# UNIT – I

## FUNDAMENTALS OF OBJECT ORIENTED PROGRAMMING

### Java Introduction:

Java is a general purpose OOP language developed by "Sun Micro Systems" of USA in 1991. Originally it was called "oak" by "James gosling".He was one of the developer of the language. Java designed for the development of computer electronic goods like TV's, VCR's, MOBILES ,etc,.
The following table lists some important milestones in the development of java.

| Year | Development |
|---|---|
| 1990 | Sun Microsystems decided to develop special software to manipulate consumer electronic devices. |
| 1991 | Most popular OOPs language Oak was introduced. |
| 1992 | Green Project team by sun, demonstrated the application of their new language to control a list of home appliances. |
| 1993 | World Wide Web appeared on the Internet and transformed the text-based internet into a graphical rich environment. |
| 1994 | A Web browser called "Hot Java" was development. |
| 1995 | Oak was renamed as Java. |
| 1996 | Sun releases Java Development Kit 1.0 (JDK 1.0) |

## OBJECT ORIENTED PARADIGM

An object is a combination of data and code designed to Operate on a physical (or) abstract entity. Programming with objects is an efficient programming technique with basic data items such as integers, floats, (or) arrays etc. Object oriented programming is a programming methodology, that associate data structure with a set of operations performed on it.

OOP allows us to decompose a problem into the number of entities called objects and they built the data and functions (methods) around these entities. Data of an object can be accessed only by the methods of an object.
The following are the characteristics (or) features of OOP:

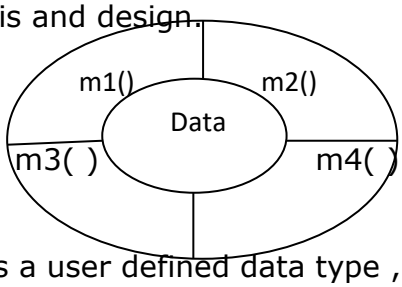
## BASIC CONCEPTS OF OOP

The following are some of the important concepts of oop ;

1. Object
2. Class
3. Data encapsulation
4. Data abstraction
5. Inheritance
6. Polymorphism
7. Dynamic binding
8. Message communication

**1. OBJECT**:- An instance of a class is called an **"object"**. Objects are the basic run-time entities in an object oriented system. They may represent a person, a place, a bank account (or) a table of data etc.

The following diagram shows the notations to represent an object in object oriented analysis and design.

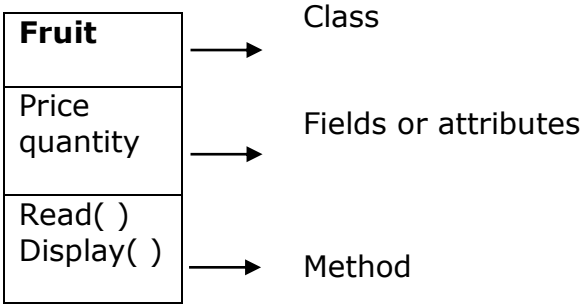



**2. CLASS:-** A class is a user defined data type , which contains data and methods to manipulate that data and Objects are variables(reference) of type class. Once a class has been defined, we can create any number of objects belonging to that class.

For example:- If 'Fruit' has been define a class then the statement:

Fruit mango;

Will create an object "mango" belonging to the class "Fruit"

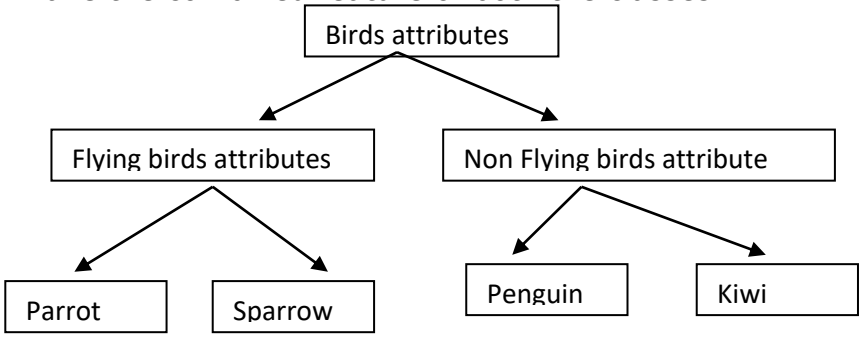



**3. DATA ENCAPSULATION:-** The wrapping up of data and methods into single unit (class) is known as encapsulation. Data encapsulation is the most striking feature of a class. The data is not accessible to the outside world and only those methods which are wrapped in the class, can access it.

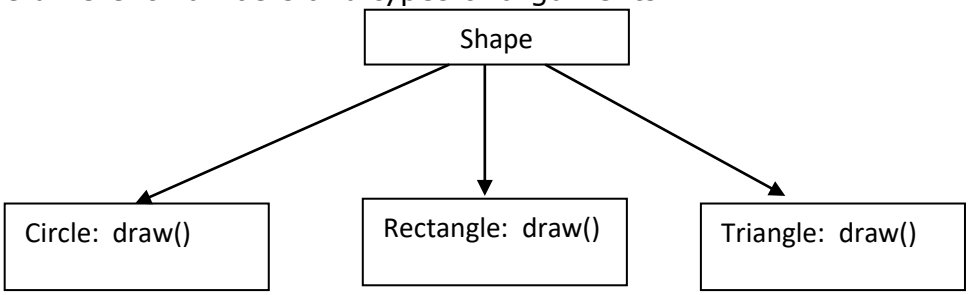
These methods provide the interface between object's data and the program .This protection of data from direct access by the program is called "data hiding".

**4. DATA ABSTRACTION:-** The abstraction means specification of essential features without including the background details (or) explanation. Classes use the concept of abstraction and are defined as a list of abstract data type elements (size, weight, cost etc) and methods that operate on these elements. Since, classes use the concept of data abstraction, they are known as abstract data types (ADTS).

**5. INHERITANCE:-** In OOP, the concept of inheritance provides the idea of reusability. That is we can add additional features to an existing class without modifying it. This is possible by deriving a new class from an existing class. The new class will have the combined feature of both the classes.



**6. POLYMORPHISM:-** Polymorphism means more than one form. For example an operation may gives different behavior in different instances. The behavior depends upon the types of data used in an operation.
For example, consider an add function for two numbers, the operation will generate the sum. If the operands are strings, then the operation will produce a third string by concatenation.
The following diagram shows that a single method name draw() can be used to handle different numbers and types of arguments

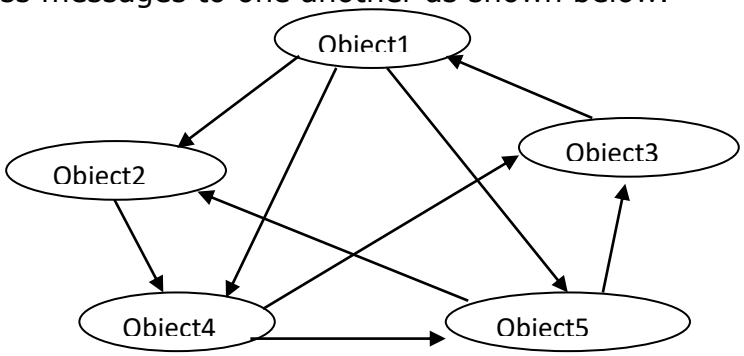



**7. DYNAMIC BINDING:**

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

**8. MESSAGE COMMUNICATION:**

An OOP consists of the set of objects that communicate with each other. Objects communicate with one another by sending and receiving information in the same way people pass messages to one another as shown below.




The message passing involves specifying the name of the object, name of the method and information to be sent. For example consider a statement:

student .marks (htno);

Information
Object Message

**Benefits (or) advantages of oops:** The following are the benefits of OOP **.**

1.Through inheritance we can eliminate repeated 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 co-exist without any interference.
4.It is possible to map (plan) objects in the problem domain.
5.Object oriented system can be easily updated from small system to large systems.
6.Software complexity can be easily managed.

**Disadvantages of oops:**

1) In OOP, the source code is too bigger hence to require maximum amounts of memory.
2) Due to the dynamic binding the execution of an object is slower that in procedural programming.
3) Bottom-up approach will create a little confusion to the programmers.
4) Emphasis is only on data rather than procedure.
5) Due to the data hiding concept sometimes the programmer has to write additional code i.e. methods.
6)

**Applications of oops:** The following are the list of applications of OOP:

1. User interface design. (Windows)
2. Real time systems.
3. Simulation and modeling.
4. Object oriented data bases.
5. Artificial intelligence (AI) and expert systems.
6. Hypertext and hyper media.
7. Network and parallel programming.
8. CAD or CAM systems (computer aided design or computer aided modeling).

## Java features

Java is a general purpose object-oriented programming language developed by Sun Microsystems of USA in the year 1991. Initially it was named as "Oak" later in 1995 it was renamed as 'JAVA'.The following are the features of java programming language:

1. Compiler and Interpreter
2. Platform independent and Portable
3. Object oriented
4. Robust and Secure
5. Distributed
6. Small ,simple and familiar.
7. Multi threaded and interactive
8. High performance
9. Dynamic and extensible

### 1.Compiled and Interpreted:

Java program execution is in two-stage process. In first stage java compiler (**javac**) translates source code into byte code and then in second stage java interpreter **(java**) generates machine code that can be directly executed by the machine.

### 2.Platform independent and portable:

Java program can be implemented in any operating system or processor due to the byte code of Java Virtual Machine (JVM). Java programs can be easily moved from one computer system to another. "Java applets" can be translated from one system to another and executed there. This is the reason why java has become most popular for Internet programming.

Java maintains the portability in two ways:

i) First java compiler generates the byte code instructions that can be implemented in any system.
ii) Second the size of the primitive data types are machine independent.

### 3.Object-oriented:

Java is a true object-oriented programming language. Almost everything in java is a object. All programs code and data are available within objects and classes.

**4.Robust and Secure:**

Java is a robust (strong) language. This means java provides many safeguards to maintain reliable code. It has strict compile-time and runtime checking for data types. It also provides exception handling, which can handle errors and eliminates any risk of operating systems.

**5.Distributed:**

Java is designed as distributed language for creating applications on networks. It has the ability to share data and programs. This enables multiple programs at multiple remote locations to collaborate and work together on a single project.

**6.Small, Simple and Familiar:**

Java is a small simple language. Java code is very reliable compared to C and C++. For example, java does not use pointers, processor header files, goto statements and many others. It also eliminates operator overloading and multiple inheritance. Java is a familiar language with all these constructs. Java is simplified version of C++.

**7.Multi-Threaded and Interactive:**

Multi-threaded means handling multiple tasks simultaneously. Java supports multi-threaded programs. This feature greatly improves the interactive performance of the graphical applications.

**8.High Performance:**

Java performance is impressive for an interpreted language mainly due to use of intermediate byte code. Java is also designed to reduce problems during run-time. The concept of multi-threading improve the overall execution speed of a java program.

**9.Dynamic & Extensible:**

Java is a dynamic language. Java is capable for dynamic linking in new class libraries, method and objects. Java program supports functions written in other languages such as C and C++. These functions are also known as "Native Methods". this facility enables the programmer to use the efficient functions available in these languages. Native methods are linked dynamically at run-time.

## Structure Of Java Program

A java program may contain many classes of which only one class defines a main method. Classes contain data members and methods. This methods access and operate the data members of the class. Methods may contain data type declaring and executable statements. A java program may contain one or more section like..

| Documentation section | Suggested |
|---|---|
| Package statement | Optional |
| Import statement | Optional |
| Interface statement | Optional |
| Class definition | Optional |
| Main method class | Essential |

**Documentation section:**

Documentation section contains a set of comment lines. These lines can specify the name of the program, author and other details.

Java permits 3 types of comments.

1. Single line comment: **//**
2. Multiline comment: **/* …….*/**

3. Documentation comment: **/** ……..*/**

**Package statement:**

This is the first statement in a java file. This statement declares a package name and informs the compiler that the class defined here belongs to this package.

Ex: package student;

This package statement is optional. i.e; our classes don't have to be part of package.

**Import statement:**

The next statement after the package statement may be a number of import statements. This is a similar to the #include statement in 'C' and 'C++'.

Ex: import student.Test;

Here load the 'Test' class from the 'student' package

**Interface statement:**

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

**Class definitions:**

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

**Main method:**

Every java stand alone program requires a main method as its starting point. This is the essential part of the java program. A simple java program may contain only this part. The main method creates objects of various classes and establishes communications b/w them. On reaching the end of the main, the program terminates and the control passes back to the operating system.

## Java tokens

The smallest individual units in a program are known as tokens. Here java program is a collection of tokens, comments and white spaces. Java language includes five types of tokens. They are

**Keywords (reserved word)**
**Identifiers**
**Literals**
**Operators**
**Separators**

**Keywords: -** In java, some words are having predefined meaning; such words are called keywords or reserved words. Keyword have specific meaning in java, we cannot use them as names for variables, classes, methods and so on. All keywords are to be written in lower case letters. Java reserve 60 keywords and at present using 53 keywords only.

Ex: byte, short, char, Boolean, synchronized, final,…

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

1. They can have alphabets, digits and underscore and dollar sign character.
2. They must not being with digits.
3. Uppercase and lowercase are distinct (different).
4. They can be of any length.

5. They don't have blank spaces.
6. Keyword can't be used as an identifier.

**Java developers have followed some naming conventions .**

1.All classes and interfaces start with a leading uppercase letter and each subsequent word with a leading uppercase letter.
Ex: MrrProgram, HelloJava, SampleProgram, EmpDetails
2.All public methods and instance variables start with a leading lowercase letter and each subsequent word with a leading uppercase letter.
Ex. totalMarks, empAge, readData (), displayData()
3.All private and local variables use only lowercase letters combined with underscore
Ex. age, roll_number, emp_age
4.Variables that represent constant values use all uppercase letters and underscore between the words.
Ex. TOTAL, PRINCIPAL_AMOUNT

**Literals:-**
Literals in java are a sequence of characters (digits, letters, and other character) that represent constant values to be stored in a variable. Java language specifies **5** major types of literals.

**1.Integer literals**
**2.Floating point literals**
**3.Character literals**
**4.String literals**
**5.Boolean literals**

**Operators:-**
An operator is a symbol that takes one or more arguments and operates on them to produce a result. Java contains **8** types of operators.

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

**Separators:-**
Separators are symbol used to indicate where groups of code are divided and arranged. They basically define shape and function of our code. The following

| Name | What it is used for |
|---|---|
| Paranthesis **( )** | Used to enclose parameters in methods definitions and it also used in expressions. |
| Braces **{ }** | Used to define a block of code for classes, methods and local scopes. |
| Brackets **[ ]** | Used to declare array types and for referencing array values. |
| Comma **,** | Used to separate consecutive identifiers in a variable declaration also used to chain statements together for statements |
| Semicolon **;** | Used to separate statements. |
| Period **.** | Used to separate package names from sub packages and classes; also used to separate a variable or method from a reference variable. |

table specifies that the list of separators and their functions.

## Creating and Executing a java program

Implementing of java program includes a series of steps. Thy are

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

**Creating the program:**

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

```
D:\satya> edit Test.java
class Test
{
public static void main(String args[ ])
{
System.out.println("Hello Java");
}
}
```

The filename should be similar to the main method class name and extension is **.java**. This file is called source file.

**Compiling the program:**

Java compiler is used to compile the source of program. **"javac"** is the java compiler.

The following is used for compilation.

```
D:\satya> Javac Test.java
```

If every thing is ok, the javac compiler creates a file called **"Test.class"** containing the byte codes at the program the compiler automatically names the bytecode as <classname>.class.

**Running the program:**

We need to use the java interpreter to run a stand alone program. At the command prompt, type

```
D:\satya>java Test
```

Now the interpreter looks for the main( ) method in the program and begins execution from there. It can be display the output as: Hello Java.

## Java virtual machine [JVM]:

All language compilers translate code into machine code for a specific computer to achieve the architecture neutrality. Java compiler produces an intermediate code known as byte code for a machine that does not exist. This machine is called 'java virtual machine' and is exists only inside the computer memory. It is a stimulated computer and does all major function of a real computer. The following figure illustrates the procedure of compiling a java program into bytecode which is also called as virtual machine code.

The virtual code isn't machine specific. The machine specific code is generated by java interpreter by acting a mediator b/w the virtual machine and real machine. An interpreter is different for different machines. The following figure

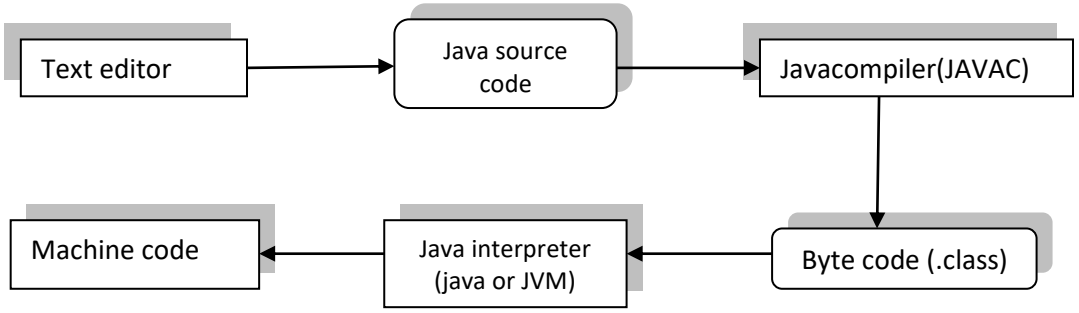


Fig: process of converting bytecode into machine code.

# 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. Examples of the program are

**Ex:** average height total_height classStrength

Variable names may consists of alphabets, digits, underscore and dollar characters.

**Rules for Variable Names :**

1.A variable name must contain combination of alphabets, digits, underscore and dollar sign.
2.They must not begin with a digit.
3.Uppercase and Lowercase are distinct. This means **Total** is not same as **TOTAL**.
4.It should not be a keyword.
5.White space is not allowed.
6.Variable names can be of any length.

**Declaration of Variables :**

Variables are the names of storage locations. After designing variable names, we must declare them to the compiler. Declaration does three things.

1.It tells the compiler what the variable name is.
2.It specifies what type of data the variable will hold.
3.The place of declaration decides the scope of the variable.

A variable must be declared before it is used in the program. Syntax for declaration of a variable is as follows.

**Syntax :** Datatype variable**1**, variable**2**, variable**3**…….variable**n**;

**Eg:-** int count; float x, y; double p1;

**Giving values to variables :**

A variable must be given a value after it has been declared and before it is used in an expression. This can be achieved in two ways.

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

**Assignment Statement :** A simple method of giving value to a variable is through Assignment statement.

**Syntax :**

```
Variablename = value;
```

**Ex:** initialvalue = 0;
finalvalue = 100;
yes = 'x';

we can also string assignment expression.

**Ex: x = y = z = 0;**

It is also possible to assign a value to a variable at the time of its declaration. The process of giving initial values to variables is known as "initialization".

**Syntax :**

```
Datatype variablename = value;
```

**Ex:** int finalvalue = 100;
double total = 75.36;

**Read Statement :**

We may also give values to variables interactively through the keyboard using the **"readLine( )"** method.

**Ex:** DataInputStream ds = new DataInputStream(System.in);
int x=Integer.parseInt(ds.readLine( ));

The readLine ( ) method reads the input from the keyboard as a string.

---

## Scope of variables

The area of the program where the variable is accessible is called it's scope. A block is begin with an opening brace and ended by a closing curly brace. Java allows variables to be declared with in any block. A block defines a scope.

The scope defined by a method begins with it's opening brace. However, if that method has parameters, they are also included within the method's scope. So we can protecting from unauthorized access and modification.Java variables are classified into three kinds.

1.Instance variables.
2.Class variables(static)
3.Local variables.

**Instance variables :**

Instance variables are declared inside a class but out side the methods and blocks.These are created when an objects are instantiated and therefore they are associated with the object. They take different values for each object.

Ex:- name , rollNo,m1, m2, m3 of the **Student** class.

**Class Variables :**

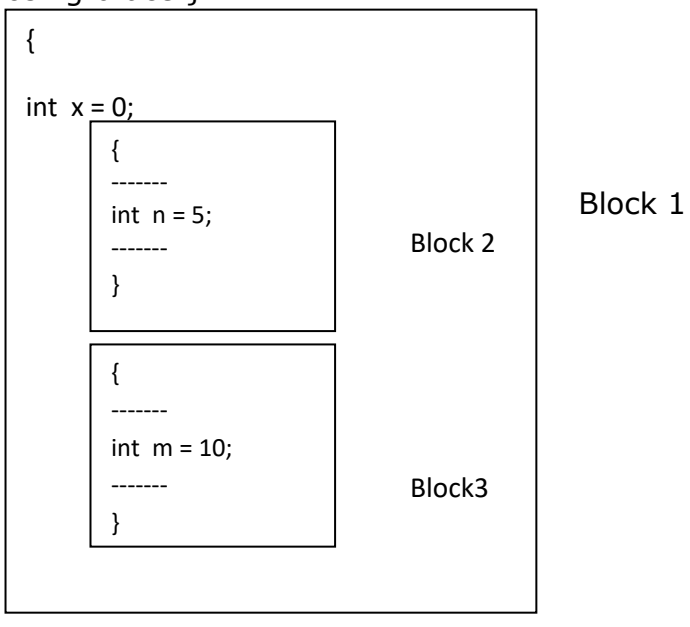
These variables are declared inside the class but outside the methods and blocks but by using **'static'** keyword. These variables also called static variables.

Class variables are global to a class and belong to the entire set of objects that class creates only one memory location is created for each class variable.

Ex:- section, average of me,m2,m3 belong to **'Student'** class

**Local variables :**

Variables declared and used inside methods are called local variables. They are not available for use outside the method definition. Local variables can also be declared inside program blocks that are defined between an opening brace { and a closing brace }.



Each block contains its own set of local variable declarations. In the above figure the variable x declared in block 1 is available in all the three blocks. The variable n declared in block 2 and is available only in block 2. similarly m is accessible in block3.

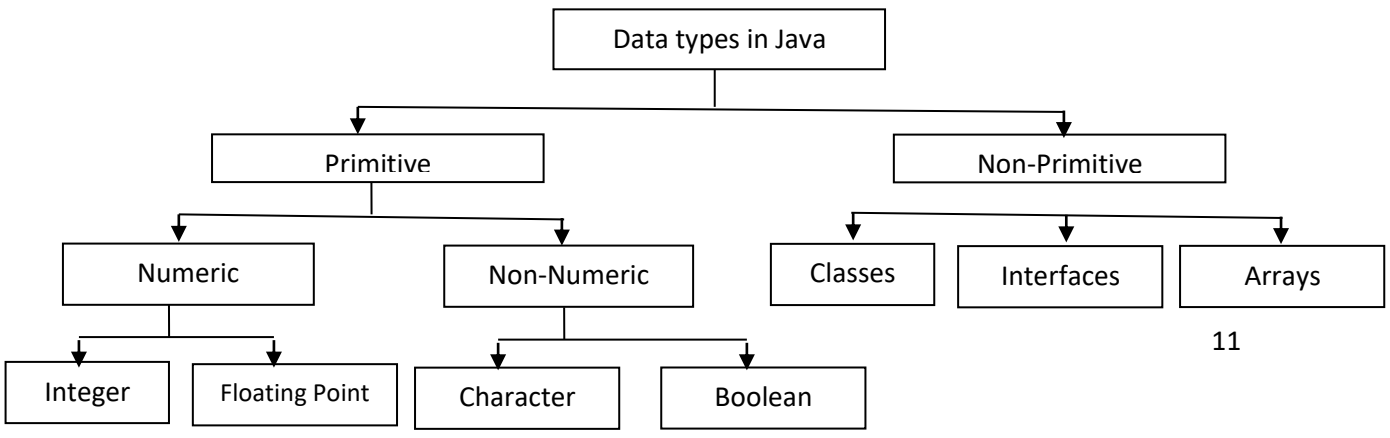
```
/* Example program for scope of variables*/
class Subtraction
{
int x, y;
static int count;
void  getData(int a, int b)
{
X=a;
Y=b;
}
void putData()
{
int c;
c=x-y;
System.out.println("x-y = "+ c);
count++;
System.out.println("count = "+ count);
}
}

class Sub
{
public static void main(String args[])
{
Subtraction obj1= new Subtraction();
obj1.getData(10,20);
obj1.putData();
Subtraction obj2= new Subtraction();
obj2.getData(40,30);
obj2.putData();
Subtraction obj3= new Subtraction();
obj3.getData(100,50);
obj3.putData();

}
}
```

## DATA TYPES

Every variable In java has a data type. Data type specify the size and type of values that can be stored. The data type allow the programmer to select the type appropriate to the needs of the application.



Primitive data types are also called intrinsic or built-in types and non primitive data types are also called 'Derived data types or reference Types'.

**Integer Types :**

Integer types can hold whole numbers such as 123, -96, 5639 etc. the size of the values that can be stored depends on the integer data type we choose. Java supports 4 types of integer data types. They are byte, short, int, long. Java does not support the concept of unsigned.

| Type | Size | Minimum Value | Maximum Value |
|---|---|---|---|
| Byte | 1 Byte | -128 | 127 |
| Short | 2 Bytes | -32768 | 32767 |
| Int | 4 Bytes | -2,147,483,648 | 2,147,483,647 |
| Long | 8 Bytes | -9,223,372,036,854,775,808 | 9,223,372,036,854,775,807 |

**Floating point types:**

Floating point types can hold numbers containing fractional parts such as 27.59,-1.375. There are two types of Floating point types in Java. They are float, double.

- The float type values are single – precision numbers.
- The double type values are double – precision numbers.

| Type | Size | Minimum Value | Maximum Value |
|---|---|---|---|
| Float | 4 Bytes | 3.4e-038 | 3.4e+038 |
| Double | 8 Bytes | 1.7e-308 | 1.7e+308 |

**Character Type :**

In order to store character constants in memory, Java provides a character data type called '**char**'. The char data type reserves 2 bytes of memory but it can hold only a single character.

**Boolean Type:**

Boolean type is used when we want to test a particular condition during the execution of the program. There are only two values that a Boolean type can hold: true or false. Boolean type is denoted by the keyword '**boolean**' and uses only one bit of storage.

# Type casting

Type casting is the process of converting the value in one data type to another data type. It uses the following syntax.

```
type  var1 =( type ) var2;
```

Ex:-
```
int m=50;
byte  b = (byte) m;
short  b = (short) m;
```

Casting is often necessary when a method returns a type different than one we required. Four integer types can be casted to any type other than Boolean. Casting into smaller type may result in the loss of data. Similarly the float and double can be casted into any type except Boolean. Casting a floating point value to an integer will result in the loss of fractional part.

**Automatic conversion(Widening):**

The following table shows the casting that results in no loss of information:

| From | To |
|---|---|
| byte | short,int,long,float,double. |
| short | int,long,float,double. |
| char | int,long,float,double. |
| Int | long,float,double. |
| long | float,double. |
| float | double |
| double | |

For some types, it is possible to assign a value of one type to a variable of a different type to a cast.java does the conversion of the assigned value automatically this is known as automatic conversion. Automatic type conversion is possible only if the destination type has enough precision to store the source value.

Automatic type conversion is possible only the process of assigning a smaller type to a large is known as promotion or widening.

Eg: byte a=10;
int a=b;

**Narrowing :**

The process of assigning a large type to a smaller type is known as "narrowing". This may result in the loss of information. Narrowing is also known as explicit type casting. It is programmers responsibility to cast(convert) large type to small data type.

Ex:- int m=129;
byte b =(byte) m; // here loss of value

note: (write example program)

# OPERATORS

An operator is a symbol used to perform an operation. If an operator acts on a single variable, then it is called *'Unary Operator'.* If an operator acts on two variables then it is called *'Binary Operator'.* If an operator acts on three variables then it is called *'Ternary Operator'.* The following are the Operators available in Java.

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

**1 Arithmetic Operators :** Arithmetic Operators are use to perform arithmetic calculations. The following are the arithmetic operators.

| Operator | Meaning |
|---|---|
| + | Addition |
| - | Subtraction |

| | |
|---|---|
| * | Multiplication |
| / | Division |
| % | Modulo Division (Remainder) |

**Example:** Let a, b are integer type and c is float type variables. If a=15, b=10 and c=2.5 then

| Expression | Result |
|---|---|
| a+b*c | 40.0 |
| a+b-c | 22.5 |
| a / b | 1(decimal part truncated) |
| a % b | 5 (remainder) |
| 15 / 4.5 | 3.3333333 |

Arithmetic operators are classified into 3 types
1. Integer Arithmetic
2. Real Arithmetic
3. Mixed mode Arithmetic

**Integer Arithmetic:** When both the operands in a single arithmetic expression are called integer expression and the operator is called integer arithmetic. Integer arithmetic always returns an integer value.

**Real Arithmetic:** When an arithmetic expression involves operands are real then that expression is called real expression and the operator is called real arithmetic.

**Note:** % operator cannot be applied on floating point data in c or c++ languages. But in java, we can apply for floating point data.

**Mixed Mode Arithmetic:** When one of the operand is real and the other is integer then the expression is called "Mixed Mode Arithmetic Expression". If either of the operand is real type then the second type is converted to real and the real arithmetic is performed. The result will be real.

**2 Relational Operators :**

The relational operators are used to compare two values and it gives either true or false. These are used to form simple conditions. All the relational operators are applied on any type of data. The following are the relational operators.

| Operator | Meaning |
|---|---|
| < | Is Less than |
| < = | Is Less than or equal to |
| > | Is Greater than |
| >= | Is Greater than or equal to |
| = = | Is Equal to |
| ! = | Is not equal to |

**Example :** Let a=15 and ch='A' then

| Expression | Result |
|---|---|
| a = = 15 | True |
| a > 15 | False |
| a< = 10 | False |
| a ! = 13 | True |
| ch = = 'A' | True |
| ch > 'A' | False |

**3. Logical Operators:**

These operators are used to combine two or more conditions and gives the result either true or false. These conditions are called compound conditions.

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

**Logical AND (&&) Operator:** This operator gives true if all the conditions are true otherwise it gives false. The truth table for && operator is as follows:

| Condition1 | Condition2 | Condition1 && Condition2 |
|---|---|---|
| TRUE | TRUE | TRUE |
| TRUE | FALSE | FALSE |
| FALSE | TRUE | FALSE |
| FALSE | FALSE | FALSE |

**Logical OR ( ¦¦ ) Operator:** This operator gives false if all the conditions are false otherwise it gives true. The truth table for || operator is as follows:

| Condition1 | Condition2 | Condition1\|\| Condition2 |
|---|---|---|
| TRUE | TRUE | TRUE |
| TRUE | FALSE | TRUE |
| FALSE | TRUE | TRUE |
| FALSE | FALSE | FALSE |

**Logical NOT (!) Operator:** This operator negates(opposite) the result of a condition. It means if the condition is true then it gives false. If the condition is false then it returns true. The truth table for ! operator is as follows:

| Condition | ! (Condition) |
|---|---|
| TRUE | FALSE |
| FALSE | TRUE |

**Example:** Let a=15, b=20 then

| Expression | Result |
|---|---|
| a==15 ¦ ¦ b== 20 | True |
| a>15&&b>=20 | False |
| a<=15&&b<=20 | True |
| ! (a = = 15) | False |

**4. Assignment Operators:**

The assignment operators are used to assign (store) a value to a variable. An assignment operator always consists of a variable on left hand side and expression / constant / variable on right hand side. There are two types of assignment operators. They are

i. Simple Assignment Operator
ii. Compound Assignment Operators

**Simple Assignment Operator:** The equal sign '=' is used as simple assignment operator. The general format of this operator is as follows:

Variable = { Expression / Constant / Variable } ;

In the above syntax, the value of Expression / Constant / Variable is assigned to the variable on the left hand side of '=' sign.

**Eg:** x = 35; y = a + b * c; z = y; a = b = 50;

**Compound Assignment Operators:** The general form of compound assignment operators is as follows

Variable Compound operator { Expression / Constant }

= ;

In the above syntax, the operator can be either +=, –=, *=, /= or %=

| Operator | Example | Meaning |
|---|---|---|
| += | 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 |

**5. Increment and Decrement Operators:**

**Increment Operator (++):**

The increment operator is a unary operator. It is used to increase the value of an operand by 1. The operand must be a variable. The increment operator '++' has different meaning depending on the position it is used. It means this operator is used in two ways. They are pre-increment and post-increment.

**Pre-increment:** If the '++' operator precedes the operand then it is called "Pre increment". In this method, the pre-increment operator increments the value of variable first and then the new value is used in the expression.

**For example:** c = ++a; means a = a + 1;
c = a;

**Post-increment:** If the '++' operator succeeds the operand then it is called "Post increment". In this method, the old value of the variable is used in expression and then it is incremented.

**For example:** c = a++; means c = a;
a = a + 1;

**Decrement Operator (– –):**

The decrement operator is a unary operator. It is used to decrease the value of an operand by 1. The operand must be a variable. The decrement operator '– –' has different meaning depending on the position it is used. It means this operator is used in two ways. They are pre-decrement and post-decrement.

**Pre-decrement:** If the '– –' operator precedes the operand then it is called "Pre decrement". In this method, the pre-decrement operator decrements the value of variable first and then the new value is used in the expression.

**For example:** c = – –a; means a = a – 1;
c = a;

**Post-decrement**: If the '– –' operator succeeds the operand then it is called "Post decrement". In this method, the old value of the variable is used in expression and then it is decremented.

**For example:** c = a– –; means c = a;
a = a – 1;

| Initial Values | | Expression | Final Values | |
|---|---|---|---|---|
| A | B | | A | B |

| | | | | |
|---|---|---|---|---|
| 3 | 7 | a = ++b | 8 | 8 |
| 3 | 7 | a = b++ | 7 | 8 |
| 3 | 7 | a = – –b | 6 | 6 |
| 3 | 7 | a = b– – | 7 | 6 |

**6. Conditional Operator:** This operator performs condition based execution. This operator operates on three operands. Hence it is also known as Ternary Operator. This operator simplifies "if…else" control statement. The general form of conditional operator is as follows:

**Syntax: var = (condition) ? Statement1 : Statement2 ;**

**Working:** The conditional operator first evaluates the given condition. If the condition is true then 'Statement1' is evaluated. If the condition is false then 'Statement2' is evaluated.

**Example:-**

```
import java.io.*;
class max
{
public static void main(String args[]) throws IOException
        {
        int a,b,large;
        DataInputStream d=new DataInputStream(System.in);
        System.out.print("Enter First Number");
        a=Integer.parseInt(d.readLine());
        System.out.print("Enter Second Number");
        b= Interer.parseInt(d.readLine());
        large = (a>b) ? a : b;
        System.out.println("Large value is " + large);
        }
}
```

**7. Bitwise Operators:** Java has the capability of manipulating the bits (0 or 1) directly stored in the memory. These operators perform bitwise operations on a bit by bit level. However, they must be used only with integer or character type of values. The operators are as follows.

| Operator | Meaning |
|---|---|
| & | Bitwise AND |
| ¦ | Bitwise OR |
| ^ | Bitwise XOR (Exclusive OR) |
| ~ | 1's complement |
| << | Left shift |
| >> | Right shift |
| >>> | Right shift with Zero fill |

**Bitwise AND (&) operator:** If all the bits are "1" then this operator returns "1" otherwise it returns "0".

**Truth table:**

| & | 1 | 0 |
|---|---|---|
| **1** | 1 | 0 |
| **0** | 0 | 0 |

**Example:** Let x = 15 and y = 19, then z = x & y is as follows

| | |
|---|---|
| Value of x : | 00001111 |

| Value of y : | 00010011 |
|---|---|
| Value of z : (x & y) | 00000011 |

**Bitwise OR ( ¦ ) operator:** If any or all the bits are "1" then it returns "1" otherwise it returns "0".

**Truth table:**

| ¦ | **1** | **0** |
|---|---|---|
| **1** | 1 | 1 |
| **0** | 1 | 0 |

**Example:** Let x = 15 and y = 19, then z = x ¦ y is as follows

| Value of x : | 00001111 |
|---|---|
| Value of y : | 00010011 |
| Value of z : (x ¦ y) | 00011111 |

**Bitwise XOR ( ^ ) operator:** If one bit is "1" and other is "0" then it returns "1" otherwise it gives "0".

**Truth table:**

| ^ | **1** | **0** |
|---|---|---|
| **1** | 0 | 1 |
| **0** | 1 | 0 |

**Example:** Let x = 15 and y = 19, then z = x ^ y is as follows

| Value of x : | 00001111 |
|---|---|
| Value of y : | 00010011 |
| Value of z : (x ^ y) | 00011100 |

**1's complement (~) operator:** This operator is a unary operator that reverses the state of each bit. That means it changes "0" as "1" and "1" as "0".

**Example:** Let x = 19, then z = ~x is as follows

| Value of x : | 00010011 |
|---|---|
| Value of z : ( ~x ) | 11101100 |

**Left Shift (<<) operator:**

This operator is used to move bit patterns to the left. It is used in the following form **op<<n**

Here, 'op' is an integer value and 'n' is number of bit positions. The left shift operator shifts all the bits of operand 'op' to the left by 'n' positions. The left most 'n' bits in the original bit pattern will be lost and right most 'n' bit positions are filled with zeros.

**Example:** Let us consider 'a' is an unsigned integer whose bit pattern is as follows:

| a → | 1100110011001111 |
|---|---|
| a<<3 → | 0110011001111000 |
| a<<5 → | 1001100111100000 |

The left shift operator is used for multiplication purpose. It means, x<<n gives the result as $x*2^n$.

**Example :** 5<<1 gives 10, 5<<2 gives 20, 5<<3 gives 40

**Right Shift (>>) operator:**

This operator is used to move bit patterns to the right. It is used in the following form

**op>>n**

Here, 'op' is an integer value and 'n' is number of bit positions. This operator shifts all the bits of operand 'op' to the right by 'n' positions. The right most 'n' bits in the original bit pattern will be lost and left most 'n' bit positions are filled with zeros.

**Example:** Let us consider 'a' is an unsigned integer whose bit pattern is as follows:

| | |
|---|---|
| a → | 1100110011001111 |
| a>>3 → | 0001100110011001 |
| a>>5 → | 0000011001100110 |

The right shift operators can also be used for division purpose. It means, x>>n gives the result as $\frac{x}{2^n}$

**Example :** 15>>1 gives 7, 15>>2 gives 3, 15>>3 gives 1

**Right Shift Zero fill (>>>) operator:**

When dealing with positive numbers, there is no difference between >>> and >> operators. The difference arises when dealing with negative numbers. The negative numbers have the higher order bit set to 1. The Right shift Zero fill operator shifts zeros into all the upper bits, including the higher order bit. Thus make a negative number into positive number.

**op>>>n**

**8. Special Operators:** The following are the special operators used in Java.

1) Instanceof Operator
2) Dot Operator
3) Cast Operator
4) new Operator

**Instanceof Operator:** The instanceof is an object reference operator and return true if the object on the left hand side is an instance of the class given on the right side. This operator determines whether the object belongs to a particular class or not.

**Ex:** s1 **instanceof ( ** student **);**

It gives ***true*** if the object person belongs to the class student; otherwise it is ***false***

**Dot Operator :** this operator is also called as member selection operator(.) . the Dot Operator is used to access the instance variables, methods of class objects, classes, and sub-packages from a package.

**Ex:** person.age;
person.salary( );

**Cast Operator :** Cast operator is used to convert one data type into another data type. This operator can be used by writing datatype inside parenthesis.

**Ex:** double x = 10**.**54;
int y = (int) x; // stores 10 into y.

**new Operator : new** operator is used to create objects to classes. Objects are created on heap memory by JVM, at runtime.

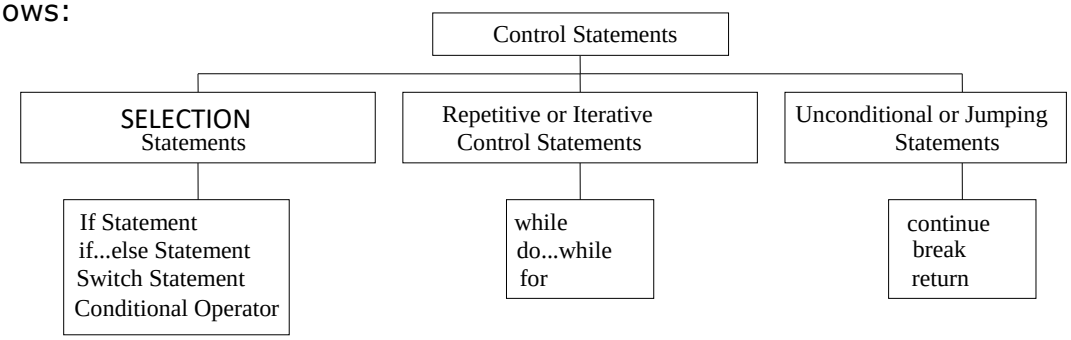
**Syntax : classname obj = new classname( );**

# DECISSION MAKING OF BRANCHING AND LOOPING

## Control Structures

Generally any high level program will be executed in sequential manner. With the control statements, the sequential flow of execution will be altered. The following are different control statements available in Java.

In Java, the control statements are classified into 3 types. They are as follows:




## Selection (Conditional) Control Statements :

A java program is a set of statements which are normally executed in the order in which they appear. When the program breaks a sequential flow and jumps to another part of the code, then it is called branching. When a branching is based on a particular condition then it is called conditional branching. When the branching is not based on any particular condition, it is known as conditional branching.

The java language passes such decisions making capabilities and supports the following statements known as control or decision making statements.

1) 'if' statement
2) 'switch' statement
3) 'conditional' statement

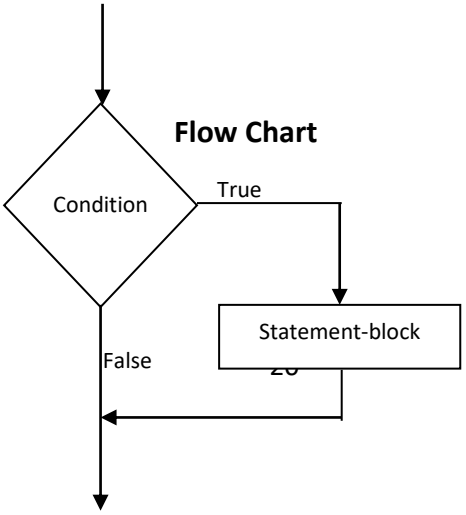
The Conditional Control Statements are also known as Decision Making statements. The conditional statements allow the programmer to select one particular group of statements from several available groups depending on the given condition. There are mainly three types of conditional statements. They are

1. if or if…else statement
2. switch statement
3. Conditional Operator Statement.

## Decision Making with if statement :

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

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




### Simple if statement :

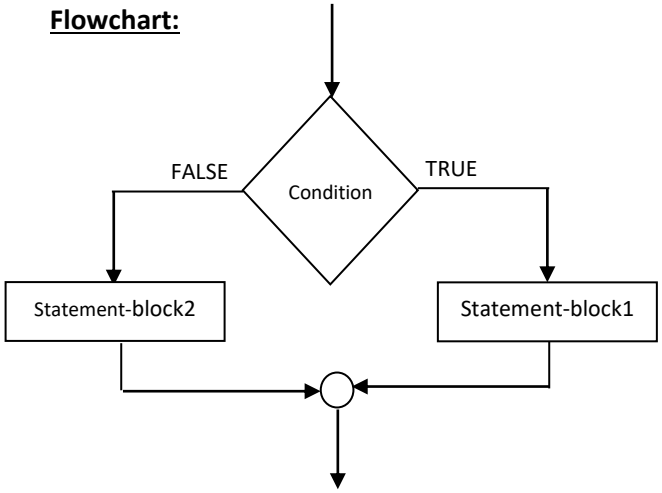
The 'if' statement is a conditional control statement that executes the specified set of instructions based on the condition.
The general form of simple **if** statement is

```
if ( Condition )
{
      Statement – block;
}
```

**Working :** if the **condition** is true, then the statement block will be executed. Other wise the statement block will be skipped and the execution will jump to the next statement after the statement block.

The **statement - block** may be a single statement or a group of statements. If there is only a single statement in the statement block, there is necessity of curly braces ({ } ).

**Flowchart:**




**If...else statement :**

It is also conditional control statement that executes the statements based on the condition. The if..else statement is an extension of simple if statement. The general form is

```
if ( Condition )
     Statement – block1;
else
    Statement – block2;
```

**Working :** if the **Condition** is true then the statement – block 1 will be executed. Otherwise the statement – block2 will be executed.

**Ex :**
```
if (n%2==0)
   System.out.println("Even Number");
else
   System.out.println("Odd Number");
```

**Nested if...else statement :**

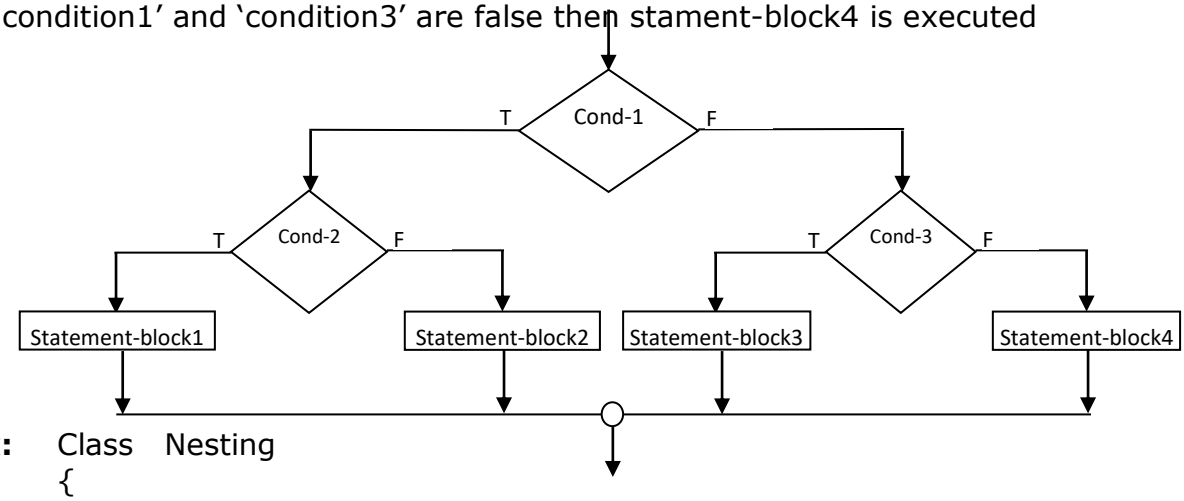
When the 'if..else' statement is used within another 'if…else' statement then it is called "nested if" statement. The general format is as follows:

**Syntax:**

```
if (condition1)
{
  if (condition2)
      statement-block1 ;
  else
      statement-block2 ;
 }
else
{
  if (condition3)
      statement-block3 ;
  else
      statement-block4 ;
 }
```

In the above syntax,

i. If 'condition1' and 'condition2' are true then stament-block1 is executed
ii. If 'condition1' is true and 'condition2' is false then stament-block2 is executed
iii. If 'condition1' is false and 'condition3' is true then stament-block3 is executed
iv. If 'condition1' and 'condition3' are false then stament-block4 is executed




**Ex:**

```
Class  Nesting
{
public static void main(String args[])
{
int a=3,b=7,c=4;
        Sytem.out.print("Larget  value is : ");
        if( a>b)
         {
            if(a>c)
           System.out.println(a);
           else
           System.out.println(b);
         }
        else
         {
           if (b>c)
           System.out.println(b);
           else
           System.out.println(c);
         }
       }
       }
```

**Switch Statement:**

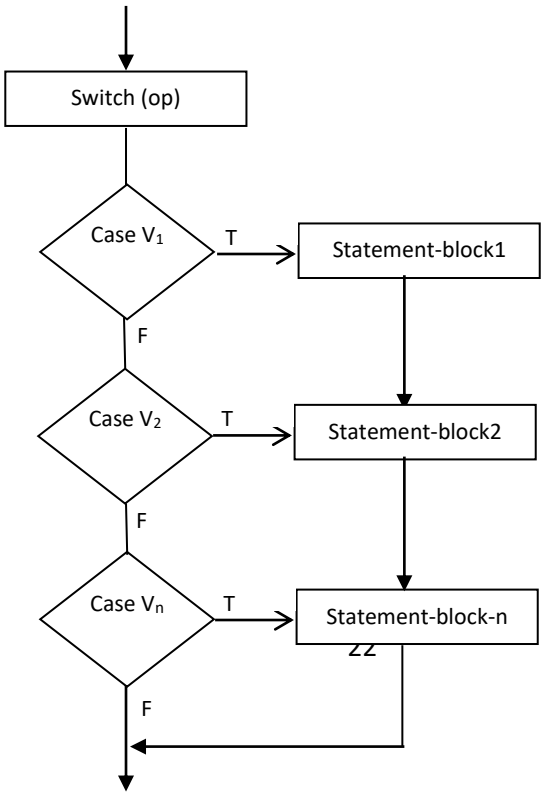
The 'switch' statement is a multiple condition statement. It can be used to make a decision from the number of choices. It is an extension of 'if…else' statement. The general format of switch statement is as follows:

**Syntax:**

```
switch (op)
{
case v1 : statement-block1;
          [break;]
case v2 : statement-block2;
          [break;]
     :         :
     :         :
case vn : statement-block-n;
          [break;]
[default : statement-block-
```

Flowchart Without break

Switch without break:



In the above syntax, 'op' is an integer or character.

$V_1$, $V_2$, …, $V_n$ are the constants that represent the value of 'op'. These are the constants of either integer or character type.

**Working:**

The switch statement searches for a match of switch variable (op) and case constants (v1, v2, …, vn). If a value matches to the 'op', then the switch statement executes all the statements in that case-block.

If there is 'break' statement is used in that block then it terminates the execution of switch statement. If 'break' is not present, then all the statements of remaining blocks are executed.

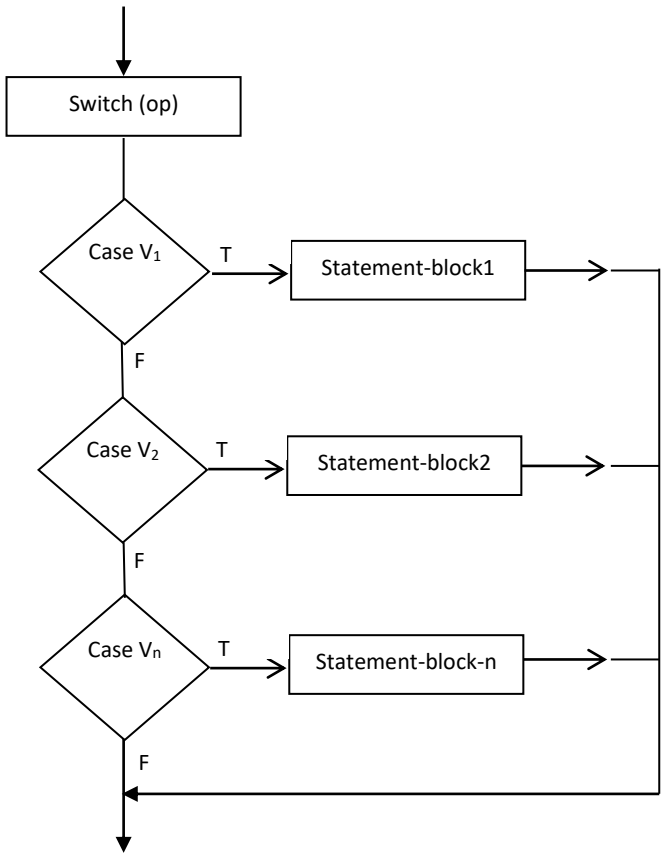
**Example :**

```
Class   Example
{
public static void main(String args[])
   {
   int op=3;
   switch (op)
   {
   case 1: System.out.println("Sunday");
           break;
   case 2: System.out.println("Monday");
           break;
   case 3: System.out.println( "Tuesday");
           break;
 case 4: System.out.println( "wednesday");
           break;
   case 5: System.out.println("Thursday");
           break;
   case 6: System.out.println("Friday");
           break;
   case 7: System.out.println("Saturday");
           break;
 default:System.out.println("WrongValue");
   }
   }
   }
```

Flowchart With break

Switch without break:

Switch with break:




## REPETITIVE (OR) LOOPING (OR) ITERATIVE CONTROL STRUCTURES

Every computer language must have features that instruct a computer to perform repetitive tasks. The process of repeated execution of a block of statements is known as "looping". The statement in the block may be executed any number of times from zero to infinite number. If a loop continues forever, it is called as an "infinite loop".

Depending on the position of the control statement in the loop, a control structure may be classified either as "entry control loop" or "exit control loop".

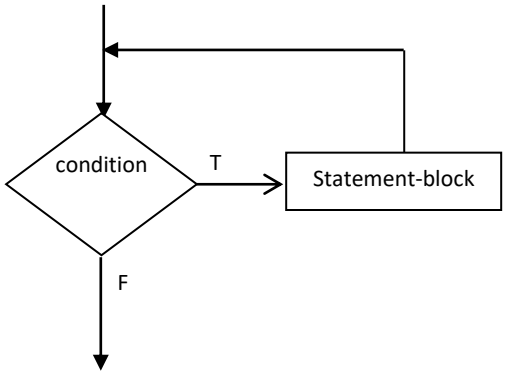
The repetitive control statements are used to execute a group of statements repeatedly for a specified number of times. The repetitive control statements are also called Iterative (or) Looping control statements.

In Java, there are 3 types of repetitive control structures. They are

1.While Statement
2.Do…while Statement
3.For Statement

**While Statement (while loop):**

This statement is used to execute one or more statements repeatedly as long as the specified condition is satisfied. It is an entry control loop structure.




**Syntax:**

```
while (condition)
{
   statement-block;
}
```

In the above syntax,

1. The 'condition' is a relational expression that must be enclosed within parenthesis.
2. The statement-block specifies one or more statements. When there is more than one statement, they must be enclosed in braces ({}).

**Working:** The 'while' statement executes all the statements in the statement-block as long as the condition returns true (non-zero) value. When the condition returns false (0) then the loop is terminated and the control is transferred to the statement after the statement-block.

**Ex :**
```
class ExWhile
{
public static void main(String args[ ])
{
        int i=1;
                while(i<=100)
                {
System.out.print(i);
++i;
}
}
}
```

**o/p:** 1 2 3 4 …….100

**'do…While' Statement:**

This statement is used to execute one or more statements repeatedly as long as the specified condition is satisfied. It is an exit control loop structure. The do…while structure executes the statement part at least one time either the condition is true or false.

**Syntax:**

```
do
{
   statement-block;
} while (condition) ;
```

**Example :**

```
class ExDoWhile
{
        public static void main(String args[ ])
        {
        int i=1;
                do
        {
        System.out.print(i);
        ++i;
        } while(i<=100);
        }
}
```

**o/p:** 1 2 3 4 …….100

**'for' Statement** :

This statement is used to execute a set of statements repeatedly as long as the specified condition is satisfied. It is an entry control loop structure.

**Syntax:**

```
for(initialisation ; condition ; increment/decrement)
{
   statement-block;
}
```

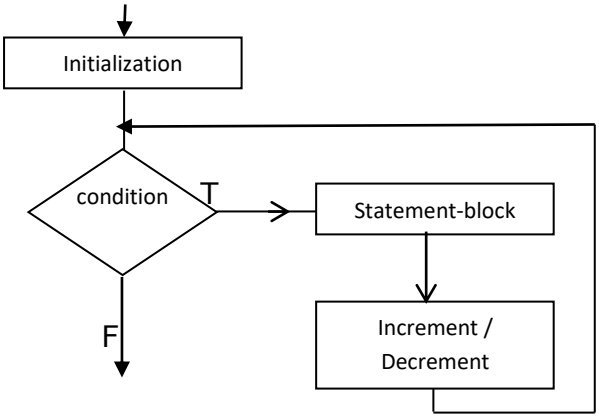
It evaluates the initialization part first and then it checks the condition. If the condition is true then it executes all the statements in the statement-block. After executing all the statements, increment/decrement part is executed. This looping process is continued as long as condition is true. When the condition becomes false, the loop is terminated.

**Example**

```
class ExFor
{
        public static void main(String args[ ])
        {
        int i;
                for(i=1 ; i<=100 ; i++)
        System.out.print(i);
        }
}
```

**o/p:** 1 2 3 4 …….100

Flowchart:




**The Enhanced for loop :**

The enhanced **for** loop is also called **for each loop,** is an extended language feature introduced with J2SE 5.0. This feature is used to retrieve the array elements efficiently rather than using array indexes.

**Syntax:**

```
for( type identifier :  Array Name )
{
   statement-block;
}
```

In the above example "**type**" represents the data type, **identifier** represents the name of the variable, Array Name indicates an Array.

**Example :**

```
class EachFor
{
        public static void main(String args[ ])
        {
        int a[ ]={7 , 15 , 12 , 8 , 4};
                for(int x : a)
        {
        System.out.print ( x );
        }
        }
}
```

**o/p:** 7 15 12 8 4

## Jumps in loops / UNCONDITIONAL CONTROL STRUCTURES

The unconditional control statements are used to change the execution sequence of a program by transferring the control from one place to another place within a program without using any condition.

The following are the unconditional control structures used in Java language.

break
continue
return

**break Statement:**

The break statement can be used in three ways.

1) break is used inside a loop to come out of it.
2) break is used inside the switch statement to come out of the block.
3) break is used in nested blocks to go to the end of the block.

**Syntax:**

**break ; (or) break label;**

**Example:**

```
class ExBreak
{
public static void main(String args [ ])
{
        int i;
        for(i=1;i<=100;i++)
        {
                System.out.print( i );
                if(i==50)
                   break;
        }
}
}       o/p: 1 2 3 4 …….50
```

```
class ExBreak
{
public static void main(String args [ ])
{
        int i, j;
   outer : for(i=1;i<=100;i++)
         {
   Inner : for (j=1; j<=100; j++)
           {
             System.out.print( j );
            if(j==50)
            break outer;
             }
        }
}
}
```

**Continue Statement:**

Continue statement is used inside a loop to repeat the next iteration of the loop. When continue statement is executed, the next statements in the loop are not executed and control of execution goes back to the next repetition of the loop.

**Syntax:**

**Continue ; (or) continue label ;**

**Example:**

```
class ExConinue
{
public static void main(String args [ ])
{
        int i;
        for(i=1 ; i<=10; i++)
        {
        if(i==4 || i==7)
          continue;
          System.out.print( i );

        }
}
}       o/p: 1 2 3 5 6 8 9 10
```

```
class ExContinue
{
public static void main(String args [ ])
{
        int i, j;
   outer : for(i=1; i<=10; i++)
         {
   Inner : for (j=1; j<=10; j++)
            {
        if(j==4 || j==7)
          continue outer;
          System.out.print( i );
               }
        }
}
}
```

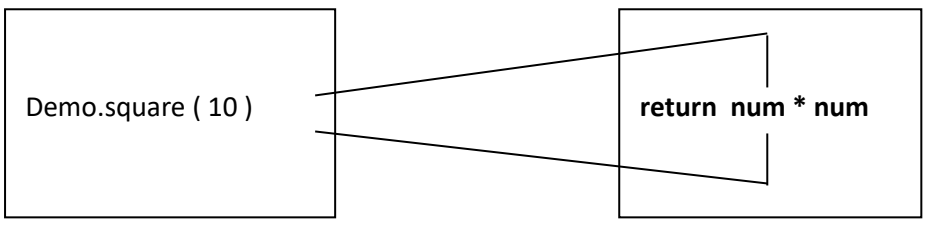
**return Statement :**

return statement is used in a method to come out of it to the calling method. return statement is often used in the methods.

**Examples :**

```
return  (x+y);  //calculates x+y and return that value
return -5;       // return -5 to the calling method.
```

```
class Demo
{
        public static void main( String args [ ] )
        {
        int res=Demo.square(10);
        System.out.println("Result = " + res);
        }
        static  int square(int num)
        {
                return  num * num;
        }
}   OP : Result = 100
```



# UNIT-2
# CLASSES, OBJECTS AND METHODS

Java is true object oriented language and therefore the underlying structure of all java programs is classes. Anything we wish to represent in a java program must be encapsulated in a class. That defines the' state' and 'behavior' of the basic program concept known as "objects". Classes create objects, and objects use methods to communicate between them.

Classes provide convenient methods for packing together a group of logically related data items and functions that work on them. In java, the data items are called fields and the "functions" are called "methods"

**Class :**

A class is a collection of related objects that share common properties and actions. Classes are used to pack a group of data items and functions. In java, the data items are called **fields** and functions are called **methods**.

**Defining a class :**

Once a class is defined, we can create any number of objects belonging to that class. In java these "variable" are called as *instances* of classes. Classes are defined using the keyword "class".

**Syntax :**

```
class classname [extends superclassname]
{
        [fields declaration ; ]
        [methods declaration ;]
}
```

**Fields Declaration :**

The variables which are declared inside a class are called ***fields.*** These variables are also called ***instance variables*** because they are created whenever an object of the class is instantiated.

Ex:
```
class Rectangle
{
        int length;
        int width;
}
```

The class Rectangle contains two integer type instance variables. No memory space is reserved for these instance variables.

**Methods Declaration :**

Methods are necessary for manipulating the data contained in the class. Methods are declared and defined inside the body of the class immediately after the declaration of instance variables.

Syntax :

```
Returntype  Methodname (Parameter – list)
{
        Method Body;
}
```

Methods declaration have four basic parts

1. The name of the method (method name)
2. The type of the value that the method returns (return type)
3. A list of parameters (Parameter – list)

4. The body of the method.

**Ex :** class Rectangle

```
{
        int length;
        int width;
        void getdata ( int x, int y )
        {
                length = x;
                width = y;
        }
}
```

**Objects :**

An object is a block of memory that contains space to store all the instance variables.

**Creating Objects :**

- Creating an object is also referred as *Instantiating* an object.
- In java objects are created using the operator **new.**
- The **new** operator creates an object of the specified class and returns a reference to that object.

**Syntax 1:**

```
Classname objname;
objname  = new   Classname ( );
```

**Syntax 2:**

```
Classname   objname = new   Classname ( );
```

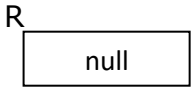
**Example :**

```
Rectangle  r;                // Declares the object
r  = new  Rectangle ( );     // instantiate the object
```

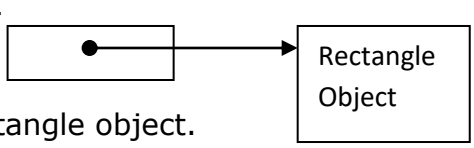
The above two statements can also be combined as

```
Rectangle  r = new Rectangle ( );
```

- Rectangle r;




- r = **new** Rectangle ( );




now **r** is a reference to Rectangle object.

**Accessing Class Members :**

We can not access the instance variables and methods directly from outside the class. To access them from outside the class, we must use the concerned object and the dot (.) operator.

**Syntax :**

```
Objectname.variablename = value;
Objectname.methodname(parameter-list);
```

Here objectname is the name of the object, variablename is the name of the instance variable inside the object.

**Example :**

```
class  A
{
int  x,y;
        void getData(int m,int n)
        {
        x=m;
        y=n;
        }
        void disp( )
        {
        System.out.println(x);
        System.out.println(y);
        }
}
```

```
class Ex
{
public static void main(String args[])
{
A  a=new A( );
a.getData(10,20);
a.disp( );
}
}
```

**O/P :** 10
20

Ex: program --2

```
class Rectangle
{
int length,width;

void getData(int x,int y)
{
length=x;
width=y;
}
int rectArea()
{
int area=length*width;
return area;
}
}

class Rect
{
public static void main(String args[])
{
int area1,area2;
Rectangle rect1=new Rectangle();
Rectangle rect2=new Rectangle();

rect1.length=10;
rect1.width=50;
area1=rect1.length*rect1.width;

rect2.getData(20,12);
area2=rect2.rectArea();
System.out.println("area1="+area1);
System.out.println("area2="+area2);
```

```
}
}
```

## CONSTRUCTORS

When we create a new instance of a class using 'new' keyword, a constructor for that class is called. A constructor is similar to a method that is used to initialize the instance variables. The purpose of constructor is to initialize the instance variables.

If you donot define a constructor for a class, a default parameterless constructor is automatically created by the compiler. The default constructor calls the default parent constructor and initializes all instance variables to default values(zero for numeric,null for object references and false for boolean) .Default constructor is created only if there is no constructor. If we define any constructor for our class, no default constructor is automatically created.
A constructor has the following characteristics:

1. The constructor's name and class name should be same.
2. A constructor does not return any value ,not even void this is because they return the instance of the class itself.
3. A constructor may have or may not have parameters. If a constructor does not have any parameters it is called 'Default constructor'. If a constructor has any parameters it is called 'parameterized constructor'.
4. A constructor is automatically called and executed at the time of creating an object. While creating an object, if nothing is passed to the object, the default constructor is called and executed, if some values are passed to the object, then the parameterized constructor is called

Ex. Program:

```
class B
{
int x;
B(int n)
{
x=n;
System.out.println("constructor 'B(int n)' called !");
}
}

public class Cons
{
public static void main(String args[])
{
B b=new B(10);

System.out.println("Sended value is ="+b.x);
}
}
```

## METHOD OVERLOADING

Java supports to create methods that have the same name, but different parameters lists and different definitions. This is called method overloading. Method overloading is used when objects are required to perform similar tasks but using different input parameters. When we call a method in an object, java matches up the method name first and then the number and type of parameters

to decide which one of the definitions to execute. This process is also known as polymorphism.

```
class Room
{
float length,breadth;
Room(float x)
{
length=breadth=x;
}

Room(float x,float y)
{
length= x;
breadth=y;
}

        float area()
        {

        return (length*breadth);
        }
}
class RoomArea
{
public static void main(String args[])
{
Room room1=new Room(25.0f,15.0f);
float roomArea=room1.area();
System.out.println(roomArea);
Room room2=new Room(15.0f);
roomArea=room2.area();
System.out.println(roomArea);
}
}
```