Lecture 5 - 1/30/2024 Last Updated 1/20/2024

Java is a strongly typed language, meaning that we must denote all variables we declare with a type. These types can be grouped into two broad categories:

1. Primitive Data Types
2. Non Primitive Data Types

You are already familiar with the following data types from previous lectures:

1. int - Uses 4 bytes of memory
2. double - Uses 8 bytes of memory
3. boolean - Uses 1 byte of memory

You do not need to worry about the amount of memory allocated for each type just yet it will be repeated in a later set of notes, just stating it here for completeness.

To declare a variable in Java - in the generic* sense - we can use the following format:

```
dataType variableName;
```

To initialize a variable in Java we can go to a separate line and do the following:

```
variableName = valueOfVariable;
```

But we do not have to do both of these separately we can combine this into a single line which yields the following:

```
dataType variableName = valueOfVariable;
```

The data type matters because it influences the output of mathematical operations, let us consider the same operation with one as a double and one an int:

```
int x = 5;
double y = 5;
System.out.println(x / 2 + “,” + y / 2); //yields 2,2.5
```

Even though y and x hold the same value (5) the nature of doubles being capable of floating point arithmetic means we will have a decimal component to the answer.

The native library of Java, which includes everything you can use out of the box comes from the java.lang package. This package is imported for you so you do not need to worry about importing it yourself. That being said, sometimes we may want to gain access to extra functionality and we can do this by writing import statements at the top of our .java files.

For instance, considering the question of user input, there is no simple way to do that with the native java library so to do so we import another class to help us out with that. In particular we import the Scanner class:

```
import java.util.Scanner;
```

From this we now have access to the Scanner class, if instead of importing a single class from a package you want to import multiple classes from the same package then you can replace the class name with the wildcare (*)

```
import java.util.*;
```

Now let us turn our attention back to the Scanner class, it is essential for collecting user input. In order to use a Scanner we must instantiate an instance of it. To do so we can write the following:

```
Scanner sc = new Scanner(System.in);
  ^     ^    ^     ^         ^
  |     |    |     |         |
  a     b    c     d         e
```

It is important to understand what all of this means so let's break it down:

a. Scanner: This is the name of the Class we wish to make an instance of
b. sc: This is the name of the reference that we will be using to access the properties of the Scanner, it is the name of our variable.
c. new: This is a reserved keyword in Java used for all* object instantiations (*there are exceptions like Strings)
d. Scanner(): This is a constructor call, you will learn more about constructors later but for now just bookmark this detail
e. System.in: This is the argument to the constructor.

Once again this process is referred to as object instantiation, with objects being instances of classes, you can either use predefined classes to make objects (like we did with the Scanner class) or you can make objects from your own classes! The following is a generic* version of the previous line of code:

```
ClassName variableName = new ClassName(argument1, … , argumentN);
```

For the record you do not actually need to have arguments for your constructor, it really depends on how the constructor is defined.

Once we have a Scanner object we can collect different types of input from the user using the following methods:

1. `int x    = sc.nextInt();`
2. `double x = sc.nextDouble();`
3. `String x = sc.nextLine();`
4. `String x = sc.next();`

We typically have a println prior to collecting input so the user has a prompt to work with, so if you are working and try running and get a blank terminal, consider adding a prompt if you do not already have one.

Also number 3 and 4 vary slightly, so please reference the Scanner class's documentation for more information: (Scanner Class: Java 8)

Lecture 6 - 2/1/2024

I see plenty of you typing your notes within the Java files that are being used in lecture and plenty of you are using single line comments as follows:

```
// I am a single line comment
```

Multilined comments also exist within java meaning you do not need to write // each time you want to jot down some notes:

```
/*
 * This is a multi line comment
 */
```

Today you saw the classic game of Rock Paper Scissors in Java and there were a few new concepts that were introduced.

The first was the concept of constants. In Java we can denote variables with the final keyword turning them into constants, we use the keyword prior to the data type, when writing out the variable name for constants the convention is to type the name in all caps with underscores separating individual words. Here is an example illustrating this:

```
final int MY_FIRST_CONSTANT = 42;
```

The second was the random() method from the Math class. The Math class is native to Java so you do not need to import it. The Math class has tons of different constants and methods that are all very useful but will be omitted from discussion as there is no legitimate reason to go down that rabbit hole. You can read the documentation here: (Math Class: Java 8).

Math.random()returns a double that can hold any floating point value across the following range: [0.0, 1.0) for a reminder on the notation, this is saying we can get values starting from 0.0 and going up to but not including 1.0, this allows us to generate random numbers and perform actions based on chance and probability rather than discretely defined values. We can manipulate the range using multiplication and addition/subtraction.

Finally you were introduced to Type Casting, this is different from Type Promotion which happens naturally by the JVM, casting is a user's choice and when desired, can yield necessary and proper results but also could lead to errors if you are not careful. To cast to a different type you simply put (datatype) in front of an expression, be careful for the order of precedence as well, as you may get undesirable output, when in doubt use parentheses to denote intention.