CS 210, Fundamentals of Computer Science I
Lab 7: Classes and Eclipse
Upload to Canvas

BELLEVUE COLLEGE
COMPUTER SCIENCE

The goal of this lab is two-fold. First, you'll gain experience in writing a class, and creating several instances of an object defined in that class. And, you'll learn about Eclipse, another IDE.

I. Writing a class `Dog.java`

Recall from lecture that a class contains fields, and several methods, that manipulate and have access to the values stored in the fields. For this part of the lab, you'll write a class, `Dog.java`, that is meant to describe a dog. Although there are many characteristics of a dog that distinguish it from another dog, for this lab, each instance of a Dog object will contain only four fields, `heightIn, ageYears`, `name`, and `likesBarking` where `heightIn` refers to the height of a dog, in inches, and `ageYears` is the age of the dog, name is the name of the dog, likesBarking is whether the dog likes barking.

To complete this portion of the lab:

1. Create a new java file, using jGRASP.

2. As you've already done with several program for your programming assignments, write a new class, using the access specifier `public`, the java keyword `class`, and the name `Dog`:

```
public class Dog {
}
```

3. Save the file as `Dog.java`.

4. Add four fields to your class (refer to the lecture slides, for a sample). By convention, the fields are placed in the class file before the methods are written. The four fields should be:

   - `heightIn`, which should be private, and of type `double`
   - `ageYears`, which should be private, and of type `integer`
   - `name`, which should be private, and of type `String`
   - `likesBarking`, which should be private, and of type `boolean`

5. Now, on to add the methods. Recall that methods in a class are usually public, and are often used to manipulate the private fields. Methods that set the values of fields are referred to as setter methods, while methods that retrieve the values of fields are referred to as getter methods. As a first step, write the setter method that will update the value of the `heightIn` field. Recall that a method declaration begins with modifiers, followed by the return type, then the name, and finally the parameter list. For example, a method that is `public`, returns an `integer`, is called `calcExponent()`, and receives two parameters, the first an `integer` and the second a `double`, would be written as the following:

Method Modifiers	Return type	Method Name	(Method Parameters)
public	int	calcExponent	(int base, double exponent)

The setter method for the field `heightInt` should be `public`, return nothing, be called `setHeight`, and receive a single argument, of type `double`. And, the method should update the value of the field `heightIn`, using the value that the method receives. Thus:

```
public void setHeight(double height){
     heightIn = height;
}
```

6. Similarly, write the setter method for the `ageYears` field:

```
public void setAge(int age){
     ageYears = age;
}
```

7. Write the setter methods for the `name` and `likesBarking` fields. (not shown)

8. Now that you've written the setter methods, write the getter methods, which are used to retrieve the values of the fields. There are four getter methods that you should write, `getHeight()`, `getAge()`, `getName()` and `getLikesBarking()`. The first one, `getHeight()`, should be:

```
public double getHeight(){
    return heightIn;
}
```

Now write the `getAge()`, `getName()` and `getLikesBarking()` methods, into your `Dog.java` class. They are very similar to the `getHeight` method.

9. At this point, that's it. You've written a class, `Dog.java`, that is meant to store information about a dog. Compile the class, and fix any syntax errors that you may have. Note that if you run the compiled class, nothing will happen, because the java code that you've written, does not contain a `main` method, which is where all java programs begin.

II. Writing a `DogKennel.java` program, that uses the `Dog.java` class

Now that you've written the *Dog.java* class, you can create objects of that type. This is similar to what was shown in lecture, with the *Bicycle.java* class example. For this part of the lab, you'll write a `main` method that creates three objects of type `Dog`. You'll use the setter methods to set the private fields of the `Dog` objects, and the getter methods to retrieve information about the dogs. To complete this portion of the lab:

1. Create a new java file, using jGRASP, and call it `DogKennel.java`. The skeleton of the file should be the following:

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

    }
}
```

2. Begin your program by issuing a `System.out.println` statement, so that the following text is printed to the screen, when your program is run: Welcome to the dog kennel

3. As was shown in lecture, now that you have a class, `Dog`, that you've written, you can create objects of that type. For this part of the lab, create three objects of type `Dog`. Remember that you need to create an object of type `Dog` using the java keyword `new`, and assign the object to a reference variable, which should be a newly created instance of an object of type `Dog`:

```
Dog dog1 = new Dog();
Dog dog2 = new Dog();
Dog dog3 = new Dog();
```

4. At this point, you have three reference variables, `dog1, dog2`, and `dog3`, that refer to three just-created objects of type `Dog`. However, the fields of those objects are empty, but you can set their values using the setter methods, that you've defined in the `Dog` class. To illustrate, to set the value of the `heightIn` field in the first `Dog` object (`dog1`) that you've created, you'd write:

```
dog1.setHeight(13.5);
```

Recall that the "dot" notation is used to invoke a method for a specific object. In the above code, you are invoking the `setHeight` method of the first `Dog` object that you've created. Similarly, to set the value of the `name`, `age` and `likesBarking` fields of the first `Dog` object, you'd write:

```
dog1.setAge(8);
dog1.setName("Igor");
dog1.setLikesBarking(true);
```

5. Update the fields of the second and third `Dog` objects that you've created. Here are the heights, ages, and names of the `Dog` objects that were used to create the output shown in Figure 1 (see below).

	age	height	name	likeBarkiing
dog1	8	13.5	Igor	true
dog2	5	8.2	Lev	false
dog3	8	15.1	Frost	true

6. At this point, you have created three objects of type `Dog`, and you've set the fields of those objects. Next, you can use the getter methods, to retrieve the information about each `Dog` object. The getter methods are used the same way as the setter methods, in that you should use the dot notation. All three getter methods that you've written, `getHeight, getAge`, and `getName`, receive as input no arguments, but return values of type `double`, `int`, and `String`. So when you invoke those methods, you can use their return values to update a variable. For this part, compute the sum of the ages of the three `Dog` objects, and print the sum to the screen. You can achieve that in just two statements:

```
int sumOfAges = dog1.getAge() + dog2.getAge() + dog3.getAge();
System.out.println("The sum of the dogs' ages is " + sumOfAges);
```

7. Use the instructions in step 6 as a sample, and also sum the heights of the three dogs, and print that value to the screen.

8. Use the `getName()` method, to retrieve the name of each `Dog` object, and use a `System.out.println` to print that information to the screen.

9. Finally, use the `getLikesBarking()` method for each dog to determine how many dogs like barking. Print `"n dogs like barking"`, where n is the number of dogs that have likesBarking to set to true.

10. Compile your program (fix any syntax errors), and run it. If done correctly, the output should be very similar to what is shown in Figure 1. With that, you are done with that programming part of this lab, and you've written (and used three times), your own class, `Dog.java`.

```
Welcome to the dog kennel
The sum of the dogs' ages is 21
The sum of the dogs' heights is 36.8
The dogs are: Igor, Lev, Frost
2 dogs like barking
```

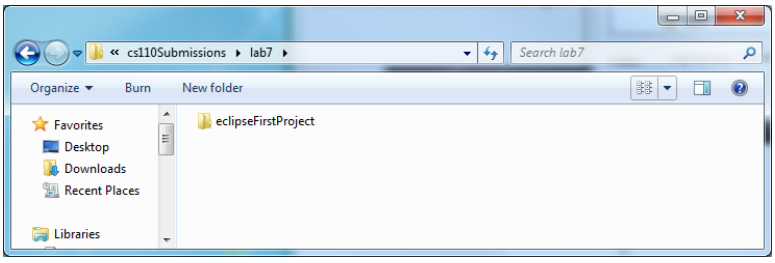
Figure 1: Sample invocation of the program DogKennel.java

III. Using Eclipse (optional)

In this final section, you'll be introduced to the Eclipse Integrated Development Environment (IDE). It is another IDE, in addition to JGrasp, that you already know. Eclipse offers many features that JGrasp does not. From this point forward, you can complete your homework assignments using either JGrasp or Eclipse; it's up to you. In this part of the lab, you'll learn about the most basic features of Eclipse: how to create a project, how to create a new java file, how to debug, and how to run a simple java program.

1. Create a folder called *eclipseFirstProject* on your computer. When you've done that and navigated to that location on your computer, it should look similar to the the image in Figure 2.

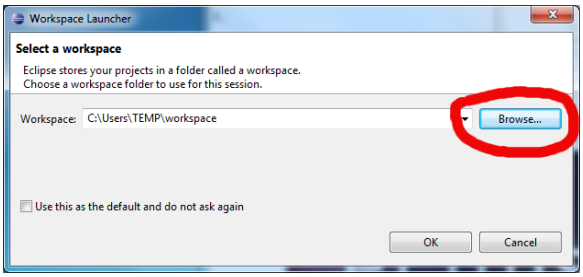
Figure 2: Contents of a folder where a directory eclipseFirstProject has been created.



2. If you don't have Eclipse downloaded on your computer, download it at: https://www.eclipse.org/downloads/packages/release/juno/sr2/eclipse-ide-java-developers.
To start Eclipse, left-mouse-click the Windows icon on the bottom right-hand-corner of the screen, and type "eclipse" into the "Search programs and files" box. Left-mouse-click on the "Eclipse-Juno" icon when the program is found.

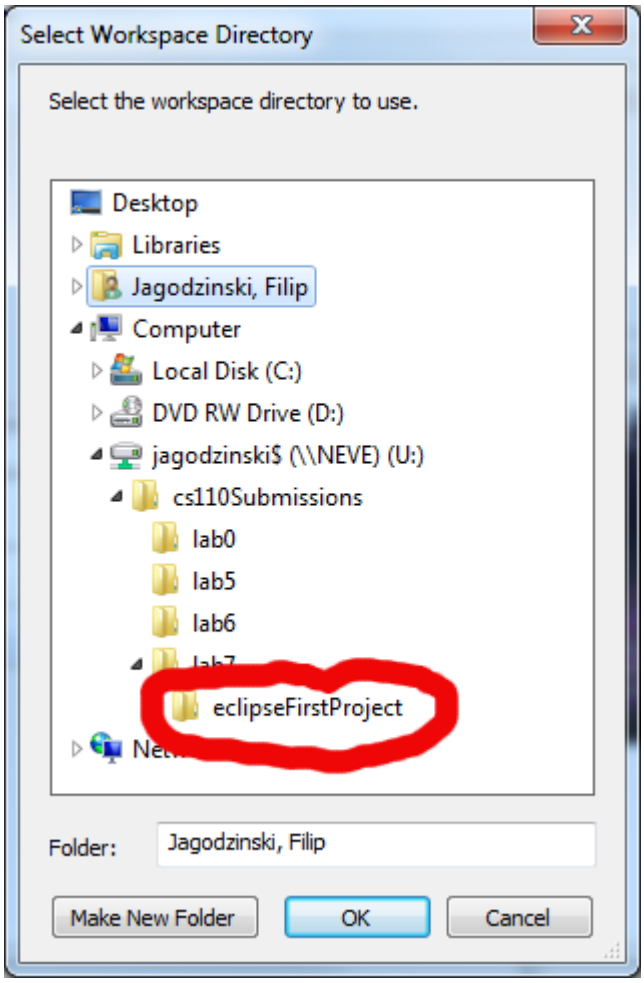
3. When Eclipse starts, you will be asked to select a workspace, as shown in Figure 3. Click on the Browse Button. **If you receive an Eclipse error (it will happen for only a few students), download the eclipseOpenErrorFix.vbs file from the course website, save it to your desktop, and double click it, to run the script (program)**.

Figure 3: The Select Workspace panel, when you first open Eclipse.



4. When the Select Workspace Directory panel appears (Figure 4), select the "eclipseFirstProject" folder that you created in step 1 of this section of the lab, and click on "OK" to confirm your selection.

Figure 4: The Select Workshops Directory panel of Eclipse.



5. Once you've selected the Eclipse Workspace directory, you'll be returned to the Workspace Launcher panel (as shown in Figure 5). Make sure that the Workspace input box lists the location where you created the directory eclipseFirstProject. Click on OK to confirm your selection.

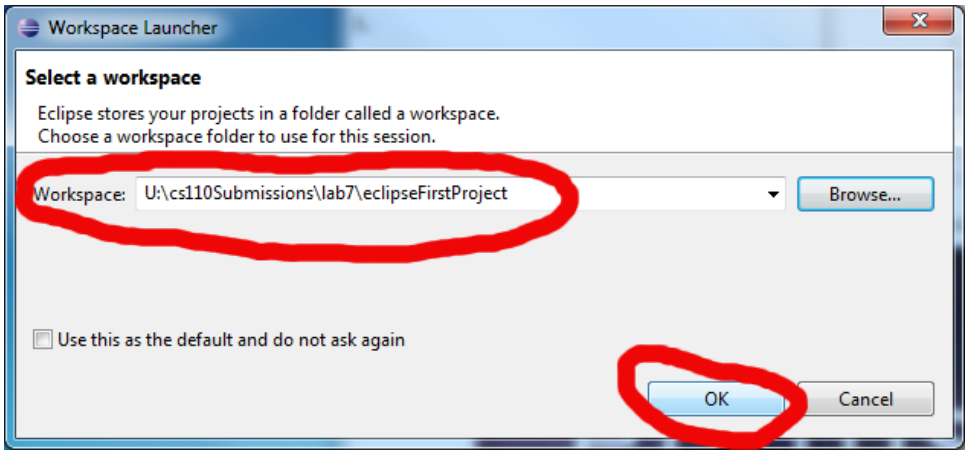


Figure 5: Confirm that the workspace directory for Eclipes.

6. Once you've indicated the workspace directory, the Eclipse desktop will appear, and should look something similar to Figure 6. You can close the welcome screen by clicking on the X to the right of the word "Welcome."

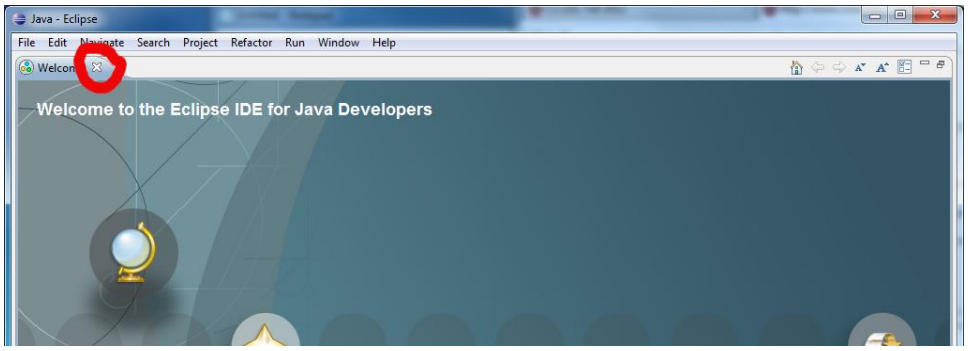


Figure 6: The welcome desktop of Eclipse

7. Eclipse is a powerful tool that can be used to create large-scale java projects, with many java classes, packages, input files, etc. To create a new java project, select File, then New, then Java Project, as shown in Figure 7.

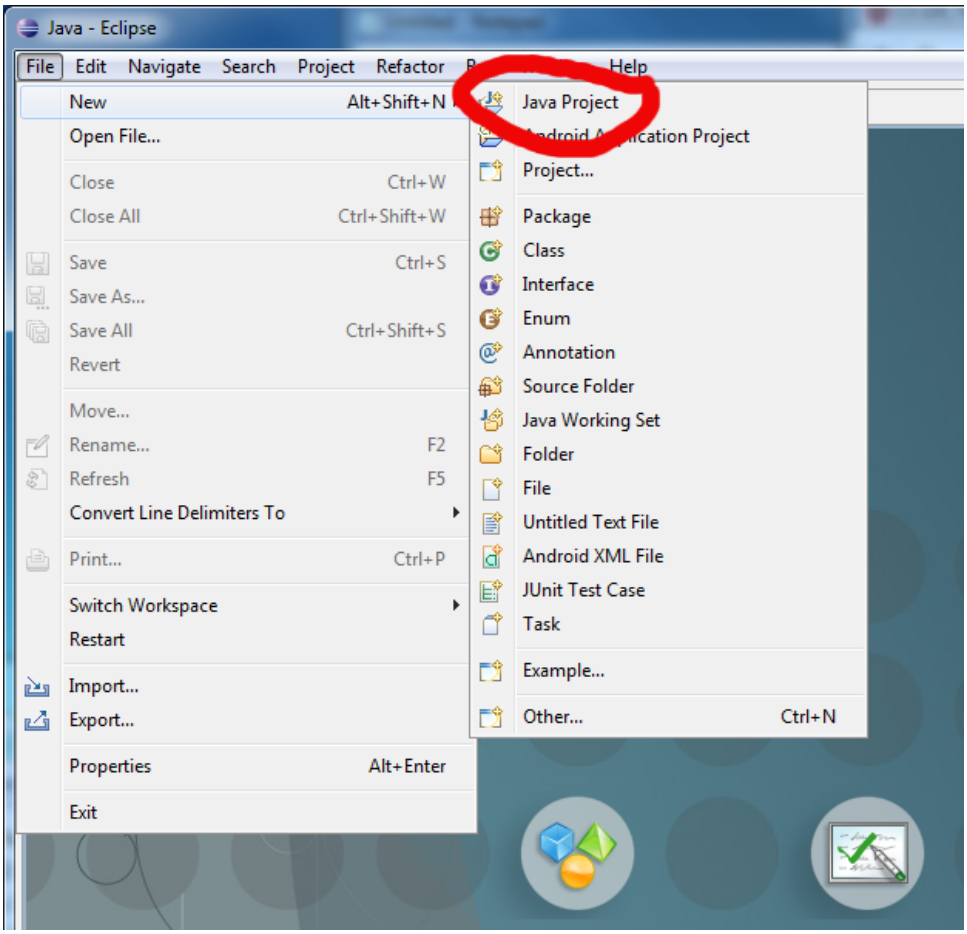


Figure 7: Creating a Java project in Eclipse

8. When the New Java Project panel appears, type “Lab7FirstEclipseProgram” into the Project name input box (Figure 8), and leave all other options as they are. Then, click on Finish.

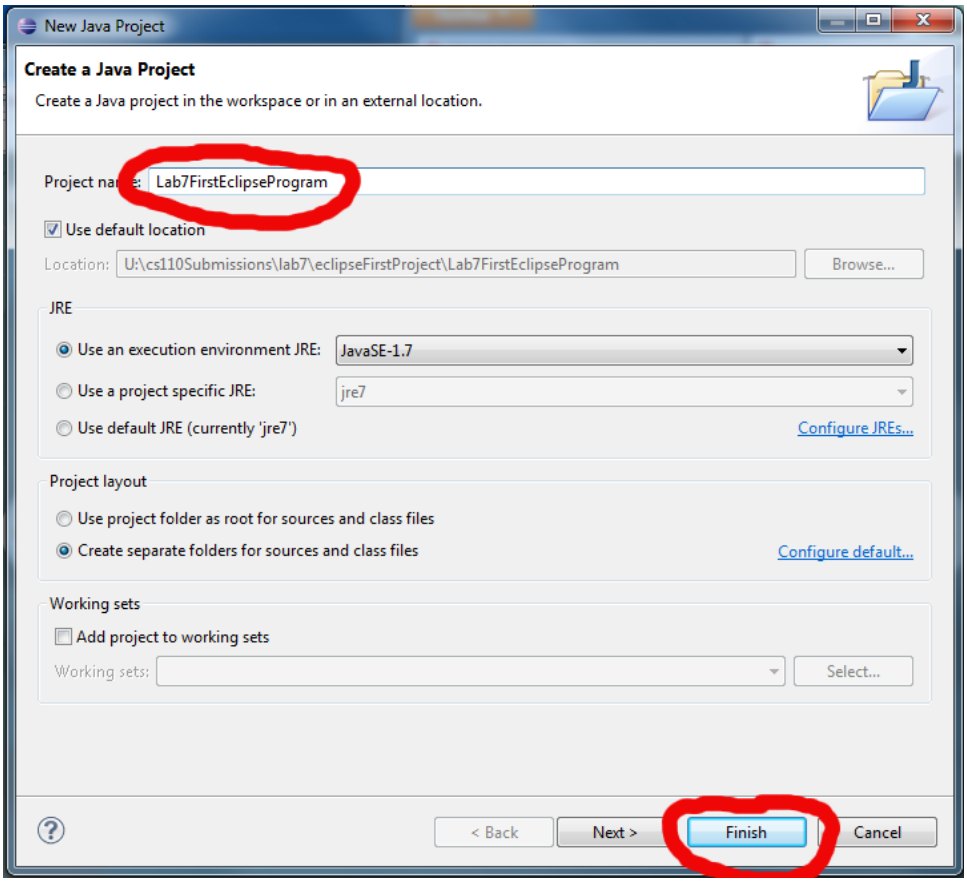


Figure 8: Creating a new Java Project called Lab7FirstEclipseProgram

9. Once you've created a new java project, you need to perform the same steps in Eclipse that you perform in JGrasp. Namely, you need to create a new class. To do that, click on the "src" icon in the Package Explorer panel of the project that you've just created, select New, followed by Class (Figure 9).

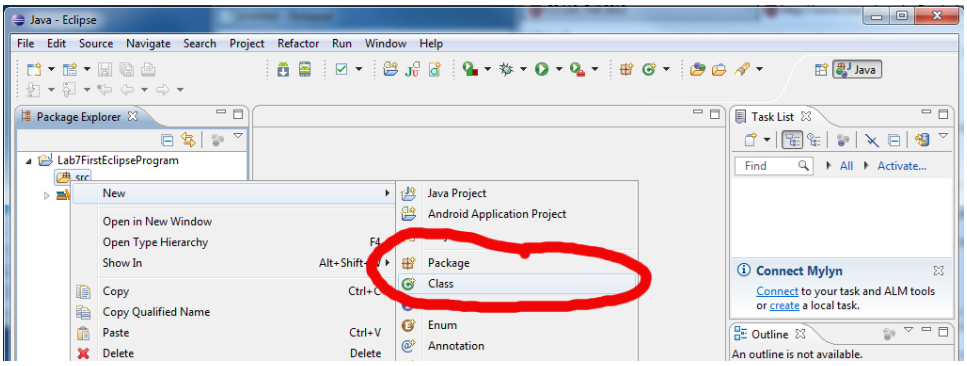


Figure 9: Creating a new Class within your new Java Project

10. At this time, the New Java Class panel should appear. There, make sure that the Source folder is listed as Lab7FirstEclipseProgram/src. Then, in the input filed named Name, type the name of the class that you want to create: MyFirstEclipseJavaProject (example shown in Figure 10). Leave all other options as they are. Click on Finish, to confirm making a new class.

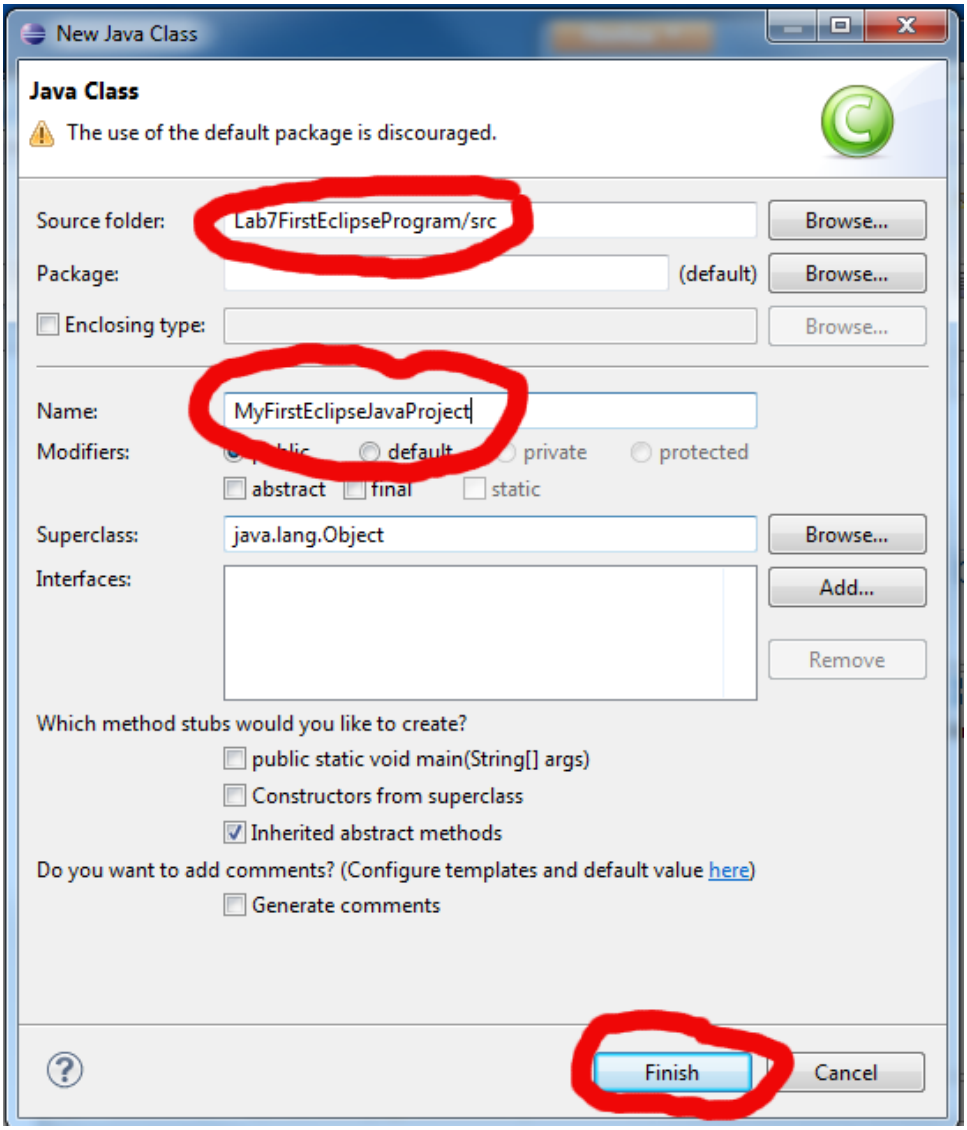
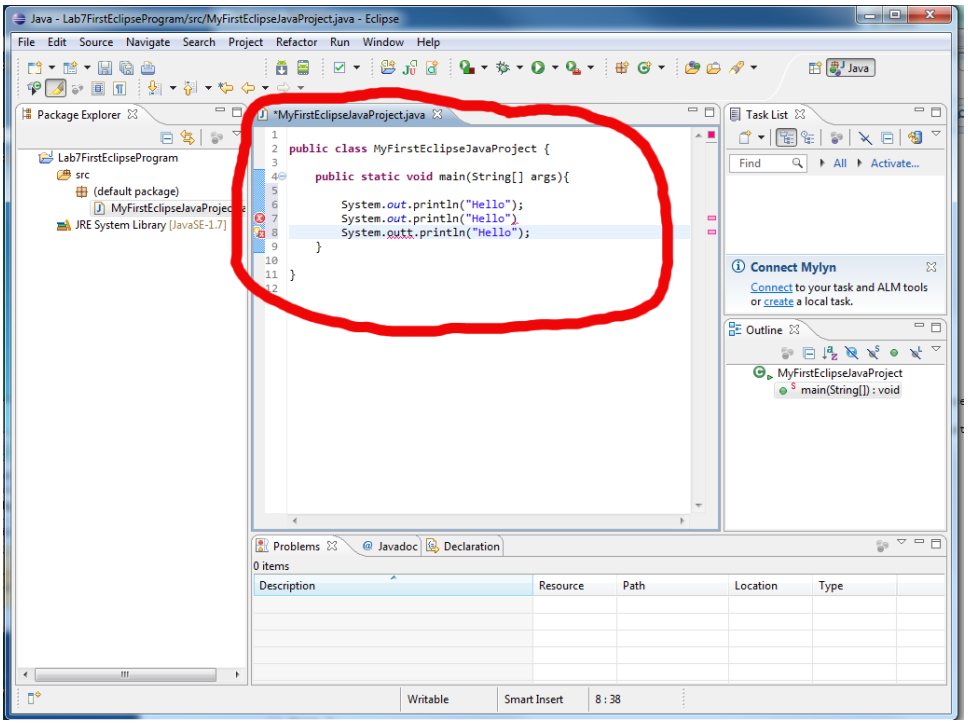


Figure 10: Create a new java class

11. At this point, you have created a new Java project, and a new Java class, named MyFirstEclipseJavaProject. Just like jGRASP, Eclipse has several windows, that you use to write, compile, and run a program. In the just created class that you've made, write the main method. Type the following code into the editor window (Figure 11) **Yes, intentionally type Syntax errors** :

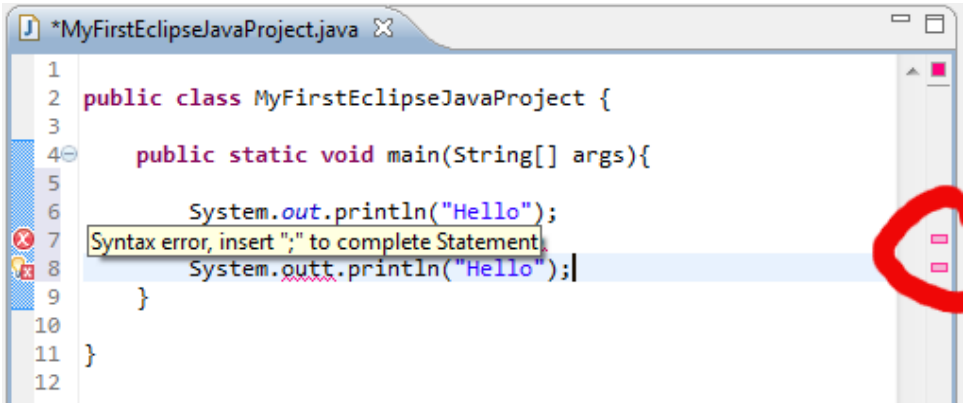
```
public static void main (String[] args){
     System.out.println(“Hello”);
     System.out.println(“Hello”)
     System.outt.println(“Hello”);
}
```

Figure 11: A sample java class file with intentional syntax errors.



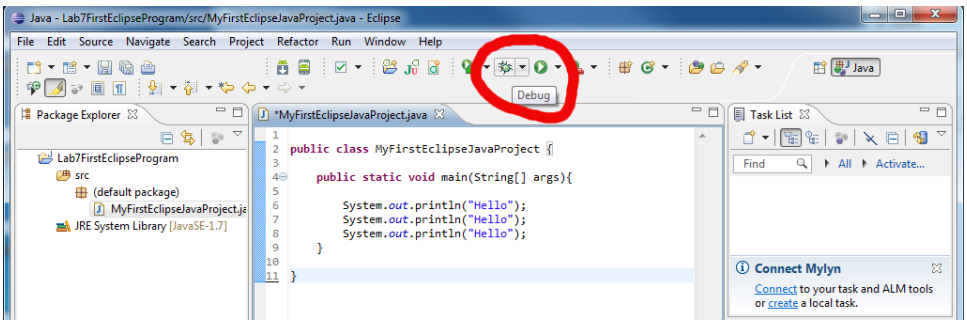
12. As you type, notice that Eclipse recognizes syntax errors, and indicates with a red square on the right-scroll bar of the class, where those errors occur. Also, Eclipse underlines with a red squiggly line, the location of the syntax error, in the code (Shown in Figure 12).

Figure 12: Eclipse locates/identifies syntax errors as you type



13. Fix the syntax errors in your code, and select the Debug button, to see if there are syntax errors remaining (Figure 13). If you've fixed your syntax errors, proceed to click the Run button, and the output of your program will appear in the Console window of Eclipse (Figure 14).

Figure 13: The Debug button in Eclipse



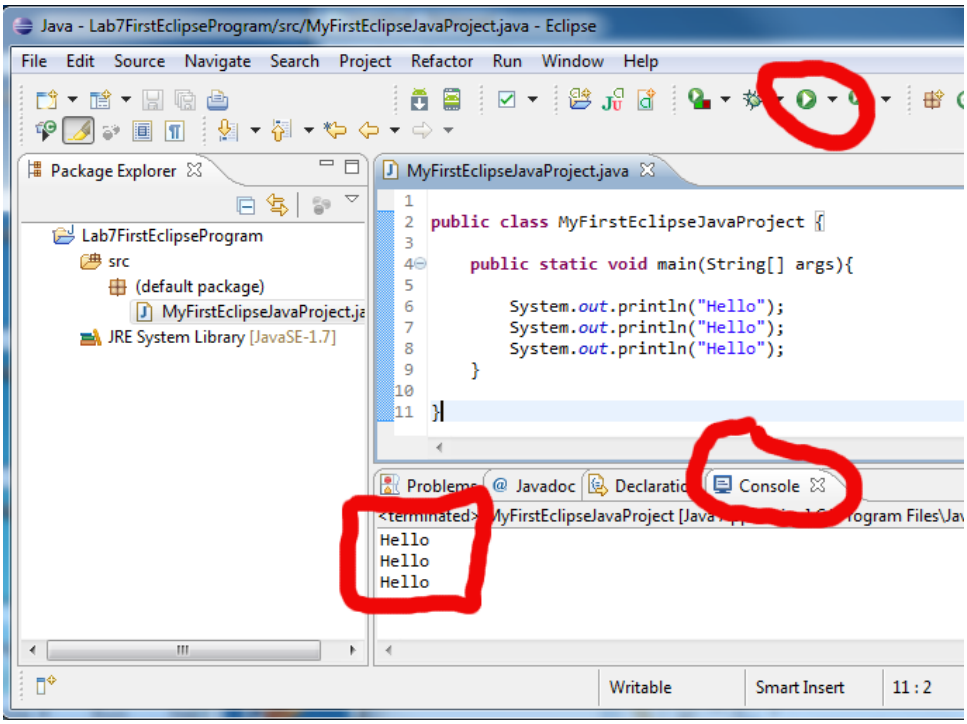


Figure 14: The run button, the console window, and the output of a program in Eclipse

14. Once your program compiles, you are done.

15. From now on, you can use Eclipse or jGRASP to complete your labs and/or homework assignments.

What to hand in

Make sure to upload the following two files to Canvas zipped:

Dog.java
DogKennel.java

Be sure that each .java file is commented, and be sure to include your name at the top of each file. Code must be indented, so that it is easy to read. Finally, make sure that you have given your variables good names.

V. Rubric

File / task	Points
Dog.java compiles and all fields and methods have been included	40
DogKennel.java compiles and correctly creates three objects of type Dog. The setter and getter methods are used correctly, and output is similar to what is shown in Figure 1.	40
All .java files are commented, and contain your name and date, code formatted properly	5
Variable names are adequate and descriptive in all java files	5
Image of MyFirstEclipseJavaProject compiled in Eclipse uploaded	10
Total	100