OBJECT ORIENTED PROGRAMMING

PCC-CS593

Laboratory Instruction Manual

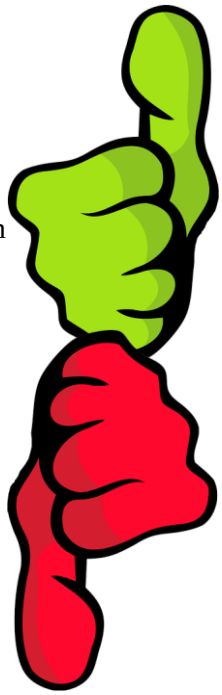
Last Revised
Jul 2023

Compiled by
Utpal Das

Dept. of CSE
Techno Main Salt Lake
(formerly Known as Techno India Salt Lake)

GENERAL INSTRUCTION FOR LABORATORY CLASSES

DO'S



- ☑ Without prior permission of teacher students should not enter into the laboratory.
- ☑ Students should come with proper uniform.
- ☑ Students should maintain silence inside the laboratory.
- ☑ After completing the laboratory exercise, make sure to shut down the system properly.

DONT'S

- ☒ Students using the computers in an improper way.
- ☒ Students scribbling on the desk and mishandling the chairs.
- ☒ Students using mobile phones inside the laboratory.
- ☒ Students making noise inside the laboratory.

INSTRUCTION FOR LABORATORY TEACHERS

1. Code submission is needed within the scheduled lab session.
2. Promptness of submission should be encouraged by way of marking and evaluation that will benefit the sincere students.

OBJECTIVES OF THE LAB

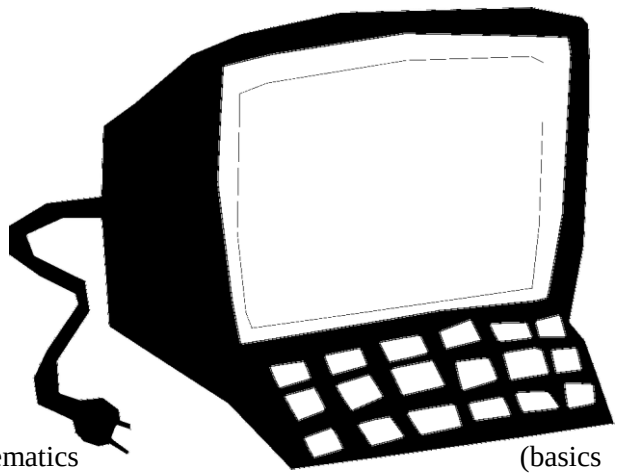
- ➢ To make the students learn a programming language.
- ➢ To teach the students to write programs in Java to solve a particular assignment.
- ➢ Students learn the concepts like looping, functions, pointers, structures etc.

Primary goal of this course is to acquaint the students with the Java programming language by developing code in the lab class. Students also learn how to test the program for different test cases. To improvise on their testing knowledge, students need to assess the assignment of other students.

REQUIREMENT AND SPECIFICATION

HARDWARE REQUIREMENTS

PROCESSOR: Dual core or HIGHER
RAM: 4GB or HIGHER
HDD: 160GB



SOFTWARE REQUIREMENTS

OPERATING SYSTEM: Ubuntu 12.04 LTS or HIGHER
JDK 1.8 or HIGHER

PREREQUISITE

Computer fundamentals, Logic building skills, Mathematics (basics like prime number, factorial, fibonacci series etc.), etc.

UNIVERSITY PRACTICAL EXAMINATION Breakups

INTERNAL ASSESSMENTS: PCA1 and PCA2 (40 marks)

Weekly assignments: 10 marks
Weekly reports: 5 marks
Exam Day Program: 10 marks
Exam Day report: 5 marks
Quiz and Viva voce: 10 marks

Total = 40 marks

UNIVERSITY EXAMINATION (60 marks):

Execution: 30 marks
Design & Documentation: 10 marks
Viva voce: 20 marks

Total = 60 marks

OBJECT ORIENTED PROGRAMMING LAB (CS- 594D) SYLLABUS

Contacts: 4P

Credits: 2

1. Assignments on class, constructor, overloading, inheritance, overriding
2. Assignments on wrapper class, arrays
3. Assignments on developing interfaces: multiple inheritance, extending interfaces
4. Assignments on creating and accessing packages
5. Assignments on multithreaded programming
6. Assignments on GUI programming

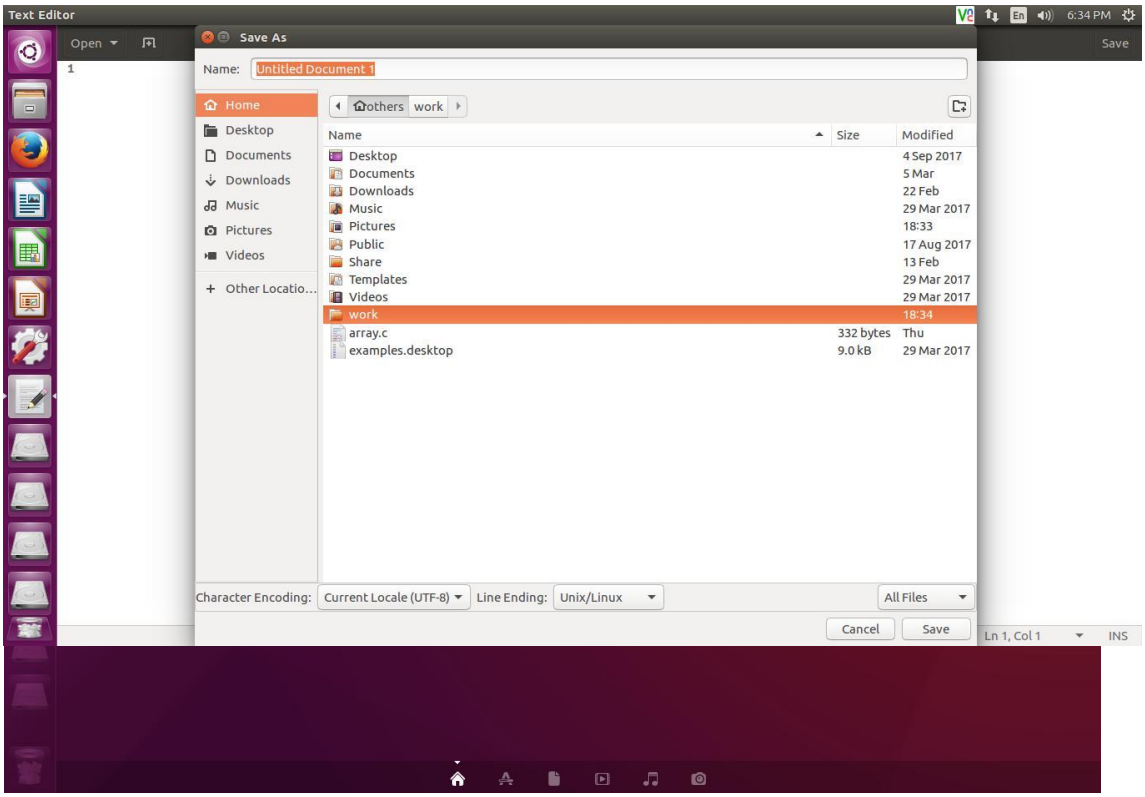
(Source: http://makautexam.net/aicte_details/aicteugdetails.html)

Laboratory Experiments

The following list shows sample experiments. Actual experiments can be found under the learning portal (etcm.ticollege.org/cms1):

Assignment No.	Assignment	Spoken Tutorial Reference	Week No.
1	Write a Java method fact(int n) for computing factorial for small n values within 20. Output should be displayed in whole integer format.	Loop & nested loop Statements	
2	Write a Java program to input couple of integers until 0 is entered, and display the second maximum value among all entered numbers.	Decision statement	
3	Write a Java program to take integer inputs for base and exponent, calculate power value without using Java Math function.	Decision statement	
4	Write a Java program to input a number, and display the most economic data type among byte, short, int and long for accommodating the input number.	Switch case statement	
5	Write a Java program to input couple of numbers, calculate their median and display. You may use hasNextInt() or hasNextDouble() method (as applicable) to check for input termination.	Function statement	
6	Calculate the area of various shape objects using dynamic time polymorphism.	Polymorphism	
7	Calculate the area of various shape objects and sort them in ascending order of their areas using polymorphism.	Polymorphism	
8	Write a Java program to evaluate a postfix expression. You may use stack-based solution with array implementation.	Array	
9	Write a Java program to input a number, Calculate the sum of Square using interface.	Interfaces	

10	Write a Java program to input n and add n number of integers. In case of InputMismatchException, display "ERROR".	Interfaces	

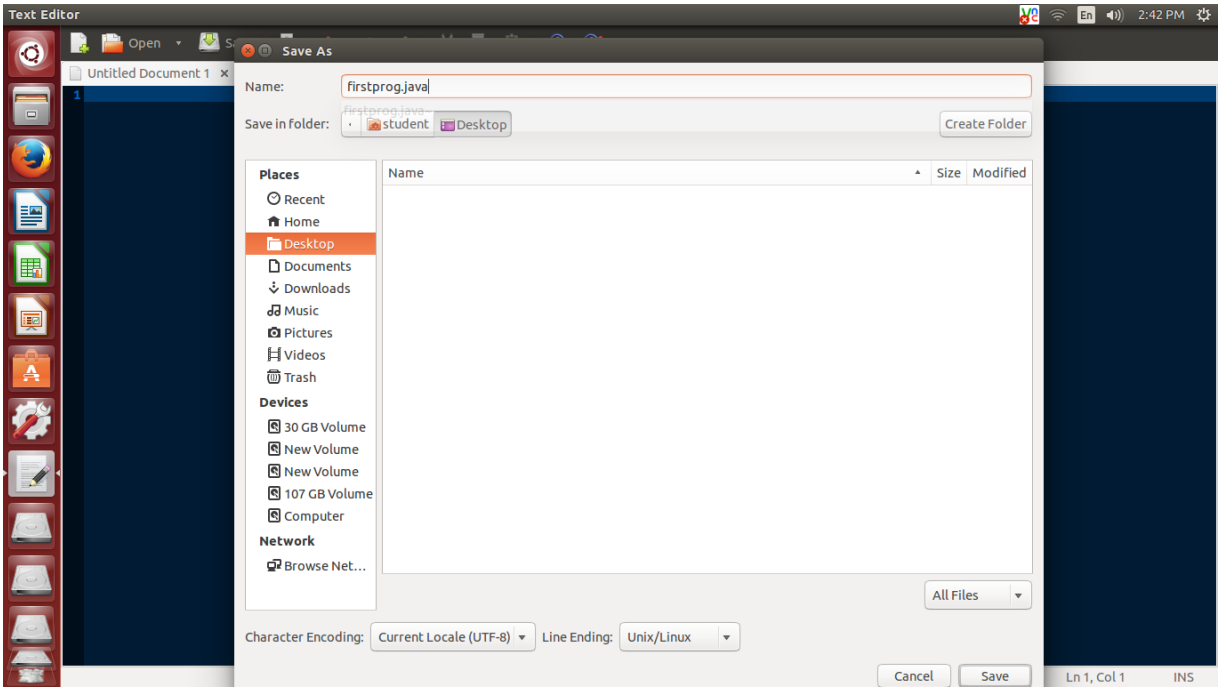


Writing JAVA Program:

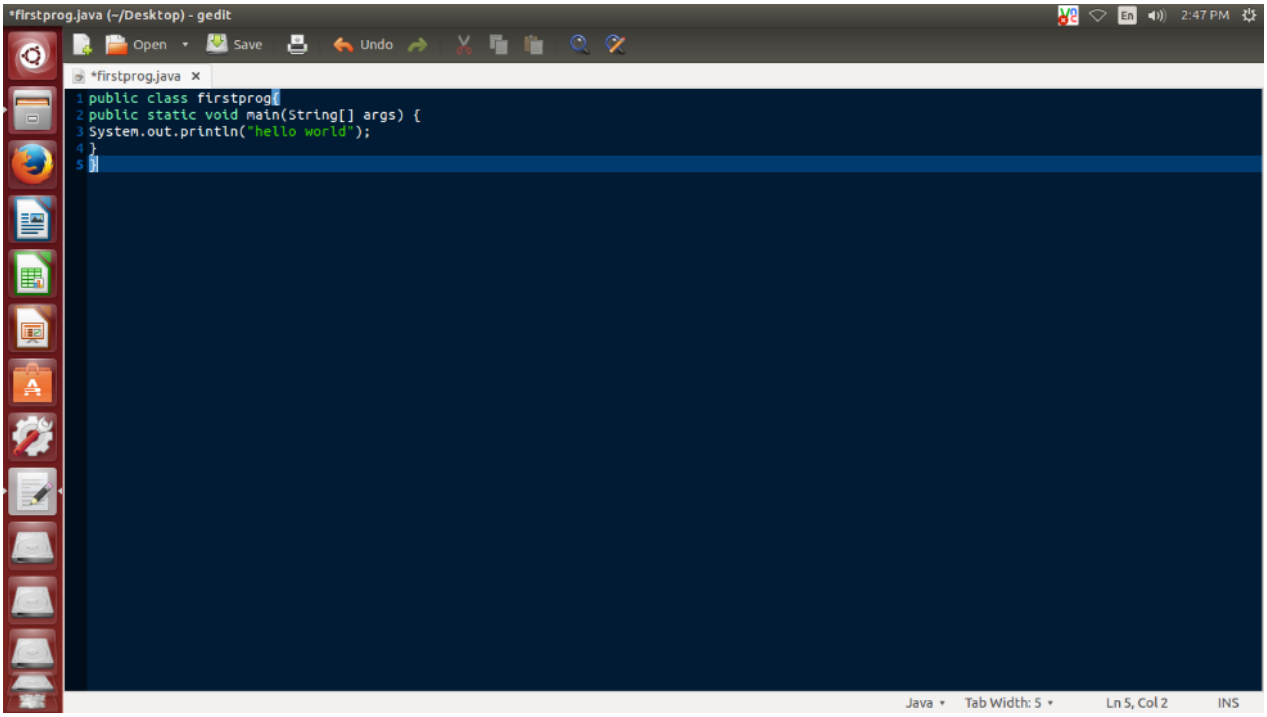
Step 1: Search for **gedit** editor in the search bar. Once **gedit** is shown in the results click on it.

The gedit window opens as shown below.

Step 2: Create a **work** folder inside Home location. Save the opened gedit file with a proper naming convention with an extension as *.java* for the filename inside **work** folder. Click on Save option to save the *.java* file.

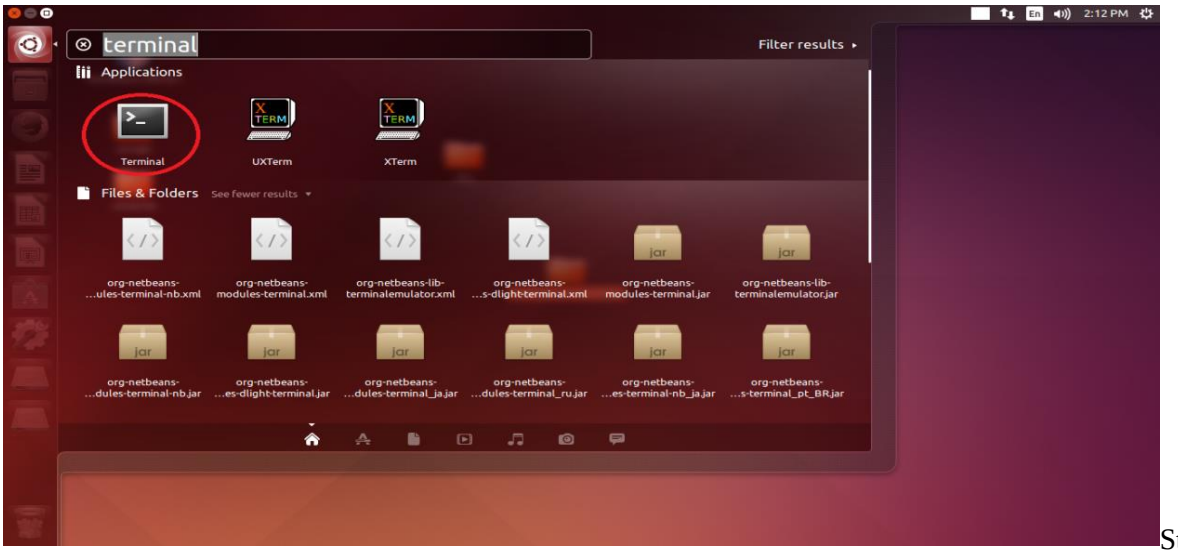


Click on Save option.



Step 2: Write the java code for *firstprog.java* file.

Step 3: Search for **terminal** in the search bar. Once terminal is shown in the results click on it.



Step 4: Th e terminal window opens.

pwd command is used to check the current directory location in terminal.

cd (change directory) command is used to change directory location in terminal. In this case, cd work changes the current directory path to work folder location(source file ie; *.java* file location).

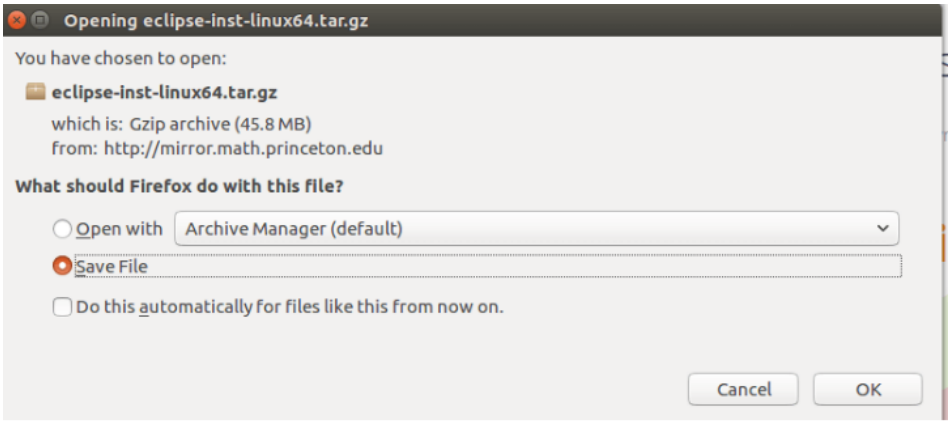
ls command is used to list the contents of any folder/directory. In this case, we can use this command to check whether the programmer is in the correct source file location, thus, assuring the source file is present in the folder location.

Step 5: Compile the Java program (source code ie; *firstProg.java*) to generate the executable code. Upon successful compilation an executable file is produced with the name **firstProg**. On failure we need to correct

the source code and recompile again till it compiles successfully. To execute the java program, follow the below steps.

HOW TO INSTALL ECLIPSE

Step 1: Install Java JDK8. Eclipse requires Java JDK prior installation. Download Eclipse Oxygen. Following link can be used.

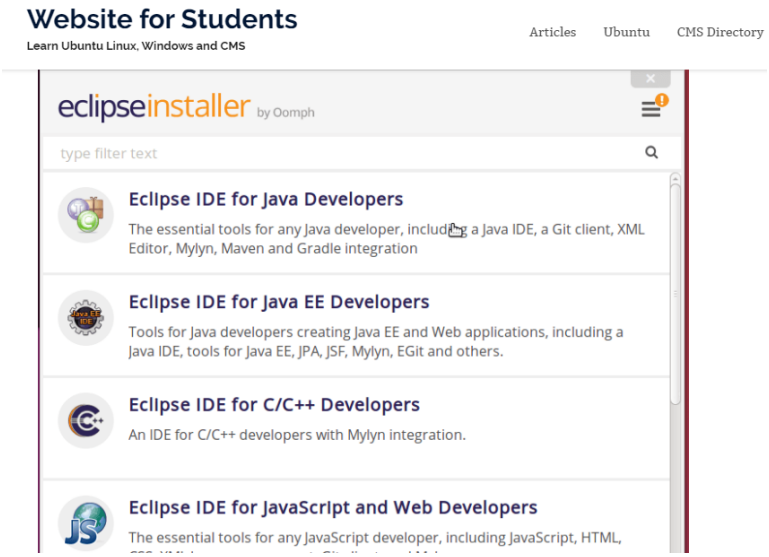


https://www.eclipse.org

Step 2: Install Eclipse IDE.

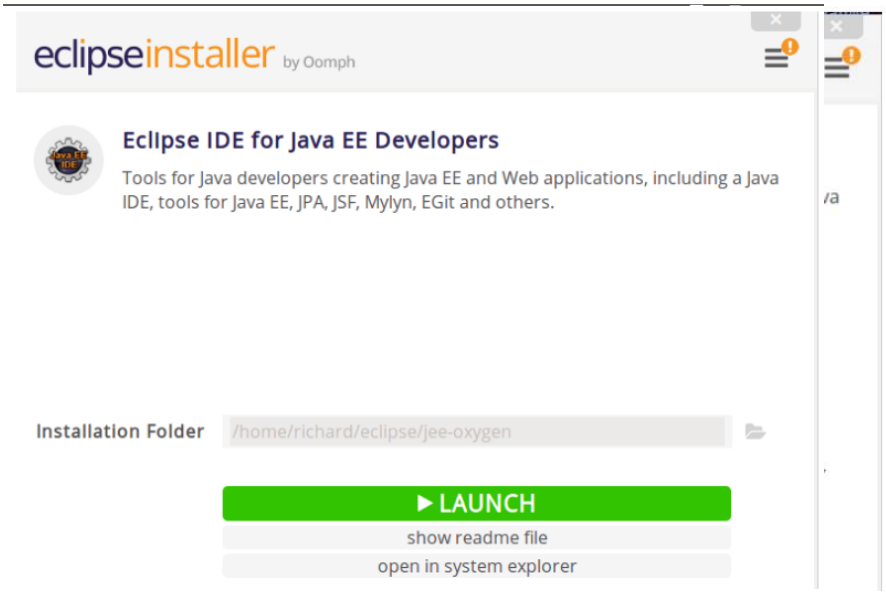
Use the commands below to extract the content in the ~/Downloads folder. The next line launches the installer.

tar xfz ~/Downloads/eclipse-inst-linux64.tar.gz
~/Downloads/eclipse-installer/eclipse-inst

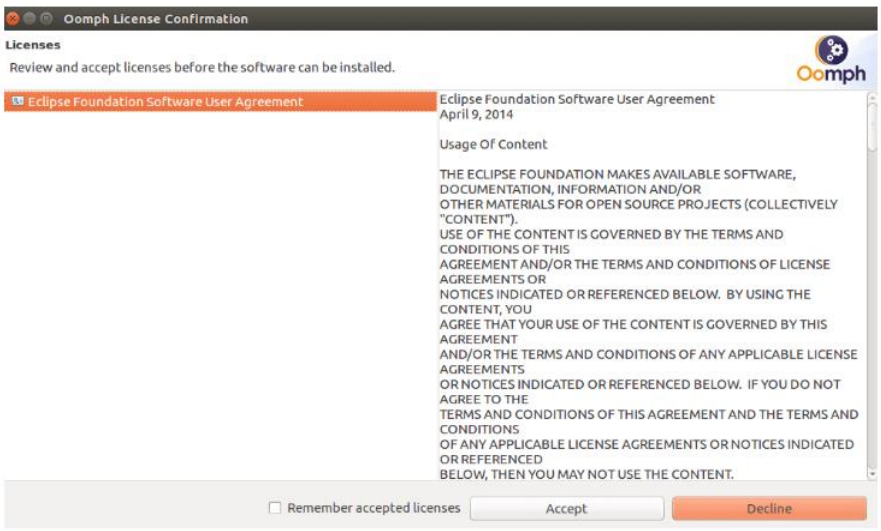


Select the package IDE you want to install and continue.

Use the onscreen instructions to complete the installer.. Accept the default installation directory and continue.



Next, accept the license terms and continue wait for Eclipse installer to download and install all the packages.



After downloading the installer should complete. All you have to do is launch the program.

Create Eclipse App Launcher.

DEBUGGING A JAVA PROGRAM WITH ECLIPSE

Step 1: To debug a open project in Eclipse, open the Debug Perspective.

Go to Window > Perspective > Open Perspective > Debug

This will open the debug window.

You'll see different tabs,

1. **Debug:** This tab informs about the running threads, classes and functions.

2. **Variables:** This tab will show all the variables and their values currently in use.
3. **Breakpoints:** This tab will list all the breakpoints set.
4. **Code Editor:** The current file being debugged will be shown here.
5. **Outline:** This will show the structure of the current method being debugged.
6. **Console:** This will show the output/errors of the code.

The positioning of the tabs might be different, but you'll find similar tabs. Here are some of the important controlls,

1 Double click on this red part to set a **breakpoint**.

2 **Start debugger:** Start the debugger.

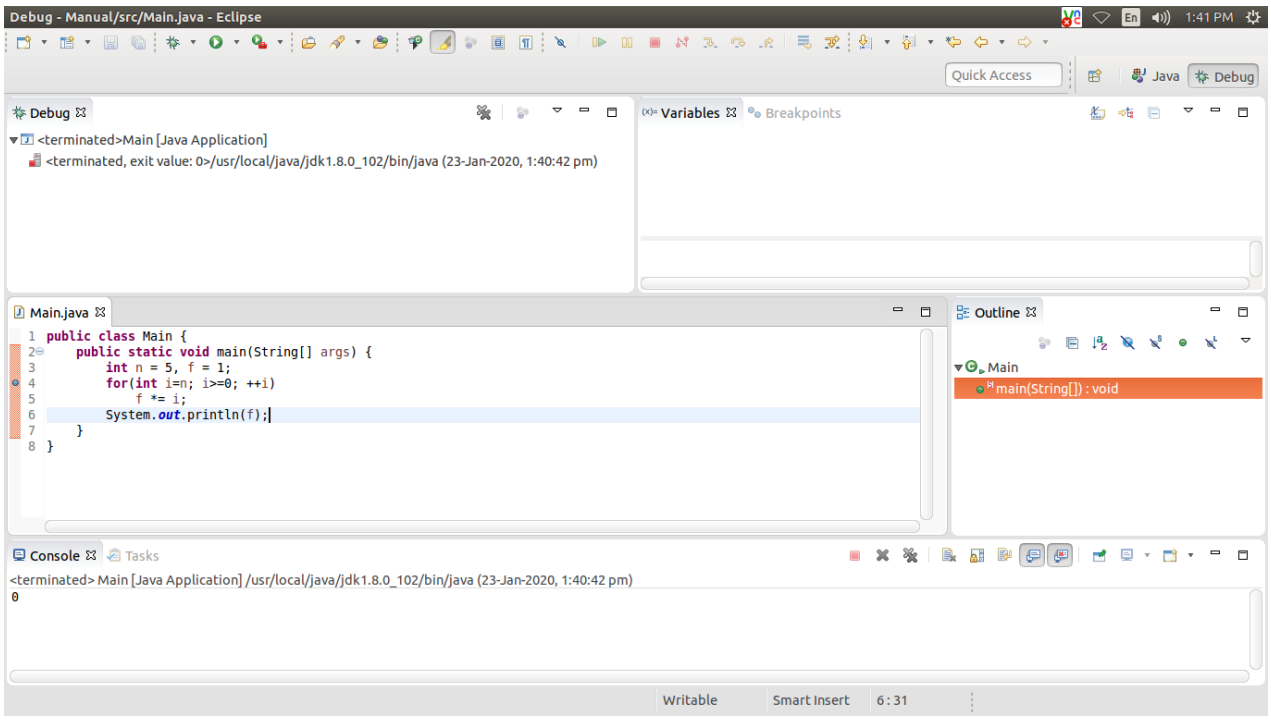
3 **Resume:** Resume the program until the next breakpoint is reached.

4 **Stop:** Stop the debugger and terminate the program.

5 **Step into:** Go to the next step in execution while jumping inside functions, if present.

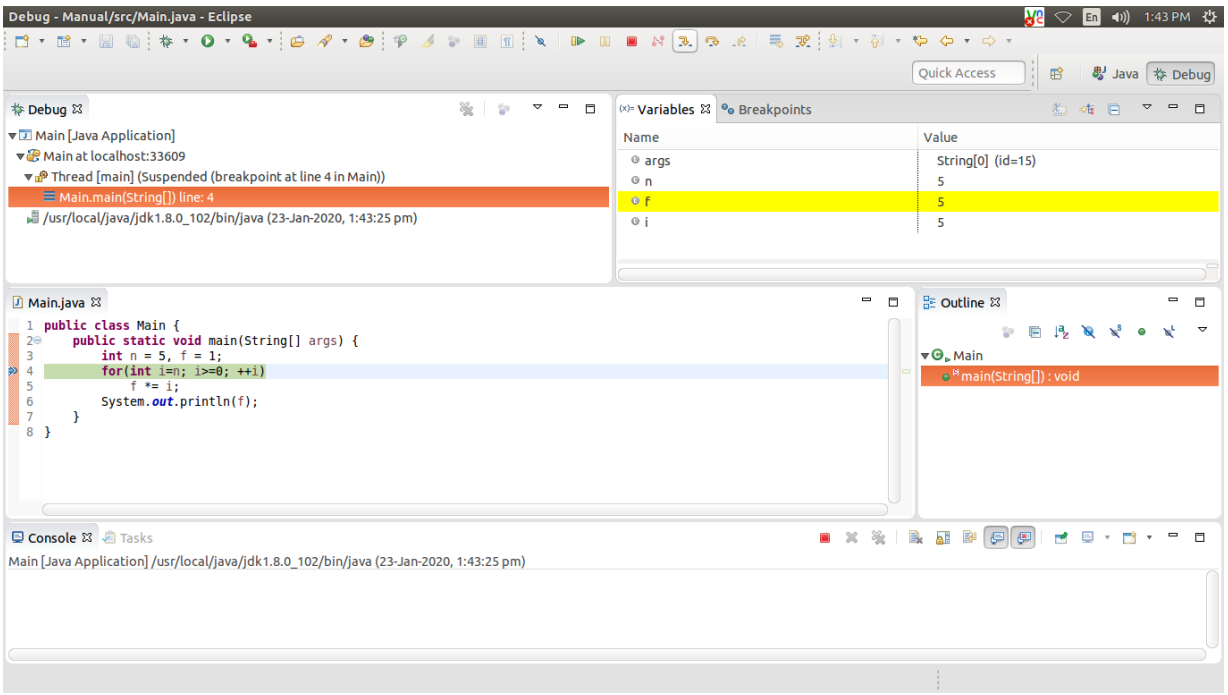
6 **Step over:** Go to the next step in execution without jumping inside functions.

7 **Step out:** Jump out of functions, if inside one.Set a breakpoint on the line where you think something might be going wrong. Double click on the red bar before the line numbers on the Editor tab to set a breakpoint. A breakpoint is a point where your program will stop after reaching. After



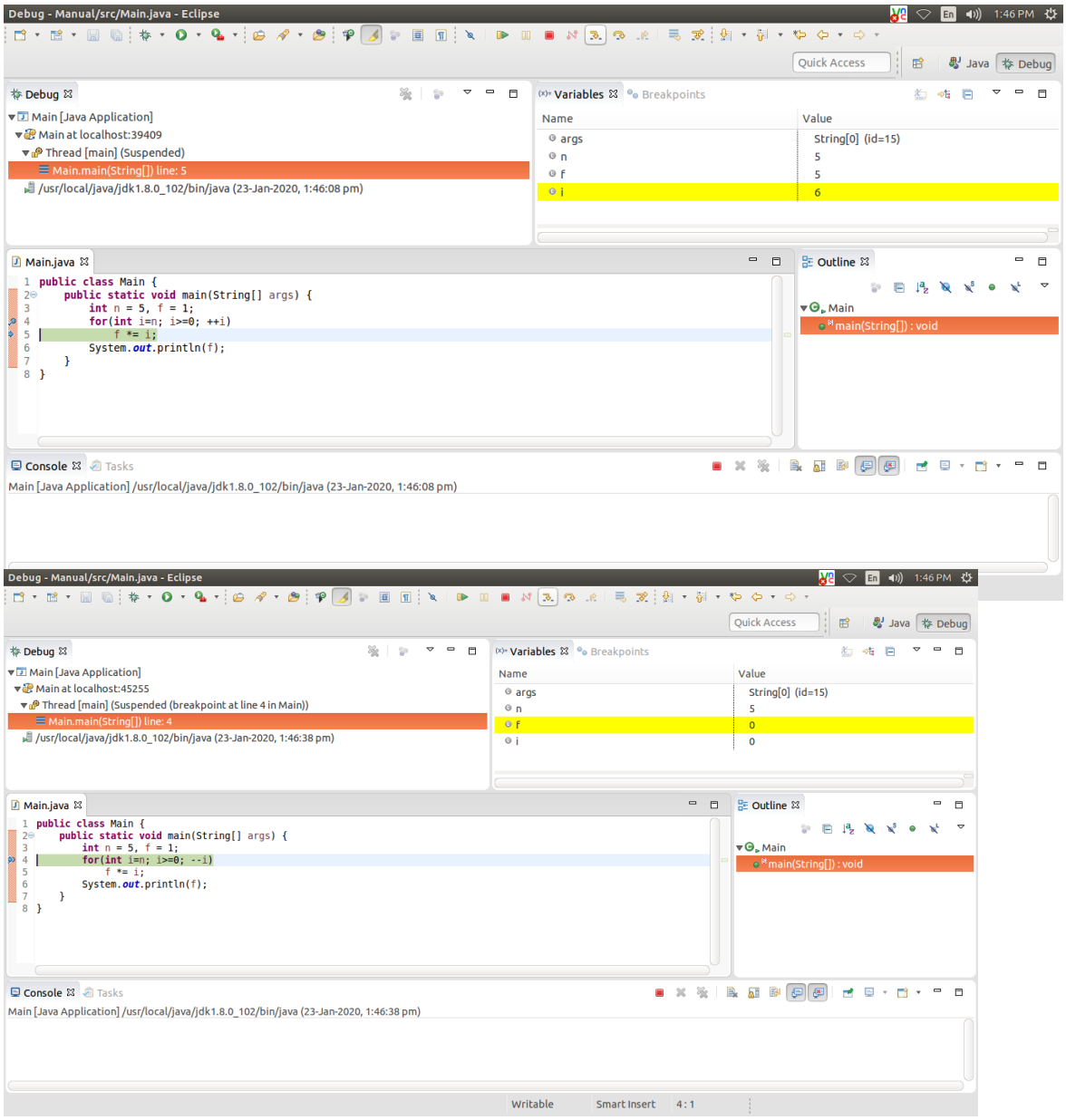
setting the breakpoint click on the *Start debugger* icon to start the debugger.

Step 2: When the first breakpoint is reached, the program will stop and the *Variables* tab will show the values of all the variables available currently. You can check these values to see if something is wrong, or if something doesn't appear the way you planned it'd.



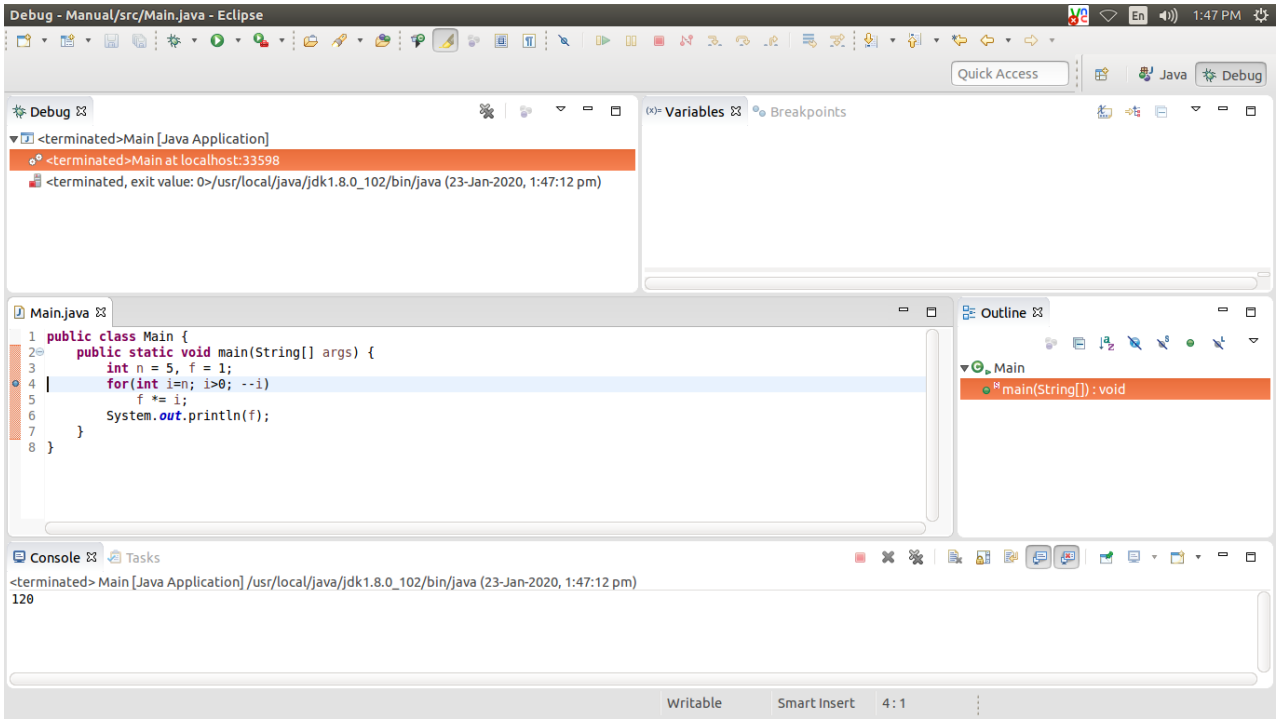
The currently running line and the currently changing variables will be highlighted. Click on the icons described in Step 1 to move through your program as you seem fit while noticing how the variables change.

1. We noticed that in this case the value of i has increased instead of decreasing. We'll change this



and check for any other errors.

2. As we proceed, we find the next error. i goes to 0 and thus f becomes 0 too.



Once all the errors were corrected, the program will complete and printt the correct result.

Lab Report:

Use a thick hard binding lab copy for writing your weekly reports and submitting on the next lab day. Typical components are:

a. **Problem statement:** Write the assignment description as given in the learning portal.

b. **Class Diagram:** Class diagram is a static diagram and it is used to model the static view of a system. Generally, UML diagrams are not directly mapped with any object-oriented programming languages but the class diagram is an exception. Class diagram clearly shows the mapping with object-oriented languages such as Java, C++, etc.

c. **Logic:** In logic part algorithm or flowchart can be written.

   An algorithm is unambiguous finite step-by-step procedure which is given input to produce output. Typically algorithms are used to solve problems such that for every input instance it produces the solution.

   A flowchart is a type of diagram that represents an algorithm, workflow or process. The flowchart shows the steps as boxes of various kinds, and their order by connecting the boxes with arrows.

d. **Input/output:** Input, output and also error handling should be written like this:

Input for the program	eg. **n** (int)
Expected outputs from the program	eg. **factorial** (int)
Error handling (for preventing wrong data processing)	eg. Display error when $n < 0$

e. **Test Cases:** Test cases is defined as the output of the program. It should be written like this:

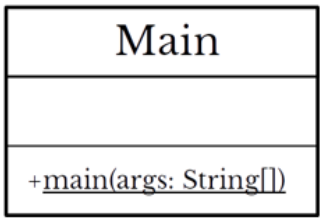
SL	Test Case	Expected Result	Observed Result	Status
1	Must start with a verb with test data as applicable, e.g. enter n value as 4)	Expected result when the program will be run, e.g. factorial output should be 24	Observed result after running the program, e.g. factorial output is 24	Status should be Ok / Failed, e.g. Ok for this test case as expected result and observed result have matched

Sample lab report:

Assuming LA3.1 with 2 problems and the first problem is to write a Java program to calculate sum of two numbers. Sample report will be as follows:

LA3.1.1 Write a Java program of sum of two numbers.

Class diagram:



Logic / flowchart:

Step 1: Start
Step 2: Declare variables num 1, num 2, and sum
Step 3: Read values num 1 and num 2
Step 4: Add num 1 and num 2 and assign the result to sum
sum <- num 1+ num 2
Step 5: Display sum
Step 6: Stop

Input/output:

Input for the program	**num1**, **num2** (int)
Expected outputs from the program	**sum** (int)
Error handling (for preventing wrong data processing)	None

Test Cases:

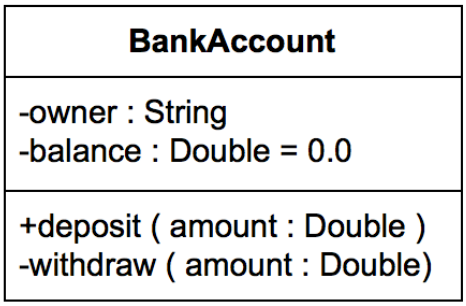
SL	Test Case	Expected Result	Observed Result	Status
1	num1 = 4 num2 = 5	sum = 9	sum = 9	ok
2	num1 = 8 num2 = 9	sum = 17	sum = 17	ok

Appendix A: Notes on UML Digram

Sample class diagram:

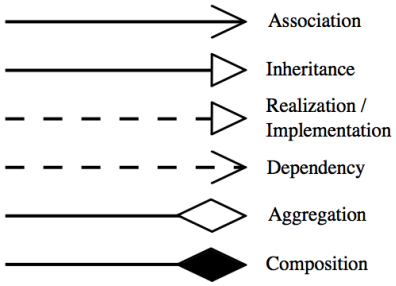
Class members (attributes and methods) have a specific visibility assigned to them:

public	+	anywhere in the program and may be called by any object within the system
private	-	the class that defines it
protected	#	(a) the class that defines it or (b) a subclass of that class
package	~	instances of other classes within the same package



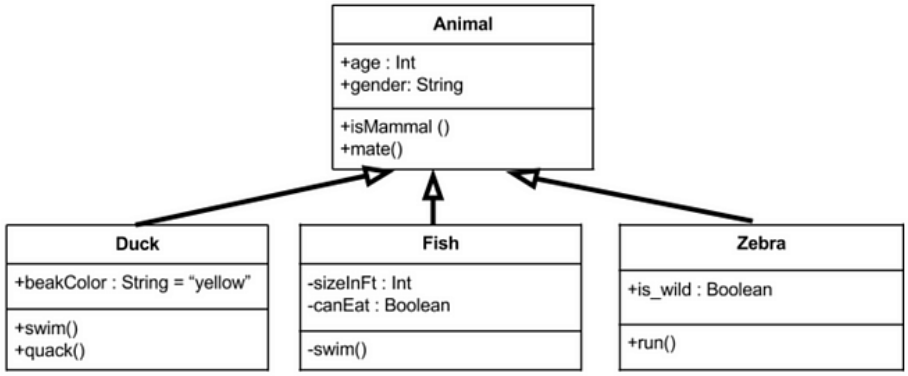
For the above class, *owner* and *balance* attributes are private as well as the *withdraw* method. But we kept the *deposit* method public, as anyone can put money in, but not everyone can take money out.

Notation of relationships:

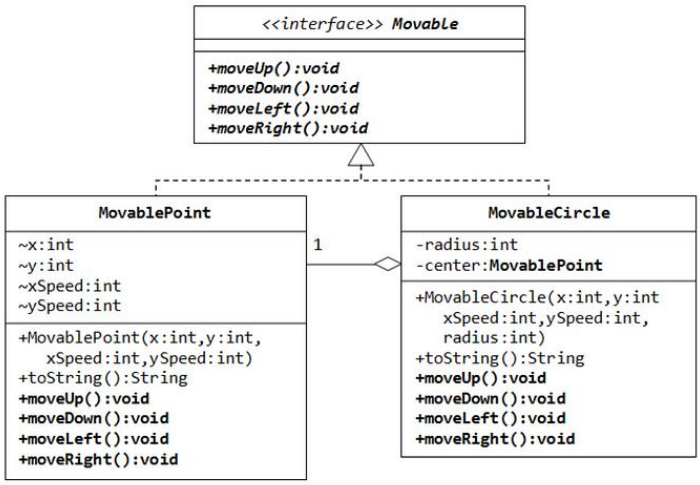


Association: Relationship between two separate classes. There are four different types of association: bi-directional, uni-directional, aggregation (includes composition aggregation) and reflexive. Bi-directional and uni-directional associations are the most common ones. This can be specified using multiplicity (one to one, one to many, many to many, etc.). Also, the relationship can be bi-directional with each class holding a reference to the other.

Inheritance: Indicates that child (subclass) is considered to be a specialized form of the parent (super class). For example, consider the following:



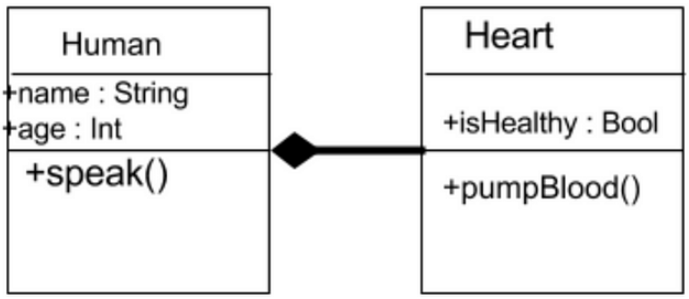
Realization/Implementation: A relationship between two model elements, in which one model element implements/executes the behavior that the other model element specifies:



Dependency

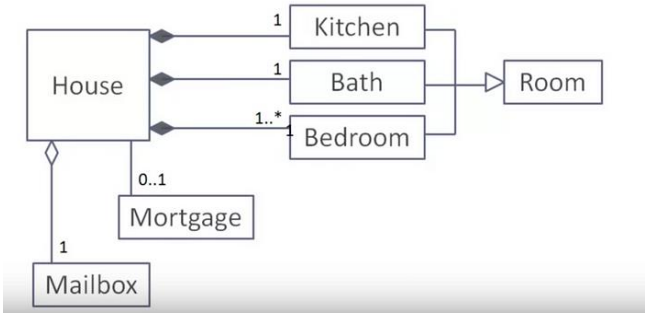
Aggregation: a special form of association which is a unidirectional (a.k.a one way) relationship between classes. The best way to understand this relationship is to call it a "has a" or "is part of" relationship. For example, consider the two classes: Wallet and Money. A wallet "has" money. But money doesn't neccessarily need to have a wallet so it's a one directional relationship.

Composition: A restricted form of Aggregation in which two entities (or you can say classes) are highly dependent on each other.



A human needs a heart to live and a heart needs a human body to function on. In other words when the classes (entities) are dependent on each other and their life span are same (if one dies then another one too) then its a composition.

Multiplicity: After specificyfing the type of association relationship by connecting the classes, you can also declare the cardinality between the associated entities. For example:



The above UML diagram shows that a house has exactly one kitchen, exactly one bath, atleast one bedroom (can have many), exactly one mailbox, and at most one mortgage (zero or one).