Lecture 7 - 2/6/2024 (Last Updated 1/27/2024)

This week in lecture we started to shift from procedural programming principles to object oriented programming. This is the main benefit of using Java compared to other programming languages. Object oriented programming in Java - for purposes of this course - relies on you defining your own data types. Like you saw and heard in lecture. Classes are blueprints used to construct objects. When creating classes from this point forward there is a few things that you'll need to make sure you understand and include:

1. Instance Variables
2. Constructor(s)
3. Mutators and Accessors

Instance variables are variables that belong to each instance of the Class that you instantiate, so like in lecture you can instantiate multiple BankAccount objects, each with their own balance, account number and name. All instance variables you create shall be marked private, this makes it so no outside entities can directly modify or access these values without using the mutators or accessors that you define.

Constructors are special methods that have no return type in the method signature - including void - the job of the constructor is to initialize the instance variables for that instance of the object. You can define multiple constructors for the same class as long as the method signatures are different.

Mutators and Accessors are the types of methods that you can write in Java. Mutator methods, modify the instance variables and do not return anything and are void. Examples of mutators are the deposit() and withdraw() methods from the BankAccount class from lecture. Accessor methods do not modify the instance variables but instead return an instance variable to the user. An example would be the getBalance() method from the BankAccount class from lecture.

At this point you should be able to define your own methods and use them in Java the general form for defining a method is as follows:

```
<access modifier> <additional modifier> <return type>
        methodName(parameterList){}
```

An example would be as follows:

```
public static void main(String[] args){}
```

The access modifier is public, the additional modifier is static, the return type is void, the method name is main and we have a single parameter called args which is of type String[]. This is the form which you will use to define all methods going forward.

Whenever we write our own data types we never run any actual code in the same file, we write another java file that uses the Class we previously wrote in order to execute code in this new file. This is referred to as a driver class, tester class, and possibly a leader class. In lecture you saw this as the BankTest.java file.

Last week you were reminded of the basic primitive types but today we introduced the rest of them today along with the amount of memory they take up, a chart reviewing that is down below:

Data Type	Bytes Required	Bits (Bigits) Required
byte	1 byte	8 bits
short	2 bytes	16 bits
int	4 bytes	32 bits
long	8 bytes	64 bits
float	4 bytes	32 bits
double	8 bytes	64 bits
boolean	1 byte	8 bits
char	2 bytes	16 bits

Lecture 8 - 2/8/2024

Not many new concepts were covered in lecture today, so this section will be short but you did learn some valuable new concepts today. When we invoke a method such as the following two examples:

```
Integer.parseInt("23");
MyAccount.withdraw(100);
```

Today you learned that there are explicit parameters and implicit parameters: Explicit parameters are the parameters found within the parentheses so for our examples above "23" and 100 are explicit parameters. Implicit parameters are the objects calling the method meaning implicit parameters only appear in non static contexts. Meaning while MyAccount is an implicit parameter, Integer is not since Integer is a static class. This static keyword is the same one you write every time you write a main method in Java.

Today you also learn that Strings are immutable meaning they cannot be changed once they are created, being immutable is not the same as being a constant. You can reassign immutables all day long. Here is an example:

```
String s = "1004 is ";
    s += "great";
```

The String s reads as you would expect "1004 is great" but it is not the same s object that we initially initialized s with. That is a separate object that no longer has a reference to it. We have access to one object but create two. In lecture today you also saw Integer.parseInt(). This is a method that takes in a String as input and returns an int version of the argument assuming the argument is valid. In the event you had an int and needed to convert it to a String, you could use Integer.toString() which does as you think it would.

Finally a brief review of the boolean operators that you have seen so far:

1. && is the logical AND operator | A && B evaluates to true iff A and B are true
2. | | is the logical OR operator | A | | B evaluates to true iff A or B is true
3. ! is the logical NOT operator | !A evaluates to true iff A is false