UNIT – I CONTENTS

1. FUNDAMENTALS OF OBJECT-ORIENTED PROGRAMMING

- Introduction
- Object-Oriented Paradigm
- Basic Concepts of Object-Oriented Programming
- Benefits of OOP
- Applications of OOP

2. JAVA EVOLUTION

- Java History
- Java Feature
- How Java Differs from C and C++
- Java and Internet
- Java and World Wide Web
- Web Browsers
- Hardware and Software Requirements
- JDK (Java Environment)

3. OVERVIEW OF JAVA LANGUAGE

- Simple Java Programming
- Java Program Structure
- Java Tokens
- Java Statements
- Implementing a Java Program
- Java Virtual Machine
- Command Line Arguments

4. CONSTANTS, VARIABLES, AND DATA TYPES

- Constants
- Variables
- Data Types
- Declaration of Variables
- Giving Values to Variables
- Scope of Variables
- Symbolic Constants
- Type Casting
- Getting Values of Variables

FUNDAMENTALS OF OBJECT-ORIENTED PROGRAMMING

Introduction:

One characteristic that is constant in the software industry today is the "change". Change is one of the most critical aspects of software development and management.

New tools and new approaches are announced almost every day.

Most important among them are *maintainability, reusability, portability, security, integrity, and user friendliness of software products.*

Since the invention of the computer, many programming approaches have been tried. *These include techniques such as modular programming, top-down programming, bottom-up programming and structured programming.* The primary motivation in each case has been the concern to handle the increasing complexity of programs that are reliable and maintainable. These techniques became popular among programmers over the last two decades.

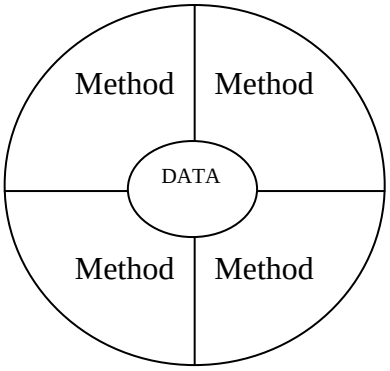
Structured programming proved to be a powerful tool that enabled programmers to write moderately complex programs fairly easily. However, as the programs grew larger, even the structured approach failed to show the desired results in terms of bug-free, easy-to-maintain, and reusable programs.

Object-Oriented Programming (OOP) is an approach to program organization and development, which attempts to eliminate some of the pitfalls of conventional programming methods by incorporating the best of structured programming features with several new concepts. It is a new way of organizing and developing programs and has nothing to do with any particular language. However, not all languages are suitable to implement the OOP concepts easily. C++ is basically a procedural language with object-oriented extension. The *latest one added to this list is JAVA, a pure object oriented language.*

Object-Oriented Paradigm:

The major objective of object-oriented approach is to eliminate some of the flaws encountered in the procedural approach. OOP treats data as a critical element in the program development. *OOP allows us*

to decompose a problem into a number of entities called Objects and then build data and functions (known as methods in Java) around these entities. The combination of data and methods make up an object.



Object = Data + Methods

The data of an object can be accessed only by the methods associated with that object. However, methods of one object can access the methods of other objects. *Some of the features object-oriented paradigms are:*

- Emphasis is on data rather than procedure.
- Programs are divided into what are known as Objects.
- Data structures are designed such that they characterize the objects.
- Methods that operate on the data of an object are tied together in the data structure.
- Data is hidden and cannot be accessed by external function.
- Objects may communicate with each other through methods.
- New data and methods can be easily added whenever necessary.
- Follows bottom-up approach in program design.

Object oriented programming means

Object based features + Inheritance + Dynamic binding

Object based programmings are data encapsulation, data hiding, automatic initialization, operator over loading.

BASIC CONCEPTS OF OBJECT ORIENTED PROGRAMMING:

Basic concepts of OOP are

1. Object
2. Class
3. Data Abstraction
4. Data Encapsulation
5. Inheritance
6. Polymorphism
7. Message Communication
8. Dynamic Binding

1) Object:

Definition (1):

"Object is defined as an entity that can store data, send and receive messages."

Definition (2):

"Object is defined as an identifiable entity with some characteristics and behavior."

Definition (3):

"Anything can be object", in other words an object be a person, a place or a thing with which the computer must be deal.

Definition (4):

"*Objects are the basic runtime entities in an object oriented system. "*

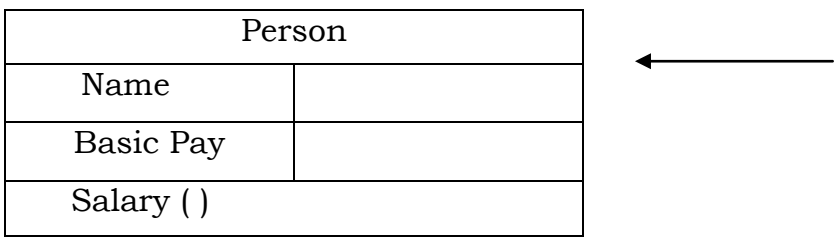
Definition (5):

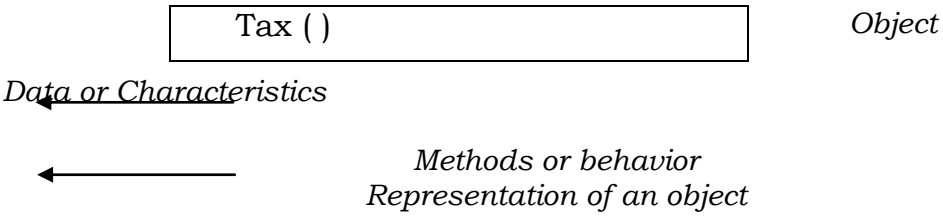
"Object = Data + Methods."

Object may represent a person, a place, a bank account, a table of data or any time that the program may handle.

When a program is executed, the objects interact by sending messages to one another. For example, 'customer' and 'account' are two objects in a banking program, then the customer object may send a message to the account object requesting for the balance. *Each object contains data and code* to manipulate the data.

.





2) Class:

Definition(1):

"A class is a collection of objects of similar type. "

Definition(2):

"A group of objects that share common properties and relationships."

Definition(3):

"The objects with the same data structure (attributes) and behavior (Operations) are grouped into a class."

In JAVA, a class is a new data type that contains member variables and member functions that operate on the variables. A class is defined with the key word class.

Example:

Person objects — abstract into → person class

Person may be
Men or women

attributes: Name, Age, Sex
operations: Speak( )
listen( )
walk( )

Vehicle objects — abstract into → Vehicle class

vehicle may be
two wheeler (or)
three wheeler (or)
four wheeler

attributes: Name, Model, Color
Operations: Start( )
Stop( )
Accelerate( )

3) Encapsulation:

The wrapping up of data and methods into a single unit is known as encapsulation. Data encapsulation is the most striking feature of a

class. The data is not accessible to the outside world and only those methods, which are wrapped in the class, can access it.

(OR)

Encapsulation is achieved by using the class, which combined data and functions that operate on the data.

(OR)

Encapsulation or data hiding is the mechanism that associates the code and the data that it manipulates into a single unit and keeps them safe from external interference and misuse. The class features of JAVA implements the concept of encapsulation by providing "Access specifiers". Access specifiers control the visibility status of the members of a class. There are three access specifiers in JAVA

- Private
- Public
- Protected

Example:

In a company, different departments work independently, with their own data. One department cannot access data of the other departments directly. Rather a request is made for the required data and the data is handed over by the members of the requested department. We may say that department data and department employees are encapsulated.

NOTE:

Encapsulation is a way of implementing abstraction. Abstraction means picking out the relevant details of an object. If this has to be done, then the irrelevant details must be hidden from the user. This is done by encapsulation.

4) Data Abstraction:

Abstraction refers to the act of representing essential features without including the background details or explanations.

(OR)

In OOP, the data abstraction is defined as on collection of data and methods (functions).

Example:

When professor X wanted to send the parcel to Dr. Y, through C-couriers. Let us understand what essential features and what non-essential features. In this example *send the parcel* to Dr. Y is *essential* and the *procedure for delivering* the parcel to Dr. Y is *irrelevant*.

5) Inheritance:

Inheritance is the process by which objects of one class acquire the properties of objects of another class.

(OR)

Inheritance is the property that allows the reuse of an existing class to build a new class.

In OOP, the concept of inheritance provides the idea of **reusability.** This means that we can add additional features to an existing class without modifying it. This is possible by deriving a new class from the existing one. The new class will have the combined features of both the classes. Thus the real appeal and power of the inheritance mechanism is that it allows the programmer to reuse a class that is almost, but not exactly, what he wants, and to tailor the class in such a way that it does not introduce any undesirable side effects into the rest of the classes.

In JAVA the existing class is known as base class and derived class is known as sub class.

6) Polymorphism:

Definition(1):

Polymorphism is another important OOP concept. *Polymorphism means the ability to take more than one form.*

Definition(2):

Polymorphism means different objects responding to the same message in different manners.

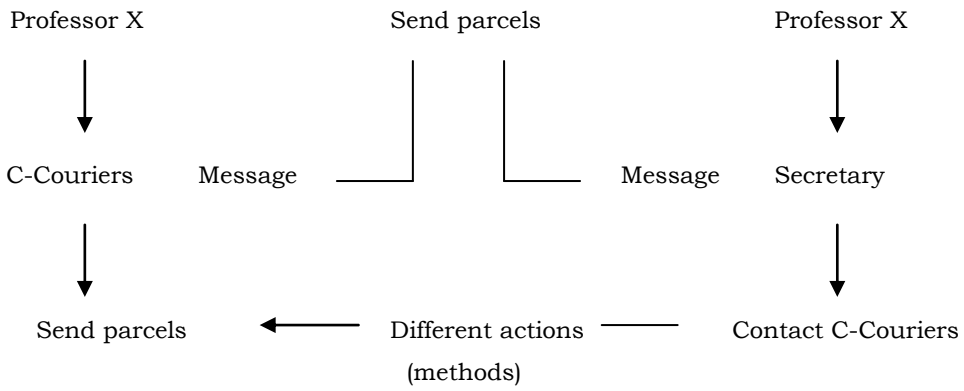
Polymorphism is extensively used in implementing inheritance.

Example: 1) Consider the operation of addition. For two numbers, the operation will generate a sum. If the operands are strings, then the operation would produce a third string by concatenation.

Example: 2) When Professor X wanted to send the parcel to Dr. Y, through C-Couriers, he could have chosen either of the two options:

- He himself could have contacted C-Couriers who would have responded to his request by sending the parcel to Dr. Y in Mumbai.

- He could have asked his secretary, Miss S to send the parcel to Mumbai. Miss S will then respond differently by contacting C-Couriers. Thus the same message by the object, Professor X, when sent to two different objects, C-Couriers and Miss S, will produce 2 different actions (methods) as shown in the below figure.



This represents the concept of polymorphism. Different objects respond indifferent ways to the same message.

7) Dynamic Binding:

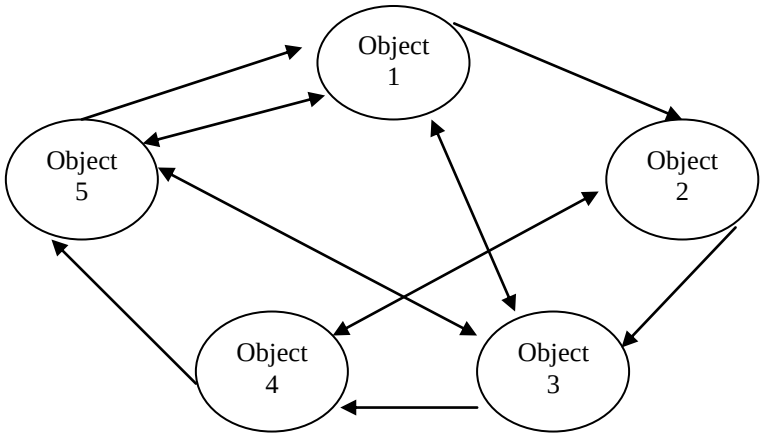
Binding refers to the linking of a procedure call to the code to be executed in response to the call. *Dynamic binding means that the code associated with a given procedure call is not known until the time of the call at runtime.* It is associated with polymorphism and inheritance.

8) Message communication:

An object-oriented program consists of a set of objects that communicate with each other. The process of programming in an object-oriented language, therefore, involves the following basic steps:

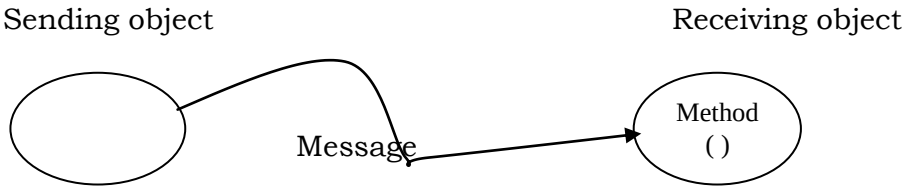
1. Creating classes that define objects and their behavior.
2. Creating g objects from class definitions.
3. Establishing communication among objects.

Objects communicate with one another by sending and receiving information much the same way as people pass messages to one another as shown in below figure.



Network of objects communicating between them

A message for an object is a request for execution of a procedure, and therefore will invoke a method in the receiving object that generates the desired result, as shown in below figure.



Message triggers a method

Message passing involves specifying

The name of the object

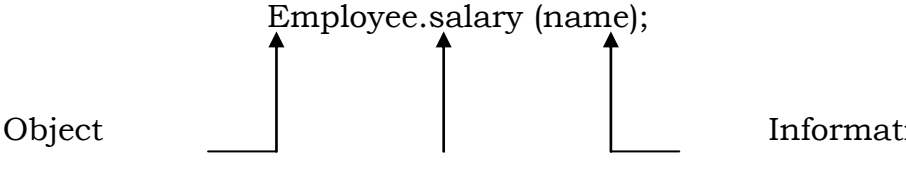
The name of the method and

The information to be sent.

Example:

Employee. Salary (name);

Here, Employee is the object, salary is the message and name is the parameter that contains information.



Message

Objects have a life cycle. They can be created and destroyed. Communication with an object is feasible as long as it is alive.

BENEFITS OF OOP:

OOP offers several benefits to both the program designer and the user. The new technology promises greater programmer productivity, better quality of software and lesser maintenance cost. The principal advantages are:

- Through inheritance, we can eliminate redundant code and extend the use of existing classes.
- We can build programs from the standard working modules that communicate with one another, rather than having to start writing the code forms scratch. This leads to saving of development time and higher productivity.
- The principle of data hiding helps the programmer to build secure programs that cannot be invaded by code in other parts of the program.
- It is possible to have multiple objects to coexist without any interference.
- It is possible to map objects in the problem domain to those objects in the program.
- It is easy to partition the work in a project based objects.
- Object-oriented systems can be easily upgraded from small to large systems.
- Message passing techniques for communication between objects make the interface descriptions with external systems much simpler.
- Software complexity can be easily managed.

APPLICATIONS OF OOP:

Applications of OOP are beginning to gain importance in many areas. The promising areas for application of OOP include:

- Real-time systems
- Simulation and modeling
- Object-oriented databases
- Hypertext, hypermedia and expertext
- Artificial Intelligence and expert systems
- Neural networks and parallel programming
- Decision support and office automation systems
- CIM/CAD/CAD system

UNIT-I :: CHAPTER – II

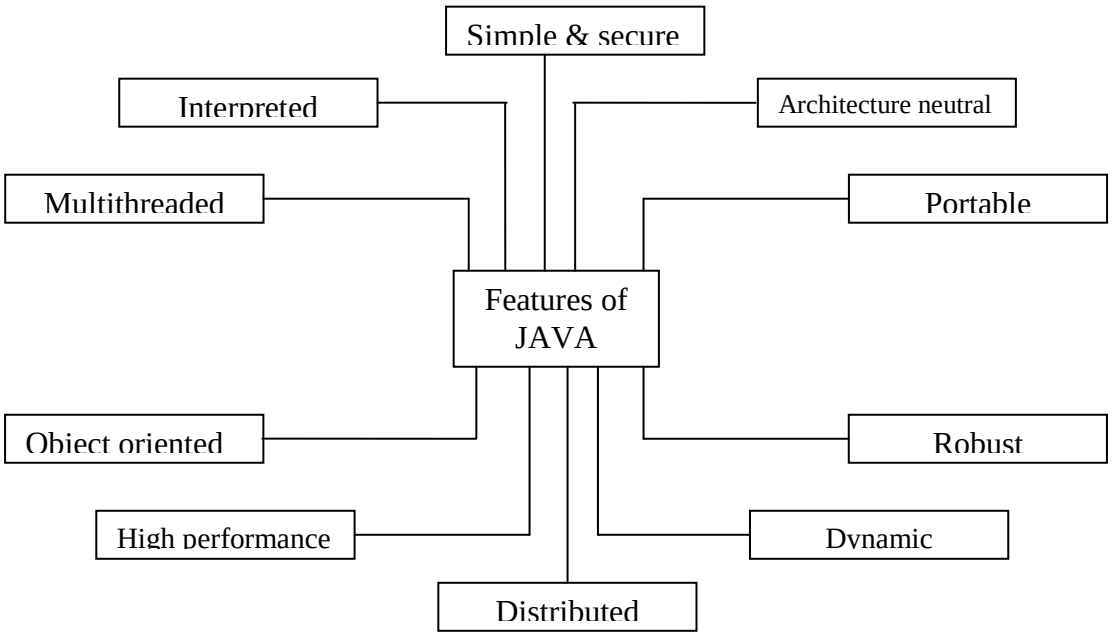
JAVA is an object oriented programming language developed primarily by "James Gosling " and colleagues at Sun Micro Systems in the year 1991. The language, initially called Oak (named after the Oak trees outside Gosling's office).

In 1991, the Sun Micro Systems (Broom field, Colorado, USA) developed a complete language as a part of research work to develop software for consumer electronics. It was developed as a full fledged programming language in which one can accomplish the same sorts of task and solve the similar problem that one can in other programming language such as BASIC, C++ etc.

The platform independent is one of the most significant advantages that JAVA has over other languages. It is the capacity of moving easily from one computer system to another. JAVA being an top language on encapsulations many features of C++. Originally, JAVA was designed to execute applets, down loaded which web browsing. But gradually, the language has been gaining wide acceptance as a programming language, very often replacing C (or) C++.

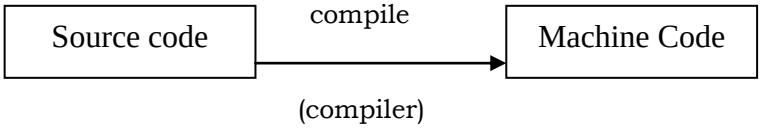
Basic Features:

The following figure shoes the various features of JAVA. We will take up each of these in detail.

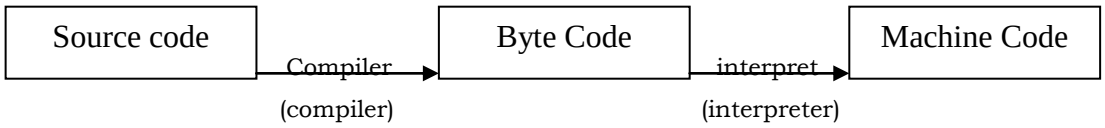


Interpreted:

JAVA is both compiled and interpreted. That is, a Java source code (program) is first compiled and then interpreted. This is made possible by the Java Virtual Machine (JVM). The Java Virtual Machine, converts the intermediate byte code into the machine code.



As shown in the above figure a C++ source code is only compiled, not interpreted.



A Java source code as shown in above figure is first compiled to produce the byte code which is then interpreted using the Java interpreter to produce the machine code.

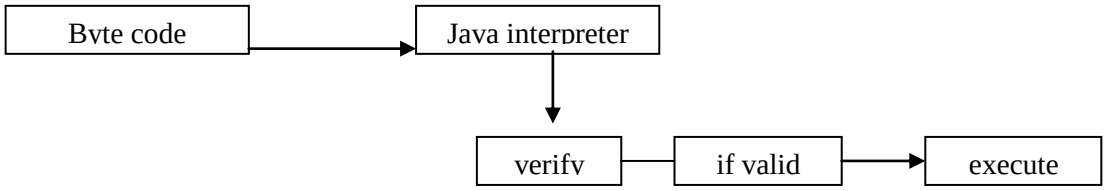
Thus, a *Java source code goes through two processes; compilation and interpretation.* The output of the compilation process is the byte code, which is machine-independent. This makes a Java program capable of being executed on any machine, provided that Java interpreter, that is, Java Virtual Machine is present (installed) on

that machine. The Java Virtual Machine (Java interpreter) is not the same for all machines. It depends on the hardware architecture and the operating system of a machine. *The Java interpreter is named java and the Java compiler is called javac.*

Simple and Secure:

Java is a simple language. Since it originated from C++, anyone familiar with C++ can learn Java easily. Besides, Java does not support operator overloading. This means, we cannot redefine the meaning of the basic components of the language. Complex codes are thus avoided.

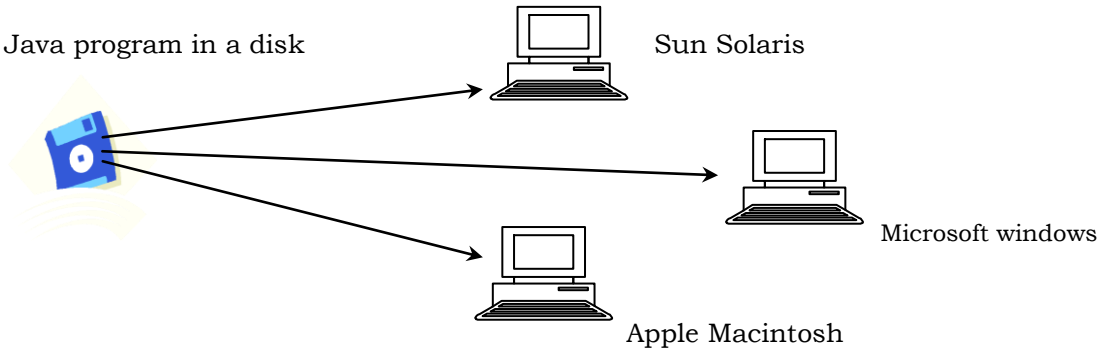
Java is highly secure. The security of Java is two-fold. Firstly, the Java interpreter verifies the byte code before executing it. If the byte code turns out to be invalid, it will not be executed at all. The following figure shows this schematically.



Secondly, Java does not allow programmers to interact with the memory of the system. That is, one cannot write a Java code that accesses the memory of the system.

Architecture Neutral Language:

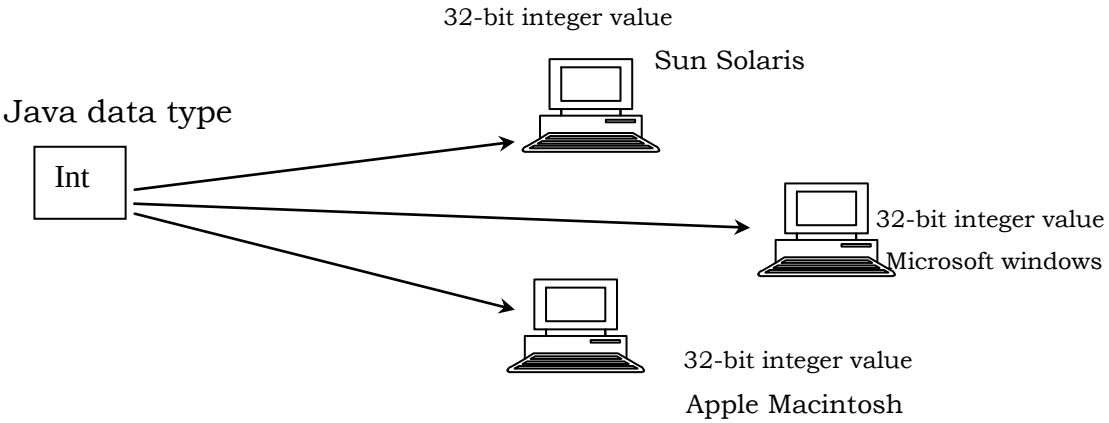
By architecture neutral, we mean that *Java is not dependent on the architecture of a particular machine. The same Java program can be run on a variety of CPUs and operating systems*. The Java compiler converts the Java source code into byte code, which is architecture neutral. If Java interpreter exists on a machine, we can run Java byte code on that machine. This is shown in below figure.



Portable:

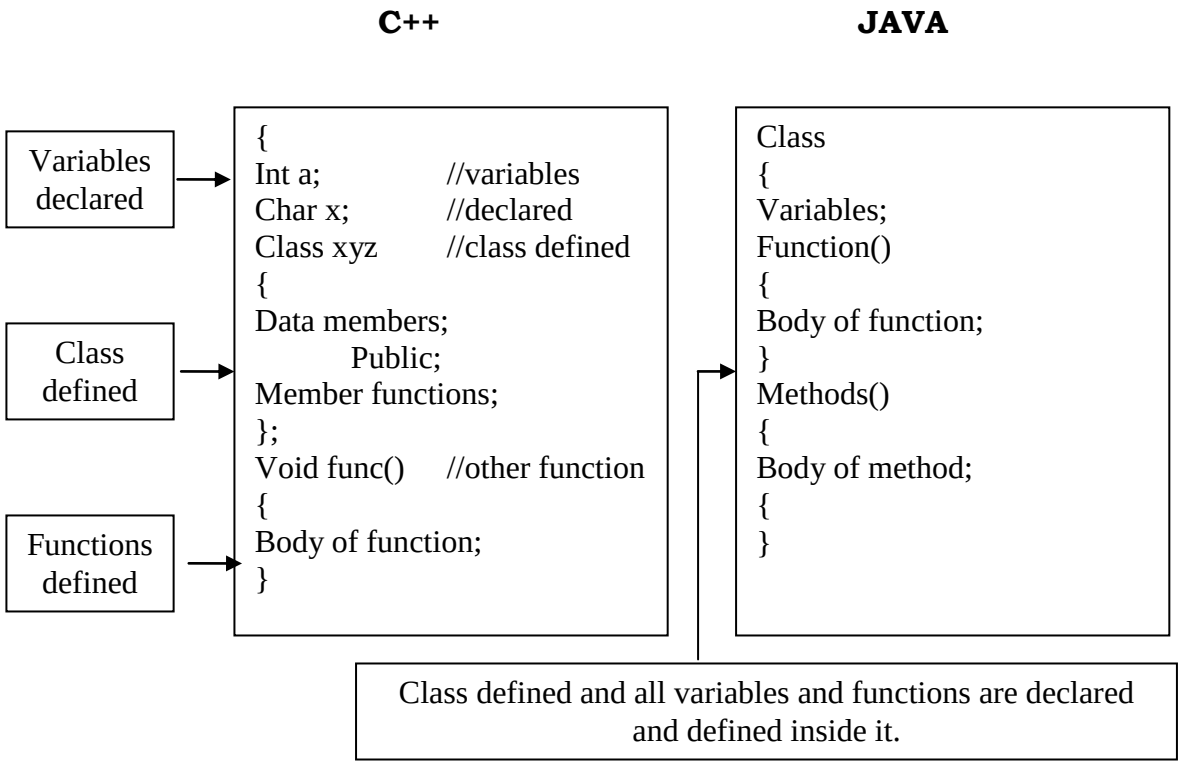
Java is a portable language. Apart from the byte code, which is machine independent and hence, can be implemented on any system with JVM installed, the data types of Java also account for its portability. The size of the basic data types in Java is compatible with for all systems making Java programs highly portable.

For example, in Java, an int is always a 32-bit value, no matter in which system it is used. Similarly, the size of other data types is also explicitly specified. The following figure represents the above feature of Java.



Object-Oriented:

Like C++, Java too is an object oriented programming language. However unlike C++, the entire code is written within a class. A class is the smallest unit in a Java program. All variables and functions are declared as well as defined within the class only. By totally object-oriented, we mean that each and every line of code in Java is written inside a class. A class is the smallest unit of a Java program. All variables and functions are defined and declared inside a class.



Robust:

Programs written in Java are robust, that is, they are reliable and their chances of failing under known circumstances are negligible.

Multithreaded:

Multithreading is a major feature of Java. *Multithreading means handling many tasks simultaneously*. In a non-threaded environment (one without multithreading), only one line of execution takes place at a time. Only after the computer has finished executing one line will it proceed to the next line. Hence one task has to be completed before another one can be performed.

In a multithreaded environment, we need not wait for the application to finish one task before starting another.

A good example of multithreading is that when you are downloading a file from the internet, you can surf through a web page.

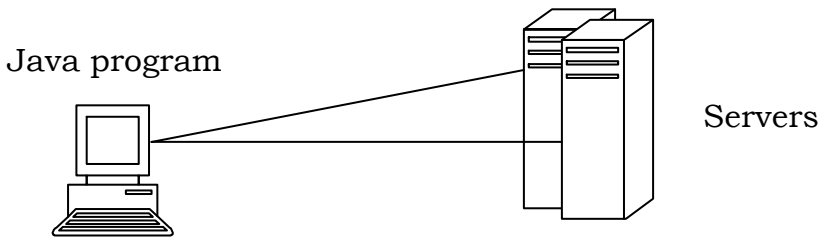
High performance Speed:

Performance speed is an important feature of a programming language. Since the Java source code is first compiled and then interpreted, Java programs are executed very fast.

Distributed:

Java is built with emphasis on network communication. It has been designed to share both data and programs. Java programs are capable of accessing data across the net as easily as they access data from a local system. Java has various libraries to deal with network protocols. A protocol refers to a set of rules and regulations to be followed while transferring data on the net. These protocols are referred to by a specific name. As a result, Java applications can access objects across the internet. The following list display some of the popularly used protocols.

1. TCP / IP -- Transmission Control Protocol/Internet Protocol
2. FTP -- File Transfer Protocol.
3. HTTP -- Hyper Text Transfer Protocol.



Dynamic:

Java is dynamic language. A dynamic language is cap[able of linking libraries, functions, methods as well as objects (depending on if it is an object oriented programming language. Java, too, can dynamically (i.e. at run-time) link in new class libraries, methods, and objects.

How JAVA differs from C and C++

Java was modeled after C and C++ languages; it differs from C and C++ in many ways. Java does not incorporate a number of features available in C and C++. The following are the major differences between C/C++ and Java languages.

Java and C

- Java is an object oriented language where as C is Structured language
- Java does not include the C unique statement keywords goto, sizeof and typedef.
- Java does not contain the data types struct, union and enum
- Java does not define the type modifiers keywords auto, register, signed and unsigned.
- Java does not support an explicit pointer type.
- Java does not have a preprocessor and therefore we cannot use #define, #include and #ifdef statements.
- Java adds new operator such as instanceof and>>>
- Java adds labeled break and continue statements.
- Java adds many features required for object –oriented programming.

Java and C++

Java is true object oriented language while C++ is basically C with object oriented extension. However, there are a few basic points where Java differs from C++. These are listed below.

- There are no header files in Java
- The programmer does not have worry about pointers because in Java. i.e. Java does not use pointers.

- Java does not support operator overloading
- The concept of multiple inheritance is not supported by Java.
- All Java programs elements must exist inside a class. Therefore, Java has no global functions or variables.
- Java does not support templates.
- Java has replaced the destructor function with a finalize ( ) function.
- C++ which is only compiled language, Java is both compiled and interpreted.
- The + operator can be used to concatenate strings in Java.
- Class definitions are similar in Java and C++, but there is no closing semicolon (;) in Java.
- Modulus operator can be applied to float values in Java which is not permitted in C++.
- There is no goto in Java. The break and continue statements can be used to jump out of the loops in the programs.
- There is no scope resolution operator (**::**) in Java.

Java and Internet

Java is designed in such a way that it can easily create Internet –enabled applications. Java programs are of the following two types.

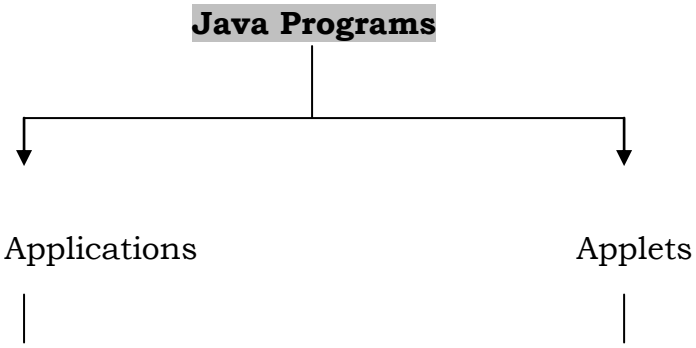
1. Applications:

These are stand-alone programs, which we can execute from the command prompt.

2. Applets:

These are programs that run in web-browsers.

The following figure shows the distinction between applications and applets.



Java is often referred to as an *internet programming language* because it supports applets. Applets form one of the most distinguishing features of Java. Applets are programs are generally executed on the web browser. Java also provides a utility called Applet Viewer to execute applets.

Java and World Wide Web

World Wide Web (WWW) is an open ended information retrieval system designed to be used in the internet's distributed environment. This system contains what are known as web pages that provide both information and controls.

Web pages contain HTML tags that enable us to find, retrieve, manipulate and display documents worldwide.

Java was meant to be used in distributed environments such as Internet. Since, both the web and Java share the same philosophy. Java could be easily incorporated into the web system. Before Java, the World Wide Web was limited to the display of still images and texts. However the incorporation of Java into Web pages has made it capable of supporting animation, graphics, games, and a wide range of special effects. With the support of Java ,the web has become more interactive and dynamic.

Java communicates with a webpage through a special tag called <APPLET>. The following are the communication steps.

1. The user sends a request for an HTML document to the remote computer's web server.
2. The web server is a program that accepts a request, processes the request, and sends the required document.

3. The HTML document is returned to the user's browser. The document contains the APPLET tag, which identifies the applet.
4. The corresponding applet byte code is transferred to the user's computer.
5. The Java enabled browser on the users computer interprets the byte codes and provides output.

WEB BROWSERS

Web browsers are used to navigate through the information found on the net. They allow us to retrieve the information spread across the Internet and display it using the Hyper Text Markup Language (HTML).

Examples of Web Browsers are

- Hot Java
- Netscape Navigator
- Internet Explorer

Hot Java

Hot Java is the Web browser from Sun Microsystems that enables the display of interactive content on the web, using the Java language. Hot java is written entirely in Java and demonstrates the capabilities of the Java programming language. Its biggest draw is that it was the first web browser to provide support for the Java language, thus making the web more dynamic and interactive.

Netscape Navigator

Netscape Navigator, from Netscape Communications Corporation, is a general purpose browser that can run Java applets. With versions available for Windows 95, NT, Solaries and Apple Macintosh, Netscape Navigator is one of the most widely used browsers today. The useful features of Netscape Navigator are visual display about downloading process and indication of the number of bytes downloaded. It also supports JavaScript, a scripting language used in HTML documents

Internet Explorer

Internet Explorer is another popular browser developed by Microsoft Windows 95 and NT workstations. Both the Navigator and Explorer use toolbars, icons, menus and dialog boxes for easy navigation.

Hardware and Software Requirements

Java is currently supported on Windows 95, Windows NT, Sun Solaris, Macintosh, and UNIX machines.

The minimum hardware and software requirements for Windows 95 version of Java are as follows.

1. IBM-compatible 486 system
2. A hard drive
3. Minimum of 8MB memory
4. A CD-ROM Drive
5. Windows 95 software
6. Mouse
7. Sound card, if necessary

Java Environment

Java environment has a number of development tools and hundreds of libraries. All the development tools are the part of a system known as Java Development Kit (JDK).Classes and methods are the part of Java Standard Library (JSL), which are also called Application Interface (API).

Java Development Kit (JDK)

Java Development Kit provides a number of tools which are essential to develop and run Java Programs.

These tools are:

java : *Java Interpreter*. It runs applets and applications.

javac : *Java compiler*. It translates Java Source codes to byte code files, so that the interpreter could read them

javadoc : It *creates HTML formats* documents from Java Source Code files.

javah : It produces *header files*.

javap : It is a *Java disassembler*. It encodes byte codes to program files.

jdb : *Java debugger*. It finds errors and removes them from

the programs.

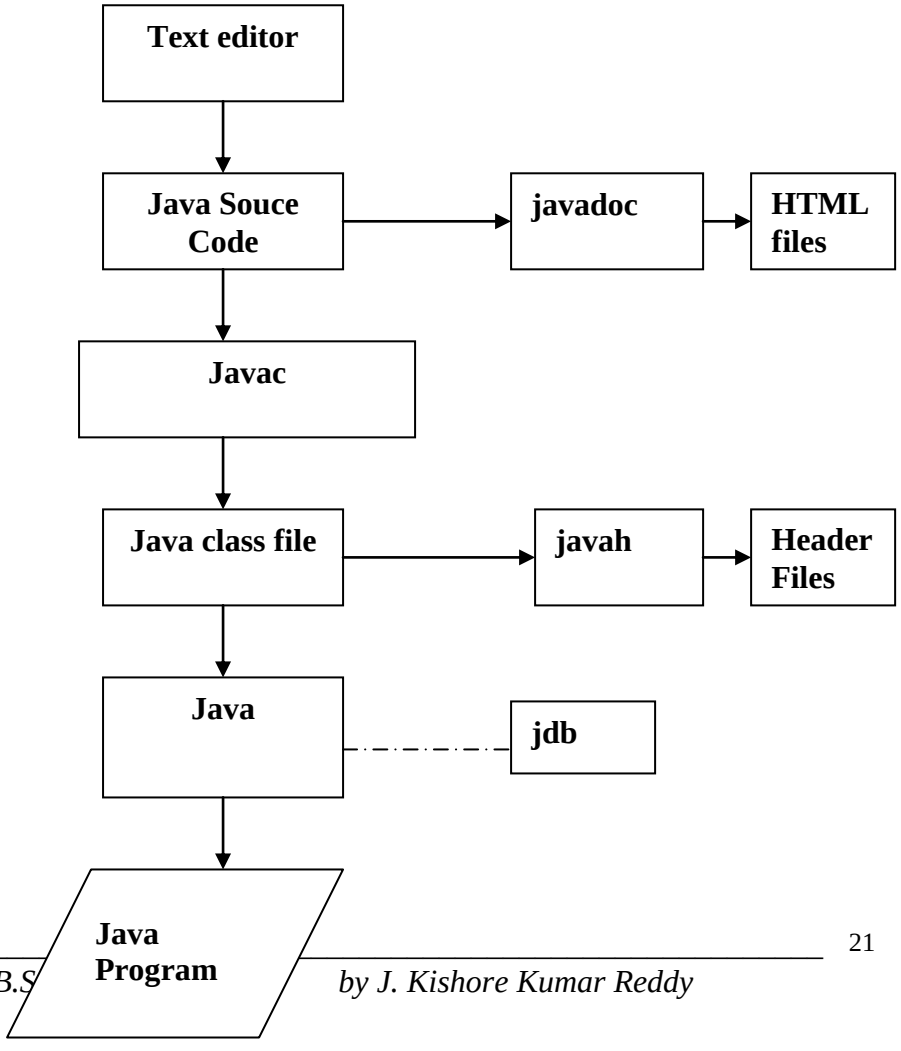
appletviewer : Enables us to run *Java applets*.

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 other computers via Internet.
- Applet Package: This includes a set of classes that allows to create Java programs.

Process of building and running Java application programs



UNIT-I: CHAPTER – III

Introduction:

Java is a general purpose, Object-oriented programming language. We can develop two types of JAVA programs.

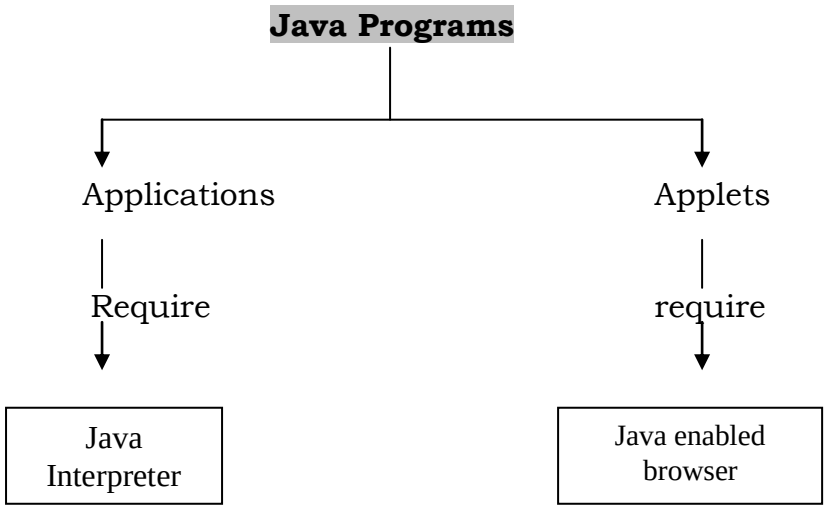
1. Applications:

These are stand-alone programs, which we can execute from the command prompt.

2. Applets:

These are programs that run in web-browsers.

The following figure shows the distinction between applications and applets.

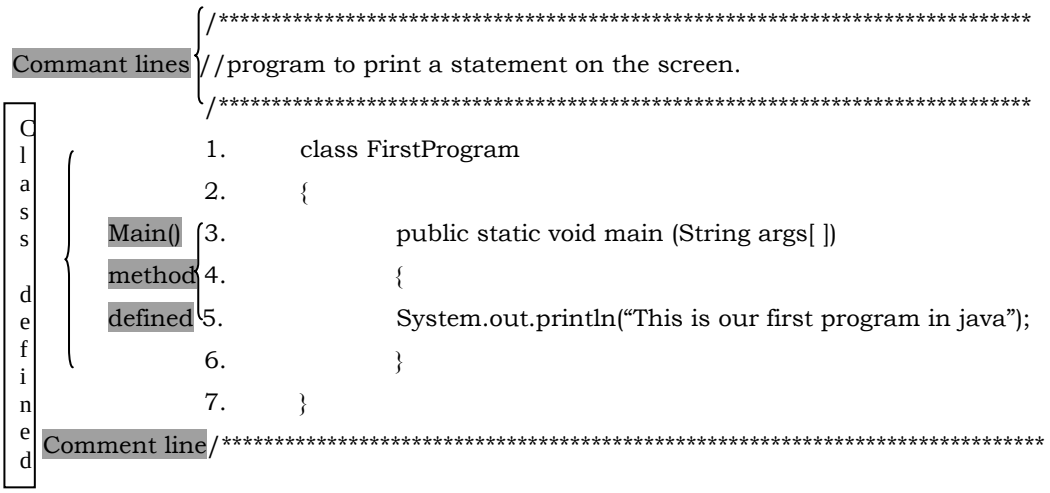


Applets are small Java programs developed for Internet applications. An applet located on a distant computer (Server) can be downloaded via Internet and executed on a local computer(Client) using a Java-capable browser.

Stand-alone programs can read and write files and program certain operations that applets cannot do. An applet can only run within a Web browser.

SIMPLE JAVA PROGRAM:

The following is a simple program that prints a line of text as output.



Comments in Java:

Comments are actually non-executable statements which are meant for the programmers own convenience. A comment is usually an explanation regarding the statement, comment does not effect the output by any means.

The Java programming language supports three kinds of comments.

1. / * text * /
2. // text
3. / ** text * /

Text specified as comments will be ignored by the compiler. Comments can appear any where in a program.

Example: 1)//int i=5;

2) /*int i=5;*/

3) //* int i=5;*/

Class Declaration:

A class being the smallest unit of a Java program, every program is bound to use at least one class. To use a class in a program, it has to be first declared.

Line 1: This line is the class declarator (or class header): It contains the keyword class along with the name of the class, which is *FirstProgram. A class is the smallest unit in Java.* Each and every line

of code in Java is written inside a class. The class declarator is stated as under.

Note: Since Java is case-sensitive, it distinguishes between upper and lower cases. Care must be taken while specifying class names. Java treats 'FirstProgram' as different from 'Firstprogram'.

Line 2 & 7: The definition of the class FirstProgram starts from Line 2, which contains the opening brace({) and ends on Line 7, which has the closing brace (}). Braces mark the beginning and end of a class, method or any block of statement(s).

The main( ) Method:

A Java program can contain any number of classes. One of these should contain the main ( ) method. The name of this class must be specified at the DOS prompt when we run the program with the Java interpreter.

If more than one class contain the main( ) method, the name of any one of these classes may be specified at the DOS prompt when we run the program.

The program starts executing from this class since it will be considered as the controlling class. Thus, it is always the **main( )** method of the controlling class which is executed first and which class the other methods required to run the program.

The main( ) method is always public, static and void.

Public:

In Java, the keyword *public* has the same meaning as it has in C++. Classes, methods and variables can be declared public. A public class can be accessed from any other class. The main( ) method has to be declared public so that it can be accessed from anywhere. If the class containing the main( ) method has been declared public, the source code file must have the same name as this class. Otherwise the compiler will display an error stating that the file name must be the same as that of the public class.

Static:

A static method is one which is associated directly with a class. The static methods are also called class methods. The keyword static allows a method to be called without creating an object of that class. The main( ) is always declared static so that it can be called without first creating an object of the respective class.

Void:

A method declared as void does not return any value.

String args[ ]:

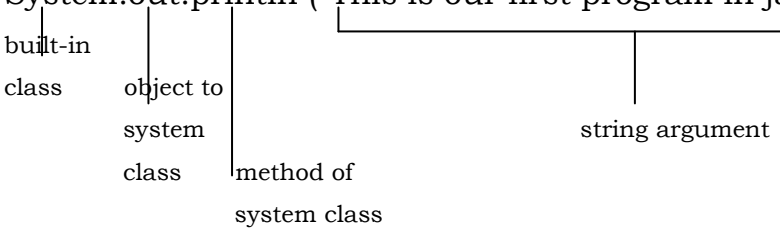
The main( ) method accepts a single argument, which is an array of type string. Word string in the parameter defines the type of the array, *args[ ] represents the name of the array and the blank subscript indicates that the length of the array* is not known the time of compilation. String is an in-built class in Java. Each string in the array represents one command line argument. However, this argument is optional.

The Output Statement:

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

This line contain single statement inside the main( ) method. When executed, it prints the string specified in the parentheses on the screen. Let us understand this statement in detail.

System.out.println ("This is our first program in java");



The various components of this statement are indicated above. The explanation of each of these elements:

System: This is a pre-defined class present in the Standard Java Library that comes along with the JDK. All variables and methods in the system class are static.

Out: It is an object of the system class.

println( ): This is a method of the **out** object. Its main function is to print the specified string on the screen and terminate the line. It takes one string argument.

Java Program Structure:

A Java program may contain many classes of which only one class defines a main method. Classes contain data members and methods that operate on the data members of the class. Methods may contain data type declarations and executable statements.

To write a Java program, we first define classes and then put them together. A Java program may contain one or more sections as shown in below figure.

Documentation Section	-------- suggested**Error!**
Package Statement	-------- optional
Import Statements	-------- optional
Interface Statements	-------- optional
Class Definitions	-------- optional

Main Method class
{
Main Method Definition
}
	-------- essential

General structure of a Java program

Documentation Section:

The documentation section comprises a set of comment lines giving the name of the program, the author and other details, which the programmer would like to refer to at a later stage. This would greatly help in maintaining the program. There are three styles of comments in Java.

1. / * text * / 2. // text
3. / ** text * /

Package Statement:

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

Example: Package student;

The package statement is optional.

Import Statements:

The next thing after a package statement (but before any class definitions) may be a number of import statements. This is similar to the # include statement in C.

Example: import student.test;

This statement instructs the interpreter to load the test class contained in the package student.

Interface Statements:

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

Class Definitions:

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

Main Method Class:

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

Java Tokens:

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

Keywords:

Keywords are an essential part of a language definition. They implement specific features of the language. Java language has reserved 60 words as keywords. These keywords, combined with operators and separators according to syntax, form definition of the Java language.

Since *keywords have specific meaning* in Java, we cannot use them as names for variables, classes, methods and so on. All *keywords are to be written in lower-case letters*. Since Java is case-sensitive. Some of the keywords are listed below.

Java keywords

Abstract	Boolean	break	byte	by value
Case	cast	catch	char	class
Const	continue	default	do	double
Else	extends	false	final	finally
Float	for	future	generic	goto

Identifiers:

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

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

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

  Example: `average  sum`

- When more than one word are used in a name, the second and subsequent words are marked with a leading uppercase letters.

  Example: `dayTemperature  firstdayofmothn  totalmarks`

- All private and local variables use only lowercase letters combined with underscores.

  Example: `length  batch_strength`

- All classes and interfaces start with a leading uppercase letter with a leading uppercase.

  Example:
  ```
  Student
  HelloJava
  Vehicle
  MotorCycle
  ```

- Variables that represent constant values use all uppercase letters and underscores between words.

  Example:
  ```
  TOTAL
  F_MAX
  PRINCIPAL_AMOUNT
  ```

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 specifie3s five major types of literals. They are:

- Integer literals
- Floating_point literals

- ❖ Character literals
- ❖ String literals
- ❖ Boolean literals

Operators:

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

Separators:

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

Java Separators

Name	What it is used for
Parentheses **( )**	Used to enclose parameters in method definition and invocation also used for defining 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 a 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.

Java Statements:

A statement is an executable combination of tokens ending with a semicolon (**;**) mark. Statements are usually executed in sequence in the order in which they appear. However, it is possible to control the flow of execution, if necessary using special statements. Java implements several types of statements.

1) Expression statement.
2) Label statement.
3) Synchronization statement.
4) Guarding statement.
5) Selection statement.
6) Iteration statement.
7) Jump statement.

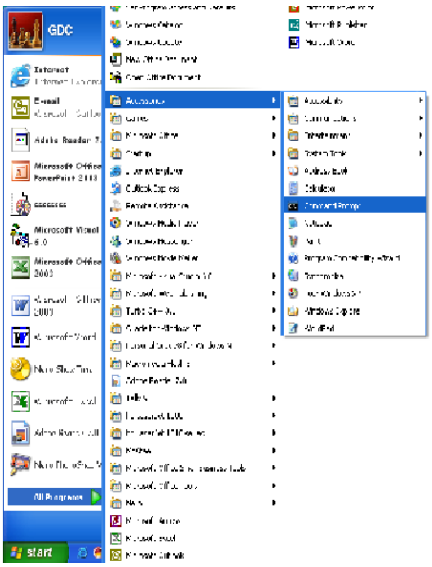
The classification of Java statements is shown the following figure:

Implementing a Java program:

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

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

I) Creating a program:



1) Click on the start button (START) on the Taskbar. A pus-up menu appears.
2) From the push-up menu, move the mouse-pointer over the programs option. A sub-menu appears.
3) From the sub-menu, click on MS-DOS prompt option. The DOS prompt appears on the screen.

4) Next, we will create a new folder with the name, MyPrograms. For this, type the following command at the DOS prompt:

```
C:\>md MyPrograms
```

Press enter key. The dos prompt appears as follows:

```
C:\>
```

MS-DOS Prompt

C:\>

5) To make the MyPrograms folder the current folder, type the following command at the DOS prompt:

```
C:\>cd MyPrograms
```

Press the enter key.

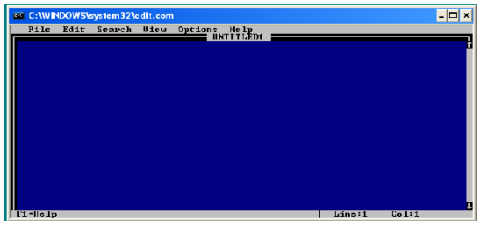
C:\>md MyPrograms----4
C:\>cd MyPrograms-----5
C:\MyPrograms>edit ---6

Now the current folder becomes MyPrograms. So we will store all our Java program files in this folder. Now open the edit window of MS-DOS by under taking the following steps.

6) Type the following command at the DOS prompt.

```
C:\Myprograms>edit
```

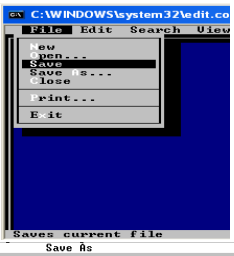
7) Press the enter key, the EDIT window appears on the screen.



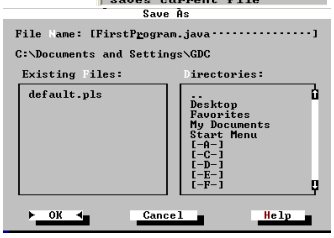
8) Let us now write our first program, which displays the following statement on the screen as the output.

```
/**************************************************************************
//program to print a statement on the screen.
/**************************************************************************
        class FirstProgram
        {
                public static void main (String args[ ])
                {
                System.out.println("This is our first program in java");
                }
        }
/**************************************************************************
```

9) Save the program by clicking on the save option in the file menu of the edit window. A save as dialog box appears on the screen.



10) Type the name FirstProgram.java against the file name option.



11) Click on OK button the program gets saved under the file name FirstProgram.java.

NOTE: The file must be saved with the . java extension.

II) Compiling the program

Now return to the DOS prompt and compile the program before running it. For this we follow the following steps:

1) Click on file menu and select the exit option. Immediately the edit window closes and DOS prompt appears on the screen.

2) Now compile the program by typing the following command at the DOS prompt.

```
C:\MyPrograms>javac FirstProgram.java
```

Press the enter key. The program gets compile.

III) Running the program:

Now run the program for this type the following command at the DOS prompt.

```
C:\MyPrograms>java FirstProgram
```

Press the enter key. The output of the program gets displayed as follows:

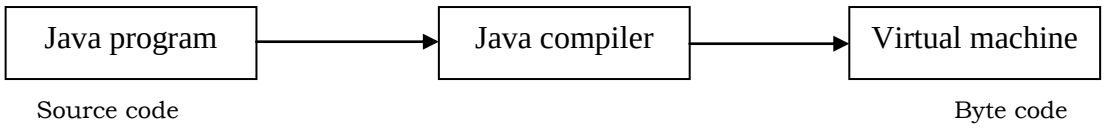
```
This is our first program in java
```

The following figure shows the output will be displayed on the screen

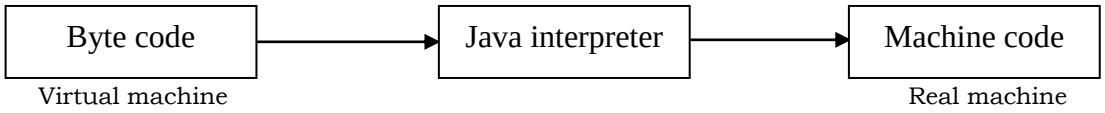
```
C:\>md MyPrograms----4
C:\>cd MyPrograms-----5
C:\MyPrograms>edit ---6
C:\MyPrograms>javac FirstProgram.java ------ compile
C:\ MyPrograms>java FirstProgram ------ run
This is our first program in java ----- output
```

Java Virtual Machine (JVM):

JAVA is both compiled and interpreted. That is, a Java source code (program) is first compiled and then interpreted. This is made possible by the Java Virtual Machine (JVM). The Java Virtual Machine, converts the intermediate byte code into the machine code.



PROCESS OF COMPILATION



PROCESS OF CONVERTING BYTE CODE

A Java source code as shown in above figure is first compiled to produce the byte code, which is then interpreted using the Java interpreter to produce the machine code.

Thus, a *Java source code goes through two processes; compilation and interpretation*. The output of the compilation process is the byte code, which is machine-independent. This makes a Java program capable of being executed on any machine, provided that Java interpreter, that is, Java Virtual Machine is present (installed) on that machine. The Java Virtual Machine (Java interpreter) is not the same for all machines. It depends on the hardware architecture and the operating system of a machine. *The Java interpreter is named java and the Java compiler is called javac.*

Command Line Arguments:

Command line arguments are parameters that are supplied to the application program at the time of invoking it for execution. It may be recalled that program was invoked for execution at the command line as follows:

```
java Test
```

We can write java programs that can receive and use the arguments provided in the command line.

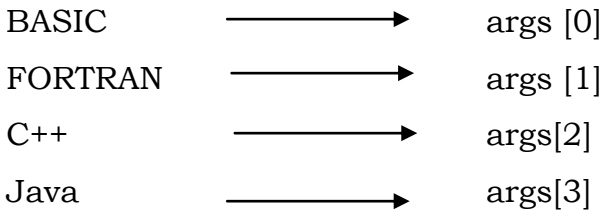
```
Public static void main (String args[ ] )
```

Here **args** is declared as array of strings (known as string objects). Any arguments provided in the command line (at the time of execution) are passed to the array **args** as its elements.

For example, consider the command line

java Test BASIC FORTRAN C++ Java

This command line contents four arguments. These are assigned to the array **args** as follows:



The individual elements of an array are accessed by using an index or subscript like **args [i].** The value of i denotes the position of the elements inside the array. For example, **args [2]** denotes the third element and represents C++.

Note: That java subscripts begin with 0 and not 1.

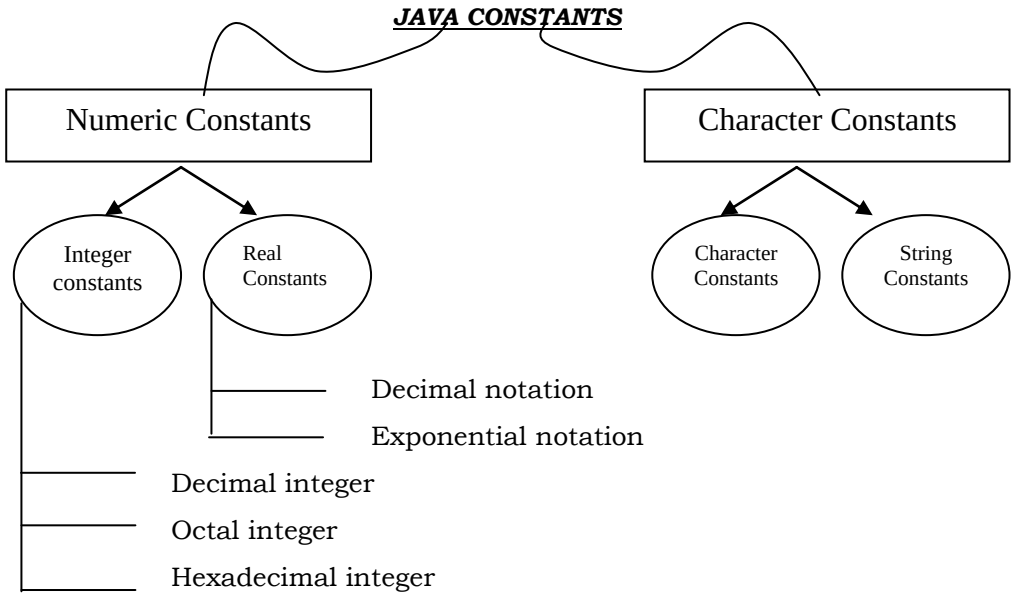
UNIT-I :: CHPTER-IV

Introduction:

A programming language is designed to process certain kinds of data consisting of numbers, characters and strings and to provide useful output known as information.

Constants:

Constants *refer to fixed values that do not change during the execution of a program*. Java supports several types of constants as:



Integer Constants:

An *integer constant refers to a sequence of digits*. There are three types of integers, namely:

- Decimal integer
- Octal integer and
- Hexadecimal integer.

Decimal integer:

Decimal integers consist of a set of digits, 0 through 9, preceded by an optional minus sign.

Valid examples of decimal integer constants are:

123 -321 0 654321

Embedded spaces, commas, and non-digit characters are *not permitted* between digits.

For example (invalid):

15 750 20,000 $1000 are illegal numbers.

Octal integer:

An *octal integer* constant consists of any combination of digits from the *set 0 through 7*, with a leading 0.

some examples of octal integer are:

037 0 0435 0551

Hexadecimal integer:

A sequence of digits *preceded by 0x or 0X i*s considered as hexadecimal integer. They may also include alphabets A through F or a through f. A letter *A through F represents the numbers 10 through 15.* Following are the examples of valid hex integers.

0x2 0x9F 0xbcd 0x

Real Constants:

Integer numbers are inadequate to represent quantities that vary continuously, such as distances, heights, temperatures, prices, and so on. These quantities are represented by numbers containing *fractional parts* like 17.548. Such numbers are called real constants.

A real number may also be expressed in *exponential notation*. For example, the value 215.65 may be written as 2.1565e2 in exponential notation. e2 means multiply by 10^{2}..

Mantissa e exponent

The mantissa is either a real number expressed in decimal notation or an integer. The exponent is an integer with an optional plus or minus sign. The letter e separating the mantissa and the exponent can be written in either lowercase or uppercase.

Examples of legal floating point constants are:

0.6e4 12e.2 1.5e+5 3.18E3 -1.2E-1

Embedded white space is not allowed, in any numeric constant.

Single Character Constants:

A single character constant (or simply character constant) *contains a single character enclosed within a pair of single quote marks.*

Examples of character constants are*:*

'5' 'X' ';' ' '

String Constants:

A *string constant is a sequence of characters enclosed between double quotes.* The characters may be alphabets, digits, special characters and blank spaces.

Examples are:

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

Backslash Character Constants:

Java supports some special backslash character constants that are used in output methods. For example, the symbol '\n' stands for new line character. A list of such backslash character constants is given in below table.

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 constants that remain unchanged during the execution of a program, a variable may take different values at different times during the execution of the program.

Some examples of variable names are:

- Average
- Height

- ❖ Total_height
- ❖ classStrength

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.

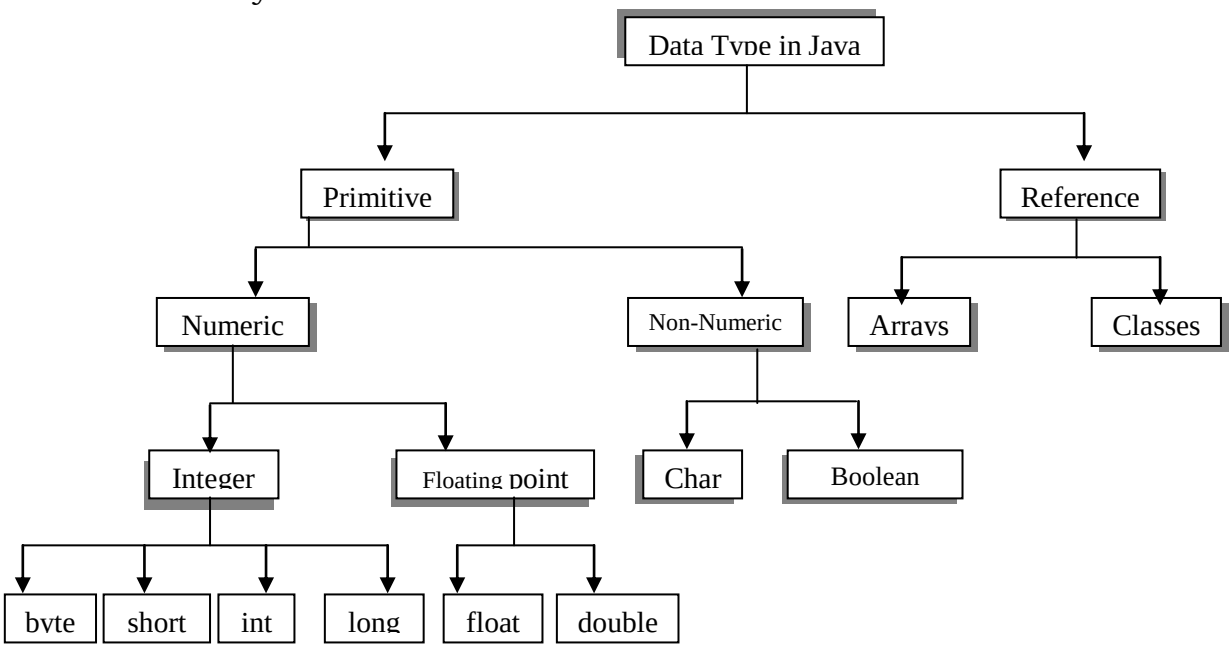
Data Types:

Every variable in Java has a data type. Data types specify the size and type of values that can be stored. Java language is rich in its data types.

Basically the data types in Java can be classified into the following two groups:

1. **Primitive or built-in data types**
2. **Reference or user-defined data types**

The following figure shows the data types offered by Java and the way in which they are classified.



Primitive (built-in) Data Types:

These form the basic data types. They are defined in the Java language. Examples of primitive data types are – int, char, byte, float, Boolean etc., i.e.

int. primitive data types are further classified into two groups namely:

a) Numeric
b) Non-Numeric

Numeric: Numeric data types fall under two categories namely:

i. Integer type
ii. Floating-point type

i) Integer type: This type can store whole numbers such as 0, 1, 2, 3, - 430, 567867, etc. It cannot store decimal numbers. Integer type can take negative as well as positive numbers. There are four integer types in Java:

- byte
- short
- int
- long

These differ in storage capacity. The following table shows the number of bytes occupied by the various integer data types and also the maximum and minimum values which they can assume.

Data type	Size (in bytes)	Minimum Value	Maximum Value
Byte	1	- 128	127
Short	2	- 32768	32767
Int	4	- 2147483648	2147483647
Long	8	-9223372036854775808	9223372036854775807

The *byte data type* is used to store values upto 1 byte. The short data type can store values upto 2 bytes, while the *int data type* can store values up to 4 bytes.

The *long data type* is used for assigning large values to the variables, which are beyond the range of int data type.

ii) Floating-point type:

Floating-point constitutes numbers which have a decimal part. Java supports the following two floating-point types.

- `float`
- `double`

The differences between these two types lie in their storage capacities. This is clarified in below table.

Datatype	Size (in bytes)	Minimum Value	Maximum Value
Float	4	3.4e-038	3.4e+038
Double	8	1.7e-308	1.7e+308

Non-Numeric type:

The second type of primitive data type is the non-numeric type. These are classified into two groups:

i. `char type`

ii. `Boolean type`

i) char type:

The variable declared of this data type is used for storing a character. It has a storage capacity of 2 bytes. Suppose we have to declare a variable c of char type and store a character z in it, then the following statement may be used:

```
char c = 'z';
```

ii) Boolean type:

The *Boolean* type can take either of the two values. *True* and *False.* Java does not allow the typecasting of a variable of Boolean type into any other data type

```
boolean d;
```

The variable of Boolean type has a storage capacity of 1 bit.

Reference (User-defined) Data Types:

The user as creates these data types and when demanded by the programming situation creates these data types. These may be arrays or classes. The rules for defining these data types are similar to those in C++ language.

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 decides the scope of the variable.

A variable must be declared before it is used in the program.

A variable can be used to store a value of any data type.

```
Type variable, variable2, . . . . . .. . , Variable n;
```

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:

The process of giving initial values to variables is known as the initialization.

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:

Variable Name = value;

For example:

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

We can also string assignment expressions 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;
```

Read Statement:

We may also give values to variables interactively through the keyboard using the readLine( ) method as illustrated in program:

Reading data from keyboard:

```
import java.io.DataImputStream;
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 = Integher.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.println("floatNumber = " + floatNumber);
    }
}
```

Scope of Variables:

Java variables are actually classified into three kinds:

- *Instance variables.*
- *Class variables*
- *Local variables.*

Instance and class variables are declared inside a class. Instance variables are created when the objects are instantiated and therefore they are associated with the objects. They take different values for each object.

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 }.

Symbolic constants:

We often use certain unique constants in a program. These constants may appear repeatedly in a number of places in the program. One example of such a constants in 3.142, representing the value of the mathematical constant "pi". We face two problems in the subsequent use of such programs. They are:

1. Problem in modification of the program.
2. Problem in understanding the program.

Modifiability:

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

Understandability:

When a numeric value appears in a program, its use is not always clear, especially when the same value means different things in different places.

For example, the number 50 may mean the number of students at on e place and the 'Pass marks' at another place of the same program. We may forget what a certain number meant, when we read the program some days later.

Assignment of a symbolic name to such constants frees us from these problems. For example, we may use the name strength to denote the number of students and PASS_MARK to denote the pass marks required in a subject. Constant values are assigned to these names at the beginning of the program. Subsequent use of the names STRENGTH and PASS_MARK in the program has the effect of causing their defined values to be automatically substituted at the appropriate points. A constant is declared as follows.

```
Final type symbolic – name = value;
```

Valid examples of constant declaration are:

```
final int   STRENGTH    =   100;
final int   PASS_MARK   =   50;
final float PI          =   3.14159;
```

Note that:

1. Symbolic names take the same form as variable names. But, they are written in CAPITALS to visually distinguish them from normal variable names. This is only a convention, not a rule.
2. After declaration of symbolic constants, they should not be assigned any other value with in the program by using an assignment statement. For example, STENGTH =200; is illegal.
3. Symbolic constants are declared for types. This is not done in C and C++ where symbolic constants are defined using the # define statement.
4. They can NOT be declared inside a method. They should be used only as class data members in the beginning of the class.

Type casting

During programming we may have to convert values of one data type to another. For example, we may need to convert int or flaot values to char type. Such conversions are necessary when there is a need to store the value of a variable of one type into a variable of another type.

Converting data of one type to another is called ***typecasting*****.** The general form of the statement for typecasting a variable is as follows:

```
Type variable = (type) variable2
```

Typecasting is done by putting the data type to which conversion is to be made in parentheses and placing it before the value to be converted and stored.

For example, to typecast the value of int data type into ***byte*** data type, the following statements are written:

Example:

```
1. int  j=4450;
2. byte d = (byte)j;
```

Typecasting can be divided into two groups:-

1. *Narrowing*
2. *Widening or promotion*

1. Narrowing:

The process of assigning a larger type to a smaller one is known as narrowing. Note that narrowing may result in loss of information.

2. Widening:

The process of assigning a smaller type to a larger one is known as widening or promotion.

In the above example the size of int data type is 4 bytes whereas that of byte data type is 1 byte. In Line 2 above, a variable whose data type is of larger bytes has been accommodated into another variable whose data type is of smaller bytes.

Therefore, narrowing is not always safe as it may result in the loss of data. Moreover, to make it worse, the compiler would not notify the loss of data to the user by means of any error message at all. The solution of the problem lies with a phenomenon referred to as widening. In sharp contract with narrowing, widening refers to the typecasting of a variable of datatype of smaller size into another type of larger size.

```
/*****************************************************/
//Program showing typecasting.
/*****************************************************/
class Typcst
{    public void main (String args[])
     {
     double d1, d2, d = 23456;
     short s1, s2, s = 56;
     long 11, 12, 1 = 198765;
     d1 = (double)s;
     d2 = (double)1;
     s1 = (short)d;
     s2 = (short)1;
     11 = (long)d;
```

```
    12 = (long)s;
    System.out.println("The value of d1="+d1);
    System.out.println("The value of d2="+d2+"\n");
    System.out.println("The value of s1="+s1);
    System.out.println("The value of s2="+s2+"\n");
    System.out.println("The value of 11="+11);
    System.out.println(:The value of 12="+12+"\n);
    })
/*****************************************************/
```

Save the program with the name typcst.java and exit to the DOS prompt. Compile the program by typing the following command at the DOS prompt:

```
javac Typcst.java
```

On successful compilation, run the program using the following statement:

```
Java  Typcst
```

The output gets displayed as shown below:

The value of d1 = 56.0

The value of d2 = 198765.0

The value of s1 = 23456

The value of s2 = 2157

The value of 11 = 23456

The value of 12 = 56

Getting values of variables:

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 the next line

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

```
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.

```
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. This means that the statements.

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

Will produce the following output:

```
Hello
Java!
```

The statement

```
System.out.println( );
```

Will print a blank line.

Program: Getting the result to the screen

```
Class Displaying
{
public static void main(String args[])
{
    system.out.println("Screen Display");
    for (int I = 1; I <= 9; 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
5 5 5 5 5
6 6 6 6 6 6
7 7 7 7 7 7 7
8 8 8 8 8 8 8 8
9 9 9 9 9 9 9 9 9
```