Lab-4 (OOP)

Algorithms, Errors, and Testing

Objectives

- Be able to write an algorithm
- Be able to compile a Java program
- Be able to execute a Java program using the Sun JDK or a Java IDE
- Be able to test a program
- Be able to debug a program with syntax and logic errors

Introduction

Your teacher will introduce your computer lab and the environment you will be using for programming in Java.

In chapter 1 of the textbook, we discuss writing your first program. The example calculates the user's gross pay. It calculates the gross pay by multiplying the number of hours worked by hourly pay rate. However, it is not always calculated this way. What if you work 45 hours in a week? The hours that you worked over 40 hours are considered overtime. You will need to be paid time and a half for the overtime hours you worked.

In this lab, you are given a program which calculates user's gross pay with or without overtime. You are to work backwards this time, and use pseudocode to write an algorithm from the Java code. This will give you practice with algorithms while allowing you to explore and understand a little Java code before we begin learning the Java programming language.

You will also need to test out this program to ensure the correctness of the algorithm and code. You will need to develop test data that will represent all possible kinds of data that the user may enter.

You will also be debugging a program. There are several types of errors. In this lab, you will encounter syntax and logic errors. We will explore runtime errors in lab 2.

1. Syntax Errors—errors in the "grammar" of the programming language. These are caught by the compiler and listed out with line number and error found. You will learn how to understand what they tell you with experience. All syntax errors must be corrected before the program will run. If the program runs, this

does not mean that it is correct, only that there are no syntax errors. Examples of syntax errors are spelling mistakes in variable names, missing semicolon, unpaired curly braces, etc.

2. Logic Errors—errors in the logic of the algorithm. These errors emphasize the need for a correct algorithm. If the statements are out of order, if there are errors in a formula, or if there are missing steps, the program can still run and give you output, but it may be the wrong output. Since there is no list of errors for logic errors, you may not realize you have errors unless you check your output. It is very important to know what output you expect. You should test your programs with different inputs, and know what output to expect in each case. For example, if your program calculates your pay, you should check three different cases: less than 40 hours, 40 hours, and more than 40 hours. Calculate each case by hand before running your program so that you know what to expect. You may get a correct answer for one case, but not for another case. This will help you figure out where your logic errors are.
3. Run time errors—errors that do not occur until the program is run, and then may only occur with some data. These errors emphasize the need for completely testing your program.

Task #1 Writing an Algorithm

1. Download the file *Pay.java* from Your google classroom.

2. Open the file in your Java Integrated Development Environment (IDE) or a text editor as directed by your instructor. Examine the file, and compare it with the detailed version of the pseudocode in step number 3, section 1.6 of the textbook. Notice that the pseudocode does not include every line of code. The program code includes identifier declarations and a statement that is needed to enable Java to read from the keyboard. These are not part of actually completing the task of calculating pay, so they are not included in the pseudocode. The only important difference between the example pseudocode and the Java code is in the calculation. Below is the detailed pseudocode from the example, but without the calculation part. You need to fill in lines that tell in English what the calculation part of *Pay.java* is doing.

```
Display    How many hours did you work?
Input      hours
Display    How much do you get paid per hour?
Input      rate
```

Display the value in the pay variable.

Task #2 Compile and Execute a Program

1. Compile the *Pay.java* using the command line.

2. You should not receive any error messages.
3. When this program is executed, it will ask the user for input. You should calculate several different cases by hand. Since there is a critical point at which the calculation changes, you should test three different cases: the critical point, a number above the critical point, and a number below the critical point. You want to calculate by hand so that you can check the logic of the program. Fill in the chart below with your test cases and the result you get when calculating by hand.
4. Execute the program using your first set of data. Record your result. You will need to execute the program three times to test all your data. Note: you do not need to compile again. Once the program compiles correctly once, it can be executed many times. You only need to compile again if you make changes to the code.

Hours	Rate	Pay (hand calculated)	Pay (program result)

Task #3 Debugging a Java Program

1. Copy the file *SalesTax.java from your googleclassroom.*

2. Open the file in text editor as directed by your instructor. This file contains a simple Java program that contains errors. Compile the program. You should get a listing of syntax errors. Correct all the syntax errors, you may want to recompile after you fix some of the errors.
3. When all syntax errors are corrected, the program should compile. As in the previous exercise, you need to develop some test data. Use the chart below to record your test data and results when calculated by hand.
4. Execute the program using your test data and recording the results. If the output of the program is different from what you calculated, this usually indicates a logic error. Examine the program and correct logic error. Compile the program and execute using the test data again. Repeat until all output matches what is expected.

Item	Price	Tax	Total (calculated)	Total (output)