

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

Tutorial 1 **Some Java Basics**

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.

It is assumed that you should know the basics of C++ before learning java (this is only an assumption). This tutorial will highlight only briefly, some concepts from C++ that appear in Java.

Variables

Every programming language contains variables. By definition, **variables** are blocks of memory that are used to store information.

There are numerous ways to use and declare variables in Java. Here is the general way to declare them:

```
type varName = [new] initial_value;
```

where, in the above, **type** is the appropriate data type of the variable (i.e int, double, char, object etc...), **varName** is a **useful** name for your variable, and the initial value will vary when given the data type. The word new is in square brackets as that is only needed when dealing with objects.

In java, here is a chart of what are called **primitive** data types:

Type	Description	Size
<code>char</code>	A Unicode character. Signed values: 0 to 65,535	16 bits

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

<code>short</code>	Short Integer. Signed values: -32,768 to 32,767	16 bits
<code>int</code>	Standard integer. Signed: -2,147,483,648 to 2,147,483,647	16 bits
<code>long</code>	Long Integer. Signed: -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807	64 bits
<code>boolean</code>	Boolean value. Either true or false. Acceptable as 1 or 0 respectively.	1 bit
<code>float</code>	Floating point number. Should not be used for precise values such as currency.	32 bits
<code>double</code>	Double precision floating point number. Should not be used for precise values such as currency.	64 bits

In Java, variables can be either local to a method or global to an object (see tutorial 3).

Logic

Here is chart of the logical units of Java. These will be seen as these tutorials continue. This is also paired with if statements as seen in the next section below.

Java Symbol	Name	Definition
<code>&&</code>	Logical And	ALL inputs must be true in order for the statement to be true
<code> </code>	Logical Or	ALL inputs must be true in order for the statement to be true
<code>!</code>	Logical Not	Will negate any logical statement (True becomes False and False becomes True)
<code>==</code>	Logical Equals	Will detect if BOTH side of the statement have the same truth value
<code>!=</code>	Logical Not Equal	Will detect if BOTH side of the statement DO NOT have the same truth value
<code>></code>	Strictly greater than	Will detect if one side of the statement is STRICTLY bigger than the other
<code>>=</code>	Greater than or equal to	Will detect if one side of the statement is bigger than or equal to the other

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

<	Strictly less than	Will detect if one side of the statement is STRICTLY less than the other
<=	Less than or equal to	Will detect if one side of the statement is less than or equal to the other
%	Modulo Division	The remainder (or residue) when two numbers are divided. As an example, 10 % 3 is 1
++	Increment	The value of a variable is incremented by 1. This can be short for (using a variable x): x = x + 1; This can also be used as either a "pre" or "post" increment.
--	Decrement	The value of a variable is decremented by 1. This can be short for (using a variable x): x = x - 1; This can also be used as either a "pre" or "post" decrement.

If statements

Any programming language requires some decision making. In Java, these are accomplished with **if statements**. These are formatted the same as C++.

```
if( condition(s) ){  
    //code for the if  
}
```

where *condition(s)* is a logical true/false statement.

There may also be an *if / else if* statement. This kind of structure will check for a condition initially and **if and only if** that initial condition is FALSE, the else if clause is checked. If the *else if* clause is FALSE and there are no other statements to check, nothing will happen. There is no limit to how many *else if* clauses you may have. Here is what that may look like:

```
if( condition(s) ){  
    //code for the if  
}else if( condition(s) ){  
    //code for the else if  
}  
.  
.  
.  
}else if( condition(s) ){  
    //code for the else if  
}
```

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

To keep a programs logic short and sweet, a more simplistic way of doing things is to make an if / else block. Here, the *else* clause is a general condition and will handle actions when the initial if is FALSE. Here is what that would look like:

```
if( condition(s) ){
    //code for the if
}else{
    //code for the else
}
```

Notice that there is only 1 else clause and that it does not contain any conditions.

In short, the rule about if statements is that the condition(s) inside the if **MUST BE TRUE** in order for the if statement to work (or be "tripped" as some say). If it is FALSE, the else or else if statements are checked.

While statements

A **while** statement is one kind of loop structure. By its definition, *while* a certain logical condition is TRUE, do something. Below is how it is formatted:

```
while( condition(s) ){
    //code here
}
```

For loops

Another kind of loop is a **for loop**. This means that **for** a certain number of times, do something:

```
for( loop counter; condition(s); counter changes ){
    //code for the for
}
```

Let's break down what each section of the for loop mean. Note that there are 3 sections, each of which (except the last) is separated by a semicolon.

First, the loop counter. The loop counter is used to keep track of how many times you want the loop to run. It is always a numeric value such as an int, double or float. What you do with the counter will come later.

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

Second, the condition(s) of the loop. This is where the logic comes into play. This is logic generally done on the loop counter. Say you wanted the loop to run 30 times. The counter would have logic that kept track of that (counter < 30).

Finally, the counter changes. Without this, you would have AN INFINITE LOOP!!! Those are quite a pain in the neck as the loop WILL NEVER END!!! Generally, you will either ++ or -- a loop but not always. Sometimes, a program calls for a loop to have a change by 2, 3, 5 etc. Just use the correct operations.

Others

As these tutorials continue, classes & objects will most definitely be discussed in addition to arrays and methods (in C++ lingo, a function).