

CSCI 1102 Computer Science 2

Fall 2018

Problem Set 1 : Getting Started

8 Points

Due 12AM (midnight), Tuesday September 4, 2018

This first problem set will be delivered to you twice: first as a PDF attachment to an email and then again after you've joined the class GitHub organization BC-CSCI1102 it will reappear as the README.md (markdown) file in your git repository for problem set 1.

You can and should complete part 1 of this problem set based on the PDF form of this writeup. To complete part 3, you'll need the git repository. You'll be receiving an email with a link allowing you to clone that repository.

All subsequent problem sets will be delivered to you just once as emails with links for cloning a git repository.

In this course, we're going to develop a lot of software. We're using the Java programming language though we could just as well use something else. The purpose of this problem set is to get your system set up and to get your feet wet with Java. There are 3 parts:

1. **Join:** You should complete part 1 by **midnight tonight Tuesday August 28;**
2. **Development Setup:** you should complete part 2 by **midnight Thursday August 30** and
3. **Java Code:** you should complete part 3 by **midnight Tuesday September 4.**

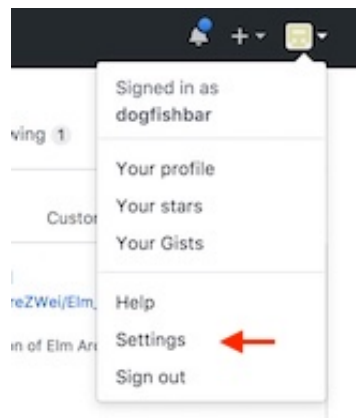
Fair warnings:

1. There are a lot of specific details in this problem set so you have to follow the steps carefully;
2. **If any part of this problem set is late, the whole problem set is late. Deadlines will be strictly enforced throughout this course.**

Part 1: Join

These three subtasks should be taken care of by **midnight tonight Tuesday August 28.**

1. You should have received an email from Piazza inviting you to join the class [Piazza forum](#). If you haven't done it already, accept that invitation. If you didn't receive that email or you've lost it, let us know.
2. If you haven't done it already, take a short [survey](#);
3. Join the course GitHub organization BC-CSCI1102. In order to join this organization, you'll need a GitHub account. Sign up if you don't have one already, it's free. You can make any GitHub ID that you'd like (and that you won't mind sending to prospective employers or admissions committees down the road ...) but for the purposes of this course, you are **required** to set the **Name** field of your GitHub account to your full name as specified in the BC system. E.g., my *GitHub ID* is `dogfishbar` (a fly-fishing spot) while my *GitHub Name* is `Robert Muller`. Note that no middle initial is required. The **Name** field is found from **Settings** (upper right):



You won't receive much credit for this first problem set unless you set the name correctly. Once you've joined GitHub, email your GitHub ID to me. Your email should have a subject line with **exactly**:

Subject: cs1102: your-github-ID

Where `your-github-ID` is the unique ID that you chose above. The body of the message can be anything you like. You'll subsequently receive an invitation to join the BC-CSCI1102 GitHub organization. Accept that invitation.

Part 2: Development Setup

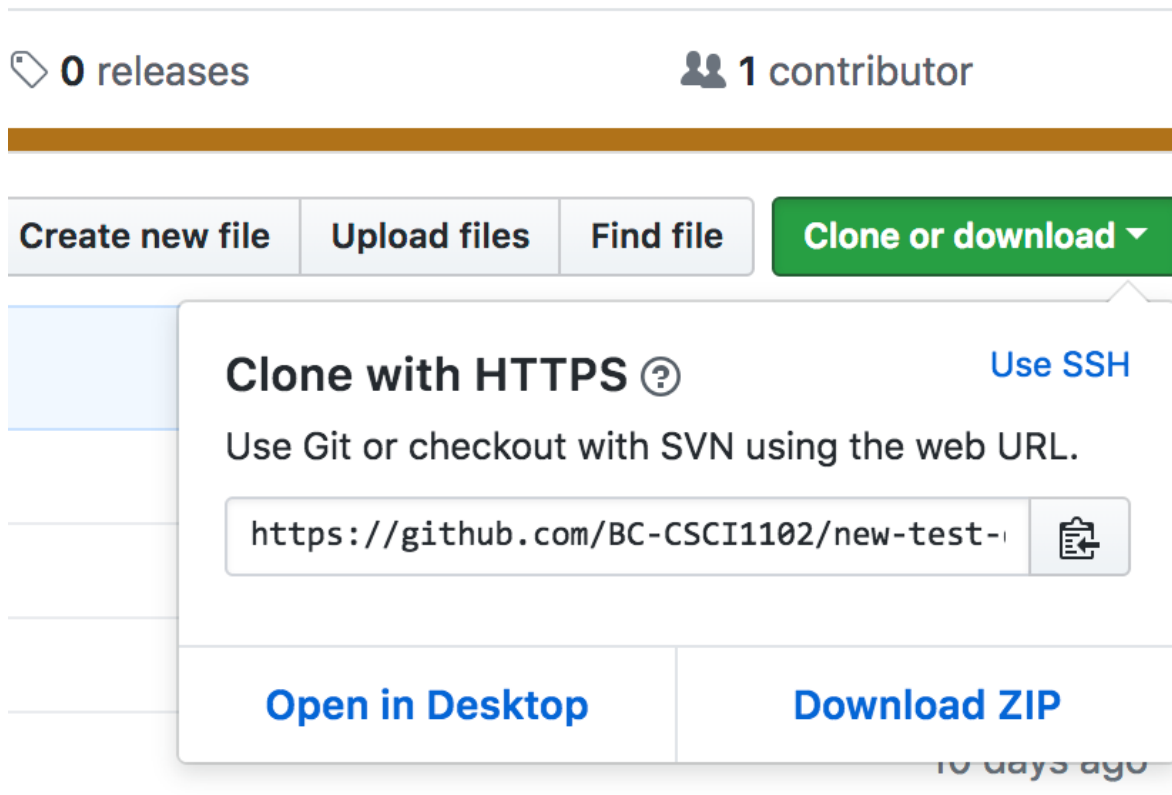
The development setup has 4 subtasks:

1. Create a course folder & accept problem set 1;
2. Install a Java compiler & runtime system on your laptop;
3. Install the algs4 course software library and
4. Install and configure the Atom editor.

1. Create a Course Folder & Accept Problem Set 1

1. Create a course directory/folder to house all of your materials for this course. Call it something like CS2 or CS1102. Put it in a convenient place in your file system. Better not to put it in Downloads.
2. You should have received an email from GitHub inviting you to accept problem set 1. If you didn't receive such an email, contact a course staffer. Accepting the invitation will cause GitHub to create a private copy of problem set 1 for you on the course GitHub website BC-CSCI1102. The problem set is in the form of a [git repository](#) that is shared between you and the course staff. The name of that repo will be `ps1-your-github-ID`. Select the link to that newly created repo.

The next task is to make a local **clone** of your repo on your laptop. This is easy to do from a command line shell, but we don't have that set up yet. For the time being we'll use GitHub's Desktop app. Select the green **Clone or download** button. Select **Open in Desktop**.



The first time through, GitHub will ask you to install GitHub Desktop, a GUI app for working with git repos stored on GitHub. Install it, open it and log into your GitHub account. It should allow you to clone the ps1 repo. Move the repo to your course folder.

You can continue to use GitHub Desktop if you prefer but, as we'll see below, all of your work with GitHub can easily be handled from within the Atom editor.

2. Install a Java Compiler & Runtime System

We're going to be using Oracle's implementation of Java integrated with a library developed by the authors of our Algorithms textbook (the authors Robert Sedgewick & Kevin Wayne, heretofore SW). Some common names and symbols:

- Java SE — Java Standard Edition (produced and maintained by Oracle);
- JVM — Java Virtual Machine;
- JDK — Java Development Kit;
- JRE — Java Runtime Environment; the JVM + class libraries, etc.;
- IDE — Integrated Development Environment (a fancy editor);
- algs4 — the symbol used in naming assets related to the 4th edition of our Algorithms textbook.

Although Oracle has released Java SE 9.0.1 and Java 10, **for this course, we are going to use the previous version: Java SE 8u181. This version of the JDK can be found on the [Java SE Downloads page](#). NOTE: YOU'LL NEED TO SCROLL DOWN TO FIND THE JDK DOWNLOAD**



BUTTON FOR SE 8u181. Downloading the JDK also downloads the JRE so you don't need to download the JRE. Follow the installation instructions and you should be all set. If something goes wrong, contact a course staffer.

3. Install the algs4 Course Software Library

One of the nice things about the Algorithms book is the extensive software library developed by the authors. The library is housed in the Java archive `algs4.jar`. The simplest option for setting this up is to use the automated installation scripts developed by SW. **NB: follow ONLY step 0 of the instructions there.**

- [MacOS](#)
- [Windows](#)

Following the instructions **only for Step 0** and executing the downloaded installer script on your computer will download the `algs4.jar` library for you (along with other tools).

Setting the CLASSPATH Environment Variable

Having followed the instructions above, you should now have Java installed and SW's `algs4.jar` library installed. If something went wrong, contact a staffer.

The java compiler (i.e., `javac`) and the JVM (i.e., `java`) both need to know how to find non-standard libraries in order to translate and run your code. Java uses a special system environment variable called `CLASSPATH` to guide the search for such libraries so this variable needs to be set correctly in order to compile and run our code.

MacOS

The next few configuration steps are most easily done by typing unix commands in a *command shell*. Using the finder, navigate to `Applications -> utilities`. Scroll down to find the `Terminal` app. Install a shortcut to it in your *Desktop Toolbar* by dragging the icon there. Then fire it up by selecting the icon. You should see something like

```
muller — -bash — 56x17
Last login: Thu Aug 23 14:44:19 on ttys003
Fri Aug 24 11:14:40 EDT 2018
rm:~\> 
```

This is a unix shell, in particular, a bash shell. The shell is waiting for you to type a textual command. You don't have to master the unix command shell for this course, but it would be helpful to study it if you aren't familiar with it. Google for a tutorial.

For now type **exactly** what is shown to the right of the `>` prompts:

```
> cd
> echo "export CLASSPATH=\$CLASSPATH:/usr/local/algs4/algs4.jar" >>
~/.bash_profile
> source ~/.bash_profile
```

The first command makes your home directory the current working directory, the second command appends a key line to your bash startup file, the line defines your `CLASSPATH` variable, the third line executes that startup file.

Confirm that Java is installed by typing

```
> javac -version
```

you should see something like

```
javac 1.8.0_181          # it doesn't matter if the last 3 digits differ
```

Confirm that your `CLASSPATH` variable is properly defined by typing

```
> echo $CLASSPATH
```

you should see something like

```
/usr/local/alg4/alg4.jar
```

If you have problems with any of these steps, reach out to a course staffer. If they worked -- congrats! -- skip over the Windows instructions.

Windows

What follows is a cribbed version of instructions specified on SW's alg4 website. In order to add the `alg4.jar` library to the `CLASSPATH` variable:

1. Windows 7: Start -> Computer -> System Properties -> Advanced system settings -> Environment Variables -> User variables -> `CLASSPATH`.
2. Prepend the following to the beginning of the `CLASSPATH` variable:

```
C:\Users\username\alg4\alg4.jar;
```

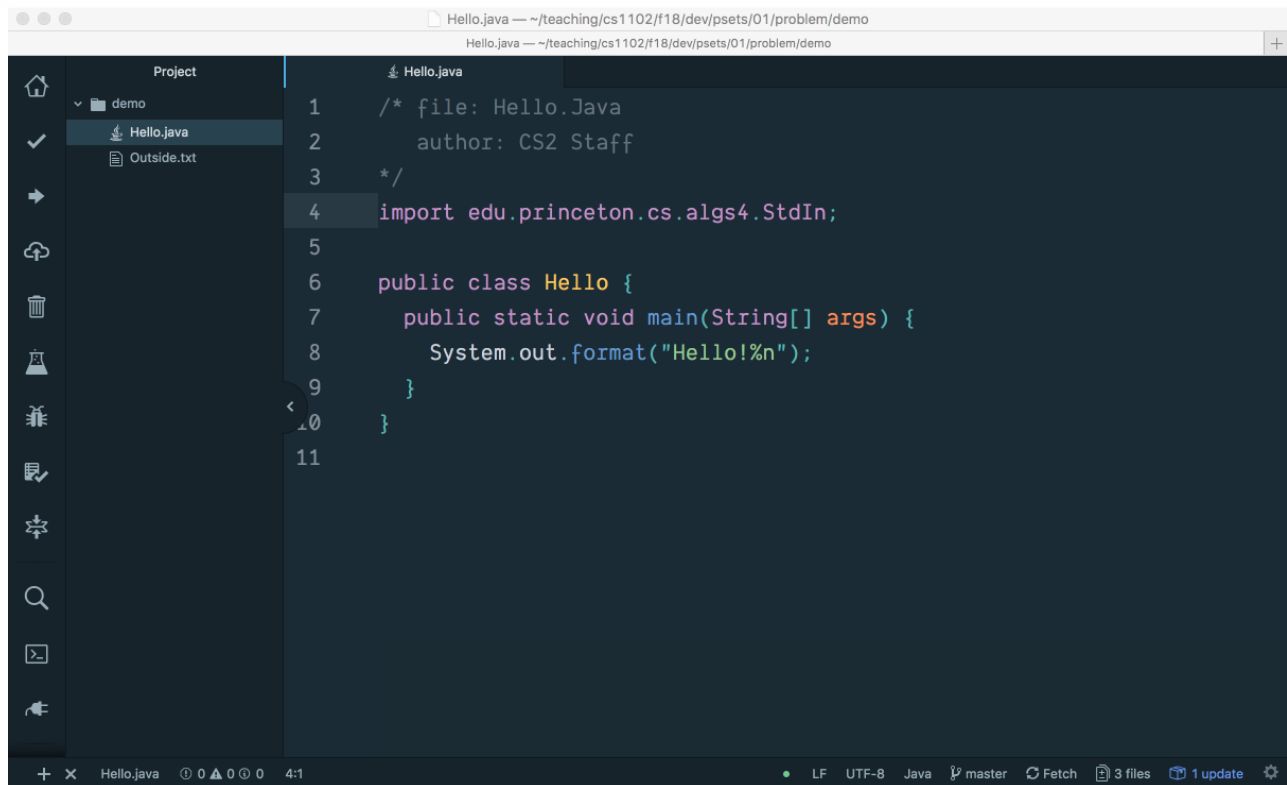
where `username` is **your username** on your computer. The semicolons separate entries in the `CLASSPATH`.

3. Click OK three times.
4. If you don't see a variable named `CLASSPATH`, click **New** and in the popup window enter `CLASSPATH` for the variable name. Then, perform the instructions above.

4. Install and configure the Atom editor

In this course we'll be using GitHub's Atom editor to write our Java code and we'll be using git and GitHub to distribute and collect materials. Fortunately, git and GitHub are nicely integrated into the Atom editor. You can find a lot of [video tutorials on using Atom](#) though you probably won't need much as it's intuitive and we'll be using only a small part of it.

1. The first step is to [install Atom](#). Move the Atom application from Downloads to your Applications folder. You'll want to keep a copy of Atom's icon in a handy place such as the Desktop Toolbar.
2. Open Atom by selecting the icon. You'll see a few welcome tabs, feel free to delete these tabs by X-ing out of them (upper right of tab). Now select `File -> Open` and navigate to the `demo/` folder which is contained within your cloned repo `ps1-your-github-ID`. If you then select the file `hello.java` on the left, you should see something like:



The Project pane on the left shows two files in the `demo/` folder:

- `Hello.java` -- a Java source file containing code;
- `Outside.txt`.

The pane on the right is the Editor pane, it displays the text contained in the selected file `Hello.java`. The 10 lines of text displayed in the pane on the right constitute a well-formed Java program. Feel free to move the cursor around with the arrow keys or with the mouse. Add a two-line comment at line 6 by typing:

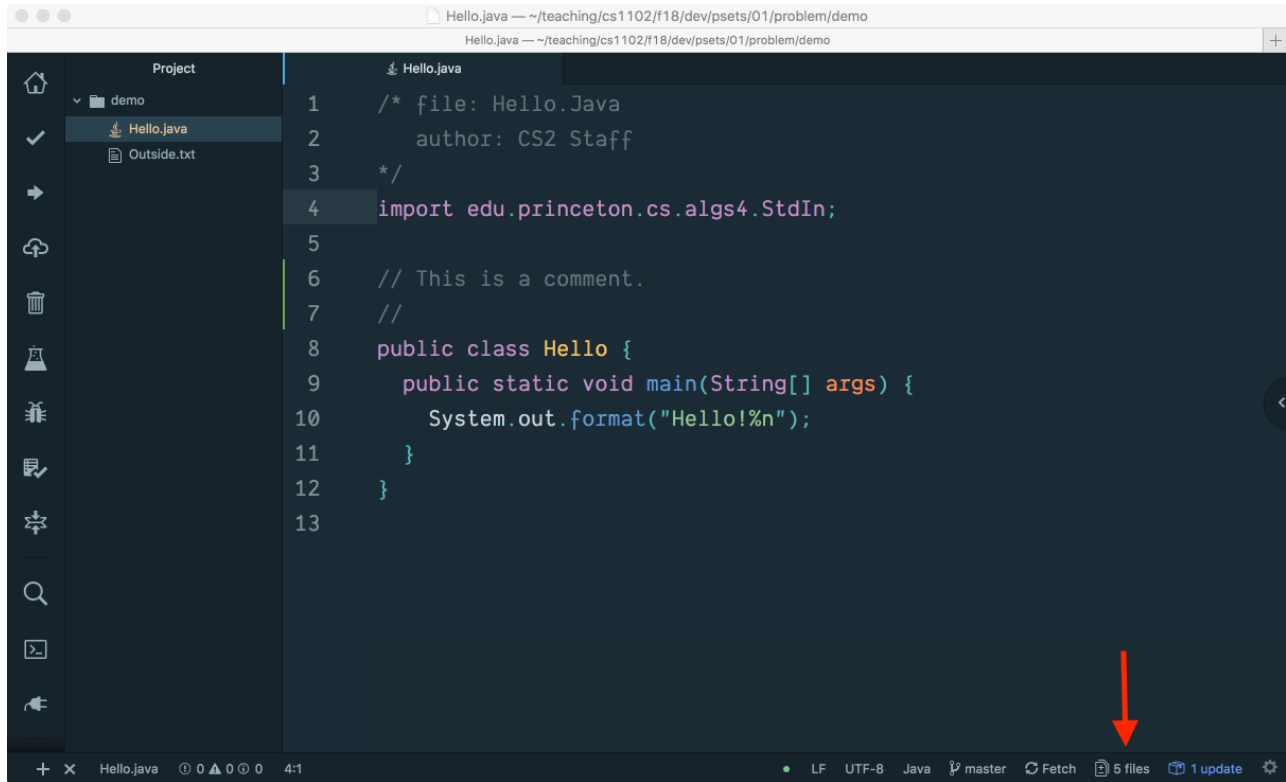
```
...  
6    // This is a short comment.  
7    //  
8    public class Hello {  
...
```

Notice that a blue dot appears on the tab to denote a file that has been changed but has not been saved. Type `Cmd-s` (MacOS) or `Ctrl-s` (Windows) to save the file.

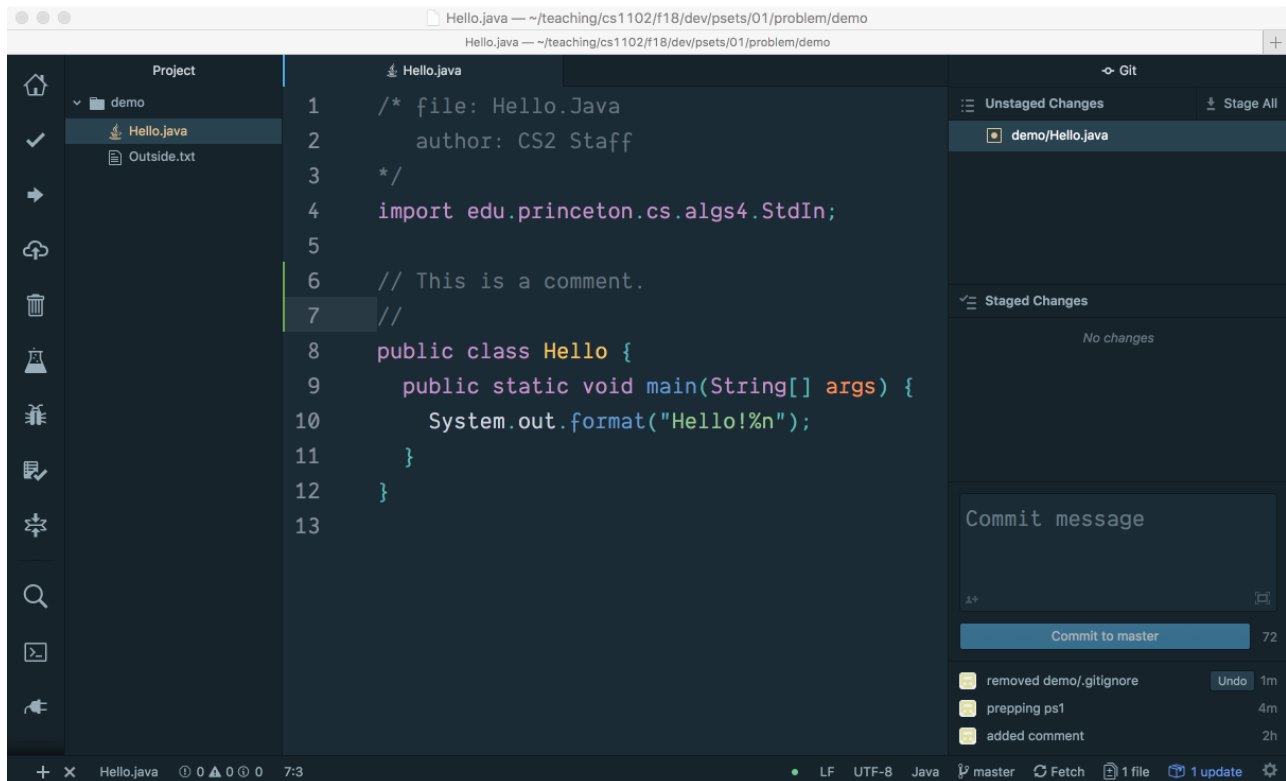
Working with Git and GitHub

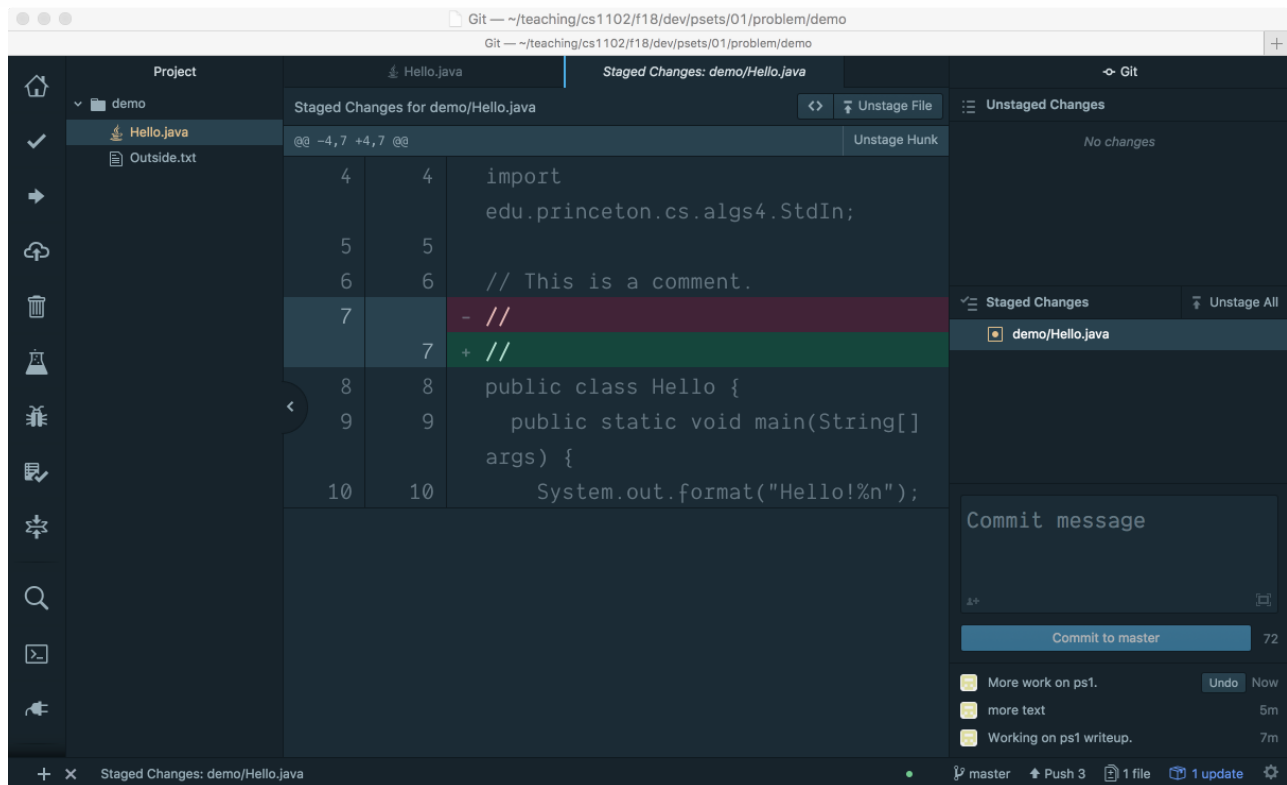
The `demo/` folder is one of several subfolders of the ps1 [git repository](#) `ps1-your-github-ID` that you cloned from your master ps1 repository on the course GitHub site. Note that after saving the file `Hello.java`, the displayed name of the file in the pane on the left has turned orange. This means that the file has changes that are not yet represented in the repo.

Toggle the small "files" icon on the lower right a few times to make a Git pane appear and disappear.

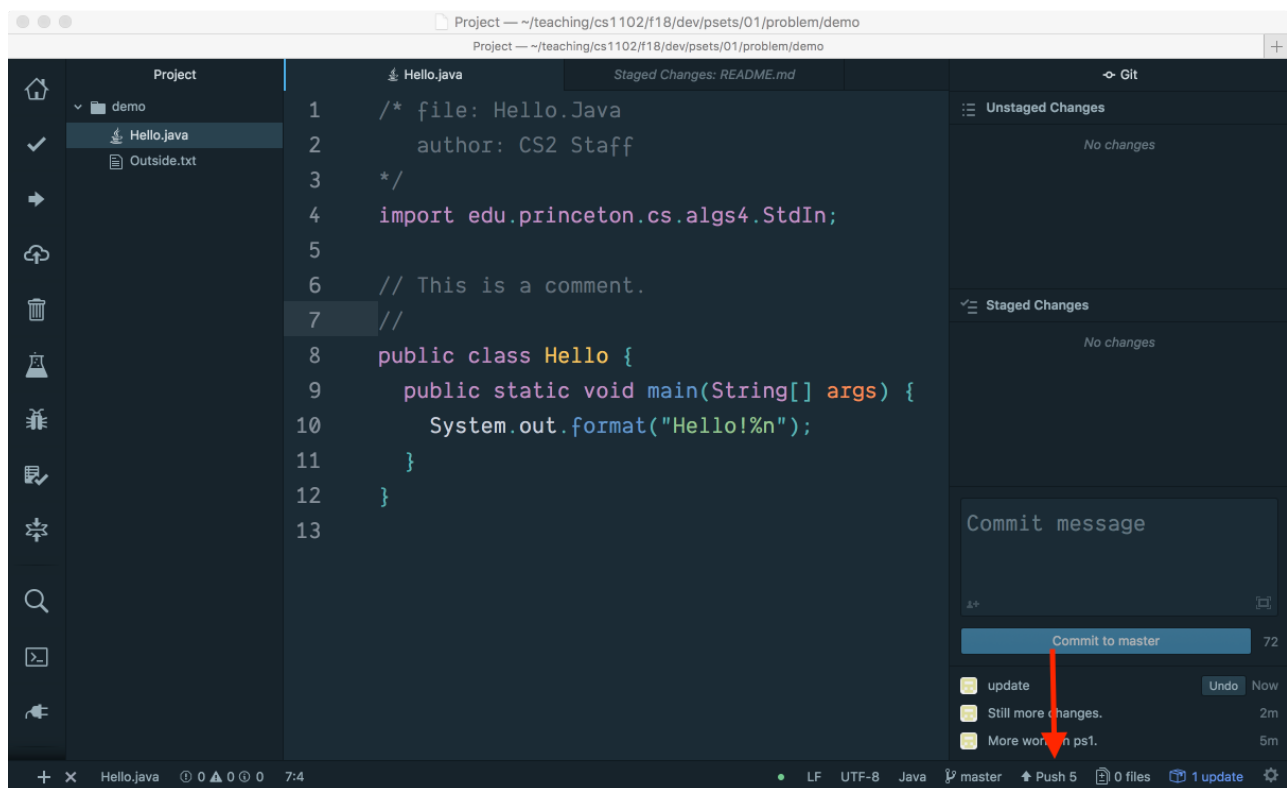


The Git pane shows the two-phase process of introducing file changes into the local git repository. Changes must first be **staged** (or **added**) and then **committed**. In order to stage the changes in file `demo/Hello.java`, double-click on the `Hello.java` entry on the right.





Before committing the staged changes, write a short **commit message** summarizing the changes. Then select the blue "Commit to master" button.



There are no orange file names in the Project pane. This means that the local repository is now consistent with the file system -- the repository contains all of your changes.

Submitting your Work

In order to submit your work for grading or for sharing with a partner, you'll need to **push** your repo to the master copy on the course GitHub site. Use the "Push" button lower right.

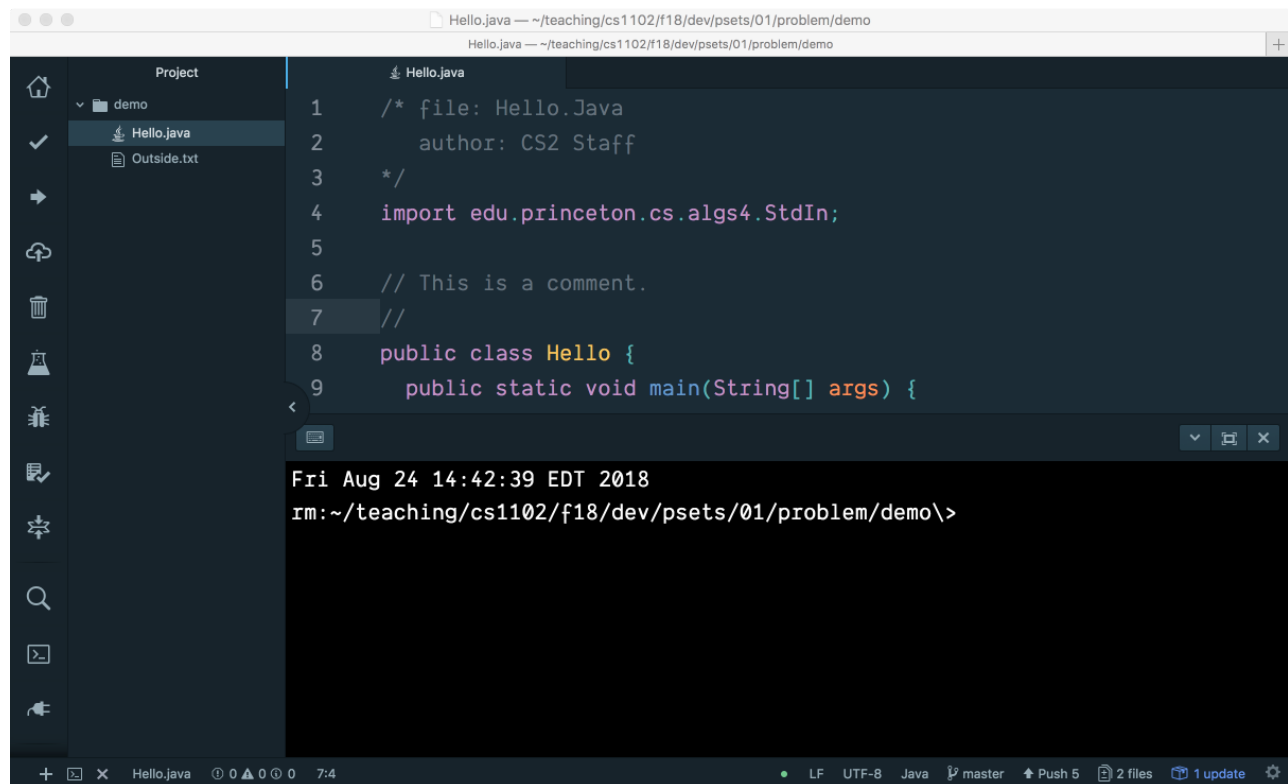
Summary

So this is the process. You'll receive a link that will allow you to clone a problem set repo. After cloning it and moving it to your course folder, you'll write and test your code. When you're done and ready to submit your work, you'll simply push your repo to the master copy on the course GitHub site. The course staff shares that repo with you. After the TAs grade your problem set, they'll post your grade and comments using the grade book on the course Canvas website.

The only item remaining is compiling and running your Java code. Both of these processes can be taken care of from within Atom.

Installing a Command Shell, Compiling & Running your Program

Atom is an extensible editor, it can be extended by anyone who writes an extension "package". Display Atom's Settings Pane by typing cmd-, (i.e., command-comma MacOS) or ctrl-, (i.e., control-comma Windows). Now select **Install** then search for the package **platformio-ide-terminal**. When it appears, install it and then delete (i.e., X-out) the Settings pane. Selecting the newly installed plus-sign icon (extreme lower left) should show you a command shell pane:

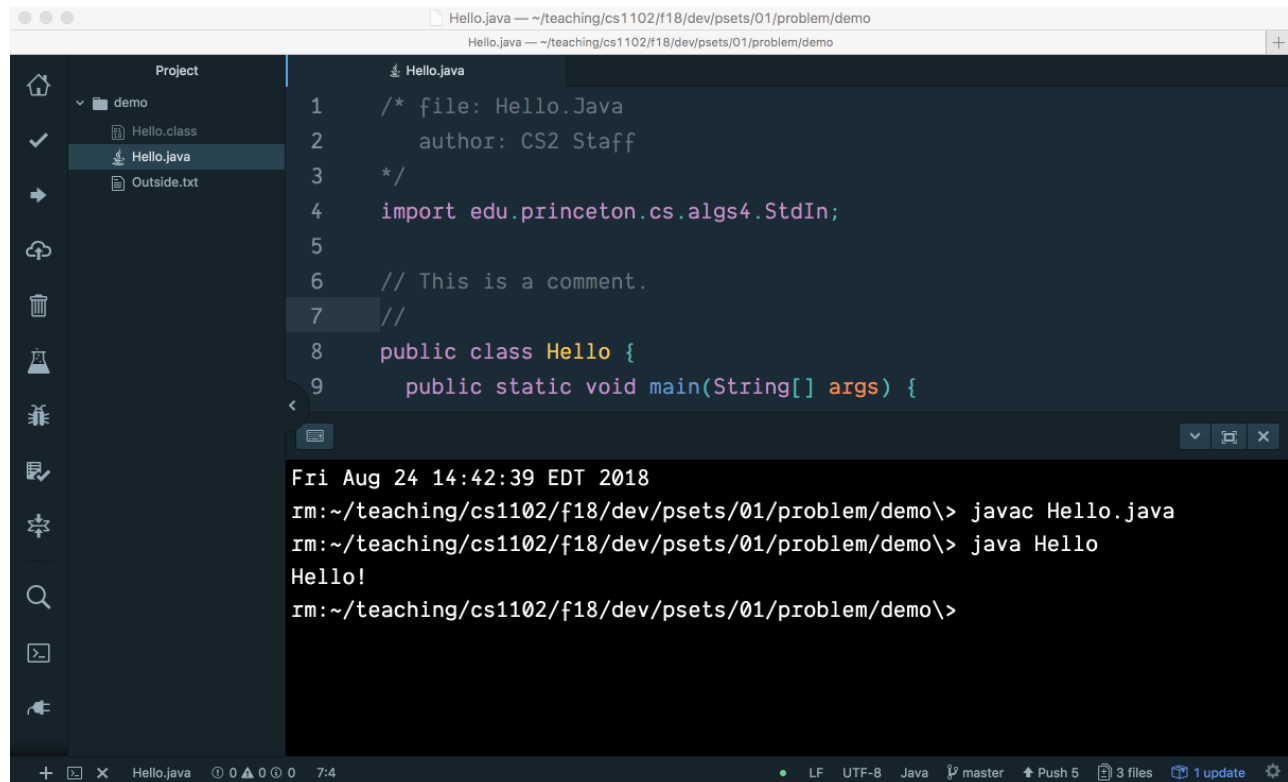


For MacOS users, the command shell is the unix bash shell. For Windows users, the command shell is the command (or CMD) shell **cmd.exe**. In either case, if you've completed the above steps correctly, you should be able to compile the above Java program by typing

```
> javac Hello.java
```

If your code is well-formed, the compiler will produce a Java byte-code file `Hello.class`. This file can be executed on the Java Virtual Machine (JVM) as follows:

```
> java Hello
```



The screenshot shows an IDE window with the following content:

- Project pane (left):** Shows a folder named 'demo' containing 'Hello.class' (grayed out), 'Hello.java' (selected), and 'Outside.txt'.
- Editor pane (center):** Displays the code for `Hello.java`:

```
1  /* file: Hello.java
2     author: CS2 Staff
3  */
4  import edu.princeton.cs.algs4.StdIn;
5
6  // This is a comment.
7  //
8  public class Hello {
9      public static void main(String[] args) {
```
- Terminal pane (bottom):** Shows the execution output:

```
Fri Aug 24 14:42:39 EDT 2018
rm:~/teaching/cs1102/f18/dev/psets/01/problem/demo\> javac Hello.java
rm:~/teaching/cs1102/f18/dev/psets/01/problem/demo\> java Hello
Hello!
rm:~/teaching/cs1102/f18/dev/psets/01/problem/demo\>
```

Note that the name of the newly created byte-code file `Hello.class` is displayed in the Project pane on the left but the name is grayed-out. This is because the repository has been configured to ignore byte-code files. How? The parent folder, i.e., the root folder of the repo, houses a special configuration file called `.gitignore`. The `.gitignore` file contains entries telling git which types of files in the folder (or subfolders) should be left out of the repo.

Tutorials

If you are new to Unix, git and GitHub and you're interested in learning how to use these systems, there are many tutorial. Two tutorials from UC Berkeley:

1. [Unix & GitHub](#)
2. [Unix](#)

Part 3: Java Code

By now you should have received an email inviting you to clone a git repository for problem set 1. If you haven't done so already, clone that repo and move the folder housing the repo into your `cs1102` folder. Then do the following 4 exercises. They are contained in folders `one`, `two`, `three` and `four`.

1. (2 Points) Java's `System` package has utilities for input and output. In particular, there's a handy function named `format` in `System.out` supporting *formatted output* (There is a companion `format` function in the `String` class: `String.format`). For example, the call

```
System.out.format("The city of %s is amazing!", "Boston");
```

would print to the console

```
The city of Boston is amazing!
```

The string `"The city of %s is amazing!"` is a *format string* and `%s` represents a hole in the string accepting a string as input. Other hole specifiers are `%d`, `%f` and `%c` for integers, reals and character (respectively). For example, the call

```
System.out.format("BEGIN--%d--%c--%f--%s--END", 12, 'A', 3.14, "Mei");
```

prints

```
BEGIN--12--A--3.140000--Mei--END
```

In the `one` folder, modify the file `Hello.java` so that it uses the `System.out.format` procedure to print `Hello world!`. Of course, this can be done with a format string with no holes at all. You are looking to write the body of the function `main` within the `Hello` class:

```
public class Hello {  
    public static void main(String[] args) {  
        // YOUR CODE HERE!  
    }  
}
```

2. (2 Points) Once a Java program `A.java` successfully compiles, the app (i.e., `A.class`) can be run on the JVM from the command shell. Java apps can receive *command line arguments* just as they can in Python and OCaml. For example, given the `Hello` code above, the body of the `main` function can access the command line arguments in the array `args`. In

```
> java Hello Boston College
```

`args.length` would be `2`, the string `"Boston"` would be in `args[0]` while the string `"College"` would be the value in `args[1]`.

In the folder `two` rewrite the `Hello` function so that it accepts a name from the command line. Running it as in

```
> java Hello Hao
```

should print

```
Hello Hao!
```

3. (2 Points) Java is notorious for its complicated IO architecture. It's flexible but it's tough on beginners! Thankfully the authors of our text provide a simpler interface for IO. In this problem you'll use the `readString` function from `StdIn`, i.e., you'll use the `StdIn.readString` function to support greetings for *two* names: one from the command line and the other from user input. All of this takes place in the `Hello` program in the folder `three`. Running that program should go as follows:

```
> java Hello Mary
Type a name: Martha
Hello Mary and Martha!
```

4. (2 Points) Maybe it's before your time, but Jack can be a bad boy. In the folder `four` there is a file named `Jack.java`. Modify it so that it accepts an integer `n` as a command line argument. Your program should use `n` to determine how many times to print the string

```
All work and no play makes Jack a bad boy
```

For example, the call

```
> java Jack 3
```

would print

```
All work and no play makes Jack a bad boy
All work and no play makes Jack a bad boy
All work and no play makes Jack a bad boy
```

For this problem you'll need two things: 1. to convert a string representation of an integer such as `"343"` to an actual `int` `343` and you'll need to call the `System.out.format` function repeatedly. For the former, take a stroll through Java's built-in `Integer` class to see if there are any pre-defined library functions that might convert a string such as `"343"` to an integer `343`. For the latter, maybe google Java for-loops. If you want style points, write it recursively!

Once your code works to your satisfaction, save the files, stage them, commit them and push them to master repo on the class GitHub site.

You're done!