

Remember "Reviewing Java" has many more examples and topics.

## **Tutorial 2** **Java Classes**

### **C++ vs. Java**

Java is similar to C++ in many ways. The first is the syntax of the language. Semicolons are required to end lines just like C++. There are also many similar structures such as arrays, classes and functions (or methods as they are called in java).

There are differences however. Most noticeably, java is strictly an **Object Oriented Language**, meaning it is made up entirely of classes (objects). There are no exceptions to this in this language.

Another big difference between the two is that there is no pointers in java! Everything is done "behind closed doors" when dealing with addressing and arrays.

Let's continue on and see how to define a simple program.

### **Classes**

Since java is made up entirely of classes, all programs will be made up of classes. The overall rule is that whatever your class name is, the name **MUST MATCH YOUR FILENAME**. If not, when compiling it, there will be an error.

Here is the simplest program that can be made:

```
public class Hello{
    public static void main(String args[]){
        System.out.println("Hello!");
    }
}
```

Wow! Short and sweet. It will print out a "Hello" message on screen. In this program of just 3 lines, so much has happened. Let's observe.

First, there are no include statements like C++. In java, a program can be written without anything in the head of the document.

Second, the "public class Hello" is of the following format:

*keyword class name*

Remember "Reviewing Java" has many more examples and topics.

where keyword is either public, private or protected, and name is the name of the class. The keyword **public** allows any java class to view the data (hence being public). There is a rule though; the class with the main method that will get the program running **MUST** be declared public. To declare a class **private** is a silly thing to do but not invalid either. Private is just the opposite of public in the sense that you cannot use it directly between classes. Variables or objects declared private can only be used within their own class.

Thirdly, the main method declaration is always of that heading. This is not something you can change in a program. The main method will take a single argument, a String array called args[] which will take, as it's elements in the array, any command-line argument you give it. The array is of the object type String (see tutorial 3).

Finally, the System.out.println() statement will be used for output. The println() means "print line", as this will print a string(s) and go to the next line. This is like the C++ "endl". To be more general, any output statement is contained in the System.out set of classes. There is also a System.out.print() that will not print a new line.

## Command-line arguments

As mentioned before, java can take command line arguments. Each of the arguments will be separated by a single space when entered on the command line. There is another rule; each argument is treated AS A STRING. Say that you need a number as an argument. Then, the number must be parsed to that type. The example below will demonstrate command line arguments.

### Example 1: Command line arguments

The purpose of this program is to demonstrate the use of command line arguments. Here, the user will enter their full name (first and last) followed by their age.

```
public class Arguments{

    public static void main(String args[]){

if(args.length < 3){
    System.out.println("Too few arguments.");
    System.out.println("Enter the first name,");
    System.out.println("Then the last name,");
    System.out.println("Then the age.");
}
```

Remember "Reviewing Java" has many more examples and topics.

```
        System.exit(1);
    }

    //correct input so preform the operations:

    String name = args[0] + " " + args[1];
    int age = Integer.parseInt(args[2]);

    System.out.println("Hello " + name + ". You are "
        + age + " years old!");
    }
}
```

Again, a short program but a lot happening. Let's look at some new things introduced here.

First, the if statement will check if the amount of arguments in the array is less than the number required. If true, then the series of instructions will be printed out, then the program will terminate. The exit statement is again contained in the System class. If the statement is false, it is implied that there are the correct number of arguments so proceed with the program.

Next, there are two variables at work here, a string variable called *name* and an integer variable called *age*. Let's look at the name variable. Notice that it will **concatenate** both args[0] and args[1] in addition to a space, by the + operator. In java, the + operator functions as the String concatenator as well as the arithmetic operator.

Next, the age variable will need to be parsed to an integer type. The third argument args[2] will contain the number for the age. As mentioned earlier, the parse method of a class will convert the argument to that type. Most numeric classes have this (Double, Float, Integer etc...). Notice that the name of the class begins with an uppercase letter. Notice the same for the String class. That is a feature of java in the sense of class names. These are called **wrapper classes**.

Finally, the program will output a simple message to the screen displaying the full name and age of the user.

## Compiling programs

Now that we have our first program in java, we need to know how to compile it. The first thing is to know that java is run on a **virtual machine**. A virtual machine is universal to a computer and can run code for a specific task. or programming language.

Remember "Reviewing Java" has many more examples and topics.

Here is how to compile the java program:

```
javac name.java
```

where javac is the java compiler itself, and *name* is the appropriate name of the program ( here Arguments). By pressing the enter key, the program will compile and will translate into virtual machine instructions.

Once successfully compiled, run the following command in the same directory where you compiled the program.

```
java Arguments firstName lastName age
```

where firstName is to be replaced by your first name, lastName is to be replaced with your last name, and age is to be replaced with your age. Notice that there is no "javac" but "java". The difference is that "java" will run the program while "javac" will compile it.