Lab walk-through #1

Topic **Eclipse and Java Setup**

1. Lab Objectives

The objective of this lab is to understand the benefits of an IDE (integrated development environment) for developing a Java project. After completing this lab, you will be able to:

- Define a workspace and get started with Eclipse
- Create a new Java project
- Create a new Java class
- Import external packages
- Navigate in Java code
- Debug Java Code

2. Lab Setup

Eclipse Luna is already installed on your machine under Programs group.

Note: YOU NEED TO HAVE YOUR OWN STORAGE DEVICE (E.G., A USB KEY) IN ORDER TO CREATE THE ECLIPSE WORKSPACE.

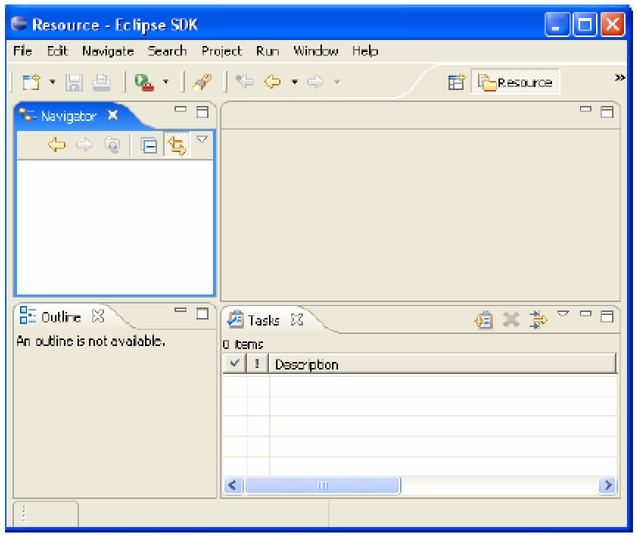
IT IS YOUR RESPONSIBILITY TO KEEP YOUR STORAGE DEVICE SAFE (WITH NECESSARY BACKUPS) FOR FUTURE USE OF THE PROJECTS YOU ARE CREATING IN THE LABS.

3. Exercise

In this lab, you will explore some basic activities in Eclipse as an IDE (integrated development environment). You will create a new project in Eclipse, create a new Java project and package, and add a simple class to the project. Then, you will learn how to import external packages, navigate inside Java code, and debug existing code.

Task 1: Run and Choose Eclipse workspace

1- Start Eclipse from the Programs group.
2- Browse to a folder on your USB key when prompted for a workspace.
3- Once Eclipse is running, you should see the following:


Resource - Eclipse SDK
File Edit Navigate Search Project Run Window Help
Resource
Navigator
Outline
An outline is not available.
Tasks
0 items
Description

Task 2: First Eclipse Java Project

1- To create a project in Eclipse, select File -> New -> Project. In the following window select Java Project -> Next -> Enter.
2- Project Name (FirstEclipseProject) -> Select Create Separate Source and Output Folders -> Finish.
3- Once completed, Eclipse will ask you to switch to the Java Perspective, select Yes and then the Java perspective will open.
4- Note different Views and Perspectives under the Window menu by selecting Open Perspective or Show View.
5- Expand FirstEclipseProject under PackageExplorer view to observe the empty project structure.

Task 3: First Eclipse Java Class

1- To create a Java class in Eclipse, select File -> New -> Class.
2- Enter Package Name (cas.lab1.firstEclipsePackage).
3- Enter Class Name (FirstEclipseClass).
4- Check public static void main and Uncheck Inherited abstract methods.
5- Finish.
6- Observe the source code file opened in the main view.
7- Under the main() method, enter the following:

```
System.out.println("My First Eclipse Class is Running");
```

8- Save the file through File -> Save or using CTRL+S.
9- Execute the main class by selecting Run -> Run… -> Java Application -> New -> Run, and see the results in the Console view.

Task 4: Importing External Packages

1- Under the main method of FirstEclipseClass, create a new method public static void printVector(Vector input).
2- Save the file and observe term Vector underlined in red and under Problems listed as "cannot be resolved".
3- Select Source -> Organize Imports or select CTRL+SHIFT+O.
4- Save the file and note that Vector was resolved and that a new package java.util.Vector was imported.
5- Alternative: instead of Source -> Organize Imports, highlight term Vector and select Source -> Add Import.

Task 5: Navigating External Types

1- Under the printVector method, type "for" and press CTRL+Space to open type navigator.
2- Select iterate over array and then replace "array.length" with "input." to get a listing of type members for Vector.
3- Select size() and in the body of the for loop, type the following and observe the type navigator open after each dot is typed

```
System.out.println(input.get(i));
```

Task 6: Start the Debugging

1- Within the main method, remove the print statement, create a new Vector of four strings: "A", "B", "C", and "D", and print the strings using the printVector method using the following code:

```
Vector input = new Vector();
input.add("A"); input.add("B"); input.add("C"); input.add("D");
printVector(input);
```

2- To the left of the print statement in printVector, click on the blue area to insert a break point.
3- Select Debug As -> Java Application and observe the Debug Perspective open (select Yes to switch over).
4- Once the debugging perspective opens, the line with the print statement should be highlighted in green.
   a. To step into the called method, press F5.
   b. To step over to the next line, pressF6.
   c. To step out to the caller, press F7.
   d. To resume regular execution, press F8.
5- In the debugging perspective in the top right corner, observe Variables view. Using this view, you can view values of primitive data types and expand and view the members of complex data types.
6- Alternative: You can also view the values of primitive data types by moving your mouse over the variable name in the code.
7- To terminate, click on Terminate (red button) or select Run -> Terminate; do not forget to do this if the session does not close.

Task 7: Refactoring

1- Right click on the definition of the printVector method and select Refactor -> Change Method Signature.
2- Change Method Name to printVectorElements, add a new parameter called "count" of type int with value null, and click Preview.
3- Observe the conflict with the call in main method.
4- Select Back -> Enter a Default Value of 3 -> Preview (all fine) -> OK.
5- Within the printVectorElements method, replace input.size() with count and then run.
6- Observe only A, B, and C are now printed.

Task 8: Formatting

To make code more readable and easier to navigate, it may be necessary to reformat it.

1- Eclipse provides automatic reformatting of files and projects using Source -> Format or CTRL+SHIFT+F command.
2- To reformat now, right click on the FirstEclipseProject in the Package Explorer and click Format.
3- Note the source code format changes.

Task 9: Navigating Type Hierarchies

1- To get more information about a specific type, you may select it and open it in different views.
2- For example, highlight Vector and select Navigate -> Open Type Hierarchy.
3- If the code is available, you can also select Navigate -> Open Declaration; this command also works for attributes.
4- You can also see all call declarations using Navigate -> Open Call Hierarchy; this command also works for methods.

Task 10: Source Code Browsing

1- As you are moving from file to file using links and references, it may be necessary to go back and forth and use bookmarks.
2- To go back or forward, select Navigate -> Back or Navigate -> Forward, or ALT+Left or ALT+Right.
3- You may also use left or right arrows above the source code viewer, with browsing history as drop-down menus.
4- To go back to the last edit location, select Navigate -> Last Edit Location, or CTRL+Q.

Task 11: Export a Project

1. In Eclipse, right-click on the name of the project, select Export->General->Archive File.
2. Ensure that just your project has a check-mark beside it, and select a path and filename to export the project to (e.g., FirstEclipseProject). Ensure that your export project has a file extension of '.zip'.

Task 12: Import a Project

1. In Eclipse, in the Package Explorer view Right click in the white-space and select Import -> General -> Existing Projects into Workspace.
2. Select 'archive file' and then select the project you want to import (e.g., FirstEclipseProject.zip) that is available in the folder you have exported the project.
3. Import the project. You should see the project ' FirstEclipseProject' in the Package Explorer view. NOTE that a project cannot be imported if another project with the SAME name already exist in the workspace.

Task 13: Provide Input to your program from the command-line.

An important aspect of a Java program model is to support a simple model for Java programs to interact with the outside world. Typically, we want to provide *input* to our programs: data that they can process to produce a result. The simplest way to provide input data is illustrated in UseArgument.java below.

```
/*****************************************************************
 *  Execution:    java UseArgument yourname
 *  Prints "Hi, Bob. How are you?" where "Bob" is replaced by the
 *  command-line argument.
 *  % java UseArgument Bob
 *  Hi, Bob. How are you?
 *  % java UseArgument Alice
 *  Hi, Alice. How are you?
 *****************************************************************/
public class UseArgument {

    public static void main(String[] args) {
        System.out.print("Hi, ");
        System.out.print(args[0]);
        System.out.println(". How are you?");
    }

}
```

Whenever this program is executed, it reads the command-line argument that you type after the program name and prints it back out to the terminal as part of the message.

In Eclipse the command-line arguments can be added in the **Program arguments** box on the **Run Configuration** menu.

Run this Java application and print the following:

```
Hi, Alice. How are you?
Hi, Bob. How are you?
Hi, [YOUR NAME]. How are you?
```

Task 14: Reading from a .txt File

There are also libraries built upon extensive capabilities that are available in Java for more efficient communication with the outside world. In this task and the following you will learn how to use Java libraries to read a txt file as an input file and how to write the results of an execution into a text file.

1. We will read a list of integers from a text file and determine whether each number is even or odd.
2. Add a new text file to the project, by right-clicking the project → New → File.
3. Change the filename to input.txt.
4. Open input.txt and type the numbers 1 to 10 **on ten separate lines**, then save the file.
5. Under the printVectorElement method of FirstEclipseClass.java, create a new method called readFromFile (it should be public static Vector<String>).
6. Add the following code to the readFromFile method:

```
Vector<String> result = new Vector<String>();
        try{
                File f = new File("input.txt");
                Scanner s = new Scanner(f);
                while(s.hasNextLine()) {
                        int i = s.nextInt();
                        if(i % 2 == 0)
                                result.add("Even");
                        else
                                result.add("Odd");
                        System.out.println(i);
                        }
                s.close();
        }
        catch(FileNotFoundException e){
                e.printStackTrace();
        }
        return result;
```

7. Add the following imports to the top of the class:

```
import java.io.*;
        import java.util.Scanner;
```

8. Add a call to the readFromFile method that you just created in your main method by typing:

```
Vector<String> results = readFromFile();
```

9. Run the application and note the output to your Console screen of the contents of the file.

Task 15: Writing to a .txt File

1. In this task, we will write the output of the previous question (is each integer in the input file odd or even?) into an output file.
2. Create a new method called writeToFile() underneath the readToFile() method (this method should be public static void with a parameter Vector<String> results,

e.g. public static void writeToFile(Vector<String> results)).

3. Add the following code to the method:

```
try{
        FileWriter f = new FileWriter("output.txt");
        for(String result: results )
                f.write(result + "\n");
        f.close();
}
catch(Exception e){
        e.printStackTrace();
}
```

4. Call the newly created method by adding the following line to your main method:

```
writeToFile(results);
```

5. Run the Application and refresh the project directory by right-clicking the project and selecting Refresh, alternatively, you can click the Project and hit F5.
6. Open the output.txt file to see the content of the file.

Task 16: Submit your work

Once all tasks are completed, you should submit your Eclipse project. This must be done before leaving the lab. Follow the instructions below for submission:

- Include a .txt file named lastname_initials.txt in the root of the project containing on separate lines: Full name, student number, any design decisions/assumptions you feel need explanation or attention.
- After checking the accuracy and completeness of your project, save everything and right-click on the name of the project, select Export->General->Archive File.
- Ensure that just your project has a check-mark beside it, and select a path to export the project to. The filename of the zipped project must follow this format: *macID_Lab1.zip.* Check the option to save the file in .zip format. Click Finish to complete the export.
- Go to Avenue and upload your zipped project to 'Lab Walk-through 1 – Lab Section X)
- IMPORTANT: You MUST export the FULL Eclipse project. Individual files (e.g. java/class files) will NOT be accepted as a valid submission.

4. Practice Problems

Try to write your first Java application using Eclipse implementing the problems in Ex. 1.1.15, and 1.1.36 from your 2C03 textbook: *Robert Sedgewick and Kevin Wayne.* If you found it hard writing the Java code for these problems, make sure you will attend the next three Java tutorials as the basics of Java programming that is required for implementing algorithms will be taught in the next three labs.