Unit-1 Fundamentals Of Object Oriented Programming

Basic Concept of Object-Oriented Programming :

❖ Object & Class :

- ✓ Objects are the basic runtime entities in an object-oriented system.
- ✓ When a program is executed, the objects interact by sending messages to one another.
- ✓ **Example:** 'Customer' and 'Account' are two objects in a banking program, then the customer object may send a message to the account object requesting for the balance.
- ✓ Each objects contains data and code to manipulate the data.
- ✓ Once a class has been defined, we can create any number of objects belonging to that class.
- ✓ Each object is associated is associated with the data of type class with which they are created.
- ✓ Classes are user-defined data types and behave like the built-in types of a programming language.
- ✓ Class is a collection of data and methods.
- ✓ The syntax used to create an object is no different than the syntax used to create an integer object in C.
- ✓ If **fruit** has been defined as a class, then the statement

  fruit mango *

  will create an object **mango** belonging to the class **fruit**.

❖ Data Abstraction and Encapsulation :

- ✓ The wrapping up of data and methods into a single unit (called class) is known as encapsulation.
- ✓ Data encapsulation is the most striking feature of a class.
- ✓ The data is not accessible to the outside world and only those methods, which are wrapped in the class, can access it.
- ✓ This insulation of the data from direct access by the program is called data hiding.

❖ Inheritance :

- ✓ Inheritance is the process by which objects of one class acquire the properties of objects of another class.
- ✓ Inheritance supports the concept of hierarchical classification.
- ✓ **Example :** The bird Robin is a part of the class flying bird, which is again a part of the class bird.

- ✓ Deriving a new class from the existing class is also known as inheritance.
- ✓ The new class will have the combined features of both the classes.

❖ Polymorphism :

- ✓ Polymorphism is another important OOP concept.
- ✓ Polymorphism means the ability to take more than one form.
- ✓ **Example :** Consider the operation of addition. For two numbers, the operation will generate sum. If the operands are strings, then the operation would produce a third string by concatenation.

❖ Dynamic Binding :

- ✓ Dynamic binding that the code associated with a given procedure call is not known until the time of the call at runtime.

❖ Message Communication :

- ✓ An object-oriented program consists of a set of objects that communicate with each other.
- ✓ Basic Steps:
    1. Creating classes that define objects.
    2. Creating objects from class definitions.
    3. Establishing communication among objects.
- ✓ Objects communicate with one another by sending and receiving information much the same way as people pass messages to one another as shown in under figure.

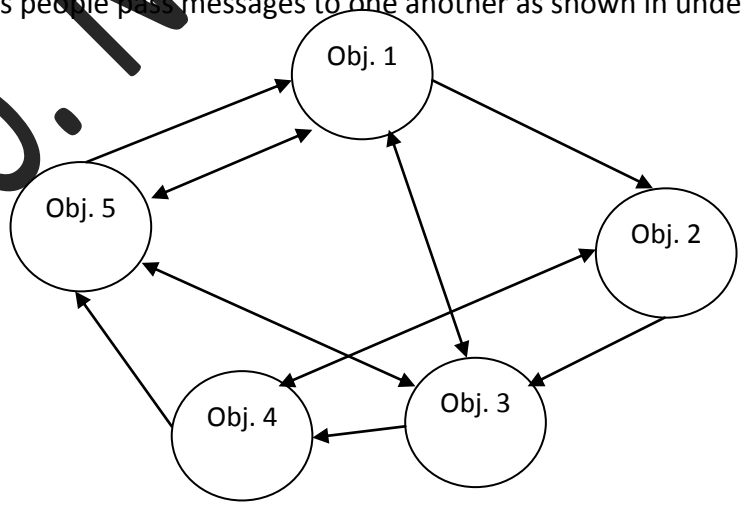


Fig. : Network of Objects communicating between them

❖ Benefits of OOP :

- ✓ OOP offers several benefits to both the program designer and the user.
- ✓ The principal advantages are:
  1. Through Inheritance, we can eliminate redundant code and extend the use of existing classes.
  2. The principle of data hiding helps the programmer to build secure programs.
  3. It is possible to have multiple objects to communicate without any interference.
  4. It is easy to partition the work in a project based on objects.
  5. Object-oriented system can be easily upgraded from small to large systems.

❖ Applications of OOP :

- ✓ There are hundreds of windowing systems developed using OOP techniques.
- ✓ The application of OOP includes:
  1. Real-time systems
  2. Object-oriented databases
  3. Hypertext, hypermedia and expertext
  4. AI and expert systems
  5. Neural networks and parallel programming
  6. Decision support and office automation systems

Basics of Java :

❖ History of Java :

- ✓ Java is an object oriented programming language developed by Sun Microsystems of USA in 1991.
- ✓ Java was designed for the development of software for consumer electronic devices like TVs, VCRs, toasters and such other electronic machines.
- ✓ The Java team modeled their new language Java on C and C++ but removed a number of features of C and C++ and thus made Java a really simple, reliable, portable and powerful language.
- ✓ The most striking feature of the language is that it is a platform-neutral language.
- ✓ Java is the first programming language that is not tied to any particular hardware or operating system.
- ✓ Programs developed in Java can be executed anywhere on any system.

❖ Java Features :

- ✓ The inventors of Java wanted to design a language to some of the problems encountered in modern programming.
- ✓ They wanted the language to be not only reliable, portable and distributed but also simple, compact and interactive.
- ✓ Various features of Java programming language are as given below:
    1. Java is very simple programming language. Even though you have no programming background.
    2. Java is popular because it is an object oriented programming language like C++.
    3. Platform Independence is the most exciting feature of Java. That means programs in java can be executed on variety of systems. This feature is based on the goal "Write once, run anywhere and at anytime, forever".
    4. Java supports multithreaded programming which allows a programmer to write such a program that can be perform many tasks simultaneously.
    5. Thus robustness is the essential features for the java programs.
    6. Java is designed for distributed systems. Hence two different objects on different computer can communicate with each other.

❖ JDK and its Components (Various Tools of JDK) :

- ✓ Java environment includes a large number of development tools and hundreds of classes and methods.
- ✓ The development tools are part of the system known as Java Development Kit(JDK) and the classes and methods are part of the Java Standard Library(JSL), also known as the Application Programming Interface(API).
- ✓ The Java Development Kit comes with a collection of tools that are used for developing and running Java programs. They include:
    - appletviewer (for viewing Java applets)
    - javac (Java compiler)
    - java (Java interpreter)
    - javap (Java disassembler)
    - javah (for C header files)
    - javadoc (for creating HTML documents)
    - jdb (Java debugger)

Tool	Description
appletviewer	Enables us to run Java applets (without actually using a Java-compatible browser).
Java	The java compiler, which runs applets and applications by reading and interpreting bytecode files.
Javac	The Java compiler, which translates Java sourcecode to btecode files that the interpretercan understand.
Javadoc	Creates HTML-format documentation from Java source code files.
Javah	Produces header files for use with native methods.
Javap	Java disassemble, which enables us to convert bytecode files into a program description.
Jdb	Java debugger, which helps us to find errors in our programs.

[Table : Java Development Tools]

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

❖ Byte Code and JVM :

- ✓ Bytecode is an intermediate form of java programs.
- ✓ Bytecode consists of optimized set of instructions.
- ✓ We get bytecode after compiling the java program using a compiler called javac.
- ✓ The bytecode is to be executed by java runtime environment which is called as Java Virtual Machine (JVM).
- ✓ The programs that are running on JVM must be compiled into a binary format which is denoted by .class files.

❖ Java Program Structure :

- ✓ Any java program can be written using simple text editor like notepad or WordPad.
- ✓ The file name should be same as the name of the class used in the corresponding java program.
- ✓ The extension of the file name should be .java.

```
FirstProgram.java
/*
This is my First Java Program
*/
class FirstProgram
{
        public static void main(String args[])
        {
                System.out.println("This is first program");
        }
}
```

➢ **Explanation:-**

- ✓ In our First Java program, on the first line we have written a comment statement.
- ✓ This is a multi-line comment.
- ✓ Then actual program starts with class FirstProgram.
- ✓ Here class is keyword and FirstProgram is a variable name of the class.
- ✓ The class definition should be within the curly brackets.
- ✓ Then comes
  public static void main(String args[])
- ✓ This line is for function void main().
- ✓ The main is of type public static void.
- ✓ Public is a access mode of the main() by which the class visibility can be defined.
- ✓ Typically main must be declared as public.
- ✓ The parameter which is passed to main is String args[].
- ✓ Hence String is a class name and args[] is an array which receives the command line arguments when the program runs,
  System.out.println("This is first program");
- ✓ To print any message on the console println is a method in which the string "This is first program" is written.
- ✓ After println method, the newline is invoked.
- ✓ Java program is a case sensitive programming language like C or C++.

Building Blocks :

❖ Tokens :

- ✓ Smallest individual units in a program are known as tokens.
- ✓ A Java program is a collection of tokens, comments and white spaces. Java language includes five types of tokens. They are:

- Reserved Keywords
- Identifiers
- Literals
- Operators
- Separators

➢ Keywords:

✓ Keywords are an essential part of a language definition.
✓ They implement specific features of the language.
✓ Java language has reserved 50 words as keywords.
✓ Understanding the meanings of all these words is important for Java programmers.
✓ Keywords have specific meaning in Java, we can not use them as names for variables, classes, methods and so on.
✓ All keywords are to be written in lower-case letters.
✓ Java is case-sensitive, one can use these words as identifiers by changing one or more letters to upper case.
✓ **Example:** abstract, class, do, for, import, switch, case, static, while, catch, else, if, etc…

➢ Identifiers:

✓ Identifiers are used for naming classes, methods, variables, objects, labels, packages and interfaces in a program.
✓ Java identifiers follow the following rules:
1. They can have alphabets, digits, and underscore and dollar sign characters.
2. They must not begin with a digit.
3. Uppercase and lowercase letters are distinct.
4. They can be any length.
✓ Identifiers must be meaningful, short enough to be quickly and easily typed and easily read.

➢ Literals:

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

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

➢ Operators:

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

➢ Separators:

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

Name	What is used for
parentheses( )	Used to enclose parameters in method definitions.
braces{ }	Used to contain the values of automatically initialized arrays and to define a block of code for classes, methods and local scopes.
brackets[ ]	Used to declare array types.
semicolon ;	Used to separate statements.
comma ,	Used to separate identifiers in a variable declaration.
period .	Used to separate package names from sub-packages and classes.

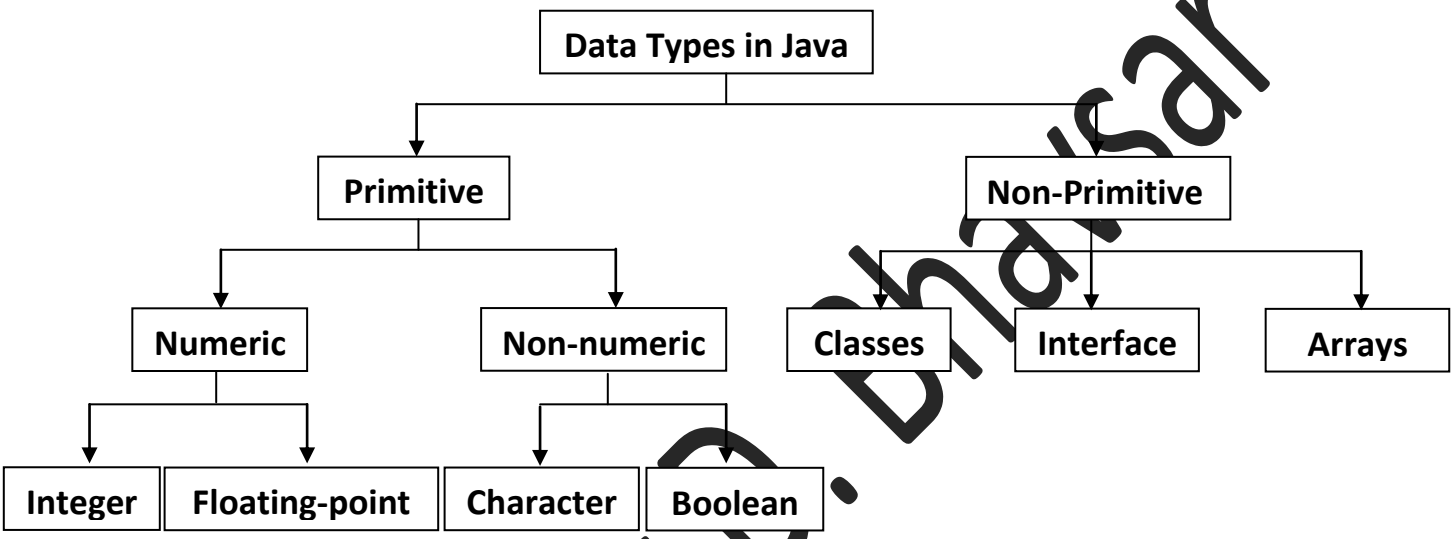
[Java Separators]

❖ Variables :

- ✓ A variable is an identifier that denotes a storage location used to store a data value.
- ✓ A variable may take different values at different times during the execution of the program.
- ✓ A variable name can be chosen by the programmer in a meaningful way.
- ✓ Some examples of variable names are:
  - average
  - height
  - total_height
  - classStrength
- ✓ Variable names may consist of alphabets, digits, the underscore(_) and dollar character, subject to the following conditions:
  1. They must not begin with a digit.
  2. Uppercase and lowercase are distinct. This means that the variable Total is not the same as total or TOTAL.
  3. It should not be a keyword.
  4. White space is not allowed.
  5. Variable names can be of any length.

❖ Data Types :

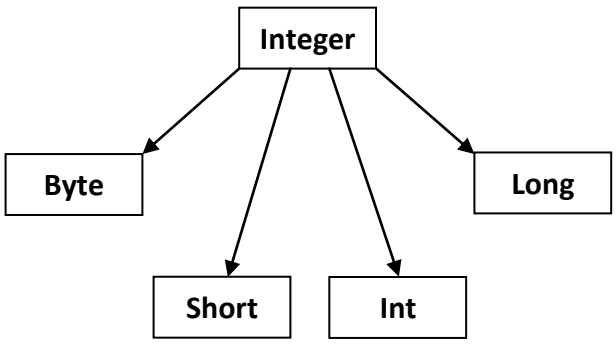
- ✓ Every variable in Java has a data type.
- ✓ Data types specify the size and type of values that can be stored.
- ✓ Java language is rich in its data types.
- ✓ The variety of data types available allows the programmer to select the type appropriate to the needs of the application.



[Fig. Data types in Java]

- **Integer Types :-**

- ✓ Integer types can hold whole numbers such as 123, -96 and 5639.
- ✓ The size of the values that can be stored depends on the integer data type we choose.
- ✓ Java supports four types of integers. They are byte, short, int, and long.



[Integer data types]

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

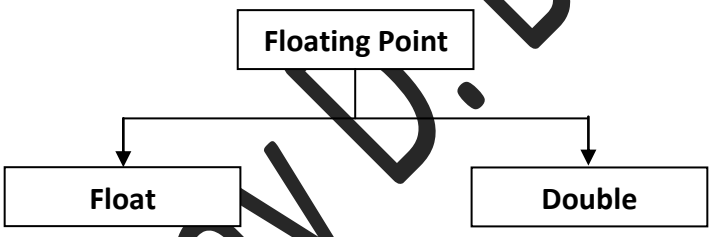
[Size and Range of Integer Types]

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

- **Floating Point Types :-**

✓ Floating point types can hold numbers containing fractional parts such as 27.59 and -1.375(known as floating point constants).

✓ There are two kinds of floating point storage in Java.



[floating point data types]

Type	Size	Minimum Value	Maximum Value
float	4 bytes	3.4e-038	1.7e+0.38
Double	8 bytes	3.4e-038	1.7e+308

[Size and Range of Integer Types]

- **Character Type :-**

✓ In order to store character constants in memory, Java provides a character data type called char.

✓ The char type assumes a size of 2 bytes but, basically, it hold only a single character.

- **Boolean Type :-**

✓ Boolean type is used when we want to test a particular condition during the execution of the program.
✓ There are only two values that a Boolean type can take: true or false.
✓ Boolean type is denoted by the keyword Boolean and uses only one bit storage.
✓ The words true and false cannot be used as identifiers.

❖ Declaration of Variables :

✓ In Java, variables are the names of storage locations.
✓ A variable must be declared before it is used in the program.
✓ A variable can be used to store a value of any data type.
✓ The declaration statement defines the type of variable.
✓ The general form of declaration of a variable is:

type variable1, variable2, ……., variableN;

✓ Variables are separated by commas.
✓ A declaration statement must end with a semicolon.
✓ **Example :**

```
int count;
float x, y;
double pi;
byte b;
char c1, c2, c3;
```

❖ Giving Values To Variables :

✓ A variable must be given a value after it has been declared but before it is used in an expression.
✓ This can be done in two ways:

1. By using an assignment statement
2. By using a read statement

1. Assignment Statement :

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

```
VariableName = Value;
```

✓ **Example :**

```
i=0;
f=100;
y='x';
```

✓ We can also string assignment expressions as shown below::

```
x=y=z=0;
```

- ✓ It is also possible to assign a value to a variable at the time of its declaration.

```
type variableName = value;
```

- ✓ **Example :**

```
int f=100;
char y='x';
double total=75.36;
```

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

2. Read Statement

- ✓ We may also gives values to variables through the keyboard using the readLine() method.
- ✓ **Example :**

```
import java.io.DataInoutStream;
class Reading
{
        public static void main(String args [])
        {
                DataInputStream in = new DataInputStream(System.in);
                int a=0;
                float b=0.0f;
                try
                {
                        System.out.println("Enter The Value Of A :");
                        a=Integer.parseInt(in.readLine());
                        System.out.println("Enter The Value Of B :");
                        b=Float.valueof(in.readLine()).floatValue();
                }
                catch(Exception e) {}

                System.out.println("A :" + a);
                System.out.println("B :" + b);
        }
}
```

- ✓ The readLine() method reads the input from the keyboard as a string which is then converted to the corresponding data type using the data type wrapper class.
- ✓ Note that we have used the keywords **try** and **catch** to handle any errors that might occur during the reading process.

❖ Type Casting :

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

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

- ✓ The process of converting one data type to another is called casting.
- ✓ **Example :**

```
int m=50;
byte n=(byte)m;
long count=(long)m;
```

- ✓ Casting to smaller type can result in a loss of data.
- ✓ Casting a floating point value to an integer will result in a lost of the fractional part.

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

[Casts that Results in No Loss of Information]

➢ Automatic Conversion :

- ✓ For some types, it is possible to assign a value of one type to a variable of a different type without a cast.
- ✓ Java does the conversion of the assigned value automatically. This is known as automatic type conversion.
- ✓ Automatic type conversion is possible only if the destination type has enough precession to store the source value.
- ✓ **Example :**

int is large enough to hold a byte value.

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

are valid statements.

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

❖ Getting Values of Variables :

- ✓ Java supports two output methods that can be used to send the results to the screen.

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

- ✓ The print() method sends information into a buffer.
- ✓ This buffer is not flushed until a newline character is sent.
- ✓ print() method prints output on one line until a newline character is encountered.
- ✓ **Example :**

```
System.out.print("Hello");
System.out.print("Java…");
```

Will display the words **Hello Java…** on one line and waits for displaying further information on the same line.

- ✓ We can display the next line by printing a newline character as shown below:

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

Will display the output in two lines as follows:

```
Hello
Java…
```

- ✓ The println() method takes the information provided and displays it on a line.
- ✓ **Example :**

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

Will produce the following output:

```
Hello
Java…
```

❖ Command Line Arguments :

- ✓ Command line arguments are parameters that are supplied to the application program at the time of invoking it for execution.
- ✓ We can write Java programs that can receive and use the arguments provided in the command line.

  public static void main(String args[])
- ✓ **args** is declared as an array of strings (known as string objects).
- ✓ Any arguments provided in the command line (at the time of execution) are passed to the array **args** as its elements.
- ✓ **Example :**

  Java Test BASIC FORTAN C++ Java
- ✓ This command line contains four arguments.
- ✓ These are assigned to the array **args[]** as follows:

BASIC	->	args[0]
FORTAN	->	args[1]
C++	->	args[2]
Java	->	args[3]

- ✓ The individual elements of an array are accessed by using an index like args[i].
- ✓ The value of I denotes the position of the elements inside the array.
- ✓ **Args[2]** denotes the third elements and represents C++.
- ✓ Not that Java index begin with 0 and not 1.

Control Statements :

❖ Selection Statement:

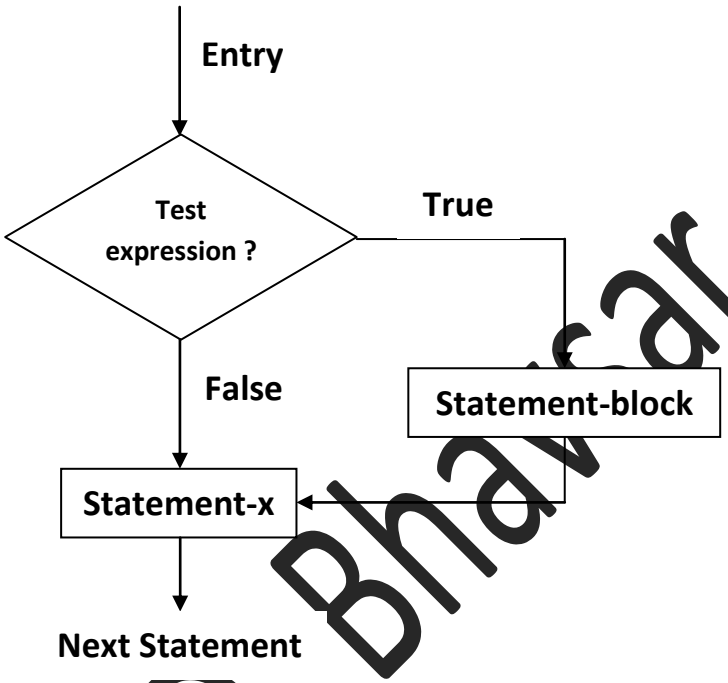
➢ Simple If Statement :

- ✓ The general form of a simple if statement is

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

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

- ✓ Note that when the condition is true both the statement-block and the statement-x are executed in sequence.



[Flowchart of Simple if control]

- ✓ **Example :**

```
if (a>b)
        System.out.println("a is Greater…");
```

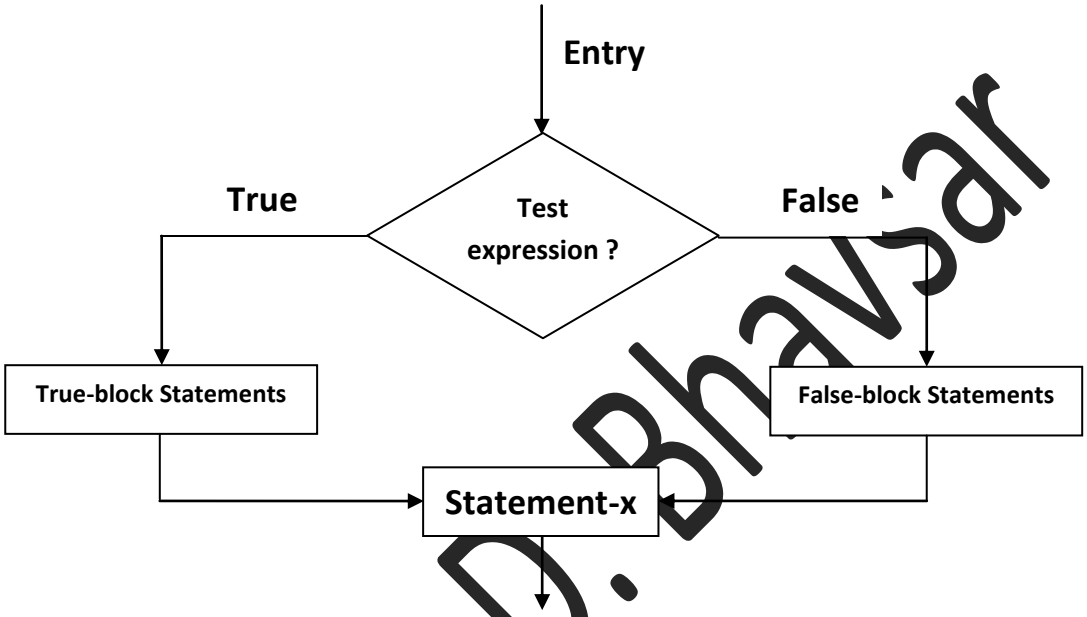
➢ The If…Else Statement :

- ✓ The if…else statement is an extension of the simple if statement.
- ✓ The general form is:

```
if(test expression)
{
        True-block statements
{
else
{
        False-block statements
```

```
}

Statement-x;
```

- ✓ If the test expression is true, then the true-block statements are executed, otherwise the false-block statements are executed.



[Flowchart of if…else control]

- ✓ **Example :**

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

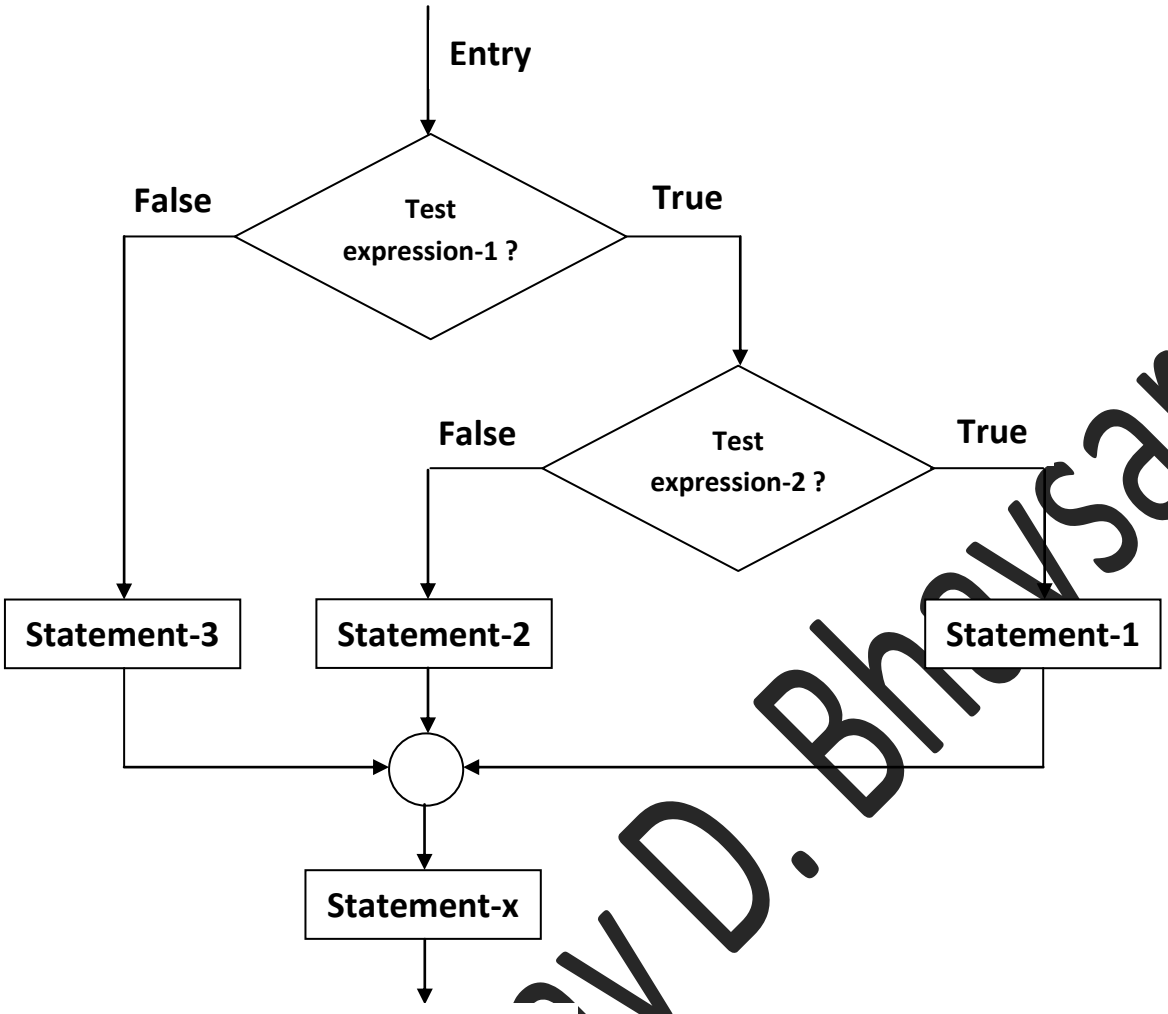
- ➢ **Nesting of If…Else Statements :**
- ✓ When a series of decisions are involved, we may have to use more than one if…else statement in nested form as follows:

```
if(test expression)
{
        if(test expression)
        {
                Statement-1;
        }
```

```
        else
        {
                Statement-2;
        }

}

else

{

        Statement-3;

}


Statement-x;
```

- ✓ If the condition-1 is false, the statement-3 will be executed; otherwise it continues to perform the second test.
- ✓ If the condition-2 true, the statement-1 will be executed, otherwise the statement-2 will be executed and the control is transferred to the statement-x.
- ✓ **Example :**

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



[Flowchart of nested if…else statements]

- **The Else If Ladder :**

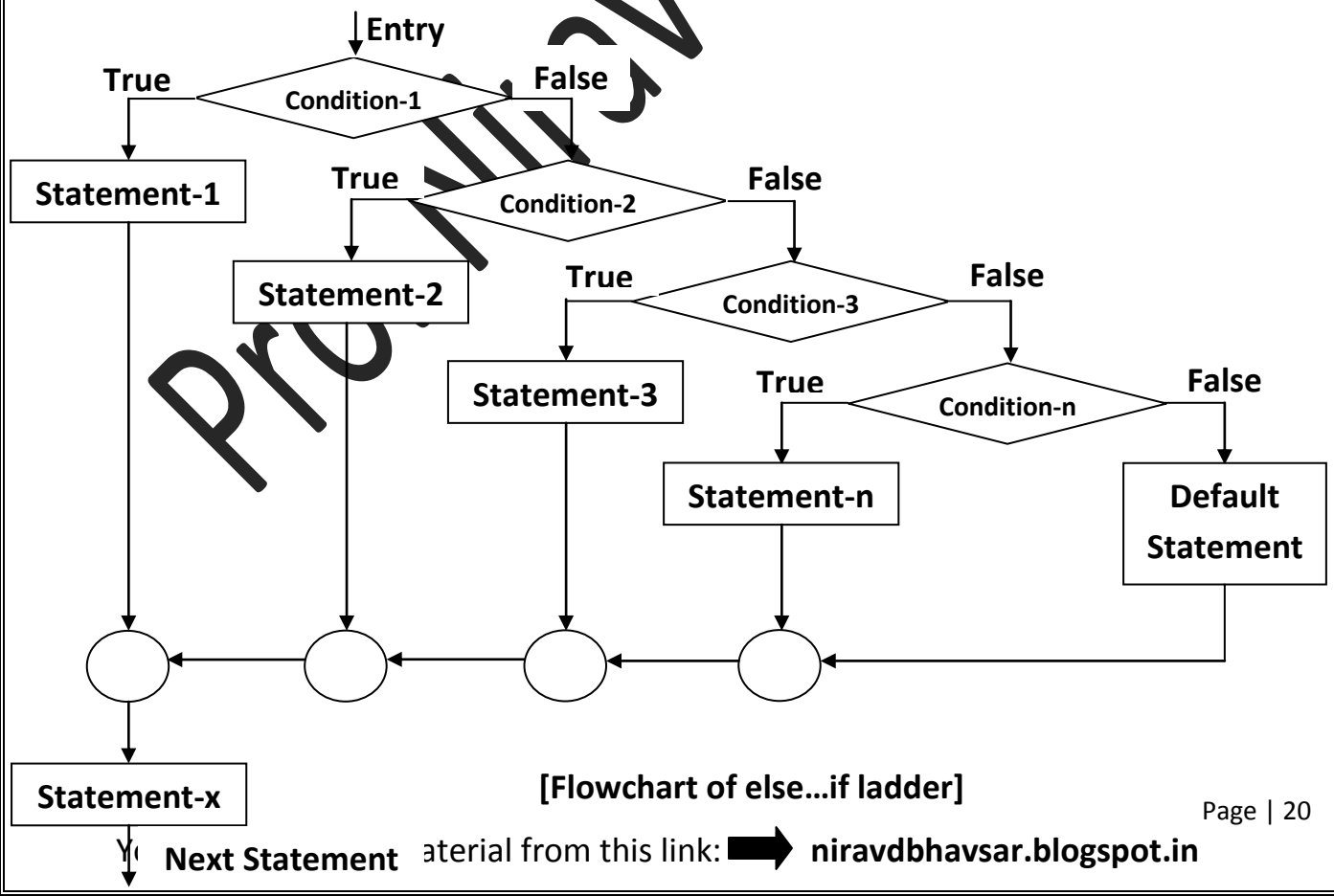
The general form is :

```
if(condition-1)
        statement-1;
else if(condition-2)
        statement-2;
else if(condition-3)
        statement-3;
else if(condition-n)
        statement-n;
else
```

```
        default-statement;
statement-x;
```

- ✓ This construct is known as the else if ladder.
- ✓ The conditions are executed from the top to downwards.
- ✓ As soon as the true condition is found, the associated with it is executed and the control is transferred to the statement-x.
- ✓ When all the n conditions become false, then the final else containing the default-statement will be executed.
- ✓ **Example :**

```
if(marks>=70)
        grade="Distinction";
else if(marks>=60)
        grade="First";
else if(marks>=50)
        grade="Second";
else if(marks>=35)
        grade="Pass";
else
        grade="Fail";
System.out.println("Grade :"+grade);
```



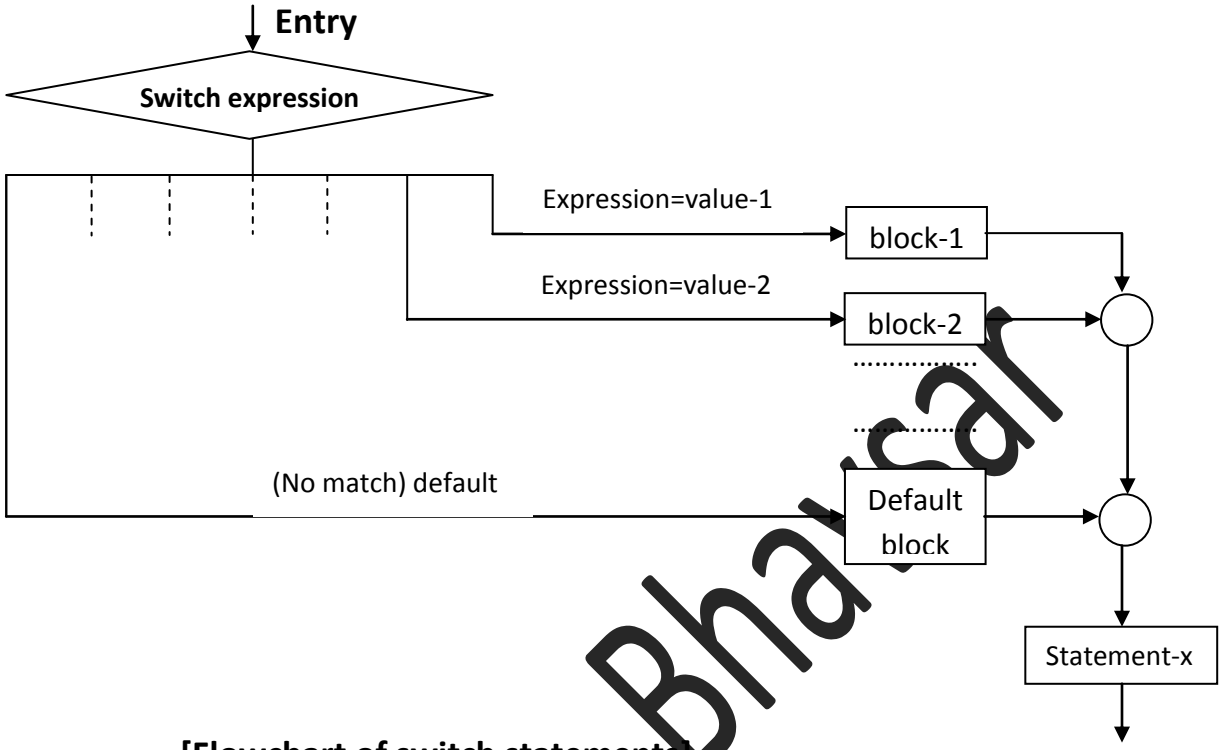
[Flowchart of else…if ladder]

➢ The Switch Statement :

- ✓ Java has a built-in multiway decision statement known as switch.
- ✓ The switch statement tests the value of a given variable(or expression) against a list of case values and when a match is found, a block of statements associated with that case is executed.
- ✓ The general form of the switch statement:

```
switch(expression)
{
        case value-1:
                block-1
                break;
        case value-2:
                block-2
                break;
        ……
        ……
        default:
                default-block
                break;
}
statement-x;
```

- ✓ The expression is an integer expression or characters.
- ✓ value-1, value-2,… are constants or constants expressions and are known as case labels.
- ✓ Each of these values should be unique within a switch statement.
- ✓ block-1, block-2,… are statement lists and may contain zero or more statements.
- ✓ Case labels end with a colon (:).
- ✓ The break statement at the end of each block signals the end of a particular case and causes an exit from the switch statement, transferring the control to the statement-x.
- ✓ The default is an optional case. When present, it will be executed if the value of the expression does not match with any of the case values.
- ✓ If not present, no action takes place when all matches fail and the control goes to the statement-x.



[Flowchart of switch statements]

✓ **Example :**

```
……..
……..
i=marks/10;
switch(i)
{
    case 1:
        grade=”Distinction”;
        break;
    case 2:
        grade=”First”;
        break;
    case 3:
        grade=”Second”;
```

```
                    break;
            case 4:
                    grade="Pass";
                    break;
            default:
                    grade="Fail";
                    break;
        }
        System.out.println(grade);
        ……..
        ……..
```

❖ Iteration Statement:

➢ The While Statement :

- ✓ The simplest of all the looping structures in java is the while statement.
- ✓ The basic format of the while statement :

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

- ✓ The while is an entry-controlled loop statement.
- ✓ The test condition is checked and if the condition is true, then the body of the loop is executed.
- ✓ After execution of the body, the test condition is once again checked and if it is true, the body is executed once again.
- ✓ This process of repeated execution of the body continues until the test condition finally becomes false and the control is transferred out of the loop.
- ✓ The body of the loop may have one or more statements.
- ✓ The braces are needed only if the body contains two or more statements.
- ✓ **Example :**

```
……..
……..
sum=10;
```

```
n=1;
while(n<=5)
{
        sum=sum+n;
        n=n+1;
}
System.out.println("Sum :"+sum);
……….
……….
```

- **The Do…While Statement :**

- ✓ In the while loop test condition is checked before the loop is executed.
- ✓ Therefore, the body of the loop may not be executed if the condition is false
- ✓ On some occasions it might be necessary to execute the body of the loop before test is checked.
- ✓ Such situations can be handled with the help of the do…while statement.
- ✓ The basic format of the do…while statement:

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

- ✓ First body of the loop is executed.
- ✓ At the end of the loop, the test condition in the while statement is checked.
- ✓ If the condition is true, the program continues to execute the body of the loop once again.
- ✓ This process continues as long as the condition is true.
- ✓ When the condition becomes false, the loop will be terminated and the control goes to the statement that appears immediately after the while statement.
- ✓ The test condition is checked at the bottom of the loop, so do…while is an exit controlled loop and therefore the body of the loop is always executed at least once.
- ✓ **Example :**

```
……….
sum=10;
n=1;
do
{
        sum=sum+n;
        n=n+1;
}
while(n<=5);
```

```
System.out.println("Sum :"+sum);
```

- Difference between **while** and **do…while** loops

While	do…while
It will execute only if the test condition is true.	It will execute at least once even if the test condition is false.
It is an entry-controlled loop.	It is an exit-controlled loop.
It is generally used for implementing common looping situations.	It is basically used where the number of statement required to be executed at least once.

➢ The for Statement :

✓ The for loop is another entry-controlled loop.

✓ The general form of the for loop:

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

✓ The execution of the for statement is as follows:

1. Initialization of the control variables is done first, using assignment statement.
2. The value of the control variable is tested using test condition. The test condition is a relational expression. If the condition is true, the body of the loop is executed, otherwise the loop is terminated and the execution continues with the statement that immediately after the for loop.
3. When the body of the loop is executed, the control is transferred back to the for statement. Now the control variable is incremented using an assignment statement(i=i+1) and the new value of the control variable is again tested to see whether it satisfies the loop condition. If the condition is true, the body of the loop is again executed. This process continues till the value of the control variable fails to satisfy the test condition.

Example :

```
for(i=1;i<=10;i++)
{
        System.out.println(i);
}
```

❖ Jump Statement:

- **Break :**

✓ When the break statement is encountered inside a loop, the loop is immediately exited and the program continues with the statement immediately following the loop.
✓ When the loops are nested, the break would only exit from the loop containing it.
✓ That is, the break will exit only a single loop.

```
While(.....)
{
        .....
        ......
        If(condition)
                break;
        .....
}
.....
```

```
do(.....)
{
        .....
        ......
        If(condition)
                break;
        .....
}while(.....);
.....
```

```
for(.....)
{
        .....
        ......
        If(error)
                break;
        .....
}
.....
```

```
for(.....)
{
        .....
        for(.....)
        {
                If(condition)
                break;
        }
        .....
}
```

- **Continue :**

✓ Like the break statement, Java supports another similar statement called the continue statement.

✓ The break which causes the loop to be terminated, the continue, as the name implies causes the loop to be continued with the next iteration after skipping any statement in between.

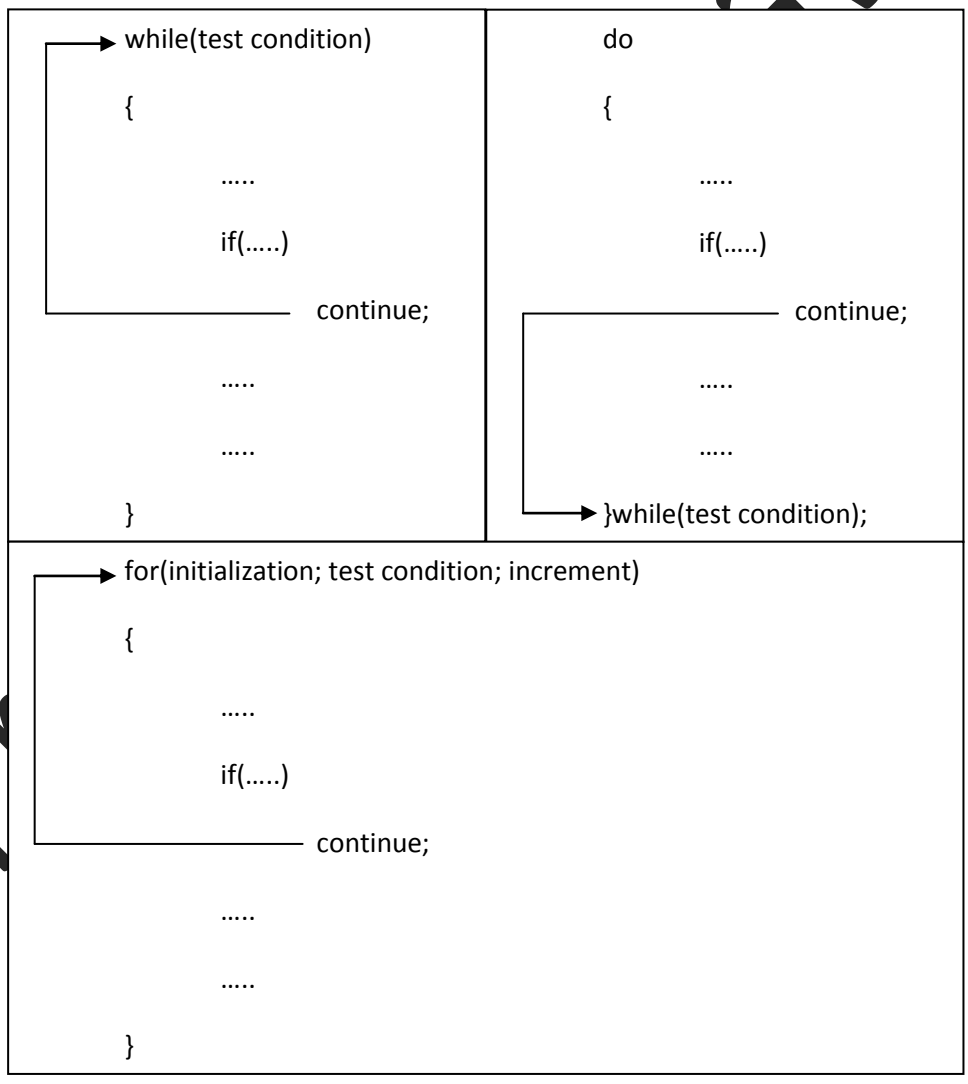
✓ The continue statement tells the compiler, "SKIP THE FOLLOWING STATEMENTS AND CONTINUE WITH THE NEXT ITERATION".

✓ The format of the continue statement is:

Continue;

✓ In while and do loops, continue causes the control to go directly to the test condition and then to continue the iteration process.

✓ In the case of for loop, the increment section of the loop is executed before the test condition is checked.



Arrays :

- ✓ An array is a group of contiguous or related data items that share a common name.
- ✓ A particular value is indicated by writing a number called index number in bracket after the array name.
- ✓ Like any other variables, arrays must be declared and created in the computer memory before they are used.
- ✓ Creation of an array involves three steps:
    1. Declaring the array
    2. Creating memory locations
    3. Putting values into the memory locations

➢ Declaration of Arrays :

- ✓ Arrays in Java may be declared in two forms:
    1. type arrayname[ ];
    2. type [ ] arrayname;
- ✓ **Example :**

```
int number[ ];
float average[ ];
int[ ] counter;
float[ ] marks;
```

- ✓ Remember, we do not enter the size of the arrays in the declaration.

➢ Creation of Arrays :

- ✓ After declaring an array, we need to create it in the memory.
- ✓ Java allows us to create arrays using new operator as shown below:

```
arrayname=new type[size];
```

- ✓ **Example:**

```
number=new int[5];

average=new float[10];
```

- ✓ These lines create necessary memory locations for the arrays number and average.
- ✓ Now, the variable number refers to an array of 5 integers and average refers to an array of 10 floating point values.
- ✓ It is also possible to combine the two steps- declaration and creation-into one as shown below:

```
int number[ ]=new int[5];
```

- **Initialization of Arrays :**

✓ The final step is to put values into the array created. This process is known as initialization.

✓ This is done as follows:

arrayname[index]=value;

✓ **Example :**

number[0]=35;
number[1]=40;
……..
……..
Number[4]=19;

✓ We can also initialize arrays automatically in the same way as the ordinary variables when they are declared.

type arrayname[ ]=(list of values);

✓ The array initialize is a list of values separated by commas and surrounded by curly braces.

✓ Note that no size if given.

✓ The compiler allocates enough space for all the elements specified in the list.

int number[ ]=(35, 40, 20, 57, 19);