

Section 1 - Edit-Compile-Run

1B.1.1 The Difference Between Source and Run

The little program we have seen so far is called the *program source*, or just *source*. It is nothing more than some text in a file named **Foothill.java**. So far, though, it is not an *executable* program - it is just some text.

It can be turned into an *executable* program, however, and that is the job of this thing we call the *compiler*. The Java compiler is an application that looks at a *text* file like **Foothill.java** and uses it to create a second *executable* file, **Foothill.class**. If successful, this second file can be executed on any computer that has a *Java Virtual Machine (JVM)*, which is part of a *Java Runtime Environment, (JRE)*.

This naming correspondence, between **Foothill.java** and **Foothill.class** helps us keep track of the program files in pairs: One that we created, the *source*, and one that the compiler created, the *executable*.

File Name	File Type	Can Edit/Print?	Can Execute?
Foothill.java	source	yes	no
Foothill.class	executable	no	yes

When we look at the *source* we are looking at the instructions that will make the program run, but we are not looking at an actual run of the program. When we *execute* the program by invoking the executable, the program runs, and what we see on the screen is the result of this run. (By the way, it is possible to create a program which executes without putting anything on the screen -- it happens all the time.)

1B.1.2 The Edit-Compile-Run Sequence

The drill is this:

1. **EDIT THE SOURCE**. Create a source file which contains the lines in the *Hello World!* program. This is an ordinary text-editing process. Let's call this file **Foothill.java**.
2. **COMPILE THE SOURCE**. Invoke the compiler (type a command or press a button). This creates a second file, which is the executable (in contrast to the source). The compiler will name this file **Foothill.class**.
3. **RUN THE PROGRAM**. Invoke the run command of your IDE, and see the happy fruit of your labor.

Remember, we are still talking in abstracts. You don't know how to do these steps yet. That's coming in a few moments.

If for any reason we want to make a change to the program, we will open the source file and modify its contents (repeat step 1). But that's not the end of it. Steps 2) and 3) must also be repeated, because after changing the source, the executable will no longer accurately reflect it. Re-compiling updates the executable from the new source, and re-running allows us to see the result of the change.

Section 2 - Errors

1B.2.1 Two Kinds of Errors

This is a good time to bring up an often over-looked point. This three-step process is not just 1-2-3, and go celebrate with a latte. The problem is, humans make errors, and programmers are mostly human.



1B.2.2 Compiler Errors

Steps 1) and 2) (editing and running the source) usually results in *compiler errors*. These are syntax errors that result from missing semicolons, misspelled words or variables you forgot to declare.

Important

Don't curse these errors. They are your friends. They are easy to fix, because the compiler usually tells you approximately where the error is. When you look closely, you will dope-slap yourself and make the correction.

Learn to love *compiler errors* because once they are eradicated the real trouble begins.

1B.2.3 Run-Time Errors

After your program compiles, you can start to sweat. If at that point it doesn't do what you expected, then you are experiencing what we call a *run-time error*. These are *not* your friends. But you will have to learn to love them too, because you will be spending most of your programming life chasing down and fixing *run-time errors*.

The problem with run-time errors is, unlike compiler errors, there is not always a good message indicating where your program went astray. Sometimes the Java virtual machine will spit out such an error, and it may be helpful, but as often as not, it will be too oblique to understand. You have to figure this out by yourself. Eventually you will learn to use debuggers, but even then it isn't always easy.

This is the true test of your mettle as a programmer. If you enjoy the hunt, and have a fight-to-the-death attitude regarding run-time errors, you will be a true programmer. If not, you should not seriously consider programming as a main vocation.

Building a Java Project in Eclipse

ECL_JAVA.1 Overview of the Eclipse Development Cycle

Every program that you do for practice or homework in this course has to be compiled, debugged and executed in some Java Integrated Development Environment (*IDE*). In this course that *IDE* is *Eclipse*. This handout is a reference to help you create and complete a console application in this *IDE*. You should use a simple program like *Hello World!* as a test the first time you do this:

```
public class Foothill
{
    public static void main(String[] args)
    {
        System.out.println("Hello World!");
    }
}
```

The general idea is:

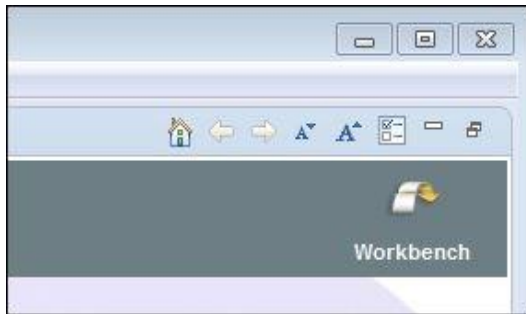
1. Start *Eclipse*
2. Create a **New Project**
3. Create a **New Class** inside the project
4. Perform the usual **Edit-Compile-Run** sequence repeatedly, until the program executes without error.
5. Save the project and exit (for later work or assignment submission)

ECL_JAVA.2 The Steps in Detail

Here's how to start and complete a Java Project in *Eclipse*. These screen shots were taken from a Windows PC, but the details are identical on a Mac.

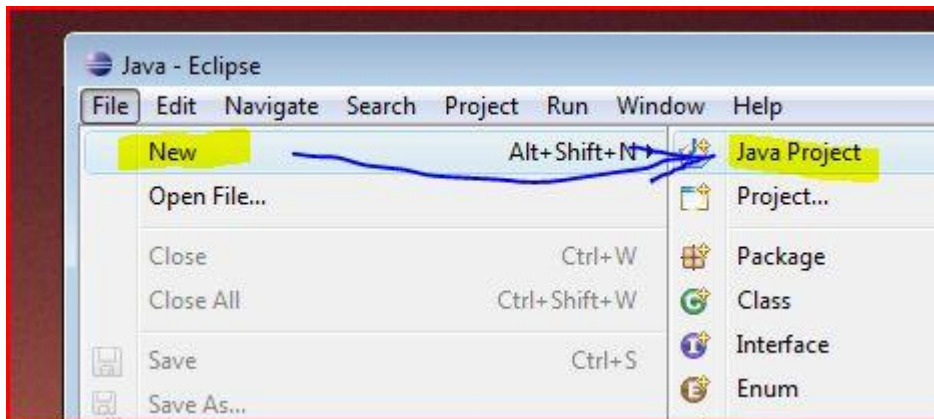
Start Eclipse

I'll reiterate: if this is the first time you run Eclipse, you may have to manually ask to go to the *Workbench*. (This step will not recur in future launches.) Click on the *Workbench* icon:

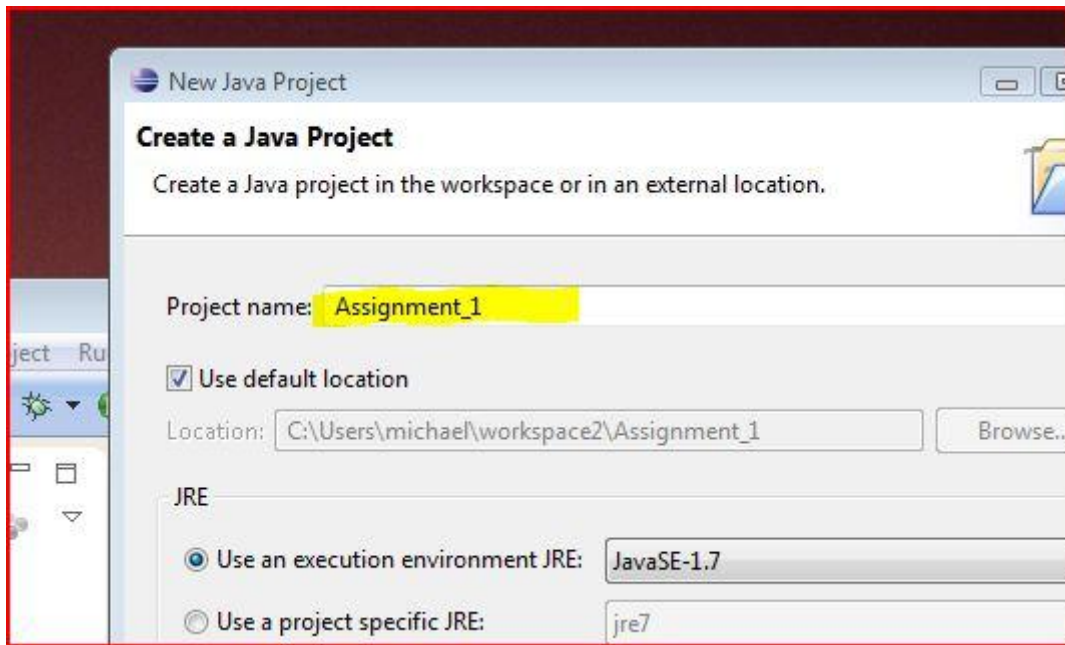


Create a New Project

Start a new project using *File* → *New* → *Java Project*:



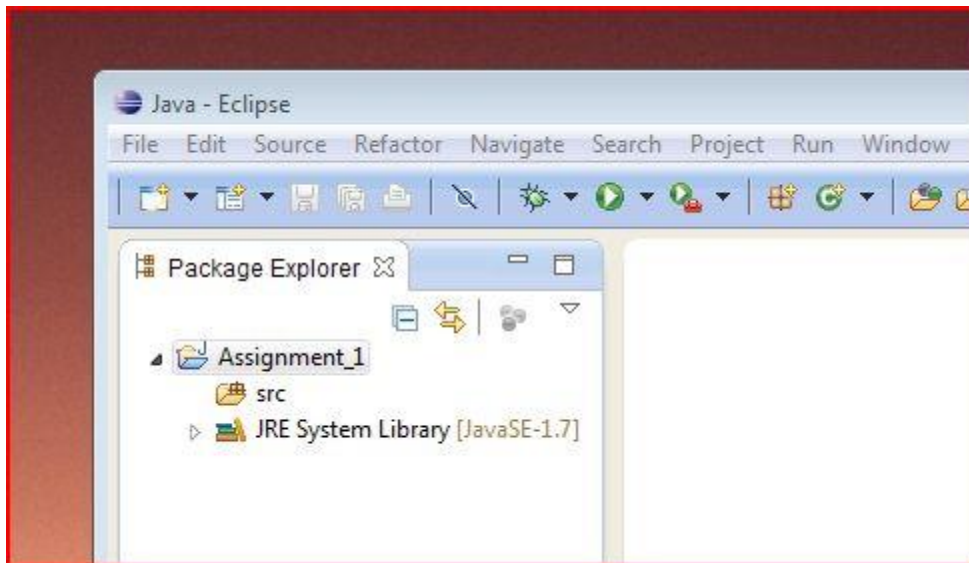
In the next screen give the project a wonderful name, like **Practice_1** or **Assignment_1**, then press *Finish* (okay to use default selections):



→

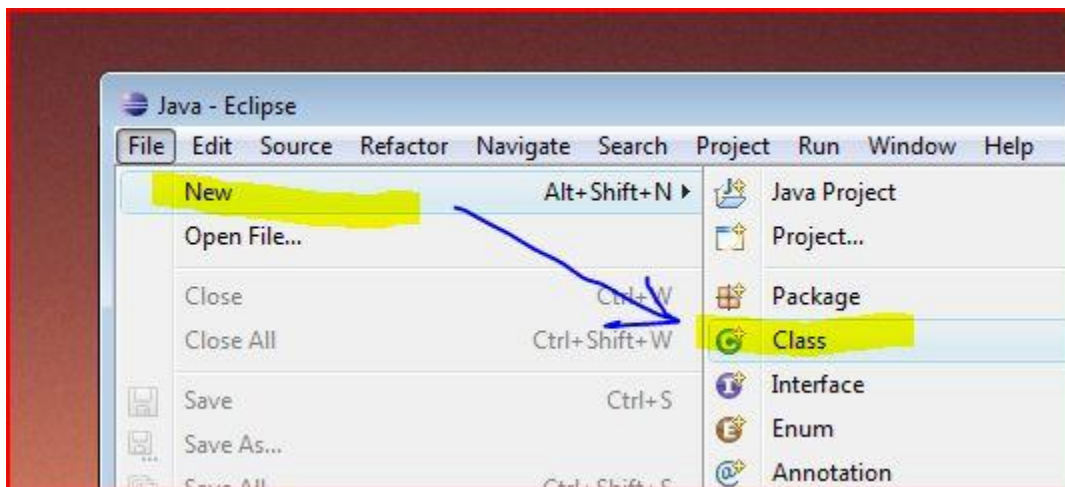


You can see the new **Assignment_1** project in the *Package Explorer*. You can expand it; there isn't much there yet.

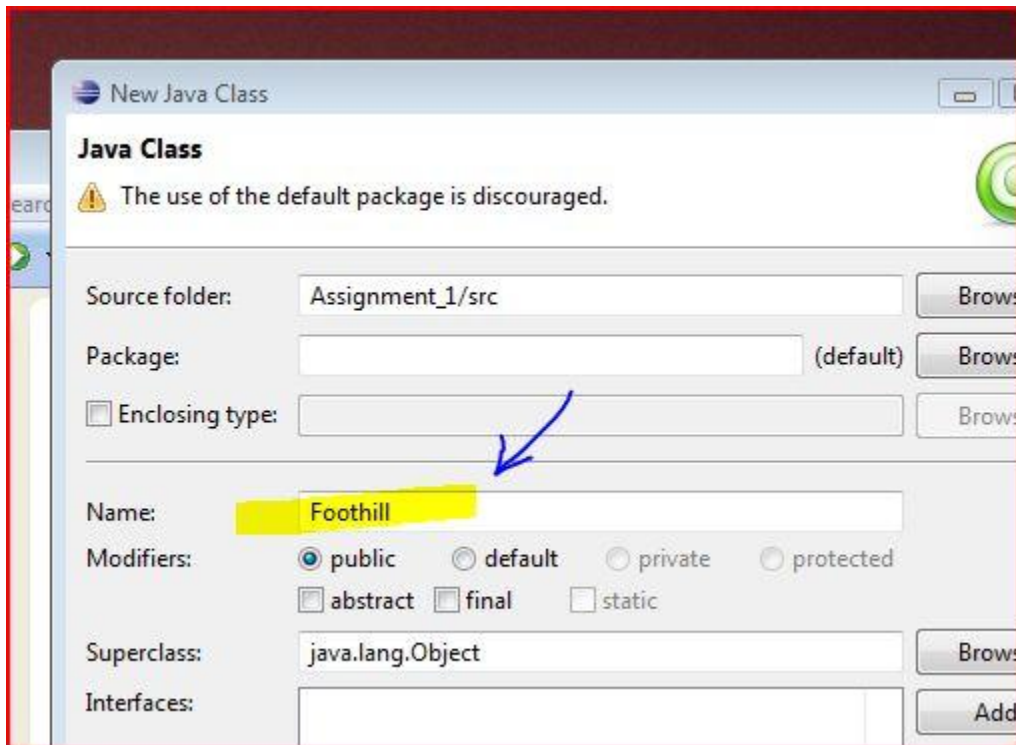


Create a Main Class in the Project

We have to create a **.java** file in which to enter our source program, which means creating a new *class*. So create a new class using **File** → **New** → **Class**:



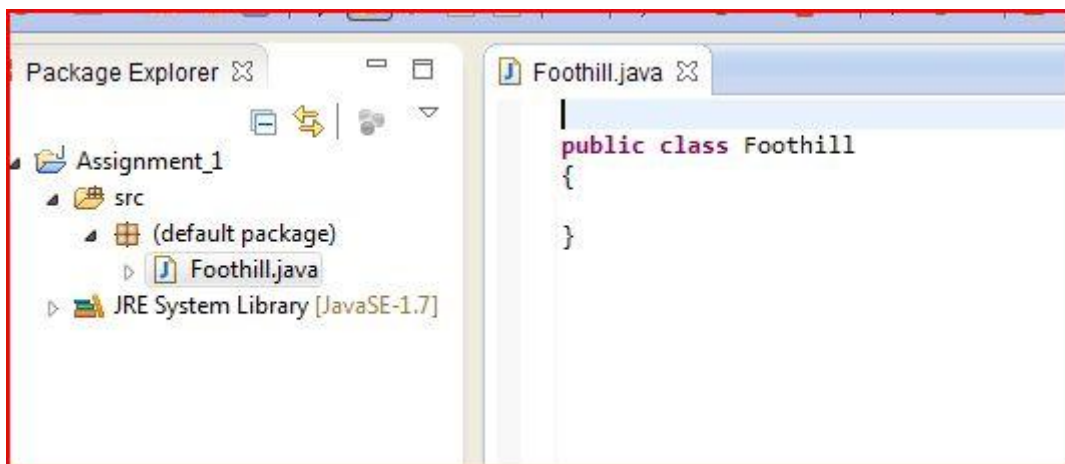
Give the class a super-fantastic name, such as ***Foothill***, then press ***Finish***:



→

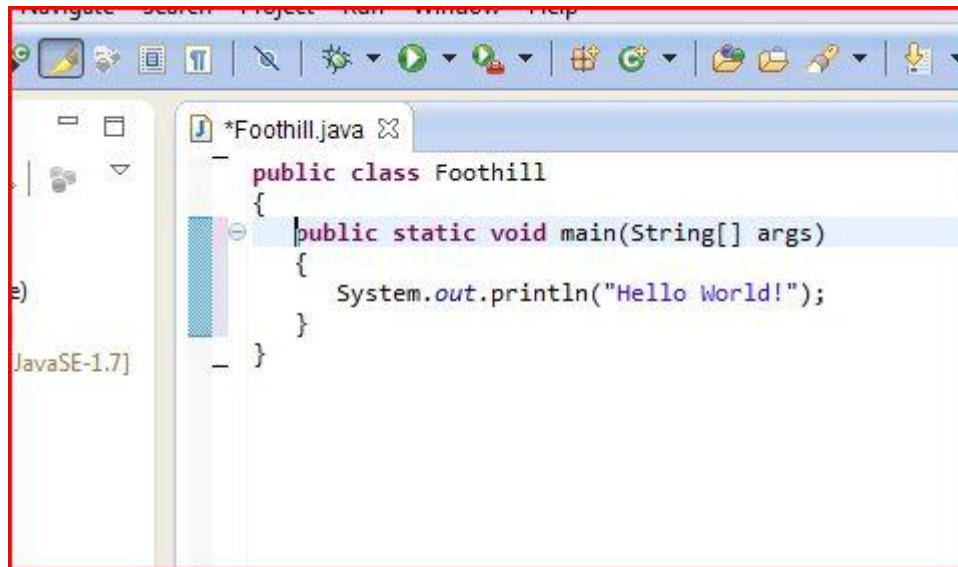


We now see the file **Foothill.java** in the left **Package Explorer** window, and this file is open in the editor in the center. It already has some stuff in it.



Write (or just Paste) Your Program

Replace that "stuff" with the ***Hello World!*** program above (or whatever source you want to write).

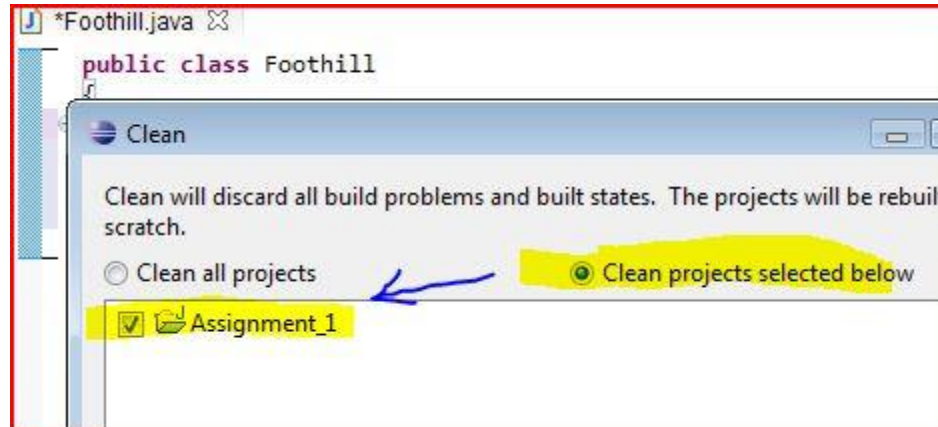
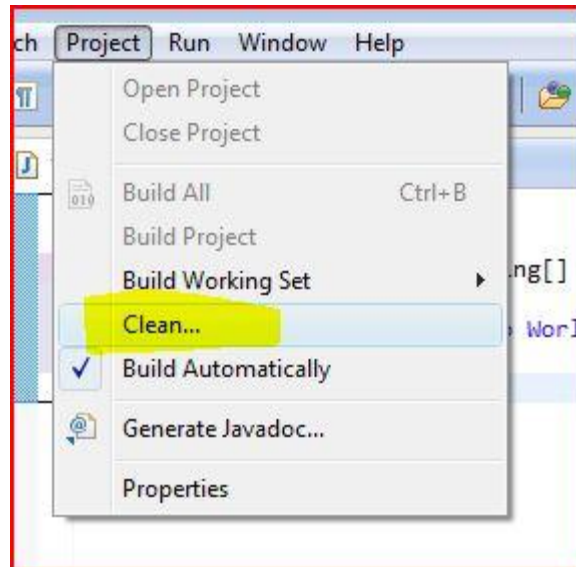


This is your program source.

Even if you are going to do a more complicated project, as you will do in subsequent weeks, always start by pasting this simple Hello World project into the window and getting it to compile. This way you will know that the project is properly constructed and any future errors will be in your programming, not in the project configuration.

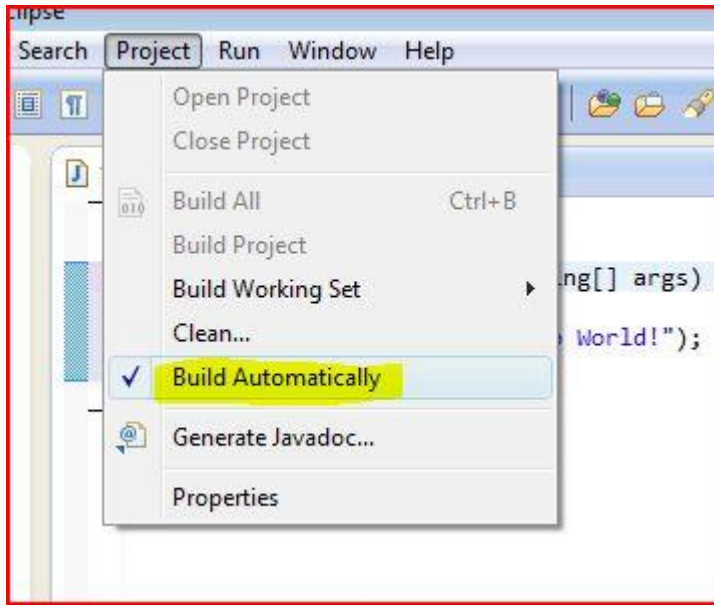
Clean

An optional step that sometimes corrects or prevents errors is to first do a ***Clean*** from the ***Project*** menu. It won't hurt to always do this, but it takes a few extra seconds. After you are comfortable with the assignment building you can skip this step and only come back to it if your program seems to be "stuck" in some bad place.

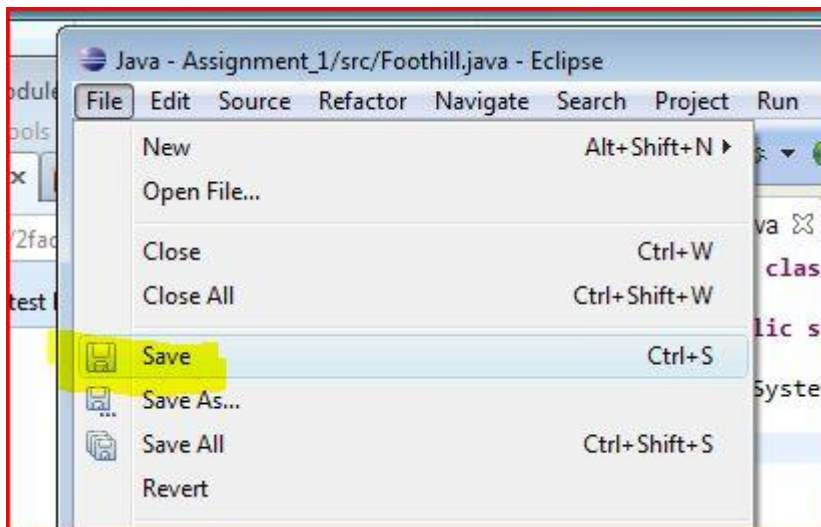


Build and Save

In the Project menu, if you don't have **Build Automatically** checked, you can check it -- or you can select **Build Project**, manually. If **Build Automatically** is already checked, don't select it (or you would be turning it off). With this option checked, you will never have to do a **Build Project** step; *Eclipse* will always **Build** (compile) your program automatically whenever you modify or save the file.

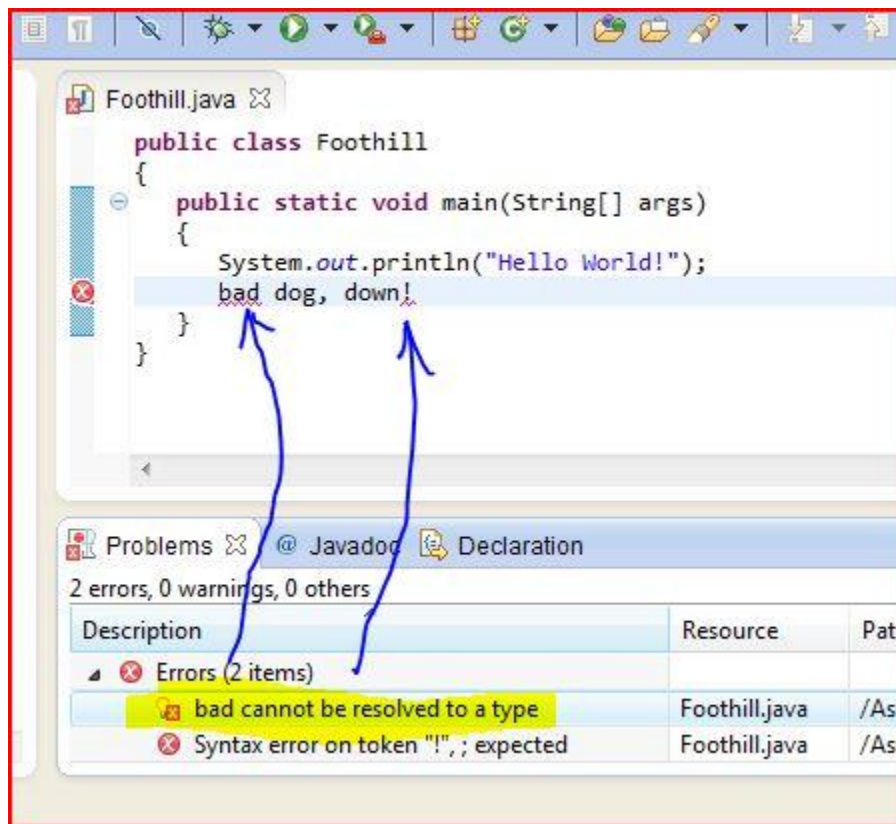


Finally, this is the correct time to *Save* your file. Saving will cause the project to be rebuilt (which you usually want - it shows any new errors you introduced in your most recent changes):

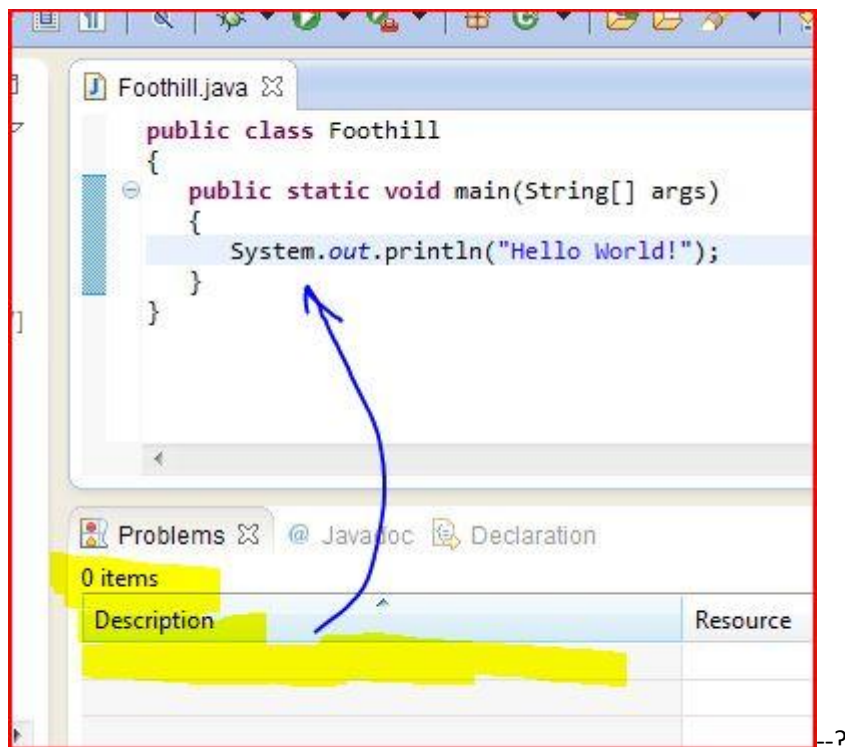


Fix Compiler Errors

You may see *warnings* (often you can ignore them) or *errors* (you will have to fix those). Errors or warnings appear in the **Problems** window at the bottom:

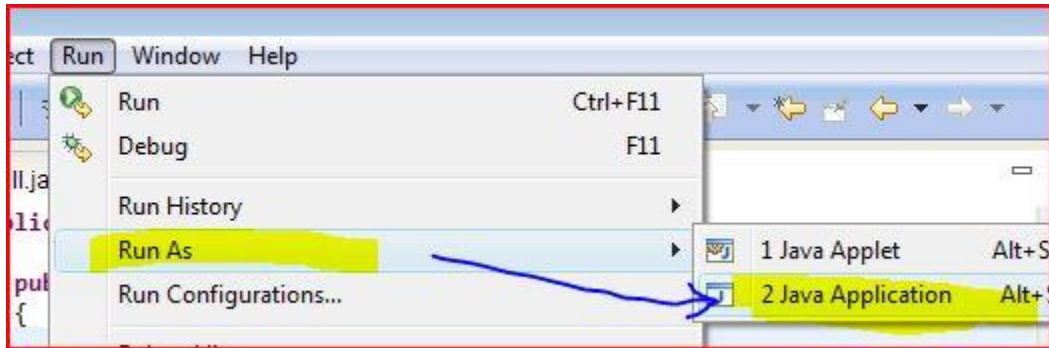


If you double click on the error, it will take you to the line that needs fixing. Fix it and *save* (causing a new build) and the error will disappear:

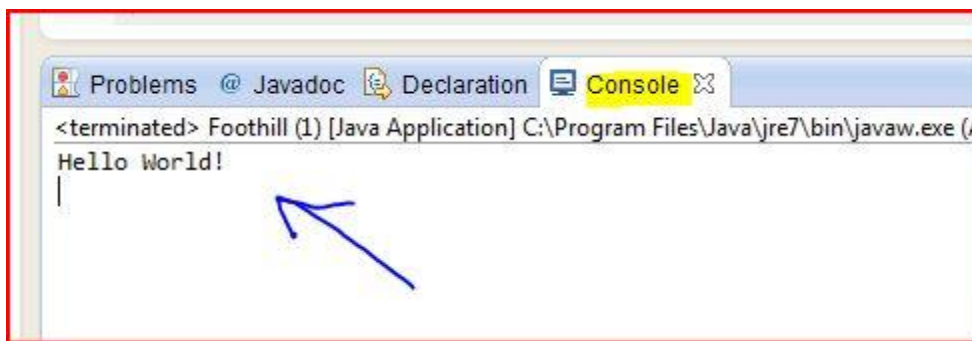


Run It

With the **Assignment_1 Project** selected in the left **Package Explorer**, go to the **Run** menu and **Run As a Java Application**:



Look at the window below the source - it is the **console**. The **console** should have your run.

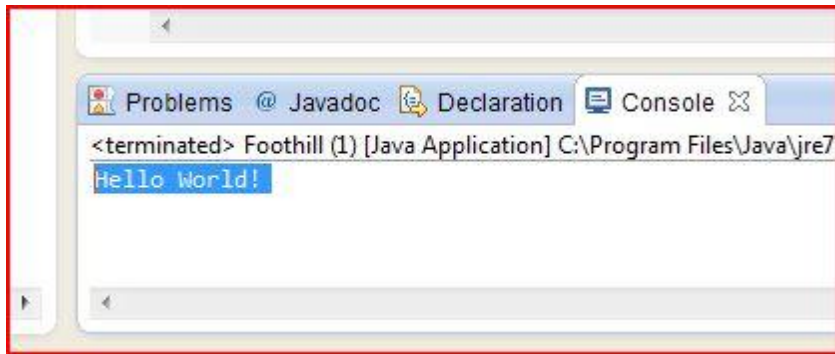


You are looking at your *program run*.

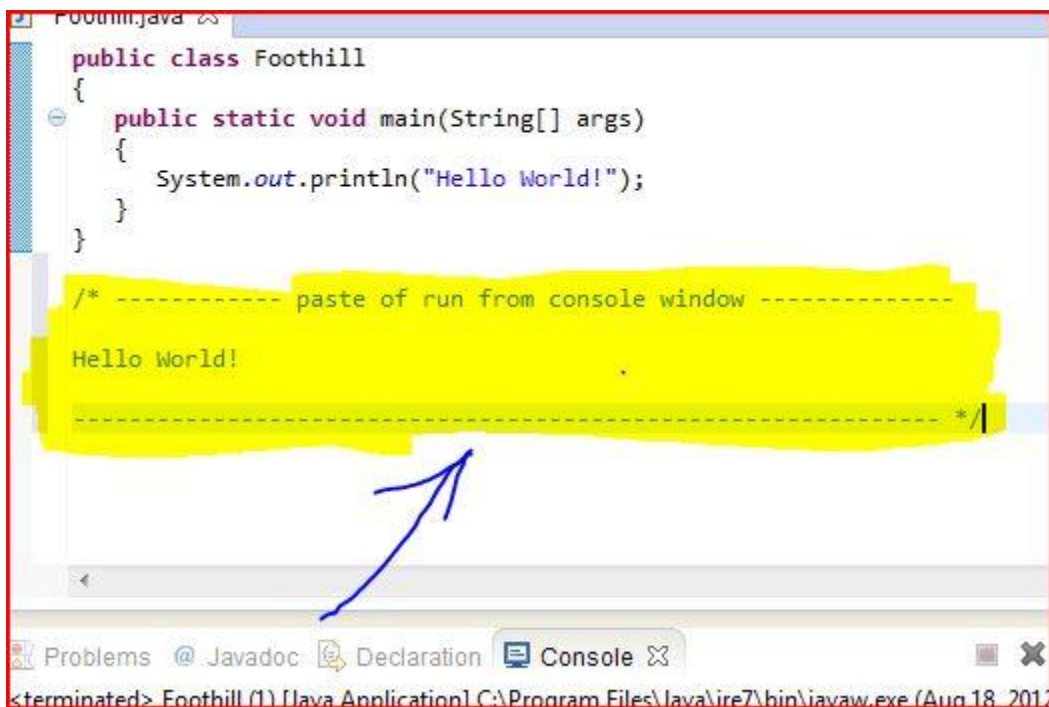
If it does not show this, you have a logic (run-time) error. You'll have to go back and re-edit the source, removing the error, then recompile and rerun. Do this until you have the program working perfectly.

Copy the Run and Paste it into the Source or a Submission File

In order to copy the run so you can hand it in with your source, select all the text in the **console** window (**right click**, then **Select All**, and follow this with a *copy* operation (**command-C** or **Edit → Copy** or **right-click → Copy**).



You might paste it at the end of your source file and place comments around it, like so:



This way, you can submit the source file which will contain, at its end, a copy of your run.

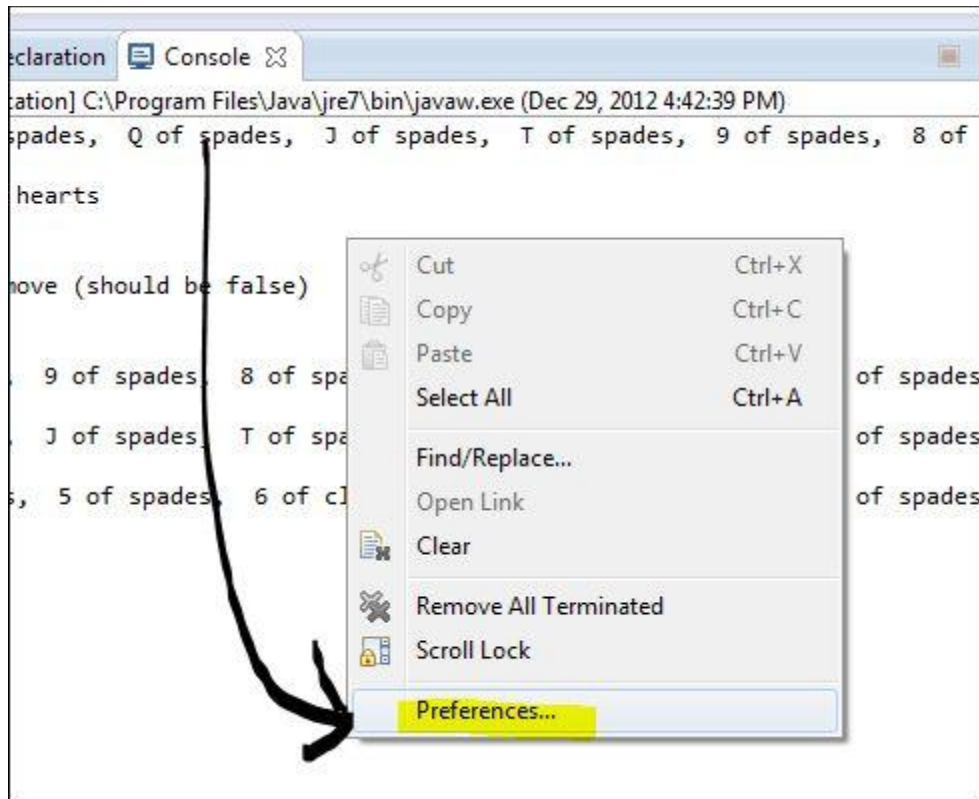
And that's it. This is the procedure you need to go through for each program. It seems long, but most of this will be automatic in a couple weeks and it will literally take you minutes or seconds to run through these steps.

Save your project and exit *Eclipse*.

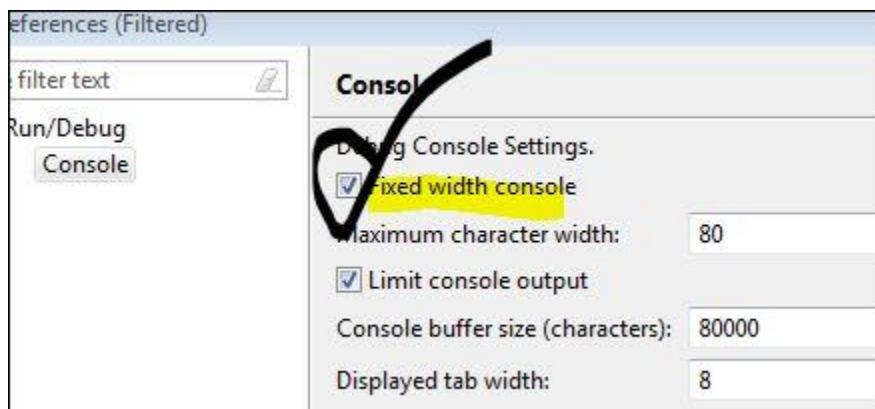
ECL_JAVA.3 Setting Console for Word Wrap

When you have programs that produce long output, you'll need to set the console to wrap, otherwise, certain output will appear as a mile long line, only the first few words of which will

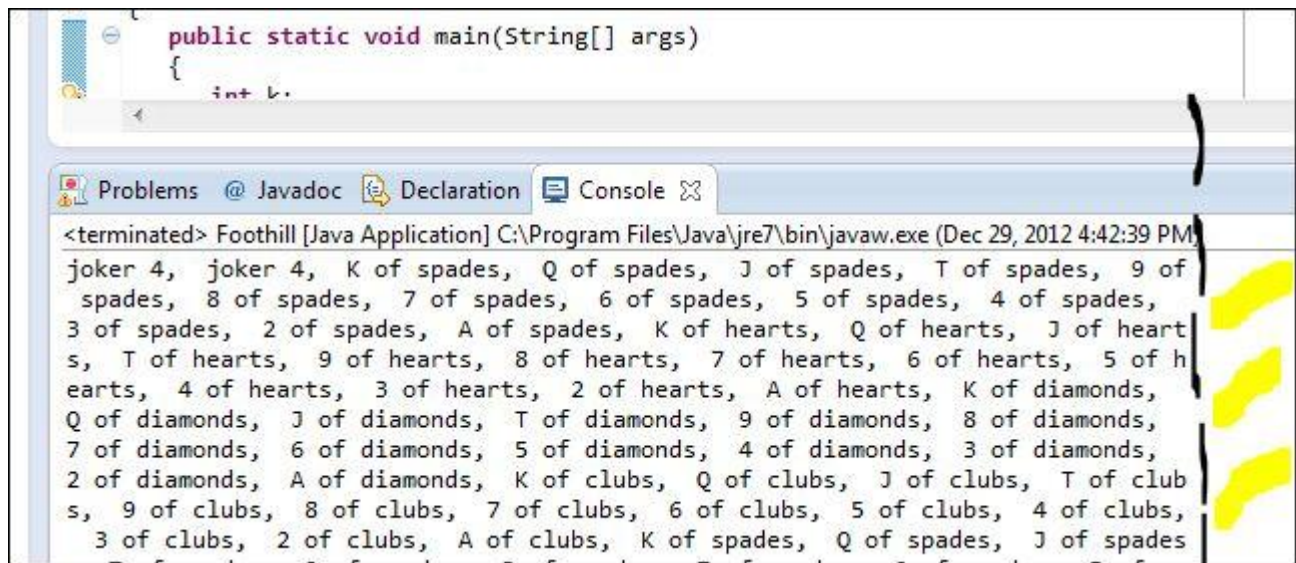
be visible. This is handled by right clicking anywhere *inside* the console (but not on the console *tab*) and selecting **Preferences...** :



Next, check the box that says **Fixed width console**, and be sure the **Maximum width** is set to **80**:



Click **OK** and all current and future text will appear, wrapped:

A screenshot of the Eclipse IDE interface. The top editor shows a Java class with a `main` method. The `Console` tab is active, displaying the output of the program. The output lists 52 playing cards in a specific order, grouped by suit: spades, hearts, diamonds, and clubs. The text is wrapped across multiple lines. On the right side of the console output, there are three yellow rectangular highlights.

```
public static void main(String[] args)
{
    int k.
```

<terminated> Foothill [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (Dec 29, 2012 4:42:39 PM)
joker 4, joker 4, K of spades, Q of spades, J of spades, T of spades, 9 of
spades, 8 of spades, 7 of spades, 6 of spades, 5 of spades, 4 of spades,
3 of spades, 2 of spades, A of spades, K of hearts, Q of hearts, J of heart
s, T of hearts, 9 of hearts, 8 of hearts, 7 of hearts, 6 of hearts, 5 of h
earts, 4 of hearts, 3 of hearts, 2 of hearts, A of hearts, K of diamonds,
Q of diamonds, J of diamonds, T of diamonds, 9 of diamonds, 8 of diamonds,
7 of diamonds, 6 of diamonds, 5 of diamonds, 4 of diamonds, 3 of diamonds,
2 of diamonds, A of diamonds, K of clubs, Q of clubs, J of clubs, T of club
s, 9 of clubs, 8 of clubs, 7 of clubs, 6 of clubs, 5 of clubs, 4 of clubs,
3 of clubs, 2 of clubs, A of clubs, K of spades, Q of spades, J of spades

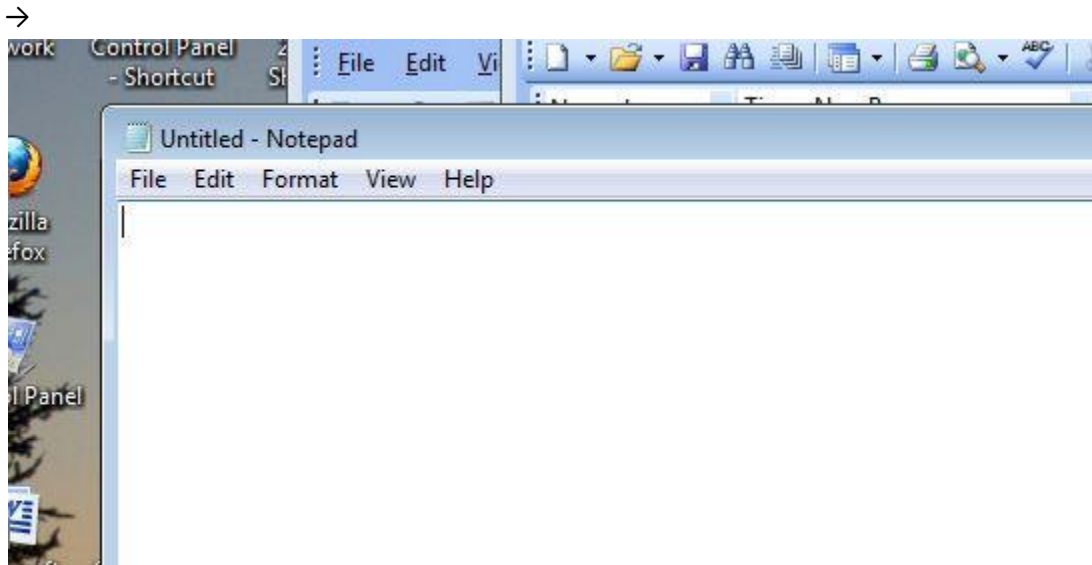
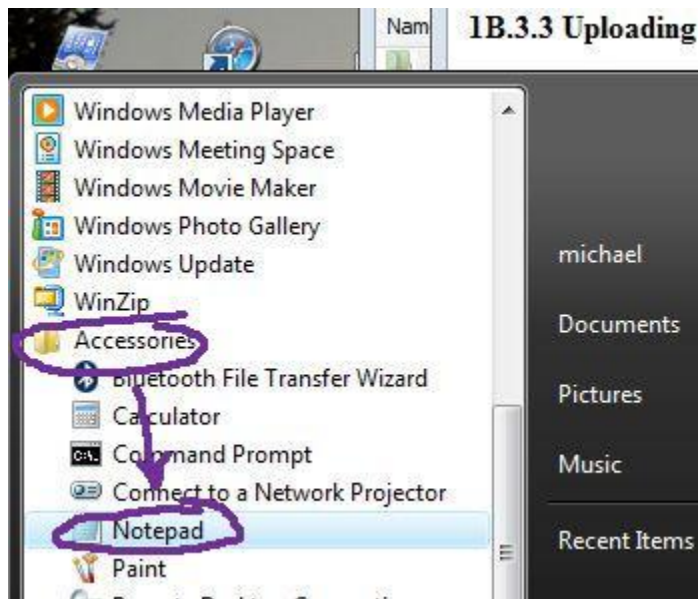
Preparing a Program for Homework Submission (Windows)

This page is for students who are using a **Windows PC**. If you are using a Mac, you can skip this page.

Step 1 - Create a .txt File for Submission Using Notepad

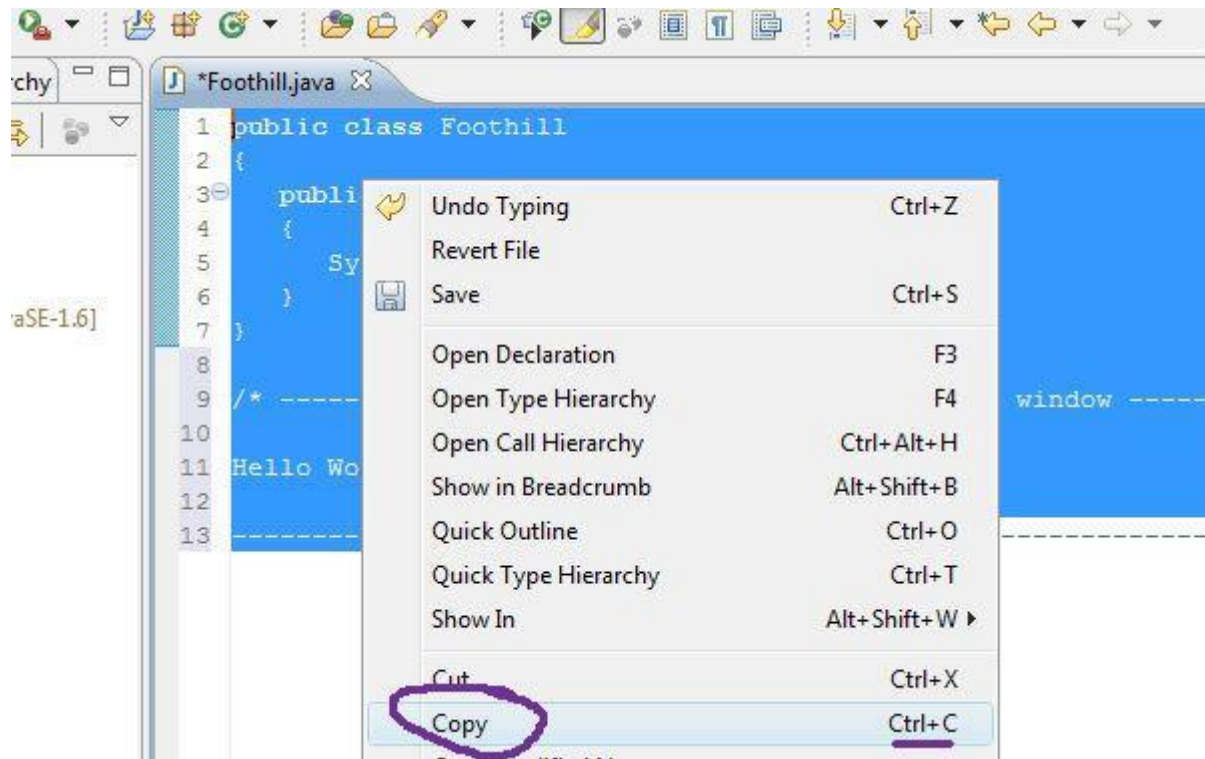
After your program is working and you've pasted a copy of the run at the end of the source, you're done with your development. But you still need to prepare a file for upload and submission into the course site. This module walks you through that process for Eclipse on MS Windows.

Save a separate copy for upload. Do this using *Notepad*. Open the *Notepad* accessory and you will see a blank document:

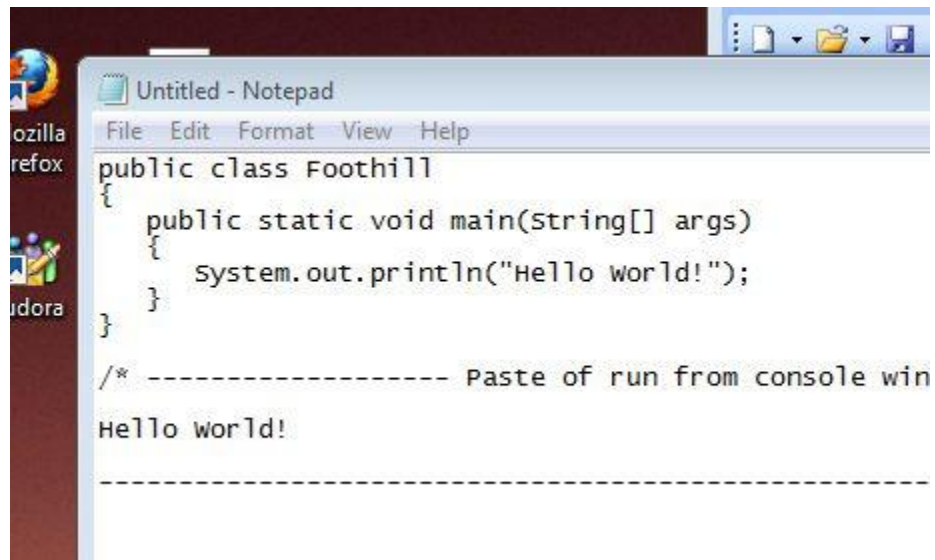


Step 2 - Copy the Finished Program into the Notepad File

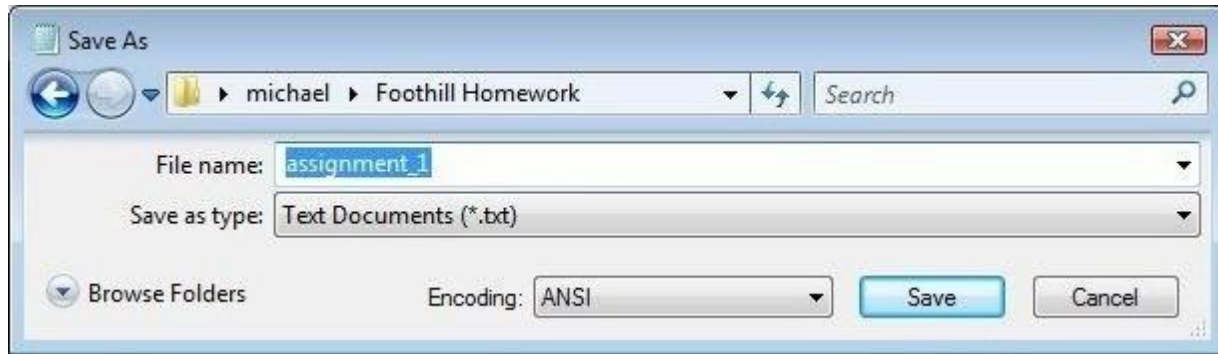
Go to your *IDE*, select all the text and **Copy** it into the clipboard:



Go back to *Notepad* and **Paste** it into the blank document:

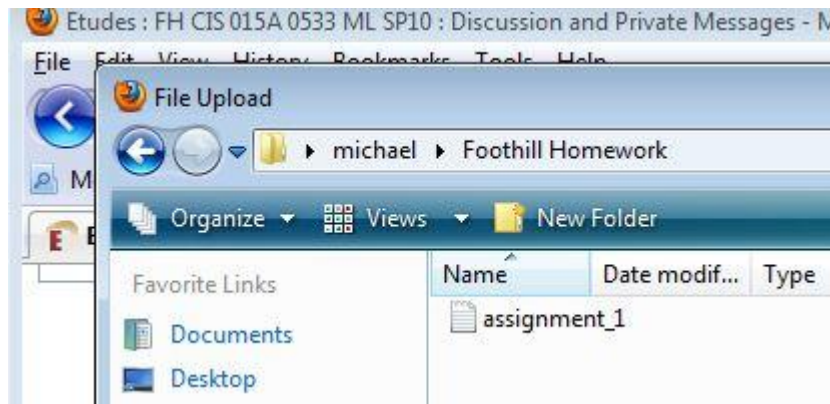


Save the file as **.txt** file with a descriptive name (*Assignment_1*, *Assignment_2*, etc.). The **.txt** extension is the default, so you don't have to type it, even if you remove the **.txt** that is given to you:



Step 3 - Upload and SUBMIT the .txt File

Finally, go to Etudes → Assignments and select the assignment for that week. Eventually, you'll see a link to upload your assignment. When you click that link, you will be given a **browse** dialog box. Navigate to the folder where your *Notepad* file, *Assignment_1* (or whatever you named it), is located:



Select the assignment for upload.

Note

You have to actually submit, i.e., **FINISH**, the assignment. Uploading is only the first part - if you don't submit, I won't receive it and you'll be charged late points. No one wants that to happen.

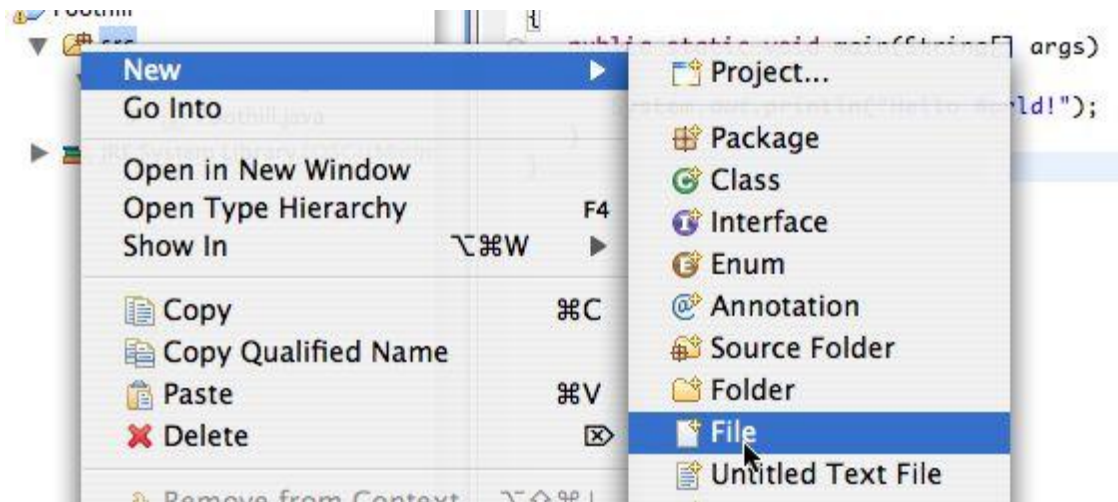
Once you have uploaded and submitted the assignment, you are done. Wait a few days for your faithful instructor to grade it.

Preparing a Program for Homework Submission (Mac)

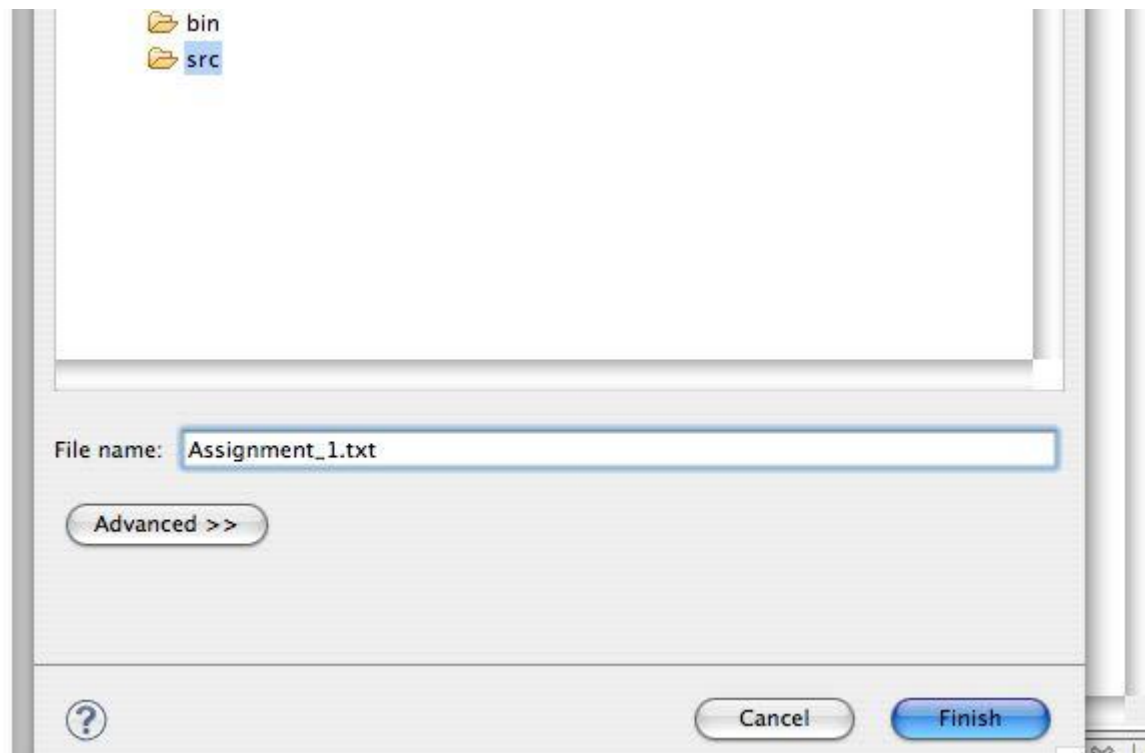
This page is for students who are using a **Mac**. If you are using a Windows PC, you can skip this page.

Step 1 - Create a .txt File for Submission

After your program is working and you've pasted a copy of the run at the end of the source, you should save a separate copy for upload. Do this by right clicking on **src** and selecting **New** → **File**:

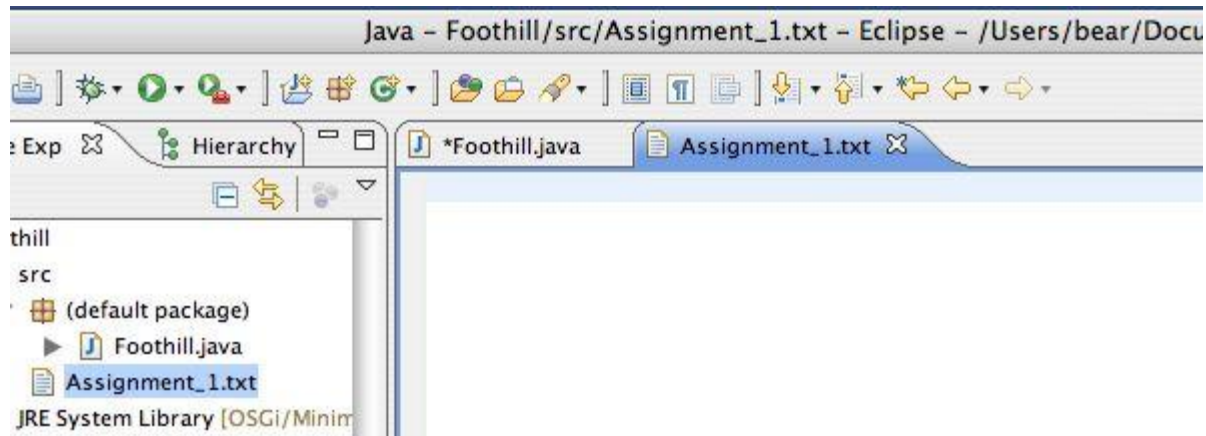


Give it a name like **Assignment_1** or something descriptive. Also, give it a **.txt** extension:

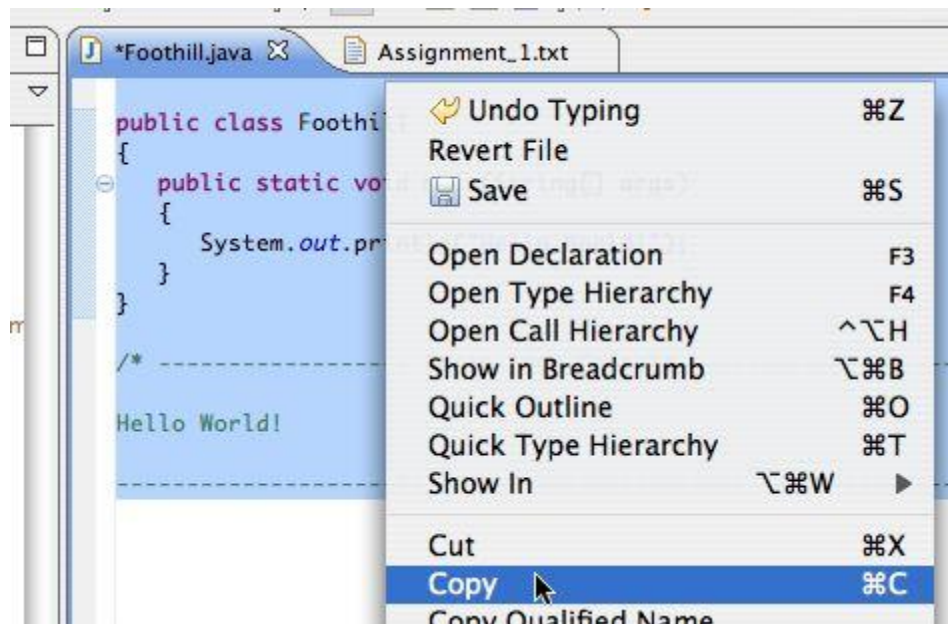


Step 2 - Copy the Finished Program into the New .txt File

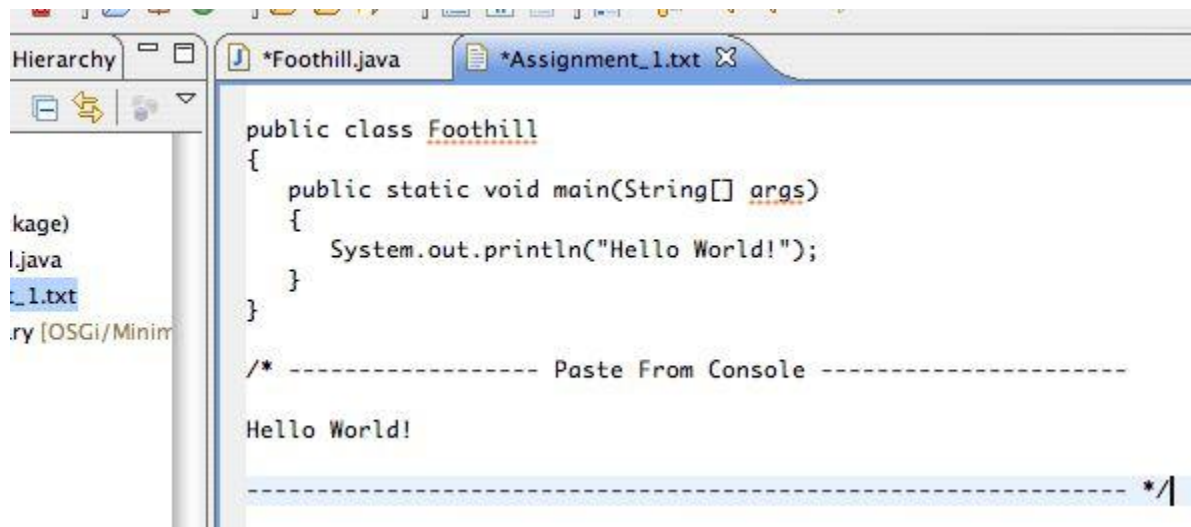
The new document will be blank, and it should show up in your main window:



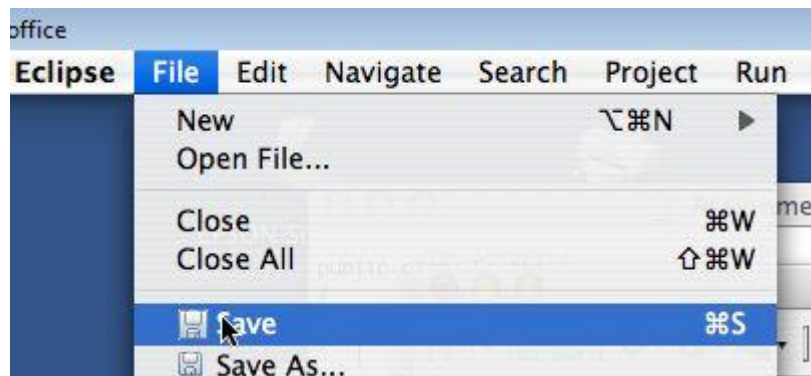
Next, go to your completed **.java** file, **select all** the text and **Copy** its contents into the clipboard:



Go back to the **.txt** file and **Paste** it into the blank document:

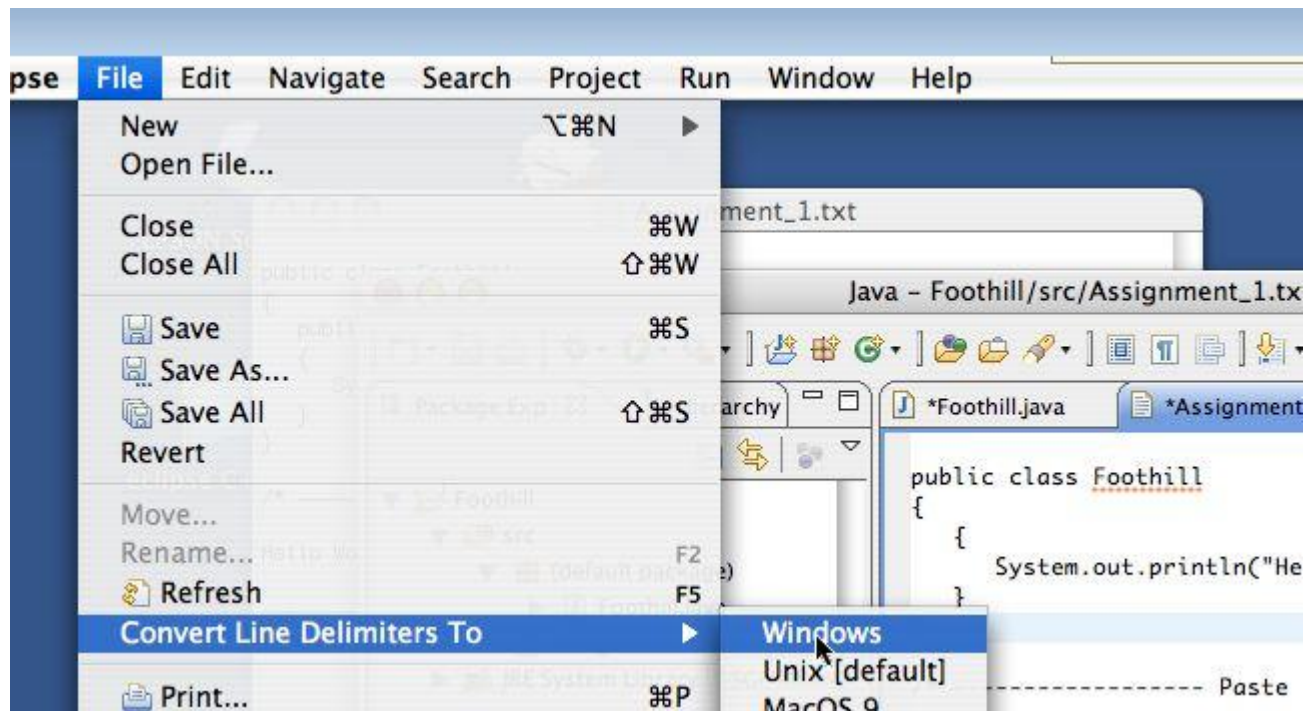


Save your new .txt document:



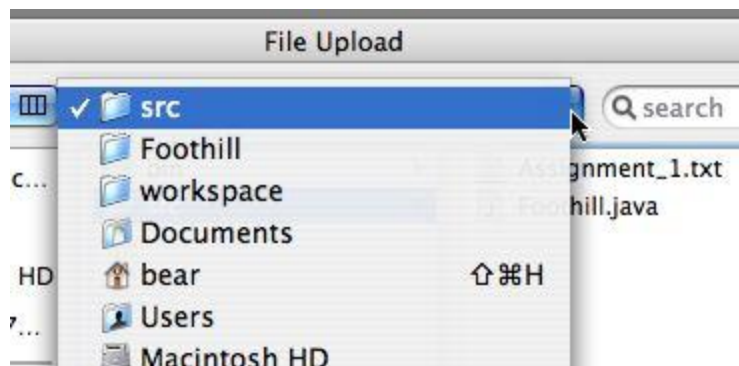
Step 3 - Convert To Windows Line Delimiters

This step is very important. With the .txt file open (in the center) and selected in your explorer window (on the left), you *must* **Convert Line Delimiters to Windows** or I will not be able to grade it. Do this directly from the **File** menu in one quick step:

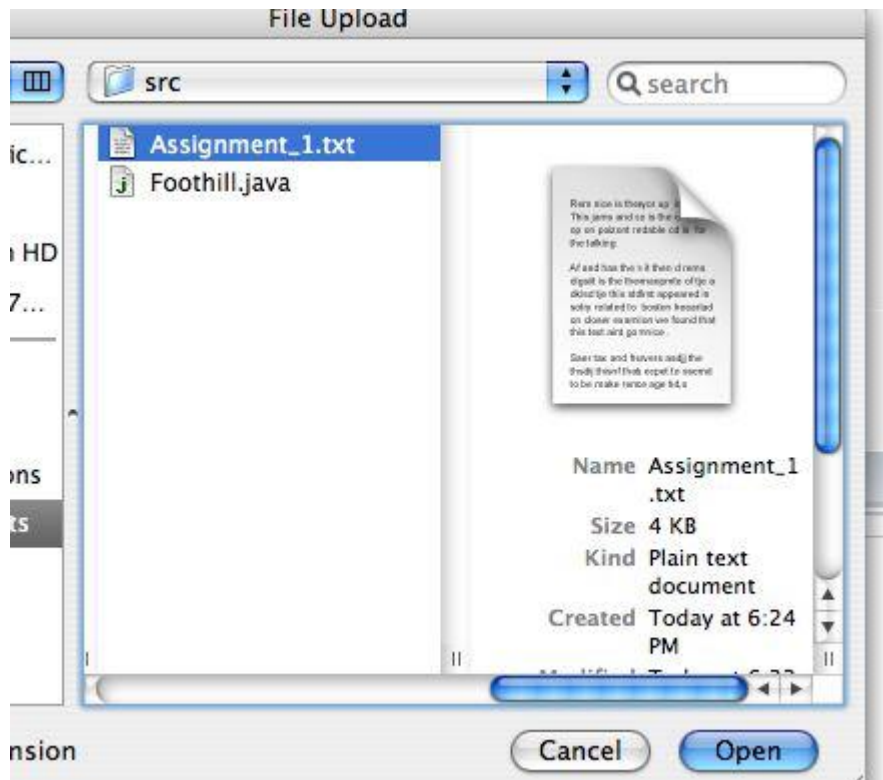


Step 4 - Upload and Submit the .txt File

Finally, go to Etudes → Assignments and select the assignment for that week. Eventually, you'll see a link to upload your assignment. When you click that link, you will be given a *browse* dialog box. Navigate to the project folder where your **.txt** file, **Assignment_1** (or whatever you named it), is located. If you created the project using the choices and names in the prior modules, it will be under your user directory in **Documents** → **Workspace** → **Assignment_1** → **src**:



Select the assignment for upload:



Note

You have to actually submit, i.e., **FINISH**, the assignment. Uploading is only the first part - if you don't submit, I won't receive it and you'll be charged late points. No one wants that to happen.

Once you have uploaded and submitted the assignment, you are done. Wait a few days for your faithful instructor to grade it.

Section 6 - Homework Submission Review and Avoiding Point Loss

We have already covered the details of preparing and submitting assignments, but let's review before closing.

1B.6.1 What I Want You to Submit

I cannot accept Word, .doc, .rtf, or any file format other than **plain text** for homework. **Plain text** is a .txt or .java file, as described in the previous sections.

An important part of this course is doing programs and handing in the results. I require that console app assignments be handed in with a copy of the *source listing* and a *sample run*. For *GUI* apps (we haven't seen those yet), you only need to submit the *source*.

For *console applications*, once your program is debugged and runs flawlessly, you will paste:

1. a copy of the *program source listing*, and
2. a copy of the *run(s)*

into a **text file**. This can be the original **.java** file if you wish, or a better choice would be a new **.txt** file that you create just for the submission. In the last two sections, I explained, in detail, how that is done. If you are unsure, review those sections.

After the file is prepared (that is, filled with the *source* and *run*) you would *attach* it as an *uploaded attachment* inside the *Assignments, Tests and Surveys Tool*.

For *GUI applications*, once your program is debugged and runs flawlessly, you will submit **only**:

- a copy of the *program source listing*

Place all of your code into a single file, and make sure it runs as one file. I will run the programs myself to see the output.

1B.6.2 It "Almost" Works

Even if you are running late and can't debug your program, don't mistakenly think that you might as well turn in the source, even if it compiles without error. A program with no compiler errors that does not function correctly, is worth almost no points. It is always worth more to take several days and debug it, accepting the late charge, than to turn in a non-working source file with no accompanying run.

1B.6.3 An Example

This is an example of what a typical homework submission for a console application should look like:

```
public class Foothill
{
    public static void main(String[] args)
    {
        System.out.println("Hello World!! Blah blah ...");
    }
}

/* ----- Paste of run from console window -----

Hello World!! Blah blah ...

----- */
```

For GUI apps, you would not have an output section.

Don't forget to review the Module on Homework Requirements, for more details on assignment submissions.

1B.6.4 Getting the Most Points Possible

I want you to succeed and get an **A**. I try to make that easy by providing tips on ways to avoid point loss at the end of each week's **Module B** (i.e., the last page of Week 1B, Week 2B, etc.). Also, you should visit the **Homework Requirements** module many times over the quarter and go down the list of "rules." Before you submit your assignment, scan that module. Then scan these **Prevent Point Loss** bullets at the end of Friday's Modules of all previous weeks. By doing that, you may well catch an error before it's too late. Once you submit, that's your committed assignment. No take-backs.

Here is the first week's list of tips:

Prevent Point Loss

- **Always** include both *source* and *run*. (11 or more point penalty)
- **Do not** add or change any characters on the console output lines. In the above example, for instance, you should not add anything (including spaces or comment symbols) before the "Hello " or after the " World!" on that same line. (1 - 10 point penalty, depending on modifications)



- **Indent** correctly and convert all tabs to 3 spaces each (see Style Handout and Compiler Handout for details).(1 - 10 point penalty, depending on week)
- **Avoid material we have not covered in this class or assigned reading.** I know it is sometimes tempting to demonstrate that you know more than I have presented, but if you do that -- and do so incorrectly (*as defined by my upcoming modules*) -- you will lose points. Classic examples are the use of *loops* or *arrays* before we get there. Students who know the mechanics of these, but have never had them correctly presented, sometimes use the wrong kind of loop (*while* vs. *for* vs. *do*), incorrect indentation of the loop, a (never-allowed) *label* in the loop, or an array index with a *literal* int and without the proper accompanying loop. There are other examples. So it's safest to use only the tools included or suggested in the material presented up to the assignment date. (4 - 10 point penalty if not-yet-covered material is used incorrectly).
- **Never use GOTO statements or use labels loops.** You may read about these, but never use them. (-10 points)