Eclipse and Java

The directory structure we build with Eclipse

Python is directly interpreted: when you run a Python program in a Python interpreter it walks through the program structure executing the Python instructions one at a time. This is simple to understand, but the programs execute relatively slowly. That is not a problem for the kind of work Python is good for, but if you are trying to process a data file with millions of entries, Python will not do it very quickly. Java takes a different approach. We write code in files that have the ".java" file extension. Each Java file corresponds to a single class. To run a Java program the .java files need to be *compiled* with a standard program *javac*, that is called the Java Compiler. The compiler takes a .java file and produces "Java byte code" that is stored in a file with extension ".class". The .class files can then be executed by a program called *java*: the Java Virtual Machine. Java byte code is much closer to the machine language that is native to your computer, and so the .class files can be executed much faster than an interpreted language like Python. Here are the steps to creating a Java program:

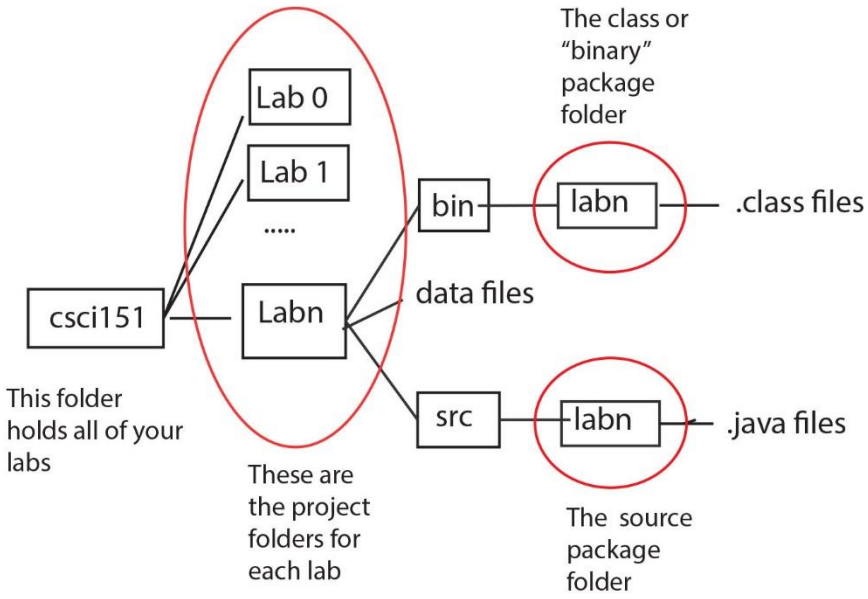
A. Write Java code for the program into one or more .java files
B. Compile the .java files into .class files
C. Execute the .class files with the Java Virtual Machine.

Eclipse handles most of this for us. It provides an editor to help with writing Java files, it automatically compiles the .java files into .class files, and it executes the .class files when we ask it to. You can do all of the labs this semester entirely within Eclipse, but it does help to have some idea of what is going on in the background.

Before we get into the file structure there is one other concept we need to discuss – the idea of a "package". Most Python programs are relatively small, occupying one or two files. Java was designed from the start as a language to handle large, complex programs that are spread over many files. If you had one folder containing 100 java files it would be hard to find anything. To add some structure to large programs Java uses *packages*. A package is a group of related classes that are all contained in one folder, called the package folder. For example, a video game might have a lighting package with a number of classes that control the way light reflects off surfaces; environment package that describes the world in which the game is taking place; and one or more character packages that control actors in the game. Java does not require the use of packages, but it strongly recommends them and the development environments like Eclipse are set up to use them. In the past I have recommended that students NOT use packages for the 151 labs but some students did anyway because Eclipse pushes them, and the fact that some students used packages and some didn't made life difficult for the graders and lab helpers. This semester I am

recommending that everyone use packages and all of the code I will give you for the labs will be set up with packages.

Here is the file structure Eclipse will build for us for lab number "n":



We will use the convention that our project folders will be Lab0, Lab1, and so forth. Although it is possible for a project to use multiple packages, our labs this semester all call for one package for each lab. The package for Lab1 will be lab1, and so forth. Note that there are two package folders. The one inside src contains all of the .java files; the one inside bin contains all of the .class files. With all of your file and folder names it is wise to avoid putting blank spaces in the names.

The Eclipse workspace

Eclipse needs a location to put information about your projects and temporary storage that is needed when it compiles you program. This is called the *workspace*. The first time you start up Eclipse it will create a workspace in a standard location that depends on the type of computer you have. My workspace (in Windows) is at C:\users\bob\eclipse-workspace. Yours will probably have a similar name and be in your home directory. Each time you start up Eclipse it will offer to change your workspace. Don't do that; just create one workspace at the start (wherever Eclipse offers to build it is fine) and use it for all of your work. You will probably never need to look at it, so it doesn't matter if you don't remember where it is. In fact, the newer versions of Eclipse have a checkbox on the opening page to tell it not to offer to change the workspace. It is fine to check this.

When you create a project in Eclipse it will offer to put it at the "default location", which is inside the workspace. You will find it easier to keep track of your files if you do *not* use this default location. In Lab0 we make a folder csci151. I suggest that you put each of your labs in that folder.

Making a new project with no initial code

The most reliable way to create a project in Eclipse and have it be where you expect is to first make a project folder, and then go into Eclipse and say to use that folder. Suppose you are starting Lab 2, which has no starter code so you are starting it from scratch. First make a new (empty) folder Lab2 in your csci151 directory. Then start up Eclipse. Don't change the workspace. Create a new Java project and name it "Lab2". Eclipse will offer to put it in the "default location", which is inside your workspace. Instead of this, uncheck the box for default location and use the **Browse…** control to the right of the textbox to navigate to your csci151/Lab2 folder. Once the Lab2 folder is highlighted you can click the Finish button at the bottom of the window. That makes the project. The first time you make a class to go in this project Eclipse will ask for its package name. For this lab the package is "lab2" (capitalize the project folder; use lowercase for the package folder). Eclipse will create the package folders when it creates the class.

Making a new project with starter code

Any code I give you will be set up the way your labs are configured. Just put the project folder in your csci151 folder and open a project with its name. If you have code for some other source you may need to modify it to work with our configurations. You might find it easiest to create a new empty project, then copy the .java files into the src package folder. If the code was not set up to use packages you will need to edit it and add a package declaration such as

```
package lab2;
```

to the top of each file – prior to any class declaration.

Eclipse windows

Eclipse has many windows, but fortunately you only need a few of them for your work this semester. There is a *Window* menu at the top of the Eclipse application. This has a **Show View** item that lists all of the available windows and allows you to navigate around them. Your main starting point is the **Project Explorer**. This lists everything in each of your projects. If you walk down from the project folder to the src and package folders you will see all of your code files. Clicking on any of them brings up your code files. There is a similar

Package Explorer which shows only the source code packages; I find the Project Explorer more useful. When you run a program the output goes to a window that Eclipse calls **Console**. Eclipse likes to subdivide windows into smaller and smaller pieces that you may be able to read, but I can't. If the console is too small for your needs, double-clicking on it will enlarge it to half or all of the Eclipse application window.

You will be able to do most of your work this semester just using the Project Explorer and the Console.

Adding code files to an Eclipse project

You might not need it this semester, but at some point you might want to add an existing file to o project you are already working on in Eclipse. You can't just add the file to the source package; Eclipse won't recognize it if it didn't put it there. There are two ways to accomplish this. One is to use Eclipse to create a new class with the name of the file you want to add, then cut and paste the existing code into your new class. Here is a more elegant approach. In the Project Explorer highlight the folder for the project that you want to add the file to. Go to the Edit menu and select **Delete**. The dialog that pops up has a small checkbox that says "Delete project contents on disk (cannot be undone)". **Do not check this box!** If you do it will delete your code. If that box is unchecked and you click the OK button, Java will eliminate the project structure while retaining all of your files and folders. Add the new file to the src/package folder along with the other .java files. Then make a new project with the name and location of the project you deleted. Eclipse will recognize the new file along with the rest of your code.

Configurations and Program Arguments

All of your projects, including many different individual programs, are in Eclipse at the same time. You can't just say "run my program"; you need to say what to run and how to run it. That is the role of a **Run Configuration** in Eclipse. A configuration specifies:

a) Which project you want to run.
b) The main class (i.e., the class with the main( ) method) that you want to run.
c) Any arguments that you want to give it.

The Run Configuration tab within the Run menu allows you to change these items. You will usually use this to change the unnamed "default" configuration. If you are in a situation where you want to run one program repeatedly with the same arguments, you can supply a name for the configuration so it will stay around while you make changes to the default configuration.

Remember that a main( ) method in Java has signature

```
public static void main( String [ ] args )
```

The arguments you supply to the Run Configuration are packed together as an array of Strings and given to the parameter *args*. The arguments are delimited by white space, so you can't have any blank spaces inside a single argument. Numeric data needs to be converted inside your program to have the write type. For example, if you supply just one argument as 23, args[0] will be the String "23". You probably want the integer 23, which you can get as Integer.parseInt(args[0]). There is a similar Float.parseFloat( ) method, but we won't need it this semester.

Where do data files go?

We will make a lot of use of data files. Rather than hard-coding the names of files into our programs, we will typically give them as arguments through a Run Configuration. But that is just the name of the file; the file itself needs to be in the project folder. For example, in Lab 2 you will write a program that reads the file "CatInTheHat.txt". Your code for this program will have a line that creates

```
new Scanner( new File("CatInTheHat.txt") )
```

Your Lab2 project file will contain 3 items: the bin folder, the src folder, and file CatInTheHat.txt. In Lab 3 you will write a program that solves mazes. I give you a folder called "mazes" with entries maze-1, maze-2, etc. This mazes folder is placed in your Lab3 project folder. Within your code for this lab you might open

```
new File( "mazes/maze-2")
```

Just remember that when Java looks for a file your code has referenced it will start in the project folder.

What to do if Eclipse gets stuck

Starting with Lab 2 we will make use of a testing framework called **JUnit** that integrates nicely with Eclipse. Junit allows us to create tests for code without having a complete working program. With older versions of Eclipse we sometimes had a problem with Eclipse getting "stuck" in the testing environment and being unable (or unwilling if you believe in personifying computers) to run normally. The current version of Eclipse seems better with that. The Run Configurations window has a sidebar that allows you to specify whether you want to run a Junit test or a Java application. Once you run a test it will continue running tests until you specify that you want to run an application. If you do not currently have a

Run Configuration for running the application, this window gives you an opportunity to make a new Configuration. Unfortunately, the "New Configuration" button is a little hard to see, at least for my eyes – it is the leftmost button at the top of the Run Configurations window.

Eclipse is a complex application. Every semester we have a few students who manage to get Eclipses tied into a knot where it is completely non-functional. If that ever happens to you, here is something that often helps. Go to the Project Explorer and highlight the folder for the confused project. Go to the Edit menu and select **Delete**. There is a small red X next to Delete. The dialog that pops up has a small checkbox that says "Delete project contents on disk (cannot be undone)". **Do not check this box!** If you do it will delete your code. If that box is unchecked and you click the OK button, Java will eliminate its project structure and (hopefully) whatever was causing Eclipse to misbehave, but it will leave all of your code. It is an easy matter to recreate the project. In most cases this fixes the problems with Eclipse.

Packages in Java

You will not need to do this in the current semester, but at some point you may need to work with Java code outside of Eclipse. If you are using packages, just follow the directory structure we are using inside Eclipse: the .java files should be inside a package folder, and at the top of each file there should be a package declaration: the word "package" followed by the name of the package folder, such as

```
package lab2;
```

Compiling Java programs manually

If you have a simple Java program consisting of one class, perhaps with name foo.java, and no packages it is simple to compile and run. You need to open a terminal and navigate to the folder containing the file foo.java The command line

```
javac foo.java
```

compiles the Java code into the file foo.class.

Then the command

```
java foo
```

(note: *not* java foo.class) executes program foo. If foo expects two arguments, perhaps the string "bob" and the number 68, you would execute it with the command line

```
java foo bob 68
```

Now suppose you have a more complex situation. Let's assume we have a project folder PROJ. Inside PROJ are src and bin folders, each with a package subfolder pack. Inside PROJ/src/pack are two Java files: A.java and B.java, with A being the main class. We need to compile and run all of this manually.

The solution is a bit verbose, but it is not particularly hard. We open up a terminal and navigate to the project folder PROJ. Since A is the main class, it must use elements of B (or there would be no reason for B to exist), so we compile B first with the command

```
javac  -d  bin   src/pack/B.java
```

The -d flag tells the compiler where to put the .class file.

We then compile A with

```
javac  -d bin  -classpath bin  src/pack/A.java
```

This adds a -classpath flag that tells the compiler where to find .class files that are not part of the standard Java environment. In very complex examples you might want to gather classes from a different location than where you are putting the .class files, so the -d and -classpath flags are both necessary.

Finally, we run class A with another classpath flag:

```
java  -classpath  bin   pack.A
```

Just as with Eclipse, if the program is going to read any data files, they need to be in the project folder PROJ.