UNIT-1

Chapter-1
Java History

1.1 Introduction

Java is a general-purpose, object-oriented programming language developed by Sun Microsystems of USA in 1991. Originally called *Oak* by James Gosling, one of the inventors of the language, Java was designed for the development of software for consumer electronic devices. The Java team which included Patrick Naughton discovered that the existing languages like C and C+ + had limitations in terms of both reliability and portability. However, they modelled their new language Java on C and C++ but removed a number of features of C and C++ that were considered as sources of problems and thus made Java a really simple, reliable, portable, and Powerful language.

1.1.1 Java Features

Java has followingfeatures.

- **Compiled and interpreted**
- **Platform independent andPortable**
- **Objectoriented**
- **Robust andsecure**
- **Distributed**
- **Familiar, Small andSimple**
- **Multithreaded andInteractive**
- **High Performance**
- **Dynamic and extensible**

Compiled and Interpreted

First Java compiler translates source code into what is known as *bytecode* instructions, Bytecodes are not machine instructions and therefore, in the second stage, Java interpreter generates machine code that can be directly executed by the machine that is running the Java program.

Platform-Independent and Portable

The most significant contribution of Java over other languages is its portability. Jaya programs can be easily moved from one computer system to another, anywhere and anytime. Changes and upgrades in operating systems, processors and system resources will not force any changes in Java programs. Java ensures portability in two ways. First, Java compiler generates bytecode instructions that can be implemented on any machine. Secondly, the sizes of the primitive data types aremachine-independent

Object-Oriented

Java is a true object-oriented language. Almost everything in Java is an *object.* All program code and data reside within objects and classes. Java comes with an extensive set of *classes,* arranged in *packages that* we can use in our programs by inheritance.

Robust and Secure

Java is a robust language. It provides many safeguards to ensure reliable code. It has strict compile time and run time checking for data types. It is designed as a garbage-collected language relieving the programmers virtually all memory management problems. Java also incorporates the concept of exception handling which captures series errors and eliminates any risk of crashing the system. Java systems not only verify all memory access but also ensure that no viruses are communicated with an applet. The absence of pointers in Java ensures that programs cannot gain access to memory locations without properauthorization.

Distributed

Java is designed as a distributed language for creating applications on networks. It has the ability to share both data and programs. Java applications can open and access remote objects on Internet as easily as they can do in a local system. This enables multiple programmers at multiple remote locations to collaborate and work together on a single project.

Simple, Small and Familiar

Java is a small and simple language. Java does not use pointers, preprocessor header files, goto statement and many others. It also eliminates operator overloading and multiple inheritance. Java is modelled on C and C++ languages. So it is familiar to programmers.

Multithreaded and Interactive

Multithreaded means handling multiple task-simultaneously. Java supports multithreaded programs. This means that we need not wait for the application to finish one task before beginning another.. This feature greatly improves the interactive performance of graphical applications.

High Performance

Java performance is impressive for an interpreted language, mainly due to the use of intermediate bytecode. Java architecture is also designed to reduce overheads during runtime. Further, the incorporation of multithreading enhances the overall execution speed of Java programs.

Dynamic and Extensible

Java is a dynamic language. Java is capable of dynamically linking in new class libraries, methods, and objects. Java can also determine the type of class through a query, making it possible to either dynamically link or abort the program, depending on the response. Java programs support functions written in other languages such as C and
C+ +. These functions are known as *native methods.* This facility enables the programmers to use the efficient functions available in these languages. Native methods are linked dynamically at runtime.

1.1.2 Difference between Java andC

- Java does not include the C unique statement keywords goto, sizeof, andtypedef.
- Java does not contain the data typesstruct, union andenum.

- Java does not define the type modifiers keywords auto, extern, register, signed, and unsigned. -
- Java does not support an explicit pointertype.
- A Java does not have a preprocessor and therefore we cannot use # define, # include, and # ifdefstatements.
- Java does not support any mechanism for defining variable arguments tofunctions.
- Java requires that the functions with no arguments must be declared with empty parenthesis and not with the void keywords done inC.
- Java adds new operators such as instance of and >>> (shift right with zerofill)
- Java adds labelled break and continuestatements.
- Java adds many features required for object-orientedprogramming

1.1.3 Difference between Java andC++

- Java does not support operatoroverloading.
- Java does not have template classes as inC++.
- Java does not support multiple inheritance of classes. This is accomplished using a new feature called"interface".
- Java does not support global variables. Every variable and method is declared with in a class and forms part of thatclass.
- Java does not usepointers.
- Java has replaced the destructor function with a finalize( )function.
- There are no header files inJava.

Java also adds some new features. While C+ + is a superset of C, Java is neither a superset nor a subset of C or C++.

1.2 HARDWARE AND SOFTWAREREQUIREMENTS

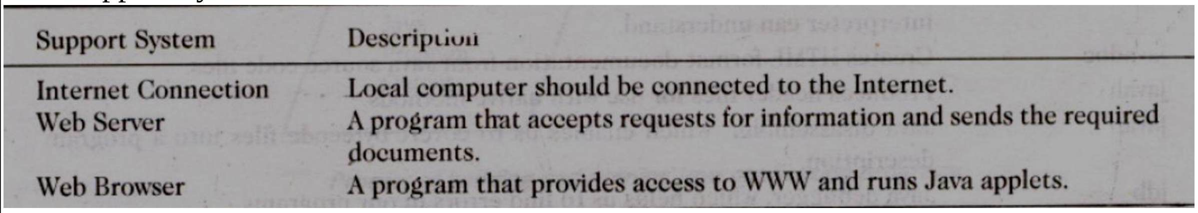
Java is currently supported on Windows 95, Windows NT, Sun Solaris, Macintosh, and UNIX machines. Though, the programs and examples in this book were tested under Windows 95, the most popular operating system today, they can be implemented on any of the above systems. The minimum hardware and software requirements for Windows 95 version of Java areas

- IBM: compatible486system
- Minimum of 8MBmemory
- Windows95software
- A Windows-compatible soundcard, ifnecessary
- A hard drive
- A CD-ROMdrive
- A Microsoft-compatiblemouse

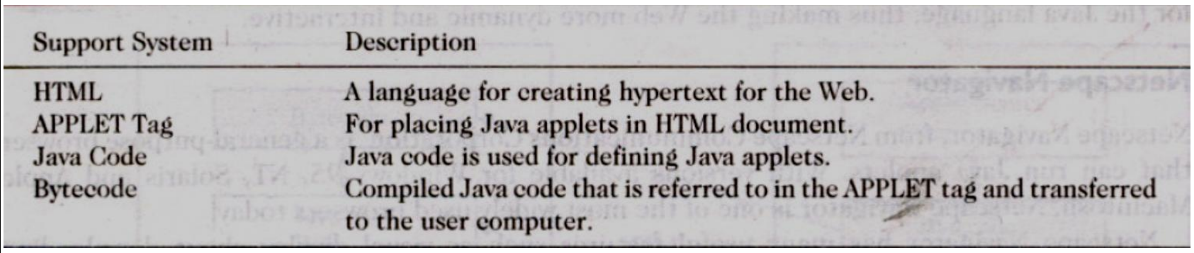
1.3 JAVA SUPPORTSYSTEMS

It is clear from the discussion we had up to now that the operation of Java and Java-enabled browsers on the Internet requires a variety of support systems. The Table lists systems necessary to support Java for delivering information on the internet.

Java Support Systems



Support System	Description
Internet Connection	Local computer should be connected to the Internet.
Web Server	A program that accepts requests for information and sends the required documents.
Web Browser	A program that provides access to WWW and runs Java applets.



Support System	Description
HTML	A language for creating hypertext for the Web.
APPLET Tag	For placing Java applets in HTML document.
Java Code	Java code is used for defining Java applets.
Bytecode	Compiled Java code that is referred to in the APPLET tag and transferred to the user computer.

1.4 JAVAENVIRONMENT

Java environment includes a large number of development tools and hundreds of classes and methods development tools are part of the system known as *Java Development Kit* (JDK) and the classes and methods are part of the *Java Standard Library* (JSL), also known as the *Application Programming Interface* (API).

1.4.1 Java Development Kit

The Java Development Kit comes with a collection of tools that are used for developing and running Java programs. They include:

appletviewer (for viewing Java applets)
javac (Java compiler)
java (Java interpreter)
javap (Java disassembler)
javah (for C header files)
javadoc (for creating HTML documents)
jdb (Java debugger)

appletviewer: Enables us to run Java applets (without actually using a Java-compatible browser).
java: Java interpreter, which runs applets and applications by reading and interpreting bytecode files.
javac: The Java compiler, which translates Java source code to bytecode files that the interpreter can understand.
javadoc: Creates HTML-format documentation from Java source code files.
javah: Produces header files for use with native methods.
javap: Java disassembler, which enables us to covert bytecode files into a program description.
jdb: Java debugger, which helps us to find errors in our programs.

To create a Java program, we need to create a source code file using a text editor. The source code is then compiled using the Java compiler javac and executed using the Java interpreter java. The Java debugger jdb is used to find errors, if any, in the source code. A compiled Java program can be converted into a source code with the help of Java disassembler javap.

1.4.2 Application ProgrammingInterlace

The Java Standard Library (or API) includes hundreds of classes and methods grouped into several functional packages .Most commonly used packages are:

Language Support Package: A collection of classes and methods required for implementing basic features of Java.

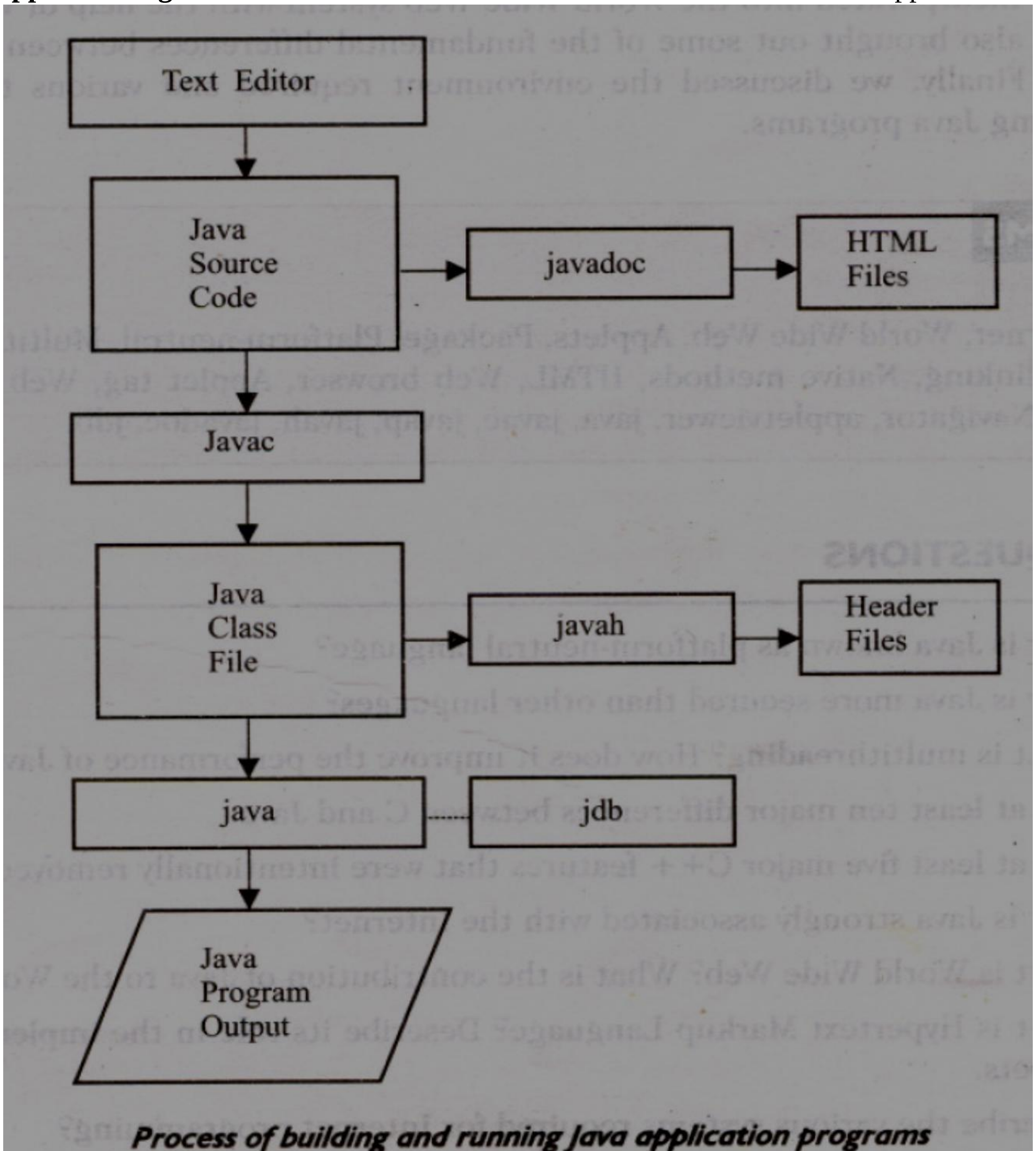
Utilities Package: A collection of classes to provide utility functions such as date and time functions.

Input/Output Package: A collection of classes required for input/output manipulation.

Networking Package: A collection of classes for communicating with other computers via Internet

Awt Package: The Abstract Window Tool Kit package contains classes that implements platform-independent graphical user interface.

Applet Package: This includes a set of classes that allows us to create Java applets



Process of building and running Java application programs

.

Chapter-2
LANGUAGE REFERENCE

Overview of Java Language
2.1 Introduction

Java is a general-purpose, object-oriented programming language. We can develop two types of java programs:

- Stand-alone applications
- Web applets

Stand-alone applications are programs written in Java to carry out certain tasks on a stand-alone local computer.
Executing a stand-alone Java program involves two steps:
1. Compiling source code into byte code using javaccompiler.
2. Executing the bytecode program using javainterpreter.

Applets are small Java programs developed for Internet applications. An applet located on a distant computer (Server) can be downloaded via Internet and executed on a local computer (Client) using a Java-capable browser. We can develop applets for doing everything from simple animated graphics to complex games and utilities. Since applets are embedded in an HTML (Hypertext Markup Language) document and run inside a Web page, creating and running applets are more complex than creating an application. Stand-alone programs can read and write files and perform certain operations that applets cannot do. An applet can only run within a Web browser.

A simple Java program

```
classSampleOne
{
  public static void main (String args[ ])
  {
    System.out.println("Java is better than C++.");
     }
}
```

Class Declaration

The first line

classSampleOne

Declares a class, which is an object-oriented construct. As stated earlier,(Java is a true object-oriented language and therefore, *everything* must be placed inside a class) class is a keyword and declares that a new class definition follows. SampleOne is a Java *identifier* that specifies the name of the class to be defined.

Opening Brace
Every class definition in Java begins with an opening brace "{" and ends with a matching closing brace"}" appearing in the last line in the example.

The main Line
The third line

public static void main (String args[ ])

Defines a method named main. Every Java application program must include the main( ) method. This is the starting point for the interpreter to begin the execution of the program. A Java application can have any number of classes but *only* one of them must include a main method to initiate the execution.
This line contains a number of keywords, public, static and void.

public: The keyword public is an access specifier that declares the main method as unprotected and therefore making it accessible to all other classes.

static: which declares this method as one that belongs to the entire class and not a part of any objects of the class. The main must always be declared as static since the interpreter uses this method before any objects are created.

void: The type modifier void states that the main method does not return any value(but simply prints some text to the screen. All parameters to a method are declared inside a pair of parentheses. Here, String args[] declares a parameter named args, which contains an array of objects of the class typeString.

The Output Line

The only executable statement in the program is System.out.println("Java is better than C++.");
Since Java is a true object oriented language, every method must be part of an object. The println method is a member of the out object,) which is a static data member of System class this line prints the string *Java is better than C++* to the screen. The method println always appends a newline character to the end of the string. This means that any subsequent output will start on a new line. *Every Java statement must end with a semicolon.*

2.2 JAVA PROGRAM STRUCTURE

A Java program may contain one or more sections.

Documentation Section
The documentation section comprises a set of comment lines giving the name of the program, the author and other details, which the programmer would like to refer to at a later stage. Comments must explain *why* and *what* of classes and how of algorithms. Java permits both the single-line comments and multi-line comments. The single-line comments begin with // and end at the end of the line. For longer comments, we can create long multi-line comments bystartingwitha/*andendingwith*/.Javaalsousesathirdstyleofcomment
/**…………*/ known as *documentation comment.* This form of comment is used for generating documentation automatically.

Package Statement

The first statement allowed in a Java file is a package statement. This statement declares a package name and informs the compiler that the classes defined here belong to this package. For example: package student;
The package statement is optional.

Import Statements

The next thing after a package statement (but before any class definitions may be a number of import statements. For example import student test. This statement instructs the interpreter to load the test class contained in the package student. Using import statements, we can have access to classes that are part of other named packages.

Interface Statements

An interface is like a class but includes a group of method declarations. This is also an optional section and is used only when we wish to implement the multiple inheritance features in theprogram.

Class Definitions

A Java program may contain multiple class definitions. Classes are the primary and essential elements of a Java program. These classes are used to map the objects of real-world problems. The number of classes used depends on the complexity of theproblem.

Main Method Class

Since every Java stand-alone program requires a main method as its starting point, this class is the essential part of a Java program. A simple Java program may contain only this part. The main method creates objects of various classes and establishes communications between them. On reaching the end of main, the program terminates and the control passes back to the operatingsystem.

2.3 JAVA TOKENS

A Java program is basically a collection of classes. A class is defined by a set of declaration statements and methods containing executable statements. Most statements contain expressions, which describe the actions carried out on data. Smallest individual units in a program are known as tokens. The compiler recognizes them for building up expressions and statements. In simple terms, a Java program is a collection of tokens, comments and white spaces. Java language includes five types of tokens. They are:

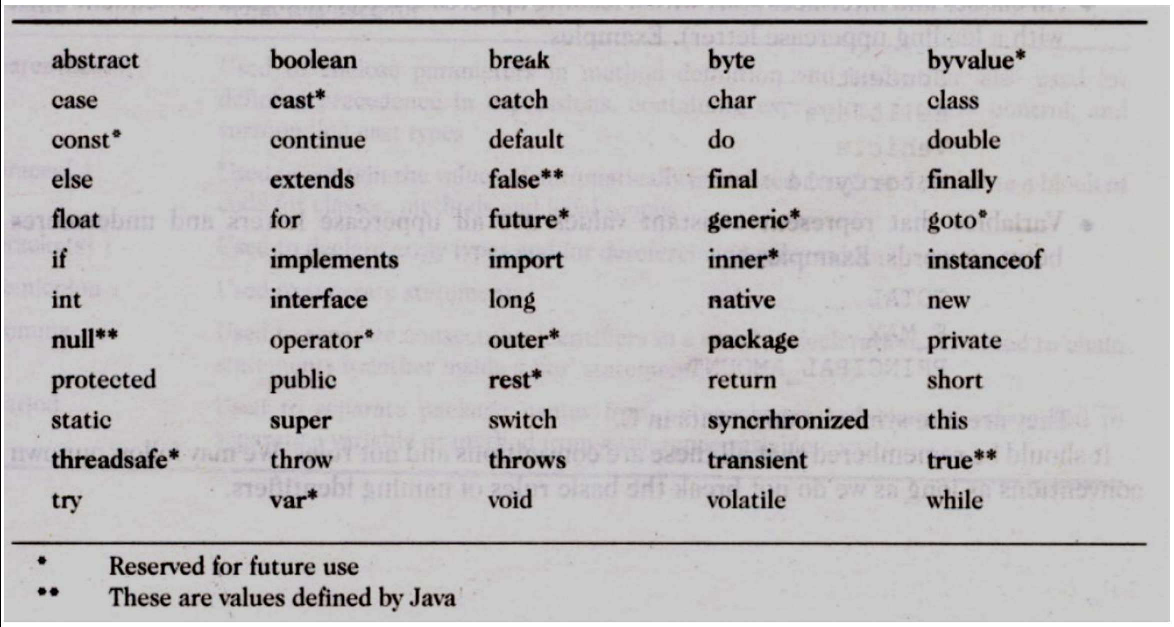
- Reserved Keywords
- Identifiers
- Literals
- Operators
- Separators

Java Character Set

The smallest units of Java language are the characters used to write Java tokens. These characters are defined by the *Unicode* character set, an emerging standard that tries to create characters for a large number of scripts worldwide.
The Unicode is a 16-bit character coding system and currently supports more than 34,000 defined characters. However, most of us use only the basic ASCII characters, which include letters, digits and punctuation marks, used in normal English. We, therefore, have used only ASCII character set (a subset of UNICODE character set) in developing the programs.

Keywords

Keywords are an essential part of a language definition. They implement specific features of the language. Java language has reserved 60 words as keywords. These keywords, combined with operators and separators according to syntax, form definition of the Java language. Since keywords have specific meaning in Java, we cannot use them as names for variables, classes, methods and so on. All keywords are to be written in lower-case letters. Since Java is case-sensitive, one can use these words as identifiers by changing one or more letters to upper case. However, it is a bad practice and should be avoided. The keywords that are unique to Java are shown inboldface.



abstract	**boolean**	**break**	**byte**	**byvalue***
case	**cast***	catch	char	class
const*	continue	default	do	double
else	**extends**	**false****	**final**	**finally**
float	for	**future***	**generic***	goto*
if	**implements**	**import**	**inner***	**instanceof**
int	**interface**	long	native	new
null**	operator*	**outer***	**package**	private
protected	public	**rest***	return	short
static	**super**	switch	**syncrhronized**	this
threadsafe*	**throw**	**throws**	**transient**	**true****
try	**var***	void	volatile	while

* Reserved for future use
** These are values defined by Java

Identifiers

Identifiers are programmer-designed tokens. They are used for naming classes, methods, variables, objects, labels, packages and interfaces in a program. Java identifiers follow the following rules:
1. They can have alphabets, digits, and the underscore and dollar signcharacters.
2. They must not begin with adigit.
3. Uppercase and lowercase letters are distinct.
4. They can be of any length.
Identifier must be meaningful, short enough to be quickly and easily typed and long enough to be descriptive and easily read. Java developers have followed some namingconventions.

1. Names of all public methods and instance variables start with a leading lowercase letter. Examples: average,sum
2. When more than one word are used in a name, the second and subsequent words are marked with a leading uppercase letters. Examples: dayTemperature, firstDayOfMonth, totalMarks
3. All private and local variables use only lowercase letters combined with underscores. Examples: length,batch_strength
4. All classes and interfaces start with a leading uppercase letter (and each subsequent word with a leading uppercase letter). Examples: Student, HelloJava, Vehicle,MotorCycle
5. Variables that represent constant values use all uppercase letters and underscores between words. Examples: TOTAL F _MAX,PRINCIPAL_AMOUNT

It should be remembered that all these are conventions and not rules. We may follow our own conventions as long as we do not break the basic rules of naming identifiers.

Literals

Literals in Java are a sequence of characters (digits, letters, and other characters) that represent constant values to be stored in variables. Java language specifies five major types of literals. They are:
Integer literals

- Floating pointliterals
- Character literals
- Stringliterals
- Booleanliterals

Each of them has a type associated with it. The type describes how the values behave and how they are stored.

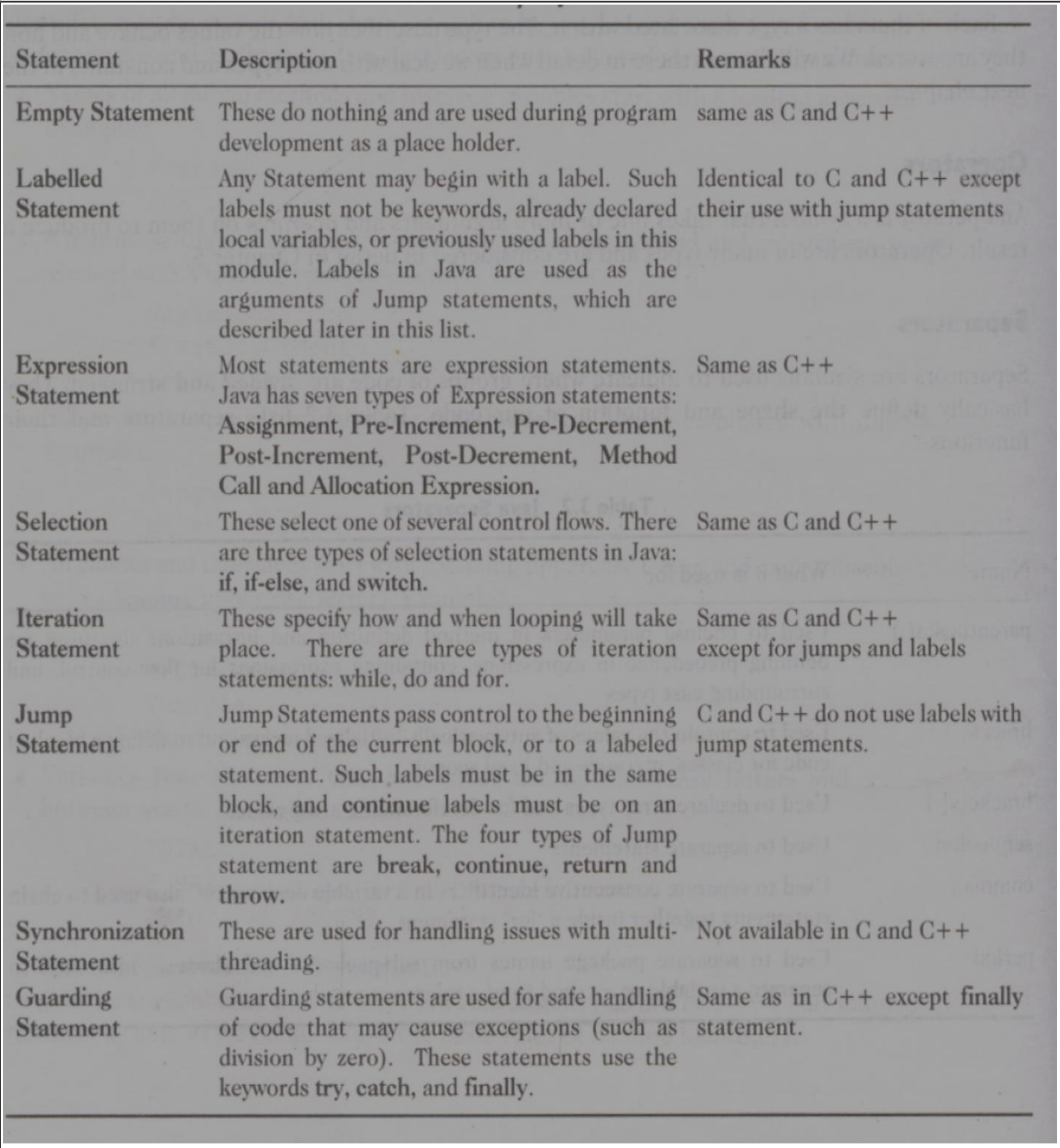
Operators

An operator is a symbol that takes one or more arguments and *operates* on them to produce a result.

Separators

Separators are symbols used to indicate where groups of code are divided and arranged. They basically define the shape and function of our code. Some of the separators are parentheses( ), braces{ },brackets[ ], semicolon; comma , period.
JAVA STATEMENTS
The statements in Java are like sentences in natural languages. A statement is an executable combination of tokens ending with a semicolon (;) mark. Statements are usually executed in sequence in the order in which they appear. However, it is possible to control the flow of execution, if necessary, using special statements. Java implements several types of statements.

Summary of Statements



Statement	Description	Remarks
Empty Statement	These do nothing and are used during program development as a place holder.	same as C and C++
Labelled Statement	Any Statement may begin with a label. Such labels must not be keywords, already declared local variables, or previously used labels in this module. Labels in Java are used as the arguments of Jump statements, which are described later in this list.	Identical to C and C++ except their use with jump statements
Expression Statement	Most statements are expression statements. Java has seven types of Expression statements: **Assignment, Pre-Increment, Pre-Decrement, Post-Increment, Post-Decrement, Method Call and Allocation Expression.**	Same as C++
Selection Statement	These select one of several control flows. There are three types of selection statements in Java: **if, if-else,** and **switch.**	Same as C and C++
Iteration Statement	These specify how and when looping will take place. There are three types of iteration statements: while, do and for.	Same as C and C++ except for jumps and labels
Jump Statement	Jump Statements pass control to the beginning or end of the current block, or to a labeled statement. Such labels must be in the same block, and **continue** labels must be on an iteration statement. The four types of Jump statement are **break, continue, return** and **throw.**	C and C++ do not use labels with jump statements.
Synchronization Statement	These are used for handling issues with multi-threading.	Not available in C and C++
Guarding Statement	Guarding statements are used for safe handling of code that may cause exceptions (such as division by zero). These statements use the keywords **try, catch,** and **finally.**	Same as in C++ except **finally** statement.

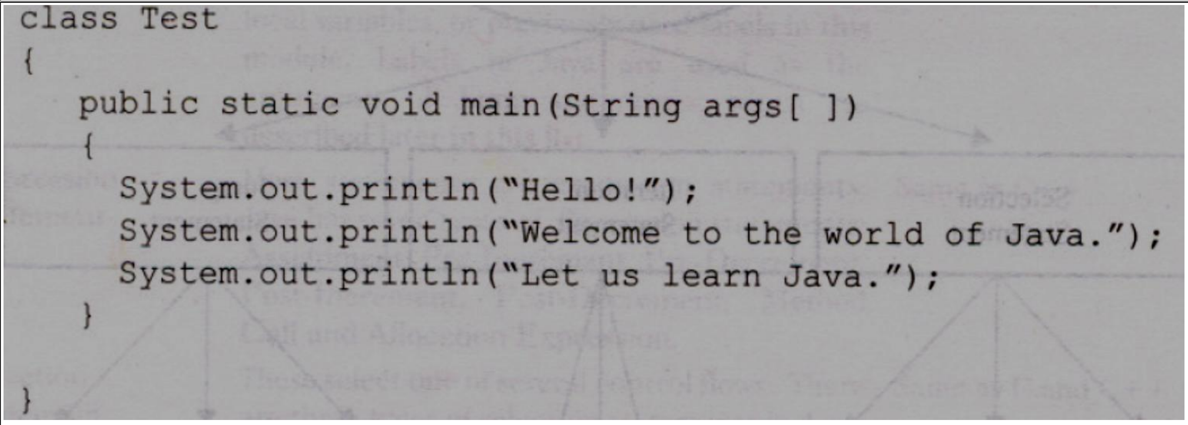
2.4 IMPLEMENTING A JAVAPROGRAM

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

- Creating theprogram
- Compiling theprogram
- Running the program

Creating the Program

We can create a program using any text editor. Assume that we have entered the following code.



```
class Test
{
   public static void main(String args[ ])
   {
     System.out.println("Hello!");
     System.out.println("Welcome to the world of Java.");
     System.out.println("Let us learn 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.* All Java source files will have the extension java. If a program contains multiple classes, the file name must be the class name of the class containing the mainmethod.

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 as *javacTest.java* .
If everything is OK, the javaccompiler creates a file called Test.class containing the bytecodes of the program. The compiler automatically names the bytecode file as *<classname> .class*

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. When executed program displays the following:

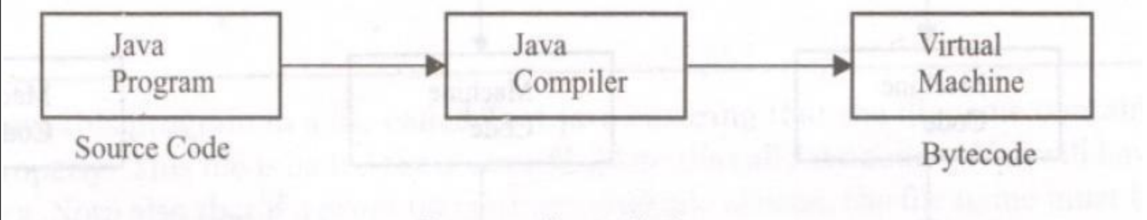
Hello!
Welcome to the world *of* Java.
Let us learn Java.

Machine Neutral

The compiler converts the source code files into byte code files. These codes are machine independent and therefore can be run on any machine. That is, a program compiled on an IBM machine will run on a Macintosh machine. Java interpreter reads the bytecode files and translates them into machine code for the specific machine on which the Java program is running. The interpreter is therefore specially written for each type ofmachine.

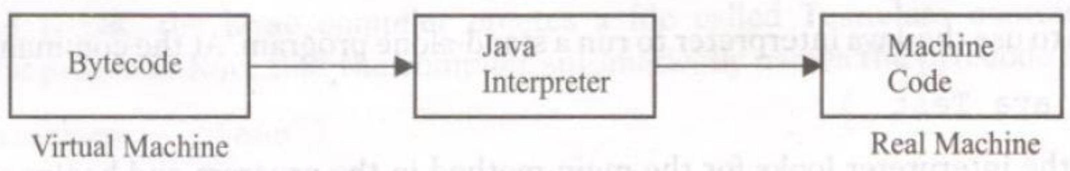
2.5 JAVA VIRTUAL MACHINE

All language compilers translate source code into *machine code* for a specific computer. Java compiler also does the same thing. The Java compiler produces an intermediate code known as bytecode for a machine that does not exist. This machine is called the *Java Virtual Machine* and it exists only inside the computer memory. It is a simulated computer within the computer and does all major functions of a realcomputer.



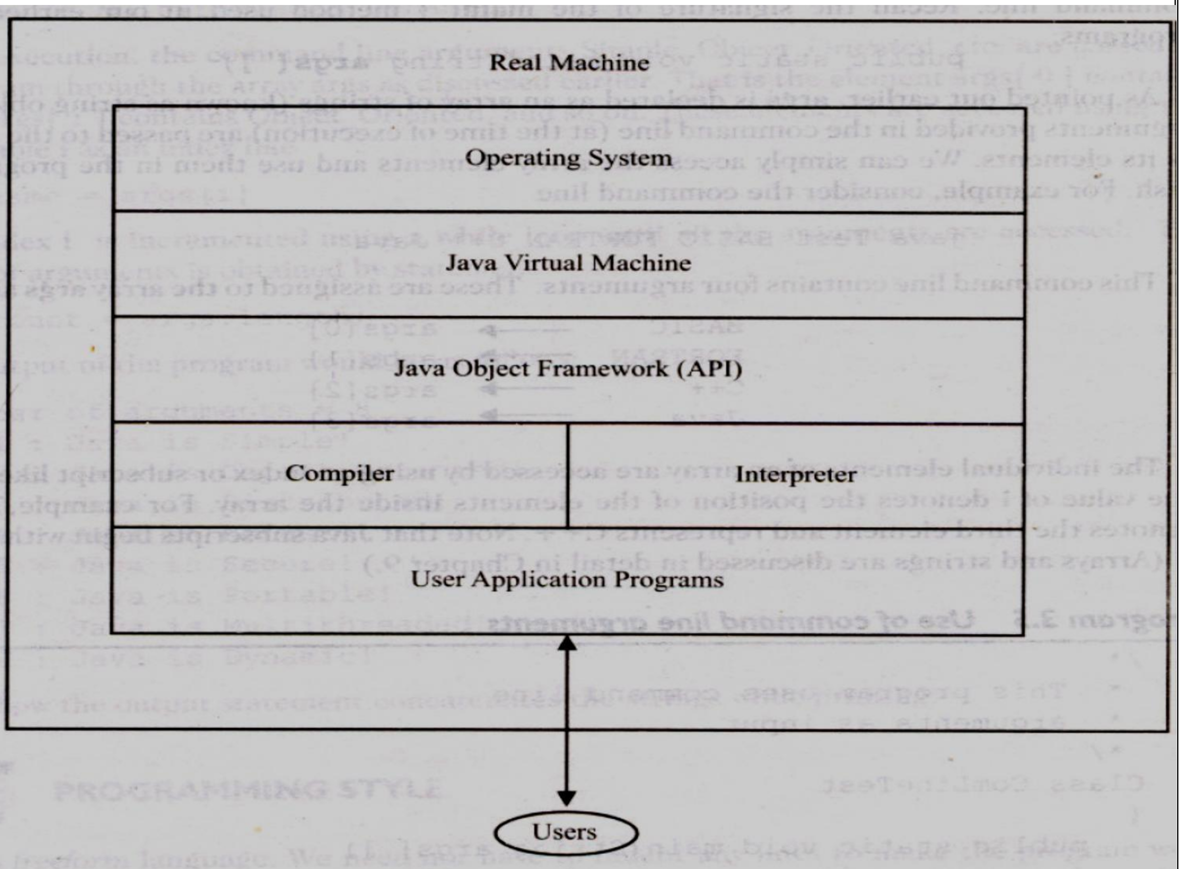
Process of compilation

The virtual machine code is not machine specific. The machine specific code known as machine code) is generated by the Java interpreter by acting as an intermediary between the virtual machine and the real machine.



Process of converting bytecode into machine code

The Java object framework (Java API) acts as the intermediary between the user programs and the virtual machine which in turn acts as the intermediary between the operating system and the Java objectframework.

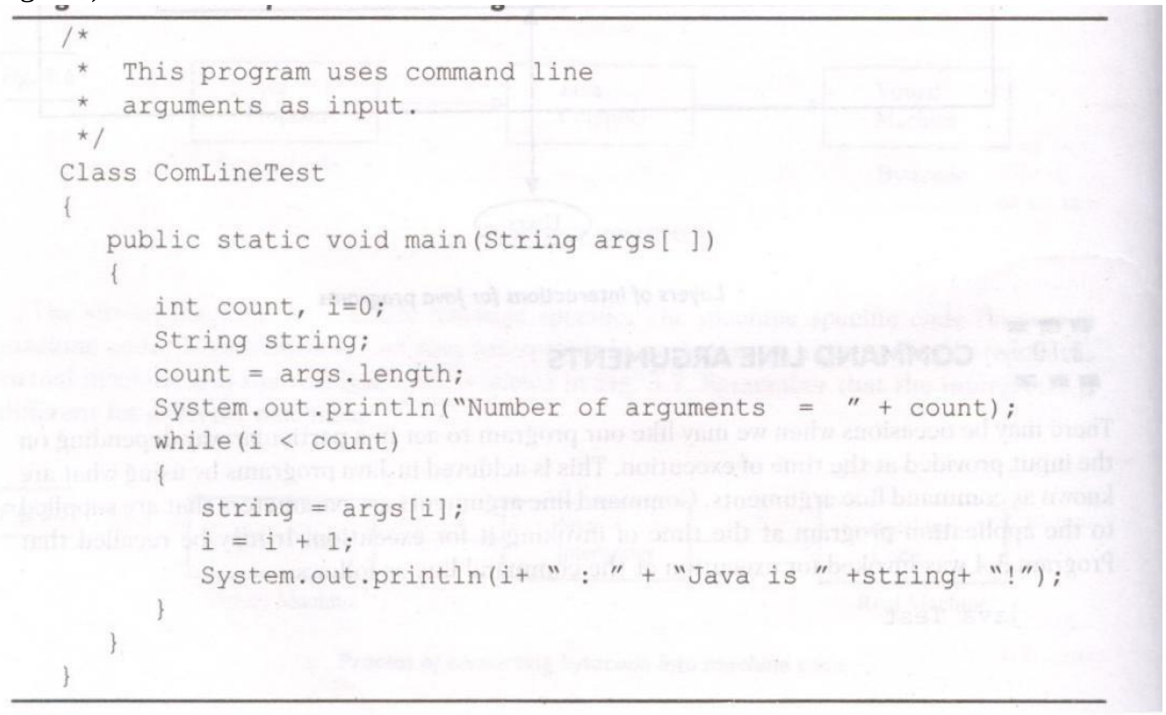


Layers of interactions for Java programs

2.6 COMMAND LINE ARGUMENTS

There may be occasions when we may like our program to act in a particular way depending on the input provided at the time of execution. This is achieved in Java programs by using what are known as command line arguments. Command line arguments are parameters that are supplied to the application program at the time of invoking it for execution.

We can write Java programs that can receive and use the arguments provided in the command line. Consider a main function statement as public static void main (String **args[ ]).** 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. The individual elements of an array are accessed by using an index or subscript like arts [i). The value of i denotes the position of the elements inside the array. For example,
args[2) denotes the third element.



```
/*
 *  This program uses command line
 *  arguments as input.
 */
Class ComLineTest
{
   public static void main(String args[ ])
   {
      int count, i=0;
      String string;
      count = args.length;
      System.out.println("Number of arguments  =  " + count);
      while(i < count)
      {
         string = args[i];
         i = i + 1;
         System.out.println(i+ " : " + "Java is " +string+ "!");
      }
   }
}
```

Compile and run the program with the command line as follows:

javaComLineTest Simple Object_Oriented Distributed Robust Secure
Portable Multithreaded Dynamic

Upon execution, the command line arguments Simple, Object_Oriented, etc. are passed to the program through the array args as discussed earlier. That is the element args[0] contains Simple, args[1] contains Object_Oriented, and so on. These elements are accessed using the loop variable i as an index like name = args[i].The index i is incremented using a while loop until all the arguments are accessed.
The number of arguments is obtained by statement
count = args.length;

The output of the program would be as follows:

```
Number of arguments = 8
1 : Java isSimple!
```

```
2 : Java is Object_Oriented!
3 : Java isDistributed!
4 : Java is Robust!
5 : Java is Secure!
6 : Java isPortable!
7 : Java is Multithreaded!
8 : Java isDynamic!
```

Note how the output statement concatenates the strings while printing.

2.7 PROGRAMMING STYLE

Java is *a freeform* language. We need not have to indent any lines to make the program work properly. Java system does not care where on the line we begin typing. Although several alternate styles are possible, we should select one and try to use it with total consistency Although several alternate styles are possible, we should select one and try to use it with total consistency.
For example, the statement

```
System.out.println("Java is Wonderful!")
```

can be written as

```
System.out.println
("Java is Wonderf6l!");
```
or, even as
```
System
.
out
.
println
 (
"Java is Wonderful!"
) ;
```

2.8 Constants variables and Data Types

INTRODUCTION

A programming language is designed to process certain kinds of *data* consisting of numbers, characters and strings and to provide useful output known as *information.* The task of processing data is accomplished by executing a sequence of instructions constituting a *program.* These instructions are formed using certain symbols and words according to some rigid rules known as *syntax rules* (or *grammar).* Every program instruction must conform precisely to the syntax rules of the language.

CONSTANTS

Constants in Java refer to fixed values that do not change during the execution of a program.

Integer Constants

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

Decimal integers consist of a set of digits, 0 through 9, preceded by an optional minus sign. Valid examples of decimal integer constants are: 123 -321 0 654321

An *octal* integer constant consists of any combination of digits from the set 0 through 7, with a leading O. Some examples of octal integer are: 037 0 0435 0551

A sequence of digits preceded by Ox or OX is considered as *hexadecimal* integer (hex integer). They may also include alphabets A through F or a through f. A letter A through F represents the numbers 10 through 15. Valid examples of hex integers are: OX2 OX9FOxbcdOx
We rarely use octal and hexadecimal numbers in programming.

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 point)* constants. Further examples of real constants are: 0.0083 -0.75 435.36

These numbers are shown in *decimal notation,* having a whole number followed by a decimal point and the fractional part, which is an integer. A real number may also be expressed in *exponential* (or *scientific) notation.* For example, the value 215.65 may be written as 2.1565e2 in exponential notation. e2 means multiply by102.
The general form is: *mantissa* e *exponent.* The *mantissa* is either if real number expressed in *decimal notation* or an integer. The *exponent* is an integer with an optional *plus* or *minus* sign. The letter e separating the mantissa and the exponent can be written in either lowercase or uppercase. A floating point constant may thus comprise fourparts:

- A wholenumber
- A decimalpoint
- A fractionalpart
- Anexponent

Single CharacterConstants

A single character constant (or simply character constant) contains a single character enclosed within a pair of single quote marks. Examples of character constantsare:'5' 'X'
.‘‘
„’

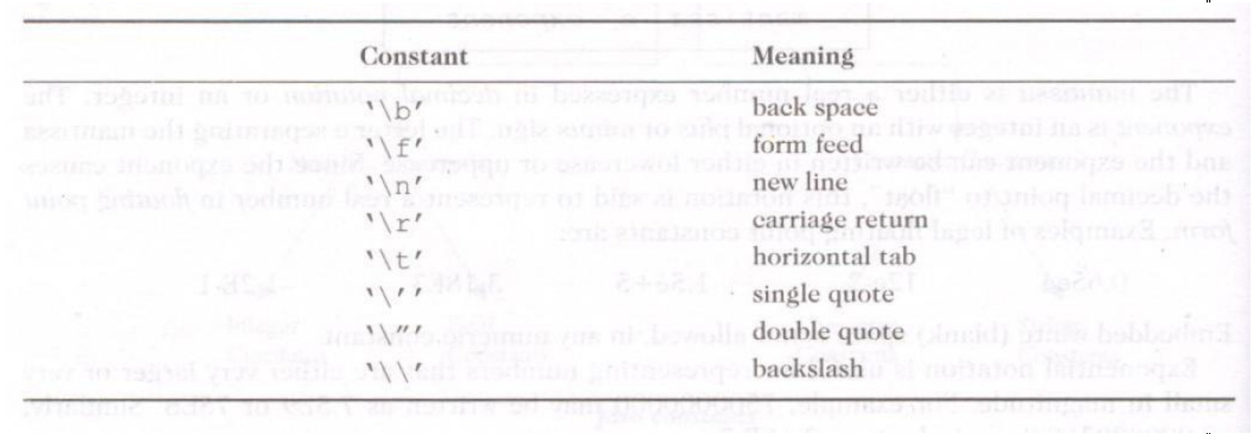
String Constants

A string constant is a sequence of characters enclosed between double quotes. The characters may be alphabets, digits, special characters and blank spaces. Examples are: "Hello Java" "1997" "WELL DONE" "?..!" "5+3""X"

Backslash Character Constants

Java supports some special backslash character constants that are used in output methods. For example, the symbol '\n' stands for newline character. Note that each one of them represents one character, although they consist of two characters. These characters combinations are known *escape sequences.*

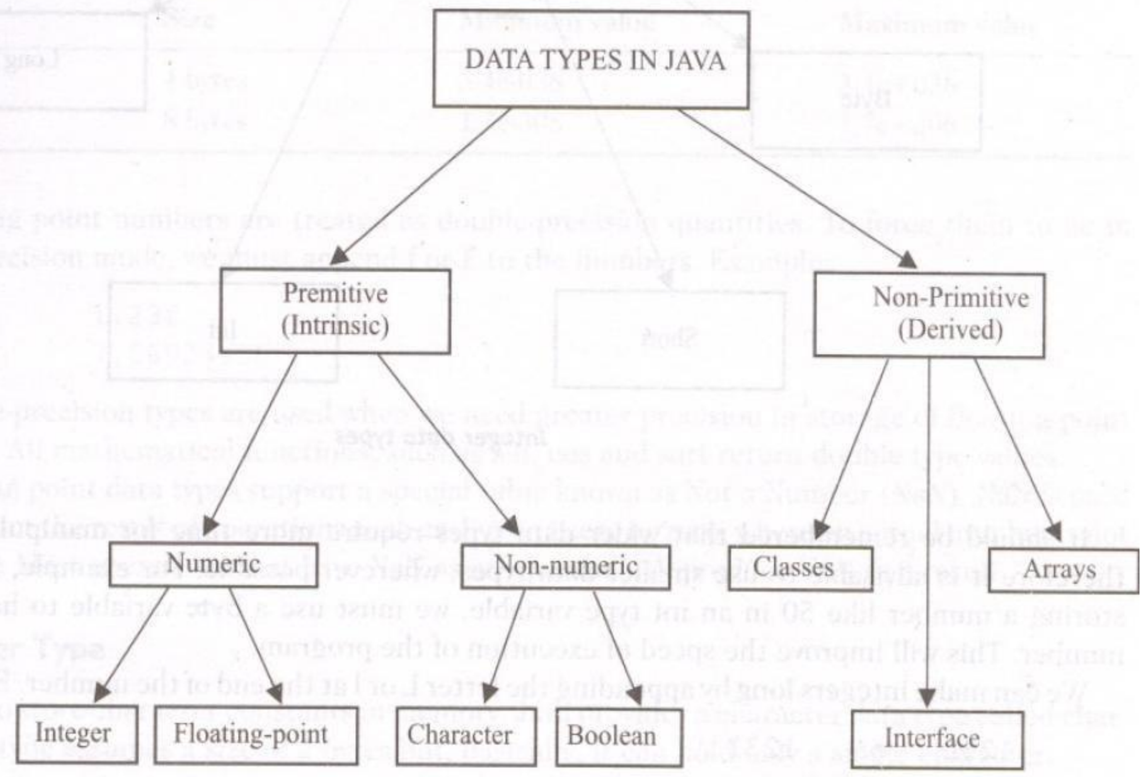
Backslash Character Constants



Constant	Meaning
'\b'	back space
'\f'	form feed
'\n'	new line
'\r'	carriage return
'\t'	horizontal tab
'\''	single quote
'\"'	double quote
'\\'	backslash

2.9 DATATYPES

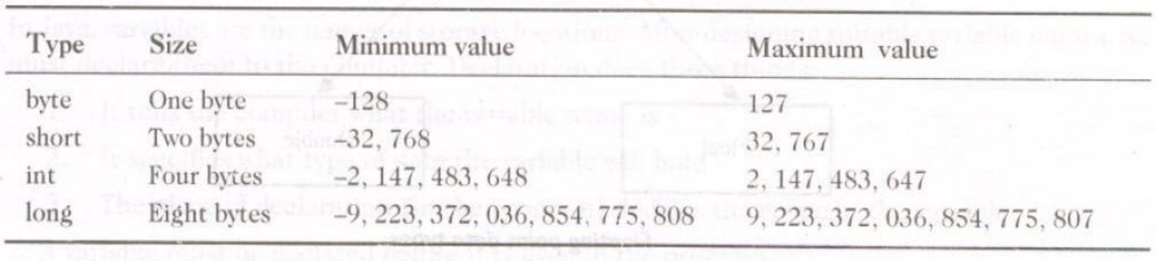
Every variable in Java has a data type. Data types specify the size and type of values that can be stored. Java language is rich in its *data types.* The variety of data types available, allow the programmer to select the type appropriate to the needs of the application.



Data types in Java

Integer Types

Integer types can hold whole numbers such as 123, -96, and 5639. The size of the values that can be stored depends on the integer data type we choose. Java supports four types of integers. They are byte, short, int, and long. Java does not support the concept of *unsigned* types and therefore all Java values are signed meaning they can be positive ornegative.

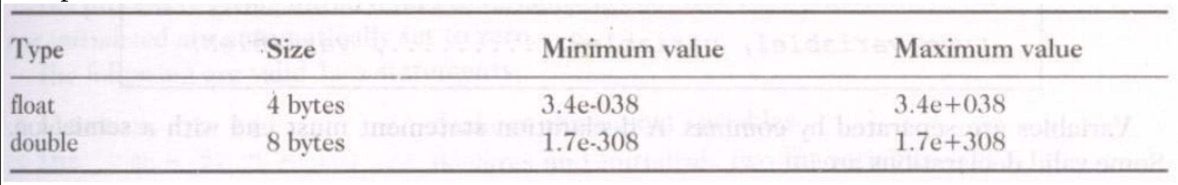


Type	Size	Minimum value	Maximum value
byte	One byte	–128	127
short	Two bytes	–32, 768	32, 767
int	Four bytes	–2, 147, 483, 648	2, 147, 483, 647
long	Eight bytes	–9, 223, 372, 036, 854, 775, 808	9, 223, 372, 036, 854, 775, 807

It should be remembered that wider data types require more time for manipulation and therefore it is advisable to use smaller data types, wherever possible.

Floating Point Types

Integer types can hold only whole numbers and therefore we use another type known *as floating point* type to hold numbers containing fractional parts such as 27.59 and -1. The float type values are *single-precision* numbers while the double types represent *double precision* numbers. Floating point numbers are treated as double-precision quantities. To force them to be in single-precision mode, we must append for F to the numbers. Example:
1.23f 7.56923e5F. Double-precision types are used when we need greater precision in storage of floating point numbers. All mathematical functions, such as sin, cos and sqrt returns double type values. Floating point data types support a special value known as Not-a-Number (NaN). NaN is used to represent the result of operations such as dividing zero by zero, where an actual number is not produced. Most operations that have NaN as an operand will produce NaN as aresult.



Type	Size	Minimum value	Maximum value
float	4 bytes	3.4e-038	3.4e+038
double	8 bytes	1.7e-308	1.7e+308

Character Type

In order to store character constants in memory, Java provides a character data type called char. The char type assumes a size of 2 bytes but, basically, it can hold only a single character.

Boolean Type
Boolean type is used when we want to test a particular condition during the execution of the program. There are only two values that a boolean type can take: true or false. Remember, both these words have been declared as keywords. boolean type is denoted by the keyword Boolean and uses only one bit of storage. All comparison operators return boolean type values. Boolean values are often used in selection and iteration statements.

2.10 VARIABLES

A *variable* is an identifier that denotes a storage location used to store a data value. Unlike constants that remain unchanged during the execution of a program, a variable may take different values at different times during the execution of the program. A variable name can

be chosen by the programmer in a meaningful way so as to reflect what it represents in the program. Some examples of variable names are:*average* *height* *total_height*
Variable names may consist of alphabets, digits, the underscore (-) and dollar characters, subject to the following conditions:
1. They must not begin with a digit.
2. Uppercase and lowercase are distinct. This means that the variable Total is not the same as total orTOTAL.
3. It should not be akeyword.
4. White space is not allowed.
5. Variable names can be of anylength.

DECLARATION OF VARIABLES

In Java, variables are the names of storage locations. After designing suitable variable names, we must declare them to the compiler. Declaration does three things:
1. It tells the compiler what the variable nameis.
2. It specifies what type of data the variable willhold.
3. The place of declaration (in the program) decides the scope of thevariable.

A variable must be declared before it is used in the program. A variable can be used to store a value of any data type. That is, the name has nothing to do with the type. Java allows any properly formed variable to have any declared data type. The declaration statement defines the type of variable. The general form of declaration of a variableis:

```
type variable1, variable2, ........., variableN;
```

Variables are separated by commas, A declaration statement must end with a semicolon,
Some valid declarations are:
int   count;
float x, y;
doublepi;
GIVING VALUES TO VARIABLES

A variable must be given a value after it has been declared but before it is used in an expression.
This can be achieved in two ways:
1. By using an assignmentstatement
2. By using a read statement

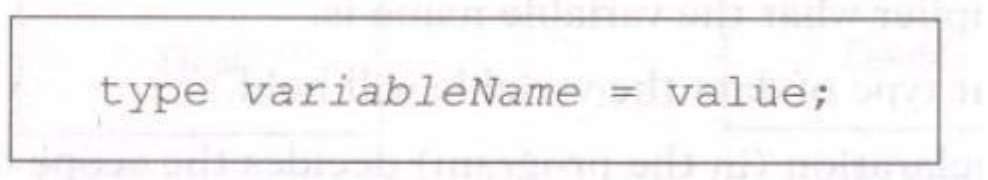
Assignment Statement

A simple method of giving value to a variable is through the assignment statement as follows

```
variableName  =  value;
```

For example: initialValue = 0;

It is also possible to assign a value to a variable at the time of its declaration. This takes the form



```
type variableName = value;
```

For Example: intfinalvalue=100;

Read Statement

We may also give values to variables interactively through the keyboard using the readLine().The readLine( ) method reads the input from the keyboard as a string which is then converted to the corresponding datatype.

SCOPE OF VARIABLES

Java variables are actually classified into three kinds:

- ✓ *instance*variables
- ✓ *class*variables
- ✓ *local*variables

Instance and class variables are declared inside a class. Instance variables are created when the objects are instantiated and therefore they are associated with the objects. They take different *values* for each object. On the other hand, class variables are' global to a class and belong to the entire set of objects that class creates. Only one memory location is created for each class variable.

Variables declared and used inside methods are called *local variables.* They are called so because they are not available for use outside the method definition. Local variables can also be declared inside program blocks that are defined between an opening brace { and a closing brace }. These variables are visible to the program only from the beginning of its program block to the end of the program block. When the program control leaves a block, all the variables in the block will cease to exist. The area of the program where the variable is accessible is called its *scope.*

SYMBOLIC CONSTANTS

We often use certain unique constants in a program. These constants may appear repeatedly in a number of places in the program. One example of such a constant is 3.142, representing the value of the mathematical constant "pi". Another example is the total number of students whose mark-sheets are analysed by a 'test analysis program'. The number of students, say 50, may be used for calculating the class total, class average, standard deviation, etc. We face two problems in the subsequent use of such programs. Theyare:

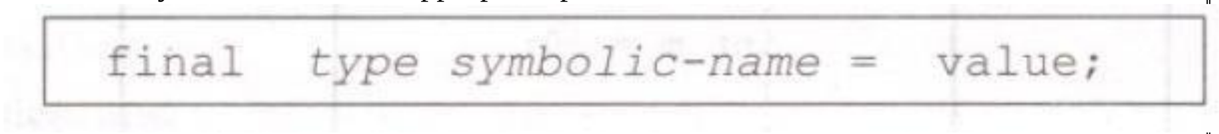
1. Problem in modification of theprogram.
2. Problem in understanding theprogram.

Modifiability

We may like to change the value of "pi" from 3.142 to 3.14159 to improve the accuracy of calculations or the number 50 to 100 to process the test results of another class. In both the cases, we will have to search throughout the program and explicitly change the value of the constant wherever it has been used. If any value is left unchanged, the program may produce disastrous outputs.

Understandability

When a numeric value appears in a program, its use is not always clear, especially when the same value means different things in different places. For example, the number 50 may mean the number of students at one place and the 'pass marks' at another place of the same program. We may forget what a certain number meant, when we read the program some days later. Assignment of a symbolic name to such constants frees us from these problems. For example, we may use the name STRENGTH to denote the number of students and PASS_MARK to denote the pass marks required in a subject. Constant values are assigned to these names at the beginning of the program. Subsequent use of the names STRENGTH and PASS_MARK in the program has the effect of causing their defined values to be automatically substituted at the appropriate points. A constant is declared asfollows:

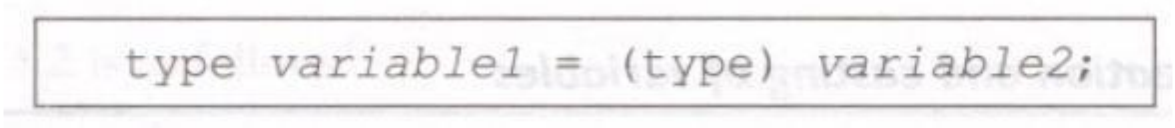


```
final  type symbolic-name =  value;
```

Valid examples of constant declaration are:
Final int STRENGTH = 100;
final int PASS MARK = 50;

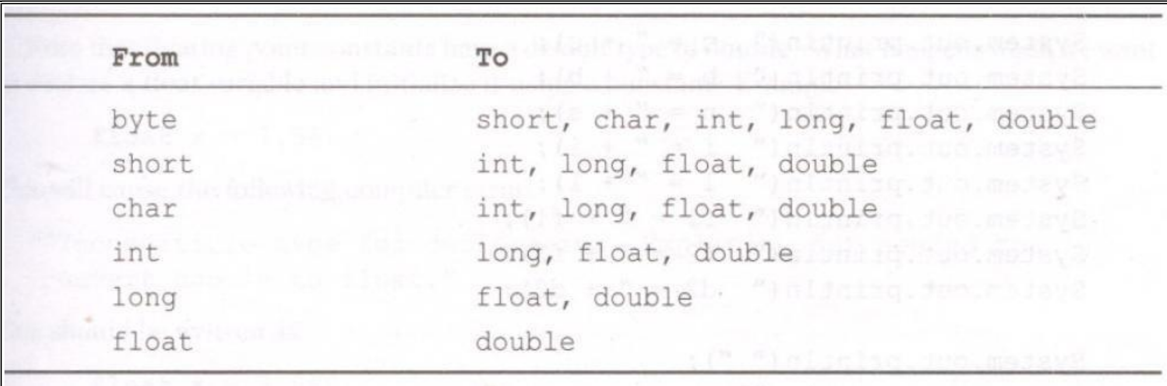
Note that:
1. Symbolic names take the same form as variable names. But, they are written in CAPITALS to visually distinguish them from normal variable names. This is only a convention, not arule.
2. After declaration of symbolic constants, they should not be assigned any other value within the program by using an assignment statement. For example, STRENGTH = 200; isillegal.
3. Symbolic constants are declared fortypes.
4. They can NOT be declared inside a method. They should be used only as class data members in the beginning of theclass.

2.11 TYPECASTING
We often encounter situations where there is a need to store a value of one type into a variable of another type. In such situations, we must cast the value to be stored by proceeding it with the type name in parentheses. The syntaxis:



```
type variable1 = (type) variable2;
```

The process of converting one data type to another is called *casting*. Four integer types can be cast to any other type except boolean. Casting into a smaller type may result in a loss of data. Similarly, the float and double can be cast to any other type except boolean. Again, casting to smaller type can result in a loss of data. Casting a floating point value to an integer will result in a loss of the fractional part.
The process of assigning a smaller type to a larger one is known as *widening* or *promotion* and that of assigning a larger type to a smaller one is known as *narrowing*. Note that narrowing may result in loss ofinformation.
Casts That Result in No Loss of Information



From	To
byte	short, char, int, long, float, double
short	int, long, float, double
char	int, long, float, double
int	long, float, double
long	float, double
float	double

Automatic Conversion

For some types, it is possible to assign a value of one type to a variable of a different type without a cast. Java does the conversion of the assigned value automatically. This is known as *automatic* type conversion. Automatic type conversion is possible only if the destination type has enough precision to store the source value. For example, int is large enough to hold a bytevalue.
Therefore, **byte**b= 75; **int**a = b; are validstatements.

The final result of an expression is converted to the type of the variable on the left of the assignment sign before assigning the value to it. However, the following changes are introduced during the final assignment.
1. float to int causes truncation of the fractionalpart.
2. double to float Causes rounding ofdigits.
3. long to int causes dropping of the excess higher orderbits.

GETTING VALUES OF VARIABLES

A computer program is written to manipulate a given set of data and to display or print the results. Java supports two output methods that can be used to send the results to the screen.

print ( ) method / / print and wait
println( ) method / / print a line and move to next line

The print() method sends information into a buffer. This buffer is not flushed until a newline (or end-of-line) character is sent. As a result, the print() method prints output on one line until a newline character is encountered. For example, the statements

System.out.print("Hello ");
System.out.print("Java!");

It display the words Hello Java! on one line and waits for displaying further information on the same line. We may force the display to be brought to the next line. by printing a newline character as shown below: System.out.print('\n');

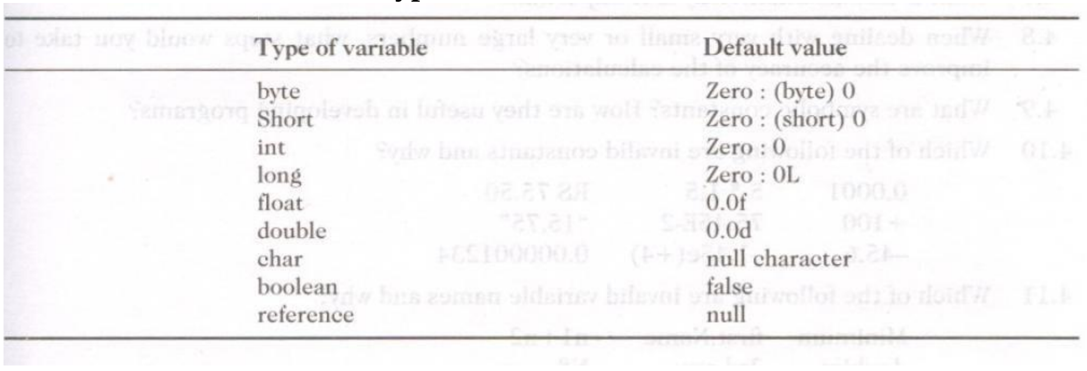
For example, the statements
System.out.print(ftHello");
System.out.print(\n");
System.out.print(Java!");

It will display the output in two lines as follows:
Hello
Java!

STANDARD DEFAULT VALUES

In Java, every variable has a default value. If we don't initialize a variable when it is first created, Java provides default value to that variable type automatically.

Default Values for Various Types



Type of variable	Default value
byte	Zero : (byte) 0
Short	Zero : (short) 0
int	Zero : 0
long	Zero : 0L
float	0.0f
double	0.0d
char	null character
boolean	false
reference	null

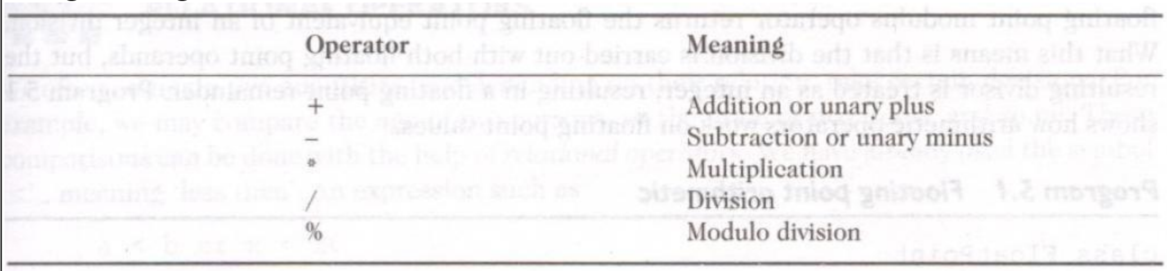
2.12 Operators andExpressions

Introduction

Java supports a rich set of operators. Operator is a symbol that tells the computer to perform certain mathematical or logical manipulations. Operators are used in programs to manipulate data and variables. They usually form a part of mathematical or logical expressions. Java operators can be classified into a number of related categories as below:
1. Arithmeticoperators
2. Relationaloperators
3. Logicaloperators
4. Assignmentoperators
5. Increment and decrementoperators
6. Conditionaloperators.
7. Bitwiseoperators
8. Specialoperators

2.12.1 ARITHMETICOPERATORS

Java provides all the basic arithmetic operators. These can operate on any built-in numeric data type of Java. We cannot use these operators on boolean type. The unary minus operator, in effect, multiplies its single operand by-I. Therefore, a number preceded by a minus sign changes its sign.



Operator	Meaning
+	Addition or unary plus
–	Subtraction or unary minus
*	Multiplication
/	Division
%	Modulo division

Arithmetic operators are used as shown below:

a-b a*b a%b a+b a/b -a*b

Here a and b may be variables or constants and are known as*operands.*

Integer Arithmetic

When both the operands in a single arithmetic expression such as a+ b are integers, the expression is called an *integer expression,* and the operation is called *integer arithmetic.* Integer arithmetic always yields an integer value

Real Arithmetic

An arithmetic operation involving only real operands is called *real arithmetic.* A real operand may assume values either in decimal or exponential notation. Since floating point values are rounded to the number of significant digits permissible, the final value is an approximation of the correct result. .

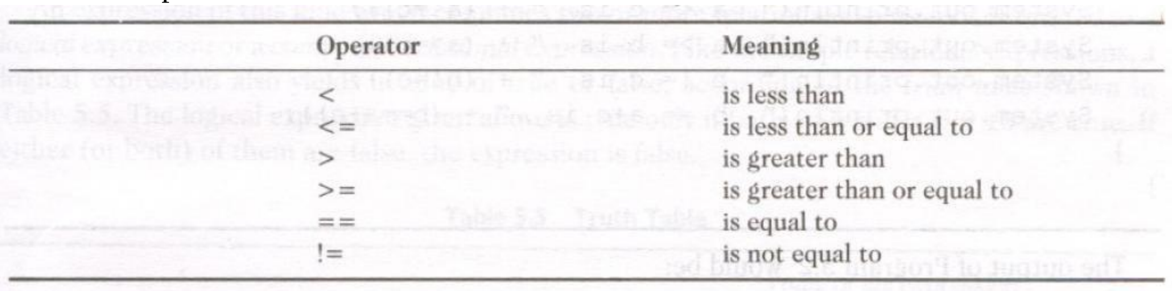
Mixed-mode Arithmetic

When one of the operands is real and the other is integer, the expression is called a *mixed-mode arithmetic e*xpression. If either operand is of the real type, then the other operand is converted to real and the real arithmetic is performed. The result will be a real.

2.12.2 RELATIONALOPERATORS

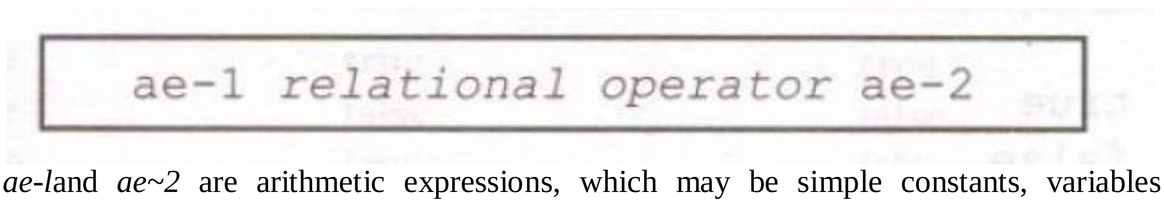
We often compare two quantities, and depending on their relation, take certain decisions. For example, we may compare the age of two persons, or the price of two items, and so on. These comparisons can be done with the help of *relational operators.*

Relational Operators



Operator	Meaning
<	is less than
<=	is less than or equal to
>	is greater than
>=	is greater than or equal to
==	is equal to
!=	is not equal to

A simple relational expression contains only one relational operator and is of the following form:



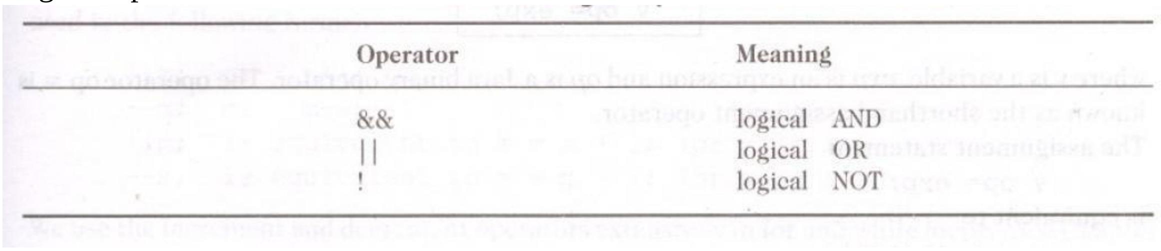
```
ae-1 relational operator ae-2
```

*ae-l*and *ae~2* are arithmetic expressions, which may be simple constants, variables or combination of them.

2.12.3 LOGICALOPERATORS

In addition to the relational operators, Java has three logical operators,

Logical Operators

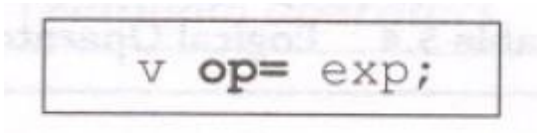


Operator	Meaning
&&	logical AND
||	logical OR
!	logical NOT

The logical operators && and | | are used when we want to form compound conditions by combining two or more relations. An example is: a > b && x 10
An expression of this kind which combines two or more relational expressions is termed as a *logical expression* or a *compound relational expression.* Like the simple relational expressions. Logical expression also yields a value of true or false.

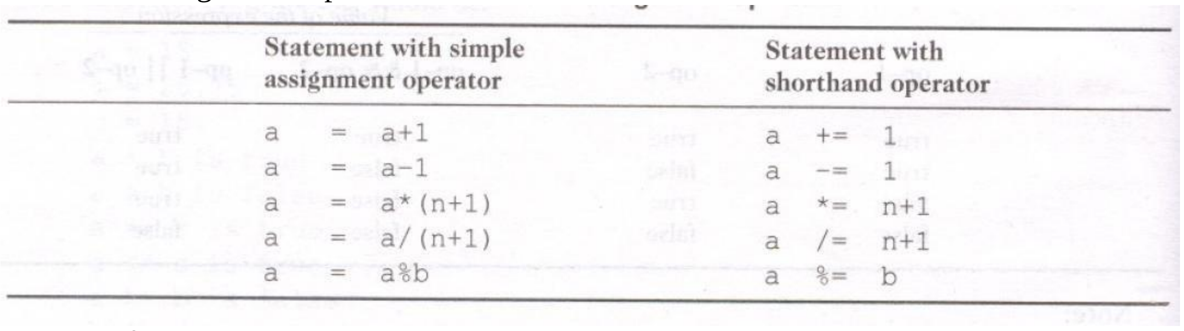
2.12.4 ASSIGNMENTOPERATORS

Assignment operators are used to assign the value of an expression to a variable. We have seen the usual assignment operator = . In addition, Java has a set of 'shorthand' assignment operators which are used in the form



```
v op= exp;
```

Where v is a variable, *exp*is an expression and *op* is a Java binary operator. The operator op = is known as the shorthand assignment operator.
The assignment statement v op= exp; is equivalent to v = v op (exp) with v accessed only once.
Shorthand Assignment Operators



Statement with simple assignment operator	Statement with shorthand operator
a = a+1	a += 1
a = a-1	a -= 1
a = a*(n+1)	a *= n+1
a = a/(n+1)	a /= n+1
a = a%b	a %= b

The use of shorthand assignment operators has three advantages:
1. What appears on the left-hand side need not be repeated and therefore it becomes easier to write.
2. The statement is more concise and easier toread.
3. Use of shorthand operator results in a more efficientcode

2.12.5 INCREMENT AND DECREMENTOPERATORS

Java has two very useful operators not generally found in many other languages. These are the increment and decrement operators: ++ and--
The operator + + adds 1 to the operand while - - subtracts 1. Both are unary operators and are used in the following form.
++m; or m++; is equivalent to m = m + 1;
--In; or m--; is equivalent to m = m - 1;

We use the increment and decrement operators extensively in for and while loops. While +
+m and m+ + mean the same thing when they form statements independently, they behave differently when they are used in expressions on the right-hand side of an assignment statement.
Consider the following:
m = 5;
y = ++m;
In this case, the value of y and m would be 6. Suppose, if we rewrite the above statement as
m = 5;
Y = m++;
Then, the value of y would be 5 and m would be 6. A prefix operator first adds 1 to the operand and then the result is assigned to the variable on left. On the other hand, a postfix operator first assigns the value to the variable on left and then increments the operand.

2.12.6 CONDITIONALOPERATOR

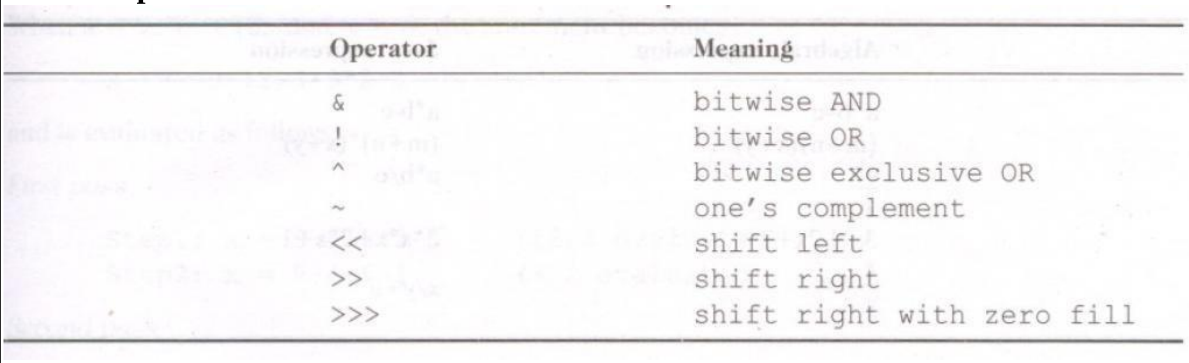
The character pair ?: is a ternary operator available in Java. This operator is used to construct conditional expressions of the form

```
exp1 ? exp2 : exp3
```

Where *expl*, *exp2*, and *exp3* are expressions.
The operator?: works as follows: *expl*is evaluated first. If it is nonzero (true), then the expression *exp2* is evaluated and becomes the value of the conditional expression. If *expl*is false, *eXp3* is evaluated and its value becomes the value of the conditional expression. Note that only one of the expressions (either *exp2* or *exp3)* is evaluated. a = 10;
b = 15; x = (a > b) ?a : b; In this example, x will be assigned the value of b.

2.12.7 BITWISEOPERATORS

Java has a distinction of supporting special operators known as bitwise operators for manipulation of data at values of bit level. These operators are used for testing the bits, or shifting them to the right or left. Bitwise operators may not be applied to float or double.
Bitwise Operators



Operator	Meaning
&	bitwise AND
!	bitwise OR
^	bitwise exclusive OR
~	one's complement
<<	shift left
>>	shift right
>>>	shift right with zero fill

2.12.8 SPECIALOPERATORS

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

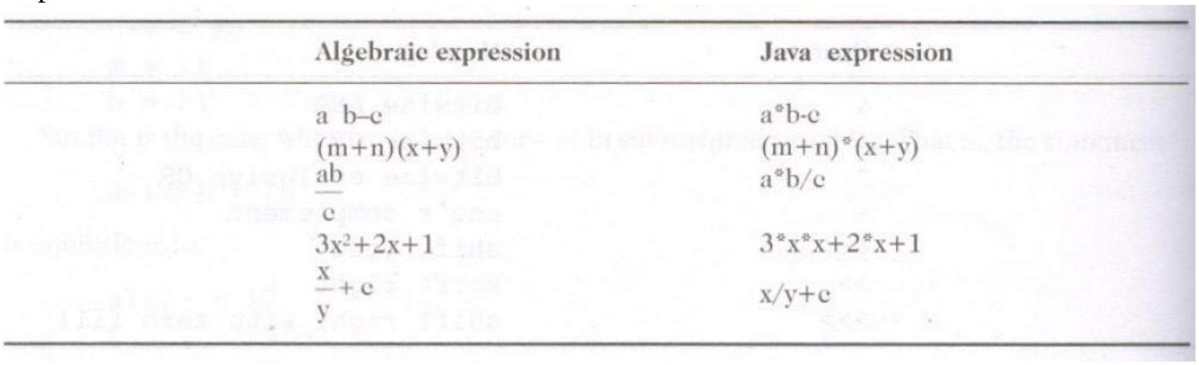
The instance of is an object reference operator and returns *true* if the object on the left-hand side is an instance of the class given on the right-hand side. This operator allows us to determine whether the object belongs to a particular class or not.
Example: person instanceof student is *true* if 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:
person1.age //Reference to the variable age
person1.salary( ) //Reference to the method salary( )

It is also used to access classes and sub-packages from a package.

2.13 ARITHMETICEXPRESSIONS

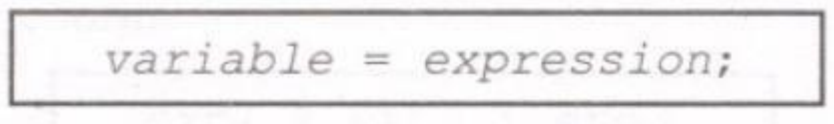
An arithmetic expression is a combination of variables, constants, and operators arranged as per the syntax of the language. Java can handle any complex mathematical expressions.
Expressions



Algebraic expression	Java expression
$a\ b-c$	a*b-c
$(m+n)(x+y)$	(m+n)*(x+y)
$\frac{ab}{c}$	a*b/c
$3x^2+2x+1$	3*x*x+2*x+1
$\frac{x}{y}+c$	x/y+c

2.13.1 EVALUATIONOF EXPRESSIONS

Expressions are evaluated using an assignment statement of the form



```
variable = expression;
```

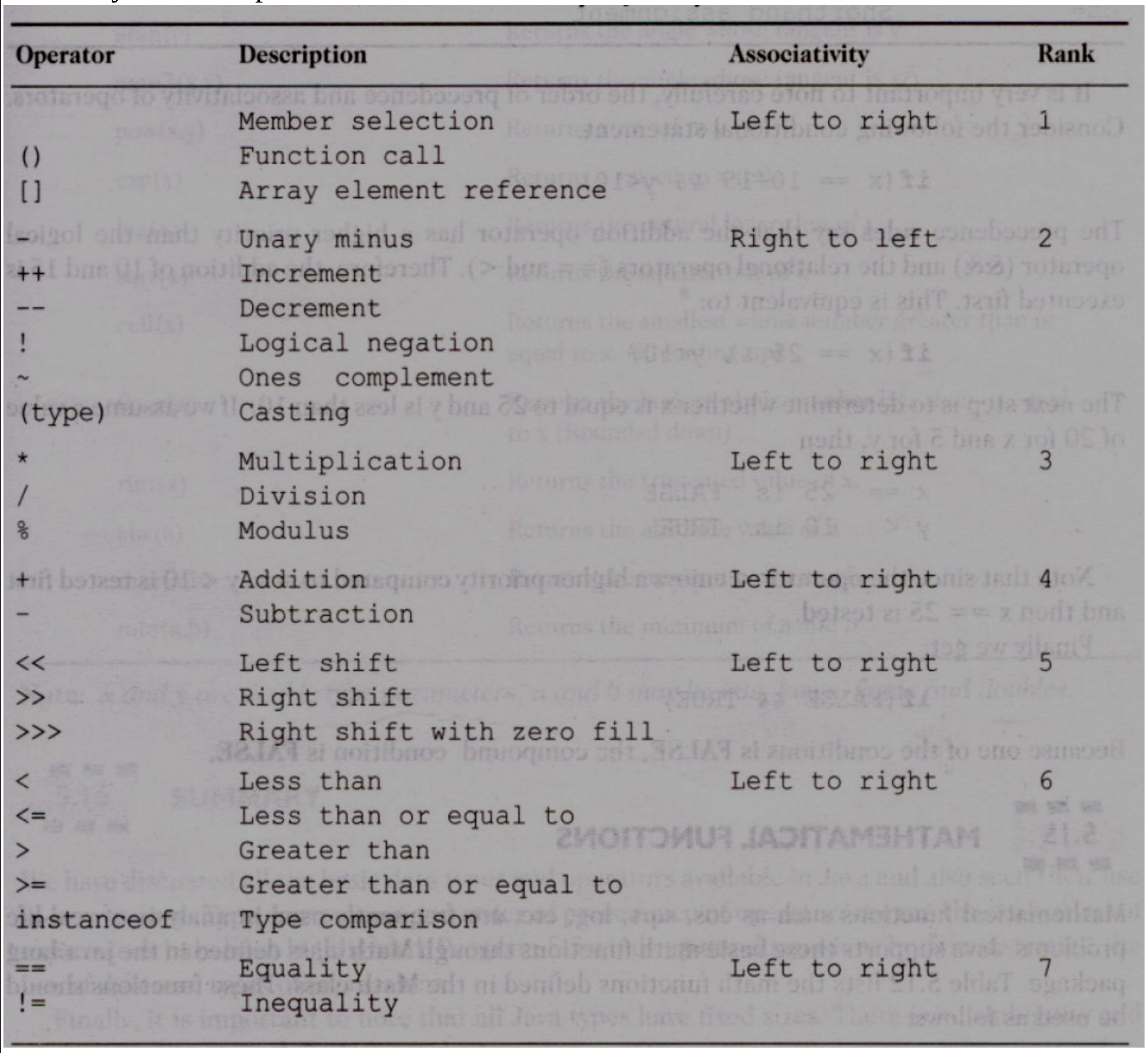
Variable is any valid Java variable name. When the statement is encountered, the *expression* is evaluated first and the result then replaces the previous value of the variable on the left-hand side. All variables used in the expression must be assigned values before evaluation is attempted. Examples of evaluation statements are
x= a*b-c; y = b/c*a; z= a-b/c+d;

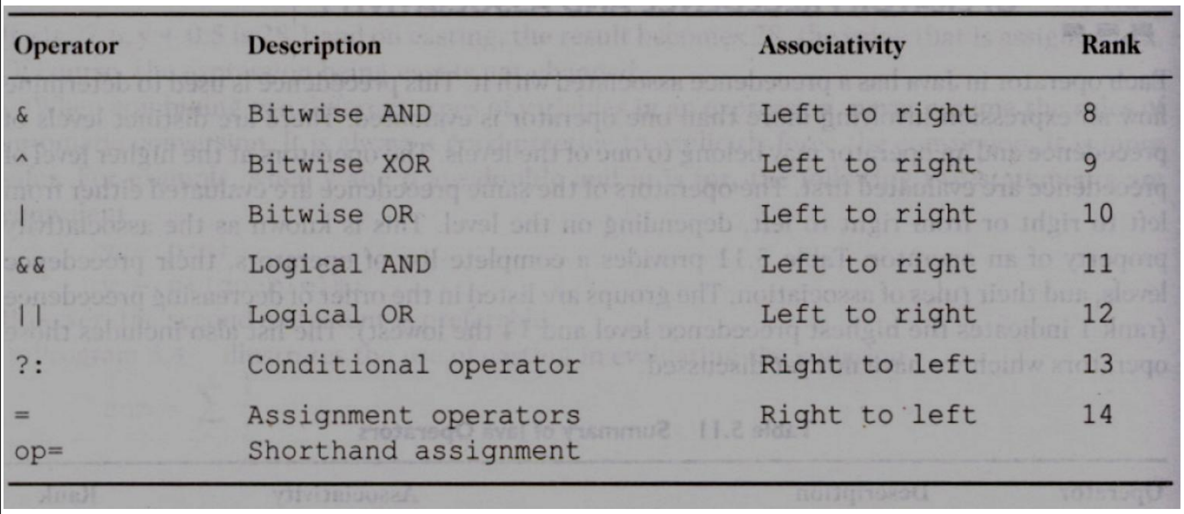
The blank space around an operator is optional and is added only to improve readability. When these statements are used in program, the variables a,b,c and d must be defined before they are used in the expressions.

2.14 OPERATOR PRECEDENCE ANDASSOCIATIVITY

Each operator in Java has a precedence associated with it. This precedence is used to determine how an expression involving more than one operator is evaluated. There are distinct levels of precedence and an operator may belong to one of the levels. The operators at the higher level of precedence are evaluated first. The operators of the same precedence are evaluated either from left to right or from right to left, depending on the level. This is known as the associativity property of an operator.

Summary of Java Operators





Operator	Description	Associativity	Rank
	Member selection	Left to right	1
()	Function call		
[]	Array element reference		
-	Unary minus	Right to left	2
++	Increment		
--	Decrement		
!	Logical negation		
~	Ones complement		
(type)	Casting		
*	Multiplication	Left to right	3
/	Division		
%	Modulus		
+	Addition	Left to right	4
-	Subtraction		
<<	Left shift	Left to right	5
>>	Right shift		
>>>	Right shift with zero fill		
<	Less than	Left to right	6
<=	Less than or equal to		
>	Greater than		
>=	Greater than or equal to		
instanceof	Type comparison		
==	Equality	Left to right	7
!=	Inequality		
&	Bitwise AND	Left to right	8
^	Bitwise XOR	Left to right	9
|	Bitwise OR	Left to right	10
&&	Logical AND	Left to right	11
||	Logical OR	Left to right	12
?:	Conditional operator	Right to left	13
=	Assignment operators	Right to left	14
op=	Shorthand assignment		

Assignment 1

1. **Which will legally declare, construct, and initialize anarray?**

A.int [] myList = {"1", "2", "3"};
B.int [] myList = (5, 8, 2);
C.int myList [] [] = {4,9,7,0};
D.int myList [] = {4, 3, 7};
Answer: Option D

2. **Which is a reserved word in the Java programminglanguage?**

A.MethodB.N
ativeC.Subclas
sesD.Referenc
e
E. Array
Answer: Option B

3. **Which is a valid keyword injava?**

A.Interface
B.StringC.F
loatD.Unsig
ned
Answer: Option A

4. **Which is a valid declarations of aString?**

A. String s1 =null;
B. String s2 ='null';
C. String s3 = (String)'abc';
D. String s4 = (String)'\ufeed';
Answer: Option A

5. **What is the numerical range of achar?**

A.-128 to 127
B.-($2^{15)}$ to (2^{15}) – 1
C.0 to32767
D.0 to65535
Answer: Option D

6. Java isa_______language.
   A. Weakly typed
   B. Stronglytyped
   C. Moderate typed
   D. None ofthese

Answer: Option B

7. **How many primitive data types are there injava?**
   A .6
   B. 7
   C. 8
   D. 9
   Answer: Option C
8. **In java byte, short, int and long all of theseare**
   A. Signed
   B. Unsigned
   C. Both of theabove
   D. None ofthese
   Answer: Option A
9. **Size of int in javais**

A. 16bits
B. 32bits
C. 64bits
D. None of theabove

Answer: Option B

10. **System class is definedin**

A. Java.util package
B. Java.langpackage
C. Java.iopackage
D. Java.awt package
E. None ofthese

Answer: Option B

11. Which of these classes defined in java.io and used for file-handling areabstract?

A. InputStream
B. PrintStream
C. Reader
D. FileInputStream
E. FileWriter

Answer: Option A and C

12. **When comparing java.io.BufferedWriter and java.io.FileWriter which capability exist as a method in only one oftwo?**

A. Closing thestream
B. Flusing thestream
C. Writing to thestream
D. Writing a line separator to thestream
E. None ofthese

Answer: Option D

13. **You want a class to have access to members of another class in the same package. Which is the most restrictive access that accomplishes this objective?**

A. Public
B. Private
C. Protected
D. Default access

Answer:OptionD

14. **Which one creates an instance of anarray?**

A. int[ ] ia = newint[15];
B. float fa = newfloat[20];
C. char[ ] ca = "SomeString";
D. intia[ ] [ ] = { 4, 5, 6 }, { 1,2,3 };

Answer: Option A

15. **Given a method in a protected class, what access modifier do you use to restrict accessto that method to only the other members of the sameclass?**

A. Final
B. Static
C. Private
D. Protected
E. Volatile

Answer: Option C

16. **Which is a valid declaration within aninterface?**

A. public static short stop =23;
B. protected short stop =23;
C. transient short stop =23;

D. final void madness(short stop);

Answer: Option A

17. **Which class does not override the *equals()* and *hashCode()* methods, inheriting them directly from classObject?**

A.
java.lang.StringB.java.la
ng.DoubleC.java.lang.St
ringBufferD.java.lang.C
haracter**Answer:
Option C**

18. **Which collection class allows you to grow or shrink its size and provides indexed access to its elements, but whose methods are notsynchronized?**

A.java.util.HashSetB.java.
util.LinkedHashSetC.java.
util.ListD.java.util.ArrayL
ist**Answer: Option D**

19. **You need to store elements in a collection that guarantees that no duplicates are stored and all elements can be accessed in natural order. Which interface provides thatcapability?**

A.java.util.MapB.java.util.
SetC.java.util.ListD.java.uti
l.Collection**Answer:
Option B**

20. **Which interface does *java.util.Hashtable*implement?**

A.Java.util.MapB.Jav
a.util.ListC.Java.util.
HashTableD.Java.util.
Collection**Answer:
Option A**

21. **Which collection class allows you to associate its elements with key values, and allows you to retrieve objects in FIFO (first-in, first-out)sequence?**

A.java.util.ArrayListB.java.
util.LinkedHashMapC.java.
util.HashMapD.java.util.Tr
eeMap**Answer: Option B**

22. **Which is valid declaration of afloat?**

A. float f =1F;
B. float f =1.0;
C. float f ="1";
D. float f =1.0d;

Answer: Option A

23. **What is the numerical range ofchar?**

A.0 to 32767
B.0 to 65535
C.-256 to 255
D.-32768 to 32767

Answer: Option B

24. **Which of the following would compile withouterror?**

A. int a =Math.abs(-5);
B. int b =Math.abs(5.0);
C. int c =Math.abs(5.5F);
D. int d =Math.abs(5L);

Answer: Option A

25. **You want subclasses in any package to have access to members of a superclass. Which is the most restrictive access that accomplishes thisobjective?**

A.Publi
cB.Priv
ateC.Pr
otected
D.Trans
ient

Answer: Option C

Part-B
Long Answer Questions

1. Explain the features of java programming language
2. Explain the structure of a java program with an example.
3. Explain about Java Virtual Machine.
4. Explain the different data types used in Java.
5. Define a variable. How do you declare the variable in Java? Explain with an example.
6. Explain different types of operators in Java.
7. Explain Command Line Arguments in Java with an example.
8. Explain the different types of tokens in Java
9. Write a note on symbolic constants in Java.
10. How do you work with arithmetic expressions in Java? Explain
11. How do you implement a Java program? Explain
12. Compare and contrast Java and C++.