Introduction

One aspect of being a software developer that is sometimes frustrating is getting your computer set up with all the tools you need. At Cal Poly, there's a pretty wide diversity of computers and tools that are used. This class is no exception: There may be tools that are unfamiliar to you. They are set up for your use on the lab computers and the unix servers. It can be convenient to install some or all of them on your own computer, but doing so will require installation work on your part.

If you choose to install tools on your own computer, your first resource if you run into problems is your classmates. This is one of the functions of lab time – you are encouraged to work on the labs with your classmates. This is particularly true if you're taking the class at a popular time.

This document is an attempt to explain in one place the key tools you'll be using. It may be updated as the quarter progresses, and it becomes more clear where the difficulties lie.

Unix, Mac, Windows, and Linux

This is a class on object-oriented programming, not Mac/Windows/Linux system administration. You're not required to use the unix server accounts that Cal Poly provides, but if you don't, we can't necessarily provide a high level of support for getting your computer set up. If you don't know how to set up environment variables, like your PATH, and new environment variables, like one called CLASSPATH for Java, you might want to use the lab computers and/or the unix servers.

Text Editor

Some popular choices are vi/vim and Sublime, but Sublime isn't installed on the Unix servers, and vi can be intimidating and has a steep learning curve. The lab machines have an editor called "gedit" installed that should be pretty simple to use, but it has a GUI (graphical user interface), so you can't always use it if you ssh in. gedit doesn't seem to be available on the Unix servers. A simple editor that is available in both places and that doesn't require a graphical system is **`pico`**. Type "pico <name of file>" to edit a file with pico.

Setting Up the Environment on the Unix Servers

On the Unix servers (unix1.csc.calpoly.edu through unix5.csc.calpoly.edu), I provide a file that sets up your environment for you. Use your favorite text editor to edit the file ~/.mybashrc, and add the following line:

```
source /home/wffoote/public/set_env_203
```

Note that ".mybashrc" is a special file that Cal Poly has set up to be read each time you log into the unix servers. On most other unix systems, you'd modify your ".bashrc" file or your .profile file to set up your environment. Like usual, when you change `.bashrc`, you should log out and log back in for the changes to take effect. If it doesn't work, please try adding the given line to .bashrc or .profile; it's possible your account might not be set up to read ..mybashrc. If you're desperate, you can always type "source /home/wffoote/public/set_env_203" each time you log in.

Git and GitHub

An important tool you'll be using is Git. Git is a very rich tool with many features, but you'll mostly only be using five main commands. Git is a source-code control system. It's designed to manage a shared repository of documents on a server. Your first step with Git is to clone the repository you'll.

GitHub is a could-based service built on Git. We are using GitHub for distribution of class materials, and for assignment submission. You'll also be "submitting" your assignments via PolyLearn, but this will just be a link to the GitHub repository where the real submission happens.

Git is installed on the unix servers. The git website is https://git-scm.com/. If you want to run it on your computer, and it's not already installed, you should be able to find a program to install there.

The first step with git is to clone a repository. This creates a local copy of the files in the repository, and set you up to read new files from the repository as they're updated (or "pull" them), and also to send any modifications you make back up to the repository ("push") them. To clone a repository, you need a special URL that GitHub provides. Note: If you press the button that says "clone or download," and you pick to download a zip file, *you have not cloned the repository.* Don't do this; it will just make your life harder later. Similarly, don't make changes directly in your repository using the GitHub web site. If you make changes both on your machine and on GitHub, you'll eventually have to merge them, and this process can be complicated.

The basic flow is that you clone a repository. Then, you make any local changes you want to make. You then add the files to the set of files you'll eventually push back, using the "git add" command. Next, you set up your files to be pushed, and you write a message describing your changes using the "git commit" command. Finally, after "git commit," you push your changes back to the repository using "git push."

One other command is needed: "git pull." If the files in a repository change, use "git pull" to copy those changes over to your local repository. This is how you can easily get updates to the general repository when I publish them.

These five git commands and more are described in general/notes/git_commands.txt.

The Java Environment

You will need a java environment to use for this class. The command-line version is installed and configured on the unix servers already. If you want to install Java on your computer, you can get it from http://java.sun.com/. Look for "Java SE." Please install Java version 8 (not 9 or 10). Newer versions have added some language features we won't be discussing, and if you use them by mistake, your program won't compile when we grade it. Also, sometimes bugs in other code, like the libraries we use in this class, might be exposed by a newer Java version.

Note that the Java environment is sometimes called the JDK ("Java Development Kit"), or the Java SDK ("Software Development Kit"). JDK, Java SE and Java SDK are all synonymous. You can tell which version you have by typing "`java -version`" and "`javac -version`."

> OSX and Linux hint: If you have a newer version of Java already installed, you might need to remove it. It seems that the OSX Java Console (available under System Preferences → Java) doesn't let you switch the current version. On OSX, the JDK is installed under `/Library/Java/JavaVirtualMachines`, and on Ubuntu Linix it's in `/usr/lib/jvm`. You should be able to use Java 8 from the command line by setting your path so that the bin directory of the JDK you want to use comes before `/usr/bin`. I only tested this on Linux. If it doesn't work on OSX, you can change the names of the directories containing the different JDKs you have installed (using `sudo mv`), then re-install Java 8. Once you've verified that it works, you might want to clean up by removing the old directories (e.g. using `sudo rm -rf`).

The yEd Editor

In this class, we make diagrams called "UML class diagrams." One tool for creating these is a free editor called "yEd." I have set this up on the Unix server. If you use the "setup_env_203" file described earlier, you can run it with the "yed" command. If you want to run it on your computer, you can download a copy from https://www.yworks.com/products/yed.

> Windows hint: On some Windows machines, the yEd menu font might be tiny. yEd is a Java program, so you should be able to find the file "yed.jar," either on your computer on the yEd web site. Try running yed with:
>
> ```
> java -Dsun.java2d.dpiaware=false -jar <directory>/jed.jar
> ```

The JUnit Library

Many of the labs and assignments use a library called JUnit. If you use the "setup_env_203" file described earlier, Java is already configured to find these libraries. This is done by setting the CLASSPATH environment variable.

If you want to use JUnit from your computer, you'll find copies of the two JAR files that make it up in general/libs. Those files are hamcrest-core-1.3.jar and junit-4.12.jar. You'll want to set your CLASSPATH to point to wherever you have those files stored on your machine. For example, setup_env_203 has this:

```
export
CLASSPATH=.:/home/wffoote/public/junit/junit-4.12.jar:/home/wffoote/
public/junit/hamcrest-core-1.3.jar:/home/wffoote/public/203_project/
processing-core.jar
```

As you can see, CLASSPATH does not use the abbreviation "~wffoote," it uses the full path (/home/wffoote). The "~" abbreviation is not available in this kind of setup file. On Windows, you set your CLASSPATH using the "edit environment variables" command that you can find with Cortana. In Windows, entries are separated with semicolons rather than colons.

Note that setup_env_203 also includes a library called "processing-core.jar". This is described next.

The processing-core.jar Library

The class uses a library called "processing-core.jar." I've made setup_env_203 so that this is added to your CLASSPATH for you. A copy of processing-core.jar is also in the general repository, in class_project/given_code/java/processing-core.jar. If you're using your computer, you may want to include this in your CLASSPATH.

The IntelliJ IDEA Integrated Development Environment

IntelliJ has a nice integrated development environment. After lab 2, you can use it if you wish. Be careful if you do, though! IDEs are big, complicated tools. Sometimes they sometimes try to automatically fill in code for you in ways you might not notice, like by automatically generating java import statements. You are responsible for all the code you submit, including anything an IDE generates for you, even if the program guesses wrong.

I have installed it on the Unix servers, and it should be available on the lab computers as well. It may be slow to run on the campus computers, particularly if you are remote. If you want to install IntelliJ IDEA on your computer, you can download it from https://www.jetbrains.com/. The free "Community" edition is perfectly fine for this class, but if you want, you can also get a free student license for the "Ultimate" edition.

To launch from the Unix server, type the command idea. The first time you launch it, it will ask you about importing settings; choose "do not import settings." It will ask you to for various other setup information; I picked "skip the rest and accept the defaults" after setting the UI theme.

IMPORTANT: Before using IDEA for the first, time, please go into "Configure," "Settings," "Editor," "Code Style" then "Java." On the "Tabs and Indents" screen, make sure "Tab size" is set to 8. If you need to change it to 8, press "Apply."

In order to create a new Java program project, pick "Create New Project." It will present an "Additional Libraries and Frameworks" dialog. You don't need to select anything here; just accept the default of Java, and press next. Similarly, at the "Create project from template" dialog, select nothing and press Next. For the project location, you can pick anywhere in your directory. If this is for an assignment, you can set the project location to the (mostly empty) git repository you've created for that assignment (using git clone).

In order to create a new Java class, click “View,”, “Tool Windows” then “Project.” Click on the name of your project (on the top of the project view). Then, right-click on the “src” directory, then pick “New” and “Java Class.” As you create classes, you can open them in the editor by double-clicking the class under src.

To run your program the first time, right-click the class that contains you main function (under src), and pick “Run <class name>.main()” from the menu. Once you’ve done that in your project, you can run your program by clicking the green triangle in the upper-right corner of the IDEA screen.

If you have existing Java source files to add to your project, simply copy them to the src directory in your project. You might need to right-click src in IDEA and choose “Synchronize ‘src’” for IDEA to register them.

If your program uses libraries (in JAR files), you’ll have to add them. Pick “File” then “Project Structure.” Next pick “Libraries” under Project Settings, and pick the little green + symbol on the top toward the left. Pick “Java,” and navigate to the directory. I have stored libraries for your use in /home/wffoote/public/junit, and /home/wffoote/public/203_project.

In order to run JUnit tests, pick the drop-down next to the run button (near the upper-right hand corner of the screen), then “Edit Configurations.” Click “+”, then “JUnit”. Under “Class,” choose the name of the class where you put the JUnit tests. Then press “OK”. Now, the drop-down next to the Run triangle can be used to select running tests, or your main program.

X Windows

At times, you may need to run programs that present a graphical interface. If you are using your own computer, or you are in the lab using one of the lab computers, this should work without issue. If you are using ssh to connect to a Unix server, however, you need to run a program called an “X server” on your laptop,

If you have Linux, an X Server is already installed. On Mac, the supported X server is called XQuartz, and it should be free. If you’re running Windows, you will need to install one. I’m told the X Server at https://sourceforge.net/projects/vcxsrv/ works. Note that on Windows you have to run the X Server yourself, before you try to launch a remote program that attempts to use it.

In order to display GUI (graphical user interface) from a remote computer that you connect to with ssh, you need to use "ssh -X," as in:

```
ssh -X unix3.csc.calpoly.edu
```

If you're using a graphical ssh program, the "-X" option might be called "tunnel X/11 protocol" or something similar.