Fourth Lesson: Intro to Java

Welcome back, y'all!
As you might expect, this week's lesson is on Java! Luckily for us, computing languages all generally have the same functions, just under different names. Now that you understand the basics of Python coding, it will be easier to transition to using Java since you will already know some stuff!

Setting Up

A good coding app for Java beginners is BlueJ. Once you install the system, just click "New Class" and you can start! Generally, you can just delete any of the starter code and write some of your own. Eclipse also works, although the aesthetics of it may leave something to be desired.

Terms to Keep In Mind

These terms will be mentioned in this lesson, but we won't go into detail about it—some will be covered in later lessons.
Package - Java "mechanism" that encapsulates a group of similar/related class, subpackages, and interfaces (focus on classes for now)
Subclass - a class that inherits attributes and/or methods from a "parent" class → a child class

Classes and Objects

Classes are the basic components in Java, and they can be referred to as a "blueprint" for creating objects. These are *general*, broad "definitions" of something that you will then turn into something more distinct, and are generally put into their own separate files. Everything you create in Java will have at least one class, and it will look a bit like this:

```
public class ClassName {}
```

Note: the curly braces are really important; you'll put everything that goes inside the class in there, so make sure you include them if they aren't put there for you!

Objects are instances of a class. As an instance, an object is a very specific thing that you are essentially "building" from the blueprint that is your class, like a particular-style house from the blueprint all houses are based off of. To put this into context, let's make our own class and object. Let's create a general class called "Person."

```java
public class Person {}
```

This Person class will include methods (more detailed explanation and implementation explained below!) that define that class. In our example, we could have a walking and speaking method, to show that people can walk and talk.

```java
public class Person {
     void speak(); //method
     void walk(); //method
}
```

Now, to create an object. First, this would have to be done either in a main method or another class. We briefly covered main methods in the Third Lesson, but they are much more important in Java since you cannot run any code without it. But we'll ignore that for right now and just go to how to declare and instantiate your object.

Assuming we've already created a main method or some other class, we are now ready to declare our object. To do this, you have to first state the type (either a data type or the name of your class)—which is our Person class—and then the name of your object; we can set it as "Emma," since we are creating a specific person: me.

```java
Person Emma;
```

By doing this, we are letting the computer know we will be using "Emma" to refer to a Person data type.

Now that we've declared our object, we can choose to instantiate it. You will have to instantiate your object before you can use it, but there's some flexibility as to when you do this—as long as it is before you use it (it's a bit more complicated than this, of course). When you are instantiating, you are allocating a piece of memory for your object, and it uses the `new` operator to indicate this. Here is where you are actually creating the object.

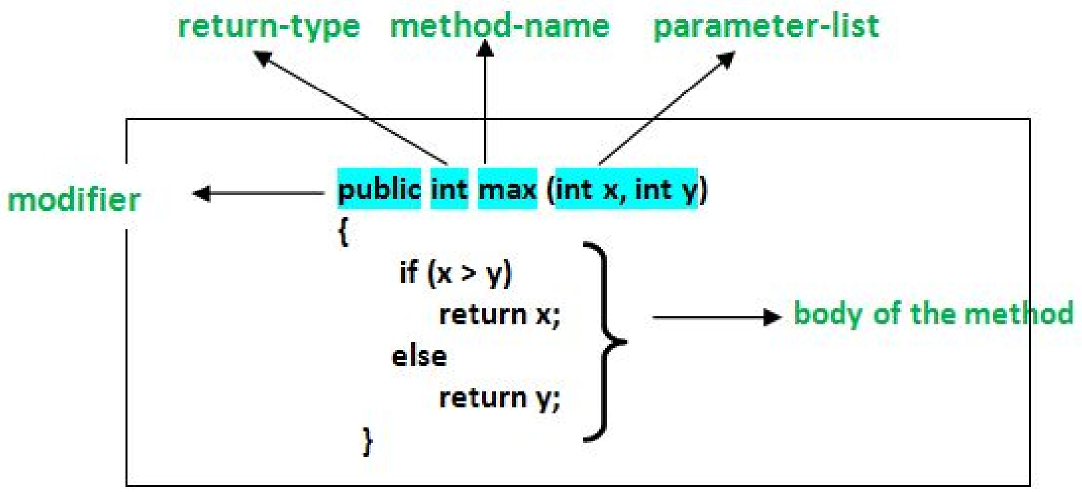
```java
Person Emma = new Person();
```

Objects can access the methods of a class since it is basically a specific copy of the class (*Note: it is not a parent-child relationship; that is a different concept altogether*). Thus, I could call on the speak method from the Person class to make me talk. Let's get into methods so everything makes a bit more sense.

Methods
In Java, methods are the directions or instructions of the blueprint; they are the part of the code that actually “does stuff,” and run when they are called. Along with classes and objects, these are pretty fundamental principles which are imperative to understanding Java.

As previously mentioned, Java has a main method from where the program executes. It’s incredibly important because it is the starting point for all Java applications. It appears within a class (as methods do), and is declared through a very distinct line of code:
```
public static void main(String[] args){
}
```
Remember that when creating programs, you should maintain the proper white space and code to make your code readable. Additionally, although a very minor error, make sure all of your “open” brackets have a matching “close” brackets!

In general, there are several parts to creating a method.



```
public int max (int x, int y)
{
    if (x > y)
        return x;
    else
        return y;
}
```

Modifiers define the access type of your method: how accessible your method is to the rest of the class. There are four general types.
- `public` - can be accessed by all classes in the application
- `protected` - accessible within the class it is defined in *plus* its subclasses
- `private` - accessible only in the class in which it is defined
- `default` (no modifier) - can be accessed by the class and any other class within the same package

You will probably see public and private (or default) most often in the beginning, at least.

Return type indicates the sort of value that your method will be returning. You can think of Python functions and how they will sometimes return values; Java is the same, though you have to clearly define the *type* of the return value within your method declaration. Data types (explained in more detail in **Data Types**) like int and String work. If your method doesn't return anything, you would use the keyword `void`.

Method-name is self explanatory: it is the name of your method. In our Person class example, we had two methods called `speak()` and `walk()`, which both had a return type of `void`.

Parameter list is similar to how you defined a function in Python. A lot of the time, methods require information from outside of itself to successfully run, and this is why you need parameters. Unlike in Python, however (you'll soon find that Java makes almost everything much more complicated), you have to clearly define the type of each of your parameters. If we use the multiplication example from last time, you'd have to use `int num1` and `int num2`.

Within the curly braces is where you will put the "do stuff" code (methods *must* have curly braces if you want to put code inside of them). If I wanted to print "Hello!" in the speak() method, I would put them inside the curly braces.

```
void speak(){
     //print hello
}
```

To call a method, you first have to create an object of the class in which your method is defined. Once you have done this, you would call on your method using the format

object_name.method_name(parameters).

Again using the Person Emma example, if I wanted myself to speak, I would do this:

```
Emma.speak()
```

For the parameters, unless you are creating new objects, you do not need to put the type since you'll already have done that earlier when you created the objects.

Commenting

As a recap, even though commenting seems tedious, it's useful when someone unfamiliar with your code looks at it and can easily understand it.

Commenting in Java is a bit different from Python. While Python utilized hashtags, Java uses forward slashes. A single line comment will begin with `//`, whereas a multi-line comment begins with `/*` and ends with `*/`.

Data Types

Java is considered a “strongly typed” language, meaning that every value type has to be defined when it’s first created, and that it can’t be changed later on. The basic ones are int, double, boolean, and String.

```
int num = 8;    // In Python, always always always end your line
                    with a semicolon.
double num2 = 8.5;
boolean isOdd = false;    // Notice how the values false and true
                         are not treated as Strings.
String num3 = “9”;  // Even though 9 is an integer, because it’s
                    written in the form of a string, it’ll be
                    treated as text.
```

If you want to print something out into the console, you’ll use the code:

```
System.out.println()
```

For example, if you wanted to print the typical “Hello World” statement, you would code something like:

```
public class HelloWorld {
     public static void main(String[] args){
          String statement = “Hello World”;
          System.out.println(statement);
          /* you could definitely combine the two into one line
          of code like so:
          System.out.println(“Hello World”);
          But for educational purposes...
          */
     }
}
```

Otherwise, the basic mathematical operations (like `+, -, *, /`) that we discussed in Python remain the same. Remember that order of operations does matter, so use parentheses if you want to specify something else.

Boolean operators (like `<, >, <=, >=, ==, !=`) all remain the same! If you need to refresh your memory on those, just check out the first Python lesson.

WARNING!!! This was previously mentioned, but will be stated again in red because it is extremely important. Java is much more of a stickler than Python regarding the syntax. While Python does not require you to end statements with a semicolon, Java *does*, and your program will not run if you are missing even one semicolon. The body of classes and methods must be encased in curly braces, as do if statements (shown below). Don't forget, or you may spend hours debugging for a missing semicolon.

Conditional Statements

The basic format of conditional statements is similar to Python. Java utilizes `if`, `else if`, and `else`, which match with `if`, `elif`, and `else` respectively. One thing you will find different is that, in Java, the conditional statement for if statements must be encased in parenthesis (), and the body of the if statements must be encased in parentheses.
Compare Python to Java and see if you can spot the difference. This basically summarizes a lot of the differences we covered today.

PYTHON:

```
a = 33
b = 54
if b > a:
     print(b)
elif b == a:
     print("=")
Else:
     print(a)


Output: 54
```

JAVA:

```
int a = 33;
int b = 54;
if (b > a){
     System.out.print(b);
} else if (b == a){
     System.out.print("=");
} else {
     System.out.print(a);
}

Output:54
```

Java also has `||` and `&&`. These are shorthands and symbolize "or" and "and," respectively, which in Python would just be "`or`"and "`and`". They're particularly nice because when they're neater and one of the only times Java syntax is shorter than Python's.

This concludes the fourth lesson and the very first intro to Java! We hoped you were enlightened; Java may be more difficult to grasp at first, but don't be discouraged. Look forward to the follow-up for this lesson in one/two weeks.
-Emma and Nicole