More Java Fundamentals

Last time we were together we covered Variables, Data-Types, Class, Objects and Constructors. These make up the building blocks of Java for the purposes of the COMS1004 class, today we will cover Strings, Methods, Loops, Control Structures, and the beginnings of Object Oriented Programming.

Strings: Strings are a special data type in Java, they are not a primitive type and they are objects but they can be declared like primitive types and objects. Strings can be declared in Java using either method as follows:

```
String myFirstString = "This is my first String.";
String my2ndString = new String("This is my Second String");
```

Since Strings are objects there are methods that can be called upon the Strings we created, these methods will be covered in detail in a bit but for now there are a few things to know about Strings:

1. Strings are simply an array of chars meaning the following is the same as the first String object:
   ```
   char[] data = {'T','h','i', 's'....};
   String myFirstString = new String(data);
   ```
2. Strings are immutable; you cannot change a String object once you make it, if you do concatenation on a String it creates a new String Object.
3. String manipulation is probably the most common thing you will do in Java

Methods in the String Class

The String Class has many classes, most of which are irrelevant for this class but we will cover the most used ones and the ones we need to know

```
//used for String comparisons, returns a boolean
equals(Object obj);
//used to find the index that a substring occurs at within a
//larger String, returns an int
indexOf(String str);
//takes the entire string to UPPERCASE
toUpperCase();
//takes the entire string to lowercase
toLowerCase();
//used to find the length of a string, returns an int
length();
//used to create strings from the contents of a string,
//returns a new String object
substring(int start, int end);
```

```
//used to see if a substring is found within a larger String
contains(String str);
//used to get the specific character at a distinct index in //a
string
charAt(int index);
```

Methods in Java

In Java it is really sloppy and unconventional to write all of your code in the main method so you can write your own methods to abstract away the details of the code. The formal definition of a method is as follows:

```
<access specifier> <modifier> returnType methodName(argumentList){
     //code goes here
}
```

Here is a proper example of creating a method by using the most used method that is required for Java programs to run: The Main Method:

```
public static void main(String[] args){
     //code goes here
}
```

Let's break it down, in this case "public" is the access specifier, meaning it can be accessed by any class that wants to use it. "static" is the modifier in use here, for now do not worry about what it means we will cover that when we talk about object oriented programming. The return type is "void" meaning this method returns nothing, the name of the method is "main" and the argumentList is a single String array called args.

With methods there are two key things that we can do: ***override and overload.*** We will talk about overriding methods when we discuss object oriented programming. For now we will stick to overloading methods. Let us assume that for some reason you wanted two versions of a method called "add" that simply adds values together so let us assume that your original method looks as follows:

```
public int add(int a, int b){
     return a + b;
}
```

You want a method, also called add, that can take three integers as input as well keeping the standard two integer method as well, well in this case we find the first way you can overload a method and that is by adding more parameters to the argument list, that being said the following is legal in Java:

```
public int add(int a, int b, int c){
     return a + b + c;
}
```

The other way to overload methods in Java would be to change the data types of the parameters that being said the following is legal in Java:

```
public double add(double a, double b){
     return a + b;
}
```

You can of course change both the number of parameters and their data types but it is never enough to just change the return type of the method, the following is not legal in Java and will not compile:

```
public double add(int a, int b){
     return a + b;
}
```

That is the basics of defining methods and overloading them now onto repetition structures.

Repetition Structures (Loops)

In Java there will be times where you will need to repeat a set of instructions. Rather than copying and pasting the instructions several times we can use a loop to repeat a set of instructions until a certain condition is met. There are 2 key types of loops in Java: The While Loop and the For Loop

The While Loop

The while loop is a repetition structure that repeats a set of instructions until a boolean condition is met. The syntax for a while loop is as follows:

```
while(booleanExpression){
     //code to run
}
```

Beware of creating infinite loops, you must make sure when you are programming that you make sure there is something in the code of your loop that eventually breaks you out of the loop.

The do-While Loop

The do-while loop is exactly like a while loop except that it is GUARANTEED to run at least once the syntax for a do-while loop is as follows:

```
do{
     //code to run
}while(booleanExpression);
```

Just like with regular while loops, be careful you do not create an infinite loop.

The For Loop

The for loop is a repetition structure that repeats a set of instructions a set number of times, it is the loop most commonly used to iterate through arrays. The syntax for a for loop is as follows:

```
for(initialization, booleanExpression, updater){
     //some code to run
}
```

Control Structures

In Java sometimes we need to do different things depending on what conditions have been met. There are two main types of control structures: If-else statements and Switch-Case statements, let us begin with the If-else statement and to start here is the traditional if statement:

```
if(booleanExpression){
     //code to run
}

//code here runs regardless of the conditional above
```

The above statement basically says if the boolean expression is true, run the code within the if statement. The code outside an if statement runs regardless of, but what if that was not the intent we wanted well now sounds like a perfect time to introduce the else statement. Else statements can only exist with an if statement. The syntax is as follows:

```
if(booleanExpression){
     //code to run if booleanExpression is true
}else{
     //code here now only runs if the above doesn't
}
```

Finally, what if we want to check multiple different conditions? Well there is an Else If statement to help us with that, you are not required to have a standard else with this but it is recommended. The syntax for it is as follows:

```
if(booleanExpression1){
     //code to run if booleanExpression1 is true
}else if(booleanExpression2){
     //code to run if booleanExpression2 is true
}else{
     //code here now only runs if the above doesn't
}
```

The next type of conditional is the Switch-Case statement. It is important to note that the Switch-Case only works with the following types: byte, short, char, and int primitive data types, the String class, and the wrapper classes for the previously mentioned primitive types (do not concern yourself with this for now). The syntax for a switch-case is as follows:

```
int month = 2;
String monthString;
switch (month) {
    case 1:  monthString = "January";
             break;
    case 2:  monthString = "February";
             break;
    case 3:  monthString = "March";
             break;

    default: monthString = "Invalid month";
             break;
}
System.out.println(monthString);
```

It is important to note the break statements, if you do not put the break statements, then if one of the cases executes all the ones below it will execute until you either hit a break statement or exit out of the switch statement, so be careful!

Before we can move on to the basics of object oriented programming it is important to talk about operators and their precedence in Java. The following is the list of operators that are relevant and their precedence, if they are on the same line they are evaluated from left to right. The further down the list you go the lower the precedence:

Operator Type	Symbols
Post-Unary	expression++, expression--
Pre-Unary	++expression, --expression
Other Unary Operators	-,+,~,!
Multiplication/Division/Modulus	*,/.%
Addition/Subtraction	+,-
Relational Operators	<,>,<=,>=. instanceof
Equation	==, !=
Short Circuit Logical Operators	&&, ||
Ternary Operator	booleanExpression ? expression1 : expression2
Assignment Operator	=, +=, -=, *=, /=, %=

If you have any questions about any of these operators please ask :)

Object Oriented Programming 101

Java by nature is an object oriented programming language, this means that the true scale of the things you can do in Java can only be done using objects. We saw last week we could create an object of a certain type in the following way:

```
ClassName variableName = new ClassName(argumentList);
```

We also talked about constructors and learned that they are special methods with no return type (not even void) There is a special keyword when talking about object oriented programming called the ***this*** keyword, it is used to refer to the current object, simply think to yourself “I want this object’s variable” etc

You do not actually have to explicitly define a constructor, if you don’t then the default constructor is called and looks like the following, important to note once you define a constructor you no longer get the default constructor:

```
public ClassName(){
     super();
}
```

The super keyword will be covered in detail in a future session when more detailed discussion on object oriented programming is necessary.

This is also the perfect time to properly define the static modifier, anything with the static keyword means it belongs to the class not the object. So if you define a static method, you do not need to declare and instantiate an object to use it. You can simply just call the method directly. If a variable has the static method then it also belongs to the class, so if you were to use a method that changed the value of that variable it would change for all of the objects of that type present and future.

If you are using objects, then an important concept to know is the dot operator "**.**" This is the operator used to call methods on an object and access its attributes. See the following example:

```
int value = 450;
public void updateValue(){
     //code to run
}
obj1.updateValue();
obj1.value
```

Objects in Java can inherit methods and the likes from other classes that they extend, all classes implicitly extend Object. Think of this inheritance like with you, you inherit certain genetic traits from your parents and your parents from your grandparents and so on. Like previously mentioned certain methods are inherited by default, but what if you want your own implementation of that method? This is where method overriding comes in. To do this all you have to do is define your own method with the exact method signature as the one you are trying to override. It is as simple as that!