[LAB-2] Java Programming Basics

Objectives

- ✓ Be able to create Object Oriented Programming (OOP) in Java
- ✓ Understood the concept of interface and inheritance in Java

Software Development Environment:

- Java SE Development Kit JDK 17
- IntelliJ IDEA version 2022

Pre-requisite Reading:

The COMP3111 course assumes you will have basic programming experience (from COMP2011/2012 or 2012H), however, you might not have Java experience. If that is the case before starting on this lab, please read the course material (available in Canvas: Lab 2):

- PPT Slides: COMP3021 Topic 1: Introduction to Java Programming

Guided Lab Exercise & Lab Assignment

- Exercise 1: Learning OOP structure in Java;
- Exercise 2: Fill in the blank to complete an OOP Java application;
- Exercise 3: Learning and practicing Interface and Inheritance in Java;
- LAB ASSIGNMENT: Part-1 OOP Java Classes; Part-2 Interface and Inheritance;

Exercise 1: Create an OOP Java Application

OOP stands for Object-Oriented Programming.

Procedural programming is about writing procedures or methods that perform operations on the data, while object-oriented programming is about creating objects that contain both attributes (data) and methods.

Object-oriented programming has several advantages over procedural programming:

- OOP is faster and easier to execute
- OOP provides a clear structure for the programs
- OOP helps to keep the Java code DRY "Don't Repeat Yourself", and makes the code easier to maintain, modify and debug
- OOP makes it possible to create full reusable applications with less code and shorter development time

Note that the DRY principle is about reducing the repetition of code. You should extract out the codes that are common as a class for the Java application, and place them at a single and reuse them instead of repeating it.

Step 1.1 Build an OOP Java application

Now we are going to demonstrate the construction of OOP programming with two Java Classes:

- [Score.java] – A library to build the object-oriented Java classes.
- [mainApp2a.java] – An application main program to call the object-oriented Java class(es).

1. Launch IntelliJ IDEA.
2. From the main menu select Recent Project | "Comp3111LEx".
3. Span project tree [src], right click [src.main.java], then select New| Package and name the package as [Lab2a].
4. We shall create [Score.java] and [mainApp2a.java] classes under [Lab2a] package:
5. Right click [Lab2a] package, then select New | Java Class and name the Java Class as [Score].
6. Right click [Lab2a] package, then select New | Java Class and name the Java Class as [mainApp2a].
7. When creating the new Java Class, we may optionally select to add it to Git or not because this lab assignment does not insist to update Git. If you choose "Cancel", you can still add it later manually.

8. Go to [Score] editor, copy relevant code from page 2/3 of Appendices.
9. Go to [mainApp2a] editor, copy relevant code from page 3 of Appendices.
10. In [Score] editor, click the green small hammer icon to build project & compile code.
11. In [mainApp2a] editor, click the green small triangle icon to run the main application.
12. You will get the program's output result shown on page 4 of Appendices.
13. Let's go to page 5 of Appendices to explore the structure of OOP Java class [Score], you will find a class contains attributes[A], constructor[C], and method[M] where,
14. Variables Quiz, MidTExam, FinalExam, Score, grade, comment in class Score{} are [A] Attributes (= data of the class);
15. Score(){} is [C] Constructor, it used to initialize value to the object through parameter(s);
16. Procedures [M1] include setQuiz, setMidTExam, and setFinalExam are methods used to run several process(es), but not return value(s).
17. Functions [M2] include getScore, getGrade, getComment are also methods that creates to run and return value(s).
18. Now you need to think about if we create another package like [Lab2b] under [src.main.java], can we have another [mainApp2b] that can call the [Score] class under [Lab2a]? Yes, for sure we can work with cross package classes within a project. But, how?

Exercise 2: Fill in the blank to complete an OOP Java application

Step 2.1 Lab 2 assignment Part-1 – Build OOP Classes

1. Launch IntelliJ IDEA.
2. From the main menu select Recent Project | "Comp3111LEx".
3. Span project tree [src], right click [src.main.java], then select New| Package and name the package as [Lab2b].
4. We shall create [Book.java] and [mainApp2b.java] classes under [Lab2b] package:
5. Right click [Lab2b] package, then select New | Java Class and name the Java Class as [Book].
6. Right click [Lab2b] package, then select New | Java Class and name the Java Class as [mainApp2b].
7. Go to [Book] editor, copy/type relevant code from page 6 of Appendices.
8. Go to [mainApp2b] editor, copy/type relevant code from page 6 of Appendices.
9. In [Book] editor, fill in the blank of code in function Book(String argument[]).
10. In [Book] editor, fill in the blank of code in function getChapter(int i).
11. In [Book] editor, click the green small hammer icon to build project & compile code.
12. In [mainApp2b] editor, click the green small triangle icon to run the main application.
13. If you have adding the code correctly, you should get the program's output result shown on "Run" tool window (see page 7 of Appendices).
14. On execution, when we print an array variable directly, the program is expected only to print out the type and its address. For instance, "[Ljava.lang.String;@4e50df2e" will be printed for last print line: "System.out.println(anotherArray)". There is an alternative we can use some APIs like "java.util.Arrays" to dump the data listed in "anotherArray". Note that the API java.util.Arrays is from another package of the Java Libraries, we need to either be verbose to refer the class together with the package name or to import the package at the top of the program by adding "import java.util.Arrays;". In mainApp2b Java class, replacing the code line: with "System.out.println(java.util.Arrays.toString(anotherArray)). Please try it and check what the change of result will be.
15. Copy & save [Book] Java class as "Book.java" and submit it through Canvas
16. Hints:

Contents in final String array[]		
Array index	Chapter	Book Name
0	1	Basic Java
1	2	Advanced Java
2	3	Guru Java

Exercise 3: Learning and practicing Interface and Inheritance in Java

Step 3.1 Create class of inheritance

1. Launch IntelliJ IDEA.
2. From the main menu select Recent Project | "Comp3111LEx".
3. Span project tree [src], right click [src.main.java], then select New| Package and name the package as [Lab2c].
4. We shall create [Computer.java], [MobileComputer.java], [Charger.java], [Phone.java], and [mainApp2c.java] classes under [Lab2c] package:
5. Right click [Lab2c] package, then select New | Java Class and name the Java Class as [Computer].
6. Right click [Lab2c] package, then select New | Java Class and name the Java Class as [MobileComputer].
7. Right click [Lab2c] package, then select New | Java Class and name the Java Class as [mainApp2c].
8. Go to [Computer] editor, copy/type relevant code from page 8 of Appendices.
9. Go to [MobileComputer] editor, copy/type relevant code from page 8 of Appendices. Here [MobileComputer] inherits from [Computer]. Note:
   a) We use the keyword extends to inherit a base class.
   b) @Override is an annotation. This annotation explicitly tells the compiler that we are overriding the parent's method (or member function in C++ terminology).
10. Go to [mainApp2c] editor, copy relevant code from page 9 of Appendices.
11. Click the build (the green hammer) icon and compile the new classes.
12. If bugs-free, click the run (the green triangle) icon and then check the result whether or not same as page 10 of Appendices.

Step 3.2 Lab 2 Add Interface - Assignment Part 2 – Fix the problem on inheritance

1. We shall add [Charger.java] and [Phone.java] classes under [Lab2c] package.
2. Right click [Lab2c] package, then select New | Java Class and name the Java Class as [Charger].
3. Right click [Lab2c] package, then select New | Java Class and name the Java Class as [Phone].
4. Go to [Charger] editor, copy relevant code from page 11 of Appendices, add interface [Chargeable] to it.
5. Go to [Phone] editor, copy relevant code from page 11 of Appendices.
6. Go to [mainApp2c] editor, add 5 code lines at the bottom of the program from page 11 of Appendices.
7. Click the build icon and compile the new classes. You will find there is an error found on code line 25 { c.charge(m); } of the [mainApp2c] class.
8. Now you need to figure out the cause of the error and try to fix it as the part 2 of labs assignment.
9. Hints: read tutorial on Interfaces and Inheritance: http://docs.oracle.com/javase/tutorial/java/IandI/index.html.

LAB ASSIGNMENT:

Part 1: Complete Exercise 2, Step 2.1 – Fill in the missing code in [Book] Java class.

Part 2: Complete Exercise 3, Step 3.2 – Figure out the problem on line 25 of [mainApp2c] class & fix the bugs (modify [MobileComputer] Java class). Write your explanation how you fixed the problem.

Copy your final code of Part 1: [Book.java] and Park 2: [MobileComputer.java] and your explanation statement of Park 2 into one text file. Add a heading with your student ID and name on the document. Save the text file as [Lab2Assignment.pdf] and submit on Canvas before Lab2 due date.

Assessment:

1. 0 mark for no submission on Part 1; +0.5 mark for incorrect code; +1.0 mark for correct code.
2. 0 mark for no submission on Part 2; +0.5 mark for problem fixed; +0.5 mark for correct explanation.