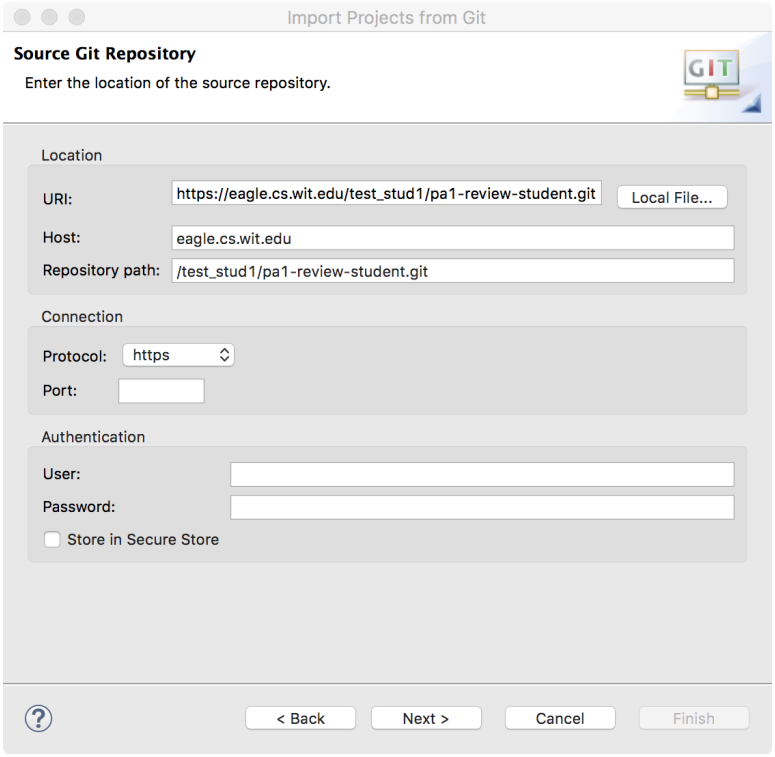
Now, in Eclipse, click the "Import" item in the "File" menu. In the resulting dialog, under the Git folder, choose "Projects from Git" and click the "Next" button.



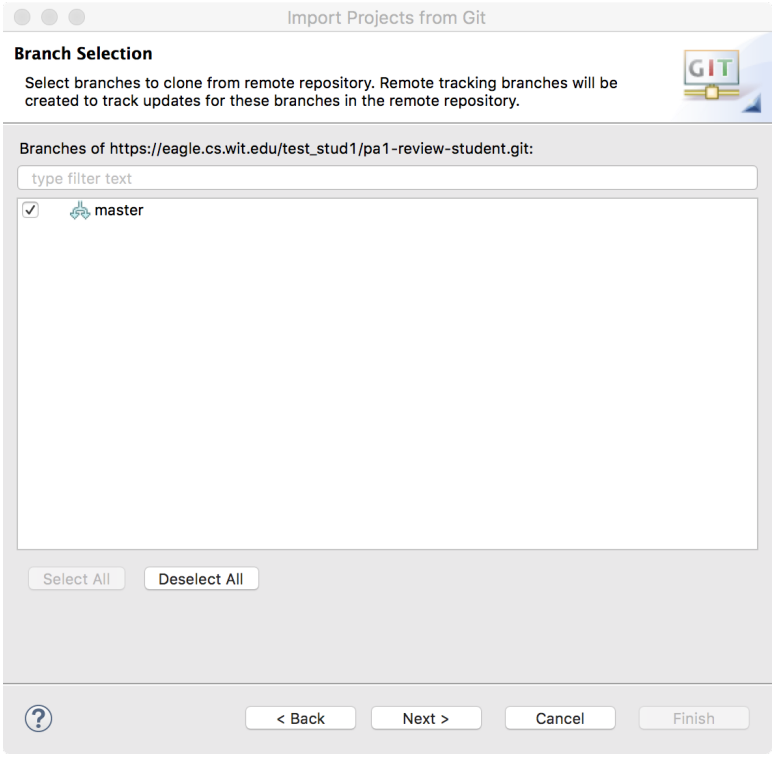
In the resulting dialog, choose “Clone URI” and click the “Next” button.



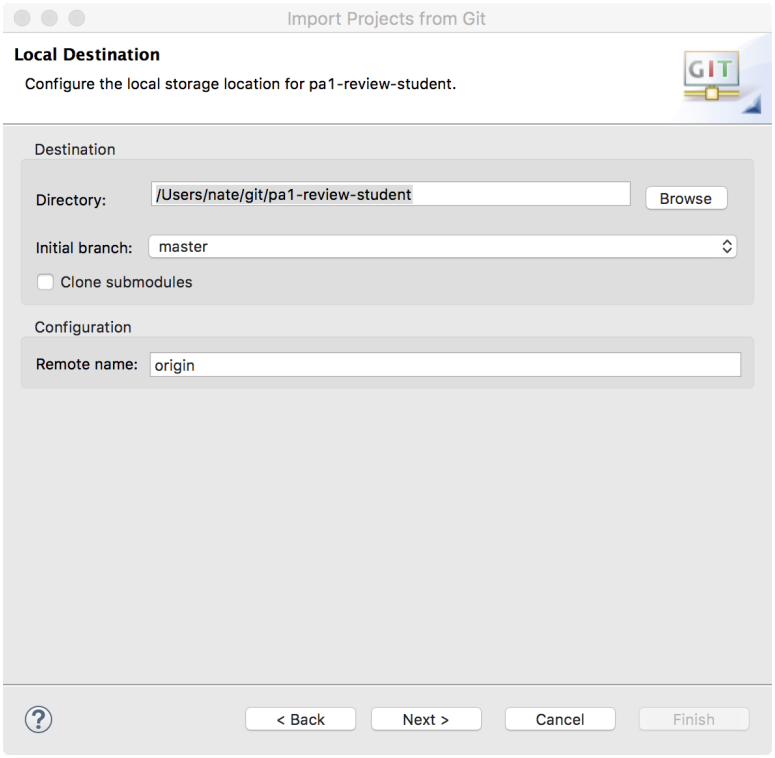
Now, paste the URL from your forked repository website into the **URI** field – Eclipse will parse this full URI automatically and fill in further fields. You may also wish to fill in and save your GitLab user/password in the Authentication section to save yourself many authentication prompts in later steps. Now click the "Next" button.



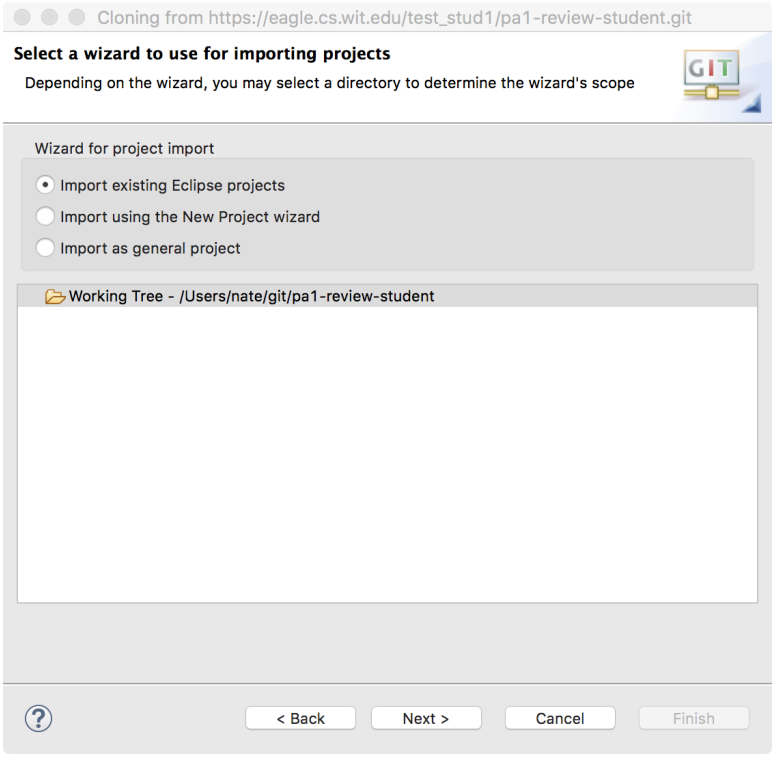
Keep the selection of the “master” branch in the following dialog, click the “Next” button.



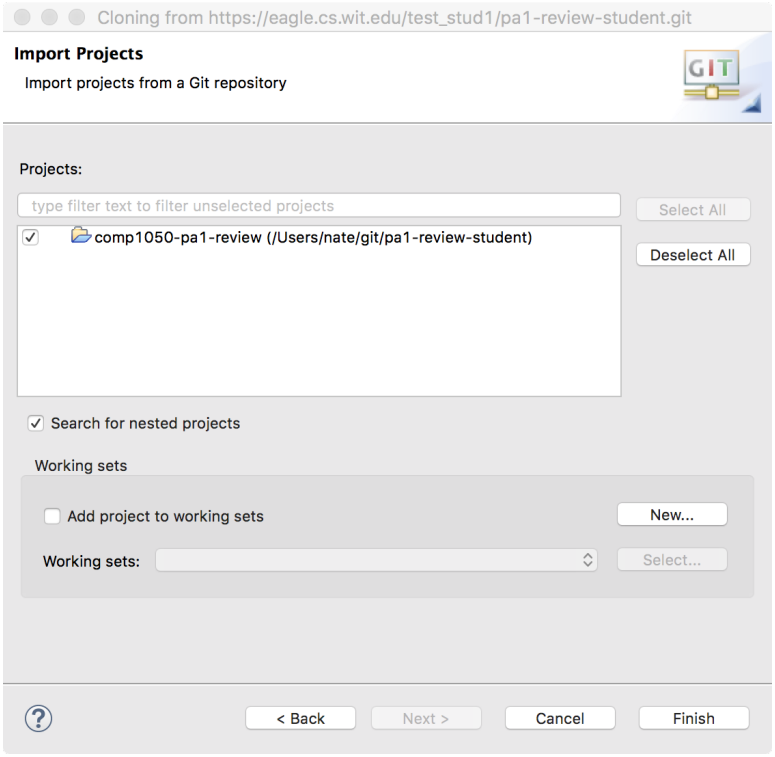
Now, choose a location for the local copy of your forked repository. This is entirely up to you, but must be a new/empty directory. Keep all other options unchanged, click the "Next" button. (You may be asked about creating a master password at this stage, and you should do so.)



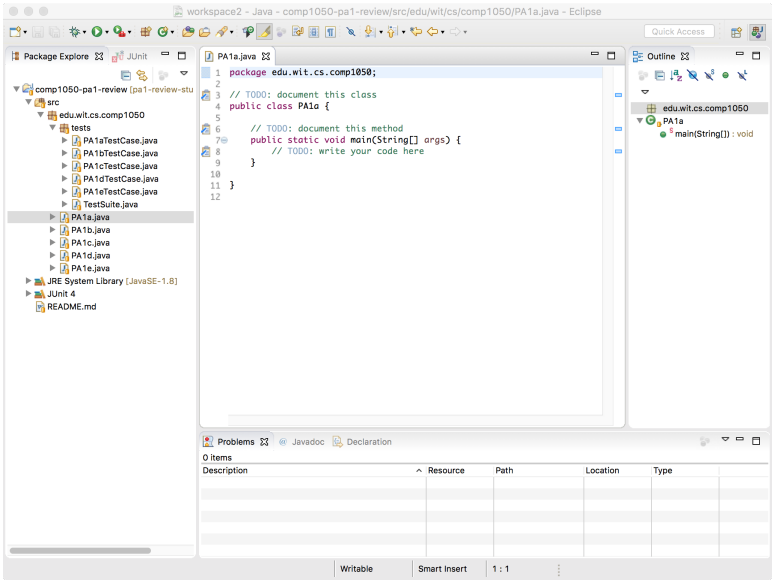
Keep the option to “Import existing Eclipse projects” on the next dialog, click “Next”.



Eclipse should automatically detect and check a project, so click the “Finish” button.



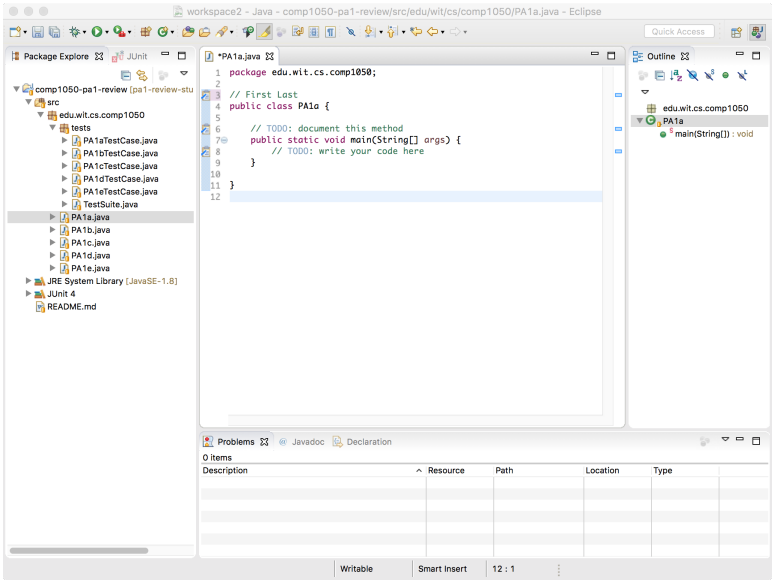
In Eclipse, you should now have a project (listed in the Package Explorer tab on the left) with base assignment code. The small orange cylinder icons show that the files are linked to a repository. Double-click PA1a.java to see the base code you have been provided.



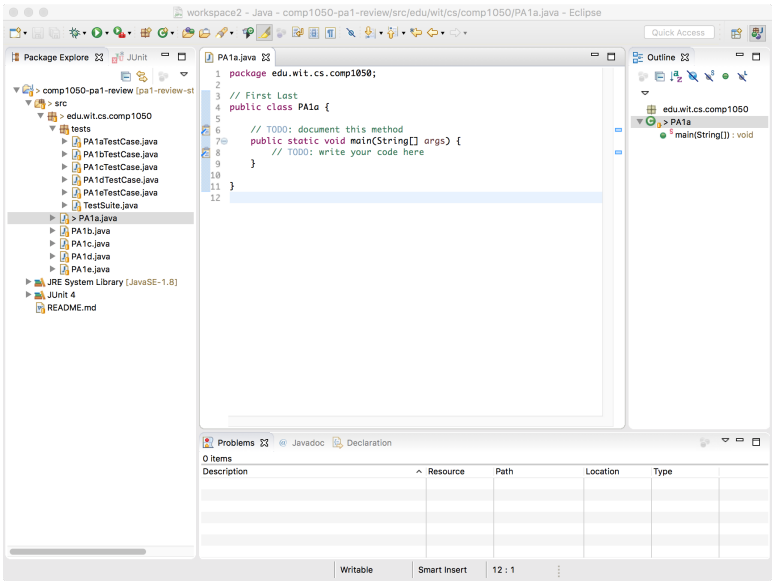
3.5. Write & Save Code

As you work on your assignments, you will save file changes, *commit* logical groupings of changes to your local copy of the repository, and then *push* these commits to your forked repository on GitLab. In this section we will proceed through these situations and see how Eclipse reflects the state of your project.

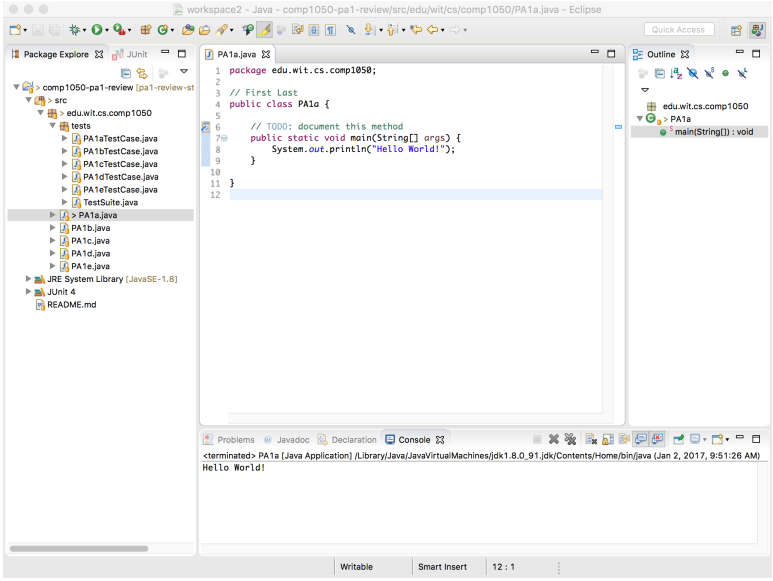
To begin, open the **`PA1a.java`** file and change the first comment in the file (TODO: document this class), replacing it with your name. As you type, notice that the tab has an asterisk (*) before the file name, indicating the file has been changed, but not saved.



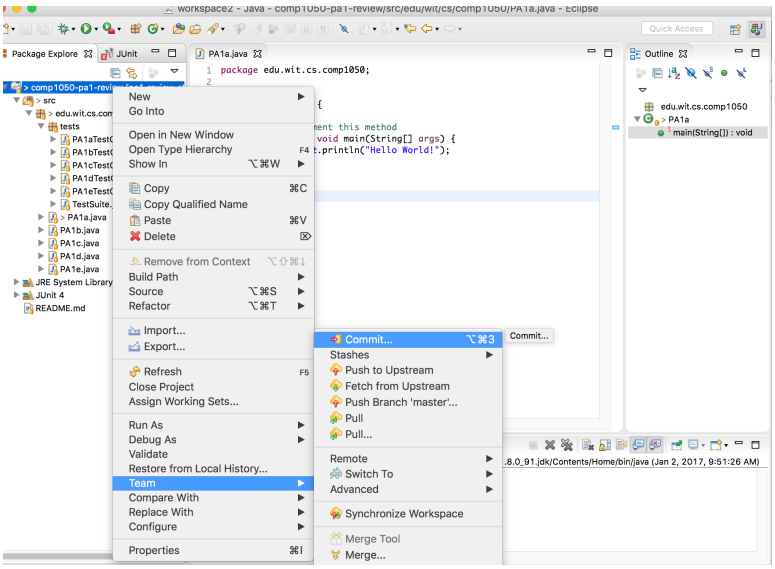
Now save the file (via the icon in the toolbar, the File menu, or keyboard shortcut). You should notice that the asterisk is no longer present, but there is a greater-than sign (>) in front of the file name – this is indicating to you that the file has been saved, but that the change has not been committed to the local repository.



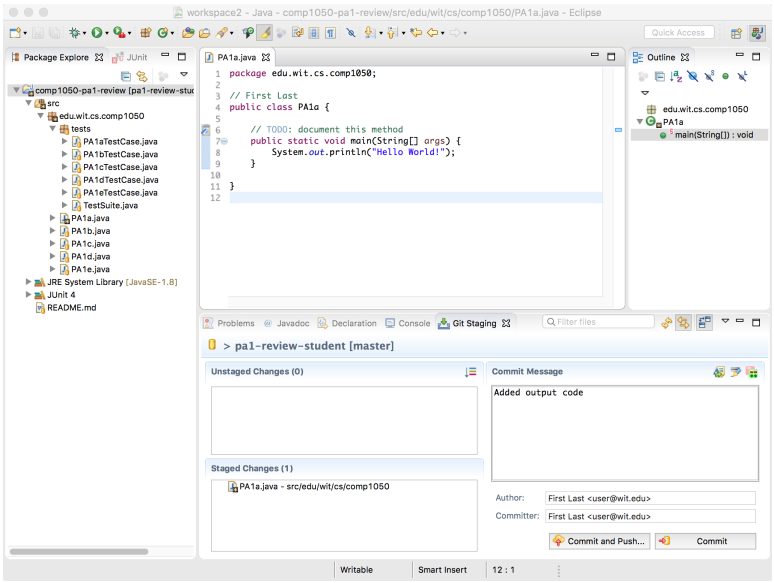
Let's make further changes to complete the assignment: replace the code within the main to output "Hello World!" to the terminal, provide a quick comment of your main method, and save these changes. Then, click "Run" from the "Run" menu (or the green play button in the toolbar) to see the program execute.



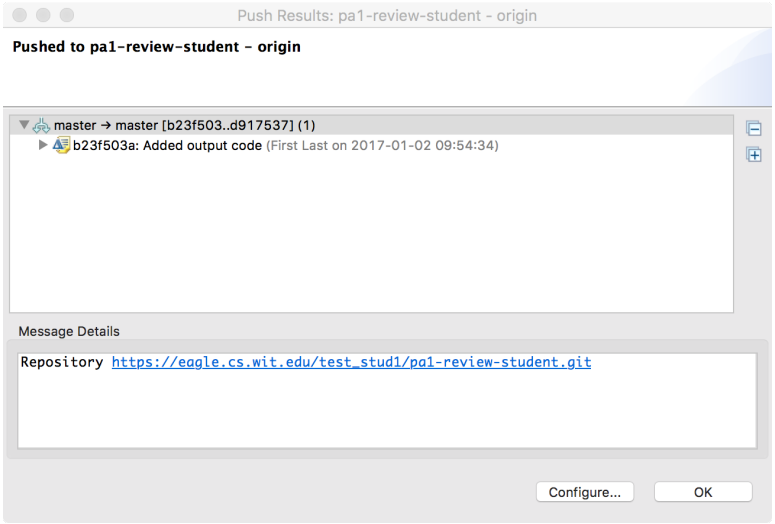
Now that you have made a significant set of changes, let's commit these to GitLab. First, right-click the project (the top-level icon) and click "Commit" under the "Team" menu.



Doing so will bring up the "Git Staging" tab. First you will indicate which file changes you wish to commit – do so by dragging files from the "Unstaged Changes" box to "Staged Changes" (if you are unsure, double-click a file to see what has changed). Now, type a comment about what you are committing in the commit message box. Now, set the Author and Committer boxes to reflect yourself – format should be "Firstname Lastname <email@wit.edu>" (without quotes).

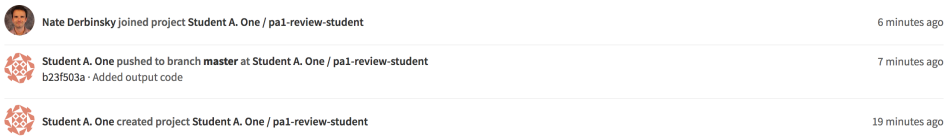


Finally, click the "Commit and Push" button to save the changes locally AND send them up to GitLab. You will see a brief window with some progress and, if things go well, a dialog confirming your work.



You can click OK to this dialog, and now close the "Git Staging" tab. Your Eclipse workspace should now look similar to how it was before you started changing files (i.e. no *'s, no >'s). This is the general process of saving code: make & save changes, then commit & push changes to GitLab. You should do this often – GitLab becomes a backup of your work in case something happens to your computer! Note that you can commit and push as many times as you wish – frequent updates will have no negative effect on your grade. HOWEVER, only work on GitLab can be graded – if you don't commit & push, you may lose credit.

Importantly, to confirm that the changes have been pushed to GitLab, go to the website for your forked repository and click the "Activity" link at the top.

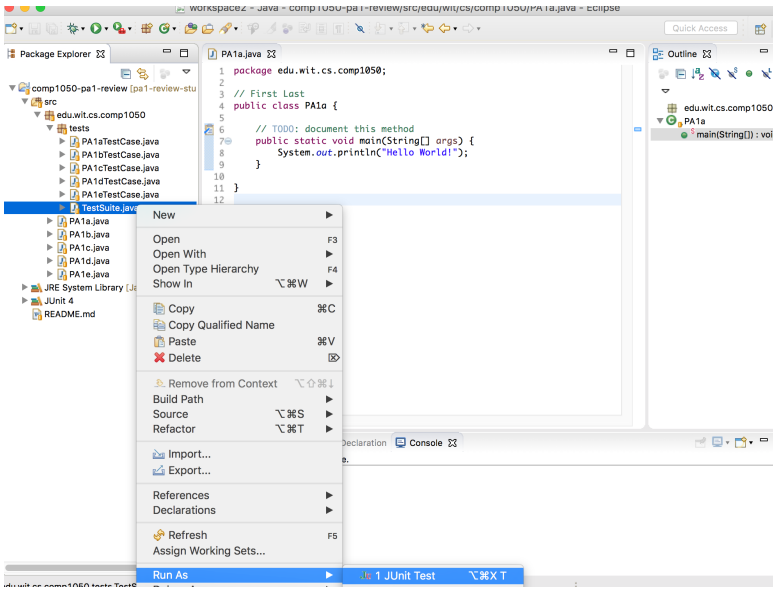


The resulting page will have a history of changes to your code. Under each push event you will see the commit message you typed (note: make your commit messages brief, but descriptive, to help understand the code changes). The letter/number before each message is a unique id associated with the commit – click it to see details of the changes. You should also see that your instructor was added to the project.

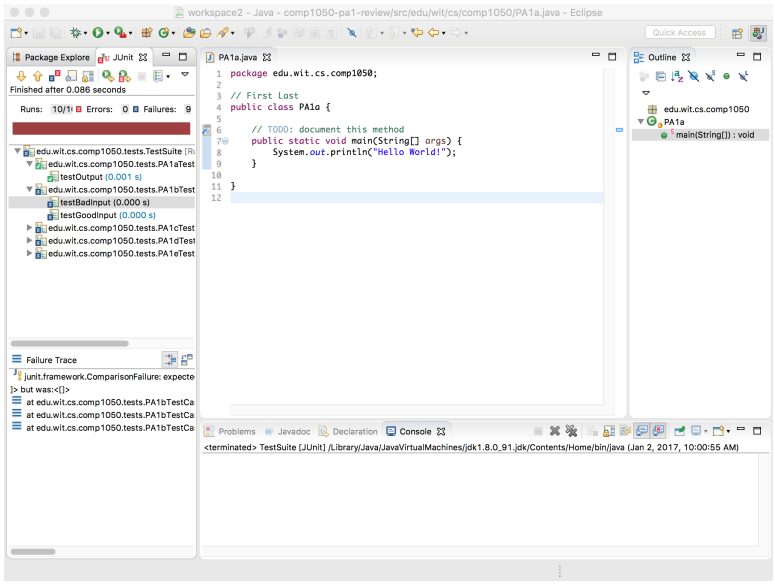
At any time, click the "Repository" link at the top to see the latest state of all files in the GitLab forked repository. Navigate through the folders and click any file to see its contents. **What you see on the website is what your instructor sees during grading – so make sure each file is how you want it before the assignment deadline.**

3.6. Test Code

Assignments this semester will come with a set of one or more tests to give you feedback on how your assignment is progressing. You can run the tests in Eclipse by right-clicking the "TestSuite.java" file and choosing "JUnit Test" under the "Run As" menu.

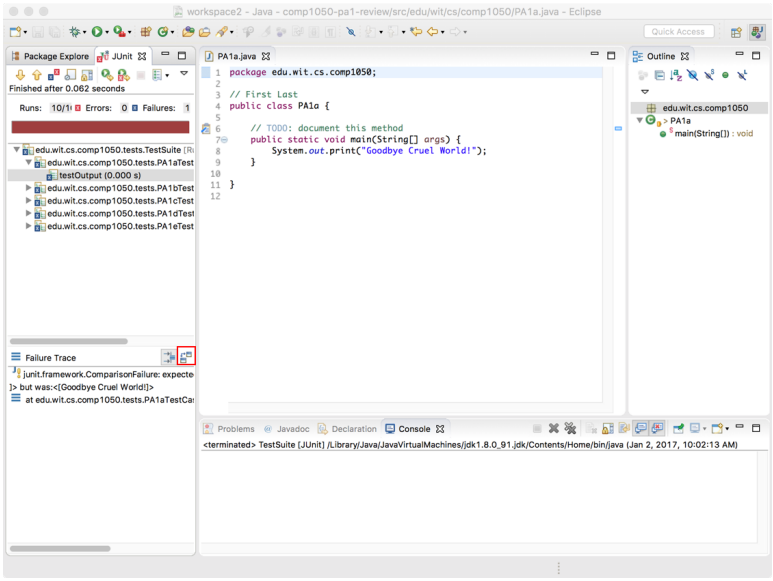


Running this file will bring up a JUnit tab on the left (next to the Package Explorer we've been using).

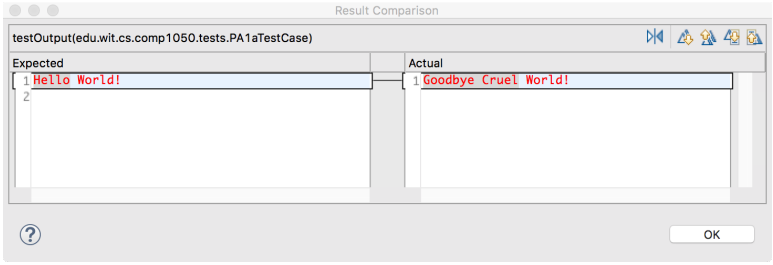


For each test run, you will see whether it passed (green check), failed (blue cross), or caused an error (red cross). If a test fails, you can get information about the reason by clicking on the test in question.

For example, in your code, change the output message from "Hello World!" to "Goodbye Cruel World!" and remove the "ln" at the end of "System.out.println" – now re-run the tests. Click on any failed test (e.g. testOutput) to see a summary of the Failure Trace (below the list of tests) – that is, the output it was expecting vs. the output it received.



Often this is hard to read the output, so click the "compare" button in the upper right of the Failure Trace section to zoom in:



This dialog is showing the expected output on the left and the actual output your program produced on the right. Importantly, you'll see **two** differences. First, the text is different (and the grey outlines these differences). Second, and much more subtly, there are TWO lines on the left whereas only ONE line on the right (see that the line numbering on the left ends at 2, whereas it ends at 1 on the right). While this may seem inconsequential, the tests are very picky – importantly, if you see the tests fail on your computer, so will the grading tests.

Now, undo the changes to the code (if done correctly, you shouldn't have any >'s in the Package Explorer), save, and re-run the test.

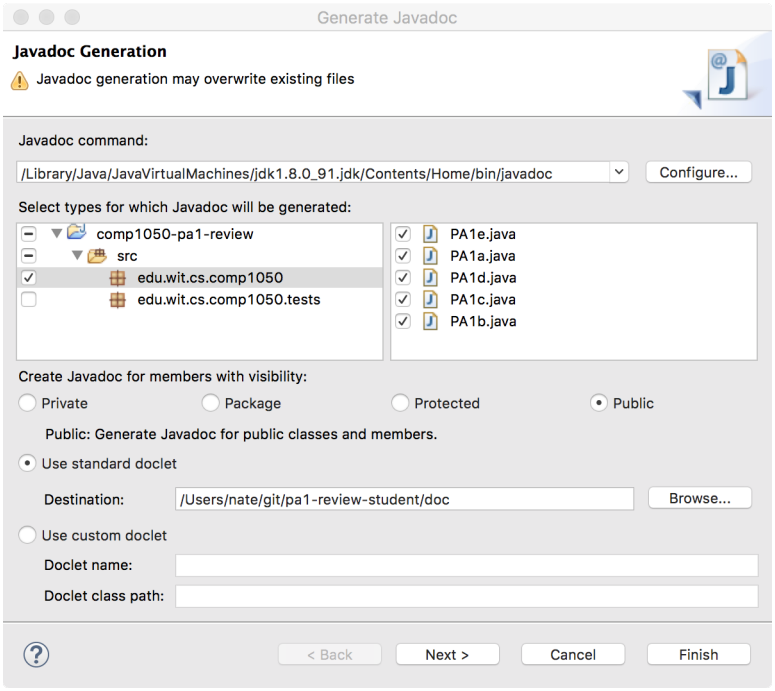
In general, if you pass all of the tests, your assignment is in good shape; however, this is no guarantee of a perfect grade, and so you should test your own code thoroughly before turning in the final version.

3.7. Document Code

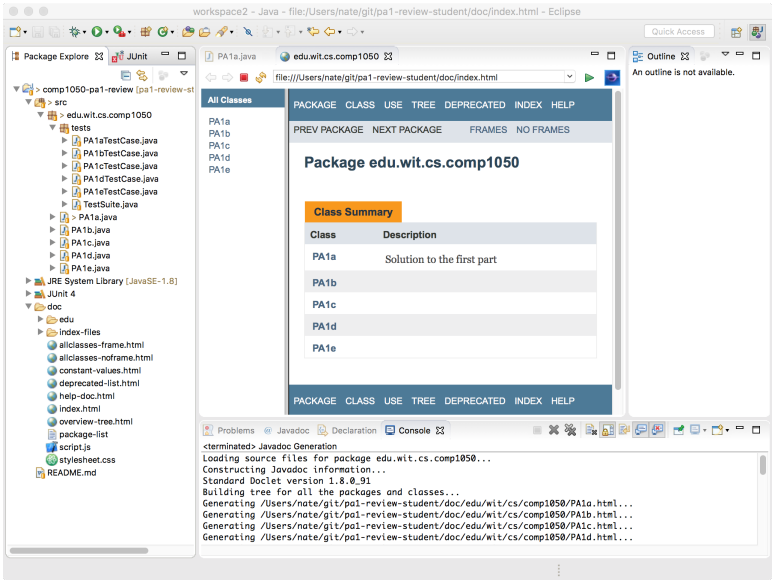
All code you turn in for this class must be properly documented using Javadoc. A Javadoc comment starts with **/**** and ends with ***/**. In your current assignment, provide a comment for the PA1a class and main method, as shown below (substitute your own name; ***note that this will NOT satisfy all the Javadoc requirements, but is used as an example***).

```java
package edu.wit.cs.comp1050;

/**
 * Solution to the first part
 *
 * @author Nate Derbinsky
 */
public class PA1a {

    /**
     * Starts the program, outputs "Hello World!"
     *
     * @param args command-line arguments, ignored
     */
    public static void main(String[] args) {
        System.out.println("Hello World!");
    }

}
```

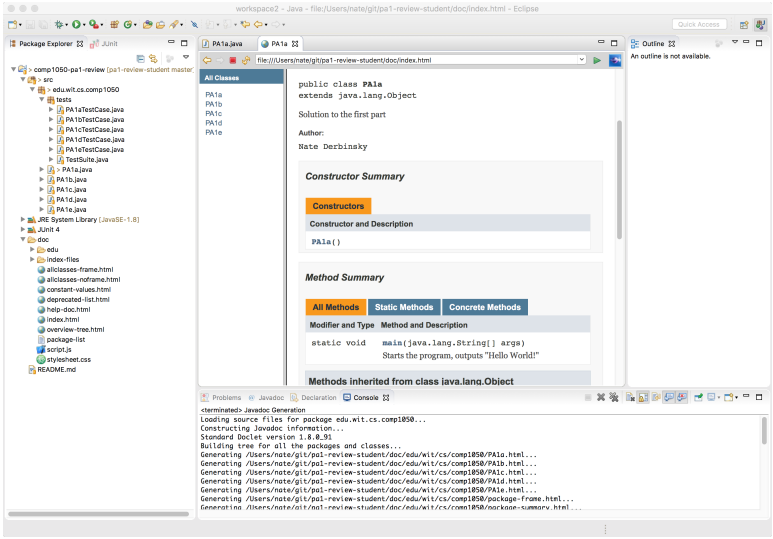
To test that you properly documented your code, run Javadoc by clicking the "Generate Javadoc" item from the "Project" menu. In the resulting dialog, choose the source file(s) you documented, **Public** visibility, and click the "Finish" button (the first time you do this, you may have to click the "Configure" button to find the *javadoc* executable in the JDK you installed).



After confirmation, the Javadoc program will run and you will see a trace of it producing HTML files. If there are no errors in red, you properly documented your code and you should now have a "doc" folder in your project. Double-click the **`index.html`** file to see the produced webpage.



You should click the link for your PA1a class to verify your overall description, authorship, and documentation of the main method (and other methods as appropriate for future programs).



If this documentation was produced correctly, then you can close the tab.

3.8. Confirm Submission

Now complete this part of the assignment, as you will EVERY assignment in this class:

- Make sure your code passes at least all the provided JUnit tests
- Create and test Javadoc code documentation
- Save, commit, and push all code changes
- Confirm the latest code is visible via the "Repository" tab of your repository website
- Confirm that the instructor has Developer access

4. Remaining Programs

In the preceding sections of this document, we successfully completed part a (PA1a) of the assignment. The remaining sections provide descriptions of the remaining programs for this assignment – be sure to complete them ALL for full assignment credit.

4.1. Part b

Write a program that prompts the user for a count of quarters, dimes, nickels and pennies. The program must output the total amount in dollars and cents. For example:

```
Enter number of quarters: 6
Enter number of dimes: 2
Enter number of nickels: 0
Enter number of pennies: 1
You have $1.71 in coins.
```

Note: you must ensure that the output properly shows two decimal places (e.g. $1.50 vs. $1.5) and, if any of the counts are negative, provide an error message after inputting all counts (supplied as a constant).

4.2. Part c

Write a program that prompts the user for a dollar amount. The program outputs the fewest number of coins needed. For example:

```
Enter total amount: $1.71
You have 6 quarters, 2 dimes, 0 nickels, and 1 penny.
```

Note: due to numerical error[5] you must convert the initial input to an integer – the number of pennies before performing any calculations. Once computed as a double (i.e. 171.0), you have been supplied a method (**`convertToInt`**, which you MUST use) to convert the double to an integer (i.e. 171). Only then proceed to compute number of coins.

[5] See https://en.wikipedia.org/wiki/Numerical_error for context.

4.3. Part d

A shipping company uses the following function to calculate the cost (in dollars) of shipping based on the weight of the package (in pounds).

$$c(w) = \begin{cases} \$3.50 \; if \; 0 < w \leq 1 \\ \$5.50 \; if \; 1 < w \; \leq 3 \\ \$8.50 \; if \; 3 < w \leq 10 \\ \$10.50 \; if \; 10 < w \leq 20 \end{cases}$$

Write a program that prompts the user to enter the weight of the package and displays the resulting shipping cost. If the weight is outside the valid bounds, display an error message (supplied as a constant). For example:

```
Enter package weight: 3.6
It will cost $8.50 to ship this package.

Enter package weight: 0.4
It will cost $3.50 to ship this package.

Enter package weight: 20.1
The package cannot be shipped!
```

To begin, first implement the **`shippingCost`** method to convert weight to cost – then you MUST use this in your main, where you will perform all user input/output.

4.4. Part e

Write a program that prompts the user to enter a string and displays the number of the uppercase letters in the string. For example:

```
Enter a string: Wentworth Is Terrific
There are 3 uppercase characters in the string.

Enter a string: break was short :(
There are no uppercase characters.

Enter a string: Glad to be back coding!
There is 1 uppercase character in the string.
```