CHAPTER - 1

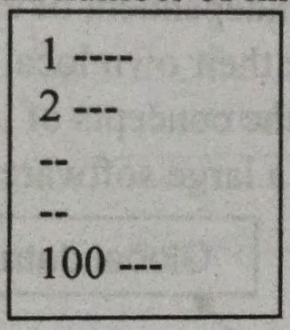
OBJECT ORIENTED PROGRAMMING

1.1 Introduction:

Programming Languages are classified into the following categories

a) Monolytic Programming Languages
b) Procedural Programming Languages
c) Structured Programming Languages
d) Object Oriented Programming Languages

a) Monolytic Programming Languages: Monolytic Programming Languages contain number of lines of data.



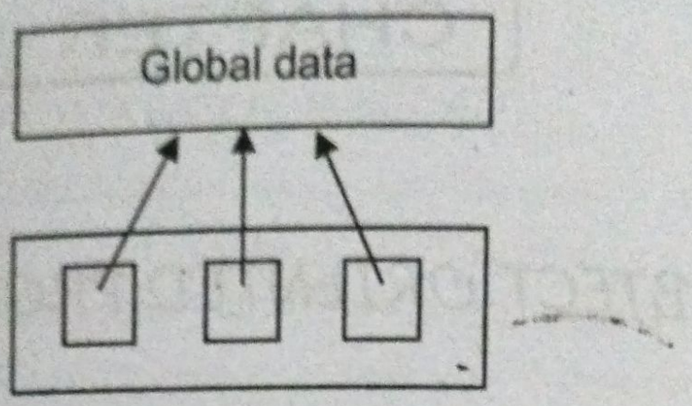
Drawback: 1. It is not subroutine concept.
2. The program code is duplicated each time it is to be used.

Examples: Assembly Language, Basic

b) Procedural Programming Languages: The important features of Procedural Programming Languages are

i) Emphasis on algorithm rather than data
ii) Programs are organized in the form of subroutine
iii) Program controls are through jumps (goto) and calls to subroutine
iv) Data move overly around the system from function to function
v) It follows top down approach in program design

vi) It is suitable for medium size software applications

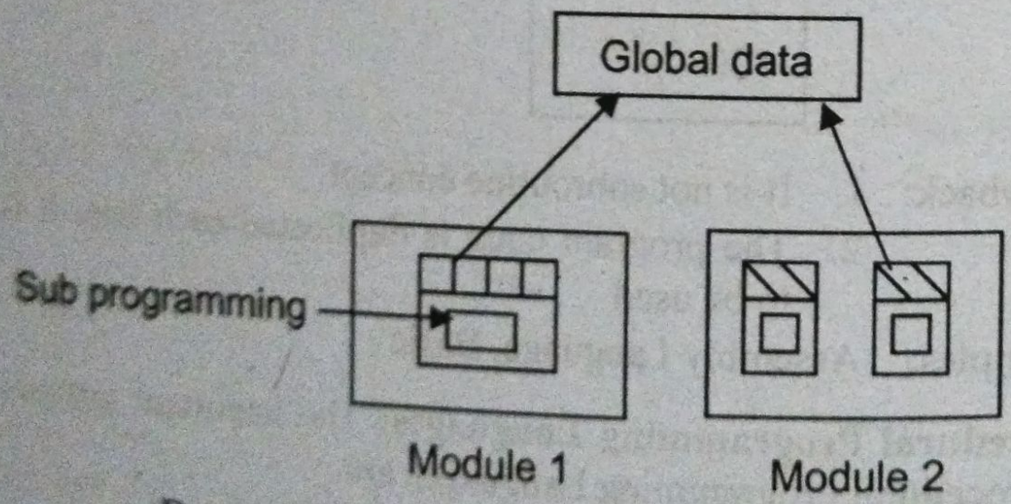


Drawback: 1. Data Security problem
2. Difficult to maintain program code
3. It doesn't model the real world problem very well

Examples: Fortran, Cobol

c) **Structured Programming Languages:** In Structured Programming Languages programs consist of multiple modules and each module has a set of functions of related types. The important features of Structured Programming Languages are

i) Programs are divided into individual procedures that performs discrete tasks
ii) Procedures are independent of each other
iii) Procedures have their own local data and processing logic
iv) Introduction of the concepts of user defined datatypes
v) Maintenance of a large software system is costly



Drawback: 1. Data Security problem
2. It doesn't model the real world problem very well

Examples: Pascal, C

d) **Object Oriented Programming Languages:** The important features of Object Oriented Programming Languages are

ii) Programs are divided into objects

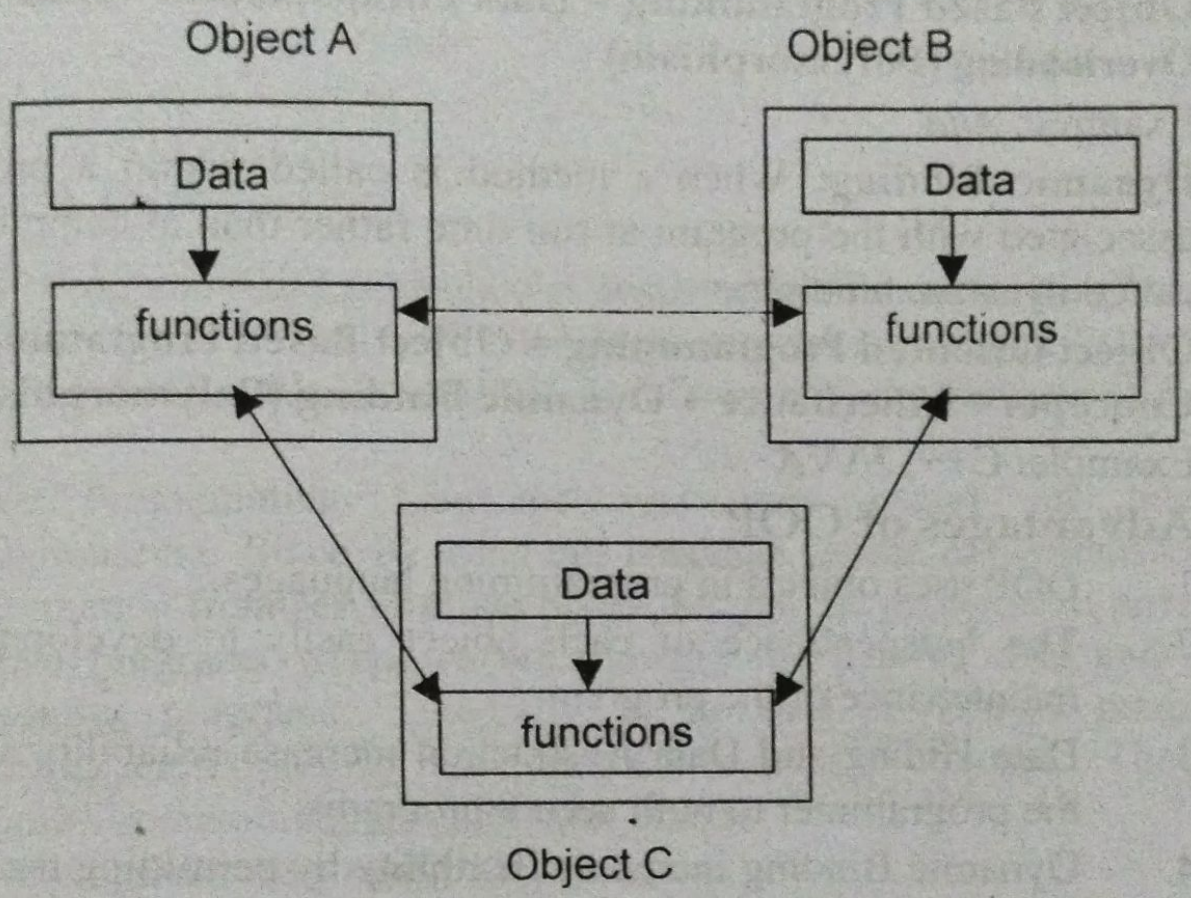
iii) Functions that operate on the data of an object are together in the data structure

iv) Data is hidden and cannot be accessed by external functions (Data security)

v) Objects may communicate with each other through functions

vi) New data can be easily added to the functions whenever necessary

vii) It follows bottom-up approach in program designing



1.2 OOP Concepts

The important concepts of Object Oriented Programming are

a) Encapsulation
b) Abstraction
c) Inheritance
d) Polymorphism

a) ***Encapsulation:*** The mechanism by which the data and functions are bound together with an object definition. By means of Encapsulation, it is possible to protect the data.

b) ***Abstraction:*** It is the process of defining data type, often called as an abstract data type, together with the principle of data hiding.

Abstract Data type refers to the program defined data type together with a set of operations than can be performed on the data. Data hiding is a property where by the internal data structure

of an object is hidden from the rest of the program. The data can be accessed only by the function declared with in the class.

c) ***Inheritance:*** Inheritance is the mechanism that allows the programmer to derive new classes from existing classes. The derived classes inherit the methods and data of the parent class.

d) ***Polymorphism:*** Polymorphism is a feature that allows giving a single name to different methods with different parameters. Depending on the parameters only one particular method is executed.

1.2.1 Object Based Programming = Data Encapsulation + Data Hiding + Overloading (Polymorphism)

Example: ***Ada***

Dynamic Binding: When a method is called within a program, it associated with the program at run time rather than at compile time is called dynamic binding.

1.2.2 Object Oriented Programming = Object Based Programming Concepts + Inheritance + Dynamic Binding (Polymorphism)

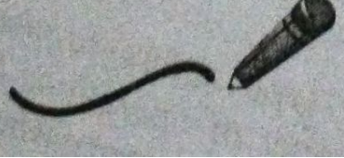
Example: C++, JAVA

1.3 Advantages of OOP

1. OOP uses objects in programming languages.
2. The independence of each object easily to development and maintenance of the program
3. Data Hiding and Data Abstraction increase reliability and helps the programmer to built secure programs.
4. Dynamic Binding increases flexibility by permitting the addition of new class objects without having to modify the existing code.
5. Inheritance with Dynamic Binding enhances the reusability of code.

1.4 Applications of OOP

1. Simulation and Modeling
2. User Interface Design
3. Artificial Intelligence and Expert Systems
4. Neural Networks
5. Web Designing
6. CAD/CAM System



CHAPTER - 2

INTRODUCTION TO JAVA PROGRAMMING

2.1. JAVA Introduction

Java Programming Language was designed by "James Gosling" at Sun Microsystems in 1991. This language was initially called "***Oak***" but was renamed as "Java" in 1995. It follows both the features of C and C++.

Java Programming Language was very useful on Internet Programming. Since, by using this language we can easily transfer the information from server to the nodes through the network. It provides active programs when we are viewing the passive data and self-executing programs. Generally, in the case of network programs Security and Portability is the main active areas of concern. The Java Applet Programming provides such things.

Java programs are classified into 2 types.

1) Application Programs
2) Applet Programs

An **Application** is a program run on the computer under the operating system of the computer. This is similar to the other programming languages.

An **Applet** is an application designed to be transmitted over the Internet and executed by a Java-compatible Web browser.

2.2. Java Features/Buzz Words

a) ***Platform Independent:*** Java is a platform independent language. Once we create the program in one operating system, that program will be worked on any other operating system that is transmitted to the operating system.

b) ***Portable:*** The most significant contribution of Java over other languages is its portability. Java programs can be easily moved from one computer system to another anywhere and any time.

Changes and upgrades in operating systems, processors, system resources will not forces any changes in Java programming.

c) **Object oriented:** Java is a true object oriented language. Why because without class we can't run the program. Without main we can run the program. Almost everything in Java is an object. All program code & data reside with in objects and classes. The object model in Java is simple and easy.

d) ***Robust & Secure*:** Java is a Robust Language. It provides many safe guards to ensure reliable code. It is designed as garbage collected language relieving the programmers, virtually all memory management problems. Java also incorporates the concept of exception handling, which captures serious errors and eliminates any risk on crashing the system.
Security becomes an important issue for a language that is used for programming on Internet. Java systems not only verify all memory access but also ensure that no viruses are communicated with the Applet.

e) ***Distributed*:** Java is designed as a distributed language for creating applications on network. It has the ability to share both data and program. Java applications can open remote object on Internet as easily as they can do in local system.

f) **Simple & Small:** Java is a simple & small language. Many features of C and C++ that are sources are unreliable code are not part of Java. For example, Java doesn't use Pointers, Preprocessors, Header files, goto statements and many others.

g) ***Multi Threaded*:** Java was designed to meet the real-world requirement of creating interactive, networked programs. Java supports multithreaded programming, which allows to handling multi tasks simultaneously. This means we need not wait for the application to finish one task before beginning another one.

h) ***Dynamic*:** Java is a dynamic language. Java is capable on dynamically linking in new class, libraries, methods and objects. Java support functions written in other languages such as C and C++. These functions are known as native methods. Native methods are linked dynamically at run time.

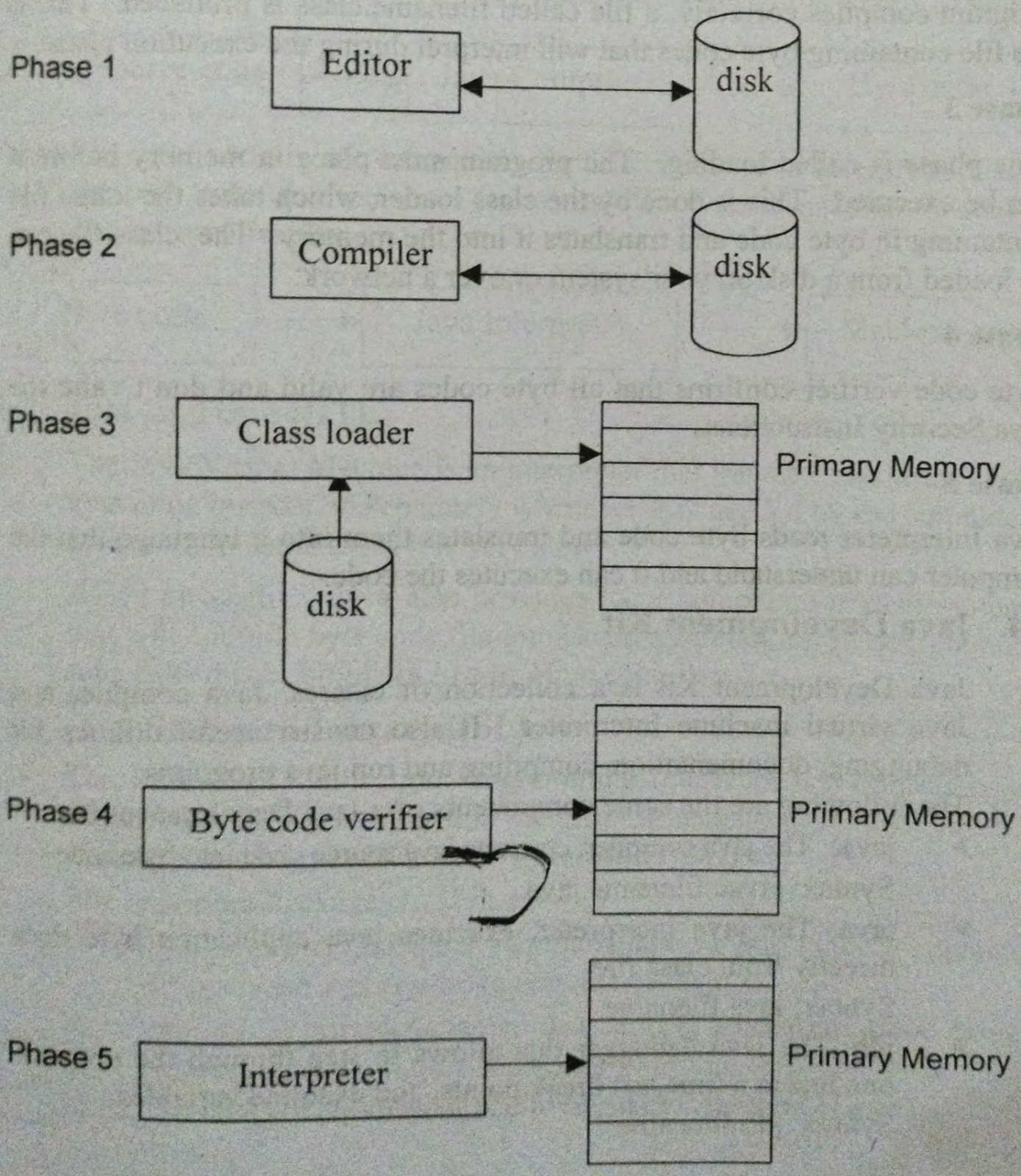
i) ***Compile & Interpreted*:** Usually a Computer language is either compiled or interpreted. Java combined both these approaches thus making. Java is a two-stage system. First Java Compiler translates source code into byte code instruction and therefore in the second stage Java Interpreter generates machine code that can be directly executed by the machine that is running the Java program.

Interpreter is called JVM (Java Virtual Machine). Java applications are platform independent. JVM, JDK are platform dependent. Interpreter is different for all operating systems like DOS, UNIX etc.,

The main drawback of C is the program cannot run in any other operating system. It is advantage of Java.

j) ***High performance***: Java performance is impressing for an interpreted language due to the use of intermediate code. Java architecture is also design to reduce overheads during runtime. The incorporation of multithreading enhances, the overall executions, speed of Java programs.

2.3. Basics of JAVA Environment:



Phase 1

Java program is typed in a text editor (Notepad, Edit plus) and makes corrections if necessary. The programmer specifies that the file in the editor should be save. The program is stored on a second storage device such as a disk. Java program files are stored with .java extension filenames.

Save: c:\jdk1.2.1\bin\filename.java

Phase 2

The programmer gives the command javac filename.java to compile the program. At this stage, the Java compiler translates the Java program into byte code that is the language understood by the Java Interpreter. If the program compiles correctly, a file called filename.class is produced. This is the file containing byte codes that will interpret during the execution phase.

Phase 3

This phase is called loading. The program must place in memory before it can be executed. This is done by the class loader, which takes the .class file containing in byte code and translates it into the memory. The .class file can be loaded from a disk on your system or over a network.

Phase 4

Byte code verifier confirms that all byte codes are valid and don't valid the Java Security Instructions.

Phase 5

Java Interpreter reads byte code and translates them into a language that the computer can understand and it can executes the code.

2.4. Java Development Kit

Java Development Kit is a collection of classes, Java compiler and Java virtual machine Interpreter. It also consist useful utilities for debugging, documentation, compiling and run java programs.

The following are the some components of a Java Development Kit.

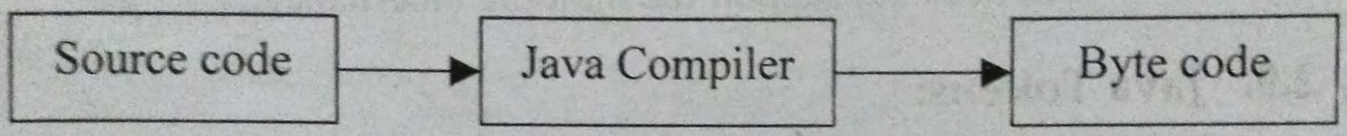
- javac: The java compiler, converts java source code into byte code
  Syntax: javac filename.java
- java: The java interpreter, executes java application byte does directly from class file
  Syntax: java filename
- jdb: The java debugger that allows to step through the program one line at a time, set break points, and examine variables
  Syntax: jdb filename

- servletrunner: A simple web server to test servlets
- appletviewer: A java interpreter that executes java applet classes hosted by HTML

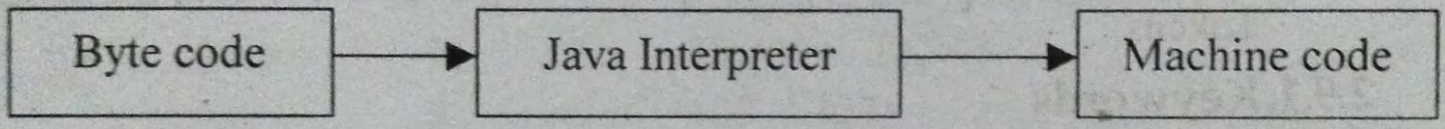
  Syntax: appletviewer filename.html

2.5. Java Virtual Machine (JVM):

Java provides both compiler and a software machine called JVM for each computer machine. The Java Compiler translates the Java Source Code (Java program) into an intermediate code known as byte code, which executes on special type of machine. This machine is called Java Virtual Machine and exists only inside the computer memory. The byte code is machine independent code and can run in any system.



The byte code is not a machine specific code. Java Interpreter takes the byte code and translates it into its own system machine language and runs the results.



2.6. Just In Time (JIT):

The Java Virtual Machine is an interpreter that translates and runs each byte code instruction separately whenever it is needed by the computer program. In some cases it is very slow.

As an alternative, Java also provides local compiler for each system that will compile byte code file into executable code for faster running. Java calls these compilers Just In Time compilers.

2.7. Java Standard Library (JSL):

The Java API (Application Programming Interface) is a collection of classes and methods grouped in the form of packages. JDK 1.2 has 58 packages, it is also known as Java Standard Library.

The important Packages in Java are

a) ***java.lang***: It contains the main language support class. It contains wrappers, strings and basic features of Java

b) ***java.util***: It provides classes that support date, time, basic event processing etc.,

c) ***java.io***: It provides reading and writing data in the form of streams.

d) java.awt: It provides classes for creating GUI programs.
e) java.applet: It includes set of classes to support for applet programming.
f) javax.swing: It provides classes for swing operation programming.

2.8. Differences between C++ and Java

1. Java does not include structure and union.
2. It is not possible to declare unsigned integer in Java.
3. Pointers don't exist in Java.
4. Java does not have a preprocessor.
5. There are no header files and delete operators.
6. Java does not allow goto, sizeof and typedef.
7. Java does not support the multiple inheritance.

2.9. Java Tokens:

Tokens are smallest individual units in a program. The compiler reads the source program one character at a time grouping the source program into a sequence of atomic units called Tokens. Reserved words, identifiers, constants, operators etc., are the examples of Java Tokens.

2.9.1. Keywords

The Keywords or Reserved words are one, which have a special and predefined meaning to the Java compiler. There are 48 reserved words currently used in Java. These words cannot be used as names for a variable, constant, class or method. Reserved words are

abstract	boolean	break	byte	case	catch
char	class	const	continue	default	do
double	else	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
true	false	null			

Note: const and goto are reserved for future use. Java reserved words true, false and null reserved words are used to represent the values.

2.9.2. *Datatype*

Datatypes are used to specify the type of data that a variable can attain. There are 2 types of datatypes. 1. Primitive Datatypes 2. Non-Primitive Datatypes

1. ***Primitive Datatype:*** It is also called as standard, intrinsic or built-in datatypes. There are eight primitive datatypes available in Java. Those are byte, short, int, long, char, float, double and boolean. These can be classified into four groups.

Integers	:	byte, short, int, long
Floating point numbers	:	float, double
Characters	:	char
Boolean	:	boolean

Type	Size		Range
	Bits	Bytes	
int	32	4	-2,147,483,648 to 2,147,483,647
long	64	8	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
short	16	2	-32,768 to 32,767
byte	8	1	-128 to 127
float	32	4	3.4 e –0.38 to 3.4 e +0.38
double	64	8	1.7 e –308 to 1.7 e +308
char	16	2	0 to 65535
boolean	8	1	true / false

Note:
1. Java does not support have any unsigned types
2. Variables of types char, byte, short, int, long, float and double are all given the value '0' by default
3. Variable of type boolean are given 'false' by default
4. All primitive datatypes in Java are portable (Platform independent)

2. ***Non-Primitive Datatype***: These are also known as Derived Datatype or Abstract Datatype or Reference type. These are built on primitive data types.

   Examples: string, class, array, interface etc.,

2.9.3. Variable:

The name given to the various elements to store the information is called an identifier or variable. The rules for variables are as follows:

- Variable must be begin with a letter, a dollar ($) symbol or an underscore (_) followed by a sequence of letters
- Variable names must be unique
- There should be no space in between any two characters of an identifiers
- Keywords cannot be used for variable name
- Java is a case-sensitive, so upper and lower case letters are distinct

Declaring a Variable: All variables must be declared before they can be used. The basic form of a variable declaration is

Syntax: type identifier;

Where type is represents datatype of the variable, identifier is the name of the variable.

Example: int a;

More than one variable of the same type can be declared by using a comma-separated list.

Syntax: type identifier1, identifier2, ..., identifiern;

Example: float x, y, z;

Initializing a Variable: Variable can also be initialized with equal sign and a value.

Syntax: type identifier = value;

Example: int a = 3;

2.10 Dynamic Initialization:

Java allows variables to be initialized dynamically, using any expression valid at a time the variable is declared.

Example:class dyna

```
{
        public static void main(String args[])
        {
                double a=3.0, b=4.0;
                double c=Math.sqrt(a*a+b*b);
                System.out.println("Hypotenuse is" + c);
        }
```

2.11 Scope & Life-time of a Variable:

An identifier duration also called lifetime is the period during which that identifier exist in memory. Identifier that represent local variables in a method i.e., parameters and variables declared in the method body have automatic duration. Automatic declaration are created when program control reaches their declaration they exist while the block in which they are declare is active and they are destroyed when the block in which they are declared is exceeded. We will refer to variables of automatic duration has automatic variables or local variables.

Java also has identifiers of static duration. Variables and references of static duration exist from the point at which the class that defines them is loaded into memory for execution until the program termination.

Identifier scope is where the identifier can be a reference in a program. The scope of an identifier is class scope and block scope.

Methods and instance variable of a class have class scope. Class scope begins at the beginning left brace of the class definition and ends at the closing right brace of the class definition.

Identifiers declared inside a block have block scope. Block scope begins at the identifiers declaration and ends at the terminating right brace of the block. Local variables of the method have block scope.

2.12 Class:

A class is a non-primitive datatype that contains member variables (data) and member functions (methods) that operates on the variables. A class is declared by the use of the class keyword. The general form of class definition is as follows.

Syntax:
```
class class-name
{
        datatype instance-variable1;
        datatype instance-variable2;
        -
        -
        datatype instance-variablen;
        datatype methodname1(parameter-list)
        {
        // method body
        }
```

```
-
-
datatype methodnamem(parameter-list)
{
// method body
}.
}
```

The data, or variables defined with in a class are called instance variables. The code is present with in methods. Collection of methods and variables defined with in a class are called members of the class.

Example:
```
class box
{
double width, height, depth;
double volume( )
{
return width*height*depth;
}
}
```

2.13 Object:

Object is an instance of the class. By means of the object only, we can access the functions (methods).

```
main( )
{
box a = new box( );
a.volume();
}
```

2.14 Comments in a Program:

Three types of comment statements are available in Java. They are

i) Single line comment (//)
ii) Multi line comment (/* and */)
iii) Documentation comment (/** and */)

1. ***Single line comment:*** For a single line comment we use double slash (//) to begin a comment. It ends at the end of line.

   Example: // Program for stack operations

2. ***Multi line comment:*** For more than one line of information, we use multi line comments. Multi line comments are starts with /* and ends with */.

   Example: /* Program for stack operations
   Designed by VIJAY BHASKAR
   MeRITS, Udayagiri */

3. ***Documentation comment:*** For the documentation purpose, we use documentation comments. Documentation comments are starts with /** and ends with */.

   Example: /** A test for JDK
   @ Author S.Prabhu
   @ version 1.0 */

2.15 Input and Output Statements:

java.lang package consists a class called 'System' which contains console I/O methods. If we want to take the information from the keyboard and display the information on the screen, Java uses Stream objects. The system provides three basic streams. Those are

a) System.in
b) System.out
c) System.err

System.in refers to the standard input stream, System.out refers to the standard output stream and System.err refers to the standard error stream.

System.out class consists of two methods to print the strings and values on the screen. They are 1. print( ) 2. println( ).

1. print( ): Syntax: System.out.print(list);

This statement prints the values of the variable name specified in the list of output unit.

Example: System.out.print("hai");
System.out.print("java");
O/P: haijava

2. println( ): Syntax: System.out.println(list);

This statement prints the values of the variable name in the list, the cursor is advanced by one line and the next statement is print in the next line.

Example: System.out.println("hai");
System.out.println("java");

O/P: hai
java

Note: Java has a version of the '+' operator for string concatenation that enables a string and a value of another datatype.

Example: sum = 50
"sum is" + sum

Here, sum is automatically converted to a string and concatenated with the "sum is" which results in the string sum is 50

Example: "y#2="+7
O/P: y#2=7

2.16 Command Line Arguments:

Java has the facility of command line arguments. Java main method consists array of string objects. When we run the program, the array will fill with the values of any arguments it was given in the command line.

Example: Write a program to read two values and print the values

```
class ex2
{
    public static void main(String[] args)
    {
        System.out.println(args[0]);
        System.out.println(args[1]);
    }
}
```

2.17 Building the Java Program:

Java is a text file that contains one or more class definitions. Java compiler requires that a source file use the .java file name extension.

Example:

```
class Example
{
    public static void main(String[] args)
    {
        System.out.println("Hello");
    }
}
```

- The keyword 'class' is used to declare the new class is being defined
- Example is an identifier that is the name of the class
- Entire class definition will be between the opening curly brace ({) and the closing curly brace (})
- 'public' keyword is an access specifier, which allows the programmer to control the visibility of class members
- The keyword 'static' allows main() to be called without having to instantiate a particular instance of the class. Since, main() method is called by the Java interpreter before any objects are made
- The keyword 'void' used to tell the compiler that main() does not return a value
- main() is a method called when a Java application begins
- String args[] declares a parameter named args, which is an array of instances of the class string. args receives any command line arguments present when the program is executed

Now the file is saved with .java extension . Let the file name is taken as Example.java

2.18 Compiling & Running the Java Program:

To compile Example.java program by the compiler javac, specify the name of the source file on the command line as

C:\> javac Example.java

Here, the Java compiler creates a file called Example.class that contain byte code version of the program. The class file is run by the Java interpreter, called java by passing the name to command line argument as

C:\> java Example

O/P: Hello

Primitive datatypes are to be converted into object type by using the wrapper classes. Integer class is used as wrapper class for primitive datatype int. parseInt( ) method converts string to an int.

Example: Write a program to read two integer values and print the sum

```
class ex3
{
    public static void main(String[] args)
```

```
{
int a,b,sum;
a=Integer.parseInt(args[0]);
b=Integer.parseInt(args[1]);
sum=a+b;
System.out.println("sum of a and b is"+sum);
}
}
```

2.19 *Operators*

Operator is a symbol that performs a specified operation between the data items. Java provides the following different types of operators.

a) Arithmetic operators:

Operator	Meaning
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus

Example:

```
class arope
{
        public static void main(String[] args)
        {
            int a,b,sum,sub,mul,div,mod;
            a=Integer.parseInt(args[0]);
            b=Integer.parseInt(args[1]);
            sum=a+b;
            sub=a-b;
            mul=a*b;
            div=a/b;
            mod=a%b;
            System.out.println("Addition="+sum);
```

```
        System.out.println("Subtraction="+sub);
        System.out.println("Multiplication="+mul);
        System.out.println("Division="+div);
        System.out.println("Remainder="+mod);
    }
}
```

b) Arithmetic Assignment Operators:

Syntax: variable operator = expression

Which is equivalent to

variable = variable operator expression

Operator	Example	Equivalent Expression
+ =	a + = 4	a = a + 4
- =	a - = 5	a = a – 5
* =	a * = 3	a = a * 3
/ =	a / = 8	a = a / 8
% =	a % = 5	a = a % 5

c) Assignment Operator: '=' is an assignment operator used to assign the value to a variable.

Syntax: variable = value;

Example: a = 7;

d) Increment and Decrement Operators: '++' and '--' are Java's increment and decrement operators. The increment operator increased its operand by one. The decrement operator decreased its operand by one.

Example:

```
class a
{
    public static void main(String[] args)
    {
    int x=4;
    System.out.println("x++:"+ ++x);   //O/P:. 5
    int y=10;
```

```
        System.out.println("x++:"+ y++);   //O/P: 5
        int p=5;
        int t=--p+3;
        System.out.println("t :"+t);  //O/P: 7
        }
}
```

c) ***Relational Operators:*** Relational Operators determines the relationship between the two operands.

Operator	Meaning	Example
= =	equal to	A= =B
!=	Not equal to	A!=B
>	Greater than	A>B
<	Less than	A<B
>=	Greater than or equal to	A>=B
<=	Less than or equal to	A<=B

d) ***Logical Operators:*** Logical operators are used to check for compound conditions, which are formed, by the combination of two or more relational expressions. Java provides the following logical operators and these are follows truth tables and produce Boolean values.

Operators: Logical AND &&

Logical OR ||

Logical NOT !

Truth table for Logical AND(&&):

Op1	Op2	Op1 && Op2
T	T	T
T	F	F
F	T	F
F	F	F

Truth table for Logical OR(||):

Op1	Op2	Op1 && Op2
T	T	T
T	F	T

F	T	T
F	F	F

Truth table for Logical NOT(!):

Op1	!Op1
T	F
F	T

g) ***Conditional Operator:*** Java has a decision operator called conditional operator. "? :" are conditional operators. These operators are also called ternary operators.

Syntax: testconditon ? expression1 : expression2

Here, if the testcondition is true, then expression1 is evaluated otherwise, expression2 is evaluated.

Example: max = (a>b) ? a : b;

h) ***Bit-wise Operators:*** The following bit-wise operators are provided by Java. These operators act upon the individual bits on their operands.

Operator	Meaning
~	Bit-wise unary NOT
&	Bit-wise AND
|	Bit-wise OR
^	Bit-wise exclusive OR
>>	Shift right
<<	Shift left
|=	Bit-wise OR Assignment
^=	Bit-wise exclusive OR Assignment
>>=	Shift right Assignment
<<=	Shift left Assignment

These operators act upon the individual bits on their operands. Bit-wise Logical Operators (&, |, ^ and ~) follows the following truth tables

Op1	Op2	Op1|Op2	Op1&Op2	Op1^Op2	~Op1
0	0	0	0	0	1
1	0	1	0	1	0
0	1	1	0	1	1
1	1	1	1	0	0

2.20 javax.swing:

It is a package that contains an important class called JOptionPane class. This class is used to display dialog boxes. It provides two important predefined methods showInputDialog and showMessageDialog called static methods. Such methods are always used their class name followed by a dot operator.

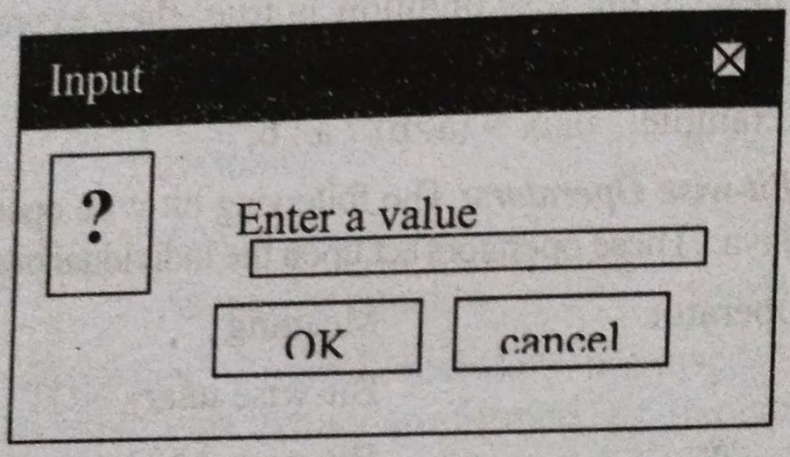
ShowInputDialog: showInputDialog is a method of class JOptionPane. It is a static method and the general format is as

Syntax: JOptionPane.showInputDialog(argument);

The argument indicates to the user what to do in the text field.

Example: JOptionPane.showInputDialog("Enter a value");

When we perform this operation the input dialog box is available as



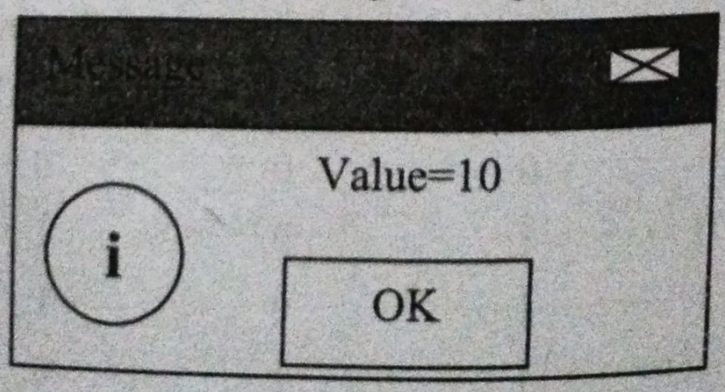
The users types characters in the text field then clicks the ok button, it returns string to the program.

showMessageDialog: showMessageDialog is method of class JOptionPane. It is a static method. The general format of showMessageDialog is in two ways.

Syntax1: JOptionPane.showMessageDialog(null,argument1);

The first argument must be null keyword. The second argument is the string that is required to display. The title bar of message dialog contains the string "Message" to indicate that the dialog is presenting a message to the user. The default icon is INFORMATION_MESSAGE.

Example: JOptionPane.showMessageDialog(null,"value="+x);



Syntax2:

JOptionPane.showMessageDialog(null,argument2,argument3,argument4)

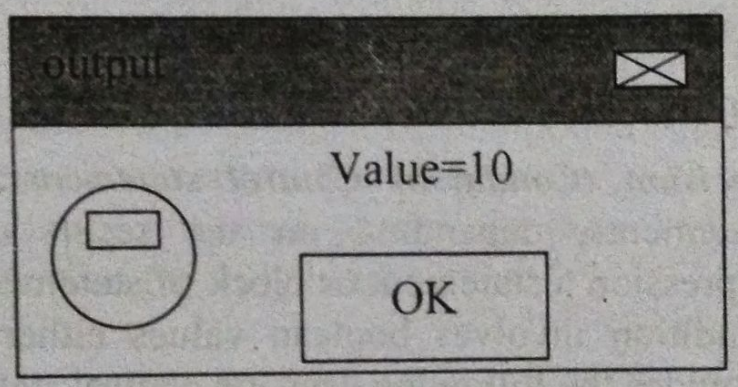
The first argument must be null keyword. The second argument is the message to display. The third argument is the string to display in the title bar of the dialog. The fourth argument is a value indicating the type of message dialog to display.

For argument4 some of the message dialogs are

JOptionPane.PLAN_MESSAGE
JOptionPane.ERROR_MESSAGE
JOptionPane.INFORMATION_MESSAGE
JOptionPane.WARNING_MESSAGE
JOptionPane.QUESTION_MESSAGE

Example:

JOptionPane.showMessageDialog(null,"value="+x,"output",
JOptionPane.ERROR_MESSAGE)



import statement: import statement is similar to the #include statement in C. Using the import statements, we can access to classes that are part of other packages.

Example: import javax.swing.JOptionPane;

This statement instructs that the interpreter to load the JOptionPane class contained in the javax.swing package.

Example:

```
import javax.swing.JOptionPane;
class ch
{
    public static void main(String[] args)
    {
    String sn1;
```

```
int x;
sn1=JOptionPane.showInputDialog("Enter x value");
x=Integer.parseInt(sn1);
JOptionPane.showMessageDialog(null,"value="+x,"output",JOpt
ionPane.ERROR_MESSAGE);
}
}
```

2.21 Control Statements

Depending on the results of computations the flow of control may require to jump from one part of the program to another part, such jumps are called Control statements. The control statements are of two categories.

1. Decision (Conditional) control statements
2. Loop control statements
3. Jump control statements

1. ***Decision (Condition) Control statements***: In decision control statements, depending on the result of evaluation of an expression a statement or block of statements are executed. The condition involves boolean values either true or false. Java provides the following decision control statements.

a) if statement

b) if-else statement

c) nested if-else statement

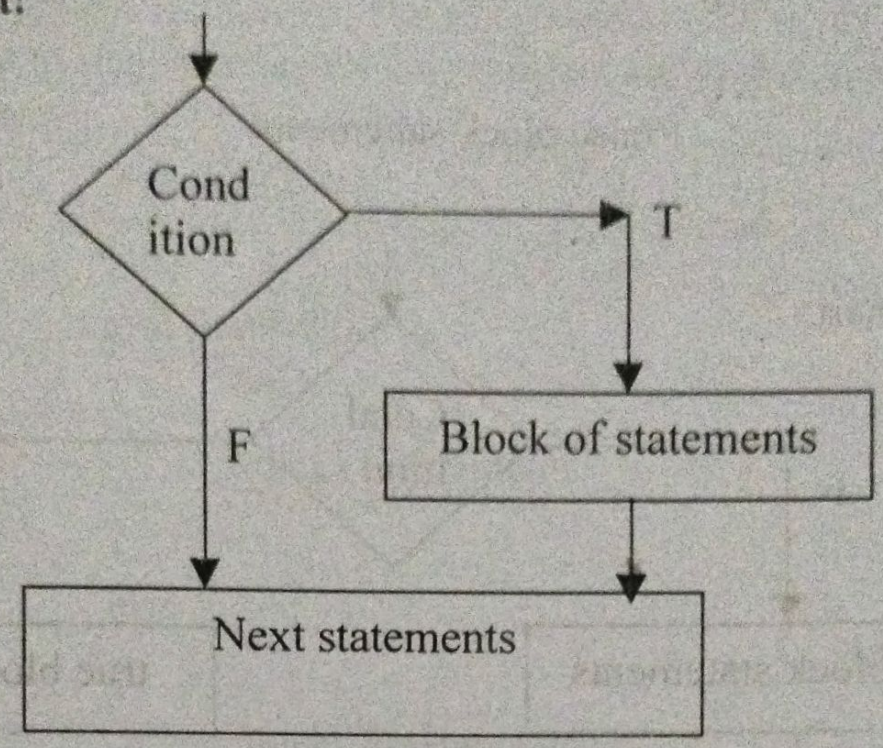
d) switch

a) ***if statement:*** The if statement conditionally process the statements when the specified test condition is true.

Syntax: if(condition)

```
{
    // Block of statements
}
```

Flowchart:



Here, if the condition is true then block of statements are executed. Otherwise, block of statements are not executed and the control is passed to the next statements available after the block statements.

The statements are either simple or compound statements. If the statements are single statement then they ended with a semicolon. If the statements are compound statements then they are placed in between the left and right curly braces.

Example:

```
class if1
{
        public static void main(String[] args)
        {
                int a=6,b=4;
                if(a>b)
                System.out.println("a is big");
        }
}
```

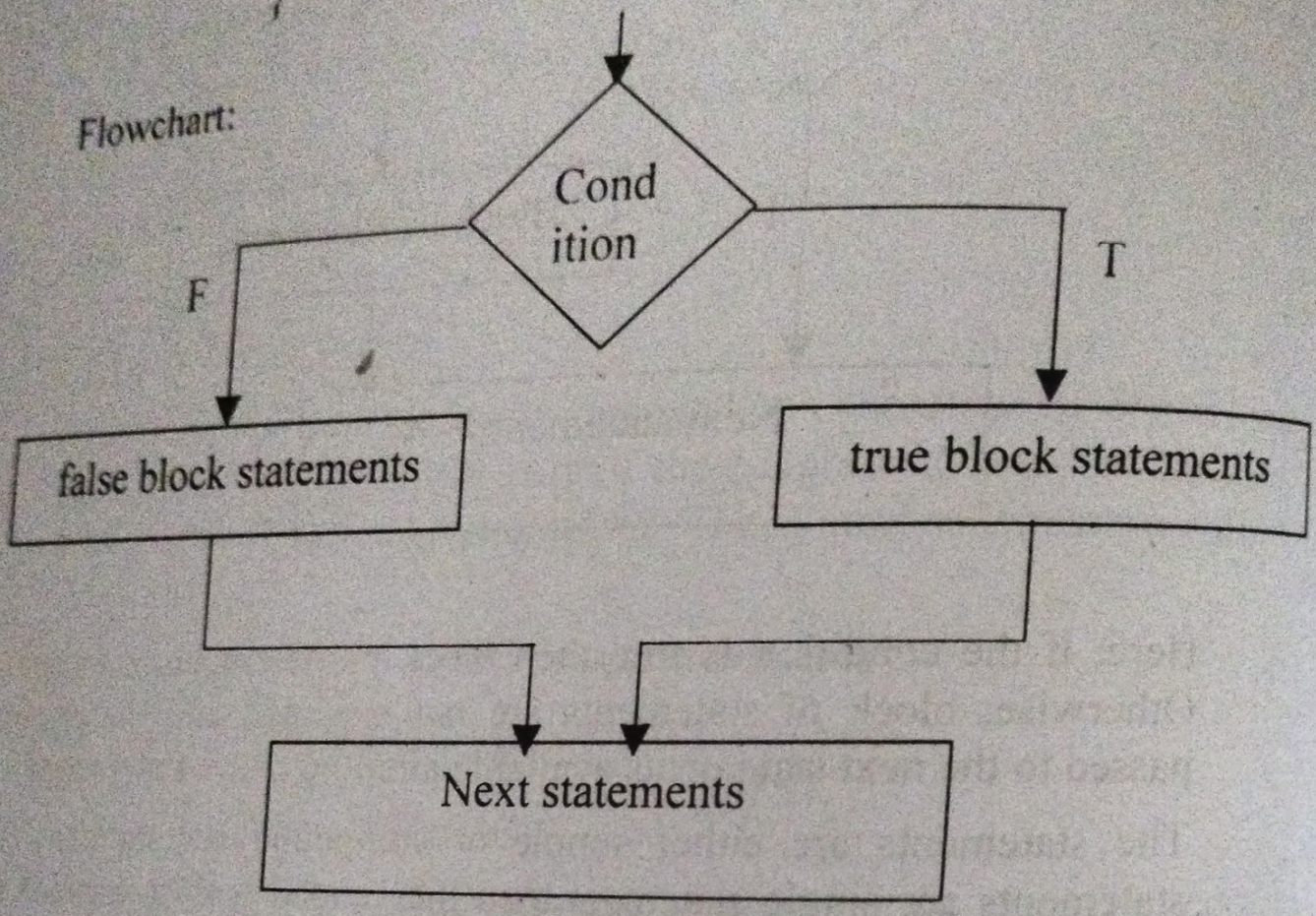
b) if-else statement: In if-else statement general form is as

Syntax:

```
if (condition)
{
    //true block statements
}
```

```
else
{
    //false block statements
}
```

Flowchart:



Here, first the test condition is evaluated. If the condition is true, then true block statements are executed otherwise false block statements are executed.

The statements are either simple or compound statements. If the statements are single statement then they ended with a semicolon. If the statements are compound statements then they are placed in between the left and right curly braces.

Example:

```
class if2
{
    public static void main(String[] args)
    {
        int a=2,b=4;
        if(a>b)
        System.out.println("a is big");
        else
        System.out.println("b is big");
    }
}
```

c) ***Nested if-else statement:*** The if-else statement is placed within another if-else statement such methods are called nested if-else statement.

Syntax:

```
if (condition1)
{
else if(condition2)
{
        //block1 statements
}
else
{
        //block2 statements
}
}
else
{
        //block3 statements
}
```

Lab Program1: Write a program to print the roots of a quadratic equation $ax^2+bx+c=0$

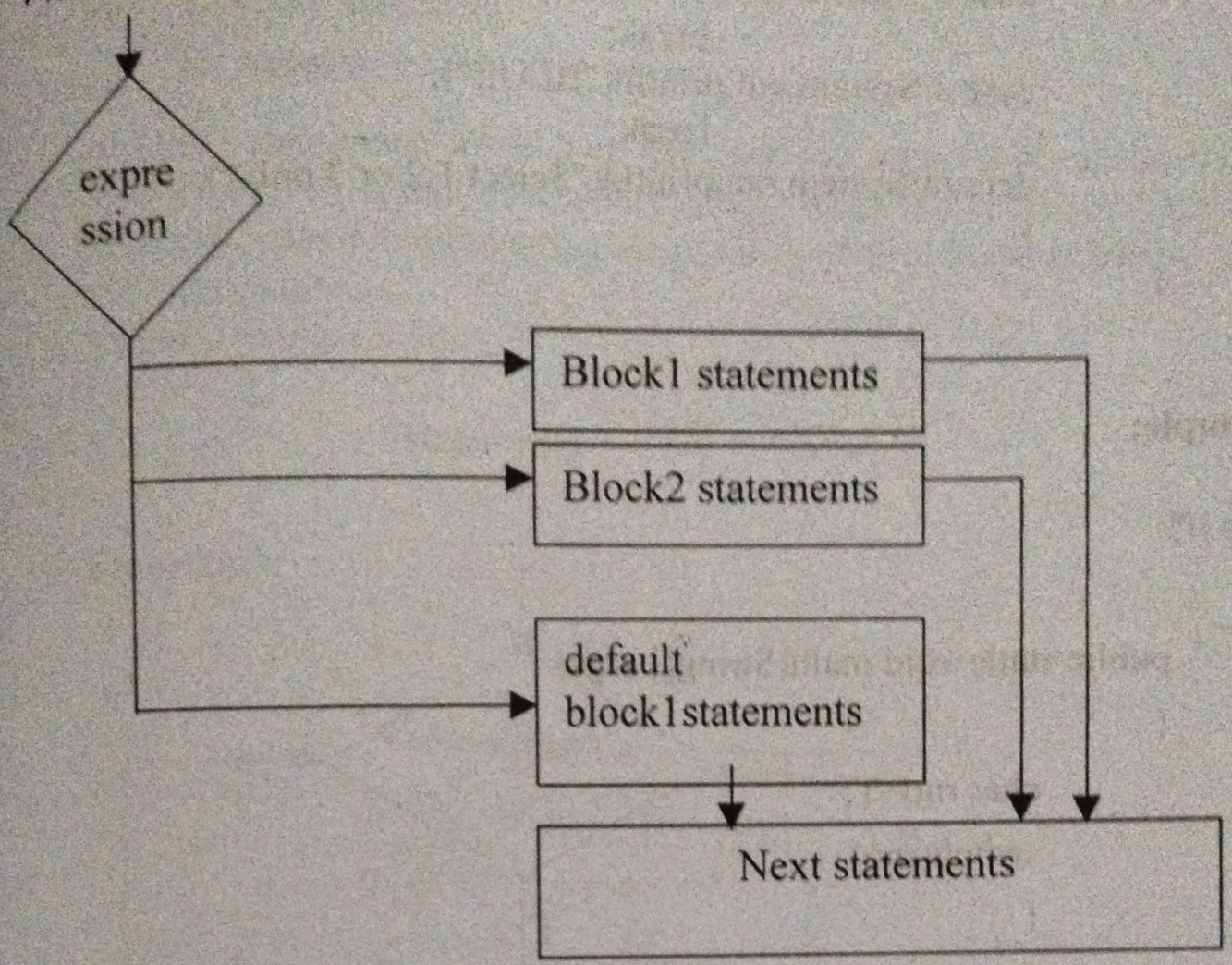
```
import javax.swing.JOptionPane;
import java.text.DecimalFormat;
class lab1
{
        public static void main(String[] args)
        {
        double a,b,c,dis,root1,root2,p;
        DecimalFormat d=new DecimalFormat("0.00");
        a=Double.parseDouble(JOptionPane.showInputDialog("Enter
        a value"));
        b=Double.parseDouble(JOptionPane.showInputDialog("Ente
        r b value"));
```

```
c=Double.parseDouble(JOptionPane.showInputDialog("Enter
c value"));
dis=(b*b-4*a*c);
if(dis==0)
{
System.out.println("Roots are real and equal");
root1=-b/(2*a);
System.out.println("Root1="+(d.format(root1)));
System.out.println("Root2="+(d.format(root1)));
}
else if(dis>0)
{
System.out.println("Roots are real and different");
p=Math.sqrt(dis);
root1=(-b+p)/(2*a);
root2=(-b-p)/(2*a);
System.out.println("Root1="+(d.format(root1)));
System.out.println("Root2="+(d.format(root2)));
}
else
System.out.println("Roots are imaginary");
}
}
```

d) ***switch statement:*** The switch statement allows section among the multiple section of a code depending on the value of an expression. The objective is to check several possible constant values for an expression.

```
Syntax:switch (expression)
        {
        case value1:    block1 statements
                        break;
        case value2:    block2 statements
                        break;
        .
        .
        default:        default block statements
        }
```

Flowchart:



First, the switch statement evaluates the expression and check whether the evaluated value is coinciding with any case value or not. If any value is coinciding, it executes that particular block of statements until the break statement is reached. After executing the statements the control is transferred out of the statements that are available after the switch statement.

If any value is not coinciding with the case value then it executes default block statements. Default block is an optional.

Note: case values are either integer constant or character constant.

Example:

```
class if4
{
        public static void main(String[] args)
        {
                int rno;
                rno=Integer.parseInt(args[0]);
                switch(rno)
                {
                case 1:System.out.println("RED");
                                        break;
```

```
                case 2:System.out.println("GREEN");
                                break;
                case 3:System.out.println("BLUE");
                                break;
                default:System.out.println("Select 1,2 or 3 only");
                }
        }
}
```

Example:

```
class if5
{
        public static void main(String[] args)
        {
                char rno='t';
                switch(rno)
                {
                case 'a':System.out.println("RED");
                                break;
                case 'b':System.out.println("GREEN");
                                break;
                case 'c':System.out.println("BLUE");
                                break;
                default:System.out.println("Select 1,2 or 3 only");
                }
        }
}
```

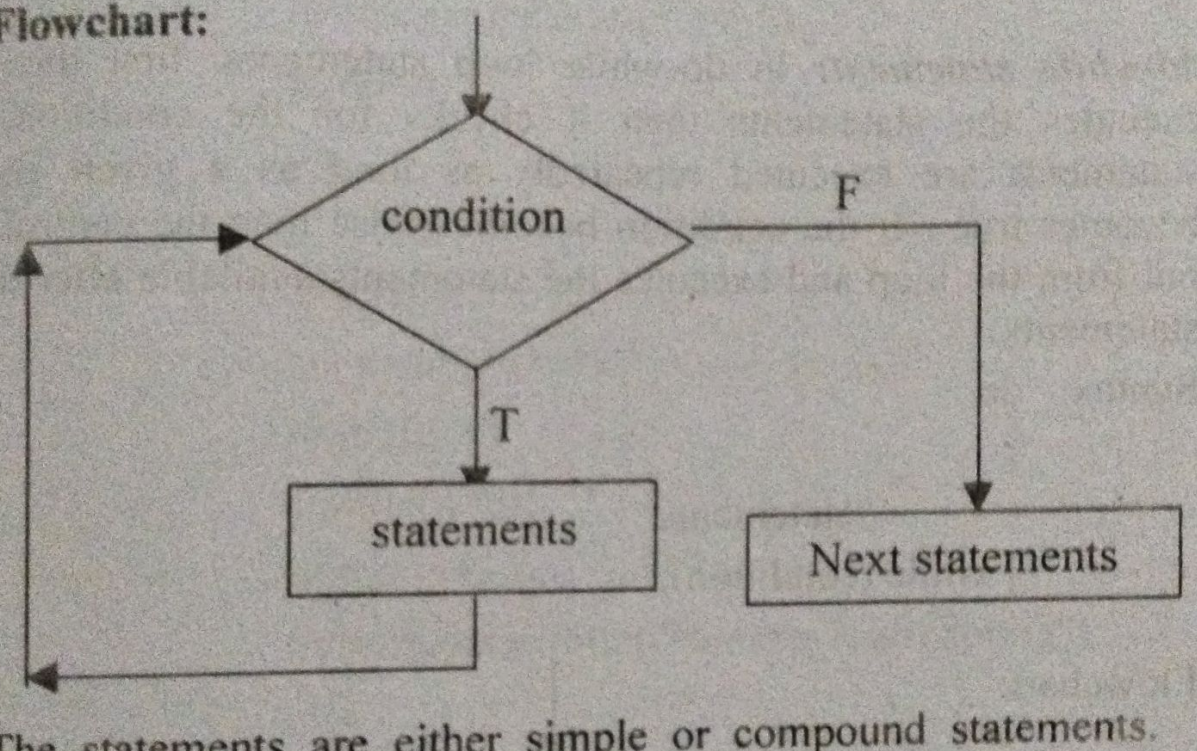
2. **Loop Control Statements:** A Loop statement allows to run a statement or block of statements repeatedly for a certain number of times. The repetition is continues while the condition is true. When the condition becomes false, then loop ends and control passed to the next statements following the loop. Repetitive execution of statements is called looping. Java provides three types of loop control statements.

a) while statement
b) do-while statement
c) for statement

a) while statement: while loop executes the statements repeatedly as long as a given condition becomes true. When the condition becomes false the control immediately pass to the next statements available after the loop statements.

Syntax:
```
while (condition)
{
        //statements
}
```

Flowchart:



The statements are either simple or compound statements. If the statements are single statement then they ended with a semicolon. If the statements are compound statements then they are placed in between the left and right curly braces.

Lab Program2: Write a program to print the first n fibonacci sequence numbers

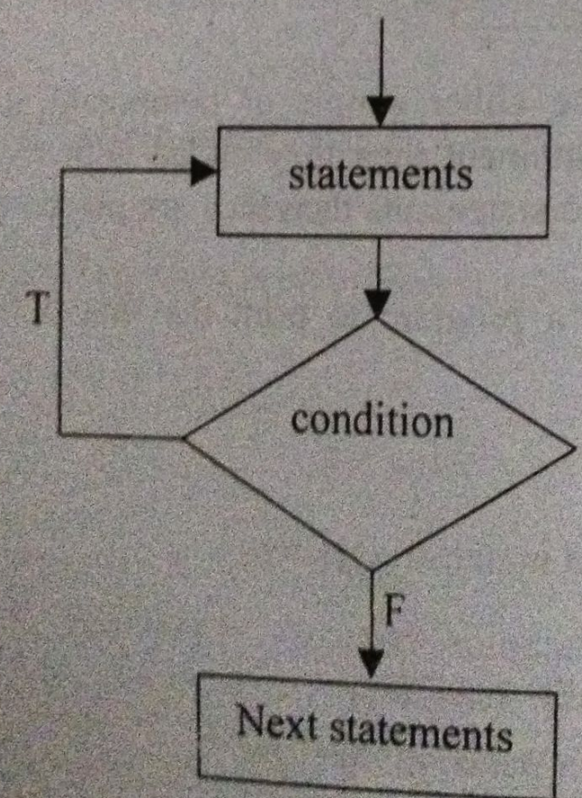
```
class lab2
{
    public static void main(String[] args)
    {
        int f0=1,f1=1,i=3,f2,n;
        n=Integer.parseInt(args[0]);
        System.out.println(f0);
        System.out.println(f1);
```

```
while(i<=n)
{
f2=f0+f1;
System.out.println(f2);
f0=f1;
f1=f2;
i++;
}
}
}
```

b) ***do-while statement:*** In do-while loop statements, first the control executes the statements then it checks for the condition. The statements are executed repeatedly as long as a given condition becomes true. If the condition becomes false then the control comes out from the loop and executes the statements available after the loop statements.

Syntax:
```
do
{
    //statements
} while (condition);
```

Flowchart:



The statements are either simple or compound statements. If the statements are single statement then they ended with a semicolon. If

the statements are compound statements then they are placed in between the left and right curly braces. The main difference between while and do-while statement is do-while statement executes the statements at least once even the condition becomes false.

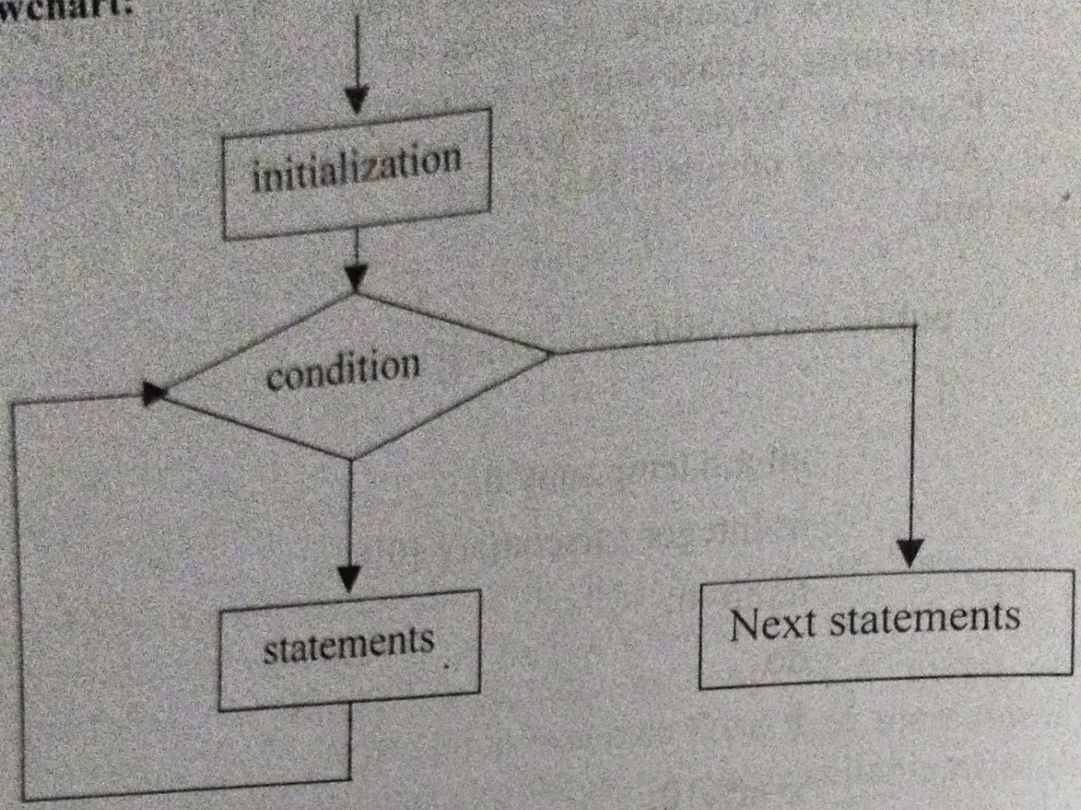
Example: Write a program to check whether a given number is Armstrong number (153) or not

```
class labd
{
    public static void main(String[] args)
    {
        int x,n,temp,sum=0;
        n=Integer.parseInt(args[0]);
        temp=n;
        do
        {
        x=n%10;
        sum=sum+x*x*x;
        n=n/10;
        }while(n>0);
        if(sum==temp)
        System.out.println("Number is armstrong");
        else
        System.out.println("Number is not armstrong");
    }
}
```

c) for statement: The for statement also repeat a statement or compound statements for a specified number of times. The general form of for statement is as

Syntax:
```
for (initialization;condition;increment/decrement)
{
    //statements
}
```

Flowchart:



Initialization is the start with the assigning value to the variable and it executes only once at the start of the loop. Then the control checks for the test condition.

In condition section, if the condition becomes true then the control pass to the block of statements and executed. After executing the statements the control pass to the increment/decrement section. After incrementing/decrementing the variable again the control is passed to the condition section. This procedure is repeated as long as the condition becomes true.

If the condition becomes false, the control passes out of the loop statements.

Note: 1. Multiple initializations are possible by comma operator.

Ex: for(i = 0,j = 5; i<n; i++, j--)

2. At the place of condition section, it is also possible to use relational expressions