Unit-I	- **Java History, Features, comparision with C & C++** - **Java and Internet, www, Java Environment** - **Java Program Structure and Simple Program** - **Implementing a Java Program , JVM** - **Java Tokens and Comments** - **Constants, Variables, Data types, Declaration of Variables, Giving values to Variables** - **Scope of Variables, Type Casting** - **Getting Values of Variables, Default Variables** - **Operators: Arithmetic, Relational, Logical, Assignment, Increment/Decrement, Conditional &**

Java History

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 like TVs, VCRs, toasters and such other electronic machines. Below lists some important milestones in the development of Java.

Year	Development
1990	Sun Microsystems decided to develop special software that could be used manipulate consumer electronic devices. A team of Sun Microsystems programmers headed by James Gosling was formed to undertake this task.
1991	After exploring the possibility of using the most popular object-oriented language C++, the team announced a new language named “Oak”.
1992	The team, known as Green Project team by Sun, demonstrated the application of their new language to control a list of home appliances using a hand-held device with a tiny touch0sensitive screen.
1993	The World Wide Web(WWW) appeared on the Internet and transformed

	the text-based Internet into a graphical-rich environment. The Green Project team came up with the idea of developing Web applets (tiny programs) using the new language that could run on all types of computers connected to Internet.
1994	The team developed a Web browser called “Hot Java” to locate and run applet programs on Internet. Hot Java demonstrated the power of the new language, thus making it instantly popular among the Internet users.
1995	Oak was renamed “Java”, due to some legal snags. Java is just a name and is not an acronym. Many popular companies including Netscape and Microsoft announced their support to Java.
1996	Java established itself not only as a leader for Internet programming but also as a general-purpose, object oriented programming language. Sun releases Java Development Kit 1.0.
1997	Sun releases Java Development Kit 1.1 (JDK 1.1)
1998	Sun releases the Java 2 with version 1.2 of the Software Development Kit (SDK 1.2)
1999	Sun releases the Java 2 Platform, Standard Edition (J2SE) and Enterprise Edition (J2EE)
2000	J2SE with SDK 1.3 was released.
2002	J2SE with SDK 1.4 was released.
2004	J2SE with SDK 5.0(instead of JDK 1.5) was released. This is known as J2SE 5.0

Features of Java

1. **Compiled and Interpreted**

   Usually computer language is either compiled or interpreted while java combines both approach thus it make two-stage system. Java compiler translates source code into byte code instruction. Byte code is not machine instruction i.e., therefore the second stage java interpreter generate machine code that can be executed by machine .i.e running language.

2. **Platform-Independent and Portable**

   The most significant contribution of java is its portability. Java program easily moved one computer system to another anywhere and any time. Change & upgrade in OS processors and system resources will not force any change in java program. So java popular an Internet which interconnect different types of systems worldwide. We can download java applets from remote computer on to our local computer via Internet and execute it locally. Java ensures portability by two ways: (1) Java compiler generates byte code instruction that can be implemented on any machine. (2) The sizes of primitive data types are machine dependents.

3. **Object Oriented**

   Java is a true object-oriented language. Almost everything in java is in 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. The object model in java is simple and easy to control.

4. **Robust and secured**

   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 garbage-collected language relieving the programmers virtually all memory management problems. Java also incorporates concepts of exception handling, which ensures series of error and eliminates any risk of crashing the system.

   Security becomes an important issue for language that is used for programming on Internet. Java system not only verifies all memory access but also ensures no viruses are communicated with an applet. The absence of pointer in java ensure that program can not gain access to memory location without proper authorization.

5. **Distributed**

   Java is designed as distributed language for creating applications on networks. It has the ability to share both data and program. Java applications can open and access remote object 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.

6. **Simple, Small and Familiar**

   Java is a small and simple language. Many feature of c & C++ that are either redundant or sources of unreliable code are not part of java. E.g. java does not use pointer, preprocessor header files, goto statement and many others. It also eliminates operator overloading and multiple inheritance.

7. **Multithreaded and Interactive**

   Multithread means handling multiple tasks simultaneously. Java supports multithreaded programs. We need not wait for application to finish one task before beginning another. For example, we can listen to an audio clip, while scrolling a page and at the same time download an applet from distant computer. This feature greatly improves the interactive performance of graphical applications.

8. **High performance**

   Java performance is impressive due to use intermediate byte code. Java speeds comparable to the native of c & c++ .of dynamically linking new class. Java architecture is also designed to reduce overheads during runtime. Further, the incorporation multithreading enhances the over all execution speed of java programs.

9. **Dynamic and Extensible**

   Java is dynamic language, capable of dynamically linking 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 program support functions written in other languages such as c & c++. These functions are known as native methods. This facility enables to programmers to use the efficient function available in these languages. Native methods are linked dynamically at runtime.

Java Environment

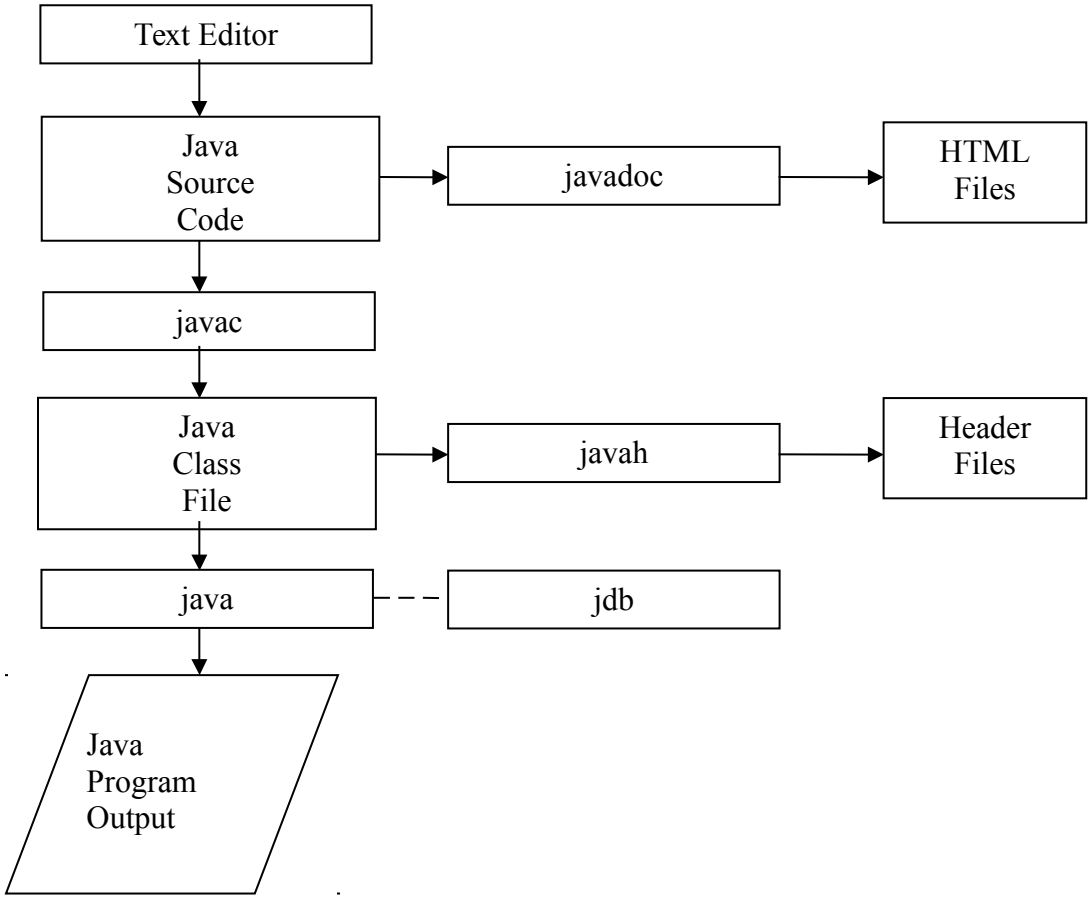
Java Environment includes a large number of development tools and hundreds of classes and methods. The development tools are part of the system known as Java Development kit (JDK) and the classes and methods are part of the Java Standard Library(JSL), also known as the Application Programming Interface(API).

1. **Java Development Kit**

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

   a. **appletviewer**(for viewing Java Applets) : Enables us to run Java applets(without actually using a Java-compatible browser).
   b. **javac**(Java Compiler) : The Java comiler, which translates Java sourcecode to bytecode files that the interpreter can understand.
   c. **java**(Java Interpreter) : Java interpreter, which runs applets and applications by reading and interpreting bytecode files.
   d. **javadoc** : Creates HTML format documentation from Java source code files.
   e. **javah** : Produces header files for use with native methods.
   f. **javap** : Java disassemble, which helps us to find errors in our programs.

The way these tools are applied to build and run application programs is illustrated in below figure. 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 disassemble javap. We learn more about these tools as we work through the book.



Process of Building and running Java application programs

2. Application Programming Interface

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 order 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.

Java program structure

Documentation Section	Suggested
Package statement	Optional
Import statement	Optional
Interface statement	Optional
Class definition	Optional
Main Method Class { main method definition. }	Essential

Documentation Section: This 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. This would greatly help in maintaining the program. Java also use a third style of comment /** …….*/ known as documentation comment. This form of comment is used for generating documentation automatically.

Package Statements: The first statement allowed in a java file a package statement. this statement declares a package name and informs the compiler that classes defined here belong to this package. E.g: package student; The package statement is optional.

Import Statements: The next statement after package statement (but before any class definition) may be number of import statements. This is similar to #include statement in c. Example: import student.test;

This statement instructs the interpreter to load the test class contained in the package student. Using import statements, we can access to classes that are part of other named packages.

Interface Statements: An interface is like a class but includes group of method declarations. This is also an optional section and used only when we wish to implement the multiple inheritance features in a program. Interface is a new concept in java.

Class Definition: A java program may contain multiple class definition. Classes are primary and essential elements of java program. These classes are used to map the objects of real-world problems. The number of classes used depends on the complexity of the problem.

Main Method Class: Since every java stand-alone program requires main method as it's starting point, this class is the essential part of 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 end of main, the program terminates and the control passes back to the operating system.

Simple Java Program

The best way to learn a new language is to write a few simple example programs and execute them. We begin with a very simple program that prints a line of text as output.

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

Let us discuss the program line by line and understand the unique features that constitute a Java program.

Class Declaration

The first line

```
class SampleOne
```

declares a class, which is an object oriented construct. Java is a true object oriented language and therefore, everything must be placed inside a class. class is a keyword and declare 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 am opening brace "{" and ends with a matching closing brace "}", appearing in the last line in the example. This is similar to C++ class construct.

The Main Line

The third line

public static void main(String args[])

defines a method name mai. Conceptually, this is similar to the main() function in C and C++. 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. This is similar to the C++ public modifier.

static: Next appears the keyword 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 declare as a 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)

Output Line

The only executable statement in the program is

System.out.println("Java is better than C++");

This is similar to the printf statement of C or cout<< construct of 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.

Implementing a Java Program

Implementation of a Java application program involved a series of steps they include :

- Creating the program
- Compiling the program
- Running the Program

Remember that, before we begin creating the program, the java development kit(jdk) must be properly installed in your system.

Creating the program

We can create a program using any text editor assume that we have entered the following program:

```
Class Test
{
        public static void main(String args[])
        {
        System.out.println("Hellow!");
        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 file name content the class name properly. This file is called the source file. Note that all java source file will have the extension java. Note also that if a program contain multiple classes, the file name must be the class name of the class containing the main method.

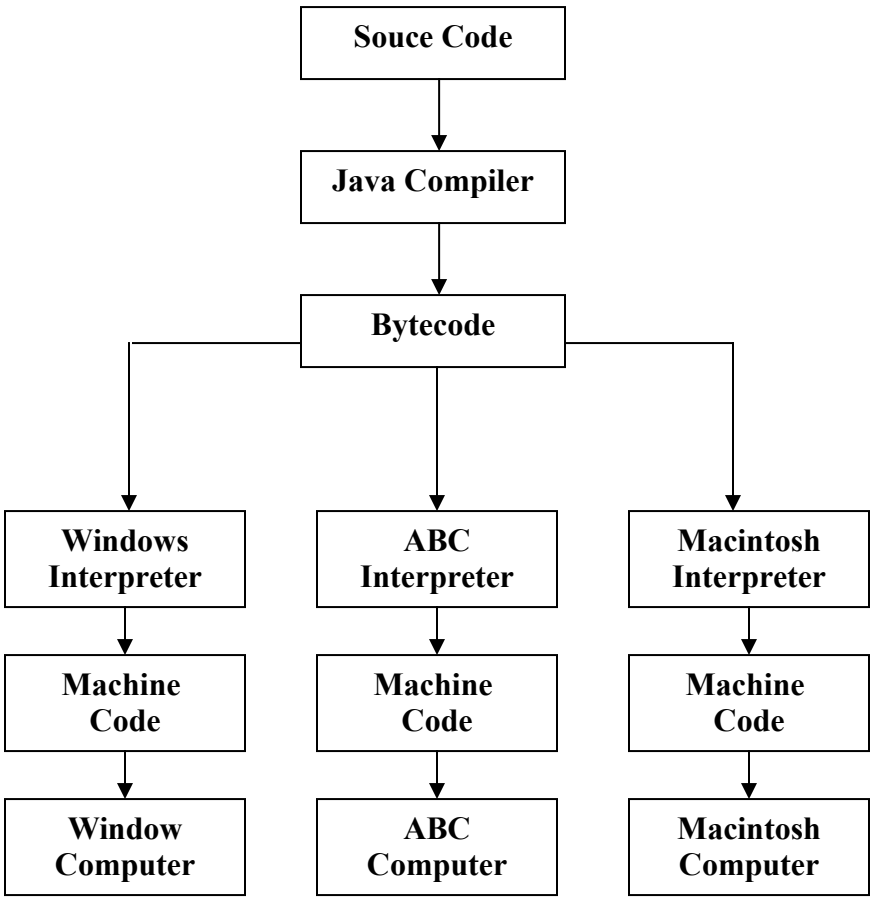
Compiling the Program

To compile the program, we must run the java compiler javac, with the name of the source file on the command line as shown below:

javac Test.java

If everything is ok, the javac compiler creates a file called Test.class containing the bytecode of the program. Note that the compiler will automatically names the bytecode as

<class name> .class



Implementation of Java Programs

Running 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, all program displays the following:

Hellow!

Welcome to the world of java

Let us learn java

Note that we simply type "Test" at the command line and not "Test.class" or "Test.java".

Machine Neutral

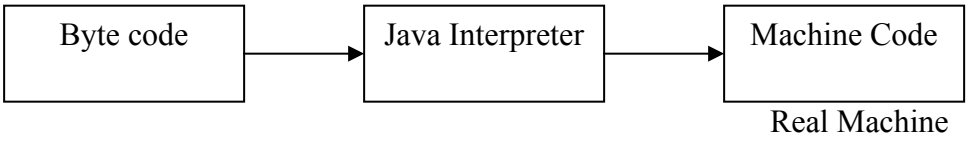
The compiler converts the source code files into bytecode files. These codes are machine-independent and therefore can be run on any machine. This is a program complied on an IBM machine will run on a Macintosh machine.

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 of machine.

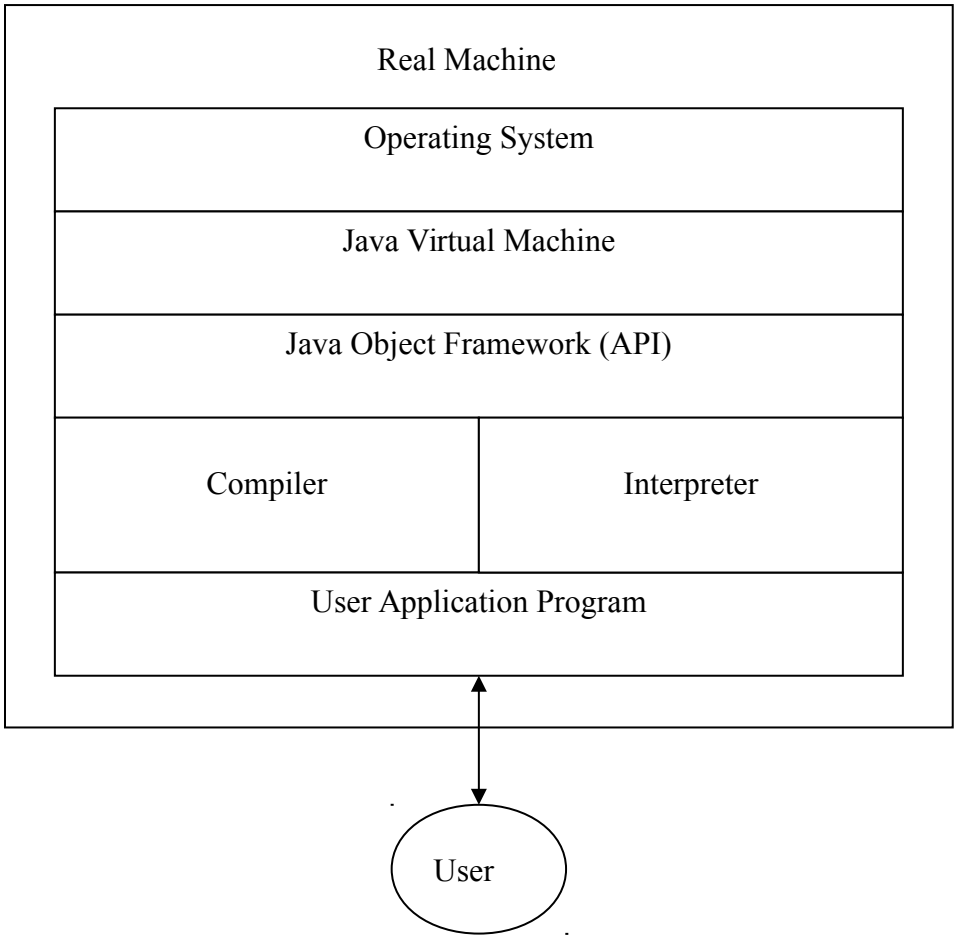
Java Virtual Machine(JVM)

All language compilers translates source code into machine code for a specific computer. Java compiler also does the same thing. Then, how does java achieved architecture neutrality? The answer is that the java compiler produces an intermedia code known as bytecode for a machine that does not exist. The machine is called the Java Virtual Machine and it exist only inside the computer memory. It is a simulated computer within the computer and does all major functions of a real computer. Below figure illustrate the process of compiling a java program into bytecode which is also referred to as virtual machine code.

The virtual machine code is not machine specific. The machine specific code(known as a machine code) is generated by the java interpreter by acting as an intermediary between the virtual machine and the real machine as shown in below figure. Remember that the interpreter is different for different machine.



Bewlow figure illustrate how java works on a typical computer. The Java object framework (Java API) acts as the intermediary between the user program and the virtual machine which in turn acts as the intermediary between the operating system and the java object framework.



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

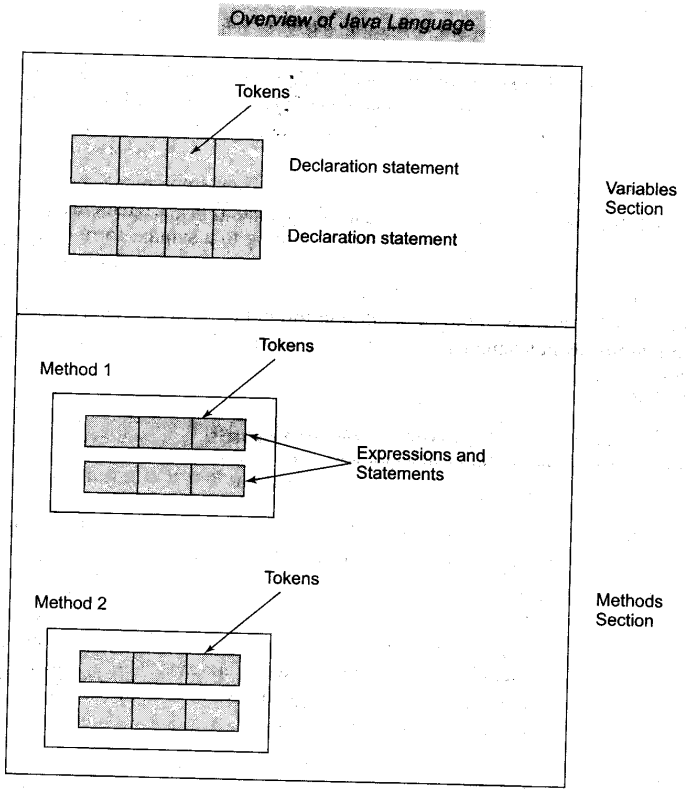


Fig. 3.3 *Elements of Java class*

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 l6-bit character coding system and currently supports more than 34,000 defined Characters derived from 24 languages from America, Europe, Middle East, Africa and Asia (including India).

Keywords

Keywords are an essential part of a language definition. They implement specific features of the language. Java language has reserved 50 words as keywords.

The keywords, combined with operators and separators according to a syntax, form definition of the java language. Understanding the meanings of all these words is important for Java programmers.

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 uppercase. However, it is bad practice and should be avoided.

Table 3.1 Java Keywords

abstract	assert	boolean	break
byte	case	catch	char
class	const	continue	default
do	double	else	enum
extends	final	finally	float
for	goto	if	implements
import	instanceof	int	interface
long	native	new	package
private	protected	public	return
short	static	strictfp	super
switch	synchronized	this	throw
throws	transient	try	void
volatile	while		

Identifier

Identifiers are programmer-designed tokens. They are used for naming classes, methods, variables, objects, labels, packages and interfaces in a program. Java identifier follow the following rules:

1. They can have alphabets, digits, and the underscore and dollar sign characters.
2. They must not begin with digit.
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 naming conventions.

- Names of all public methods and instance variables start with a leading lowercase letter.

  Examples: average, sum
- when more than one words are used in a name, the second and subsequent words are marked with a leading uppercase letters, Examples:

  dayTemperature, firstDayOfMonth, totalMarks
- All private and local variables use only lowercase letters combined with underscores.

  Examples:

  length, batch_strength
- All classes and interfaces start with a leading uppercase letter (and each subsequent word with a leading uppercase lener). Examples:
  Student
  HelloJava
  Vehicle
  MotorCyc1e
- Variables that represent constant values use all uppercase letters and underscores between words. Examples:
  TOTAL
  F_MAX
  PRINCIPAL_AMOUNT

They are like symbolic constants in C.

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 rule 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
- Floatingjoint literals
- Character literals

- String literals
- Boolean literals

Each of them has a type associated with it. The type describes how the values behave and how they are stored. We will discuss these in detail when we deal with data types and constants later.

Operators

An operator is a symbol that takes one or more arguments and operates on them to produce a result. Operators are of many types.

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. Below table lists separator and their functions.

Parentheses () – Used to enclose parameters in method definition and invocation, also used for definition precedence in expressions, containing expressions for flow control, and surrounding cast types.

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

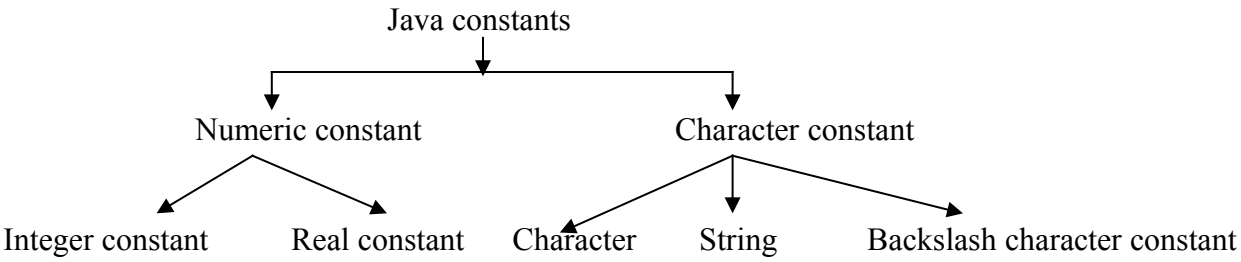
Brackets [] - Used to declare array types and for dereferencing array values.

Semicolon ; - Used to separate statements

Comma , - Used to separate consecutive identifiers in variable declaration, also used to chain statements together inside a 'for' statement.

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

Constants



Constant constant

Constants in java refer to fixed value that do not change during the execution of a program. Java supports several types of constants as illustrated in fig.

Integer Constants

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

Decimal integer consist of a set of digits, 0 through 9,preceded by an optional minus sign, Valid example of decimal integer constants are:

123 -321 0 654321

Embedded space, commas , and non digit character are not permitted between digits: For example,

15 750 20.000 $1000

Are illegal Numbers.

An octal integer constant consists of any combination of digit from set 0 through 7, with a leading 0. Some example of octal integer are:

037 0 0435 0551

A sequence of digit preceded by 0x or 0X is considered as hexadecimal integer(her integer). They may also include alphabets A through F or a through f . A letter A through F represents the number 10 through 15. Following are the example of valid hex integers.

0X2 0X9F 0xbcd 0x

We rarely use octal and hexadecimal number in programming.

Real Constants

Integer number 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 number are called real(or floating point)constants. Further example of real constants are

0.0083 -0.75 435.36

These number are shown in decimal notation, having a whole number followed by a decimal point and the fractional part, which is an integer. It is possible that the number may not have digits before the decimal point or digits after the decimal point. That is,

215. .95 -.71

Are all valid real numbers.

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 by $10^{2.}$ .The general form is :

mantissa e exponent

The mantissa is either a real number expressed in decimal notation or 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. Since the exponent causes the decimal point to "float", this notation is said to represent a real number in floating point form. Example of legal floating point constants are:

0.65e4 12e-2 1.5e+5 3.18E3 -1.2E-1

Embedded while (blank) space is not allowed, in any numeric constant.

Exponential notation is useful for representing number that are either very large or very small in magnitude. For example , 7500000000 may be written as 7.5E9 or 75E8. Similarly , -0.000000368 is Equivalent to -3.68E-7.

A floating point constant may thus comprise four parts:

- A whole number
- A decimal point
- A fractional part
- An exponent

Single Character Constants

A single character constants(or simply character constant) contains a single character enclosed within a pair of quote marks. Example of character constants are:

'5' 'X' ':' ' '

Note that the character constants '5' is not same as the number 5. The last constants is a blank Space.

String Constants

A string constants is a sequence of characters enclosed between double quotes. The character may be alphabets, digits, special character and blank spaces . For Example are:

"Hello Java" "1997" "WELL DONE" "?...!" "5+3" "X"

Backslash Character constants

Java support some special backslash character constants that are used in output methods. For Example , The symbol '\n' stands for newline character. A list of such backslash character constants is given in table. Note that each one of them represents one character, although they consist of two character. These character combinations are known as escape sequences.

Constant	Meaning
'\b'	back space
'\f'	Form feed
'\n'	New line
'\r'	Carriage return
'\t'	Horizontal tab
'\' '	Single quote
'\" '	Double quote
'\\'	Backslash

Variables

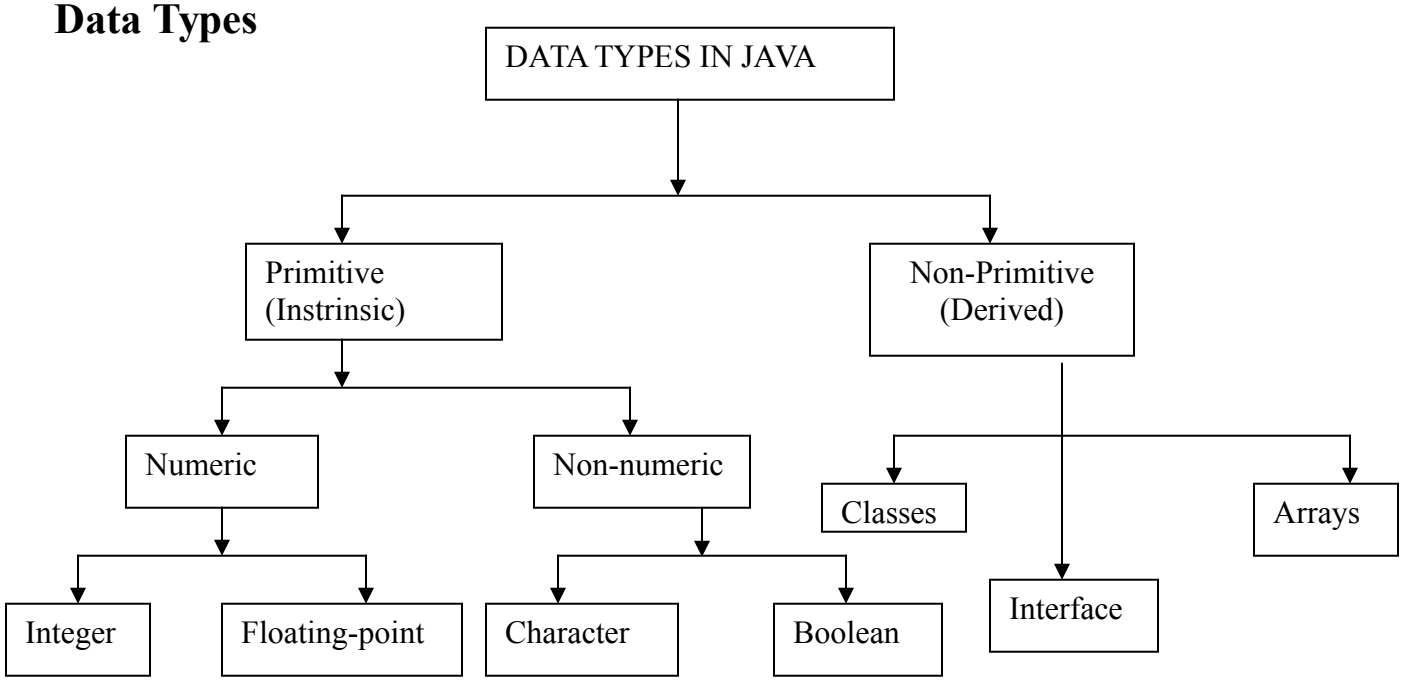
A variable is an identifier that denotes a storage location used to store a data value. Unlike constant that remain unchanged during the execution of the program. A variable name can be chosen by the programmer in a meaningful way so as to reflect what it represents in the program. Some examples of variable names are:

- average
- height
- total_height
- classStrength

As mention earlier,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 or TOTAL.
3. It should not be a keyword.
4. White space is not allowed.
5. Variable names can be of any length.

Data Types

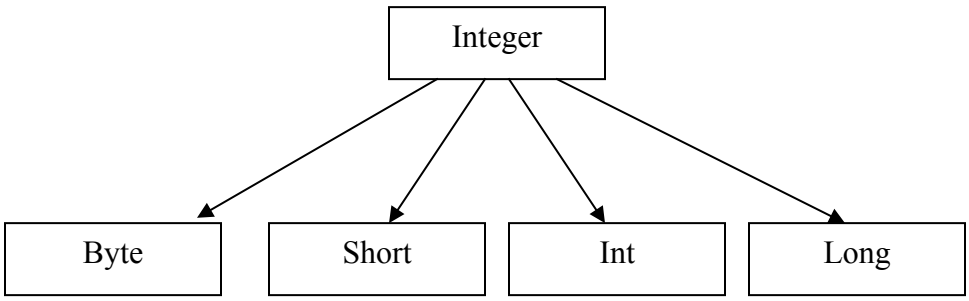


Data Types in Java

Every variable in java as a data type. Data types specify the size and type of values that can be stored. Java language is rich in it's data types. The variety of data types available allow the programmer to select the type appropriate to the need of the application. Data types in java under

various categories are shown in figure. Primitive types (also called intrinsic or built-in types) are discuss in detailed in this chapter. Derived types (also known as reference types) are discussed later as and when they are encountered.

Integer Types



Integer types can be hold whole number such as 123,-96 and 5639. The size of the values that can be stored depends on the integer data types we choose. Java supports four types of integers as shown in figure. They are byte , short ,integer and long java does not support the concept of unsigned types and therefore all java values are signed meaning they can be positive or negative. Below table shows the memory size and range of all four integer data types.

SIZE AND RANGE OF INTEGER

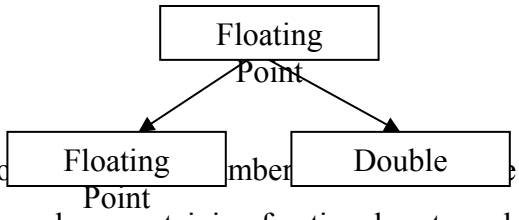
TYPE	SIZE	MINIMUM VALUE	MAXIMUM VALUE
byte	one byte	-128	127
short	two bytes	-32,768	32767
int	four bytes	-2,147,483,648	2,147,483,647
long	eight bytes	-9,223,372,036,854,775,808	9,223,372,036,854,775,807

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. For example instead of storing a number like 50 in an integer type variable, we must use a byte variable to handle this number. this will improve the speed of execution of the program.

We can make integer long by appending the letter L or 1 at the end of the number. example:

123L or 1231

Floating Point Types



Integer types can hold numbers use another type known as floating point type to hold numbers containing fractional parts such as 27.59 and -1.375(known as point constants).There are two kinds of floating point storage in java as shown in fig.

The float type values are single-precision numbers while the double types represent double-precision numbers. Table give the size and range of the two types.

Floating point numbers are treated as double-precision quantities. to force them to be in single-precision mode, we must append f or F to the number, Example:

1. 23f

7.56923e5F

Size and Range of Floating Point Types

TYPE	SIZE	MINIMUM VALUE	MAXIMUM VALUE
float	4 bytes	3.4e-038	1.7e+0.38
double	8 bytes	3.4e-038	1.7e+308

Double –precision types are used when we need greater precision in storage of floating point number. All mathematical functions, such as sin ,cos and sqrt return double type values.

Floating point data types supports special value known as Not a-Number (NaN). NaN is used to represent the result of operation such as dividing zero, where an actual number is not producted. Most operations that have NaN as an operand will produce NaN as aresult.

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. Boolean type is denoted by the keyword boolean and uses only one bit of storage.

All comparison operation return boolean type values. Boolean values are often used in selection and iteration statements . The words true or false cannot be used as identifiers.

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 name is.
2. It specifies what type of data the variable will hold.
3. The place of declaration (in the program) decides the scope of the variable.

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 variable is:

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;
double pi;
byte    b;
char    c1, c2, c3;
```

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 assignment statement
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:

```
InitilValue     =     0;
finalValue      =     100;
yes             =     'x';
```

we can also string assignment expression as shown below:

```
x = y = z = 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;

Examples:

```
int     finalValue    =    100;
char    yes           =    'x';
double  total         =    75.36;
```

The process of giving initial values to variable is known as the initialization. The ones that are not initialized are automatically set to zero. The following are valid Java statements:

```
float   x, y, z;          // declares three float variables
int     m = 5, n = 10;    // declares and initializes two int variables
int     m, n = 10;        // declares m and n and initializes n
```

Read Statement

We may also give values to variables interactively through the keyword using the readLine() method as illustrated in below example.

```
import java.io.DataInputStream;
class Reading{
    public static void main(String args[]){
    DataInputStream in = new DataInputStream(System.in);
    int intNumber = 0;
    float floatNumber = 0.0f;
    try
    {   System.out.println("Enter an integer:");
        intNumber = Integer.parseInt(in.readLine());
```

```
            System.out.println("Enter a float number");
            floatNumber = Float.valueOf(in.readLine().floatValue());
        }catch(Exception e) { }
        System.out.println("intNumber = " + intNumber);
        System.out.prinlnt("floatNumber=" +floatNumber);
        }
    }
```

The interactive input and output of above Program are shown below:

```
Enter an integer:
123
Enter a float number:
123.45
intNumber = 123
flaotnumber = 123.45
```

The readLine() method (which is invoked using an object of the class DataInputStream) reads the input from the keyword as a string which is then converted to the corresponding data type using the data type wrapper classes.

Note that we have used the keywords try and catch to handle any errors that might occur during the reading process.

Scope of Variables

Java variables are actually classified into three kinds:

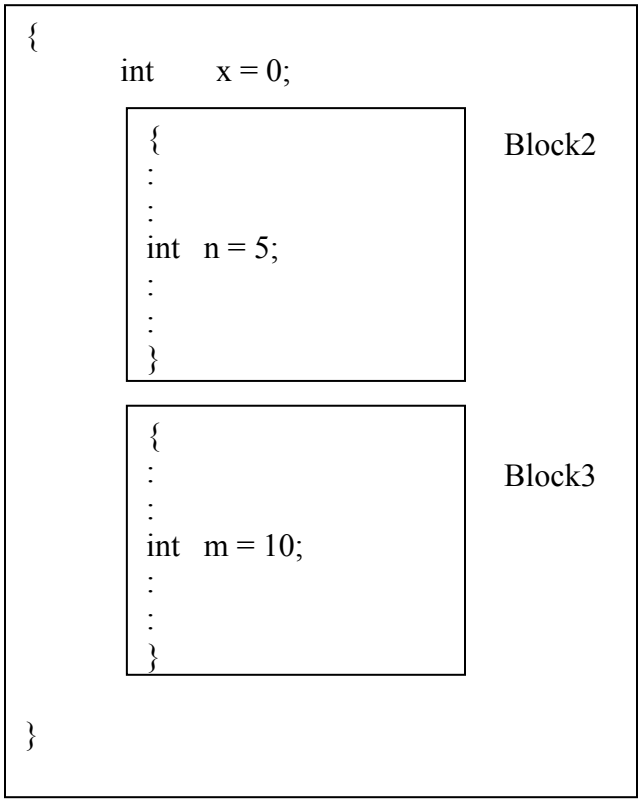
- instance variables
- class variables, and
- local variables.

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. Instance and class variables will be considered in detail in later chapters.

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 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 program where the variable is accessible is called its scope.

We can have program blocks within order program blocks (called nesting) as shown in below figure.



Each block can contain its own set of local variable declarations. We cannot, however, declare a variable to have the same name as one in an outer block. In above figure variable x declared in Block1 is available in all the three blocks. However, the variable n declared in Block2 is available only in Block2, because it goes out of the scope at the end of Block2. Similarly, m is accessible only in Block3.

Note that we cannot declare the variable x again in Block2 or Block3.

Type Casting

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 syntax is:

type variable1 = (type) variable2;

The process of converting one data type to another is called casting. Examples:

```
int    m=5;
byte   n= (byte)m;
long count=(long)m;
```

Casting is often necessary when a method returns a type different than the one we require.

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. Below table lists those casts, which are guaranteed to 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 precession to store the source value. For example, int is large enough to hold a byte value. Therefore,

```
byte   b = 75;
int    a = b;
```

are valid statements.

The process of assigning a smaller type to a longer 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 of information.

Program illustrate the creation of variables of basic types and also shows the effect of type conversions.

```
class  TypeWrap
{
        public static void main(String args[])
        {
          System.out.println("Variables created");
          char  c = 'x';
          byte b = 50;
          short s = 1996;
          int  i = 123456789;
          long l = 1234567654321L;
          float f1 = 3.142F;
          float f2 = 1.2e-5F;
          double d2 = 0.000000987;
          System.out.println(" c =" + c);
          System.out.println(" b ="+ b);
          System.out.println(" s ="+ s);
          System.out.println(" i ="+ i);
          System.out.println(" l ="+ l);
          System.out.println(" f1 ="+ f1);
          System.out.println(" f2 ="+ f2);
          System.out.println(" d2 ="+ d2);
          System.out.println("");
          System.out.println("Types converted");
          short s1= (short) b;
          short s2= (short) b;      // Produces incorrect result
          float n1= (float) l;
          int  m1= (int) f1;        //Fractional part is lost
          System.out.println(" (short) b = " + s1);
          System.out.println(" (short) i = " + s2);
          System.out.println(" (float) l = " + n1);
```

```
        System.out.println(" (int) f1 = " + m1);
    }
}
```

Output of Program

```
Variables created
c = x
b = 50
s = 1996
i = 123456789
l = 1234567654321
f1 = 3.142
f2 = 1.2e-005
d2 = 9.87e-007
Types converted
(short)b = 50
(short)i = -13035
(float)l = 1.23457e+012
(int) f1 = 3
```

Note that floating point constants have a default of double. What happens when we want to declare a float variable and initialize it using a constant? Example:

```
float    x = 7.56;
```

This will cause the following compiler error;

"Incompatible type for declaration. Explicit cast needed to convert double to float"

This should be written as

```
float    x = 7.56F;
```

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!");
```

Will 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("Hello ");
System.out.print("\n");
System.out.print("Java!");
```

will display the output in two lines as follows:

```
Hello
Java!
```

The println() method, by contrast, takes the information provided and displays it on a line followed by a line feed(carriage-return). This means that the statements

```
System.out.println("Hello");
System.out.println("Java");
```

will produce the following output:

```
Hello
Java!
```

The statement

```
System.out.print( );
```

will print a blank line. Program illustrated the behavior of print() and println() methods.

```
class   Displaying
{
        public static void main(String args[])
        {
                System.out.println("Screen Display");
                for(int i=1; i<=4; i++)
```

```
                {
                        for(int j=1; j<=i; j++)
                        {
                                System.out.print(“”);
                                System.out.print(i );
                        }
                System.out.print(“\n”);
                }
        System.out.println(“Screen Display Done”);
        }
}
```

Program displays the following on the screen:

Screen Display

1

2 2

3 3 3

4 4 4 4

Screen Display Done

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 as shown in below Table.

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

Operators

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. Arithmetic operators.
2. Relational operators.
3. Logical operators.
4. Assignment operators.
5. Increment and decrement operators.
6. Conditional operators.
7. Bitwise operators.
8. Special operators.

Arithmetic Operators

Arithmetic operators are used to construct mathematical expressions as in algebra. Java provides the following basic arithmetic operators:

Operator	Meaning
+	Addition or unary plus
-	Subtraction or unary minus
*	Multiplication
/	Division
%	Modulo division (Remainder)

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.

Eg:- If a and b are integers, then for a=14 and b=4 we have the following results:

a – b = 10
a + b = 18
a * b = 56
a / b = 3 (decimal part truncated)
a % b = 2 (remainder of integer division)

For modulo division, the sign of the result is always the sign of the first operand (the dividend). i.e.

-14 % 3 = -2
-14 % -3 = -2
14 % -3 = 2

Real Arithmetic

An arithmetic operation involving only read operands is called real arithmetic. A real operand may assume values either in decimal or exponential notation.

Mixed-mode Arithmetic

When one of the operands is real and the other is integer, the expression is called a mixed-mode arithmetic expression. 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.

Eg:-	15 / 10.0	produces the result	1.5
Whereas	15 / 10	produces the result	1

Relational Operators

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

<	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-1 and ae-2 are arithmetic expressions, which may be simple constants, variables or combination of them.
Eg:

4.5 <= 10 TRUE
-35 > = 0 FALSE
10 < 7+5 TRUE

When arithmetic expressions are used on either side of a relational operator, the arithmetic expressions will be evaluated first and then the results compared. That is, arithmetic operators have a higher priority over relational operators.

```
class Relational0perators
{
public static void mairl(String args[])
{
float a = 15.0F, b = 20.tSF, c = 15,0F;
System.out.pnintln(" a = " + a);
System.out.println(" b = " + b);
System.out.println(" c = " + c);
System.out.println(" a < b is ': + (a<b)); .,
System.out.println(" a > b is " + (pb)):
System.out.pnintln(" a =: c is " + (a==c));
System.out.println(" a <= c js " + (a<=c));
System.out..println(" a >= b is " + (a>=b));
System.out.println(" b != c is " + (bl:c));
System.out.println(" b == a+c is " + (b=:a+c)):
)
)
```

The output of Program 5.2 would be:
a=15
b : 20.75
c=15
a<b is true
a>b is false
a == c is true
a <= c is true
a >= b is false
a != c is true
b == a+c is false

Relational expressions are used in decision statements such as, if and while to decide the

course of action of a running program.

Logical Operators
Java has three logical operators, which are given in below:

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, a logical expression also yields a value of true or false.

Assignment Operators

Assignment operators are used to assign the value of an expression to a variable. Java has 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 the shorthand assignment operator.
The assignment statement

v op= exp;

is equivalent to

v = v op(exp);

with v accessed only once. Consider an example

x += y+1:

This is same as the statement

x = x+(y+l);

The shorthand operator += means 'add y+1 to x' or 'increment x by y+1'. For y=2, the above statement becomes ***x+=3;***
and when this statement is executed, 3 is added to x. If the old value of x is, say 5, then the new value of x is 8. Some of the commonly used shorthand assignment operators are illustrated below:

Shorthand Assignment Operators

Statement with simple assignment operator	Statement with shorthand operator
a = a+l	a += 1
a = a-1	a -= 1
d = 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 to read.
3. Use of shorthand operator results in a more efficient code.

Increment and Decrement Operators

Java has two very useful operators not generally found in many other languages. These are i increment and decrement operators;

++ and --

The operator ++ adds 1 to the operand while – subtracts1. Both are unary operators and are used in the following form:

++m; or m++;
--m or m--;
++m, is equivalent to m = m + l, (on m += 1;)
-- m; is equivalent to m = m - 1; (or m -= 1;)

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 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.

Increment Operator illustrated

```
class IncrementOperator
{
 public stalic void main(String angs[])
 {
        int m=10,n=20;
        System.out.println("m = " + m);
        System.out.println("n = " + n);
        System.out.println("++m = " + ++m);
        System.out.println("n++ = " + n++);
        System.out.println("m "+ m);
        System.out.println("n = " + n);
 }
}
```

Output of above Program is as follows:

```
m = 10
n = 20
++m = 11
n++ = 20
m = 11
n = 21
```

Similar is the case, when we use ++ (or --) in subscripted variables. That is, the statement

a[i++] = 10

is equivalent to

a[i] = 10

i = i+1

Conditional Operator

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 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 its value becomes the value of the conditional expression. Only one of the expression (either exp2 or exp3) is evaluated.

For example, consider the following statements:

```
a = l0:
5 =15;
x = (a > b) ? a : b;
```

In this example, x will be assigned the value of b. This can be achieved using the if..else-statement as follows:

```
if(a > b)
        x=a;
else
        x=b;
```

Bitwise Operators

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

Special Operators

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

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:

```
person1.age          // Reference to the variable age
person1.salary()     // Reference to the method salary()
```

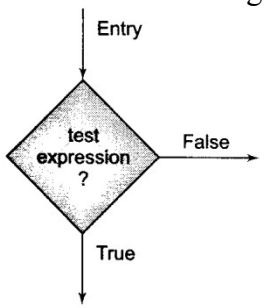
It is also used to access classes and sub-packages from a package.

Decision Making with If Statement

The if statement is a powerful decision making statement and is used to control the flow of execution of statement .it is basically a two –way decision statement and is used in conjunction with an expression . it takes the following form:

If (test expression)

It allows the computer to evaluate the expression first and then ,depending on whether the value of the expression (relation or condition) is 'true' or 'false' , it transfer the control to a particular statement. This point of program has two paths to follow, one for the true condition and the other for the false condition as shown in the below figure



Some example of decision making, using if statement are: -

1. If (bank balance is zero)
   Borrow money
2. If(room is dark)
   Put on lights
3. if(code is 1)
   Person is male
4. if(age is more then 55)
   Person is retired

The if statement may be implemented in different forms depending on the complexity of conditions to be tested.

1. Simple if statement
2. If...else statement
3. Nested if...else statement
4. Else if ladder

Simple if statement

The general form of a simple if statement is

```
If (test expression)
{               Statement - block;
}
  Statement - x:
```

The 'statement –block' may be a single statement or a group of statements. If the test expression is true , the statement-block will be executed ; otherwise the statement –block will be skipped and the execution will jump to the statement-x.

It should be remembered that when the condition is true both the statement –block and the statement –x are executed in sequence. This is illustrated in the below give figure

Consider the following segment of a program that is written for processing of marks obtained in an entrance examination.

```
……
……
If (category  ==  SPORTS)
 {
marks  =  marks   + bonus _marks;
 }
System.out.println(marks);
……
……
```

The program tests the type of category of the student .if the student belongs to the SPORTS category , then addition bonus _marks are added to his marks before they are printed . For others ,bonus _marks are not added.

Consider a case having two test condition ,one for weight and another for height. This is done using the compound relation

```
If (weight  < 50  &&   height > 170) count =count +1;
```

This would have been equivalently done using two if statement as follow:

```
If (weight <50)
        If (height>170)
                Count = count+1;
```

If the value of weight is less then 50, then the following statement is executed , which in turn is another if statement. This is statement tested height and if the height is greater than 170, then the count is incremented by 1 . program illustrates the implementation of the above statement .

```
classIfTest
        {
         public   static void main(String arg[])
            {
int i,count,count1,count2;
float[] weight = {45.0f , 55 .0f, 47 .0f, 51.0f, 54.0f};
float [] weight ={ 176.5f,174.2f,168.0f,170.7f,169.0f};
count  =0;
                    count1=0;
                    count2=0;
                For (i = 0 ;i<=4 ; i++)
                        {
If(weight[i] <50.0 && height[i]>170.0)
                        {
                                count1=count1+1;
                         }
count=count +1;// total  persons
```

```
                    }
                count2=count –count1;
        System.out.println("Number of person with...");
        System.out.println("weight<50 and height>170 =" +count1);
        System.out.println("Other  = "+count2);
                }
        }
```

The output of program will be

```
Number of person with...
Weight < 50 && height >170 =1
Other = 4
```

THE IF…ELSE STATEMENT

The if…else statement is an execution of the simple if statement .the general form is

```
if (test expression)
    {
      True- block statement(s)
    }
else
    {
      False - block statement(s)
    }
Statement-X
```

If the test expression is true ,then the true –block statement(s) immediately following the if statement, are executed ;otherwise ,the false –block statement (s) executed. In either true-block or false-block will be executed ,not both . this is illustrated in fig 6.3. In both the cases is transferred subsequently to the statement-x.

Let us consider an example of counting the number of boys and girls in a class .we use code 1 for a boy and 2 for a girl. The program statements to do this may be written as follows:

```
        ……….
        ……..
if (code  ==  1)
boy = boy+1;
if (code == 2)
girl =girl+1;
      ……..
      ……..
```

The first test determine whether or not the student is a boy, if yes, the number of boys is increment by 1 and program continues to the second test. The second test again determine whether the student is girl. This is unnecessary. Once a student is identified as a boy, there is no need to test again for a girl. A Student can be either a boy or girl, not both . the above program segment can be modified using the else clause as follows:

```
        ……….
        ……..
if (code  ==  1)
boy = boy+1;
```

```
else
girl =girl+1;
      X XX ;
          ..................
```

Here, if the code is equal to 1, the statement boy=boy+1; is executed and the control is transferred to the statement X XX , after skipping the else part . if the code is not equal to 1, the statement boy = boy+1; is skipped and the statement in the else part girl=girl+1; is executed before the control reaches the statement xxx.

Program 6.2 counts the even and odd numbers in a list of number using the if…else statement .number[ ] is array variable containing all the number and number. length gives the number of elements in the array.

Program 6.2 Experimenting with if…..else:-

```
Class IfElseTest
   {
        public   static void main(String arg[])
         {
        int number []={50,65,56,71,81};
        int even=0; odd=0;
        for(int i=0; i<number.length;i++)
              {
        if(number[i] % 2) = = 0)    //    decide even or odd
              {
                even+=1;   //    counting even numbers
               }
        else
               {
                odd +=1;   //  counting odd numbers
                }
          }
        System.out.println("Even Numbers:"+ even + "Odd Numbers: "  + odd);
         }
}
```

Output of program

```
Even Numbers : 2   Odd Numbers : 3
```

NESTING OF IF ….ELSE STATEMENTS

When a series of decisions are involved ,we may have to use more than on if ….else statement in nested form as follows:

The login of execution is illustrated in fig 6.4 .if the condition -1 is false ,the statement-3 will be executed: otherwise it continues to perform the second test. if the condition

-2 true then statement 1 will be evaluated : otherwise the statements-2 will be evaluated and then the control is transferred to the statement-x.

A commercial bank has introduced an incentive policy of giving bonus to all its deposit holder. The policy is as follows: a bonus of 2 per cent of the balance held on 31 st December is given to every one,
Irrespective of their balances, and 5 per cent is given t female account holders if their balance is more than rs 5000. This login can be coded as follows:

```
                ……….
If(sex is female)
{
If(balance > 5000)
  Bonus=0.0.5*balance;
Else
Bonus=0.02*balance;
}
Else
{
bonus=0.02*balance;
}
Balance=balance+bonus;
……..
……
```

When nesting care should be exercised to match every if with an else .consider the following alternative to the above program (which looks right at the first sight):

```
            If (sex is female)
          If(balance > 5000)
                  Bonus=0.05*balance;
              Else
                  Bonus=0.02*balance;
                  Balance=balance*bonus;
```

There is an ambiguity as to over which if the else bonus belongs to . Java an else is linked to the closed non-terminated .therefore , the else is associated with the inner if and there is no else option for the outer if.This means that the computer is trying to executed the statement

balance=balance+bonus;

without really calculating the bonus for the male account holder.
Consider the another alternative ,which also looks correct:

```
            If (sex is female)
             {
             If(balance > 5000)
                      Bonus=0.05*balance;
                  }
              Else

                  Bonus=0.02*balance;
                  Balance=balance*bonus;
```

IN this case, else is associated with the outer if and therefore bonus is calculated for the male account holders. However , bonus for the female account holder, whose balance is equal to or less then 5000 is not calculated because of the missing else option for inner if .

THE ELSE IF LADDER

THERE IS another way is putting ifs together when multipath decision are involved. A multipath decision is a chain of its which the statement associated with each else is an it. It takes the following general form.

This construct is known as the else if ladder. The condition are evaluated from the top(of the ladder), downwards. As soon as the true condition is found ,the statement associated with it is executed and the control is tarnsferred to the statement-x(skipping the rest of this ladder). When all the n condition becomes false, then the final else containing the default-statement will be the executed. figure 6.5 shows the logic of execution of else if ladder statements.
Lets us consider an example of grading the students in an academic institution. the grading is done according to the following rules:

Average marks	Grade
80 to 100	Honours
60 to 79	First Division
50 to 59	Second Division
40 to 49	Third Division
0 to 39	Fail

This grading can be done using the else if ladder as follows:

```
If(marks > 79)
        Grade=”Honours”;
Else if(marks  > 59)
Grade =”First Division”;
Else if(marks > 49)
 Grade =”Second  Division”;
Else if(marks > 39)
 Grade =”Third  Division”;
Else
Grade =”Fail”;
System.out.println(“Grade:”+grade);
```

Consider another example given below:

```
.....
.....
    If (code == 1)
Color= ”Red”;
else     If (code == 2 )
```

```
Color= ”Green”;
else     If (code == 3 )
Color= ”WHITE”;
Else
Color=”Yellow”;
……..
……..
```

Code number other then 1,2,or 3 are consider the represented YELLOW color. The same result can be obtained by using the nesting if……else statements.

```
If(code  != 1)
If(code  != 2)
If(code  != 3)
color=”YELLOW”;
else
color=”WHITE”;
else
color=”GREEN”;
else
color=”RED”;
```

in such situation , the choice of the method is left to the programmer. However , in order to choose an if structure that is both efficient, it is import to that the programmer is fully aware of the various forms of an if statement and the rules governing their nesting.

Program 6.4 demonstrates the use of if ….else ladder in analyzing a mark list.

THE SWITCH STATEMENT

We have seen that when one of the many alternative is to be selected, we can design a program using the if statement to control the selection. However, the complexity of such a program increase dramatically when the number of alternative increase. the program becomes difficult to read and follow. At times, it may confuse as switch. The switch statement tests the value of a given variable against a list of case values and when a match is found ,a block of statement associated with that case is executed. the general form of the switch statement is as below:

```
Switch(expression)
    {
                    Case  value-1
                          Block-1
                          Break;
                    Case  value-2
                          Block-2
                          Break;
……..
…….
…….
```

```
Default:
                    Default –block
                Break;
}
Statement-x;
```

The expression is an integer expression or character .value-1,value-2….are constants or constant expression and are known as case lables, each of these values should be unique within a switch statement ,block-1,block-2..are statement lists and many contain zero or more statement . there is no need to put braces these blocks but it is important to note that case labels end with a colon(;).

When the switch is executed ,the value of the expression is successively compared against the values value-1,value-2….if the a case is found whose value the matches with the value of the expression ,then the block of statement that follows the case are executed.

The break statement at the end of the each of block signals the end of a particular case and cause an exit from the switch statement, transferring the control to the statements-x following the switch.

The default is an optional case. When present ,it will be executed if the value of the expression does not match with any of the case values. If not present , no action takes place when all matches fail and control goes to the statement-x.

The selection process of switch statement is illustracted in the flowchart shown in fig 6.6:=
The switch statement can be used to grade the students as discussed in the lat section. This is illustrated below:

```
……………….
……………….
 Index=marks/10;
Switch(index)
{
case  10:
case 9:
case 8:
   grade=”Honours”;
break;
case 7:
case 6:
   grade=”First division”;
break;
case 5:
   grade=”second division”;
break;
case 4:
   grade=”Third division”;
```

```
break;
default:
      grade=”fail”;
break;
}
System.out.println(grade);
……..
……..
```

Note that we have used a conversion statement

Index =marks/10;

Where, index is defined as an integer. The variable index takes the following integer values:

MARKS	INDEX
100	10
90-99	9
80-89	8
70-79	7
60-69	6
50-59	5
40-49	4
30-39	3
20-29	2
10-19	1
0-9	0

The statement of the program illustrated two important features, first it used empty cases. The first three cases will executed the same statement:

```
     grade=”Honours”;
break;
```

Same is the case with case 7and case 6. Second, default condition is used for all other cases where marks is less then 40.

Program 6.5 illustrates the use of switch for designing a menudriven interactive program:

Program 6.5:=

6.8 THE ?:OPERATOR:=

The java language has an unusual operator, useful for the making two-way decision . this operator is a combination of ? and : , and the task three operands. This operator is popularly known as the conditional operator. The general form of use of the conditional operator is as follows:

Conditional expression ? expression1: expression2

The Conditional expression is evaluated first. if the result is true expression 1 is evaluated and is returned as the value of Conditional expression. Otherwise ,expression 2 is evaluated and its values is returned. For example, the segment

```
   If(x<0
Flag=0;
```

Else
Flag=1
;
Can be written as flag=(x<0)?0:1;
Consider the evaluated of the following function:
Y=1.5x + 3 for x<=2
Y=2x + 5 for x>2

This can be evaluated using the condition operator as follows:

Y=(x>2) ? (2*x+5) : (1.5*x+3);

The conditional operator may be nested for evaluated more complex assignment decision .for example ,consider the weekly salary of a sales girl who is selling some domestic products. If x is the number of products sold in week, her salary is given by

{ 4x + 100 for x<40 }
Salary= { 300 for x=40 }
{ 4.5x +150 for x>40 }

This is complex equation can be written as

Salary =(x!=40)? ((x<40) ? (4*x+100):(4.5*x+150)):300;

The same can be evaluated using the if…..else statements as follow:

```
If(x <= 40)
        If(x<40)
                        Salary=4*x+100;
        Else
                                Salary=300;
Else
  Salary =4.5*x+150;
```

When the conditional operator is used , the code becomes concise and perhaps, more efficient .however the readability is poor . it is better to use if statement when more then a single nesting of conditional operator is required.

Unit 2
7.1
Decision making and looping
A looping process, in general, would include the following four steps :

1. Setting and initialization of a counter
2. Execution of the statement in the loop
3. Test for a specified condition for execution of the loop
4. Incrementing the counter

The test may be either to determine whether the loop has been repeted the specified number of times or to determine whether a particular condition has been met with.
The java language provides for three construct for performing loop operations. They are

1. **WHILE** construct
2. **Do** construct

3. **FOR** construct
   we shall discuss the features and applications of each of these constructors in this chapter.

7.2
The while statement
The simplest all the looping structure in java is the **while** statement. The basic format of the **while** statement is

```
Intialization :
While (test condition)
{
        Body of the loop
}
```

The **while** is an *entry controlled* loop statement. The ***test condition*** is evaluated and if the condition is true, then the body of the loop is executed. After execution of the body, the test condition is once again evaluated and if it is true, the body is executed once again . this process of repeated execution of the body continues until the test condition finally becomes false and the control is transferred outof the loop. On exit, the programs continues with the statement immediately after the body of the loop.

The body of the loop may have one or more statements. The braces are needed only if the body contains two or more statements. However , it is good pratice to use braces even if the body has only one statement.

Consider the following code segment

```
.....
.....
Sum=0;
N=1;
While(n < = 10)
{
        Sum  = sum + n * n;
        n = n + 1 ;
}
System.out.println("Sum= " + sum);
.....
.....
```

The body of the loop is executed 10 times for n = 1,2 ,.....,10 each time adding the square of the value of n, which is incremented inside the loop. The test condition may also be written as n < 11; the result would be the same. Program 7.1 illustrates the use of the **while** for reading a string of the characters from the keyboard. The loop terminates when c= '\n', the newline character.

Program 7.1 *using while loop*

```
Class WhileTest
{
```

```
Public static void main (String args [ ])
{
        StringBuffer string = new stringBuffer();
        Char c;
        System.out.println("ENTE R THE STRING ");
        try
        {
                While  ( ( c= (char ) System.in.read() ) != '\n')
                        {
                                String.append(c) ;  //  Append character
                        }
         }
        Catch (Exception e )
{
        System.out.println("Error in input");
}
        System.out.println("You have entered . . . . . ");
        System.out.println(string);
        }
}
```

Given below is the output of Program 7.1 :

```
Enter a string
Java is a true object-oriented language
        You have entered . .
Java is a true object-oriented language
```

7.3 The do STATEMENT

The **while** loop construct that we have discussed int he previous section makes a test condition *before* the loop is executed . therefore, the body of the loop may not be executed at all if the condition is not satisfied at the very first attempt. On some occasions it might be necessary to execute the body of the loop before the test is performed. Such situations can be handled with the of the **do** statement .this takes the form.

```
Initialization;
do
{
        Body of the loop
}
While ( test condition );
```

On reaching the **do** statement , the programs proceeds to evaluate the body of the loop first. At the end of the loop , the *test condition* in the **while** statement is evaluated. If the condition is true, the programs continues to evaluate the body of the *loop once again* .this process continues

as long as the *condition* true . when the condition becomes false, the loop will be terminated and the control goes to the statement that appears immediately after the **while** statement .

Since the *test condition* is evaluated at the bottom of the loop, the **do....while** construct provides and *exit-controlled* loop and therefore the body of the loop is always executed at least once.

Consider an example :

```
        .....
        .....
 i =  1;
sum = 0 ;
do
{
        Sum = sum + i ;
        i =  i + 2 ;
}
While ( sum< 40 || i < 10 ) ;
.....
.....
```

The loop will be executed as long as one of the two relations is true . programs 7.2 illustrates the use of **do. ...while** loops for printing a multiplication table.

Programs 7.2 *printing multiplication table using do..... while loop.*

```
Class Dowhiletest
{
        Public static void main ( Stringargs [ ] )
        {
                Int row, column, y;
                System.out.println (" MULTIPLICATION TABLE \n");
                row = 1;
                do
                {
                        Column = 1 ;
                do
                {
                        y = row * column ;
                System.out.print( " " + y );
                Column =  column +  1 ;
        }

        While ( column< = 3) ;
        System .out.println( "\n");
        row = row + 1 ;
}
        While  ( row < = 3 ) ;
```

```
    }
}
```

Program 7.2 uses two **do-while** loops in nested form and produces the following output :
Multiplication Table
1 2 3
2 4 6
3 6 9
Table 7.1 lists the differences between **while** and **do-while** loops.

while	*do-while*
it is a looping construct that will execute only if the test condition is true.	It is a looping constuct that will execute At leat once even if the test condition is false .
it is an entry- controlled loop .	it is an exit-controlled loop.
It is generally used for implementing Common looping situation .	it is typically used for implementing menu based programs where the menu is required to be printed at least once

7.4 THE for STATEMENT
The **for** loop is another *entry- controlled* loop that provides a more concise loop control structure. The general form of the **for** loop is

```
For ( initialization ; test condition ; increment)
{
        Body of the loop
}
```

The execution of the **for** statements is as follows :

1. *Initialization* of the *control variables* is done first, using assigment statements such as i = 1 and count = 0. The variables **i** and **count** are known as loop-control variables .

2. The value of the control variable is tested using the *test condition* . the test condition is arelational expression, such as i < 10 that determines when the loop will exit . if the condition is TRUE , the body of the loop is executed ; otherwise the loop is terminated and the execution continues with the statement that immediately follows the loop .

3. When the body of the loop is executed , the contol is transfered back to the FAR statement after evaluating the last statement in the loop . now , the control variable is INCREMENTED using an assignment statement such as i = i + 1 and the new value of the control variable is again tested to see whether it satisfies the loop condition . if the condition is satisfied , the body of the loop is again executed . this process continues till the value of the control variable fails to stisfy the test condition .

Consider the following segment of a program.

```
For ( x = 0 ; x <=  9 ; x = x  + 1 )
{
        System .out.println(x);
}
```

This **for** loop is executed 10 times and prints the digits 0 to 9 in one line . the three sections enclosed within parantheses must be seperated y semicolons . note that there is no semicolon at the increment section , **x = x + 1 .**

This **for** loop allows for negative *increments.* For exemples, the loop discussed above can be written as follows :

```
 For ( x = 9 ; x >=  0 ; x = x-1 )
System.out.println(x);
```

This loop is also executed 10 times, but the output would be from 9 to 0 instead of 0 to 9 . note the braces are optional when the body of the loop contains only one statement.

Since the conditional test is always performed at the beginning of the loop , the body of the loop may not be executed at all , if the condition fails at the start, for example .

```
For  ( x = 9 ; x  < 9 ; x = x – 1 )
{
.....
.....
}
```

Will never be executed because the test condition fails at the very beginning itself . .

Let us consider the problem of the sum of the squares of the integers discussed in section. this problem can be coded using the **for** statements as follows . ;

```
.....
.....
Sum  =  0 ;
For  ( n =  1 ; n <= 10 ; n = n + 1 )
{
        Sum  =  sum  +  n*n ;
}

.....
.....
```

The body of the loop

Sum = sum + n*n ;

Is executed 10 times for n= 1, 2 . . . , 10 each times incrementing the **sum** by the square of the value of **n .**

One of the important points about the FOR loop is that all the three actions , namely *intialization, testing and incrementing ,* are placed in the **for** statements itself, thus making them visible to the programmes and users , in one place . the FOR statement and its equivalent of WHILE and DO statements are shown in table

Table comparison of the Three loops

for	*while*	*do*
for (n =1; n<=10;++n)	n =1	n = 1
{	while (n <= 10)	do
.....	{	{
.....	.....	.....
}	.....	.....
	n = n + 1 ;	n = n + 1 ;
	}	}
		While (n< = 10) ;

Programs illustrates the use of **for** loop for computing and printing the " power of 2 " table .

Programs *computing the ' power of 2 ' using for loop*

```
Class ForTest
{
        Public static void main ( Stringargs [ ] )
        {
                long p ;
                int  n;
                double q;
                system.out.println(" 2 to power –n n 2 to power n ")
                p=1;
        for ( n = 0 ; n < 10 ; ++n)
{
        if ( n == 0)
                p = 1;
        else
                p  = p * 2 ;
        q = 1.0 / ( double ) p ;
        system.out.println( "  "  + q + "  " + n + "  " + p );
```

```
        }
    }

}
```

Output of program would be :

2 to power –n	n	2 to power n
1	0	1
0.5	1	2
0.25	2	4
0.125	3	8
0.0625	4	16
0.03125	5	32
0.0152625	6	64
0.00390625	7	128
0.00195313	8	256
0.00195313	9	512

Addtional features of for Loop

the **for**loop has several capabilities that are not found in other loop constructs . for example , more than one variable can be initialized at a time in the **for**statement . the statements

p = 1 ;
for(n = 0 ; n < 17 ; ++n)

can be rewritten as

for(p = 1 , n = 0 ; n< 17 ; ++n)

Notice that the initialization section has two parts p = 1 and n = 1 seperated by a *comma*. Like the initialization section , the increments sections may also have more than one part. For example, the loop

```
For ( n = 1, m = 50 ; n = n + 1 , m = m-1)
{
        .....
        .....
}
```

Is perfectly valid. The multiple arguments in the increments section are seperatedby *commas.*

The third feature is that the test condition may have any compound reation and the testing need not be limited only to the loop control variable. Consider the example that follows :

```
Sum = 0 ;
For( i = 1 , i < 20 && sum < 100 ; ++i)
```

```
{
        .....
        .....
}
```

The loop uses a compound test condition with the control variable i and external variable **sum**.

The loop is executed as long as both the conditions i<20 and sum<100 are true .the sum is evaluated inside the loop .

It is also permissible to use expression in the assignment statements of initialization and increment sections. For example , a statements of the type .

For (x = (m +n)/2; x > 0 ; x = x/2)

Is perfectly valid.

Another unique aspect of **for** loop is that one or more section can be omitted, if necessary .consider following statements :

```
.....
.....
m=5;
for( ;m ! = 100 ; )
{
        System.out.println(m );
        m = m + 5 ;
}
.....
.....
```

Both the initialization and increment sections are omitted in the FOR statement. The initialization has been done before the FOR statement and the control variable is incremented inside the loop .in such cases, the sections are left blank. However, the semicolons seperating the sections must remain . if the test condition is not present , the FOR statement sets up an infinite loop .

We can set up time delay loops using the null statements as follows :

```
For ( j = 1000 ; j > 0 ; j = j – 1 )
;
```

This loop is executed 1000 times without producing any output; it simply causes a time delay. Notice that the body of the loop contains only a semicolon , known as an *empty* statement . this can also be written as

```
For ( j = 1000 ; j>0; j = j – 1 );
```

This implies that the compiler will not give an error message if we place a semicolon by mistake At the end of a FOR statement . the semicolon will be consider as an *empty* statement and the program may produce some nonsense. .

Nesting of for Loops

Nesting of loops, that is , one FOR statements within another FOR statement, is allowed in Java . wehava used this concept in Program 7.2 . similarly, FOR loops can be nested as follows :

```
.....
.....

For  ( i = 1 ; i < 10 ; ++ i )
{
        .....
        .....

                For  ( j = 1 ; j < ! = 5 ; ++ j )
                {

                .....
                .....                                   inner loop

        }
        .....
        .....


}
```

.....
.....

The loops should be properly indented so as to anable the reader to easily determine which statements are continued with in each FOR statement.

A program segment to print a multiplication table using FOR loops is shown below :

.....
.....

```
For ( row = 1 ;   row  < = ROWMAX : ++row)
{
        For (column  = 1 ; column  < = COLMAX : ++column)
        {
                Y= row * column

                System.out.print( " + Y);
}
```

.....
.....

The outer loop controls the rows while the inner one controls the columns.
Program 7.4 illustrates the use of nested FOR loops

Programs *displaying right angle triangle of @ using nested* ***for*** *loops*

```
Class nested_loop
{
        Public static void main ( Stringargs [  ] )
        {
                Intp , q ;
                System.out.print( " The right angle triangle of @ is shown below :\n" );
                For ( p = 1 ; p  <=9; p ++)
                {
                        For ( q = 1 ; q < = p ; q++)
                        {
                                System.out.print( " @ " );
                        }
                        System.out.println( "   " ) ;
```

```
        }
    }
}
```

Output of program
The right angle trangle of @ is shown below :

```
@
@@
 @@@
@@@@
@@@@@
@@@@@@
@@@@@@@
@@@@@@@@
@@@@@@@@@
```

In the above program , the outer **for** loop (with variable **p)** controls the number of rows whereas the inner **for** loop (with variable **q**) controls the number of columns.

The Enhanced for LOOP

The enhanced **for** loop , also callled *foreach loop ,* is an exended language features introduced with the J2SE 5.0 release . this feature helps us to retrive the array of elements efficiently rather than using array indexes. We can also use this features to eliminate the iterators in a for loop and to retrive the elements from a collection .the enhanced for loop takes the following form :

```
For ( type Identifier : Expression )
{
        // statements ;
}
```

Where, TYPE represents the data type or object used ; IDENTIFIER refers to the name of a variable ; and EXPRESSION is an instances of the java.lang . Iterable interface or an array.

For example , consider the following statements :

```
Intnumarry[3] = {56, 48 , 79 } ;
For (int k=0; k<=3; k++)
{
        If  (numarray [k1>50 &&numarray[k]<100)
        {
                System.out.printl( " The selected value is " + numarray [k ] );
`       }
}
```

Which is equivalent to the following code :

```
Intnumarray[ 3] = { 56, 48 , 79 } ;
For(int k : numarray )
{
                If ( k> 50 && k < 100 )
                {
                        System.out.println( " the selected value is " +k) ;
                }
}
```

Thus , we can use the enchanced for loop to track the element of an array efficiently . in the same manner , we can track the collection elements using the enhanced for loop as follows :

```
Stack samplestack = new stack ();
Samplestack.push( new Integer (56));
Samplestack.push( new Integer (48));
Samplestack.push( new Integer (79));
For ( objectobj : samplestack  )
{
        System.out.println(obj );
}
```

Programs *use of enhanced for loop to retrive the elements of arrays*

```
Import java.util.*;

Class EnhanceForLoop
{
        Public static void  main ( String args [ ] )
        {
        System.out.println( );
        String
        States[] = { "Tamilnadu", "Andhrapradesh", "UttarPradesh", "Rajasthan"};
        For (  int i =0 ; i <states.length ; i++ )
                {
```

```
                    System.out.println(" STANDARD FOR – LOOP : STATE NAME :"+
states[i]);
                }

        System.out.println( );

        for(String i:states)  // enhanced for loop
        {
                System .out. Println (" Enhanced for-loop : state name : " + i );
        }

        System.out.println( );

        ArrayList< string > cities = new ArrayLists< string > ();

                Cities.add("Delhi");
                Cities.add("Mumbai");
                Cities.add("Calcutta");
                Cities.add("Chennai");
                System.out.println( );

        For (  int i =0 ; i <cities.size( ) ; i++ )
                {
                    System .out.println(" STANDARD FOR – LOOP : CITY NAME :"+
cities.get[i] ) );
                }

        System.out.println( );

        For (String city : cities)   // enhanced for loop
                    System .out.println(" STANDARD FOR – LOOP : CITY NAME : "+ city );

        System.out.println( );
        System.out.println( " In collections "  );
        System.out.println( );
                Printcollection( cities) ;
}
        Public  static<AnyType> void printcollection(Collection<AnyType> c )
{

For (AnyTypeval : c)
        System.out.println (val)) ;
        }
}
```

The output of the program is as follows

Standard for – loop : state name : Tamil Nadu
Standard for – loop : state name : Andhra Pradesh
Standard for – loop : state name : Uttar Pradesh
Standard for – loop : state name : Rajasthan

Enhanced for – loop : state name : Tamil Nadu
Enhanced for – loop : state name : Andhra Pradesh
Enhanced for – loop : state name : Uttar Pradesh
Enhanced for – loop : state name : Rajasthan

Standard for – loop : city name : Delhi
Standard for – loop : city name : Mumbai
Standard for – loop : city name : Calcutta
Standard for – loop : city name : Chennai

Enhanced for – loop : city name : Delhi
Enhanced for – loop : city name : Mumbai
Enhanced for – loop : city name : Calcutta
Enhanced for – loop : city name : Chennai

In Collections :
Delhi
Mumbai
Calcutta
Chennai

JUMPS in LOOPS

Loops perform a set of operations repeatedly until the control variable fails to satisfy the test Condition .the number of times a loop is repeated is decided in advance and the test condition is written to achieve this. Sometimes , when executing a loop it becomes desirable to skip a part of the loop or to leave the loop as soon as a certain condition occurs . for example loop written for reading and testing the names a 100 times must be terminated as soon as the desired name is found . javapermits a jumps from one statement to the *end* or *beginning* of a loop as well as a *jump out of a loop* .

Jumping Out of a Loop

An early exit from a loop can be accomplished by using the **break** statement. We have already seen the use of the **break** in the **switch statement .** this statement can also be used within **while, do** or **for** loops as illustrated in fig . 7.2 .

When the **break** statement is encounter inside a loop , the loop is immediately exited and the programs with the statement immediately following the loop . when the loops are nested, the break would only exit from the loop containing it . that is , the break will exit only a single loop .

Skipping a Part of a Loop

During the loop operations, it may be necessary to skip a part of the body of the loop under certain conditions. For example , in processing of applications for some job , we might like to exclude the processing of data of applicants belonging to a certain category. On reading the category code of an applicant , a test is made to see whether his application should be considered or not. If it is not be considered, the part of the programs loop that processes the application details is skipped and the execution continues with the next loop operation .

Like the **break** statement, java supports another similar statements called the **continue** statement .however, unlike the **break** which causes the loop to be terminated , the **continue** as the name implies , causes the loop to be continued with the next iteration after skipping any statements in between . the continue statement tells the compiler .” SKIP THE FOLLOWING STATEMENTS AND CONTINUE WITH THE NEXT ITERATION “ .the format of the **continue** statement is simply.

While (. . . . . .) { . . . . If (condition) **break;** . . . . } . . . . . . (a)	do { . . . . If (condition) **break ;** . . . . } while (. . . . . .) . . . . . . (b)

```
For  ( . . . . . .  .)
{
      . . . . . .
      . . . . . .. . . .
       If ( error  )
break ;
      . . . . . .. . . . . .
      . . .  .. . . . . . . . . . .
}
      . . . . . .. . . . . .. .
                              ( c )
```

```
For  ( . . . . . .  .)
{
      . . . . . .
For  ( . . . . . .. . . )
      {
. . . . . .. . . . . .
if ( condition )
break ;
      . . . . . .. . . . . .. .
    }
      . . . . . .. . . . . .. .
}
   ( d )
```

Fig. ***Exiting a loop with break statement***

The use of the **continue** statement in loops is illustrates in fig 7.3 .in while and do loops ,continue causes the control to go directly to the *test condition* and then to continue the iteration process . in the case of **for** loop, the *increment* section of the loop is executed before the *test condition* is evaluated .

```
While  ( test condition )do
{
   . . . . . .
       If (  . . . . . ..  . )
       Continue ;
       ... . . . . .
       .....
}
       ( a )
```

```
{
                .....
       if ( .....)
       continue;
                .....
                .....
} while ( test condition ) ;
       ( b )
```

```
For ( initialization ; test condition  ;  increment )
{
       .....
       If ( .....)

       Continue ;
       .....
       .....
}
```

(c)

Fig . By passing and continuing in loops

LABELLED LOOPS

In java , we can give a label to a block of statements. A label is any valid java variable name .to give a label to a loop , place it before the loop with a colon at the end . example :

```
Loop 1 : for ( . .....)
         {
                  .....
                  .....
         }

         .....
```

A block of statements can be labelled as shown below :

```
Block 1: {
                  .....
                  .....
Block 2:          {
                  .....
                  .....
                  }

                  .....
                  .....
         }
```

We have seen that a simple **break** statement causes the controls to jump outside the nearest loop and a simple **continue** statement restarts the current loop . if we want to jump outside a nested loops or to continue a loop that is outside the current one . then we may have to use the labelled **break** and labelled **continue** statements . examples :

```
Outer : for ( int m = 1; m < 11 ; m ++)
         {
                  For ( int  n  = 1 ; n  <  11 ;  n ++ )
                  {
                  System.out.print( "   " +  m * n ) ;
                  If  ( n == m )
```

```
                    Continue  outer ;
            }
}
```

Here , the continue statements terminates the inner loop when n = m and continues with the next iteration of the outer loop (counting m) .

Another example :

```
Loop 1 :        for  (  int  i = 0 ; i < 10 ; i ++ )
                {
                        While  ( x < 100 )
                        {
                                Y = i * x;
                        If ( Y> 500 )

                        break loop1;
                        .....
                        .....
                }

                        .....
                        .....
                }

                .....
```

Here , the label **loop1** labels out of the loops . programs 7.6 illustrates the use of **break** and **continue** statements .

Program : Use of continue and break statement

```
Class ContinueBreak
{
        Public static void main ( String args [  ] )
{
        Loop 1 : for  (  int  i =1 ; i < 100 ; i ++ )
                {
                        System.out.printn (  "  " );
                        If  ( i > = 10 ) break ;
                        For ( int  j = 1 ; j < 100 ; j ++ )
```

```
                    {
                        System.out.println ( “  *  “);
                    }

                If ( j == i )
                    Continue LOOP1;
                    }
                }

                System.out.println( “ TERMINATION BY BREAK “ );
        }
}
```

Program : produces the following output :

```
*
**
***
****
*****
******
*******
* * * * * * * *
* * * * * * * * *
```

Termination by BREAK