Learning Notes for Unit 1 – Part A

Writing Computer Programs

A computer program is a set of instructions that you write to tell a computer what to do. The most important part of programming is not simply writing the code. Before you sit down to write a single line of code, you need to analyze the problem at hand and break it down into sequential steps (i.e. generate pseudocode). This allows you to focus on the logic without being distracted by the syntax of the language you are using.

For example, if you were asked to write an interactive program that calculates the area of a lawn and display the results, the pseudocode could look like:

(1) Get length from user
(2) Get width from user
(3) Calculate and store the area (length*width)
(4) Display result to user

Each of these steps requires just a few lines of code.

When writing programs, you may find it useful to (1) generate the skeleton of your program, (2) insert your pseudocode as comments, and (3) put code into the program (under each comment). It will help keep your program organized.

Procedural Programming

In PROG1100 (Programming I), you learned the basic concepts of program writing such as:

- creating pseudocode
- using variables to store values
- performing various manipulations using variables (e.g. mathematics)
- writing functions
- decision making using control statements (if, switch, for loops, while loops)
- using arrays
- variable scope

The programming approach introduced during that first course is called “procedural programming”. In a procedural program, each statement of the program is executed, one after another, until the end of the program has been reached. During the first two modules of this course, you will learn how to implement these concepts in an object-oriented programming language.

What is object-oriented programming?

This course introduces you to the principles of object-oriented programming and how to implement an object-oriented programming solution.

Software objects are essentially reusable code units that you can implement to achieve a programming solution to a problem. For example, in Part B of this unit, you will use a pop-up dialog box to interact with someone using your program. Reusable software objects are useful as It would be a time-consuming affair to have to write all the required code to do this task (generate dialog pop-up) from scratch each time you need to get user input into a program. Fortunately, someone has already created the blueprint for collecting information from users in a pop-up dialog and it was incorporated into the programming language for you to simply import into your program and (re)use as required. Code reuse is one of the biggest advantages of an object-oriented programming approach.

In this course, you will learn:

- the underlying concepts behind object-oriented programming,
- learn how to build custom classes
- interact with pre-built and custom objects

There are several terms you need to become familiar with:

- A ***class*** describes a collection of data variables along with functions that work with them.
- ***Attributes*** refer to the collected data variables within a class
- ***Methods*** are functions that allow you to access the stored attributes
- ***Encapsulation*** is the process of hiding data and methods within an object (to prevent inadvertent changes)
- ***Inheritance*** is the process of a creating new object by re-using a preexisting object and customizing functionality
- ***Polymorphism*** allows you to reuse methods with the same name but having different behaviors

Object-Oriented Programming with Java

For this course, we will use Java for the object-oriented programming language. There are many different object-oriented development environments that are available to use, but Java offers a number of advantages:

(1) Java is “simple” – Java leaves out many complex features of other high-level programming environments such as pointers and multiple inheritance.
(2) Java is “compiled” – Java has an intermediate step when projects are compiled. It produces byte code that is machine independent and can be executed on different platforms
(3) Java is “multi-threaded” – Java has a built-in ability to easily handle multiple programming threads
(4) Java performs “garbage collection” – Java handles the freeing up of system memory automatically
(5) Java is “robust” – Java verifies all system calls performed in a program. A java program cannot crash the system (however, a faulty java interpreter can)
(6) Java is “secure” – Java verifies all memory access to protect from unauthorized access

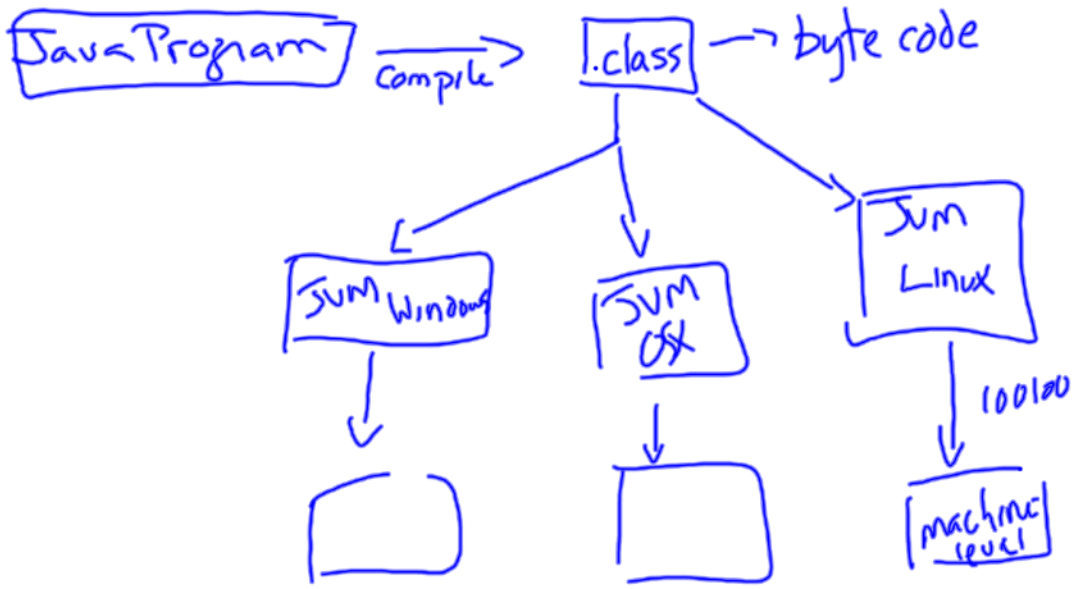
Moving to other languages

An additional advantage to learning Java is that the language is structurally similar to many other programming languages. PHP and C share many syntactic similarities, such as the use of semicolons to end a statement, the use of **{** and **}** to indicate blocks of code, and the formatting of **for**, **if**, **switch**, and **while** controls.

Working with Java

When you write a java program, you “translate” your program pseudocode using the Java language syntax to create a human readable program file with the .java extension (ex. AreaCalculator.java).

When you compile your program, a .class file is created. This code (also known as **btye code**) is what makes Java portable. This single file can be run on any operating system with an installed **Java Virtual Machine** (**JVM**). The JVM interacts with the computer at a machine-level.



Reserved words/Valid variable names

When naming variables within a Java program, there are a number of rules that must be followed:

- There are a number of reserved keywords in Java that cannot be used as variable names. http://java.about.com/od/javasyntax/a/reservedwords.htm
- The variable name must consist of allowed Unicode characters (127 ASCII characters + other)
- There is no limitation on the length of the variable name
- You cannot have a space in the variable name
- The first letter must be a letter, underscore, or $
- You cannot have two variables visible in the same scope with the same variable name

Writing Java programs

For the first few units, your Java programs will have the same general format which is outlined below:

```
public class NameOfProgram {

    public static void main ( String [] args ) {

        //the code to be executed goes here

    }

    //any additional functions/methods for your program go here

}
```

Breakdown of the skeleton:

- ***NameOfProgram*** must match the exact spelling of the file's name (***NameOfProgram.java***)
- The ***public*** keyword indicates that your code is accessible by other programs that need to access the code inside
- The ***class*** keyword indicates that the name following defines a custom data type
- The ***static*** keyword means allows you to reference a function (the main function in this case) without having to allocate memory first.
- The function **main () { }** is the main block of code to be executed. Execution starts at the start of the block (just after the opening "**{**") and ends at the end of the block (just before the closing "**}**")
- **String [] args** is an array of String data that accepts parameters sent to the main function at run-time

The programs you write during these first two units will follow the format shown above. You will change the **NameOfProgram**, add code to the **main** function, and add variables and functions as required.

Commenting Your Code

Adding comments to your program will help explain what the purpose of the code that follows is. Inserting pseudocode into a your program skeleton as comments will actually help you put your program together. There are three types of comments you can use in your program:

- Line comments which start with a pair of forward slashes (**//**). Everything from that point to the end of the line will be ignored by the compiler when you build and run your program.
- Block comments which start with a forward slash and asterisk (**/***) and end with an asterisk and forward slash (***/**).
- Javadoc comments are a type of block comment that start with (**/****) and end with (***/**). It is used to write supporting documentation for the class as a whole rather than small segments of code.

Setting up your development environment

Please refer to the guide in the Handout section to set up your development environment.

- Please note that if you use a different version of the software, the installation paths will be different.

- Java software installed on your computer from java.com will run pre-built classes, but it is not capable of compiling the code you write into byte code that can be run. What you get from java.com is a **Java Runtime Environment** (**JRE**). You need a **Java SDK** (**software development kit**) for this course.

- Please note that you need to use the full path to the javac.exe and java.exe files to compile and run your programs (unless you work out of the folders that these programs are in). However, you can add the java bin directory to your PATH environment variable.

  Beware: If you add the directory to your path, be sure to add a semicolon at the end of the entries already in the path **BEFORE** you add the path to the java executables.

  BE CAREFUL: If you delete/replace entries in the PATH, your operating system will cease functioning.

  http://www.java.com/en/download/help/path.xml

Writing command-line programs

- Open a text editor such as Notepad, Notepad2, Notepad++ (Windows) or TextWranger (OSX). Create a new file
- Add the basic structure of your java program

```
public class NameOfProgram {

    public static void main ( String [] args ) {

        //the code to be executed goes here

    }

    //any additional functions/methods for your program go here

}
```

- Change **NameOfProgram** to the name of your java class (e.g. Room)
- Save the file with the same name as your java class with the .java extension (e.g. Room.java)
- Open a command prompt (Start>Run>cmd + enter)
- Browse to the directory that you saved your file.
    - If you need to change drive paths (i.e. your file is not saved on the c: drive), type the drive letter followed by a colon, then press enter (e.g. **H:**)
    - Change directories to where your file is saved. Type cd, then put a space, then the full path to your directory (e.g. **cd /java**) then press enter
    - You should then be in the directory your .java file is in. Type dir followed by enter to verify
- Compile the .java file by typing **javac NameOfClass.java** (e.g. javac Room.java)
- If there are errors, go back to your text editor and fix the first error identified. Then recompile your file. Repeat until there are no errors.
- Verify that there is a .class file for your program in the working directory (e.g. typing **dir** followed by enter reveals that **Room.java** and **Room.class** both exist in the same folder.
- Run your program by typing **java NameOfClass** (e.g. java Room). Note that .class is assumed and does not need to be added
- Please refer to the slides in the Resources area if you get stuck

Writing console output:

Creating a program that does not return a visible result to the screen to notify you of progress is not very useful.

You can use the following code to display a message to the console window:

System.out.println(“message goes here”);

Learning Notes for Unit 1 – Part B

Examining data types

There are a number of **primitive** data types in java that you can use to store values in your programs. Table 1 on page 129 of your text lists what is available.

There are four options if the value you are storing is an integer (i.e. has no decimal component): byte, short, int, or long.

There are two options if the value you are storing is a real number (i.e. has decimal component): float or double

If the value you are storing is a single character, you would use the ***char*** data type.

If the value you are storing is word or sentence (i.e. multiple characters), you would use the ***String*** data type.
Note: This data type is not considered a primitive data type as it stores multiple character values. It is considered a **reference** data type

The difference between similar types is the range of values that can be stored.

A ***byte*** data type can store numbers from -127 to +127.

A ***short*** data type can store numbers from -32768 to +32767.

As the range increases, so does the demand on memory storage. For simple programs, demands on memory is not great so the ***int*** data type is usually used to store integer values and the ***double*** data type is usually used to store real numbers.

If you are programming for mobile devices or any other platform with limited resources, your choice of storage type must reflect the range of values to be stored. For example, if you need a counter in a loop that goes from 0 to 10, you would use a byte data type.

Variable Declaration Syntax:

DataType variableName;

or

DataType variableName = value;

Examples: int x; int x=0; double y; double y=5.5;

Performing mathematic operations

When manipulating variables to calculate a new value (e.g. calculating an area using stored width and height, adding a value to a running sum, determining if a value is odd or even), there are a number of operators that can be used.

Adding:

The **+** operator adds two values together. The result has the data type of the type with the larger storage capacity. A ***long*** data type added to a ***int*** data type yields a ***long*** data type for the result. An ***int*** data type added to a ***double*** data type yields a ***double*** data type for the result.

Subtracting:

The **-** operator subtracts the value on the right from the value on the left. The result has the data type of the type with the larger storage capacity (see adding for examples).

Multiplying:

The ***** operator multiplies two values together. The result has the data type of the type with the larger storage capacity (see adding for examples).

Dividing:

The **/** operator divides the value on the right into the value on the left. The result has the data type of the type with the larger storage capacity (see adding for examples).

Caution: Dividing two integers gives an integer result.

7 / 2 yields 3 as the result (NOT 3.5)

Preventing integer division: One of the variables must be converted to a double before performing the math. There are two ways to do this: (1) perform type casting (see below); or (2) add 0.00 to one of the values before division (7 + 0.00)/2 would yield 3.50000

Remainder:

The **%** (modulus) operator returns the remainder of the division of the value on the right into the value on the left.

Ex. 7 % 2 => 2 goes evenly into 7 three times (comprising 6 of the 7). The remainder is 1

Ex. 100%32 => 32 goes evenly into 100 three times (96). The remainder is 4

Caution: Remainder math only works on two integers.

Order of Operations:

1. Math within parentheses is done first
2. Multiplication, division, remainder math all have the same precedence. Perform operations proceeding from left to right
3. Addition and subtraction both have the same precedence. Perform operations proceeding from left to right

Type casting:

From time to time, you may have to make a data type behave in a different manner.

Such as treating an int data type as a double data type

Example:

int x = 7;

int y = 2;

x/y => integer division yields 3

Two possible solutions:

(double) x => temporarily treats x as an double

(double) x / y => regular division yields 3.50000

Sometimes you may need to convert a string to an int or a double.

String => int Conversion (use the methods below):

int x = Integer.parseInt(StringVariable);

double y = Double.parseDouble(StringVariable);

Writing console output (Multiple values):

To add variables to your console message, there are two method you can use:

(1) System.out.println method => whenever you need to display a variable, stop the string with a double quote, use a plus and then the variable. If you need to resume the text, use another + and a double quote to restart the string.

System.out.println("Text " + variable1 + "more Text " + variable2 + "…");

or

(2) System.out.printf method => Display the whole text that you want to display but put placeholders where variables are to be displayed. When the string is finished, end your double quotes and list all variables to be displayed in the placeholders (use commas to separate)

System.out.printf("Int is %d, double is %f, string is %s, formatted double is .2%f", intVariable, doubleVariable, StringVariable, doubleVariable);

The advantage to printf is that you can format the displayed values (i.e. to 2 decimal places as shown above).

Using JOptionPane to interact with users:

To make your programs more user-friendly, it is desirable to add some pop-up boxes to present messages to the users and to retrieve information from them. To add this functionality to your program, you need to add an import statement to the very top of your program:

```
import javax.swing.JOptionPane;
```

To pop-up a message box, use:

```
JOptionPane.showMessageDialog(null, “message to display”);
```

To get input from users, use:

```
String strInput = JOptionPane.showInputDialog(null, “question to ask”);
```

When you retrieve your input, you may have to type cast the String to a different format.

Setting up IDE (Eclipse):

Please refer to the notes in the Handouts section to set up the Eclipse IDE. The notes also cover how to create a Java Project, create a new program, and run it.