Week One Class Notes: Chapter 2: Basic Programming Elements

**Goal**: Review basic **input**, **processing** and **output**, programming concepts, and introduce Object Oriented Programming (OOP)

**So you want to write a program**.

In Java we write a program by first creating a class, and then putting one or more methods inside of the class to hold the code. All of the programs created in the course will have one class that contains the **Main** method. This is the first method that runs when the program runs.

Here is an example of a project called “SampleProject”. It has the required **main** method and a second optional method. There is no code in the methods yet. The program will run but it will not do anything.

public class **SampleProject** {

public static void **main**(String[] args) {

// TODO put code application logic here

} //End of Main Method

public void **method2** (){

// TODO put more code application logic here

} //End of Method2

} //End of Class

The code we write goes inside **methods**. Methods use curly brackets {} to hold blocks of code. Each beginning bracket MUST have a corresponding ending bracket. Do **NOT** put code after the ending bracket of a method or class. When methods become long a comment at the end of the method can help remind us were it ends. We create our own methods in Chapter 4. Additional blocks of code can be created within a method to help organize the code. The style of how the code is written is important for making the code easy to read. Comments are not code but help to organize the code.

As programs get more complicated other methods will be created to help manage the different actions the program accomplishes. When we want to be even more organized we will create new classes to hold different types of related methods. There will be a class that holds all of the methods that process dates and time, another that processes text data, and yet another class with methods for working with a database. These classes do not need to have a main method. Think of them as **helper** classes. Many helper classes have already been created for the Java programming language. A class that only has related methods is said to have **high cohesion**. A class with unrelated methods, such as finding the local tax rate and calculating a baseball player’s statistics, has **low cohesion**.

Every meaningful program will (1) get **input**, (2) **process** the input, and (3) create **output**.

**Input, Processing, and Output**

The **Scanner** class is a helper class that gets **input** from the user as they type on the keyboard. To keep a program small, only the **java.lang** package is automatically included in a Java program; any other class must be imported. Thus the Scanner class, which is part of the **java.util** package, must be ***imported*** before it can be used in the code.

The scanner class is discussed in chapter 3. This is how the scanner class creates a new variable, named ‘sc’, and stores input from the user in a variable called ‘name’.

Scanner sc = new Scanner(*System.in*);

String name = sc.nextLine();

**How to hold data**

Before we **process** the **input** data retrieved from the scanner object we will store it in a variable. Variables are containers that hold data. Each type of variable is designed to hold a different type of data. Each data type is exactly like a class. When a variable is created the syntax to make the variable is similar to that used for objects. The only difference is that the ‘***new***’ keyword is not used. A value is always assigned to the variable when it is created.

int myIntegerVariable = 0;

String myStringVariable = ""; //this is an empty string

Once a variable is created we can change the value anytime with an ***assignment*** statement. The **assignment** statement takes the value on the Right Hand Side of the equal sign and puts it into the variable on the Left Hand Side of the equal sign.

myStringVariable = "stuff";

**Processing Examples**

Two ways to **process** data are with the **If** statements and **Loop** statements. Both are covered in more detail in chapters 3 and 8. They both use a ‘**condition’** or ‘**Boolean expression**’ to decide what processing should occur. There are six **relational operators** used in the Boolean expression to compare *numbers*: Equality (==), Inequality (!=), Greater Than (>), Less Than (<), Greater Than or Equal (<=), Less Than or Equal (<=).

An If statement always has a **true** branch and optionally can have a **false** branch by using an **Else** statement. When there is only one line of code in a True branch of an If statement, curly brackets are not needed.

if (myIntegerVariable != 2)

System.out.println("myIntegerVariable is NOT a two");

else

System.out.println("myIntegerVariable is a two");

**String** variables are different from numbers and do not use the relational operators. Because a String variable is more like an **object** we use **methods** that are found in the String variable. To determine if the value of a string variable equals a literal string the **equals**() method is used.

if (myStringVariable.equals(""))

System.out.println("myStringVariable is empty");

else

System.out.println("myStringVariable is NOT empty");

Variables that hold String objects, or any other type of object, are considered Null before an object is created and assigned to the variable. The code to test if an object is null is

if (myObjectVariable == null)

System.out.println("myObjectVariable is null");

To **output** data to the console we will use the **System.out** object. These objects were used in the examples above. They are objects that are automatically created so we do not have to write the line of code to create the object from the class. But like any other object it has **methods** we want to use. The two methods of interest are **print()** and **println()**. The print() method prints any string argument found in the parentheses to the screen and stops. The println() method prints the string argument found in the parentheses to the screen, and then ‘hits the enter key’, causing the cursor to be at the beginning of the next line. We can use either method with no arguments. This is common with println(), but not print().

**Practice Exercise**: Create a hello world project

At the top of the program add **comments** that have your **Name**, the **Date**, and a **Description** of the program.

**Input**: Ask the user for their name. Store the name in a **variable**.

**Processing**: Create a **variable** that holds the output message, such as “Hello Mike” based on the user entering “Mike” as the name.

**Output**: Print the message

\*\* Create a **Word** document with the Output and Code. Get rid of extra empty lines in the Word document. \*\*