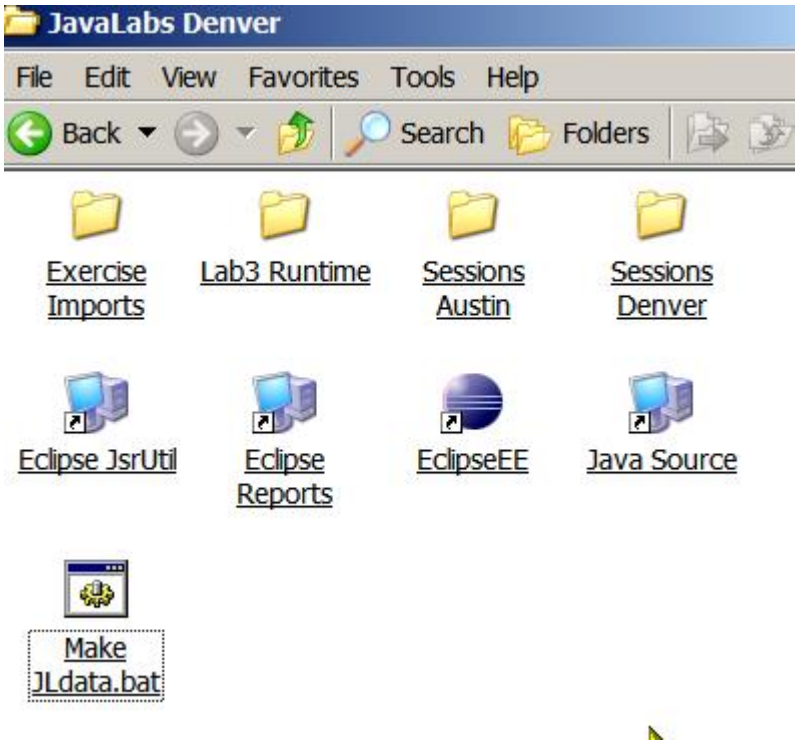
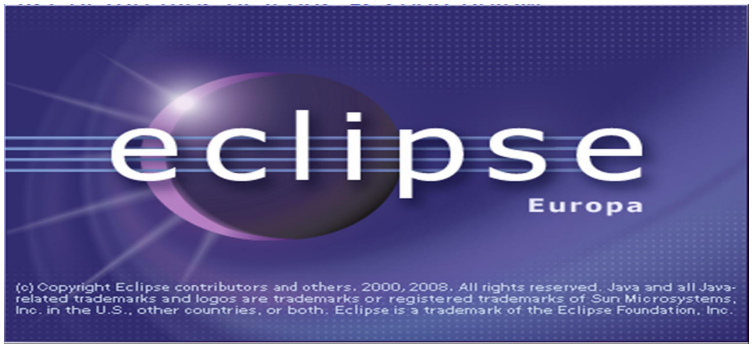
S1181 Java and Eclipse for the Beginner Programming Hands-on Lab Part I

In this introductory hands-on lab session you will be using the open source Eclipse development environment. The Eclipse IDE provides all of the tools necessary to build, compile and run Java applications.

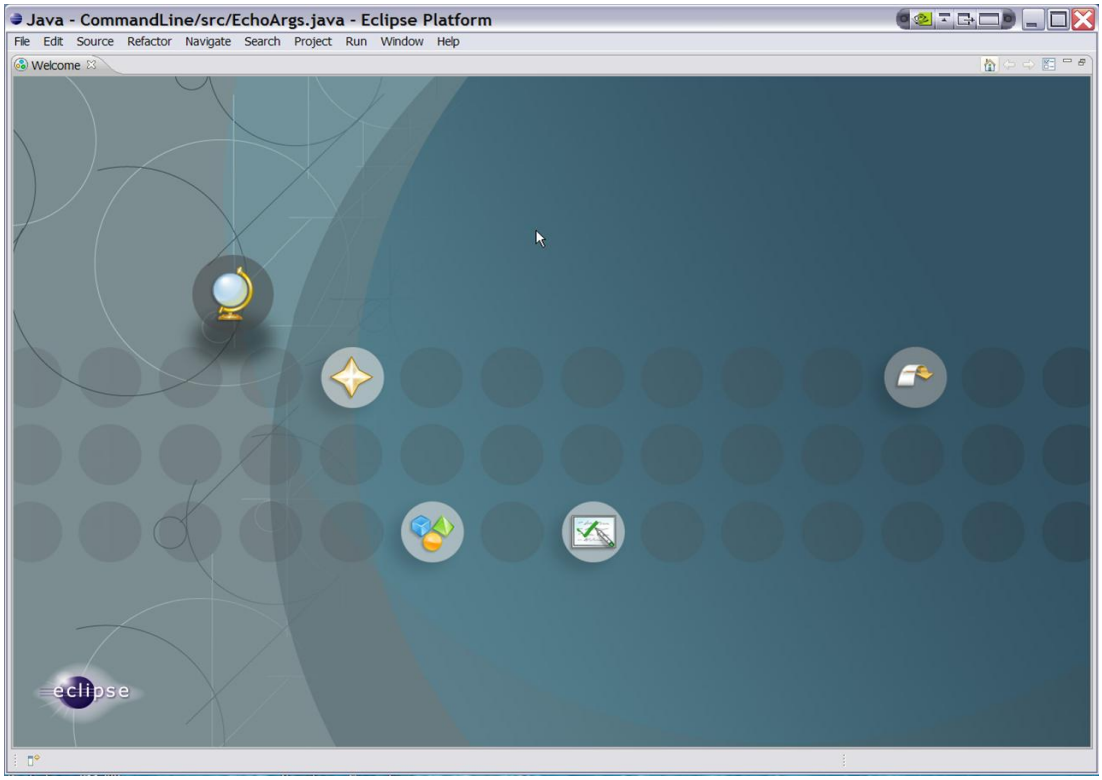
From the lab machine look for JavaLabs folder and double click it. You will see either a list of the files or icons as seen below.



The lab instructor will demonstrate how to use Eclipse tool (already loaded) on the lab machines. Please look for  Eclipse icon on your desk top and click to open the tool. You should see the following visualization…

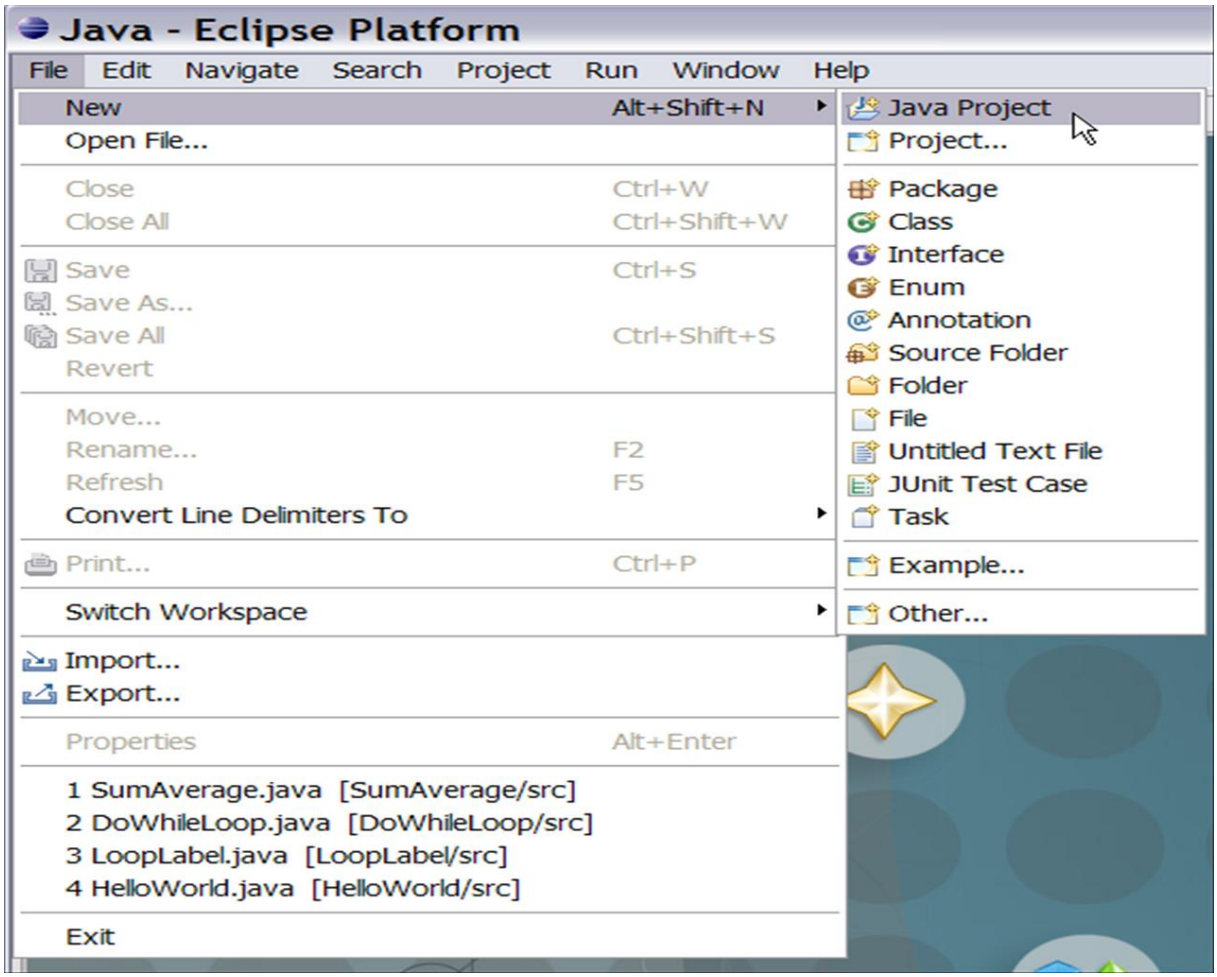


The Welcome page:

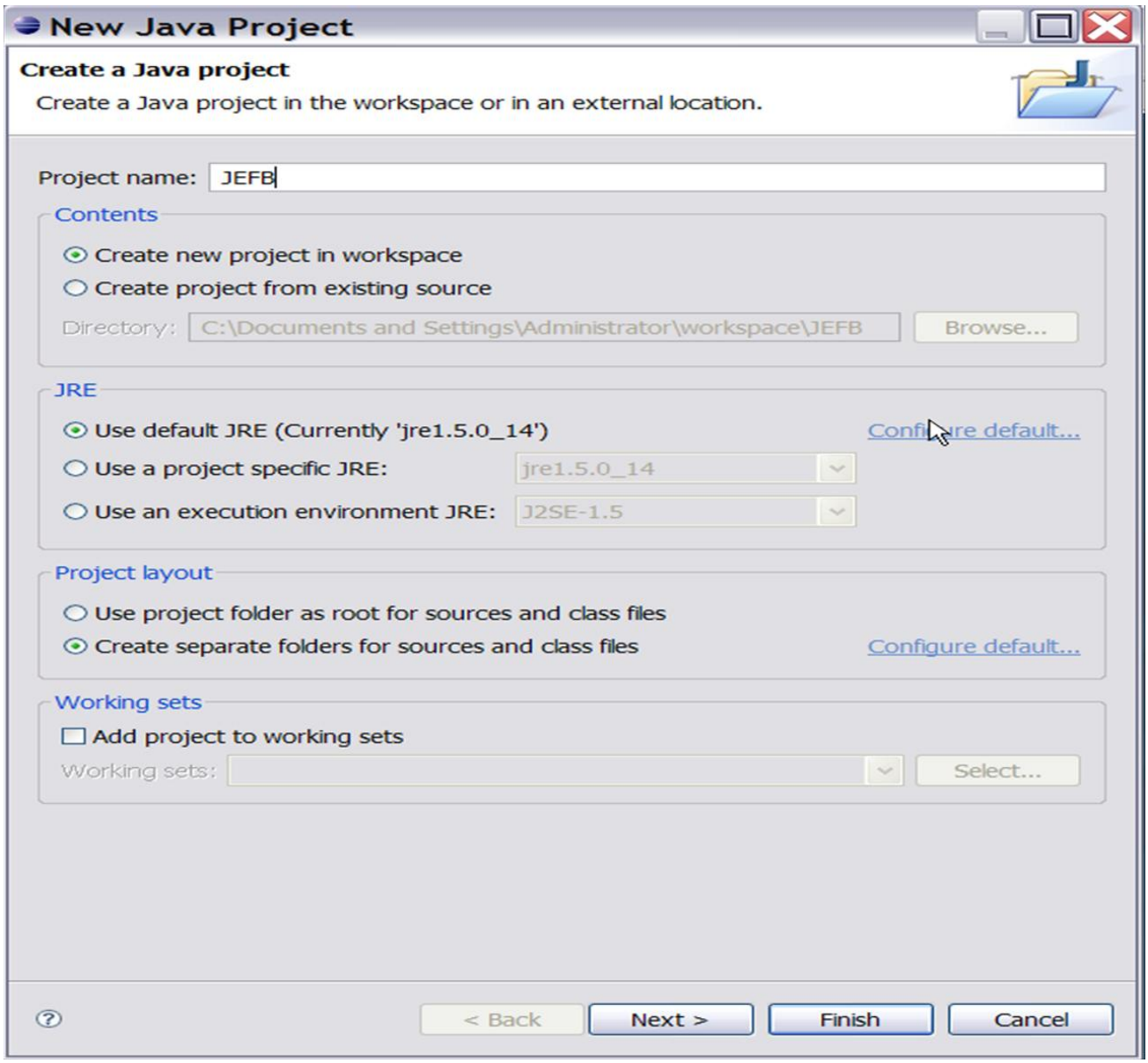


Please note: You'll see the Welcome page only on the first entry into Eclipse.

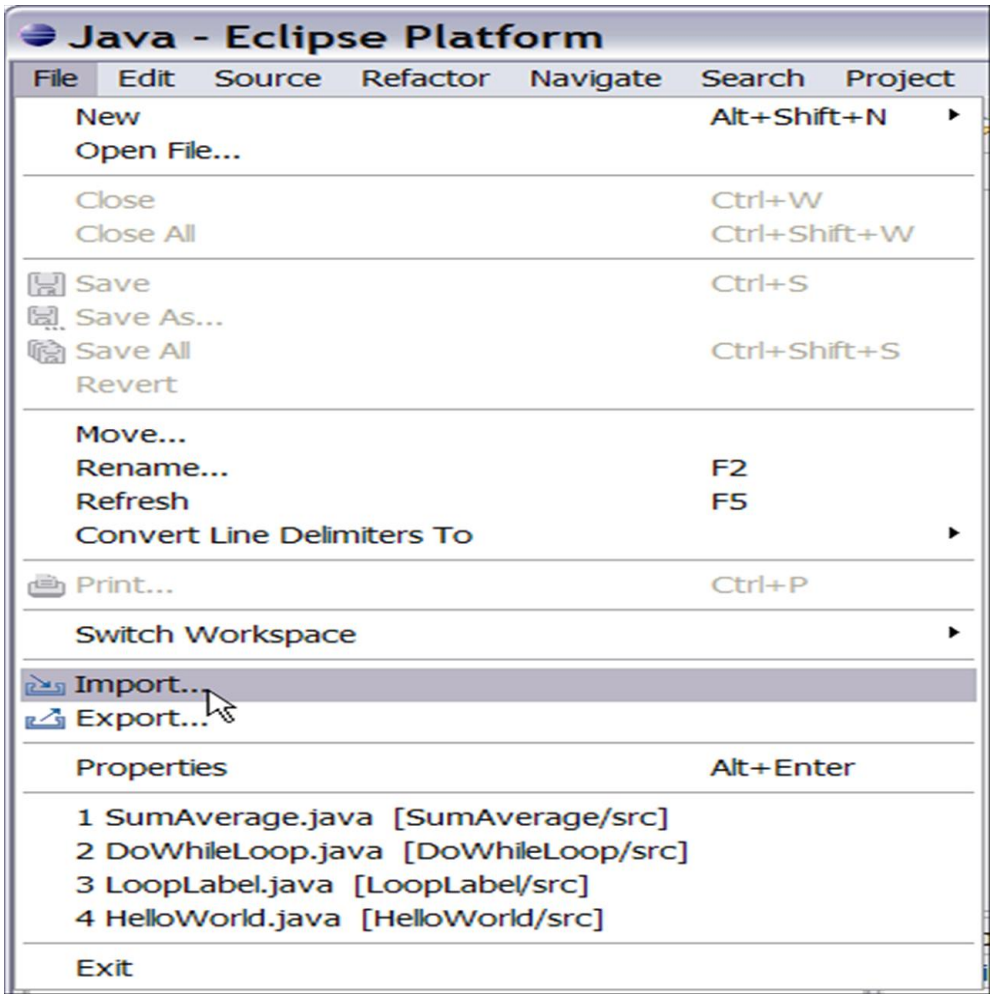
Once Eclipse has loaded, create a new project by selecting File -> New -> Java Project.



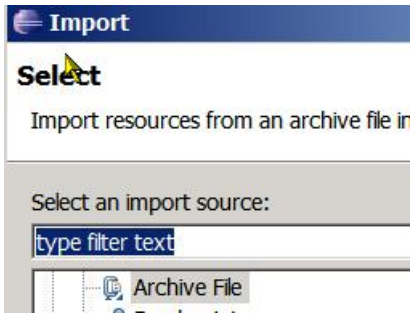
In the Project Name text box, enter JEFB, which stands for “Java and Eclipse for the Beginner” and click Finish. This will create a new project in Eclipse, which will contain our lab exercises.



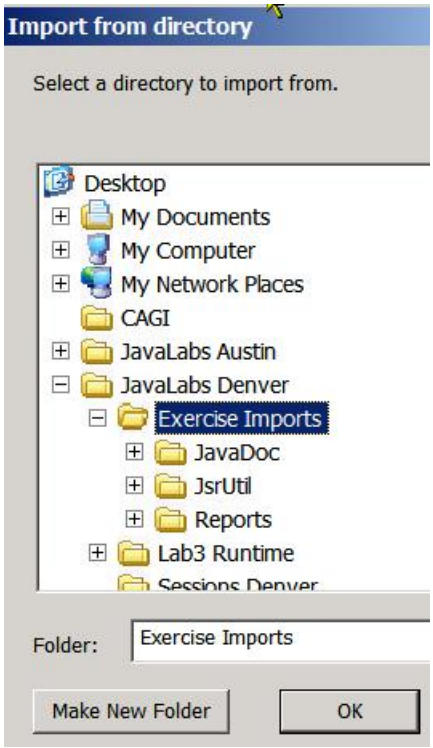
Once the new JEFB project has been created, select File and Import

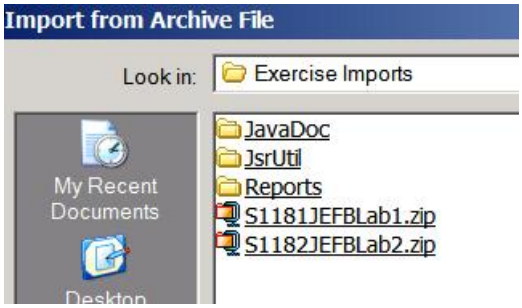


Select General folder Archive file and click Next.

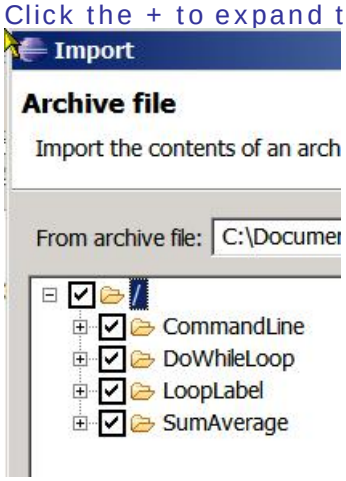


Click the Browse button to the right of the From Archive file: text box and search for the folder under
Desktop > Javalabs Denver > Exercise Imports





Click to Select S1181JEFBLab1.zip
Click the + to expand the contents of the zip file which will be selected by default.

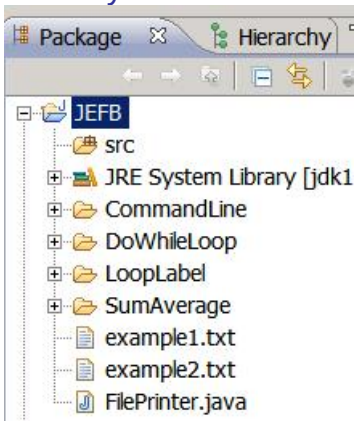


and click Finish. Repeat for S1182JEFBLab2.zip.

If you receive any messages asking about whether Eclipse should overwrite certain files, such as the .classpath file, then click Yes To All.

Once the project has been imported, you will be presented with the Java Perspective, which shows the project you have just imported. This project can be expanded, by clicking the “plus” symbol to the left of the project name.

Note: you will see either JEFB (Java and Eclipse For the Beginner) as shown.



Also notice the tasks view in the bottom-right of the screen. This reports that two compilation errors currently exist in the project. These will be resolved as part of the exercises…

Exercise #1 Labeled for Loops and break Statement

In the following code example LoopLabel, you will have the opportunity to code a Java class, method using most of the basic construct introduced in the lecture. Please pay attention to the purpose of the label "myloop". In the for loop, when a particular condition is met, a break statement would exist the inner loop and resume execution with the outer loop. The code exercise below contains a nested for loop. Inside the innermost loop, if the summed values of the two counters is greater than 5 both loops exit at once.

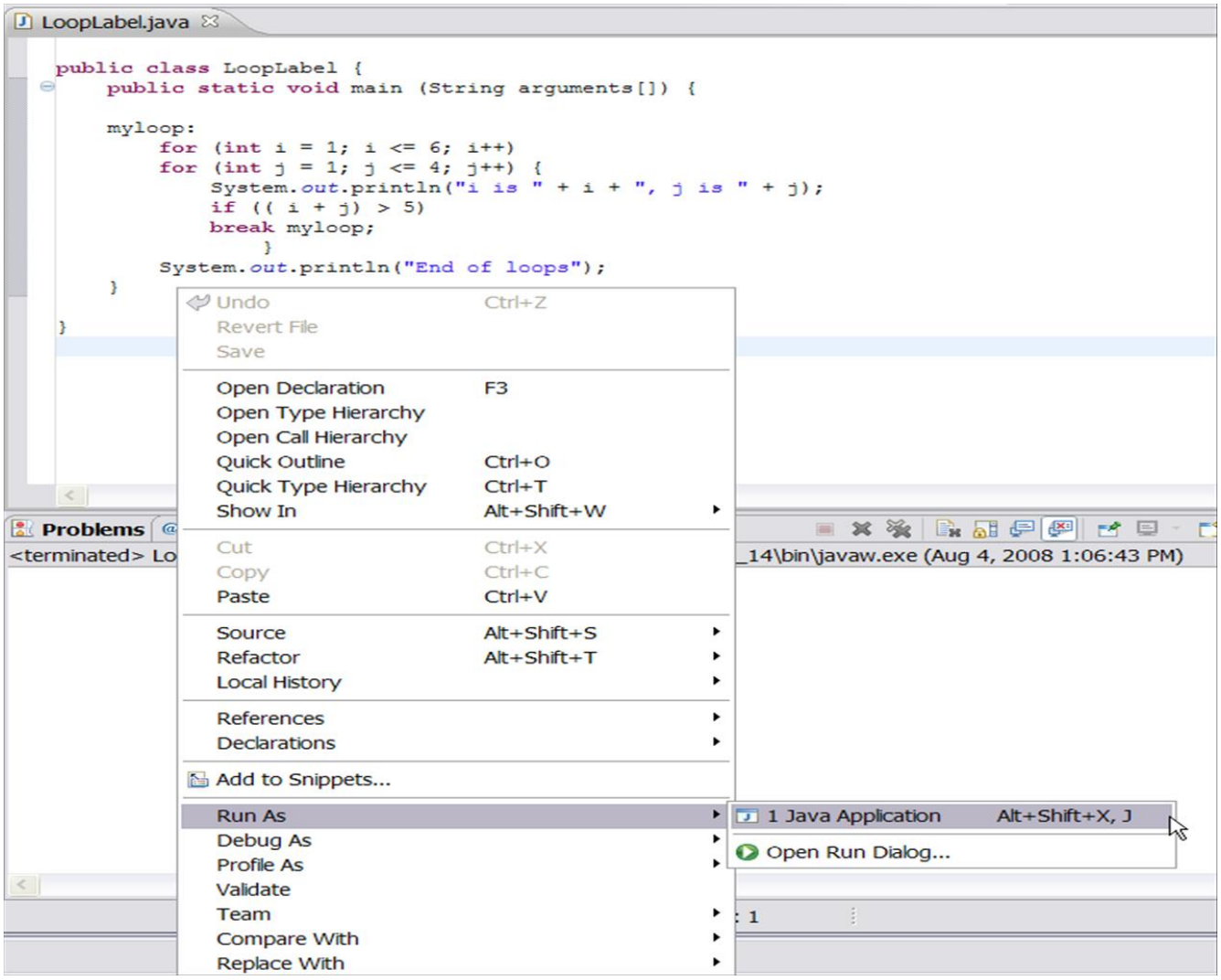
You can either cut-n-paste the code below or type as is into LoopLabel.java file under the LoopLabel folder. The purpose of exercise #1 is for you to get comfortable with the Java perspective, using the Java editor, syntax check, save and run the Java application, and see the result set under the console view. If you are familiar with Eclipse, please skip ahead to Exercise #2.

```
public class LoopLabel {
      public static void main (String arguments[]) {

      myloop:
            for (int i = 1; i <= 6; i++)
            for (int j = 1; j <= 4; j++) {
                  System.out.println("i is " + i + ", j is " + j);
                  if (( i + j) > 5)
                  break myloop;
                        }
         System.out.println("End of loops");
      }

}
```

After you have completed the code snippet, Ctrl+S to save or simply click on the  Save icon. You are now ready to Run the LoopLabel.java

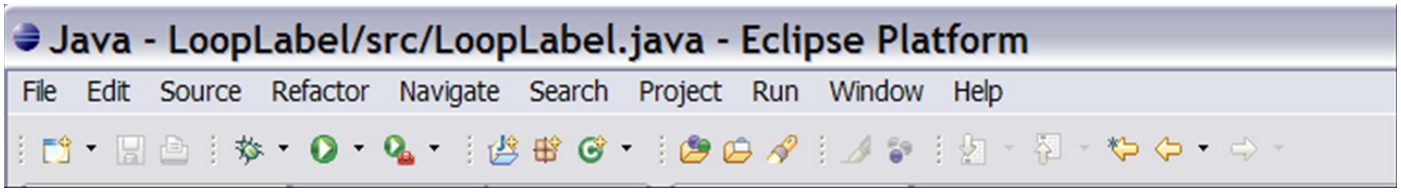
Remember to do a save every time you make changes to the code before running the application.

Typically you can run Java application by right click inside the editor of LoopLabel.java. Select the Run As in the pop-up window and click Java Application.

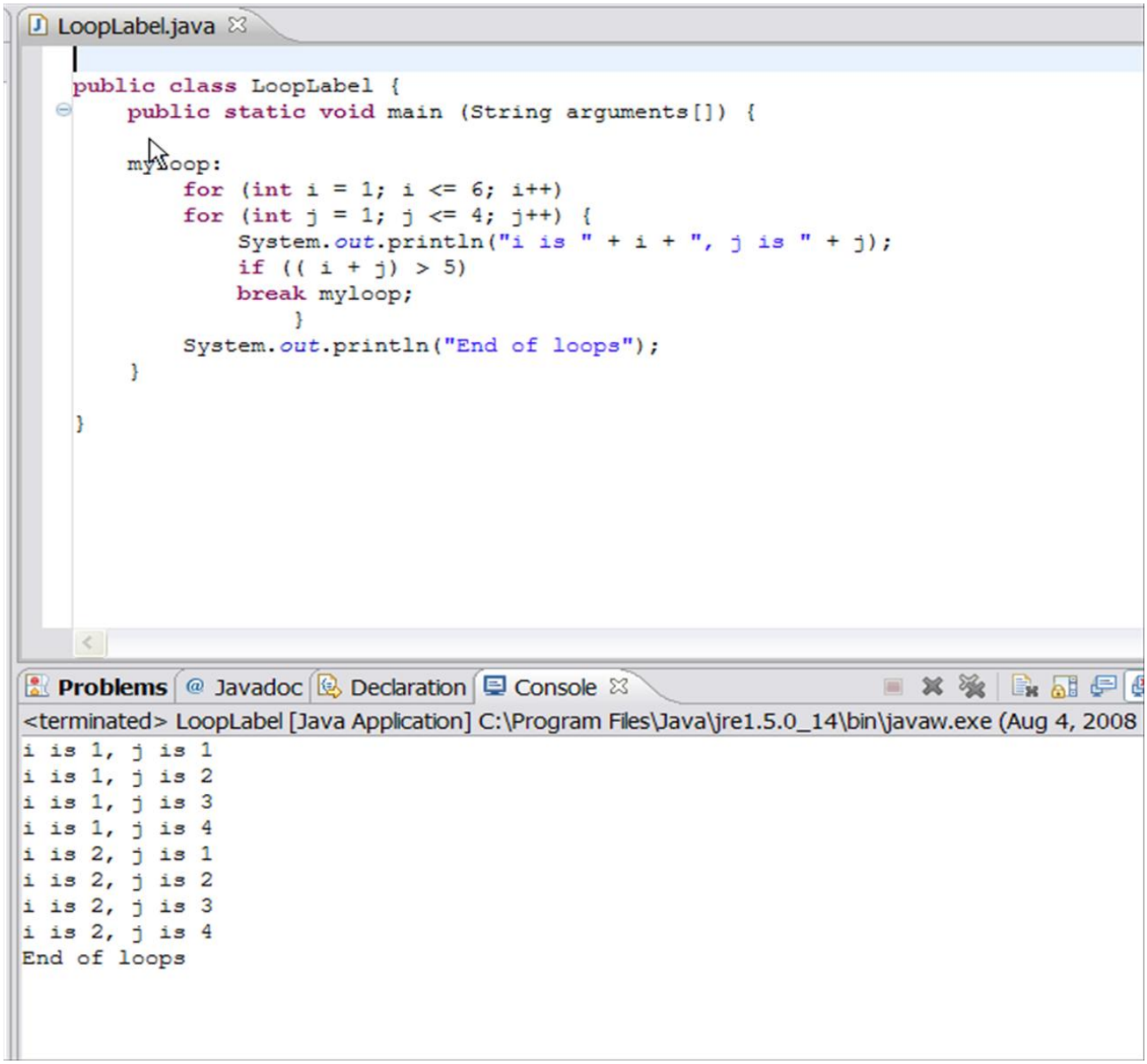


Or you can run the LoopLabel.java application by point and click on the run menu icon  from the toolbar (see below) and click on Run As… Java

Application. The run menu toolbar also has the Open Run Dialog that you will be using with the Command-line exercises.



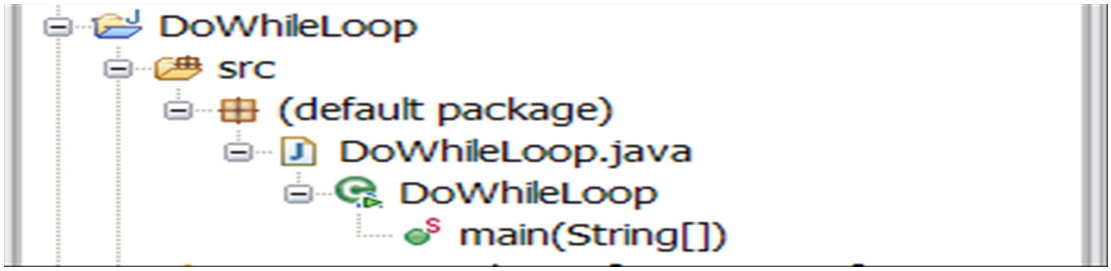
You should see the result set in the console view below:



Exercise #2 The do…while Loops

Please complete the following DoWhileLoop by providing a do while loop that will print a message each time the loop iterates “Looping, round # “ 1 through 10. Keep in mind that the body of the loop executes at least once with do loops.

1. Open the DoWhileLoop.java file under the DoWhileLoop folder

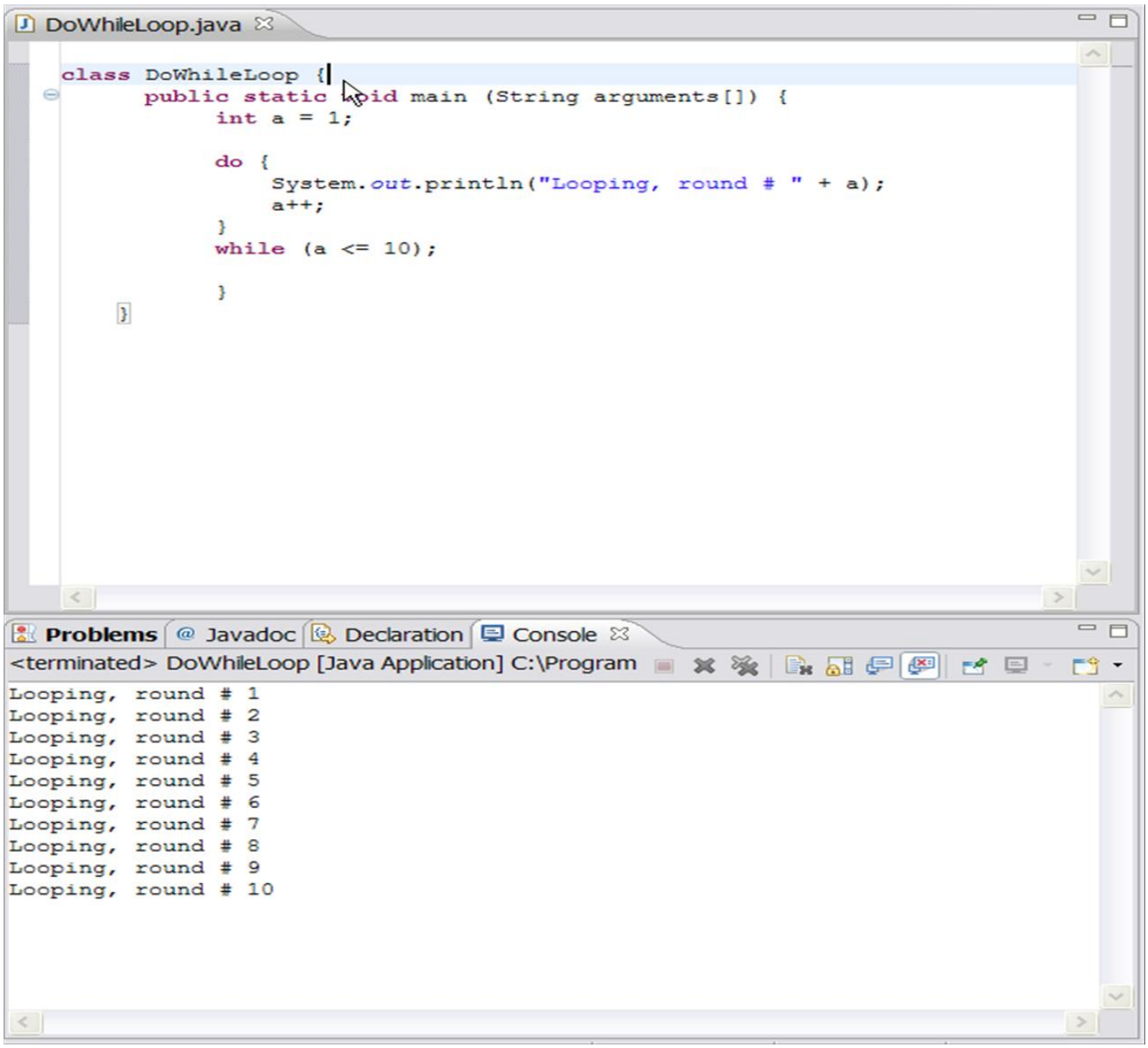


2. You now need to declare an integer variable and a do while loop.

```
// TO DO:
      // Declare an integer variable with an initial value of 1

class DoWhileLoop {
       public static void main (String arguments[]) {
          int

      // TO DO:
       // A print statement of  “Looping, round # “  and the number,
         increment the variable by 1 on each iteration, and exit when the
         variable reached number 10.

           do {
              System.out.

           }
           while ();

           }
     }
```

After you have the clean code (no more x), Ctrl+S to save or simply click on the  Save icon. You are now right to Run the DoWhileLoop.java

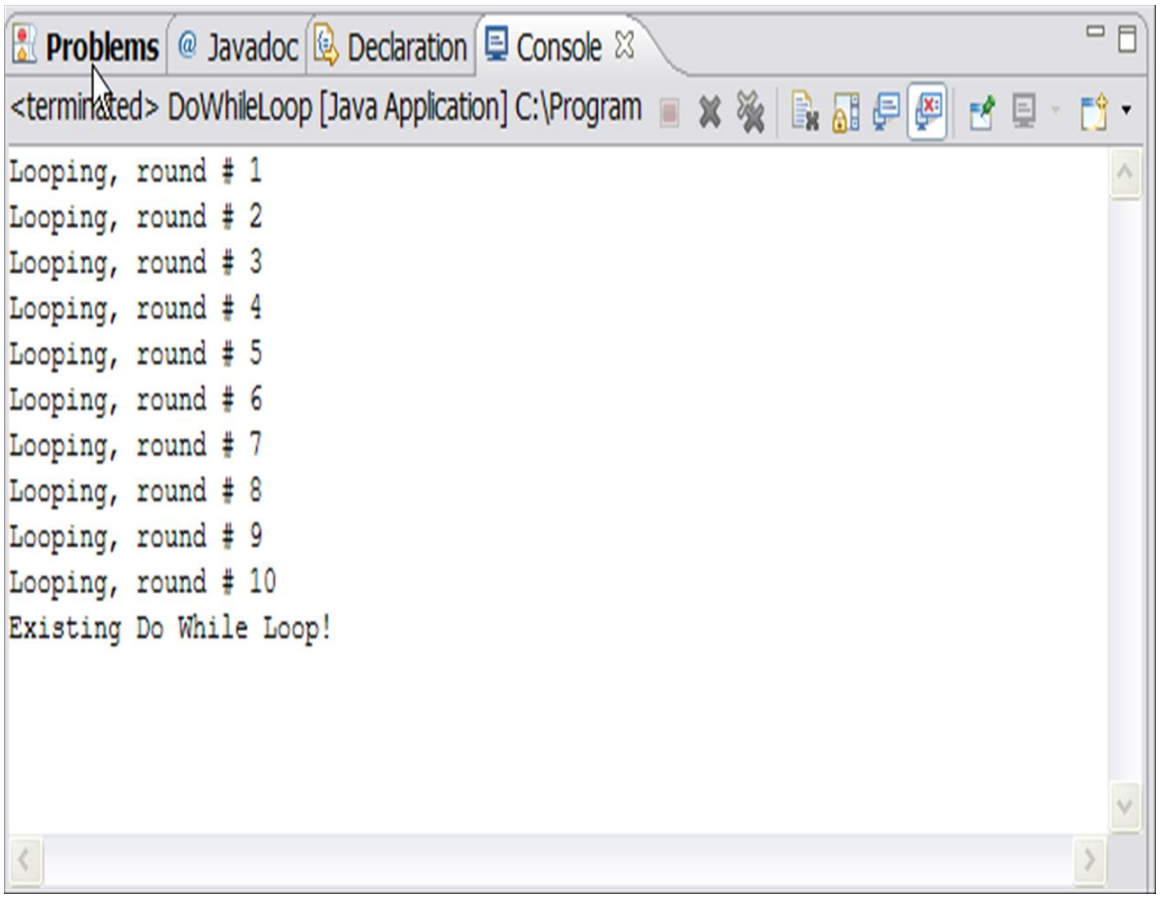
Sample Solution for Exercise #2:



Exercise #2-1 Adding a System.out.println statement after the do while loop (Optional exercise)

```
class DoWhileLoop {
        public static void main (String arguments[]) {
            int a = 1;

            do {
               System.out.println("Looping, round # " + a);
               a++;
            }
            while (a <= 10);
               System.out.println("Existing Do While Loop!");
            }
      }
```

Please note of the result set for the 2nd System.out.println statement after the do…while loop.



Exercise #3 Java Applications and Command-Line Arguments

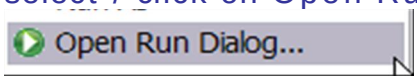
We are going to write a simple Java application that handle arguments from the interactive Run dialog. Java stores the arguments as an array of strings and passes the array to the application main() method. The fun part is that you can use the arguments to determine how your application is going to run.

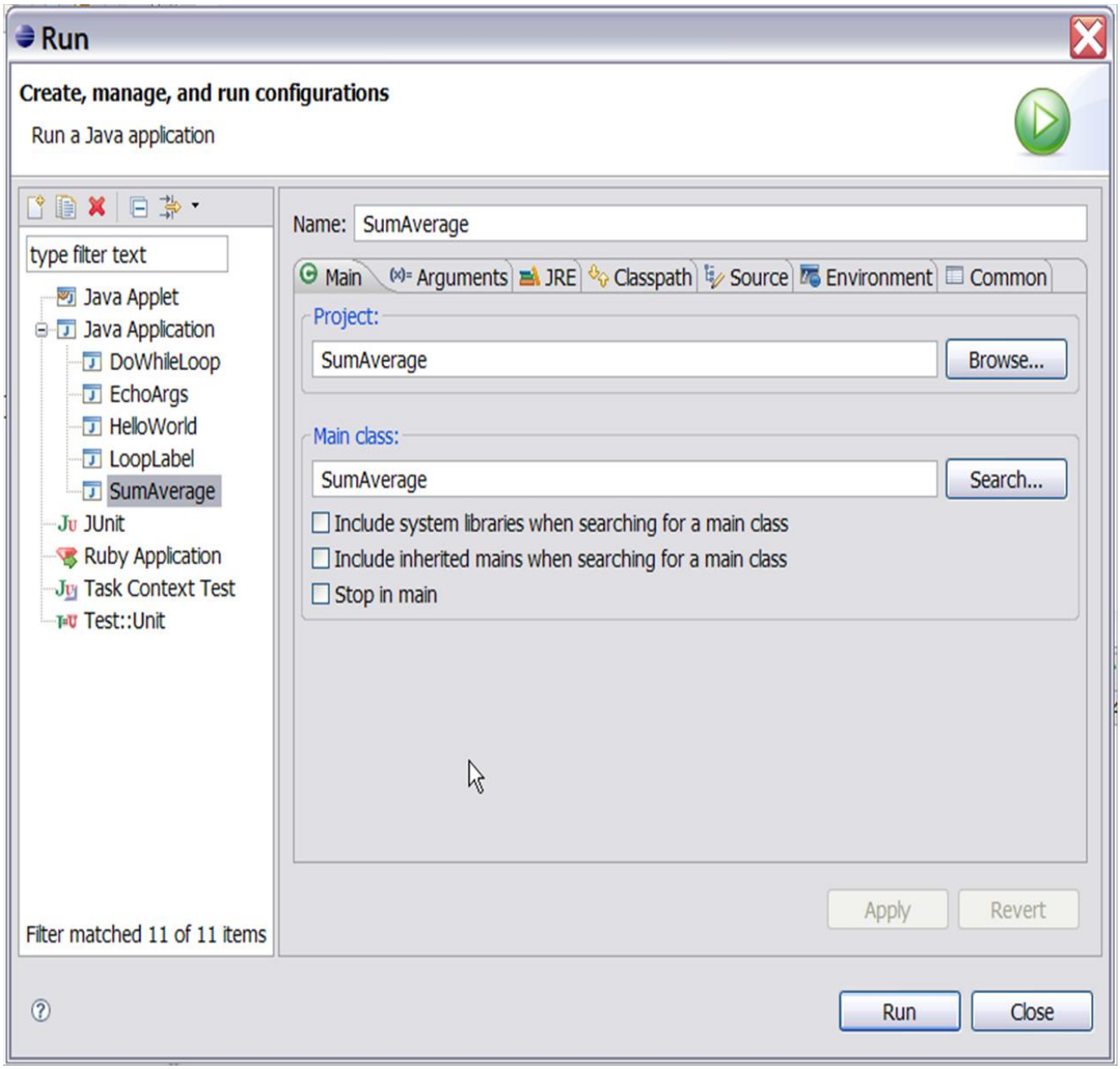
From the Java perspective, find the SumAverage folder and click on SumAverage.java file.

```
// TO DO:
   // declare a integer of sum and complete the for loop

public class SumAverage {
      public static void main(String arguments[]) {
            int

            for (int i = 0; i  < arguments.; ) {
                  sum += Integer.parseInt();
            }
```

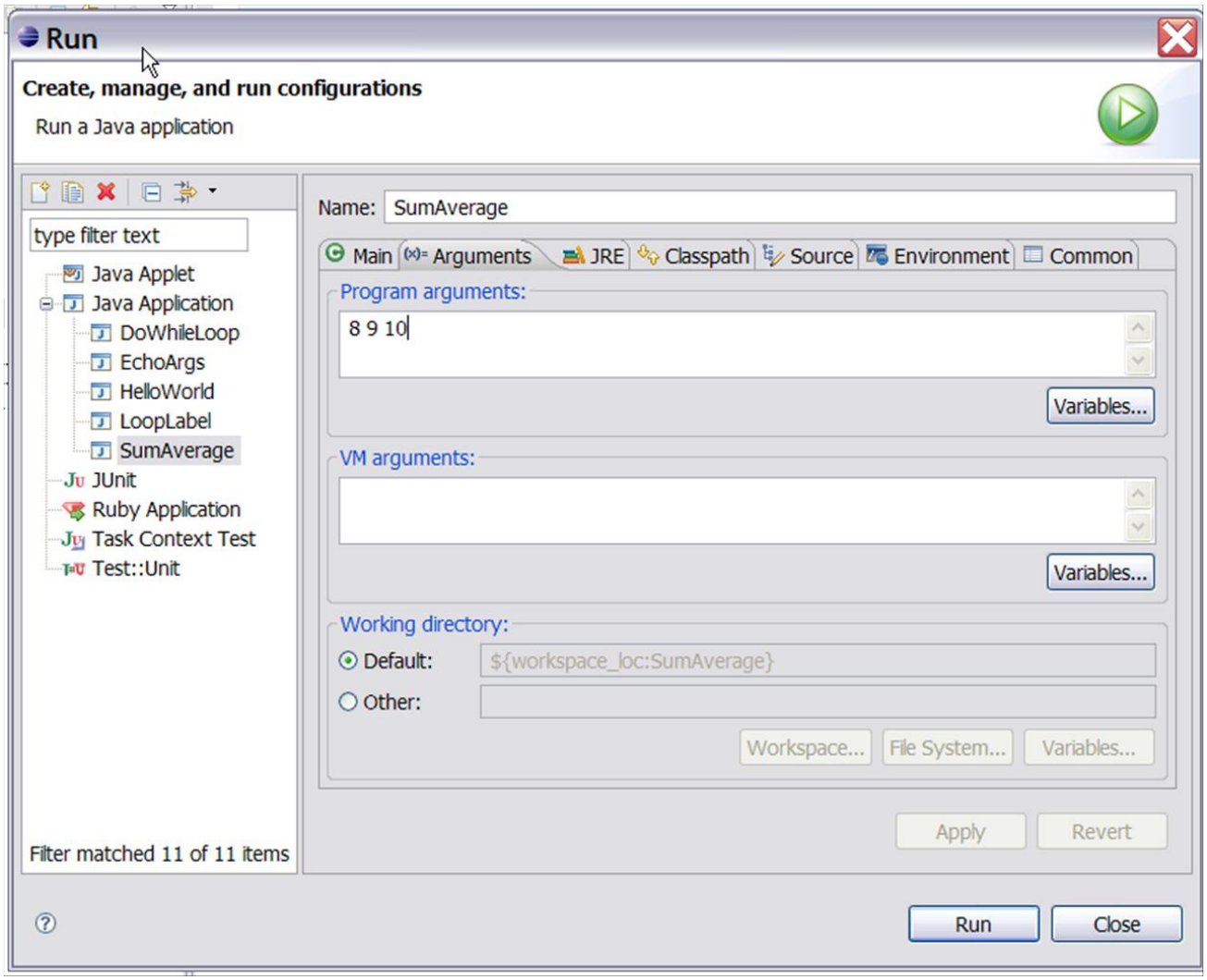
```
// TO DO:
   // Complete the print statement to read Sum is: …. between the ( ); and a
   // second print statement to read Average is: … between the ();

            System.out.println();
            System.out.println()
                (float)sum / arguments.length);
      }

}
```

To run the SumAverage.java application, click on the  Run menu icon and select / click on Open Run Diaglog, a Run wizard pop-up will appear…





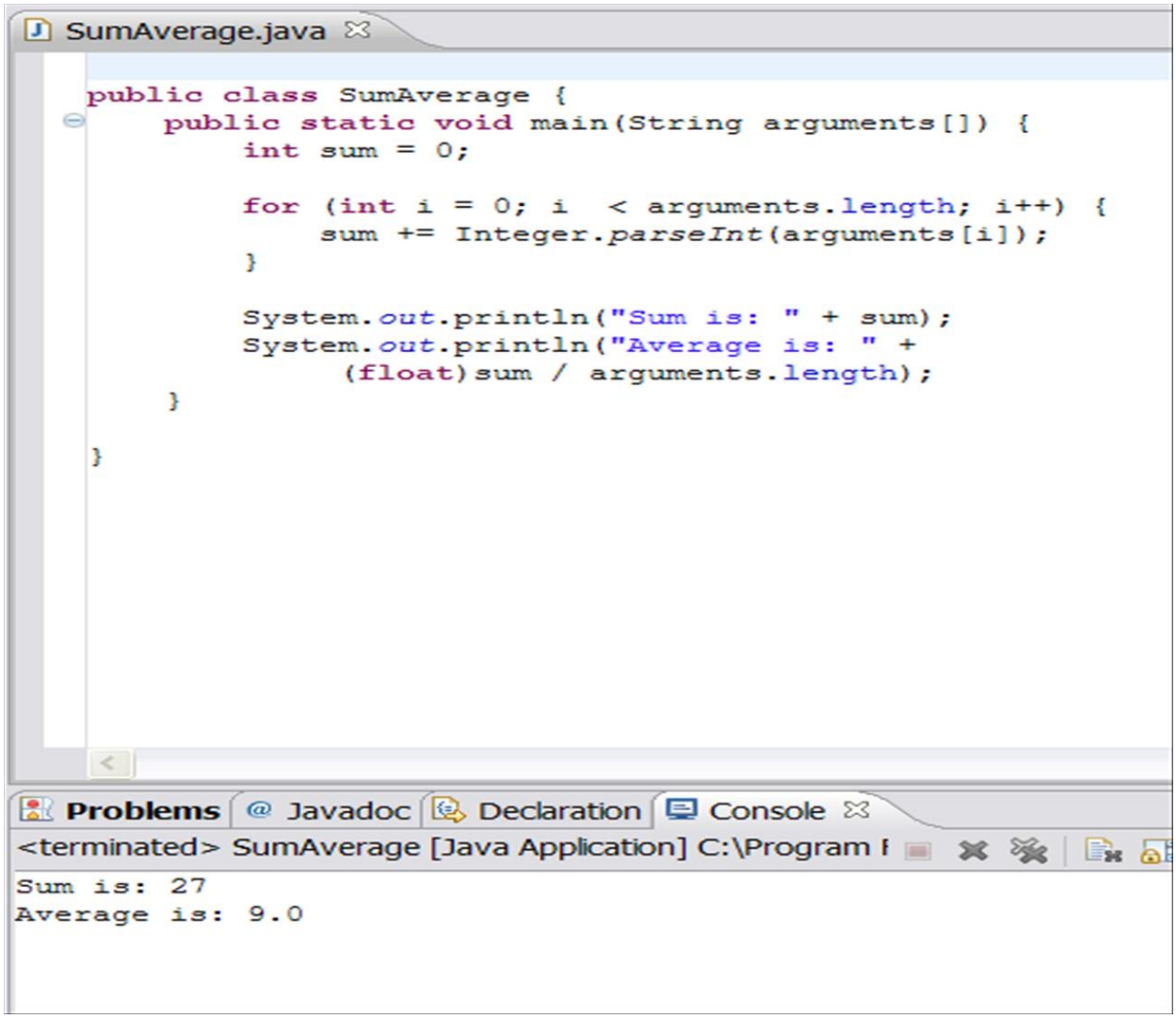
Click on Arguments tag to enter the arguments.



The sample arguments used was 8, 9 and 10 (see Program arguments). Remember to place a blank (hit the space bar once) between the arguments, followed a click on the Run button, and check the result set in the Console view.

Please note: you can use different set of arguments for this exercise.

Exercise #3 SumAverage.java Sample Result Set



Congratulations!

You have successfully completed the Lab session.

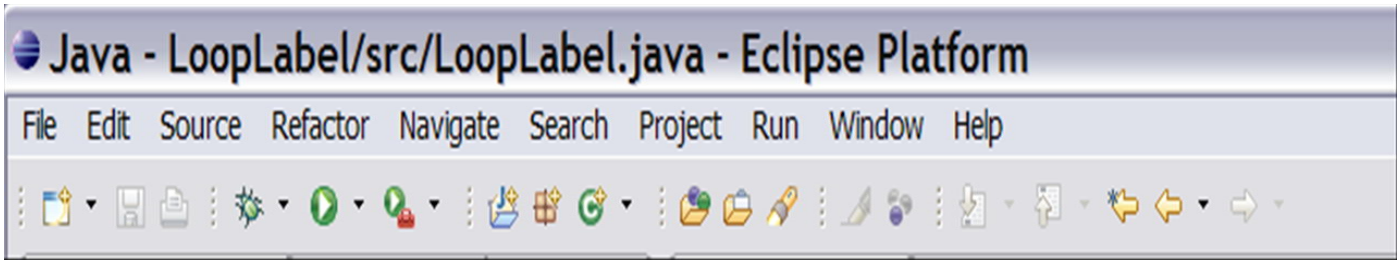
Thanks!

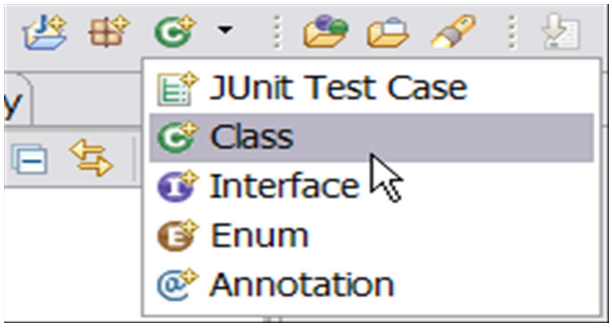
Optional Coding Exercise:

The following code sample allow you to repeat arguments that you enter from the command-line and run from the open dialog.

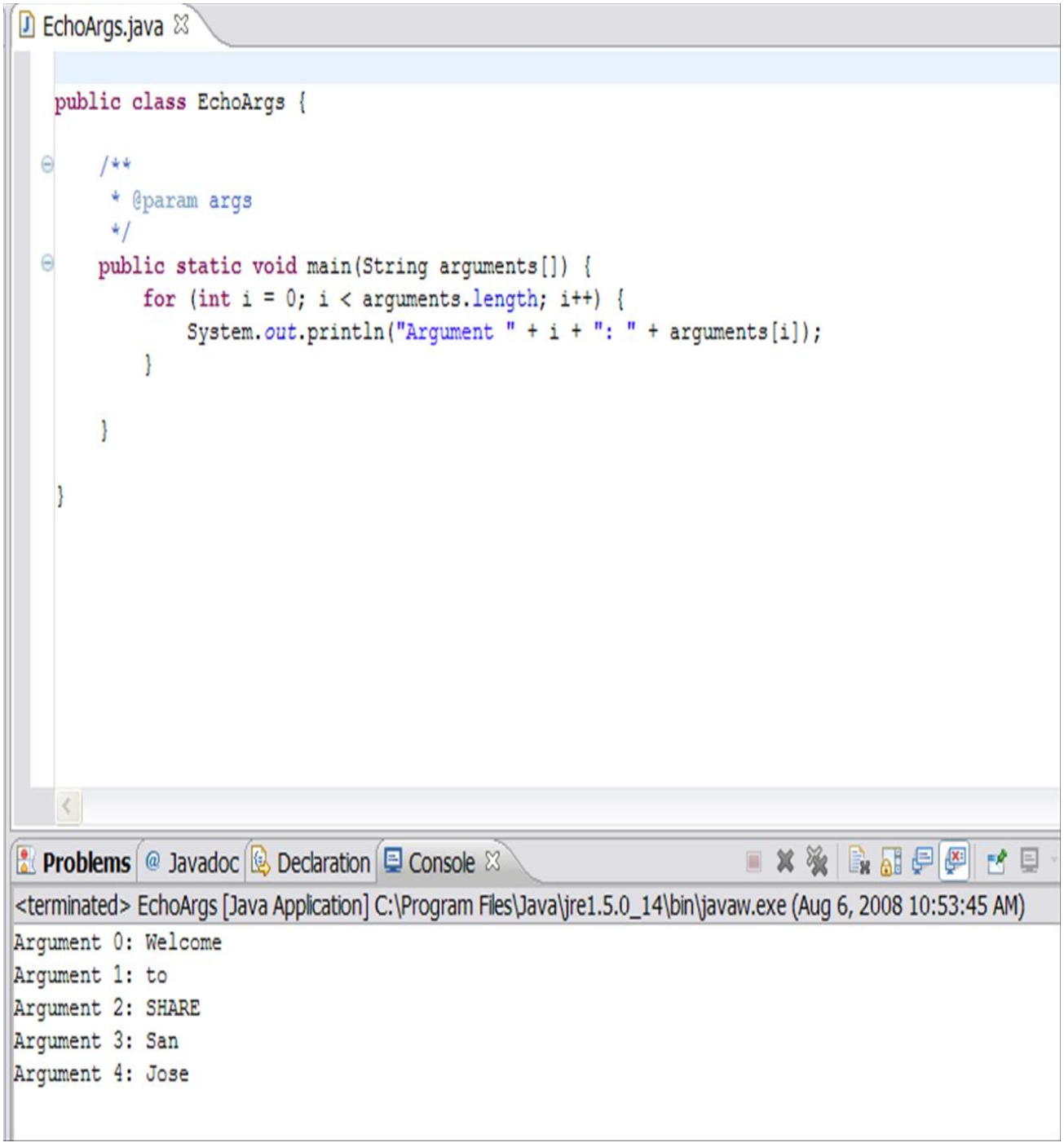
You need to create a new Java project, a new Java class… You can simply click on the J+ folder wizard to create a new Java Project And the New Java Class wizard to create the java class… as to the naming of the project and class… try CompareString or be creative.

Have fun!





Please see the sample result set below.



The crib sheet of Java syntax:

Java Comments:

- /* text */
  - Java supports the familiar C-style comments /* text */
  - The Java compiler ignores everything from /* to */
- /** documentation */
  - A documentation or “doc” comment, used by the javadoc tool
- // text
  - The compiler ignores everything to the end of the line

Variables and Data Type:

- Variable declaration
  - Name
    - Can begin with letter, dollar sign, or underscore
    - Followed by letters, underscores, dollar signs, or digits
    - Convention is Upper case
  - Type
    - Java’s compiler cares about type
    - Determines value and operations
- Two kinds of variables
  - Primitive
  - Object Reference

Primitive Types:

- Hold fundamental values (simple bit patterns)
  - Numeric data types
    - Integers
      - *8-bit byte*
      - *16-bit short*
      - *32-bit int*
      - *64-bit long*
    - Floating point numbers
      - *Real numeric types are 32-bit float and 64-bit double*
  - Booleans
    - “TRUE”, “FALSE”, “YES”, “NO” or similar constructs
  - Characters
    - Char myChar = ‘A’;

Type	Bit Depth	Value Range
Boolean	Varies	True or False
char	16 bits	0 to 65535
byte	8 bits	-128 to 127
short	16 bits	-32768 to 32768
int	32 bits	-2147483648 to 2147483647
long	64 bits	-huge to huge
float	32 bits	varies
double	64 bits	varies

Reference Types:

- Anything that is not primitive
  - Objects such as
    - Strings
    - Arrays
    - Classes
    - Interfaces

Operators – Arithmetic

Operator	Use	Description
+	op1 + op2	adds op1 and op2
-	op1 - op2	subtracts op2 from op1
*	op1 * op2	multiplies op1 by op2
/	op1 / op2	Divides op1 by op2
	op1 % op2	Computes remainder of dividing op1 by op2

Operators – Increment / Decrement

Operator	Use	Description
++	op++	Increments op by 1; evaluates to the value of op before it was incremented
++	++op	Increments op by 1; evaluates to the value of op after it was incremented
--	op--	Decrements op by 1; evaluates to the value of op before it was decremented
--	--op	Decrements op by 1; evaluates to the value of op after it was decremented

Operators – Relational

Operator	Use	Return true if
>	op1 > op 2	op1 is greater than op2
>=	op1 >= op2	op1 is greater than or equal to op2
<	op1 < op2	op1 is less than op2
<=	op1 <= op2	op1 is less than or equal to op2
==	op1 == op2	op1 and op2 are equal
!=	op1 != op2	op1 and op2 are not equal

Operators – Conditional

Operator	Use	Returns true if
&&	op1 && op 2	op1 and op2 are both true, conditionally evaluates op2
||	op1 || op2	either op1 or op2 is true, conditionally evaluates op2
!	! op	op is false
&	op1 & op2	op1 and op2 are both true, always evaluates op1 and op2
|	op1 | op2	either op1 or op2 is true, always evaluates op1 and op2
^	op1 ^ op2	If op1 and op2 are different – then if one of the other of the operands is true but not both

Operators – Assignment

Operator	Use	Equivalent to
=	op1 = op 2	assign op1 to the value in op2
+=	op1 -= op2	op1 = op1 + op2
-=	op1 < op2	op1 = op1 - op2
*=	op1 *= op2	op1 = op1 * op2
/=	op1 / op2	op1 = op1 / op2
%=	op1 %= op2	op1 = op1 % op2
&=	op1 &= op2	op1 = op1 & op2
|=	op1 |= op2	op1 = op1 | op2