String Literals

The string literal is always enclosed in double quotes. Java uses a **String** class to implement strings, whereas C and C++ use an array of characters. For example:
"Hello World!"
String message = "Hello World";

Character Literals

Character literals are similar to String literals except they are enclosed in single quotes and must have exactly one character. For example 'c' is a character literal. A backslash is used to denote the non-printing characters such as

Description	**Escape Sequence**
Line feed	**\n**
Carriage Return	**\r**
Horizontal tab	**\t**
Backslash	\\
Single quote	**\'**
Double quote	**\"**

Boolean Literal

A Boolean literal can have either of the values: **true or false**. They do not correspond to the numeric values, 1 and 0, as in C and C++.

Numeric Literals

An integer constant refers to a sequence of digits. There are three types of integers namely, decimal integer, octal integer and hexadecimal integer.

Real Constants

Integer numbers are inadequate to represent quantities that vary continuously, such as distances, heights, temperatures, prices and so on. These quantities are represented by numbers containing fractional parts like 17.548, such numbers are called real (or floating) constants. e.g, 0.0083, -0.75, 423.87

A real number may also be expressed in exponential (or scientific) notation. For e.g, 215.65 may be written as 2.1565e2. e2 means multiply by 10^2. mantissa e exponent. e.g, 7500000000= 7.5E9 or 75E8

-0.000000368 = -3.68E-7

OPERATORS

Operators are used to build expressions. They are described here in several related categories. Operators can be broadly categorised as:

a) **Relational and equality operators**

Relational operators are binary operators that require two operands to work on. The operands can be constants, variables, or expression. The operators are as follows:

> greater than

>= greater than, or equal to
< less than
<= less than, or equal to
! Boolean NOT
!= not equal to

b) **Assignment Operators**

= assignment
^= bitwise XOR and assign
&= bitwise AND and assign
%= take remainder and assign
= subtract and assign
*= multiply and assign
/= divide and assign
|= bitwise OR and assign
>>= shift bits right with sign extension and assign
<<= shift bits left and assign
>>>= unsigned bit shift right and assign

c) **Arithmetic**

- subtraction
x multiplication
/ division
+ addition
% modulo

d) **Bitwise**

| bitwise OR
^ bitwise XOR
& bitwise AND
>> right shift
<< left shift
~ bitwise NOT
>>> unsigned bit shift right

e) Logical

&& (Logical AND)
|| (Logical OR)
! (Logical NOT)

f) Increment and Decrement Operators

++ , --

The operator ++ adds 1 to the operand, while – substracts 1.

Both are unary operators and takes the following form :

++m ; or m++
--m ; or m - -
++m is equivalent to m=m + 1 or m+=1
--m is equivalent to m=m-1 or m- = 1

g) Conditional operator : A ternary operator " ?: " is available in Java to construct conditional expressions of the form

exp1 ? exp2 : exp3

where exp1, exp2 and exp3 are expressions.

The operator ?: works as follows : exp1 is evaluated first. If it is nonzero (true), then the expression exp2 is evaluated and becomes the value of the expression. If exp1 is false, exp3 is evaluated and its value becomes the value of the expression.

e.g, a = 10;
b = 15;
x=(a>b) ? a:b;

x will be assigned the value of b.

-> if (a>b)
x=a
else
x=b

h) Special Operator

Java supports some special operators of interest such as **instanceof** operator and **member selection operator(.).**

i) instanceof operator -> The instanceof is an object reference operator and returns true if the object on the left-hand side is an instance of the class given on the right-hand side. This operator allows us to determine whether the object belongs to a particular class or not.

Example : person instanceof student

is true if the object person belongs to the class student; otherwise it is false.

Dot Operator

The dot operator (.) is used to access the instance variables and methods of class objects. Examples:

Person.age // Reference to the variable age
Person1.salary() // Reference to the method salary

Seperators

Seperators are symbols used to indicate where groups of code are divided and arranged. They basically define the shape and functions of our code.

a) Parenthesis() -> Used to enclose parameters in method definition and invocation, also used for defining precedence in expression, containing expressions for flow control, and surrounding cast types.

b) braces {} -> Used to contain the values of automatically initialized arrays and to define a block of code for classes, methods and local scopes.

c) brackets[ ] -> Used to declare array types and for dereferencing array values.

d) semicolon ;-> Used to separate statements.

e) period . -> Used to separate package names from sub-packages and classes; and also used to separate a variable or method from a reference variable.

Implementing a Java Program

Implementation of a Java application program involves a series of steps. They include

1) Creating the program
2) Compiling the program
3) Running the program

1) Creating the program

We can create a program using any text editor.

```
class Test
  {
  public static void main(String args[])
    {
    System.out.println("Welcome to Java");
    }
 }
```

We must save this program in a file called Test.java ensuring that the filename contains the class name properly. This file is called the source file. Note that all java source files will have the extension java. Note also that if a program contains multiple classes, the file name must be the classname of the class containing the main method.

2) Compiling the Program

To compile the program, we must run the Java Compiler javac, with the name of the source file on the command line.

Javac Test.java

If everything is OK, the javac compiler aerates a file called Test.class containing the bytecodes of the program. Note that the compiler automatically names the bytecode file as

<classname> . class

3) Running the program

We need to use the Java interpreter to run a stand-alone program. At the command prompt, type

java Test

Now, the interpreter looks for the main method in the program and begins execution from there.

Java interpreter reads the bytecode files and translates them into machine code for the specific machine on which the Java program is running.

Java Virtual Machine

The Java virtual machine is a specification for an abstract computer. JVM consists of a class loader and a java interpreter that executes the bytecode. The class

loader loads class files from both the java program and java API for execution by the java interpreter. The Java interpreter may be a S/W interpreter that interprets the bytecode one at a time or, it may be a just-in-time compiler that turns the architectural neutral bytecode into native machine language for the host computer. The Java interpreter may be implemented in H/W.

Command Line Arguments

In command line arguments input provided at the time of execution. Command line arguments are parameters that are supplied to the application program at the time of invoking it for execution.

public static void main(string args[])

Here args is declared as an array of strings (known as string objects). Any arguments provided in the command line (at the time of execution) are passed to the array args as its elements. We can simply access the array elements and use them in the program as we wish. For example,

Java Test BASIC FORTRAN C++ Java

The command line contains for arguments. These are assigned to the array args

BASIC -> args[0]
FORTRAN -> args[1]
C++ -> args[2]
Java -> args[3]

The individual elements of an array are accessed by using an index or subscript like args[i]. The value of i denotes the position of the elements inside the array.

```
class ComTest
  {
  public static void main(String args[])
    {
    int count, i=0;
    String str;
    count=args.length;
    System.out.println("Number of arguments =" +count);
    while(i<count)
       {
       str=args[i];
        i=i+1;
        System.out.println(i + " : " + " Java is" + string + " ! ");
        }
     }
  }
```

Java ComTest Simple Distributed Robust Secure Portable