**EECS 1720**
**LAB 0 :: Eclipse IDE & Editing and Compiling Java Programs / Basic Syntax**

*Prerequisites* –

1. *Please ensure that you have an EECS account setup. Go to the following link to sign up for an EECS account if you have not already:*
   *https://webapp.eecs.yorku.ca/activ8/*

2. *Please make sure you are familiar with logging into remote lab (you should review the following video on this topic before you get started – PLEASE NOTE, this video refers to the previous version of remote lab which uses a slightly older version of linux (CentOS), however the current version is RockyLinux – very similar but slightly different look: [VIDEO: Using EECS Remote lab ]*

3. *Please make sure you know how to open the following (in remote lab)*
   a. *A Linux Terminal*
   b. *The Eclipse IDE*

*OTHER RESOURCES (Video Links) – to review later (everything already installed in lab)*

- *For your own computers => Installing Eclipse ( MAC / WINDOWS )*
- *Github (PRIVATE) account setup =>Start at Tutorial 01 ( Github setup YorkU )*

**IF YOU ARE UNFAMILIAR WITH EECS LAB COMPUTERS & LINUX**: Please review the section in the appendix of this document entitled "The Linux File System").

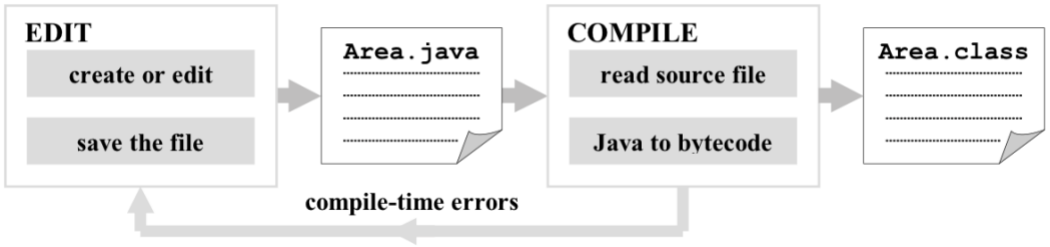
## 1. Editing, Compiling and Running Java Programs (Terminal vs. IDE)

In this part, we are going to compare two methods for creating our Java programs. The first is using a very simple text editor (called *gedit*) and the terminal, while the second uses the *Eclipse Integrated Development Environment* (IDE).

The first tool is all we really need to make simple programs, however there are many advantages to using a full IDE. Most importantly, it helps simplify the process of "building" our programs (i.e. converting our text file program "recipes" into bytecode that the java virtual machine (JVM) can understand.

The JVM ultimately translates the bytecode into a set of instructions that a particular CPU can understand and execute – so on an intel PC, it would translate into instructions a particular intel CPU can understand (likewise for other hardware that differs from PC). Thus, Java is extremely **portable**, as the recipes we define (write) once using the Java language specification can be translated to run on many different computers and devices.

The process of creating a java program begins with editing a Java source file, which is "compiled" into a bytecode version, and then "run", at which point the bytecode is interpreted by the JVM. Java source files have a name that ends with the extension **.java*, while bytecode (compiled) versions of these files have the extension **.class* :




**TASK 1:** Edit, Compile and Run simple Java programs in the terminal

*** there is a primer on these linux commands in the appendix if you are unfamiliar*

i) Open a new terminal window, and use "pwd" to check you are in your home directory.

```
pwd
```

ii) Let's use a new command "mkdir" (make directory) to make a new directory in your home folder called "lab0_ht"

```
mkdir lab0_ht
```

iii) Use "ls" to check the contents of our home directory

```
ls -la
```

iv) Use "cd" to switch into this new directory

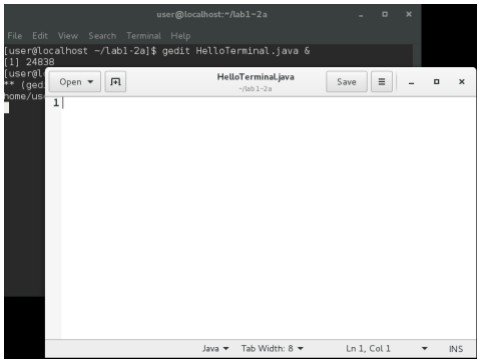
```
cd lab0_ht
```

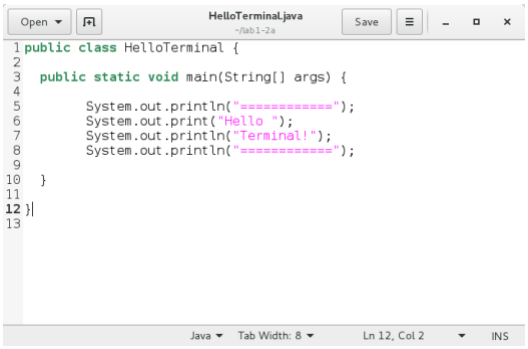
v) We can start the text editor (gedit) from the terminal (or from the Graphical User Interface Menus (on the left/top of the desktop). Let's use the terminal this time:

```
gedit HelloTerminal.java &
```

*(This will create a new file in the current directory called "HelloTerminal.java" and will open it using the application called "gedit")*

*gedit* is short for "graphical editor". It is one of many text editors that could be used in Linux (and is similar to *Notepad* in windows, or *TextEdit* in mac). It is very easy to use, and opens with a blank window (a file with no text). The ampersand symbol ("&") in the above command just tells Linux to execute gedit in the background, so we are still able to type commands into the terminal while gedit is running (see figure below-left).




vi) Type the following text into the **gedit** window (figure above-right):

```
public class HelloTerminal {
      public static void main (String[] args) {

            System.out.println("=========");
            System.out.print("Hello ");
            System.out.println("Terminal!!");
            System.out.println("=========");
      }
}
```

vii) Now save the file (click the Save button or hit CTRL+S) and keep *gedit* open. Return to the terminal window, and let's list the contents of the current directory and *cat* the contents of the file we just edited

```
ls -la
cat HelloTerminal.java > output_HTgedit
```

*(You should see the HelloTerminal.java file now, and its output should reflect the source code you just typed into the editor – if you cannot see anything in the output_HTgedit file, then you need to make sure you save properly in the gedit editor – if there is an asterisk showing, then this means the file still needs saving)*

Now save the file (click the Save button or hit CTRL+S) and keep *gedit* open. Return to the terminal window, list the contents of the current directory and *cat* the contents of the file you just edited into a new text file, and submit it:

```
ls -la
cat HelloTerminal.java > output_HTgedit
submit 1710 lab0 output_HTgedit
```

*(You should see the HelloTerminal.java file now, and its output should reflect the content of the source file)*

viii)

The command *javac* is used to compile java source code (which converts the text source file into bytecode for the JVM). Type:

```
ls -la
javac HelloTerminal.java
ls -la > output_HTdirAfterCompile
submit 1710 lab0 output_HTdirAfterCompile

ls -la
```

*(You should now see a file called HelloTerminal.class)*

Run the java program in the terminal window (Type) and observe the output:

```
java HelloTerminal
```

You should see the following in the Eclipse Console:

```
=========
Hello
Terminal!!
=========
```

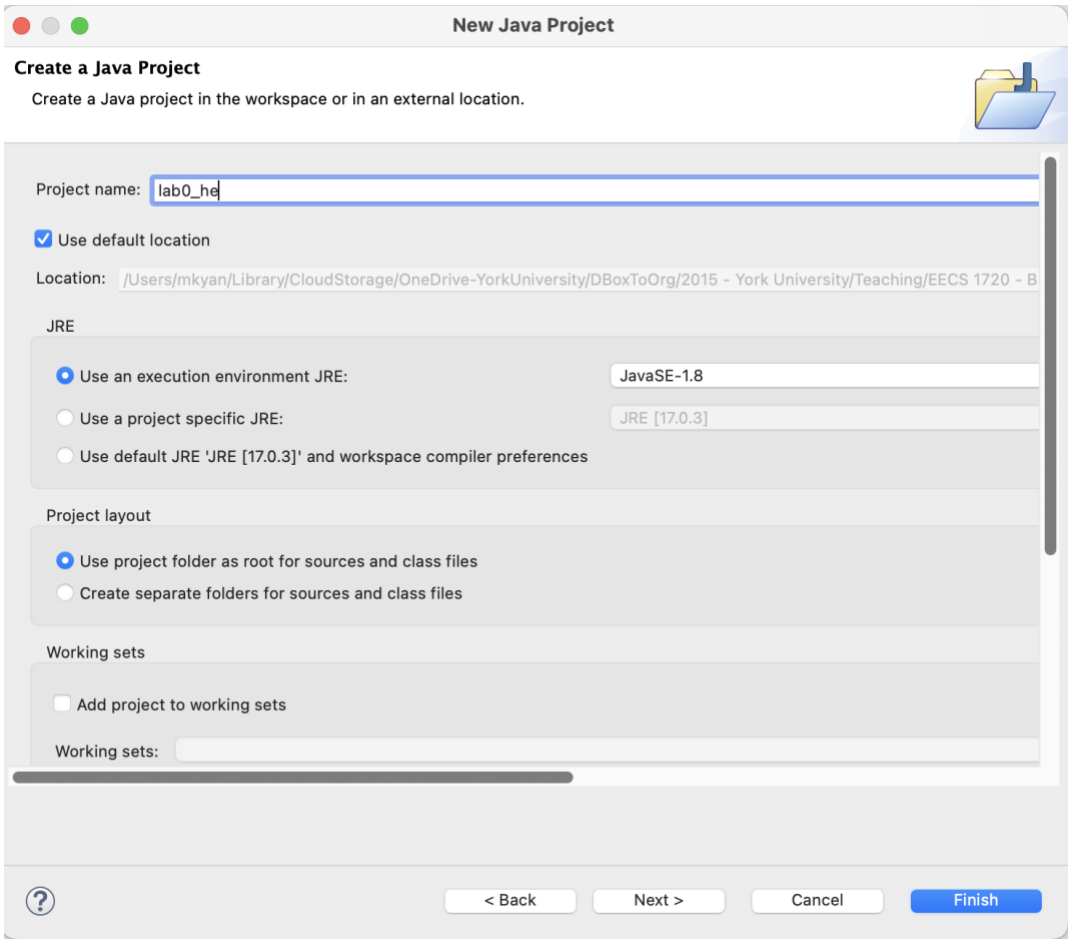
**TASK 2:** Edit, Compile and Run simple Java programs in Eclipse

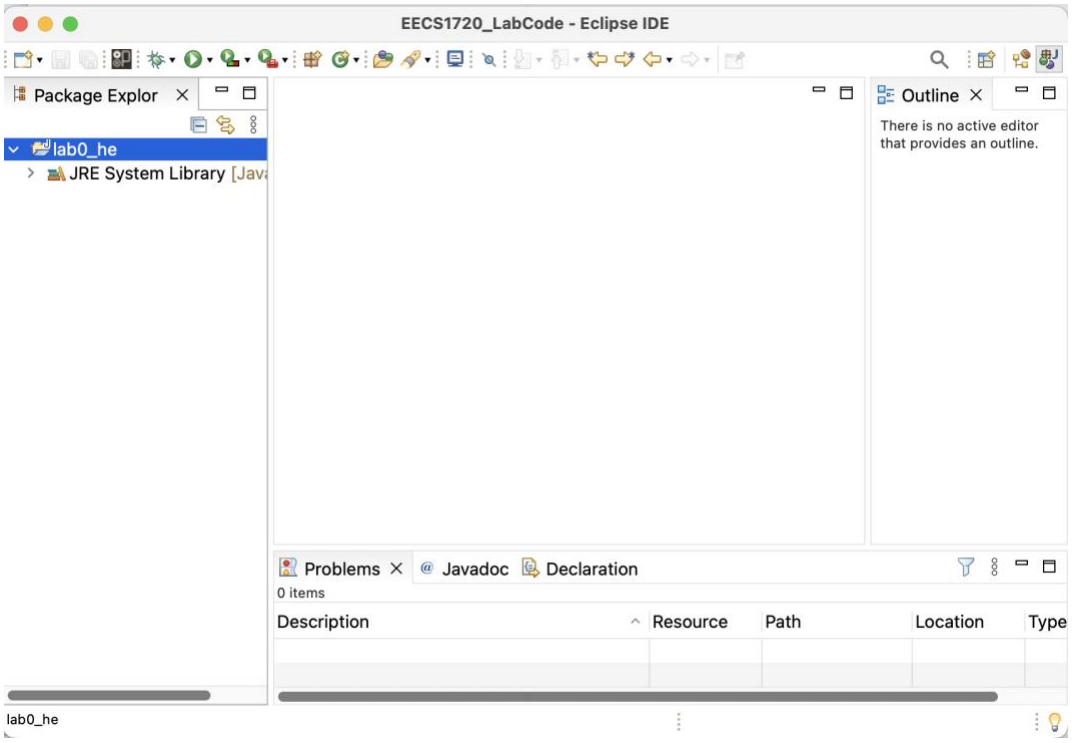
i) Launch the Eclipse IDE (as described in the 2nd prerequisite for this lab). Keep the workspace as "eclipse-workspace". Open a new Java project called "lab0_he". You can do this from the File→New→Project. Select the Java option, and Java Project.

You will see a wizard similar to the following. Use the settings shown below for your project. When you click "Finish" it will ask you if you want to use the Java perspective (say yes). The java perspective sets up the environment visually with tools you need for java programming.
Settings:

- execution environment JRE = JavaSE-1.8
- use project folder as root for sources and class files

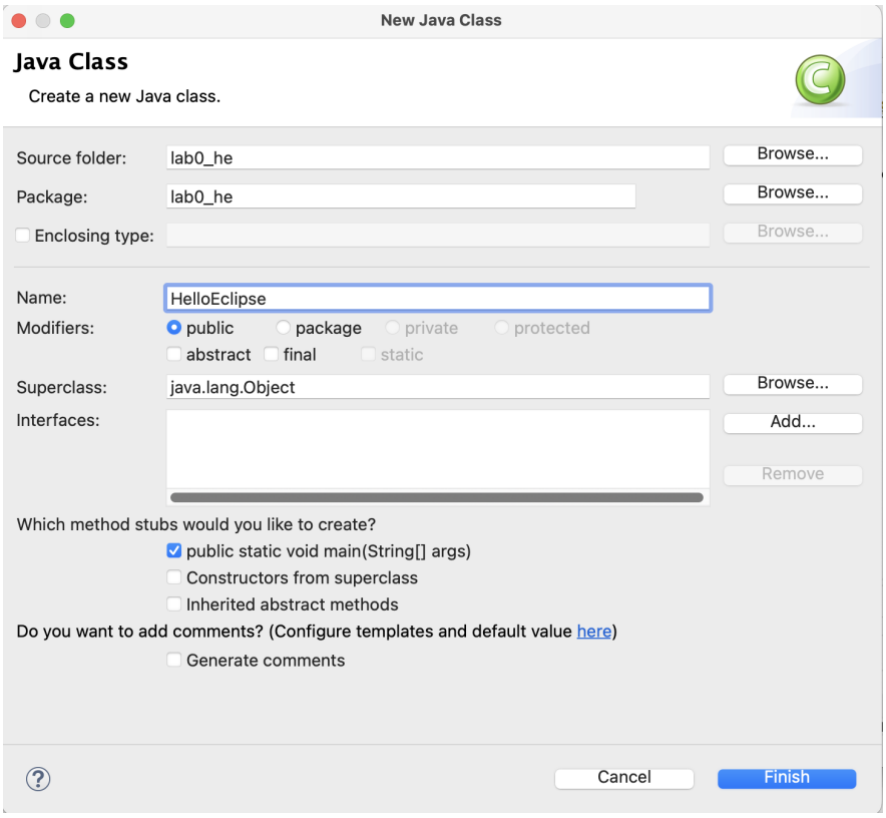
*** we will learn about separating source and class folders later*

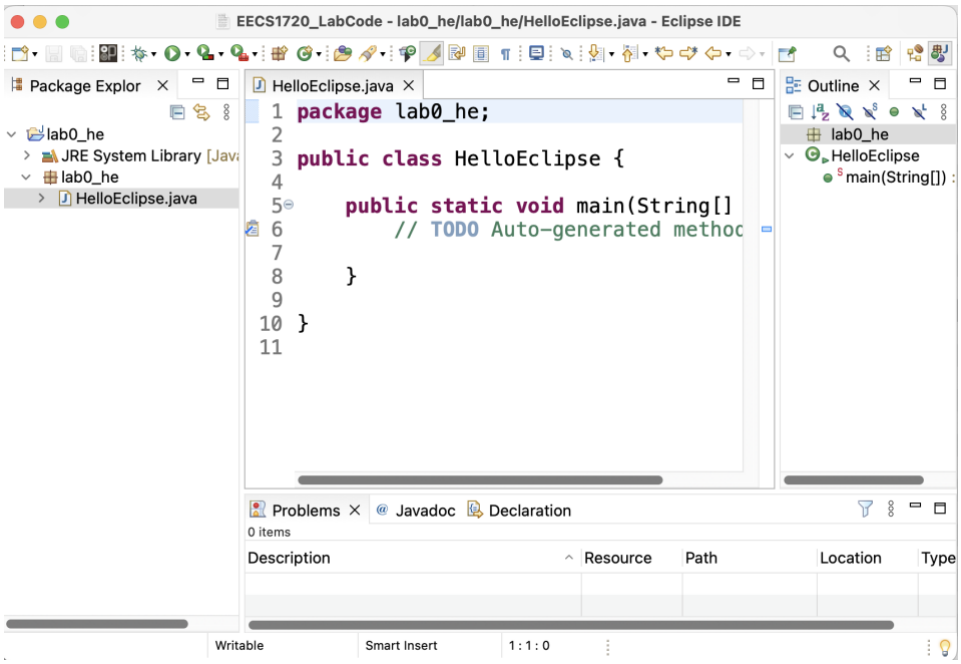

New Java Project
Create a Java Project
Create a Java project in the workspace or in an external location.
Project name: lab0_he
Use default location
Location: /Users/mkyan/Library/CloudStorage/OneDrive-YorkUniversity/DBoxToOrg/2015 - York University/Teaching/EECS 1720 - B
JRE
Use an execution environment JRE:
JavaSE-1.8
Use a project specific JRE:
JRE [17.0.3]
Use default JRE 'JRE [17.0.3]' and workspace compiler preferences
Project layout
Use project folder as root for sources and class files
Create separate folders for sources and class files
Working sets
Add project to working sets
Working sets:
< Back
Next >
Cancel
Finish


EECS1720_LabCode - Eclipse IDE
Package Explor
lab0_he
JRE System Library [Java
Outline
There is no active editor that provides an outline.
Problems
Javadoc
Declaration
0 items
Description
Resource
Path
Location
Type
lab0_he

ii) Now create a new "class" called "HelloEclipse". Do this by selecting the HelloEclipse project in the left margin, and navigating to File→New→Class.

The class wizard will open, looking something like this. Use the settings shown and click "Finish" (the new java source file will open in the editor – see below)



iii) Copy the text in the main method over from the *gedit* window (or re-type the text in this document) to the Eclipse editor window, paste it and save the project. Then edit the **`main`** method so that the program outputs "Hello Eclipse" instead of "Hello Terminal".

iv) Now, before we build and run the project from the IDE, let's find where all the source files generated and edited from within Eclipse are located. Open another terminal and cd to the home directory, then cd to "eclipse-workspace". **`cd`** again to the project directory (you should have called it "lab0_he").

Use **`ls`** command to check that the directory exists in the eclipse workspace. The location of the files generated by the IDE will be different to the location of the HelloTerminal program created in the previous step.

v) List the contents of the *lab0_he* directory, there should be only one java file (or there may be a .class file if the *Project→"Build Automatically"* option in the Eclipse main menu is checked).

Uncheck this option and select *Project → Clean* (this should remove any .class files). Use "ls -la" to list all the contents. Notice that this folder has some other files too.. these are project settings files used by the IDE.

vi) Build and Run the project in the IDE by clicking on the green "play" button in the Eclipse window.

Notice the output does not show in a terminal, but rather, shows in the console window within Eclipse! When using the IDE we typically edit, build and run programs entirely within the IDE. Details of compiling and where these components actually reside on the computer are hidden. But this does not mean we cannot access them.

You will see that the terminal allows us to access the java **source code** and JVM **byte code** for this project.

vii) We could also potentially run the *HelloEclipse* program manually from within the terminal: do this by typing the following into the terminal:

```
java HelloEclipse
```

viii)

Submit your source file for the HelloEclipse (from the terminal):

```
submit 1720 lab0 HelloEclipse.java
```

## TASK 4: Importing an Archived Project and Completing Tasks within

In this task, you will learn how to import an archived project into your Eclipse workspace, where you may then continue with the lab (exercises are embedded within the project with instructions commented in the *.java files).

TO COMPLETE THIS TASK – YOU DO NOT HAVE TO USE REMOTE LAB (you can do this on your own machine, running eclipse. The video links for installing eclipse natively are available through the eClass course shell. If you want to continue using remote lab, that also works.

1a. First, download the **lab0** project file from the following link:

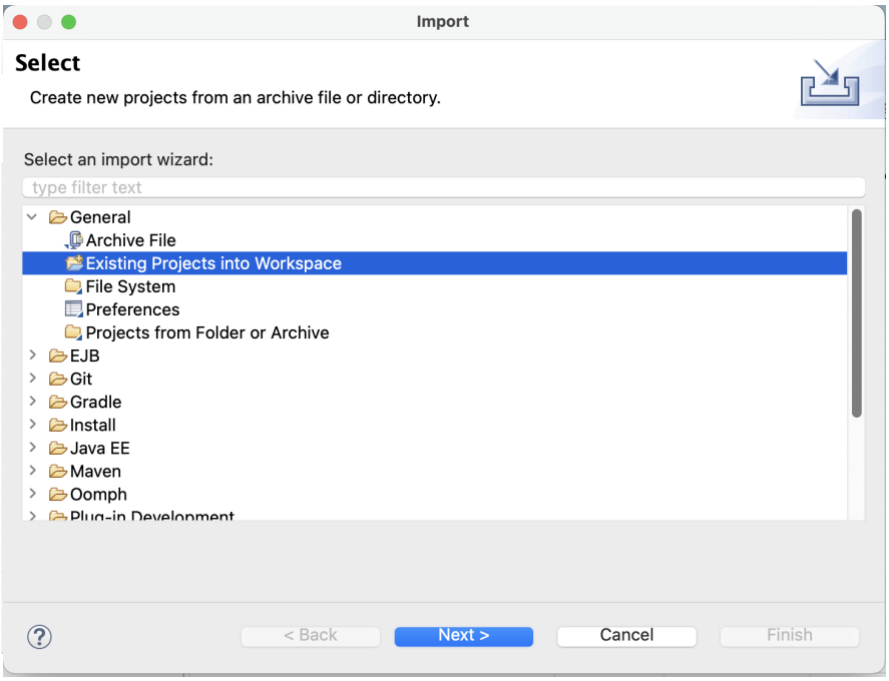
This file is a *zip file*, also known as an *archive file*. Click (or double-click) on the file to download it. In the dialog, choose **Save**, not **Open**.

https://www.eecs.yorku.ca/course_archive/2022-23/W/1720/labs/lab0/lab0.zip

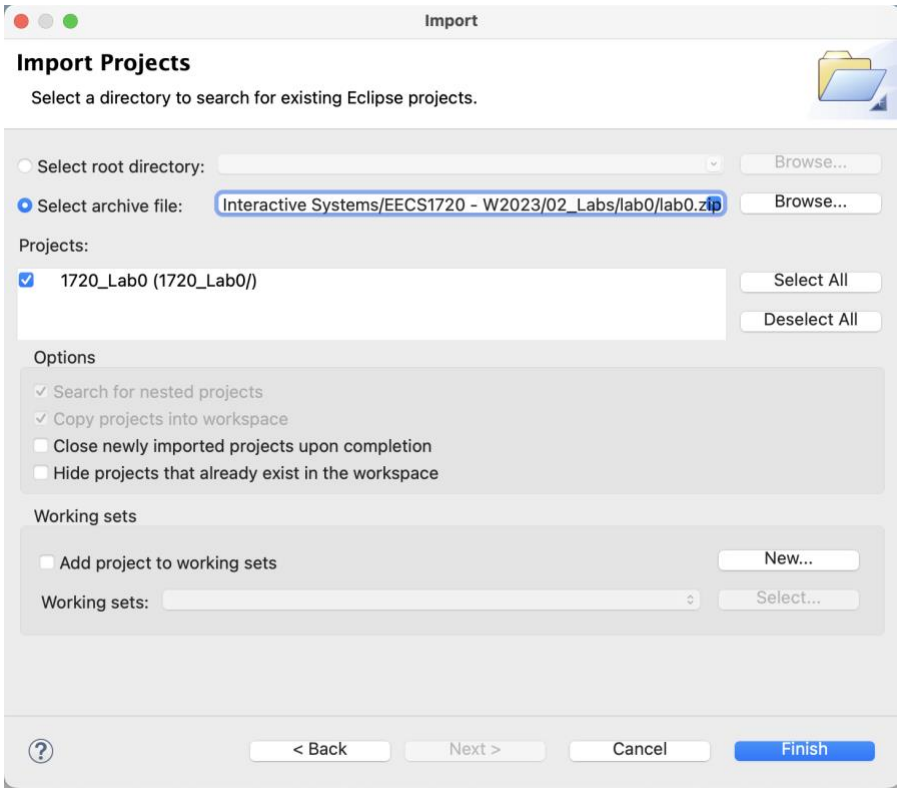
** save this to your downloads folder, and we will import into Eclipse from there.

1b. You might want to create a new workspace first, for all your future labs
this step is not necessary for this lab, but will help for future labs if it is already setup.

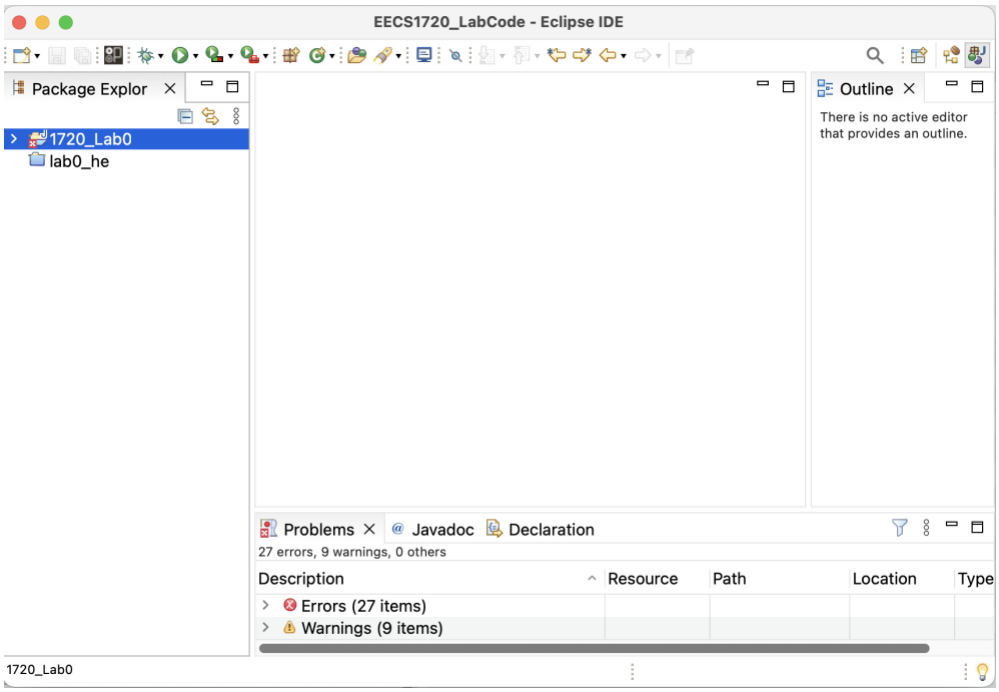
1c. Close 'welcome', and navigate to **File->Import**, you should see the following dialogue window. Select **General->Existing Projects into Workspace** & hit Next.



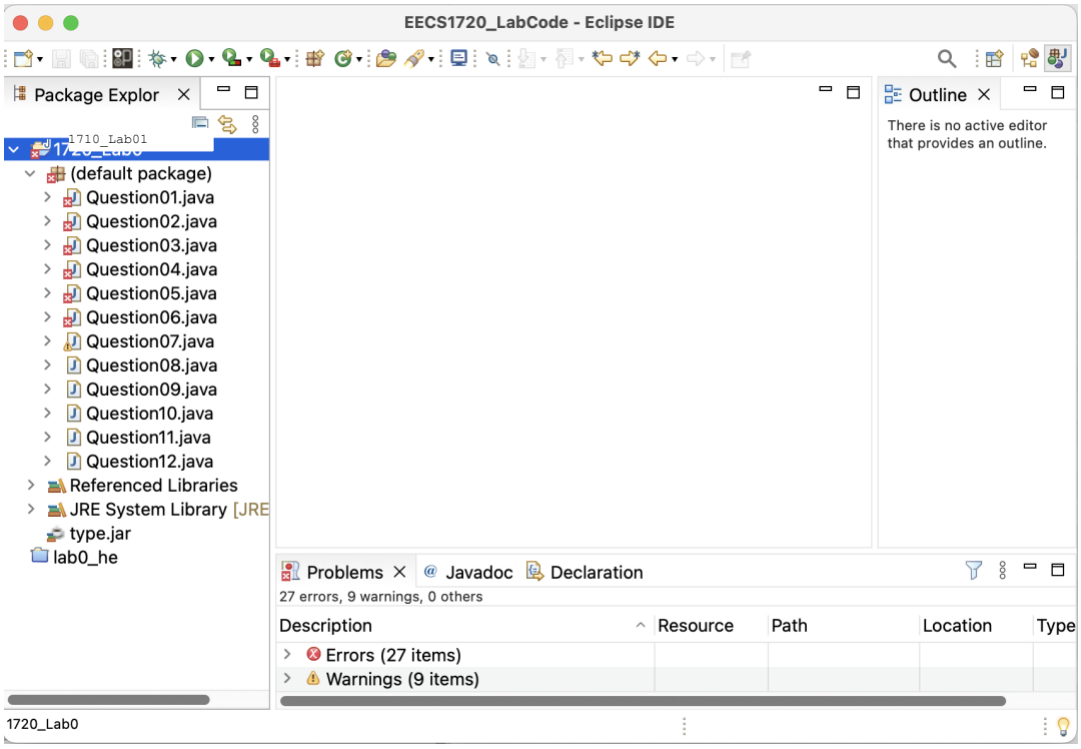
Choose Select Archive File (see below), and Browse to where you downloaded the **lab0.zip** file.
**IT IS IMPERATIVE THAT YOU OPEN PROJECT ARCHIVES THIS WAY**
This process will be the same for all labs and lab tests!!




After selecting the file, hit **Finish** and the file will open up as a project in your project explorer (left hand margin).




Open (expand) the project in the Project Explorer (files are located in the “default package” item in the project tree in the left margin).



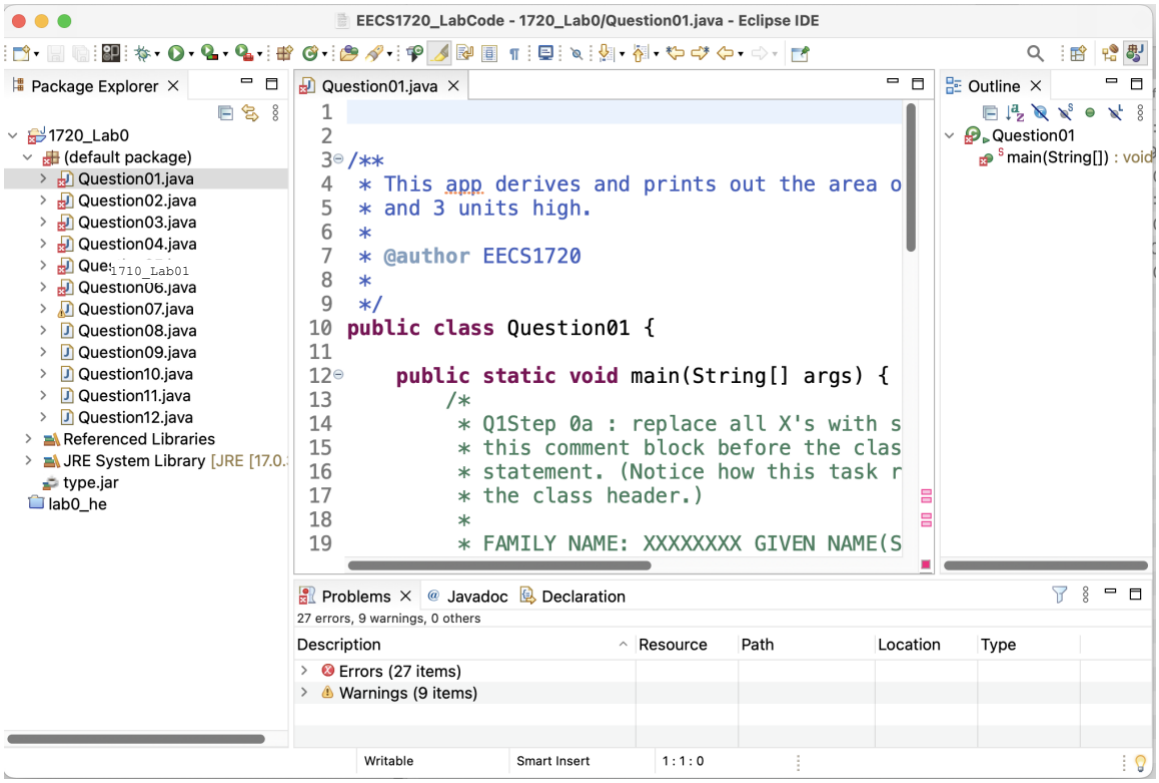
Let us review the contents and discuss them first. The contents are:

- (default package) – this direcory contains several Java class definitions. The red "X" indicates that **at least one java file did not compile correctly**.
- doc, doc.resources – these directories contains the Application Programming Interface (API) for the Java classes within this project. These are in html format and can be viewed from within Eclipse (right-click on an html file and select **Open With... > Web Browser**
- Referenced Libraries – this directory identifies libraries (jar files) used in this project
- JRE System Library – this is the directory that contains the files that constitute the Java Runtime Environment.
- type.jar – "type" stands for The York Programming Environment. This file is an archive of the java classes that are used in the textbook examples. Some of the lab exercises use these classes.

Once you have imported the project archive and have familiarized yourself with its components, then you can move to the next step.

Expanding the default package, you will see 12 *.java files (you can ignore the *.class files), the *.java files are source code that needs to be repaired/completed.. the questions serve as a quick practice with basic syntax. Note: to print in java, the print() or println() methods are accessed using “System.out.print()” or “System.out.println()” => more on this in lectures.

Start with Question01.java, open by double clicking this file in the project browser (left hand margin), this will open the file in the editor. When you modify the file, you will see that is NOT SAVED with an * on the file name. CTRL+S (linux/windows) will save (or CMD+S in mac).




Follow the instructions in each file to complete, fix, and run the file. Each file has its own "main" method, and all runnable statements must be within the main method (the exception is variable declarations – more on this later… but you can assume everything can be done inside the main method). For this lab you don't need to do all of these exercises, but it is good revision. Practice some of these, and make sure you attempt the submit.

*** The main idea of exercises 1-6 is to take corrective action (fix syntax etc) in order that the code can run. For each individual question file, to execute, you must fix the code to the point that there are no more red 'x"s … the green play button will run the code, and the results should be outputted in the console pane (at the bottom). You can confirm if the result is expected or not.*

Now attempt the remaining questions:

**Questions 7-8** concern the *char* datatype

**Questions 9-12** are based on exercises from the textbook (the TA's will provide more explanation on answers)

## SUBMISSION: NO DUE DATE

Practice submitting the java files associated with questions 8-12 in this task. To do so, use the submit tool from terminal (outlined in the appendix below), or use the web submit tool: *https://webapp.eecs.yorku.ca/submit/*

Remember, you are submitting to lab0. You should also see the other files you have already submitted in earlier tasks. You have a week to complete these tasks, so do not worry about getting everything completed in the lab session itself.

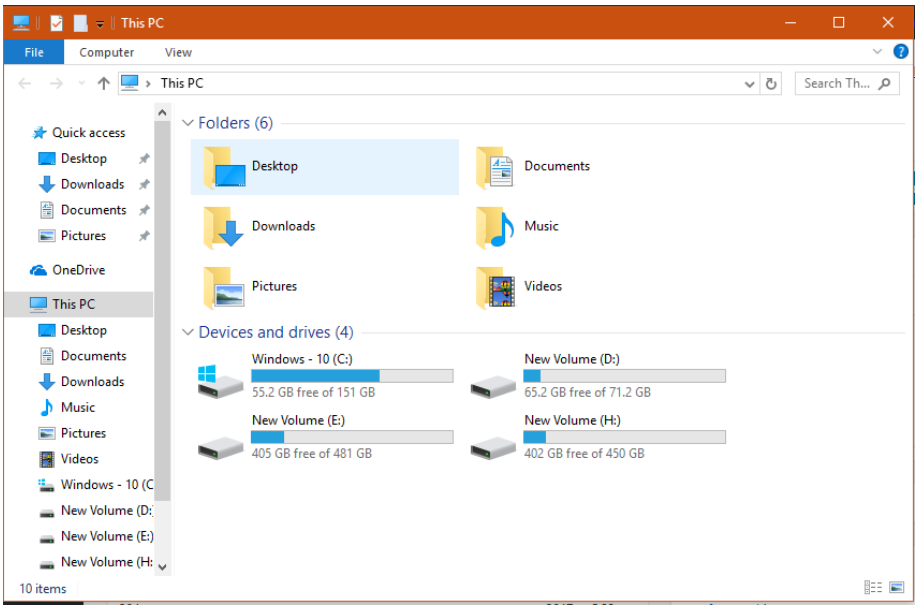
---

## APPENDIX

***review of getting around the linux file system & using SUBMIT / WEB SUBMIT***

### 1) The Linux File System

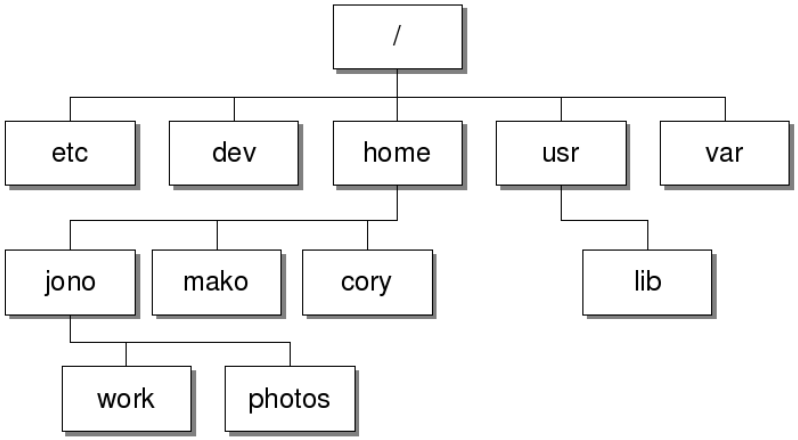
If you are familiar with Windows, you might realize that each storage device (e.g. hard drive) on the system is organized in a hierarchical way. For example, if you have one hard drive on your computer, the drive would be referred to as a letter (e.g. C:). Files are then organized within that particular drive into folders according to a specific hierarchy (folders within folders). The top of this hierarchy (parent folder) is typically "C:\", which then has sub-folders (e.g. "C:\Windows", or "C:\Program Files", etc.). If you have other storage devices, they have a different letter (e.g. D: or E:), and each has its own hierarchy of folders, and so on. If you double click on one of the "Devices and drives" shown in the figure below, you would be able to see the contents of that device/drive (its files and sub folders).




In Linux, similar is true, however the organization of all files on the system is extremely simple. Basically, all files are organized into a **single** hierarchy (a folder is referred to as a directory), beginning with the main (or parent) directory called "/", which is known as the *root* directory.

The command "cd" is used to do this (cd stands for "change directory"). The command "ls" is used to list the contents of a directory (which may be files and/or other sub-directories).

While the "pwd" command ("pwd" stands for "print working directory"), was used to tell you where you were currently located within the larger file system.
Here is a typical diagram of how files/directories are organized in Linux:




Of course, there are more folders typically than this. However, we are mostly concerned with the *home* directory (`/home`), and its sub directories, which is where you will be creating, storing and running most of your programs.

The home directory houses one directory for each user on the system (when you log into Linux, you are auto-magically connected to your own directory). For instance, in the above graphic, `/home/mako` is the directory belonging to the user "mako", while `/home/jono` is the directory belonging to the user "jono". If jono logs into the system, usually they will create files/sub-directories (such as work or photos) in their personal home directory `/home/jono`. If you open a terminal and type "cd" you will always be placed directly into *your* home directory (since the system knows who you are after login).

## Navigating the file system, recording command output, and submitting work.

In this first set of tasks, we are going to learn and use a set of basic Linux commands for moving around the file system, displaying contents of directories and files, and creating new files. First, here is a quick summary of some standard linux commands you will typically use:

Summary

| Command | Meaning |
|---|---|
| pwd | Print the full (absolute) pathname of the current working directory. |
| cd *dirname* | Change to the named directory. |
| cd .. | Change to the parent directory of the current working directory. |
| cd | Change to the user's home directory. |
| ls | List the contents of the current working directory. |
| ls *dirname* | List the contents of the named directory. |
| ls -l | List using long format the contents of the current working directory. |
| ls -l *dirname* | List using long format the contents of the named directory. |
| ls -ld *dirname* | List using long format the name (but not the contents) of the named directory. |

i) Firstly, lets open a terminal and navigate (using the "cd" command) to the root folder on your system, and list its contents. Type:

```
cd /
```

*(This points the terminal to the system's root directory)*

Type

```
pwd
```

*(This shows you where you are w.r.t. the root folder "/")*

Type

```
ls -la
```

*(this lists the files and their details in the current directory. Observe the output in the terminal)*

** see if you can confirm this using a file manager "Files" application (a graphical way to access the same file system) – this can be launched by selecting *Places* → *Computer* from the Places menu on your VM, or by searching for *Files* in Activities.

ii) Now we will learn two more Linux commands: "concatenate" (`cat`) and "redirect" (>) so that we can dump some output from the "list" (`ls`) command into a text file.

Type

```
cd
```

*(This returns the terminal to your home directory)*

Type

```
ls -la > output_homeDir
```

```
```

*(this lists files in current directory and redirects (dumps) the resulting output into a new file called "output_homeDir" instead of showing it on the screen)*

Type

```
ls -la
```

*(this lists the files in the current directory again, notice the new file that was just created)*

Type

```
cat output_homeDir
```

*(this shows the contents of the file "output_homeDir" in the terminal window)*

iii) Type

```
ls -la / > output_rootDir
```

*(this lists the contents of the root directory, and redirects the output into a new file called "output_rootDir")*

iv) Type

```
cat output_rootDir
```

*(this outputs the contents of the file output_rootDir to the terminal window)*

v) Finally, let's learn a special command that allows you to **submit** components of your lab work (for grading/assessment) from the terminal inside the remote lab portal. Note this will only work if/when you do your lab work on the remote lab machine. If you do your lab work (for future labs) from your machine, then you will submit your work using a special web submit tool (outlined later).

Submitting work from the **terminal window** (via remote lab):

*Essentially, lets say we want to submit all files that start with "output_" in the current directory to the "lab0" assignment defined under the course "1720":*

Type

```
submit 1720 lab0 output_*
```

*(this will upload/send all the files that start with the word "output_" for submission for lab1. Actually, we plan to submit a number of different files in this lab. We can do each file separately, or in groups, or we can even submit all the files in a given directory. However, we will usually be very specific about what you will need to*

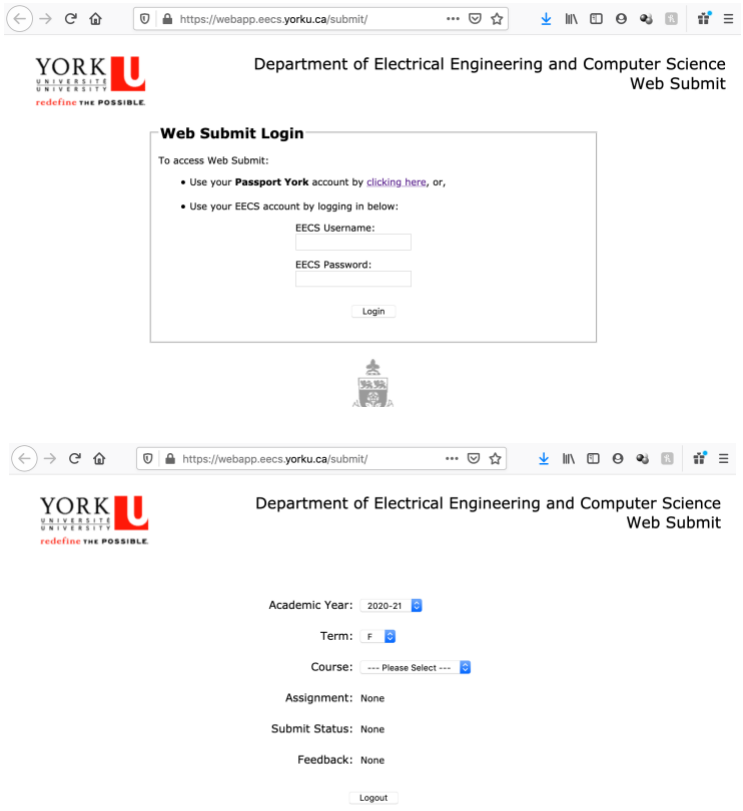
*type to submit at various places in each lab (or in a single location at the end of the lab document).*

You should see some output indicating the files that were submitted. You can always re-submit these files up until the lab deadline. Any files with the same name submitted multiple times will just overwrite older versions of those files.

Alternative way to submit – using **web submit** (submitting files from home):

*The other way to submit files for a lab (which you need when submitting from home), is called web-submit. To submit files this way, you need a browser pointed at the following URL:*

*https://webapp.eecs.yorku.ca/submit/*




If you use your EECS account name and password to login, you will see a screen like that above (right). Choose *W* from the Term dropdown menu and *1720* from the Course dropdown menu, and you will see a list of assignments that are open for submission (only listed if submission is available).

In the Assignment dropdown menu you will see *lab0*. Then you must individually *Browse* for each file you want to submit (see the figure below-left, note that the academic year will

be **2022-23** not 2018-19 as in the figure below). When you have chosen the files to submit, hit the *Submit Files* button at the bottom of the page. The web page will then indicate the files that have been successfully submitted (shown in the figure below-right).

*** after labs are submitted (and submission folders are closed), you can also see the files you submitted through this web URL*