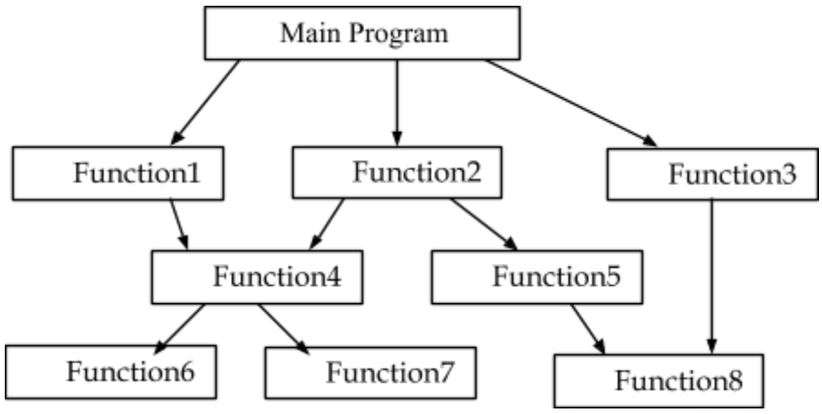
UNIT-I

FUNDAMENTALS OF OBJECT – ORIENTED PROGRAMMING

Procedure-oriented programming & Object-oriented programming

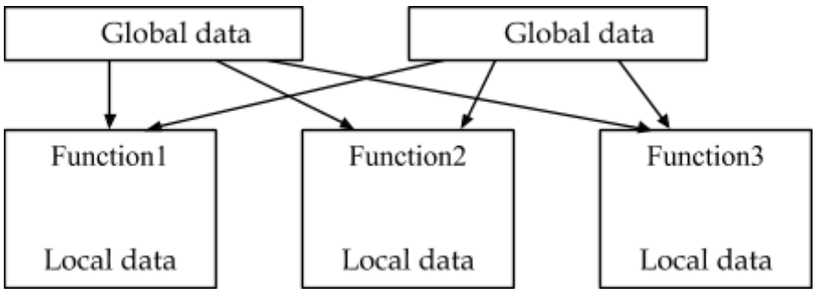
Procedure-oriented programming: Conventional programming, using high level language such as COBOL, FORTRAN and C, is commonly known as procedure-oriented programming. In the procedure oriented approach, the problem is viewed as a sequence of things to be done such as reading, calculating and printing. A number of functions are written to accomplish these tasks. The primary focus is on functions. The technique of hierarchical decomposition has been used to specify the tasks to be completed for solving a problem.

Procedure-oriented programming basically consists of writing a list of instructions for the computer to follow, and organizing these instructions into groups known as functions. We normally use a flow chart to organize these actions and represent the flow of control from one action to another. A program structure for procedure oriented program is as below



While we concentrate on the development of functions, very little attention is given to the data that are being used by various functions.

In a multi-function program, many important data items are placed as global so that they may be accessed by all the functions. Each function may have its own local data. The following figure shows relationship between data and functions in a procedure-oriented program.



Global data are more vulnerable to an inadvertent change by a function. In a large program it is very difficult to identify what data is used by which function. In case we need to revise an external data structure, we also need to revise all functions that access the data. This provides an opportunity for bugs to creep in.

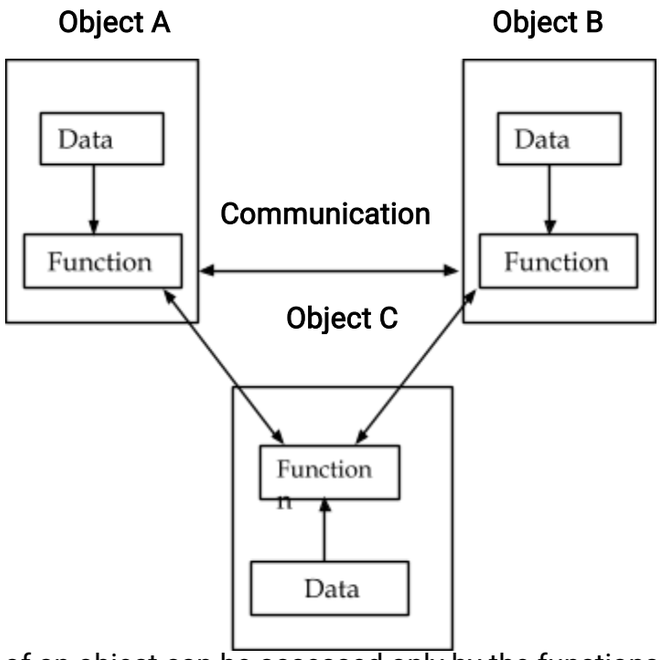
Another serious drawback with the procedural approach is that it does not model real world problems very well. This is because functions are action-oriented and do not really corresponding to the elements of the problem.

Characteristics of procedure-oriented programming:

- Emphasis is one doing things.
- Large programs are divided into smaller programs known as functions.
- Most of the functions share global data.

- Data move openly around the system from function to function.
- Functions transform data from one form to another.
- Employs top-down approach in program design.

Object-oriented Programming: The major motivating factor in the invention of object-oriented approach is to remove some of the flaws encountered in the procedural approach. OOP treats data as a critical element in the program development and does not allow it to flow freely around the system. It ties data more closely to the functions that operate on it, and protects it from accidental modification from outside functions. OOP allows decomposition of a problem into a number of entities called objects and then builds data and functions around these objects.



The data of an object can be accessed only by the functions associated with that object. However, functions of one object can access the functions of other objects.

Features of object-oriented programming:

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

Object-oriented programming is the most recent concept among programming paradigms. “Object-oriented programming is an approach that provides a way of modularizing programs by creating partitioned memory area for both data and functions that can be used as templates for creating copies of such modules on demand.

Differences between Procedural oriented and Object oriented programming:

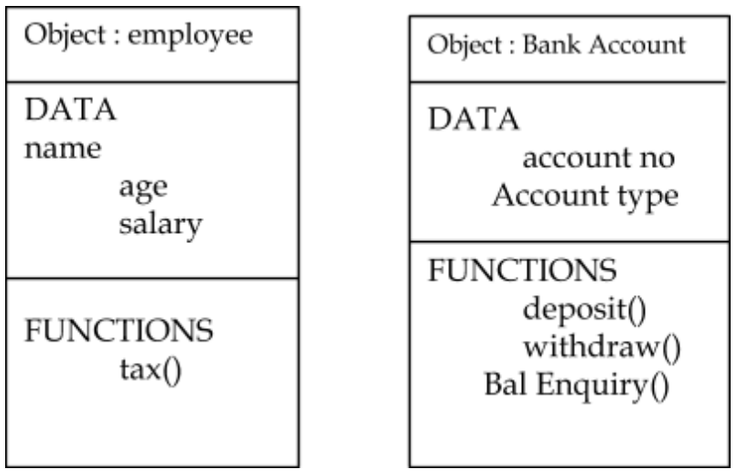
Procedural oriented programming	Object Oriented Programming
It is known as POP	It is known as OOP
It deals with algorithms	It deals with data
Programs are divided into functions	Programs are divided into objects
Most of the functions share global data	Data Structure characterizes objects
Data move from function to function	Functions that operate on data are bind to form classes
Functions are responsible for transforming from one form to another	Data is hidden cannot be accessed by outside functions
It is top down approach	It is Bottom Up approach
It needs very less memory	It needs more memory that POP
Example:- C, Fortran	Example:- C++, JAVA,.NET
It do not have any access specifiers	It has access specifiers like private, public and protected.
It is less secure	It is more secure
It follows no overloading	It follows operator overloading and function overloading

RINCIPLES OF OBJECT-ORIENTED PROGRAMMING:

Object:

objects are basic run time entities represent a person, place, bank account or any item that the program has to handle. They may also represent user defined data type such as vectors, lists. Objects take up space in the memory and have an associated address like structure in C.

Each object contains data and code to manipulate the data. When a program is executed, objects interact by sending messages to one another. For ex: if customer and account are two objects in a program the customer object may send a message to the account object requesting for the bank balance.



Class: we just mentioned that object contains data and code to manipulate data. The entire set of data and code of an object can be made, a user-defined data type with the help of a class. Objects are variables of the type class.

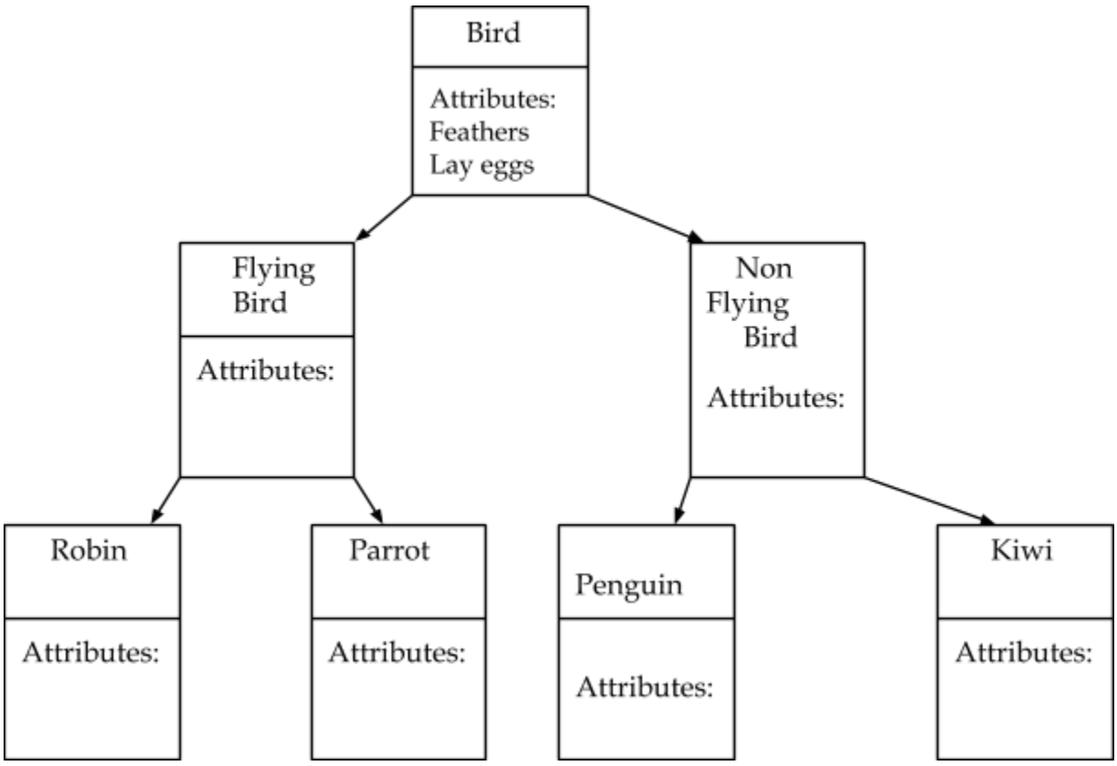
- ✔ Class is keyword
- ✔ Class is way to bind the data and its related functions
- ✔ A class contain any number of objects belong to that class
- ✔ **A class is thus collection of objects of similar type.**

For example apple, orange are objects of class fruit.

Encapsulation : The wrapping up of data and functions in to a single unit is known as encapsulation. Data encapsulation is the most strictly feature of a class. The data is not accessible to outside world, and only those functions, which are wrapped in the class, can access it. This process is called “data hiding” or “information hiding”.

Data Abstraction:
Creating new data types using encapsulated items that are well suited to an application to be programmed is called data abstraction. The data types created by data abstraction process are known as Abstract data types (ADT).

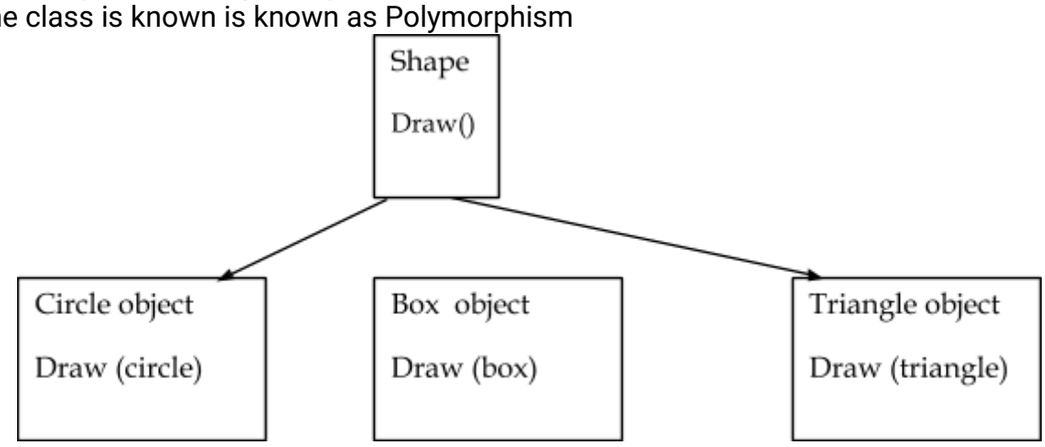
Inheritance: inheritance is the process by which objects of one class acquire the properties of objects of another class. (or) Creation of new class from existing class called inheritance.



In OOP, the concept of inheritance provides idea of reusability. This means that we can add additional features to an existing class without modifying it. This is possible by deriving a new class from existing one. The new class will have combined features of both the classes.

Polymorphism: polymorphism means the ability to take more than one form.

The concept of defining multiple functions with the same name is known polymorphism.
(or)
The concept of defining multiple functions with the same name associated with the object of same class is known is known as Polymorphism



The above figure tells single function name can be used to handle different number and different types of arguments.

This is similar that a word containing more than one meaning. Using single function name to perform different types of tasks is called function overloading.

Benefits of OOPs:

1. Through inheritance we can eliminate redundant code and extent use of existing classes
2. data hiding helps the programmer to build secure programs
3. it is easy to portion the work in project based objects
4. object oriented systems can be easily upgraded from small to large systems
5. Software complexity can be easily managed.

Applications of OOPs:

1. Object Oriented databases
2. Artificial Intelligence and Expert Systems
3. Neural Networks
4. Web Designing
5. Simulation and Modeling
6. CAD/CAM Systems
7. Client server

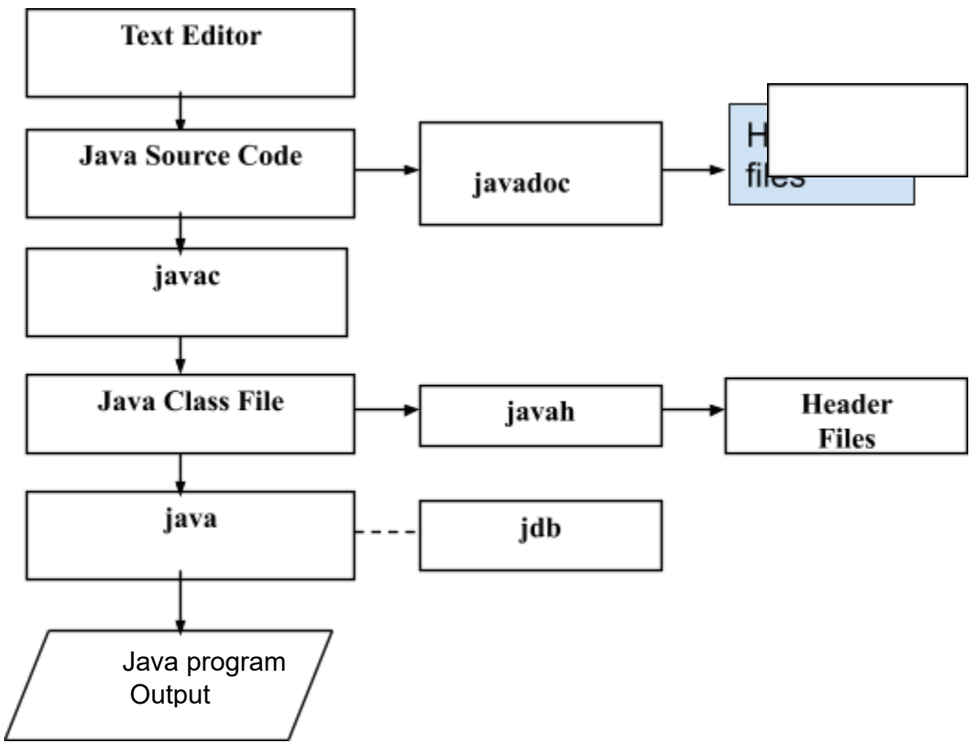
systems Java Development

Kit (JDK):

The Java Development Kit comes with a collection of development tools that are used for developing and running Java programs. They include

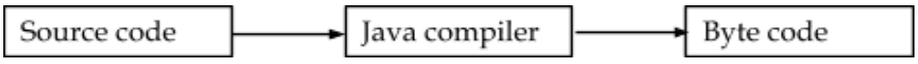
- appletviewer - Enables us to run Java applets.
- java - Java interpreter, which runs application by reading and interpreting byte code files.
- javac - The Java compiler that converts java source code to byte code files that the interpreter can understand
- javadoc -creates HTML format documentation from Java source code files.
- javah - produces header files for use with the native methods.
- javap - Java dissembler, which enables us to convert byte code files into Program description.
- jdb - Java debugger, which helps us to find errors in our program

Figure below shows the process of building and running of java application programs

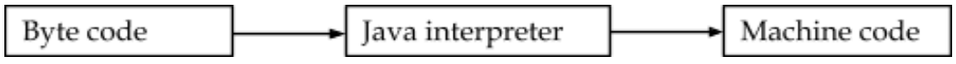


Java Virtual Machine (JVM):

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



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



Java Features:

Features of language are nothing but the services or functionalities provided by language Developers to the industry programs
Sun Microsystems provided 12 features in java

They are

- simple
- Plat form Independent
- High performance
- Highly interpreted
- Robust
- Dynamic
- Secured

- portable
- Multithreaded
- Object oriented
- Distributed
- Architecture neutral

Simple:

Java is one of the simple programming language.java programming eliminates complex Concept called pointers so that application development time is less and application execution time is less because of magic of byte code

Platform Independent:

Java is a platform independent language. Once we create the program in one operating system, that program will be worked on any other operating system.

High performance:

Java is one of the high performance language because of following factors

1. Magic of byte code
2. Automatic memory management

The garbage collector collects unused memory space for improving performance of java

Highly interpreted:

In the later versions of java sun developers has developed a program called JIT (just in time compiler) and added as a part of JVM whose role is speed up interpretation phase by reading Entire section of byte code and converting into native understandable os

In the current versions of java interpretation phase is very fast even compare with compilation phase due to this sun developers has populated java is one of the highly interpreted language

Robust:

Java is one of the strong programming languages because it contains facility called Exception handling. Whereas java whose applications are treated as strong because java contain error handling facility called Exception handling

Dynamic:

The concept of allocating the memory loading the functionalities and data to the ram dynamically at the runtime as and when under the program is execution

Object oriented programming languages like java working based on the concept of dynamic.

Secured:

Security is one of the principle used in industry projects for protecting confidentional information from unauthorized users.java is high secured programming language because the concept of Encapsulation

Security becomes an important issue for a language that is used for programming on Internet. Threat of viruses and abuse of resources is everywhere. Java systems not only verify all memory access but also ensure that no viruses are communicated with an applet. The absence of pointers in Java ensures that programs cannot gain access to memory locations with out proper authorization.

Architectural neutral:

A language whose applications are said to be Architectural neutral if they runs on every processor without considering their Architecture vendors

Portable:

A portable technology and whose applications run of every os and every processor without considering their vendors

Portability=platform independent + Architectural neutral

Distributed:

Java is designed as a distributed language for creating applications on networks. It has the ability to share both data programs. Java applications can open and access remote objects on Internet as easily as they can do in a local system. This enables multiple programmers at multiple remote locations to collaborate and work together on a single project.

Multi-threaded:

A thread is like a separate program, executing concurrently. We can write Java programs that deal with many tasks at once by defining multiple threads. The main advantage of multi-threading is that it shares the same memory. Threads are important for multi-media, web applications and so on..

Object-Oriented:

Java is a true object-oriented language. Almost everything in Java is an object. All program code and data reside within objects and classes. Java comes with an extensive set of classes, arranged in packages that we can use in our programs by inheritance. Java supports OOP concepts polymorphism, data encapsulation, and data abstraction.

History of java:

Java is one of the programming language used in industry for each and every application development it becomes more popular for developing distributed applications
The java software developed at sun micro systems under the guidance of James gosling and others.

The Java software started its development in early of the year 1990 & released to the industry in middle of the year 1991 on the name of OAK, which is one of the original name of Java & scientifically it, is one of the “Tree name” and OAK has taken 18 months to develop.

The software OAK is popular for developing software for electronic, mechanical and automobile devices. In other words the OAK was able to fulfill some other requirements of industry. This is an industry opinion.

Sun micro systems developers under the guidance of games gosling the software OAK was revised and released to the industry in the year 1995 on the name of “java”

The Java software is available in the real industry `in 3 categories.
They are:
1. J2SE(Java Standard Edition)
2.J2EE(Java Enterprise Edition)
3.J2ME(Java Micro/Mobile edition)

In the industry J2SE is used for developing client side applications. J2EE is used for developing server side applications and J2ME is used for mobile/wireless based applications.

The following table gives Java technology version and Java lang version.

Java Tech.version	Java Lang.version
JAVA	JDK1.0 JDK1.1
JAVA2	JDK1.2 JDK1.3 JDK1.4
JAVA5	JDK1.5
JAVA6	JDK1.6
JAVA7	JDK1.7

In the current industry J2SE must be called as JSE or JAVASE. J2EE must be called as JEE or JAVAEE and J2ME must be called as JME or JAVAME.

Java is one of the open source code and it can be downloaded freely from www.sun.com and www.oracle.com.

Java Program structure:

Documentation Section
Package Statement
Import Statement
Interface Statement
Class Definitions

Main Method Class
{
Main Method Definition
}

Documentation Section:

The documentation section comprises of comment lines giving the name of the program, the author and other details. Java supports three types of comments.

Type	Usage
/* comment */	All characters between /* and */ are ignored.
// comment	All characters after the //, up to the end of the line are ignored.
/** comment */	Same as /* */, except that the comment can be used with the javadoc tool to create automatic

Package statement:

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

Example: package student;

The package statement is optional. That is our classes do not have to be part of a package.

import Statement:

The statement after the package statement and class definition may be number of import Statements. This is similar to the #include statement in C.

Example: import student. test;

This statement instructs the interpreter to load the test class contained in the package student. Using import statement we can have access to classes that are part of another package.

Interface Statements:

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

Class Definitions:

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

Main Method class:

Since every Java stand alone program requires a main method as its starting point, this class is the essential part of the Java program. A simple Java program may contain only this part. The main method creates objects of various objects of various classes and establishes

communication between them. On reaching the end of main, the program terminates and the control passes back to the operating system.

Simple java program:

```
Class sample
{
Public static void main(String args[])
{
        System.out.println("java is better than c++:");
}
}
```

The first line **class Sample** declares a class. A class is a keyword and declares that a new class definition follows.
very class definition in java begins with { and ends with }.

The third line **public static void main(String args[])** defines a method named main. Conceptually this is similar to the **main()** function in C/C++. this line contains three keywords **public, static** and **void**.

public: The keyword public is an access specifier that declares the main method as unprotected and accessible to all other classes. This is similar to c++ public modifier.

static: static means this method as one that belongs to the entire class and not a part of any objects of the class. The main must always used as static since the interpreter uses this method before any objects are created.

void: void states that the main method does not return any value.

The only executable statement in the program
is System.out.println("java is better than c++");

This is similar to printf() statement of C or cout<< construct of c++. Since java is a true object oriented language, every method must be part of object. The println method is a member of the out object, which is a static data member of system class. This line prints the string java is better than c++.

Variables:

A variable is an identifier that denotes the storage location to store a data value. A variable name may consist of alphabets, digits, the underscore (_) and dollar characters subjects to the following rules.

- A variable name must be begin with a letter, a dollar ($) symbol or an underscore (_) followed by a sequence of letters.
- Variable name for a specific program must be unique.
- There should be no white spaces in between any two characters of an variable name
- Key words cannot be used for naming the variable
- Variable names can be of any length.

Declaration of variables:

All variables must be declared before they can be used. Variable declaration does three things.

- It tells the compiler what the name of the variable is.
- It specifies what type of data the variable will hold
- The variable declared anywhere in the program.
- The place of the declaration of the variable (in the program) decides the scope of the variable.

The basic declaration of the variable is as follows:

Syntax: datatype variablename;

Where type describes the data type of the variable and identifier describes the name of the variable.

Example: int a;

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

Syntax: **data**type variable1,variable2,…….,variablen;

Example: int a,b,c;

Giving values to variables:

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

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

Assignment statement:

A simple method of giving value to a variable is through the assignment as follows.

```
type variable_name=value
```

For example
int number=10;
float height=23.6f;
String name=”abc”

Java Tokens:

A Java Program is basically collection of classes. A class is defined by b y a set of statements and methods containing tokens. A token is a predefined statement in Java.In simple terms a Java program is 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 feature of the language. Java language has reserved 60 words as keywords. These keywords combine with operators and separators according to syntax, form definition of the Java language. Since keywords have specific meaning in Java we can not use them as names as variables, classes and methods and so on. All keywords are to be written in lower-case letters

Although Java has been modeled on C and C++ languages, Java does not use many of C/C++ keywords and add some new keywords to implement new features of the language. Some of the Java keyword are

abstract	double	int	super
boolean	else	interface	switch
break	extend	long	Synchronized
byte	false	native	This

by value	final	new	thread
case	finally	null	throw
catch	float	package	transient
char	for	private	true
class	protected	try	short
const	if	public	void
continue	implements	return	while
default	import		

Identifiers:

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

1. They can have alphabets, digits and the underscore and dollar sign character.
2. They must not begin with a digit.
3. Uppercase and lowercase letters are distinct.
4. They can be of any length.

In addition to the above rules to create Identifiers it is better to use the following conventions.

- Names of all public methods and instance variables start with a leading lower case letter.
  Example: average, sum
- When more than one word are used in a name, the second and subsequent words are marked with a leading upper case letters.
  Example: dayTemperature, firstDayOfMonth, totalMarks.
- All private and local variables use only lower case letters combined with underscores
  Example: length, batch_strength.
- All classes and interfaces start with a leading uppercase letter (and subsequent word with a leading uppercase letter).
  Example: Student, Hello Java, Vehicle, MotorCycle.
- Variables that represent constant values use all uppercase letters and underscores between words.
  Example: TOTAL, F_MAX, PRINCIPAL_AMOUNT.

Literals:

Literals in Java are a sequence of characters (digits, letters,and other characters) that represent constant values to be stored in variables. Java language 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 stored and how they behaved.

Operators:

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

- Arithmetic operators
- Relational operators
- Logical operators
- Assignment operators
- Increment and decrement operators
- Conditional operators
- Bit wise operators
- Special operators

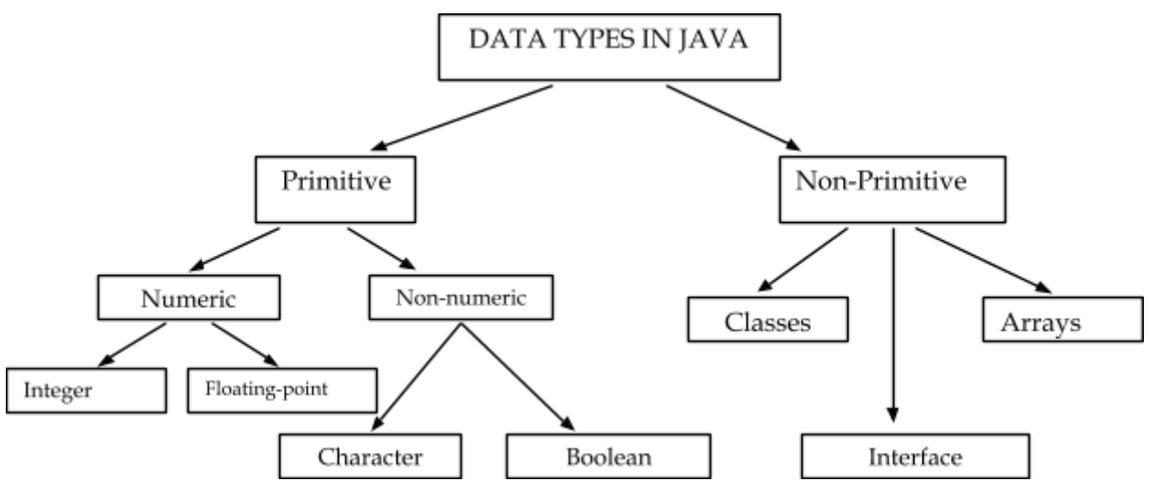
Separators:

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

- Parentheses: It is represented with the symbol **( )** . It is used to enclose parameters in method definition and invocation, also used for defining precedence in expressions, containing expressions for flow control and surrounding cast types.
- Braces: It is represented with the symbol **{ }**.It used store array values and to define block of code for classes, methods.
- Brackets: It is represented with **[ ]** and used to declare array types and for dereferencing array values.
- Semicolon: It is represented with the symbol ; and used to separate statements.
- Comma: It is represented with symbol **','** and used to separate variables, objects.
- Period: It is represented with the symbol **'.'** and used separate package names from sub packages and classes;

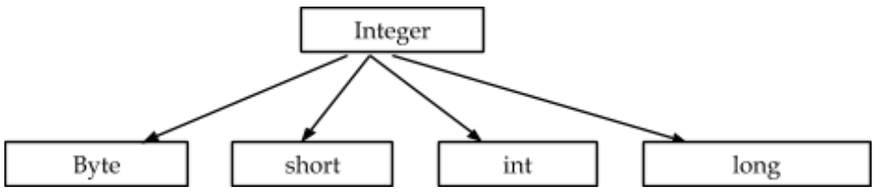
Data Types:

Every variable in java has a data type. Data type specifies the size and type of values that can be stored. Basically Java has two types of data types. They are Primitive and non-primitive. Figure below shows the various categories of 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 as shown below table. Java does not supports the concept of unsigned types and therefore all Java values are signed meaning they can be positive or negative.



The below table shows the size and range of Integer 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

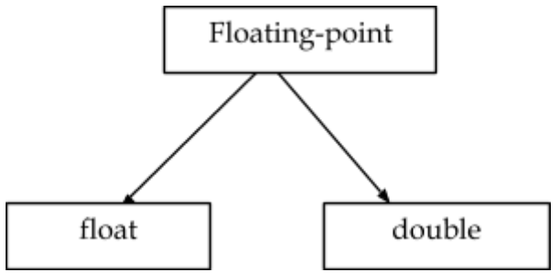
It should be advised that wider data types require more time for manipulation and therefore it is advisable to use smaller data types, where ever possible.

We can make integers **long** by appending the letter L or l at the end of the number.

Example: 123L or 123l

Floating-point types:

Integer types can hold only whole numbers and therefore we use another type known as floating-point type to hold numbers containing fractional parts such as 27.59 and –1.375There are two types floating-point storage in java as shown below



The **float** type values are single precision numbers while **double** types are double precision numbers. Table below gives the size and range of these two types.

Type	Size	Maximum value	Minimum value
Float	4 bytes	3.4e-0.38	3.4e+0.38
Double	8 bytes	1.7e-308	1.7e+308

Floating point numbers are treated as double precision quantities .To force them to be in single precision mode ,we must append F or f to the numbers.

Example: 1.2f, 7.3456F

Double precision types are used when we need greater precision in storage of floating point numbers. All mathematical functions, such as sin, cos and sqrt return double type values.

Character Type:

In order to store character constants in memory. Java provides a character type called **char** .The character size assumes a size of 2 bytes and it can hold a single character.

Example: char ch='a';

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. They are **true** and **false.**

Remember that both these words have been declared as key words. Boolean type is denoted by the keyword **Boolean** and uses only one bit of storage.

Example: boolean ismanager=false;

Type casting:

The process of converting one data type to another is called casting.

Syntax is

```
Type variable1=(type) variable2
```

Ex: int m=50; byte
n=(byte)m;

Four integer types can be cast to any other type except Boolean. Casting into a smaller type may result in a loss of data. Similarly float and double can be cast to any other type except Boolean.

The following table shows casting with no loss of information

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

Automatic conversion:

For some types it is possible to assign a value of one type to other without cast. Java does the conversion automatically. This is known as *automatic* type conversion. Automatic conversion is possible only if the destination type has enough precision to store the source value. For example int is large enough to store byte value.

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

These are valid statements. The process of assigning smaller type to a larger one is known as *promotion* and assigning a larger type to a smaller one is known as *narrowing*. Note that narrowing may result loss of information.

Standard default values:

In java, every variable has a default value. If we don't initialize a variable when it is first created, Java provides default value to that variable type automatically as shown in the following table.

Type of variable	Default Value
byte	Zero : (byte)0
short	Zero : (short)0
int	Zero : 0
long	Zero : 0L
float	0.0f
double	0.0d
char	Null character
boolean	false
reference	null

Operators:

An operator is a symbol that tells the computer to perform certain mathematical or logical manipulations. Operators are used in programs to manipulate data and variables. They usually form a part of mathematical or logical expressions. Java operators can be classified into a number of related categories as below.

- Arithmetic operators
- Relational operators
- Logical operators
- Assignment operators
- Increment and decrement operators
- Conditional operators
- Bit wise operators
- Special operators

Arithmetic operators:

Arithmetic operators are used to construct mathematical expressions as in algebra. Java provides all basic arithmetic operators. These arithmetic operators can operate on ant built-in numeric data type of java. We cannot use these operators on Boolean type. Below table list the java arithmetic operators and their meaning.

Operator	Meaning
+	Addition or unary plus
-	Subtraction or unary minus
*	Multiplication
/	Division
%	Modulo division (Remainder)

Arithmetic operators are used as shown below:

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

Here a and b may be variables or constants are known as operands.

Integer Arithmetic:

When both the operands in a single arithmetic expression such as a+b are integers ,the expression called an integer expression ,and the operation is called Integer arithmetic.Integrer arithmetic always yields(produce)an integer value. In the above examples, if a and b are integers, then for a=14 and b=4 we have the following results.

a – b = 10
a + b = 1 8
a * b = 56
a / b = 3 (decimal part truncated)
a%b = 2 (remainder of integer division)

For modulo division, the sign of the result is always the sign of the first operand (the dividend). That is

-14 % 3 = -2
-14 % -3 = -2
 14 % -3 = 2

Real arithmetic:

An arithmetic operation involving only real operands is called real arithmetic. A real operand may assume values either in decimal or in exponential notation. Since floating point values are rounded to the number of significant digits permissible, the final value is an approximation of the correct result.

In java modulo operator % can be applied to the floating-point data as well. The floating-point modulus operator returns the floating-point equivalent of an integer division. Below program explains hoe arithmetic operators works on floating point values.

```
class FloatPoint
{
        public static void main(String args[ ] )
        {
                float a = 20.5f , b = 6.4f;
                System.out.println("a = " + a );
                System.out.println("b = " + b );
                System.out.println("a + b = " +( a + b) );
                System.out.println("a - b = " +( a - b) );
                System.out.println("a * b = " +( a * b) );
                System.out.println("a / b = " +( a / b) );
                System.out.println("a % b = " +( a % b) );
        }
}
```

The output of the program is as follows:

```
a = 20.5
b = 6.4
a + b
=26.9 a-b
= 14.1
a * b = 131.2
a / b =
3.20313 a % b
= 1.3
```

Mixed mode Arithmetic:

When one of the operand is real and the other is integer, the expression is called a mixed-mode arithmetic expression. If either operand is of the real type, then the other operand is converted to real and the real arithmetic is performed. Thus

15 / 10.0 produces the result
1.5 Where as
15 / 10 Produces the result 1

Relational Operators:

We often compare two quantities , and depending on their relation takes certain decisions. For example we may compare the age of two persons, or the price of two items, and so on. These comparisons can done with the help of relational operators. Java supports six types of relational operators. These operators and their meanings are shown in below table.

Operator	Meaning
<	Is less than
<=	Is less than or equal to
>	Is greater than
>=	Is
= =	Is equal to
!=	Is not equal to

A simple relational expression contains only one relational operator is of the following form:

ae-1 relational operator ae-2

ae-1 and ae-2 are arithmetic expressions, which may be simple constants, variables or combination of them. Table below shows some examples of relational expressions and their values.

Expression	Value(Result)
4.5<=10	True
4.5<-10	False
-35>=0	False
10<7+5	False
12>10	True
12<=10	False

Logical operators:

In addition to the relational operators java has three logical operators, which are given in the table below.

Operator	Meaning
&&	Logical And
||	Logical Or
!	Logical Not

The logical operators && and || are used when we want to form compound conditions by combining two or more relations. An example:

a > b && x==10

An expression of this kind which combines two or more relational expressions is termed as a logical expression or compound relational expression. Like the simple relational expressions a logical expression also yields a value of true or false, according to the truth table shown in figure. The logical expression given above is true only if both a>b && x= =10 is true. If either (or both) of them is false the expression is false.

Op-1	Op-2	Value of expression	
		op-1&&op-2	op-1||op-2
True	true	true	true
True	false	false	true
False	true	false	true
False	false	false	false

Assignment operator:

Assignment operators are used to assign the value of an expression to a variable. We have seen the usual assignment operator , ‘ = ‘. In addition , Java has a set of ‘ short hand’ operators which are used in the form.

v **op=** exp;

Where v is the variable , exp is an expression and op is a Java binary operator . The operator **op** = is known as the shorthand assignment operator. Commonly used short hand assignment operators are illustrated in the table below.

Statement with simple assignment operator	Statement with shorthand operator
a = a + 1	a += 1
a = a – 1	a -= 1
a = a * 1	a *= 1
a = a / 1	a /= 1

The use of short hand operators has three advantages:

- What appears on the left-hand side need not to be repeated and therefore it becomes easier to write
- The statement is more concise and easier to read
- Use of shorthand operator results in a more efficient code.

Increment and decrement operator:

Java has two very useful operators not generally found in many other languages. These are the increment and decrement operators. **+ + and - -**

The operator + + adds 1 to the operand while - - subtracts 1. Both the unary operators and are used in the following form.

+ + m or m + +
- -m or m- -

+ + m is equivalent to m = m + 1 or m + = 1
- - m is equivalent to m = m – 1 or m - =1

when + +m and m + + mean the same thing when they form statements independently , they behave differently when they are used in expressions on the right – hand side of an assignment statement .Consider the following

m = 5
y =+ + m

In this case the value of y and m would be 6. Suppose ,if we rewrite the above the statement as

m = 5
y = m + +;

Conditional operator :

It has the following form

exp1?exp2:exp3

where exp1,exp2,exp3 are expressions. exp1 is evaluated first , if it is true, then exp2 is evaluated , if exp1 is false then exp3 is evaluated.

Ex: a=10;
b=15;
x=(a>b)?a : b ;

here x is assigned the value of b.

Bit wise operators:

These are the operators for manipulation of data at values of bit level. Bit wise operators may not be applied to **float** or **double**.

operator	meaning
&	bitwise AND
!	bitwise OR
^	bitwise exclusive OR
~	one's complement
<<	left shift
>>	right shift
>>>	right shift with zero fill

Special operators:

Java supports the following special operators.

instance operator - This operator allows us to determine whether the object belongs to a particular class or not.

Ex: person *instanceof* student is *true* if the **object person** belongs to the **class student**; otherwise it is *false*.

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

```
sample.age        // reference to the variable age
sample.show() // reference to the method show()
```

Arithmatic Expressions:

An arithmetic expression is a combination of variables, constants ,and operators. Java can handle any complex mathematical expressions. some of the examples of java expressions are

Algebric expression	Java expression
ab-c	a*b-c
(m+n)(x+y)	(m+n)*(x+y)
$\frac{ab}{c}$	a*b/c

Evaluation of expressions:

Expressions are evaluated using an assignment statement of the form variable=expression;

Variable is any java valid variable name. When the statement is encountered the expression is evaluated first and the result then replaces the previous value of the variable on the left hand side.

Precedence of arithmetic operators:

There are two distinct priority levels of arithmetic operators in java

High priority * / %
Low priority + -

Operator precedence:

When several operators are used in a statement, it is important to know which operator will execute first and which will come next. To determine the following *operator precedence* are followed

- first the contents inside the braces () and [] will be executed
- next, ++ and –
- next, *,/and %
- + and – will come next
- relational operators are executed next
- Boolean and bit wise operators
- Logical operators
- Ternary operator
- Assignment operators are executed at last.

DECISION MAKING & BRANCHING

In decision control statements, depending on the result of evolution of an expression, a statement or block of statements are executed .The condition involves Boolean values either true or false. Java provides the following decision control statements.

- if statement
- if-else statement
- nested if-else statement
- switch

If statement:

In the if statement first control checks the condition that condition is true statements will be executed

yntax: if (condition)

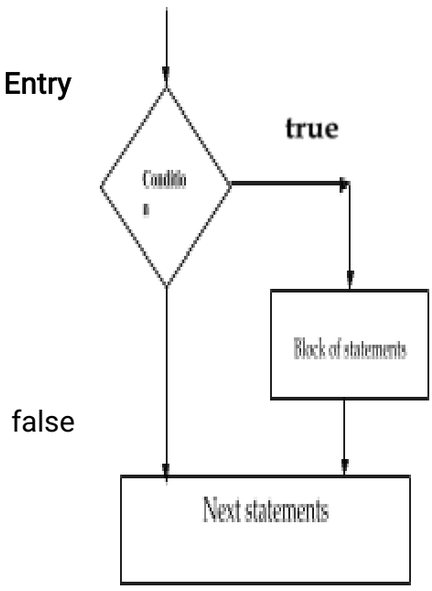
S

{

Block of statements;

}

Flow chart:



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

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

Example:

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

If-else statement:

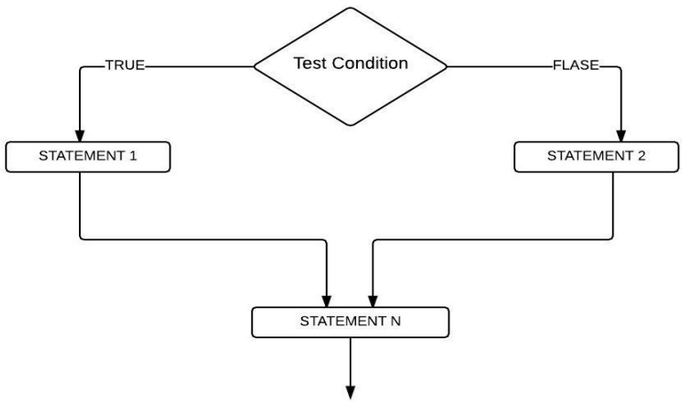
The if –else statement first check the condition and execute the true block statements, if the condition is true otherwise execute the false block statements.

Syntax:

```
if(condition)
{
    true block statements
}
els
e
{   false block statements

}
```

Flowchart:



.

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

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

Example:

```
class if_else_demo
{
        public static void main(String args[])
        {
                int a=2, b=6;
                if(a>b)
                System.out.println("a is big");
                else
                System.out.println("b is big");

                }
}
```

Nested if-else statement:

If The if-else statement is placed within another if-else statement is called nested if-else statement.

Syntax:

```
if(condition1)
{
        if(condition2)
        {
                statement1;
        }
        els
        e       {

                        statement2;

                }
}
        else
{

                if(condition3)
                {
                statement3;
                }
                else
                {
                statement4;
                }
```

```
                    }
```

Example:

```
                     class nested_if_else_demo
                    {
                        Public static void main (String
                            args[]){ int a=10, b=20, c=30;
                            if (a>b)

                                {
                                if(a>c)
                                {
                                    System.out.println (“a is big”);
                                }
                                else
                                {
                                    System.out.println(“ c is big”);
                                }
                            else
                                {
                                if(b>c)
                                {

                                System.out.println(“ b is big”);
                                }
                                else
                                {
                                    System.out.println(“ c is big””);
                                }
        }
        }
```

else if ladder:

This statement is used when multiple conditions are involved.it has the following syntax.

Syntax:

```
if (condition1)
     Statement1;
 else if (condition2)
     Statement2;
 else if (condition3)
     Statement3;
 else if (condition 4)
     Statement4;
  ....
 else if (condition n)
    Statement n;
 else
   default-statement;
```

Switch statement :

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

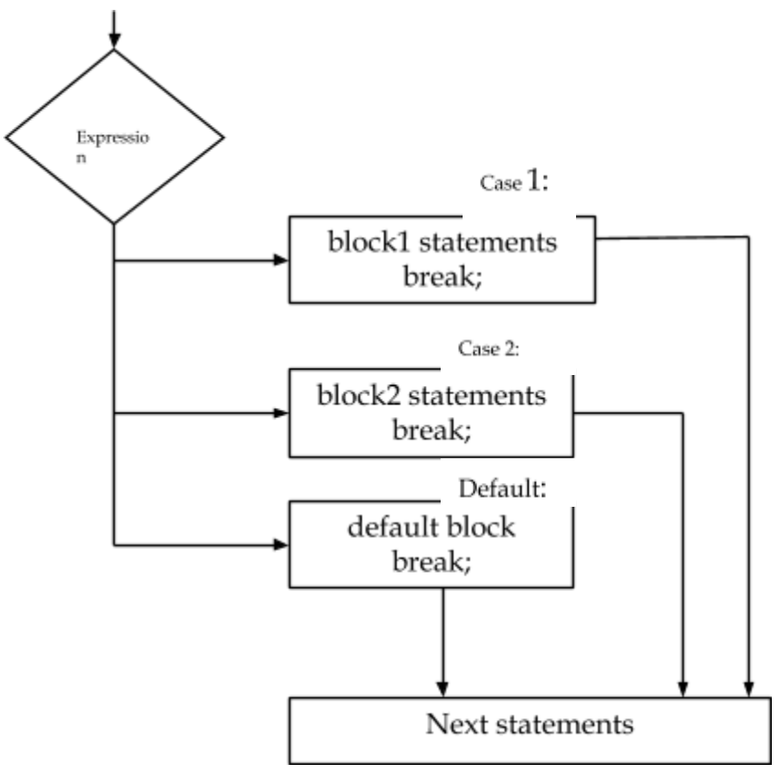
If any value is not coinciding with the case values then it executes default block statements. Here the default block is optional.

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

Syntax: switch (expression)

```
{
        case value1: block1 statements
                        break;
        case value 2: block2 statements
                        break;
                .
                .
                .
        default         :       default block
        statements
                                                        break;
}
```

Flow chart:



Example:

```
class switch_demo
{
        public static void main (String args[])
        {
                int i;

        i=Integer.parseInt(args[0]);
                switch(i)
                {
                        case 1: System.out.println("one");
                                break;
                        case 2: System.out.println("two");
                                break;
                        default :System.out.println("invalid");
                                break;
                }
        }
}
```

`

DECISION MAKING & LOOPING

A loop statement allows running a statement or block of statements repeatedly for a certain number of times. The repetition is continues while the condition id true. When the

condition becomes false, then loop ends and control passed to the next statements following the loop. Repetitive execution of statements is called looping. Java provides three types of loop control statements.

- while statement
- do-while statement
- for statement

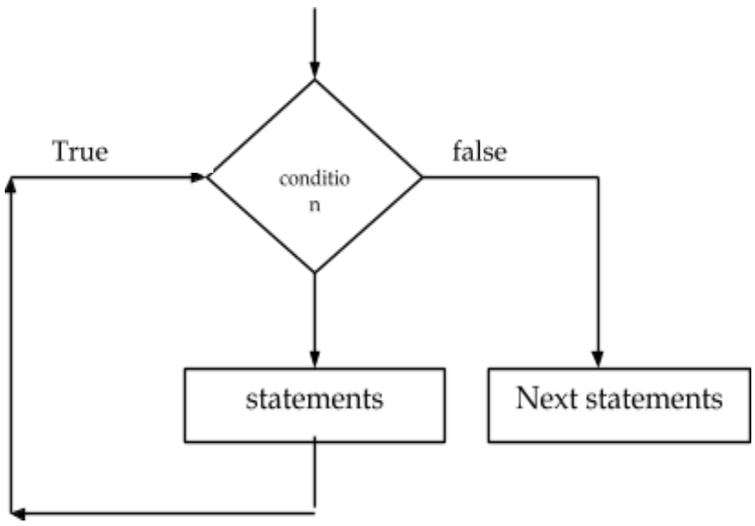
While statement:

While loop executes the statements repeatedly, as long as the given condition is true. When the condition becomes false the control immediately passes to the next statements available after the loop statements.

Syntax:
```
while (condition)
{
        Block of statements
}
```

Flow Chart:

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



Example: to print numbers

```
class while_demo
{
        public static void main (String args[])
        {
                int i=0,n=10;
                while (n>i)
                {
                        System.out.println(i);
                        i++;
                }
        }
}
```

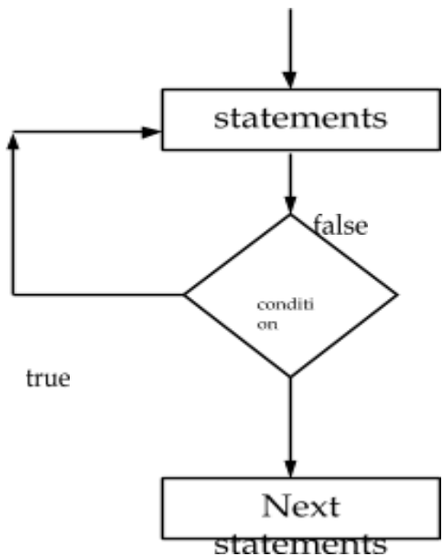
do-while statement:

In do-while loop statement , first the control executes the statements and then it checks for the condition .That means the statements are executed one time without checking the condition. The statements are executed repeatedly as long as given condition becomes true. If the condition becomes false then the control comes out from the loop and executes the statements available after the loop statements.

Syntax:
```
do
{
```

```
        Statements
} while (condition);
```

Flow Chart:



Example:

```
class dowhile_demo
{
    public static void main (String args[])
    {
        int i=0,n=10;
        do
        {
            System.out.println(i);
            i++;
        }while(i<n);
    }
}
```

for statement:

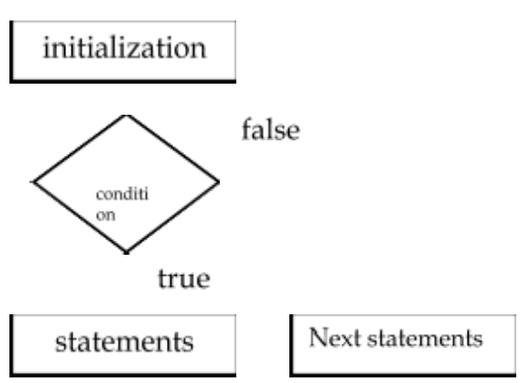
The for statement also repeat a statement or compound statements for a specific number of times.

Syntax:

```
for (initialization; condition; increment/decrement)
    {
        //body of the loop
    }
```

Flow chart:

.



Initialization starts with the assigning a value to the variable and it executes only once at the start of the loop, then control checks for the test condition.

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

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

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

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

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

Example:

```
class prime
{
    public static void main (String args[])
    {
        int n, i,j, count=0;
        n=Integer.parseInt (args[0])
        for (int i=0;i<n; i++)
        {
            if (n%i= =0)
            count++;
        }
        if(count= =2)
        System.out.println("prime number");
        else
        System.out.println("not a prime number");
    }

}
```

Jump control statements:

Java provides the following jump statements

1. break statement
2. continue statement

1.break statement:

Statements executed in a while,do-while,for and switch statements causes immediate exit from that structure. It can be used in two ways as unlabeled break and labeled statement.

a) Unlabeled break statement:

Syntax: break;

Example:

```
for (int k=1;k<=10;k++)
{
if (k==5)
break;
System.out.print(k);
}
```

o/p: 1 2 3 4

b) Labeled break statement:

To break out of nested structure s, we can use the labeled statements. This statement, when executed in a while, do-while and for statements causes immediate exit from that structure and any number of enclosing repetition structures. Program execution resumes with the first statement after the enclosing labeled compound statement.

Example:

```
class ex
{
        public static void main (String args[])
        {
        stop:  for (int i=1;i<=10;i++)
                {
                for (int j=1;j<=5;j++)
                {
                if(i==3)
                break stop;
                System.out.print("*");
                }
                System.out.println("\n");
                }
}
}
}
```

output:
```
* * * * *
* * * * *
```

2) continue statement:

Continue statement executed in a while, do-while or for statements to skip the remaining statements in the body of that structure and proceeds with the next iteration of the loop.It can be used in two ways as unlabeled continue and labeled continue statement.

a) Unlabeled continue statement:

Syntax: continue;

Example:

```
for (int k=1;k<=10;k++)
{
        if (k= =5)
        continue;
        System.out.println(k);
}
```

output : 1 2 3 4 6 7 8 9 10

b) Labeled continue statement:

The labeled continue statement when executed in a repetition structure skips the remaining statements in that structure body and any number of enclosing repetition on structure and proceeds with the next iteration of an enclosed labeled repetition .

Example:

```
class ex
{
        public static void main(String args[])
        {
                stop:for (int row=1;row<=4;row++)
                {
                        for(int col=1;col<=3;col++)
                        {
                                if(col= =2)
                                continue stop;
                                System.out.print(col);
                        }
                        System.out.println("\n");
                }
                                                                        }
}
```

output: 1 1 1 1

□□□□□