UNIT-1

I. Program Structure in Java:

1. Introduction
2. Writing Simple Java Programs
3. Elements or Tokens in Java Programs
4. Java Statements
5. Command Line Arguments
6. User Input to Programs
7. Escape Sequences
8. Comments
9. Programming Style.

II. Data Types Variables and Operators :

1. Introduction Data Types in Java
2. Declaration of Variables Data Types
3. Type Casting
4. Scope of Variable Identifier
5. Literal Constants, Symbolic Constants
6. Formatted Output with printf() Method
7. Static Variables and Methods
8. Attribute Final
9. Introduction to Operators
10. Precedence and Associativity of Operators
11. Assignment Operator (=)
12. Basic Arithmetic Operators
13. Increment (++) and Decrement (- -) Operators
14. Ternary Operator Relational Operators
15. Boolean Logical Operators
16. Bitwise Logical Operators.

III. Control Statements:

1. Introduction
2. if Expression
3. Nested if Expressions
4. if–else Expressions
5. Ternary Operator?:
6. Switch Statement
7. Iteration Statements
8. while Expression
9. do–while Loop
10. forLoop
11. Nested for Loop
12. For–Each for Loop
13. Break Statement
14. Continue Statement.

I. Program Structure in Java:

1. Introduction

Java is an Object Orient and high-level programming language, originally developed by Sun Microsystems and released in 1995. Java runs on a variety of platforms, such as Windows, Mac OS, Linux etc.

In Java, there are three types of programs as follows:
Stand- aline application programs : These programs are made and run on users computers
Applet programs : These programs are meant to run in a web page on the Internet.
Java Servlets : These programs run in computers that provide web services. They are also oftern called server isde programs or servlets.

2. Writing Simple Java Programs

Java is an Object oriented language. Every java program imports packages which are provides necessary classes and interfaces.
For example:

```
import  java.util.*;
import java.io.*;
```

After import statement, every java program starts with the declaration of the class. A program may have one or more classes.

A class declaration starts with the keyword class, followed by the identifier or name of the class. Giving the name of a package at the top is optional.

Class declarion contains name of the class and body of the class. The body of the class may consist of several statements and is enclsed between the braces {}.

A sample of class declarions is as follows.

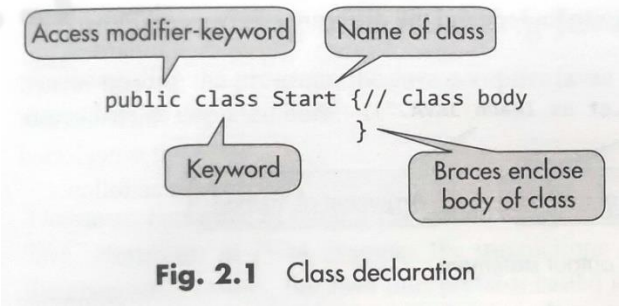


Fig. 2.1 Class declaration

Here:

public is access specifier. This class is accessible to any ouside code. Otherwise the class is accessibe to only same package classes.

class is a keyword of java language which is used to declare the class.

The class body starts with the left brace { and ends with the right colsing brace }.

// are comments which are neglected by the compiler.

A class body may comprise statements for declaration of variables. constants, expressions, and methods.

Simple program in Java. Save this code in Text Document as "**Start.java"**

```
public class Start()
{
   public static void main()
   {
         System.out.println("Hello World");
   }
}
```

Here:

class is a keyword

Start is a name the class. Here class is declared as public, so it is available to all the classes.

main() is the method which initiate and terminate the program (in java functions are called as Methods)

println() is method of object "**out"**. "**out**" is the object of class "**System".** println() prints the string "Hello World" on the screen/monitor.

Compiling and Running Java program

Java compiler first converts the source code into an intermediate code, known as bytecode or virtual machine code. To run the bytecode, we need the Java Virtual Machie (JVM). JVM exists only inside the computer memory and runs on top of the Operating System. The bytecode is not machine specific. The Java interpreter converts the bytecode into Machine code. The following diagram illustrates the process of compiling and running Java programs.

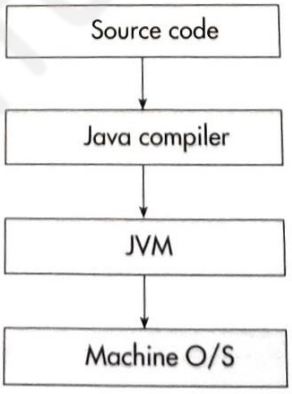


Fig. 2.3 Compiling and running a Java program

For compiling the program, the Java compiler javac is run, specifying the name of the source file on the command line as depicted here:

javac Start.java

The Java compiler creates a file called Start.class containing the bytecode version of the program. The java interpreter in JVM executes the instructions contained in

this intermediate Java bytecode version of the program. The Java interpreter is called with "java" at the command prompt.

C:\> java Start

Output:

Hello World

Here java command calls the Java interpreter which executes the Start.class (bytecode of Start.java).

3. Elements or Tokens in Java Programs

Java program contains different types of elements like white spaces, comments and tokens. A token is the smallest program element which is recognized by the compiler and which treats them as defined for the compiler. A program is a set of tokens which comprise the following elements:

Identifiers or names: Identifier is the name of variables, methods, classes etc.

Rules for framing Names or Identifiers.

- It should be a single word which contains alphabets a to z or A to Z, digits 0 to 9, underscore (_).
- It should not contain white spaces and special symbols.
- It should not be a keyword of Java.
- It should not be Boolean literal, that is, true or false.
- It should not be null literal.
- It should not start with a digit but it can start with an underscore.
- It can comprise one or more unicode characters which are characters as well as digits.

Conventions for Writing Names

- Names of packages are completely in lower-case letters such as mypackage, java.lang.
- Names of classes and interfaces start with an upper-case letter.
- Names of methods start with a lower-case character.
- Names of variables should start with a lower-case character.

Keywords: These are special words defined in Java and represent a set of instructions.

- The keywords represent groups of instructions for the compiler.
- These are special tokens that have a predefined meaning and their use is restricted.
- keywords cannot be used as names of variables, methods, classes, or packages.
- These are written in the lower case.
- Keywords of Java Language are as follows:

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

Literals: These are values represented by a set of character, digits, etc.

- A literal represents a value which may be of primitive type, String type, or null type.
- The value may be a number (either whole or decimal point number) or a sequence of characters which is called String literal, Boolean type, etc.
- A literal is a constant value.

Types of Literals:

i. Integer literals

- Sequences of digits.
- The whole numbers are described by different number systems such as decimal numbers, hexadecimal numbers, octal numbers, and binary numbers.
- Each number has a different set of digits.

Decimal Integer Literals

- These are sequences of decimal digits which are 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9.
- Examples of such literals are 6, 453, 34789, etc.

Hex Integral Literals

- These are sequences of hexadecimal digits which are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F.
- The values 10 to 15 are represented by A, B, C, D, E, and F or a, b, c, d, e, and f.
- The numbers are preceded by 0x or 0X. Examples are 0x56ab o0X6AF2, etc.

Octal Integer Literals

- These are sequences of octal digits which are 0, 1, 2, 3, 4, 5, 6, and 7.
- These numbers are preceded by 0. Examples of literals are 07122, 04, 043526.

Binary Literal

- These are sequences of binary digits.
- Binary numbers have only two digits—0 and 1 and a base 2.
- Examples of such literals are 0b0111001, 0b101, 0b1000, etc.

ii. Floating point literal

- These are floating decimal point numbers or fractional decimal numbers with base 10. Examples are 3.14159, 567.78, etc.

iii. Boolean literal

- These are Boolean values. There are only two values—true or false.

iv. Character literal

- These are the values in characters.
- Characters are represented in single quotes such as 'A', 'H', 'k', and so on.

v. String literal

- These are strings of characters in double quotes. Examples are "Delhi", "John", "AA", etc.

vi. Null literal

- There is only one value of Null Literal, that is, null.

Separators: These include comma, semicolon, period(.), Parenthesis (), Square brackets [], etc

Symbol	*Name*	*Description*
()	Parentheses	Parentheses are used for the following purposes: 1. To change precedence level in expressions 2. To contain a list of parameters in a definition of methods 3. To contain expression in control statements 4. To enclose cast types in explicit type casting
{}	Braces	Braces are used for the following purposes: 1. To enclose definitions of classes and methods 2. To enclose blocks of statements 3. To initialize arrays and string objects 4. To enclose local scopes
[]	Square brackets	Square brackets are used for enclosing index values in array elements and for defining arrays
,	Comma	Commas are used for separating the following: 1. Variables in parameter or declaration lists 2. Identifiers in variable declarations 3. Expressions in for loop statement
;	Semicolon	A semicolon is used for terminating statements in a program.
.	Period	The period is used for the following: 1. To link an object of class with its method 2. To link classes with sub-packages and sub-packages to packages

Operators: Operators are mostly represented by symbols such as +, -, *, etc

Types of Operators:

Arithmetic Operators:

Operator	Description
+	Addition or Unary plus
-	Subtraction or Unary minus
*	Multiplication
/	Division
%	Modulus

Relational Operators:

Operator	Description
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to
==	Equal to
!=	Not equal to

Logical Operators:

Operator	Description
&&	Greater than
||	Greater than or equal to
!	Less than

Assignment Operators:

Operator	Description
+=	Add and assign to
-=	Subtract and assign to
*=	Multiply and assign to
/=	Divide and assign to
%=	Modulus and assign to

Increment / Decrement Operators:

Operator	Description
++	Increment by one
--	Decrement by one

Bitwise Operators:

Operator	Description
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR
~	Bitwise compliment
>>	Shift Right
<<	Shift Left
>>>	Shift right with Zero fill

Conditional Operators:

Operator	Description
?:	Used to construct Conditional expression

4. Java Statements

A Statement is a instruction to the computer. A program is a set of statements or instructions. The statements specify the sequence of actions to be performed when some method or constructor is invoked. The statements are executed in the sequence in the specified order. The important Java statements are as follows.

Statement	Description
Empty statement	These are used during program development.
Variable declaration statement	It defines a variable that can be used to store the values.
Labeled statement	A block of statements is given a label. The labels should not be keywords, previously used labels, or already declared local variables.
Expression statement	Most of the statements come under this category. There are seven types of expression statements that include assignment, method call and allocation, pre-increment, post increment, pre-decrement, and post decrement statements.
Control statement	This comprises selection, iteration, and jump statements.
Selection statement	In these statements, one of the various control flows is selected when a certain condition expression is true. There are three types of selection statements including if, if-else, and switch.
Iteration statement	These involve the use of loops until some condition for the termination of loop is satisfied. There are three types of iteration statements that make use of while, do, and for
Jump statement	In these statements, the control is transferred to the beginning or end of the current block or to a labeled statement. There are four types of Jump statements including break, continue, return, and throw.
Synchronization statement	These are used with multi-threading
Guarding statement	These are used to carry out the code safely that may cause exceptions (such as division by zero, and so on). These statements make use of try and catch block, and finally

5. Command Line Arguments

- A Java application can accept any number of arguments from the command line.
- These arguments can be passed at the time of running the program. This enables a programmer to check the behavior of a program for different values of inputs.
- When a Java application is invoked, the runtime system passes the command line arguments to the application's main method through an array of strings.
- It must be noted that the number of arguments
-
- in an array. To ensure this, we can make use of the length property of the array. This is illustrated in

Program 2.2: Example showing command line argument

```
class vehicle
 public static void main(String args[])
{
      int x = args.length;
      for(int i=0; i<x; i++)
    {
      System.out.println(args[i]);
    }
}
Output
```

(After compiling, type the following lines on the command prompt. It produces the output as)

```
C:\> Java vehicle “Car Cycle Motorbike"
    Car
    Cycle
    Motorbike
```

Program 2.3: Illustration of command line

```
class Sum
{
public static void main{String args)

int  s=1;

for(int i e; icargs length; 1++){

s=s+Integer.parseInt(args(1]);

 System.out.prinln(" The addition of passed arguments 15" );

}
```

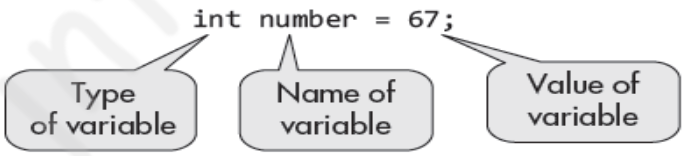
Output

```
After compiling, the following lines are typed on the command
prompt: It produces the output as

C:\> Java Sun 20 15 17
```

Declaration of Variables

- A program may involve variables: variables are objects whose values may change in the program.
- A variable is declared by first writing its type, followed by its name or identifier as illustrated here.



- However, a variable should also be initialized, that is, a value should be assigned to it before it is used in an expression. The line ends with a semicolon (:) as shown in the above figure.

Examples:

```
double price = 28.5;
char ch = "C":
String name - "John";
```

Program 2.4 illustrates the declaration and output of some data types

```
class  PrintOut
{
  public static void main (String args[])
  {
   String name = "Sunita"; //"name" is variable, value is "Sunita"
   String str = "Hello"; //String set has value- "Hello!"

   int length = 50; // Variable name "length", value 50
   int width = 8; // Variable name "width" value 8

   System.out.print("Name= " + name);
   System.out.print("Str = "+ str);  //print statement

   System. out.println();

   System.out.println("Length "+ length);
   System. out.println("Width " + width);

   System.out.println(str + name); // println statement
  }
}

c:\>javac PrintOut.java

c:\>java PrintOut
Name= SunitaStr = Hello
Length 50
Width 8
HelloSunita
```

6. User Input to Programs

- The class Scanner of package java.util to carry out input to the program.
- Java Scanner class is a text scanner that breaks the input into tokens using a delimiter. The delimiter is whitespace by default.
- Importing the class is the first step.
  `import java.util.Scanner;`
- The object of the class Scanner is declared as follows.
  `Scanner scaninput = new Scanner (System. in);`
- The object "`scaninput`" invokes the method `nextInt()` which reads the value typed by the user. This value is assigned to n. The value of m is similarly obtained.
- The other useful methods of Scanner class are nextDouble() and nextLine().

Program 2.6: Illustration of a user's input from keyboard into program

```
import java.util.Scanner;
public class Arithmetic
{
      public static void main(String[] args)
      {
      Scanner scaninput = new Scanner (System. in);
      int n;
      int m;
      System.out. print( "Enter the value of n : ");
      n=scaninput.nextInt();
      System.out. print( "Enter the value of m : ");
      m=scaninput.nextInt();

      System.out.println("Sum of two numbers is ="+(n+m));

      System.out.println("Product of two numbers is ="+ n*m);

      System.out.println("Modulus of (n % m) is =" + n%m);

      System.out.println("Division of two numbers is =" + n /m );
      }
}
```

Output

```
C:\>javac Arithmetic.java

C:\>java Arithmetic
Enter the value of n : 10
Enter the value of m : 3
Sum of two numbers is =13
Product of two numbers is =30
Modulus of (n % m) is =1
Division of two numbers is =3

C:\>
```

7. Escape Sequences

Escape Sequences character is preceded by a backslash (\)has a special meaning to the compiler. Escape sequences are as follows.

Table 2.7 Escape sequences

Escape sequences	Description	Example of code
\’	Single quote	“\’”
\”	Double quote	“\”””
\\	Backslash	“\\”
\b	Back space	“\b”
\ddd	Octal character	“\132”
\uxxxx	Hexadecimal unicode character	“\u0042”
\f	Form feed	“\f”
\n	New line	“\n”
\r	Carriage return	“\r”
\t	Tab	“\t”

Program 2.9: Program on Escape Sequences

```
class Escape
{
   public static void main(String[] args)
   {
   int n=256, a=0, b=70;
   System.out.println("Value of a =" + a + "\n b= "+b +"\n");
   System.out.println("\u0041 \t" + " \u0042 \t"+"\132");
   System.out.println("\"Value of b\" = "+b);
   System.out.println("\'Value of n\' = " +n);
   }
}
```

Output:

```
C:\>javac Escape.java

C:\>java Escape
Value of a =0
 b= 70
A        B      Z
"Value of b" = 70
'Value of n' = 256
```

8. Comments

- Comments are Line of Text which is not a part of the compiled program.
- Comments are used for documentation to explain source code.
- They are added to the source code of the program.
- Java supports three types of comments as:

1. Single-line comment: These comments are started with two front slash characters (//)
   Example:
   ```
   // This is Single line comment
   ```

2. Multi-line comment : These comments are enclosed with /* and */
   Example:
   ```
   /* It is Multi line
      Comments */
   ```

3. Documentation comment: These comments are enclosed with /** and */. It is different from multi line comments in that it can extracted by javadoc utility to generate an HTML document for the program.
   Example:
   ```
   /** It is documentation
      Comments */
   ```

9. Programming Style.

- An analysis of the programming exercises will throw some light on the look and feel of a program. The team members should easily understand each other's code.
- For a beginner, it is better to develop a habit of writing a program in a proper style so that there is no conflict between the current habits and the company's imposition of style rule at a later stage.
- There are no set patterns of good style and bad style: however, if the programmer takes care of a few requirements on the programs as discussed, the resulting style will be better

1. **The program should present a clean and orderly look. In order to develop a clean program, the programmer can adapt the following measures:**

(a) The white spaces are neglected by the compiler.

(b) Indenting is another method often used to improve the looks and readability

(c) Use of Lambda expression and method reference of Java 8 enhances the look

(d) Include a space before and after operators

2. **The program should be easy to understand. The programmer should take care of the following aspects:**

(a) If it is team work, certain conventions about naming should be preset so that a team member can easily identify an item.

(b) The makers of Java have a set of rules which are followed in the Java library. The same rules or an even better convention may be set.

(c) Judicial use of comments can increase understandability. Use of too many comments makes confusion in the program.

(d) It is better to use already defined and tested library methods rather than user-defined methods

(e) Expressions like z = x++ + - - c*++k; should be avoided.

3. **Debugging should be easy. The programmer may adopt the following measures to ensure easy debugging.**

(a) The vertical alignment of a similar item enhances the ability to find errors.

(b) The code line should not be too long.

(c) The variables should be declared close to the places of their use

(d) If it is a big program, it should be divided into small segments. In Java, it is easy because the program may comprise separate classes.

(e) The methods should not be too big.

(f) Each source code file should have one class.
(g) The braces should be vertically aligned, if possible.

4. The program should be easy to use. The following points

(a) The input into program should be easy,

(b) The output should be self-explanatory should be taken care of

(c) The names used should imply the output type such as price, weight, length, and so on.

(d) Confusing and long names must be avoided.

5. **The program should be easy to maintain**

(a) The program should be easily modifiable to ensure simplicity in fixing errors.

(b) The comments can help in modification of the program, fixing errors.

6. The program should be fail-safe. The failure of a program should not be catastrophic.

II. Data Types Variables and Operators :

i. Introduction Data Types in Java

The Java language programs deal with the following entities:

I. Primitive data

2. Classes and objects

3. Interfaces and their references

4. Arrays

5. Methods

The primitive data and their types are defined independent of the classes and interfaces, and the arrays and methods derive their types from the first three entities.

An array is n collection of items that may be of primitive type, class objects, or references. The type of an array can be determined from the type of elements present in it.

Data Types in Java

- Data Type is the type of the data which computer accepts. Every variable and expression has a data type that is known at the compile time.
- The declaration of data type with a variable or expression limits the types of values that a variable can have or the expression it can evaluate.
- Java is an object-oriented programming language based on classes and interfaces.
- Java defines some primitive (basic) data types that are not reference types of any class or interface. Eight primitive (basic) types of data are defined in Java. The type names are also the keywords shown here in bold letters
  1. **Integral types**—byte, short, int, long
  2. **Floating point types** - float, double
  3. **Character type** -char
  4. **Boolean type** – Boolean values – True, False

There is a non-data type called void and no data can be of type void. This type is used for methods that do not return any value.

Java is a case-sensitive language. This means that it takes Area, area, and AREA are three different objects.

2. Declaration of Variables - Data Types

A variable is declared by first declaring its type followed by its identifier or name, which is given as follows:

```
type  Identifier;
```

Here type is the primitive data type and Identifier is the name of the variable.

Declaration and Initialization of the variable is as follows.

```
type identifier = value;
```

Ex:

byte n; // declares a variable of type byte.

short m = 67; // declares and initiates a short number

int length; //declares length-a variable of type int

length - 50; // value is re-assigned after declaration

char ch = 'A'; // declares a character variable ch.

Non-primitive Types

These are the class and interface types. The name of a class or interface is the name of type. A class object is declared as

```
Class_identifier   object_identifier;
```

Similarly, an interface reference is declared as

```
Interface_identifier   reference_identifier;
```

Example:

```
String str "Delhi":
```

Data Types

i. Integers

Integers are whole numbers, that is, they represent numbers that do not have a fractional part. The integers can be declared in four types according to the size of memory allocated for them

byte

short

int

long

Table 3.1 Integer types and ranges

Type name	Memory allocated in bytes	Range of values of variables
byte	1 byte (8 bits)	–128 to 127
short	2 bytes (16 bits)	–32,768 to 32,767
int	4 bytes (32 bits)	–2,147,483,648 to 2,147,483,647
long	8 bytes (64 bits)	–9,223,372,036,854,775,808 to 9,223,372,036,854,775,807

Program 3.1 illustrates the integer data types.

```
class DataType
{
      public static void main (String args[])
      {
      byte a= 4, b=8 ;// variables of type byte
      short c = 67,  d = 98; // variables of type short
      int e = 7000, f = 20000; // variables of type int
      long secondsInYear = 365 * 24 * 60 * 60; // long type
      //e=d+f;
      System.out.println("(b + a) = " + (b+a));
      System.out.println("b + a = " + b + a);
      System.out.println("c = "+ c + " \td = " + d);
      System.out.println("e = " + e + "\t f= " + f);
      System.out.println("Seconds in a year = "+  secondsInYear);
      }
}
```

Output:

```
C:\ >javac DataType.java
C:\ >java DataType
(b + a) = 12
b + a = 84
c = 67  d = 98
e = 7000         f= 20000
Seconds in a year = 31536000
```

ii. Characters

- A variable may have value in terms of a character in which the type of variable is char.
- These characters represent integer values.
- Java supports Unicode for the representation of characters.
- Unicode supports all the character sets of all the major languages of the world.
- The initial version of Unicode allocated 2 bytes for storing characters.
- The range of values for characters in the initial version of Unicode comprised from '\u0000' to '\uffff ' that is from 0 to 65535 both end values inclusive.

Program 3 2 illustrates anthmetic operations on character constants and variables

```
class Datachar {
      public static void main (String args[])
      {
      char ch1='E', ch2, ch3 ;

      System.out.println("ch1 =" +ch1); // printing ch1

      ch2=ch1++;
      System.out.println("ch2 =" + ch2); // printing ch2

      ch3=++ch1;
      System.out.println("ch3 =" + ch3); // printing ch3
      }
}
Output:
C:\ >javac Datachar.java
C:\ >java Datachar
ch1 =E
ch2 =E
ch3 =G
```

iii. Floating Point Numbers

- The numbers that are not whole numbers, or those that have fractional part, Examples are 3.141, 476.6, and so on.
- Java supports two types of such numbers.

Float: This type is used for single-precision decimal floating point numbers, that is, 7 digits after the decimal point. These numbers are stored on 4 bytes.

Program 3.4: Illustration for working with float and double data

```
class FloatType
{
      public static void main (String args[])
      {
      float width =20.0f,  length = 40.5f;
      float rectArea = length *  width;

      System.out.println("Width = "+ width);
      System.out.println("Length = "+ length);
      System.out.println("Rectangle Area = "+ rectArea);

      double dia=10.0, pi=3.14159;
      double areaCircle = pi* dia * dia / 4;

      System.out.println("Diameter " + dia);
      System.out.println("Area of Circle = " + areaCircle);
      }
}
```

Output

```
C:\>javac FloatType.java
C:\>java FloatType
Width = 20.0
Length = 40.5
Rectangle Area = 810.0
Diameter 10.0
Area of Circle = 78.53975
```

iv. Boolean Type - Data

- For dealing with logical statements, variable of type boolean is supported.
- The value of boolean type variable is either *true* or *false*.
- The boolean type variables are *unsigned* as similar to char type variables.
- The variable may be declared as follows :

```
boolean a;
a = x > y;
```

If the aforementioned logical statement is correct, The value of a is `true`, otherwise the value of a is `false`. In Java `true` and `false` are not converted into numerical values, which is the case in other Languages. A boolean type variable is allocated to one byte, that is, 8 bits for storing its values Program

3.5 illustrates the application of boolean type variables,

```
class Boolean
{
      public static void main (String args[])
      {
      double x= 5.5, y=10.5, p=4.0;
      int n=40, m=50;
      boolean a,b,c,d;

      a= x>y;
      b= y>p;
      c= y==x;
      d= x<=y;

      System.out.println("a = " + a +" and b =" +b);
      System.out.println("Now c= "+c+ " and d = "+d);
      }
}
```

```
C:\>javac Boolean.java

C:\>java Boolean
a = false and b =true
Now c= false and d = true
```

3. Type Casting

Converting one data type to another data type is called as Type Casting. There are two types of type casting. They are,

i. Implicit Type casting
ii. Explicit Type casting

i. Implicit Type casting

Implicit type casting is done automatically by a compiler when we assign a value to the variable.

Example:

```
int a=10;
double b;
b=a;
```

Here a is integer and d is double variables. Integer value is automatically converted into double by the compiler.

ii. Explicit Type casting

The explicit type casting is carried out by the following code:

type variable = (new_type) variable;

- It is illustrated by the following code lines:

```
double D = 6.865;

int A = (int) D;
```

- In such a conversion, there is loss of data

Program 3.7: Illustration of type casting.

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

      int a=4, b = 8, c = 9, d,e;
      double x= 3.0, y=6.5, z,k;

      d=c/a;
      k=a+y;
      e = a + (int)y;
      z=(double)c/a;

      System.out.println("k = " + k +" and e =" +e);
      System.out.println("d= "+ d + " and z  = "+ z);
      }
}
```

Output

```
C:\ >javac TypeCast.java

C:\ >java TypeCast
k = 10.5 and e =10
d= 2 and z  = 2.25
```

4. Scope of Variable Identifier

The scope and lifetime of a variable is the part of the program in which it is visible and holds the last entered value In Java, there are distinctly two types of scopes.
(a) class scope and
(b) method scope

- A variable declared in a class has class scope and scope of a variable declared in a method has method scope.
- The variables declared in a block have block scope.
- Thus, the variables defined in main() at the beginning have scope in entire main() method, however, those defined in a block have block scope.
- A block starts with the left brace ({) and ends with the right brace (}).

In the case of nested blocks of statements, the scope of variables is governed by the following rules.

1. The scope starts from the point the variable is defined in the block (declared and value assigned to it).

2. Although the variable may be defined anywhere in a block, it can be used only in the statements appear ing after its definition Therefore, there is no use in defining a variable at the end of block.

If there are nested blocks, a variable defined in the outer block is visible in the inner blocks also and it cannot be redefined with the same name in the inner blocks

Program 3.10: Illustration of block scope

```
public class ScopeA
{
static int x = 5; // Variable scope within  the Class
public static void main (String args[])
 {
 System.out.println("Class Scope variable - outside main()  x = " + x);
 int y = 10;
 System.out.println("Variable y Scope  within main()  y = " + y);
  { // Anonymous Block
    int z = 20;
    System.out.println("Variable z Scope within Anonymous Block  z = " + z);
  }
 }
}
```

```
E:\>javac ScopeA.java

E:\>java ScopeA
Class Scope variable - outside main()  x = 5
Variable y Scope  within main()  y = 10
Variable z Scope within Anonymous Block  z = 20

E:\>java ScopeA
```

5. Literal Constants, Symbolic Constants

i. Literal Constants

Literal Constants are as follows.

- Each character is a constant value.
- An array of characters or a string also represents a constant value.
- The digits in decimal system, octal system, or hexadecimal system represent constant values.
- The char literals are enclosed in single quotes (‘ ’). The examples ‘A’ and ‘B’. These may also be written as `\u0041` and `\u0042`.
- The string literals are enclosed in double quotes.
  Examples : "Delhi”, "I am going out."
- The decimal integral literals examples are 5417, 684
- The octal number literals are prefixed with 0 (Zero). Example. `072 , 042`
- The floating point literal examples are 684.62f, 5.245E+ 2f
- The literal of Boolean data type are true and false. In Java, neither true is converted to 1 nor

ii. Symbolic Constants

A Symbolic constant is a variable whose value does not change throughout the program. Some of the examples include PL NORTH, EAST etc.

Name of constant	*Value of constant*
Pi	3.145926535....
c (speed of light)	299,792,458 m/s
Gravitational Constant	$6.67300X\ 10^{-11}$ ($m^3\ Kg^{-1}\ s^{-2}$)
Natural Log (e)	2.718281828

It is usually preferred to declare the symbolic constants using all the capital letters in a program as follows:

```
public static final int c =299792458;
public static final double PI = 3.21415;
```

Program 3.13 illustrates the use of symbolic constant

```
public class SymbolicConst
{
  public static final double PI = 3.1415926535;
  public static void main (String args[])
  {
    double r = 25.0, perimeter;
    perimeter = 2*PI*r;
    System.out.println("radius=" + r);
    System.out.println("Perimeter of circle = " + perimeter);
   }
}

C:\>javac SymbolicConst.java

C:\>java SymbolicConst
radius=25.0
Perimeter of circle = 157.079632675
```

6. Formatted Output with printf() Method

In Java, the formatting of output may be carried out in two ways:

1. By using class Formatter 2. By method printf()

- The Formatter class is used to format the output.
- The printf() method is easy and simple to use, and hence, it is mote popular than other methods.
- The following formatting objectives may be realized by using print method
  i. Right and left justification
  ii. Precision of floating point numbers by regulating the number of digits after the decimal point
  iii. Aligning a number of numbers in a column
  iv. Controlling the placement of characters and strings at desired locations
  v. Representing integers in octal and hexadecimal systems
  vi. Representing floating point numbers in exponential form or regular form
  vii. Representing the time and date in different formats.

The syntax of the method printf () method is as follows

```
System.out.printf("Formatting string" variables separated by comma);
```

- The formatting string specifics the output format for each variable that consists of percent (%) sign followed by a conversion letter.
- Thus, the format string for output of an integer and character is "X" and or respectively.
- The order of variables in variable list should match with the list of formats in formatting string.
- The following Table lists the conversion letters for different types of variables.

Table 3.4 Conversion characters for different types of variables

Type variable	*Conversion letter*	*Formatting string*
Integers (base 10)	d	"%d"
Integer (base 8) octal	o	"%o"
Integer (base 16) hexadecimal	X or x	"%X" or "%x"
Character	C or c	"%C" or "%c"
String	S or s	"%S" or "%s"
Floating point number base 10	f	"%f"
Floating point number base 10 exponential form	E or e	"%E" or "%e"
Floating point number base 16	A or a	"%A" or "%a"
Floating point number in exponential (e) or regular format (f) depending upon the magnitude of variable. If the magnitude lies between 10^{-3} and 10^{7}, regular format is used. Outside these limits, it is formatted in exponential format.	G or g	"%G" or "%g"
For providing space between output values, the empty strings or tab "\t" may be used.	" " or \t	Example of code "%d %f %s" or "%d\t%f\t%s"

Program 3.14: illustration of formatting strings for output of different types of variables

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

      int n = 713;
      float x = 45.86f;
      double d= 56.754;

      String str = "Delhi";
      char ch = 'A';
      System.out.printf( "%d   %f  %f  %c \t %s \n", n, x, d,ch,
str);

      //for conversion into hexadecimal number
      System.out.printf("Hexadecimal value of 163 = %X \n", 163);

      //for conversion into octal
      System.out.printf("Octal value of 163 = %o\n", 163);
      }
}
```

```
Output

C:\>javac FormatPrintf.java

C:\>java FormatPrintf
713   45.860001  56.754000  A    Delhi
Hexadecimal value of 163 = A3
Octal value of 163 = 243
```

7. Static Variables and Methods

Static Variables:

- **The static variables are class variables**. **Only one copy of such variables is kept in the memory and all the objects share that copy**.
- **The static variables are accessed through class reference**, whereas the instance variables are accessed through class object reference
- The variables in a class may be modified by modifier static.
- The non-static variables declared in a class are instance variables Each object of the class keeps a copy of the values of these variables.

Static Methods:

- The static methods are similar to class methods and can be invoked without any reference of object of class, however, class reference (name of class) is needed, as in the following example The method like sart() is declared as static method in Math class and is called

```
Math.sqrt(5); // Finds square root of 5
```

The static method is called using the method name that is preceded by the class name; in this case. Math and period ().

Program 3.18: illustration of using static methods of class Math

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

 System.out.println("The Square root root of 16 = "+ Math.sqrt(16));
 System.out.println("The cubroot root of 27 = "+ Math.cbrt(27));
 //printing five random variables
 for(int i =1; i<=5;i++)
     System.out.println("Random Number "  + i + " = " +
                        (int)(100 *Math.random()));
     }
}
```

```
E:\>javac StaticMethods.java

E:\>java StaticMethods
The Square  root root of 16 = 4.0
The cubroot root of 27 = 3.0
Random Number 1 = 77
Random Number 2 = 69
Random Number 3 = 83
Random Number 4 = 2
Random Number 5 = 66

E:\>java StaticMethods
The Square root root of 16 = 4.0
The cubroot root of 27 = 3.0
Random Number 1 = 67
Random Number 2 = 31
Random Number 3 = 10
Random Number 4 = 13
Random Number 5 = 40
```

8. Attribute Final

Final Variable:

The value of a variable declared final cannot be changed in the program. It makes the variable a constant. A few examples of declarations are as follows:

```
final double PI =  3.14159; // The value of PI cannot be changed in its scope

final int M = 900;  // The value of M cannot be changed in its scope

final double X = 7.5643; // The value of x cannot be changed in its scope.
```

- As mentioned in the comments, the values of PI, M, and x cannot be changed in their respective scopes.

Final Method:

- The attribute final may be used for **methods** as well as for classes. These are basically connected with inheritance of classes.
- When final keyword is used with Java method, it becomes the final method.
- A final method cannot be overridden in a sub-class.

Final Class:

- A Java class with final modifier is called final class A final class cannot be sub-classed or inherited. Several classes in Java are final including String, Integer, and other wrapper classes.

- There are certain important points to be noted when using final keyword in Java
  i. New value cannot be reassigned to a variable defined as final in Java.
  ii. Final keyword can be applied to a member variable, local variable, method, or class.
  iii. Final member variable must be initialized at the time of declaration.
  iv. Final method cannot be overridden in Java
  v. Final class cannot be inheritable in Java
  vi. Final is different from finally keyword, which is used on Exception handling in Java

Example- 1:

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

	int n =10; // Normal variable
	final int f = 20; // final variable

	System.out.println("n = "+ n);
	System.out.println("f = "+ f);

	n = 50; // Now the value 50 is assigned to variable n
	f = 60; // Error : f is final variable can not be changed
	}
}
```

Output:

```
C:\>javac Final.java
Final.java:13: error: cannot assign a value to final variable f
        f = 60; // Error : f is final variable can not be changed
        ^
1 error
C:\>
```

Example-2:

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

	int n =10; // Normal variable
	final int f = 20; // final variable

	System.out.println("n = "+ n);
	System.out.println("f = "+ f);

	n = 50; // Now the value 50 is assigned to variable n
	System.out.println("n = "+ n);
	}
}
```

```
C:\>javac Final.java

C:\>java Final
n = 10
f = 20
n = 50
```

9. Introduction to Operators

An operator is a symbol that tells the computer to perform certain mathematical and logical calculations.

-The different types of Java operators are,

a) Arithmetic operators
b) Relational operators
c) Logical operators
d) Increment or decrement operators
e) Assignment operators
f) Conditional operators
g) Bitwise operators
h) Special operators

10.Precedence and Associativity of Operators

If the expression contains several operators with the same precedence level, the expression is evaluated according to its *associativity*.

For example, in

Z = 6*4%5;

Both the operators * and % have same precedence level.

Highest						
++ (postfix)	– – (postfix)					
++ (prefix)	– – (prefix)	~	!	+ (unary)	– (unary)	(*type-cast*)
*	/	%				
+	–					
>>	>>>	<<				
>	>=	<	<=	instanceof		
==	!=					
&						
^						
|						
&&						
||						
?:						
->						
=	op=					
Lowest						

11. Assignment Operator (=)

Assignment operators:

-Assignment operators are used to assign the result of an expression to a variable.

Operator	Meaning
=	Assignment

-In addition, java has a set of short-hand assignment operators of the form

V OP=EXP;

- It is equivalent to

V=V OP EXP;

-The short-hand assignment operators are

+= , -= , *= , /= , %=

Example:

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

12. Basic Arithmetic Operators

Operator	Meaning
+	Addition or Unary plus
-	Subtraction or Unary minus
*	Multiplication
/	Division
%	Modulo division

-Integer division truncates any fractional part.

-The modulo division produces the remainder of an integer division.

Examples: a+b, a-b, a*b, a/b, a%b

Arithmetic Expression Types:

Integer arithmetic expression:

-An arithmetic operation involving only integer operands is called integer arithmetic.

int + int = int
int - int = int
int * int = int
int / int = int (truncates decimal part)
int % int = int (results remainder)

Real arithmetic expression:

- An arithmetic operation involving only real operands is called real arithmetic.

real + real = real
real - real = real
real * real = real
real / real = real
real % real = not allowed

Mixed-mode arithmetic expression:

-An arithmetic operation involving one operand is real and the other operand is integer, then it is called mixed-mode arithmetic.

real+int=real
real-int=real

real*int=real
real/int=real
real%int=not allowed

13. Increment (++) and Decrement (- -) Operators

-The increment operator is ++ which means +1
-The decrement operator is - - which means -1
-The operand must be either incremented or decremented by 1.
Example:
m=5;y=++m; results m=6 and y=6.
m=5;y=m++; results m=6 and y=5.
m=5;y= --m; results m=4 and y=4.
m=5;y=m--; results m=4 and y=5.

14. Ternary Operator or Conditional operators:

-It is also known as Ternary operator.
-The symbols used to construct a conditional expression are ? and **:**
-A conditional expression is of the form,

exp1?exp2:exp3;

Example:
(a>b)?System.out.println("a is greater")**:** System.out.println ("b is greater");
- It is equivalent to if-else statement in java

Example:

```
if(a>b)
   System.out.println("a is greater");
else
   System.out.println ("b is greater");
```

15. Relational Operators

-The comparison between two operands or expressions is done with the help of relational operators.

Operator	Meaning
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
==	Equality
!=	Inequality

Example:

```
a>b
a>=b
a<b
a<=b
a==b
a!=b
```

16.Boolean Logical Operators

-The java language has three logical operators.

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

-The logical operators &&, || are used when we we want to test more than one condition.
&& - used when all the conditions must be true.
|| - used when any of the conditions must be true.

-A logical expression yield a value 0 or 1, according to the truth table.

Operand1	Operand2	Operand1&&operand2	Operand1||operand2
Non-zero	Non-zero	1	1
Non-zero	0	0	1
0	Non-zero	0	1
0	0	0	0

Example:
(a>b)&&(a>c)
(a>b)||(a>c)
-The logical not is used as a negation or complement of the expression.
Example:
!(y<10) means (y>=10)

17.Bitwise Logical Operators.

Bitwise operators:

-'C' provides bitwise operators that operate on data at the bit-level.
-Bitwise operators interpret operands as string of bits.
-These bit strings are then interpreted according to data type.

-Bitwise operators are of 2 types.
i) Logical bitwise operators.
ii) Shift bitwise operators.

i) Bitwise Logical operators

Operator	Meaning
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR
~	One's complement

Bitwise AND operator:

-The Bitwise AND (&) is a binary operator that requires two integral operands(character or integer).
-It does a bit-by-bit comparison between two operands.
-The result of the comparison is 1 only when both bits are 1, otherwise it is 0.

First operand bit	Second operand bit	Result(&)
0	0	0
0	1	0
1	0	0
1	1	1

Bitwise OR operator:

-The Bitwise inclusive OR (|) is a binary operator that requires two integral operands(character or integer).
-It does a bit-by-bit comparison between two operands.
-The result of the comparison is 0 only when both bits are 0, otherwise it is 1.

First operand bit	Second operand bit	Result(|)
0	0	0
0	1	1
1	0	1
1	1	1

Bitwise exclusive OR:

-The Bitwise exclusive OR (^)is a binary operator that requires two integral operands(character or integer).
-It does a bit-by-bit comparison between two operands.
-The result of the comparison is 1 only if one of the operands is 1, otherwise it is 0.

First operand bit	Second operand bit	Result(^)
0	0	0
0	1	1
1	0	1
1	1	0

One's complement:

-The one's complement (~) is a unary operator applied to an integral value.
-The result is 1 when the original bit is 0 and it is 0 when the original bit is 1.

Original bit	Result(~)
0	1
1	0

ii) Shift Bitwise operators:

-The shift Bitwise operators move bits to the right or left.

Operator	Meaning
>>	Bitwise shift right
<<	Bitwise shift left
>>>	Shift right with zero fill

Bitwise shift-right operator:

-It moves some number of bits from right to left.
-It requires two integral operands.

Syntax:

Operand1 >> Operand2;

-Here, Operand1 is value to be shifted.
Operand2 is number of bits to be shifted.

Bitwise shift-left operator:

-It moves some number of bits from left to right.
-It requires two integral operands.

Syntax:

Operand1 << Operand2;

-Here, Operand1 is value to be shifted.
Operand2 is number of bits to be shifted.

18.Special Operators.

- Java supports some special operators such as,
    a) instanceof operator
    b) member selection operator

a) instanceof operator:

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

Example:

    person instanceof student

- It is true if the object person belongs to the class student, otherwise it is false.

b) member selection operator:

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

Example:

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

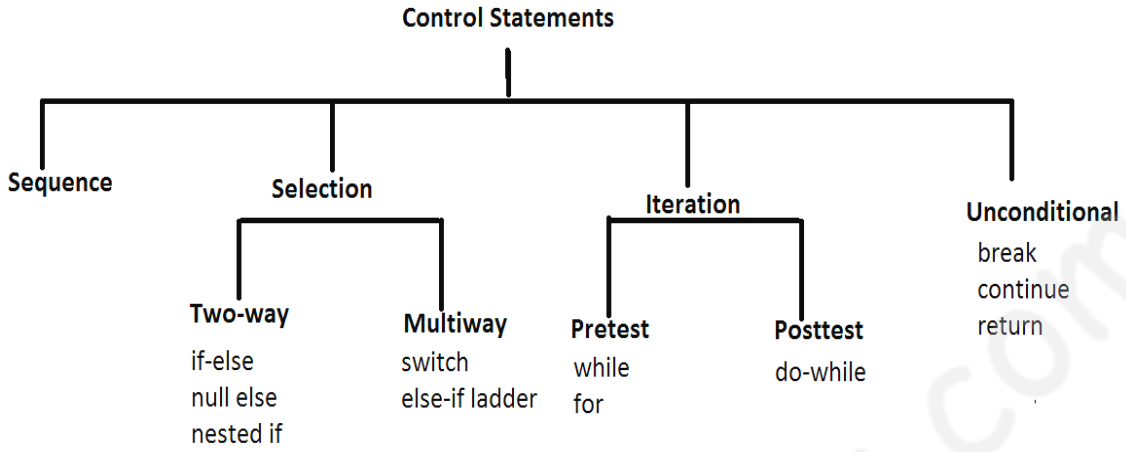
- It is also used to access classes and sub packages from a package.

III. Control Statements:

1. Introduction

CONTROL STATEMENTS: (FLOW OF CONTROL)

- A Control statement is a statement used to control the flow of execution in a Java Program.



SELECTION STATEMENTS:

-Also called as conditional or decision-making control statements.

-There are two types in Selection control statements.

i) Two-way selection control statements

ii) Multi-way selection control statements

i) Two-way selection control statements:

-The different two-way selection statements are,

a) if-else statement

b) null else statement

c) Nested-if statement

2. if Expression - also called as " null else statement "

Syntax:

```
if(condition)
   {
      statements;
   }
next statement;
```

Example:

```
if(a==2)
{
     p++;
 }
System.out.println("program over");
```

3. Nested if Expressions

-if within if is called as Nested-if.

Syntax:

```
if(condition-1)
{
      if(condition-2)
      {
            Statement-1;
      }
      else
      {
            Statement-2;
      }
}
else
{
      if(condition-3)
      {
            Statement-3;
       }
      else
      {
            Statement-4;
      }
}
next statement;
```

Example:

```
if(a>b)
   {
      if(a>c)
      {
            System.out.println("a is greater");
      }
      else
      {
        System.out.println ("c is greater");
      }
   }
else
    {
      if(b>c)
      {
            System.out.println("b is greater");
      }
      else
      {
            System.out.println("c is greater");
      }
   }
```

4. if–else Expressions

Syntax:

```
if(condition)
   {
        true-block statements;
   }
else
   {
        false-block statements;
   }
next statement;
```

Example:

```
if(a>b)
   {
        System.out.println("a is greater");
   }
else
   {
        System.out.println("b is greater");
   }
```

b) else-if ladder statement:

Syntax:

```
if(condition-1)
   {
     Statement-1;
   }
else if(condition-2)
   {
     Statement-2;
   }
……..……
……..……..
else if(condition n-1)
   {
     Statement-(n-1);
   }
else
   {
     Statement-n;
   }
next statement;
```

Example:

```
if(a>b&&a>c&&a>d)
   {
     System.out.println("a is greater");
   }
else if(b>a&&b>c&&b>d)
   {
     System.out.println("b is greater");
   }
else if(c>a&&c>b&&c>d)
   {
     System.out.println("c is greater");
   }
else
   {
     System.out.println("d is greater");
   }
```

5. Ternary Operator?:

In Java, the **ternary operator** is a type of Java conditional operator. The meaning of **ternary** is composed of three parts.

The **ternary operator (? :)** consists of three operands. It is used to evaluate Boolean expressions. The operator decides which value will be assigned to the variable. It is the only conditional operator that accepts three operands.

It can be used instead of the if-else statement. It makes the code much more easy, readable, and shorter.

Syntax:

Expression1 ? Expression2 : Expression3

- The first expression is the test condition.
- If it evaluates true, the Expression2 is executed; otherwise Expession3 is executed.

Example-1:

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

           int a=10;
           int b = (a<20)? 100 :200; // a < 20  if statement
           System.out.println("b= "+b);
     }
}
```

```
Output:
C:\>javac Ternary.java

C:\>java Ternary
b= 100
```
Example-2:

```
public class Ternary
{
      public static void main (String args[])
      {
            int a=10;
            int b = (a>20)? 100 :200; // a>20 – else statement
            System.out.println("b= "+b);
      }
}

Output:
C:\>javac Ternary.java

C:\>java Ternary
b= 200
```

6. Switch Statement

Syntax:

```
switch(expression)
   {
case value-1:statement-1;break;
case value-2:statement-2;break;
……..
……..
case value-n:statement-n;break;
default: default statement;
   }
next statement;
```

Example:

```
switch(digit)
   {
case 0: System.out.println(“ZERO”);break;
case 1: System.out.println(“ONE”);break;
case 2: System.out.println(“TWO”);break;
case 3: System.out.println(“THREE”);break;
case 4: System.out.println(“FOUR”);break;
case 5: System.out.println(“FIVE”);break;
case 6: System.out.println(“SIX”);break;
case 7: System.out.println(“SEVEN”);break;
case 8: System.out.println(“EIGHT”);break;
case 9: System.out.println(“NINE”);break;
default: System.out.println(“Enter between 0-9”);
}
```

7. Iteration Statements

LOOP STATEMENTS:

-The iteration control statements are also called as Repetition or Iteration control statements.
-A looping process includes the following four steps.

-Setting and initialization of a counter.
-Execution of the statements in the loop body.
-Test for a specified condition (loop control expression) for execution of a loop
-Incrementing or Decrementing counter.

i) Pretest and Posttest loops:

-In a **Pretest loop**, the condition is checked before we execute a loop body.
-It is also called as **entry-controlled loop**.
-In the **Posttest loop**, we always execute the loop body atleast once.
-It is also called as **exit-controlled loop**.

8. while Expression

a) while statement:

Syntax:

```
while(condition)
  {
loop body;
  }
next statement;
```

Example:

```
n=10,i=1,sum=0;
while(i<=n)
   {
sum=sum+i;
i++;
        }
System.out.println("sum="+sum);
```

9. do–while Loop - forLoop

Syntax:

```
do
   {
      loop body;
   }while(condition);
next statement;
```

Example:

```
n=10,i=1,sum=0;
do
   {
sum=sum+i;
i++;
   } while(i<=n);
System.out.println("sum="+sum);
```

10. forLoop

for statement:

Syntax:

```
for(initialization;condition;inc or dec)
  {
      loop body;
  }
next statement;
```

Example:

```
n=10,i,sum=0;
for(i=1;i<=n;i++)
{
      sum=sum+i;
}
System.out.println("sum="+sum);
```

11.Nested for Loop

Syntax:

```
for(initialization;condition;inc or dec)
   {

   for(initialization;condition;inc or dec)
   {
        Inner loop body;
   }

   Outer loop body;
   }
next statement;
```

Example:

```
   n=10,i, j, sum=0;
   for(i=1;i<=n;i++)
   {

        sum=sum+i;
   }
   System.out.println("sum="+sum);
```

12.For–Each for Loop

The Java for-each loop or enhanced for loop. It provides an alternative approach to traverse the array or collection in Java. It is mainly used to traverse the array or collection elements. The advantage of the for-each loop is that it eliminates the possibility of bugs and makes the code more readable. It is known as the for-each loop because it traverses each element one by one.

***Advantages*:**

1. Less clutter in code, especially when iterators are used.
2. Less chances of errors.
3. Improves overall readability of program.

***Limitations*:**

1. It is designed to iterate in forward direction only.
2. In iteration, it takes a single step at a time.
3. It cannot simultaneously traverse multiple arrays or collections.

Syntax:

```
for (Object obj : Collection_name)
{
   Body of loop
}
```

Example:

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

      int myArray[] = {10,20,30,40,50};

      for (int x:myArray)
       {
            System.out.println(" "+x);
       }
      }
}
```

Output:

```
   C:\>javac ForEach.java

   C:\>java ForEach
    10
    20
    30
    40
    50
```

13.Break Statement

Unconditional control statements:

-The unconditional control statements are,

a) break statement
b) continue statement

break statement:

-The break statement skips from the loop or block in which it is defined.
-The control then automatically goes to the first statement after the loop or block.
-The general format is

break;

Example:

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

   //printing the values from 1 to 10
   for(int i =0; i<10;i++)
      {
       if (i==5)
         break;  //Break Statement
       System.out.println( i );
      }
   }
}
```

Output:

```
C:\>javac Break.java

C:\>java Break
0
1
2
3
4
```

Here loop is stopped due to break statement.

14.Continue Statement.

-The continue statement is used for continuing next iteration of loop statements.
-When it occurs in the loop, it does not terminate but it skips the statements after it.
-It is useful when we want to continue the program without executing any part of the program.
-The general format is

continue;

Example:

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

      //printing the values from 1 to 10
      for(int i =0; i<10;i++)
         {
          if (i==5)
            continue; // Continue statement
          System.out.println( i );
         }
      }
}
Output:
-------
C:\>javac Break.java

C:\>java Continue
0
1
2
3
4
6
7
8
9
```

Here 5 is not printed because of continue statement. The Iteration at the condition `i = =5` is skipped or jumped to next statement.