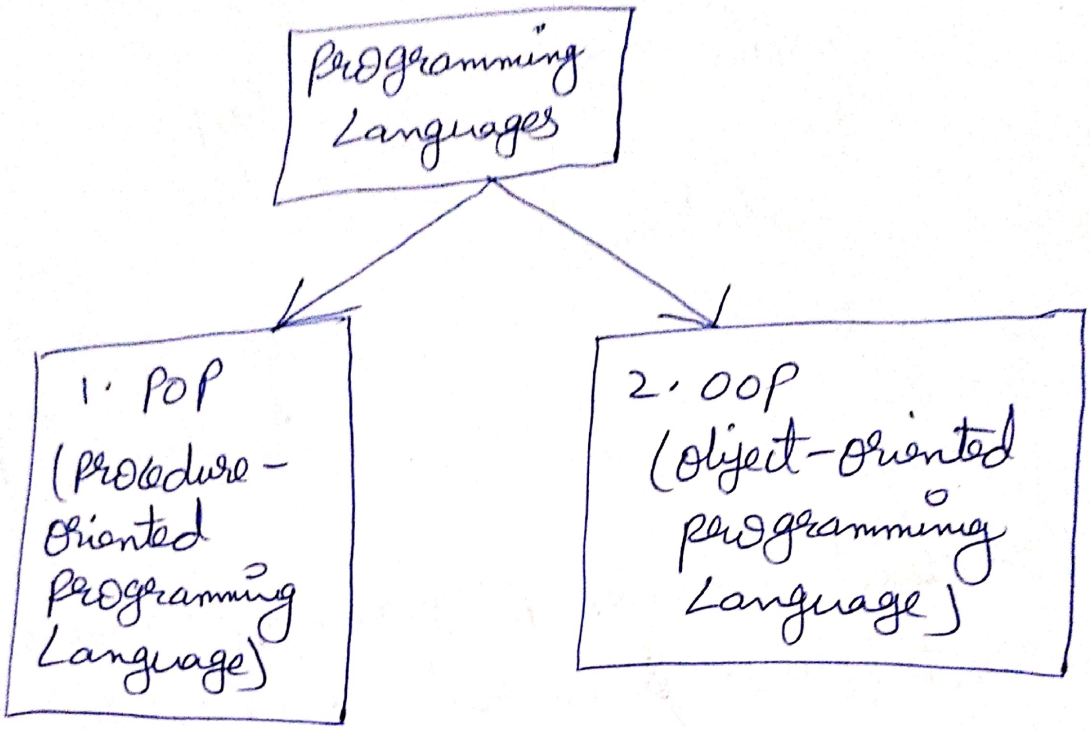
OOP's through JAVA

→ there are 2 types of programming Languages available.

1. POP :- procedure-Oriented programing Language.
2. OOP :- object-Oriented programing Language.

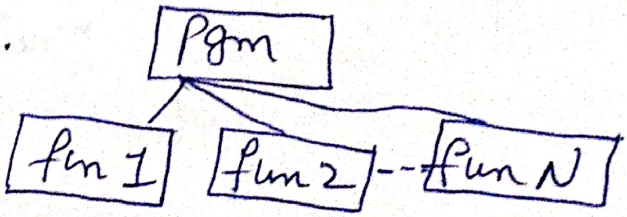


1. **POP:-** it depends on the procedures called function

function:- it is a group of statements that together perform a task.

→ Every program has at least one function, which is main().

→ in POP, total program is divided into number of functions.



⇒ POP follows top-down approach.

⇒ in C language the execution of program starts from main method. from main method all the statements of program are executed.

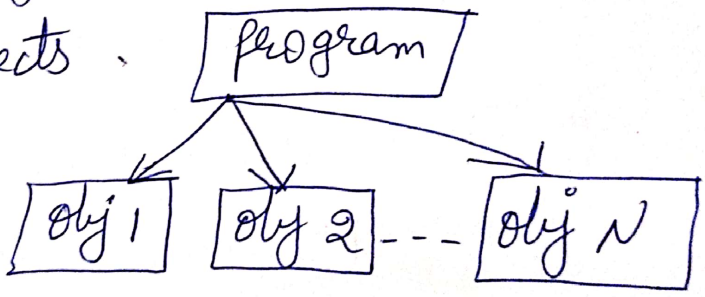
```
main()
{
  statements;
}
```

TOP

↓ down approach.

2. OOP:- Oject-oriented programming Language depends on objects (object is an real-world entity).

⇒ In this the program is divided into small parts called objects.



⇒ it follows Bottom-up approach. (first it consists of class declaration & method declaration, at last we had main method but execution starts from main method.

```
class
{
  methods
}
main()
{
}
```

↑ TOP/up exection.

Bottom-to

Diff. b/w POP and OOP:-

POP	OOP
1. Pgm is divided into small parts called functions.	1. Pgm is divided into small parts called objects.
2. it follows Top-down approach.	2. it follows Bottom-Top or Bottom-up approach.
3. no Garbage collection	3. Provides Garbage collection
4. NO Access specifiers	4. it consists Accesses specifiers (Public, Private, protected).
5. NO Security	5. it provides security
6. Eg:- C, FORTRAN, PASCAL.	6. Eg:- C++, Python, Java.
7. Ex:- clock watch.	7. Eg:- Vechile, automatic door, dish washer, etc.

History of Java :- **Key Notes:** By the end of the 1980's and the early 1990's, object-oriented programming using C++ took hold.

The creation of JAVA:- Java was conceived by James Grosling, Patrick Naughton, Chris Warth,

Ed Frank, and Mike Sheridan at SUN Microsystems, inc. in 1991. it took 18 months to develop the first working version. This Language was intially called "oak" but was renamed "Java" in 1995.

→ Java was not the Internet! Instead, the primary motivation was the need for a platform-Independent (that is, architectureneutral) language that could be used to create software to be embedded in various consumer electronic devices, such as microwave ovens and remote controllers. many different types of CPUs are used as controllers. The trouble with c and c++ is that they are designed to be compiled for a specific target. The problem is that compilers are expensive and time-consuming to create.

→ Gosling and others began work on a portable, platform-independent language that could be used to produce code that would run on a variety of CPU's under different enviroments.

→ The output of a Java compiler is not executable code, it is a bytecode. (is a highly optimised set of instructions executed by Java runtime system

→ Java is an oliject - oriented Programming Language mainly designed to develop internet applications (J2SE), Enterprise applications (business) J2EE, mobile applications (J2ME).

→ Java syntax is similar to c, c++. Java omits many confusing features from c Pointers, from c++ overloading and multiple inheritance.

JAMES GOSLING :- Father of JAVA :-

Java was developed by Gosling at SUN Micro systems. Inc (it is a software company).

→ James Gosling is a famous software developer.

→ Patrik Naughton, Chris Warth, Ed Frank and Mike Sheridan are small Team members of SUN Engineers called "Green Team". * firstly, it was called "Green Talk" and the file Extension was ".gt". Later it was called as OAK and was developed as a part of Green project.

OAK :- OAK Technology name came from OAK tree. That represents the "Symbol of Strength" and choosen as a National Tree of Many countries like the U.S.A, France, Germany, Romania, etc.

⇒ OAK tree that stood outside of the James Grosling's office

OAK Tree "Symbol of Strength". National Tree for many countries like U.S.A, France, Germany, Romania etc.

⇒ In 1995, OAK was re-named as JAVA because it was already a "Trade Mark" by OAK Technologies.

JAVA Abbreviation :- Java name was chosen by Grosling while having cup of coffee near by his office with his team members. Java coffee is a famous coffee. JAVA is an Island in Indonsia, Java coffee was produced by Espresso Bean (coffee seed) where first coffee was produced. Java refers to Hot, Aromatic drink coffee; Icon for Java Language is coffeecup.

JAVA Phrase / SLOGAN :-

"WORA" write Once Run Anywhere.

Platform:- it is Hardware or Software Environment to run programs.

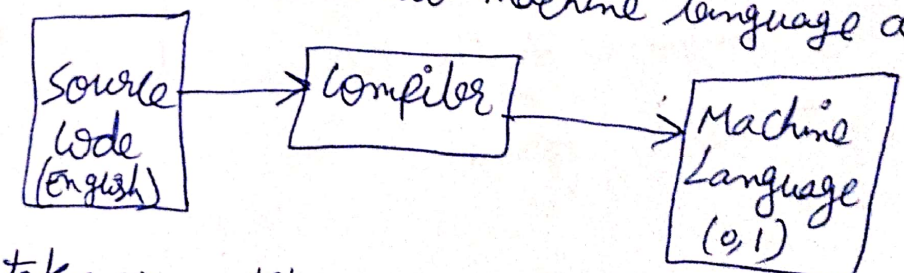
Platform Dependent:- An application that is compiled in one operating system and Run in that system only is called platform dependent. Eg: C, C++.

Platform Independent:- the compiled code application is able to run in different operating systems (like windows, Unix, Linux, Solaries) is called platform Independent. Eg:- Java, Python.

Source code:- Developer written program; it is written according to the programming language Syntax.

Compiled code:- Compiler generated program that is converted from source code.

1. **Compiler:-** it is a translation program that converts source code into machine language at once.



Compiler:- it takes Java file name as its input & generates byte codes for all classes defined in that Java file & stores each class →

→ Each class bytecodes in a separate ".class" file with name same as class name.

Interpreter :- it is also a translation program that converts source code into machine language but line-by-line.

Executable code :- os understandable readily executable program (.exe files).

Compilation :- it is a process of translating source code into compiled code.

Execution :- it is a process of running compiled code to get output.

Applets :- to develop web supportive applications.

Servlet and JSP :- to develop web applications.

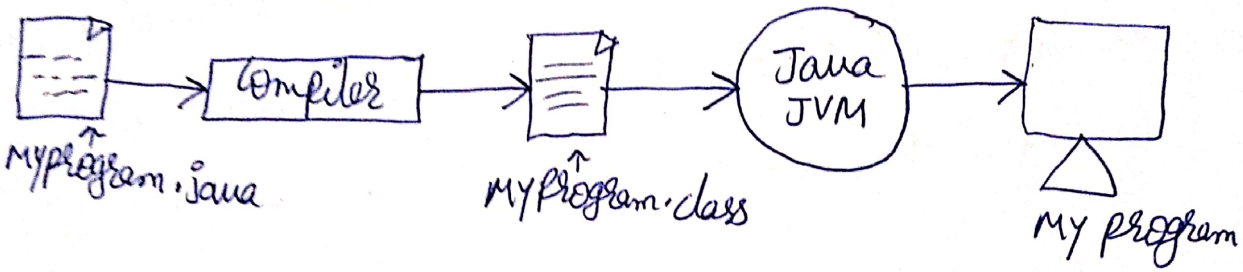
JVM :- Java Virtual Machine :- is a software that executes Java bytecodes by converting bytecodes into machine language of the current operating system's ~~language of the current client~~ os. understandable format.

Java file's extensions :-

"**.java**" :- extension file it called source code which is developed by developer.

"**.class**" :- extension file it is called code contains bytecode which is generated by compiler from source code.

Java program compilation and execution :-



Path :- Used by OS to identify binary files.

classpath :- Used by compiler and JVM to identify Java library files.

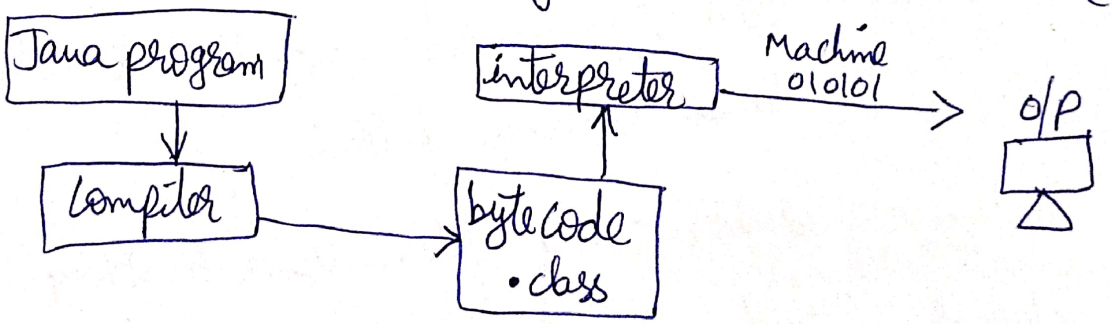
Package :- it is a Java folder used to group related classes, interfaces and enums also used to separate new classes from existed classes if both have same name.

Compilation time errors:- the errors thrown by compiler at the time of execution compilation are called compiled time errors. These errors are raised due to syntax mistakes, like spelling mistakes, wrong usage of keywords, missing ";" at end of statements etc.

Runtime errors:- the errors thrown by JVM at the time of execution are called runtime errors or exceptions. These errors are raised due to logical mistakes like executing class without main method, dividing integer number with ZERO, accessing array values with wrong index, etc.

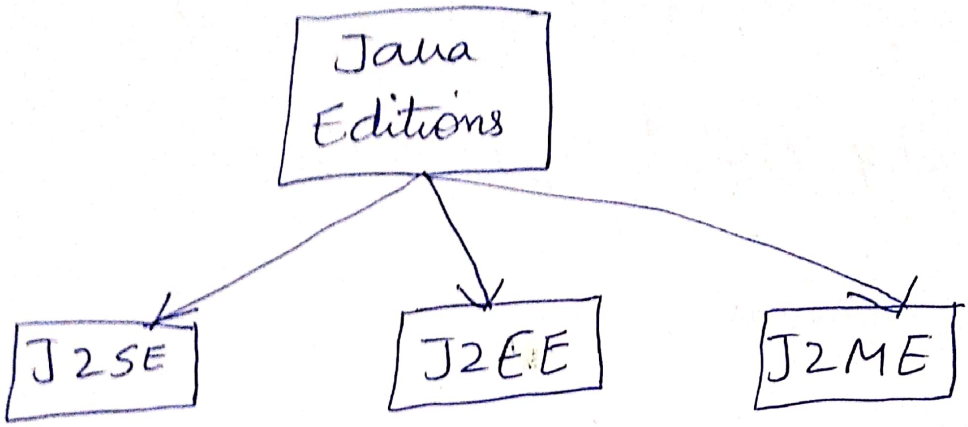
Compiler:- check the Errors.

interpreter:- used for converting bytecode to Machine code.



Editions of JAVA :- Java has 3 types of Editions.

1. J2SE :- Java 2 standard Edition.
2. J2EE :- Java 2 Enterprise Edition (Business).
3. J2ME :- Java 2 Mobile/Micro Edition.



1. J2SE (standard Edition) :- it covers the fundamental concepts of JAVA (core Java). Core Java is nothing but OOP's concepts like Encapsulation, Inheritance, Polymorphism, Overloading, interfaces, strings, Applets.

→ J2SE is mainly used to develop stand-Alone applications.

2. J2EE (Enterprise / Business Edition) :- it covers advanced concepts of JAVA like servlets, JSP's.
→ it is mainly used in order to develop client/server applications.

3. J2ME (Mobile / MICRO Edition) :- it is used to develop Mobile applications.

JAVA Versions :-

1. Java Apple & Beta (1995)
2. JDK 1.0 (1996)
3. JDK 1.1
4. J2SE 1.2
5. J2SE 1.3
6. J2SE 1.4
7. J2SE 5.0
8. Java SE 6

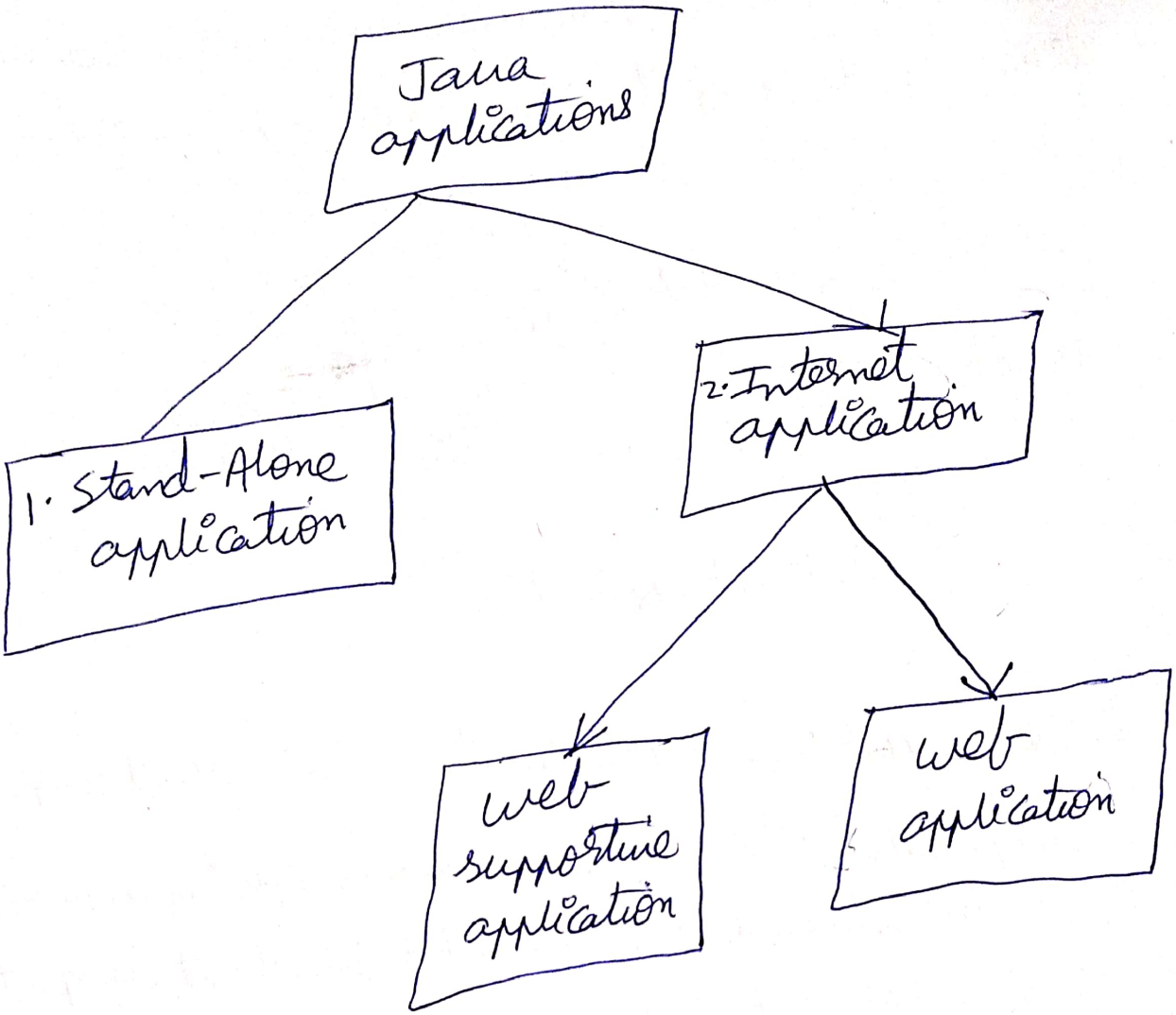
} these versions are owned by Sun Microsystems.

9. Java SE 7
10. Java SE 8
:
13 (2019)

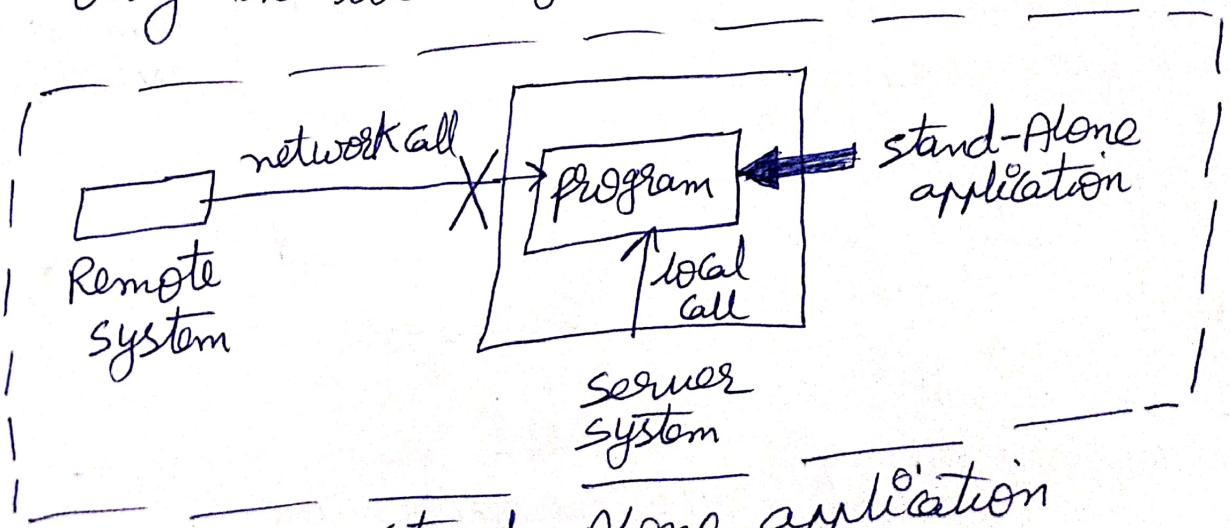
} owned by Oracle.

→ In 2010 oracle own all permissions/rights of JAVA.

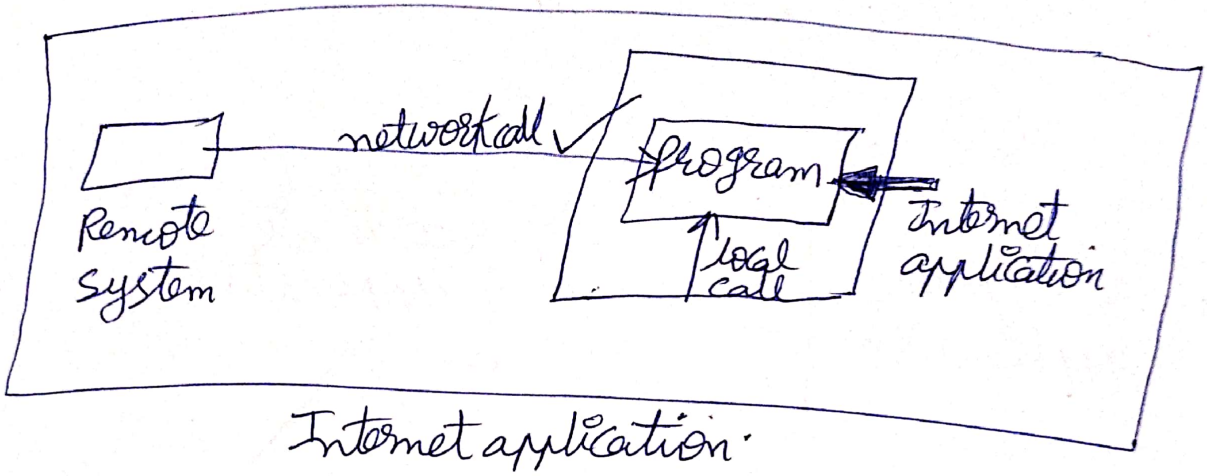
Java Technologies / applications :-



1. stand-Alone application :- these can execute only in local system with local call.

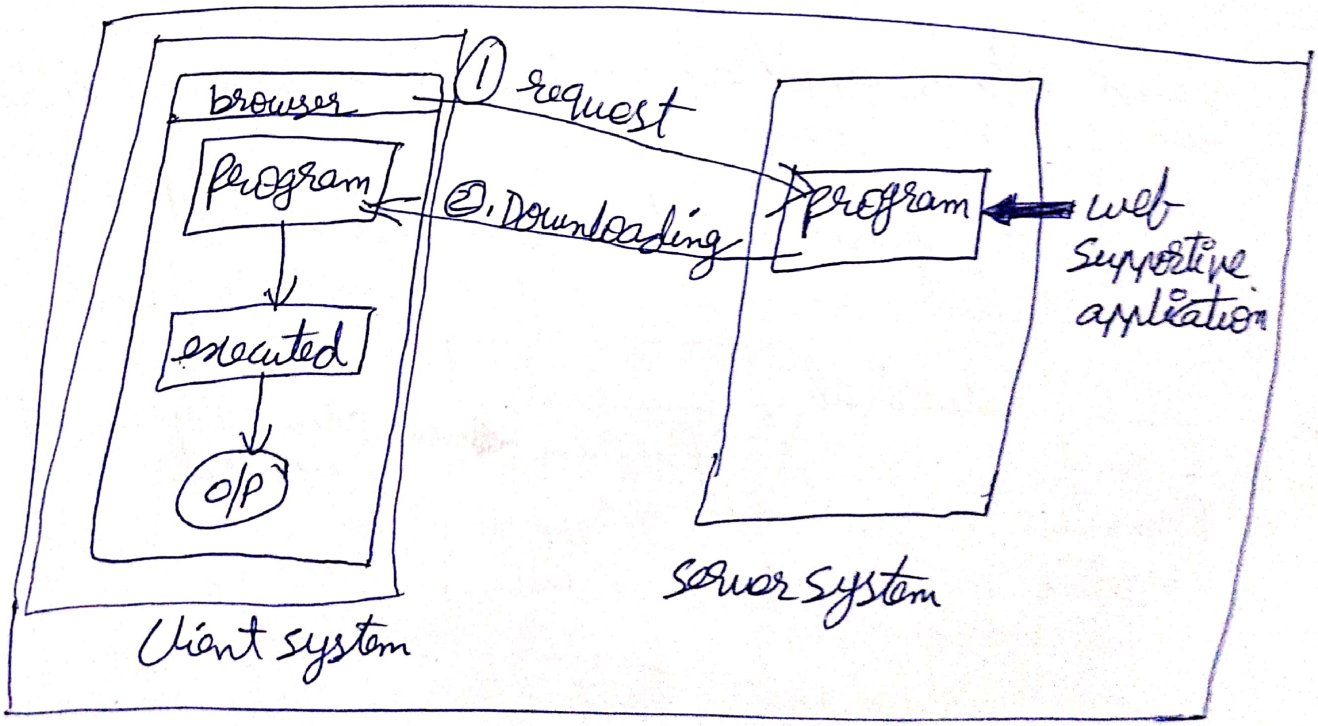


2. Internet applications :- An application that can be executed in local system with local call and also from remote computer via network call(request).



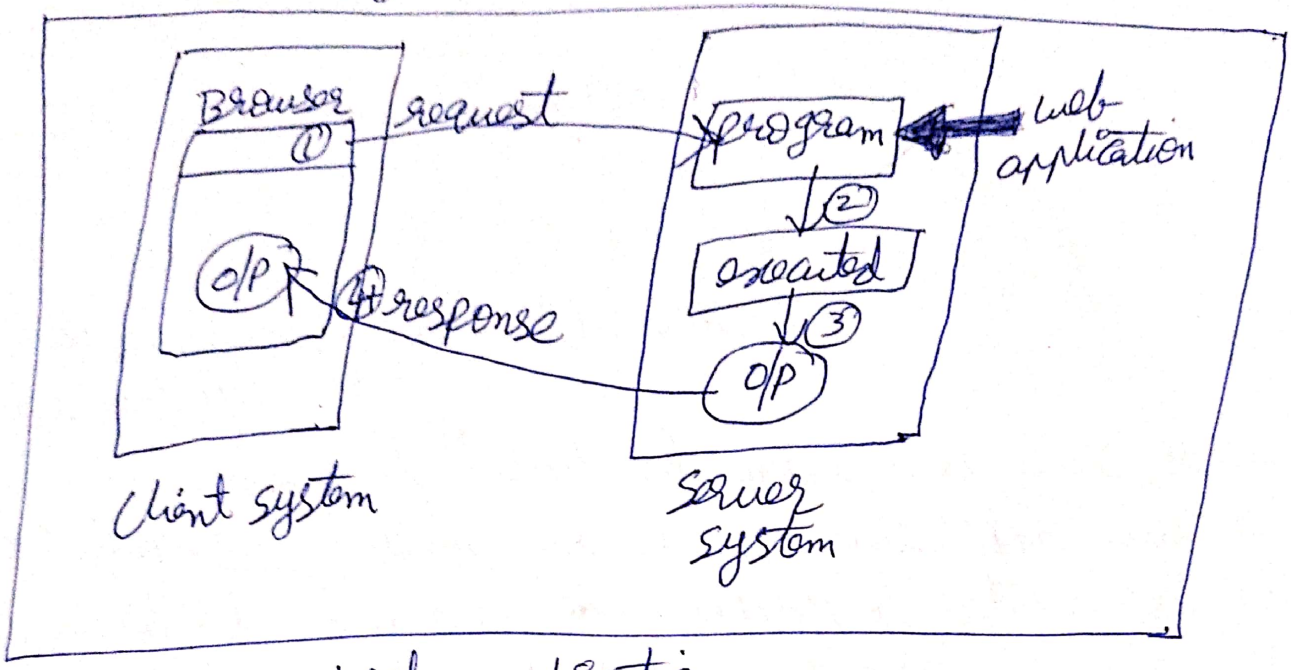
Internet application.

a) web supportive application(executed in client computer):- an application that resides in server system and that is downloaded and executed in client computer via network call.



Web Supportive application

2.) Web application :- it resides in server and that is executed directly in server system via network call and sending response (output) back to client.

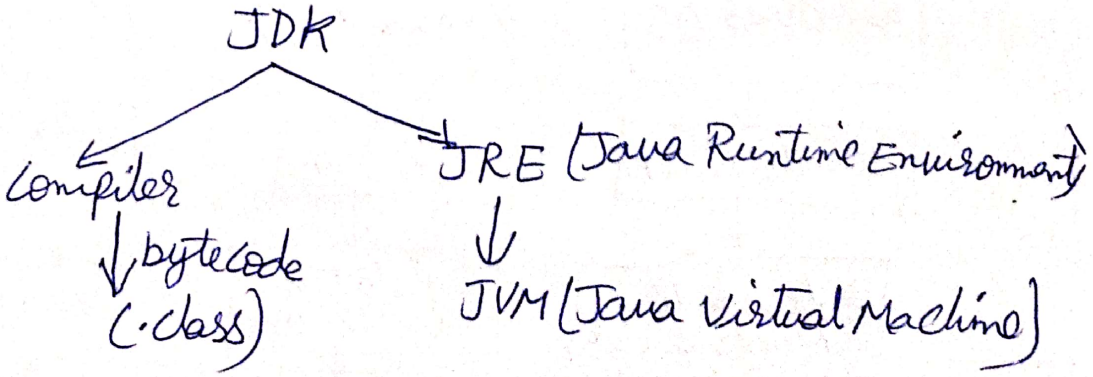


Web application.

Java Technologies :- How Java works :-

Java Development Tool Kit (JDK) :-

⇒ JDK provides an environment to develop compile and execute a Java application. JDK is a combination of compiler and JRE.



JRE :- it provides environment for executing Java applications.

JVM :- it executes Java program line-by-line.

JVM (contains platform dependent).

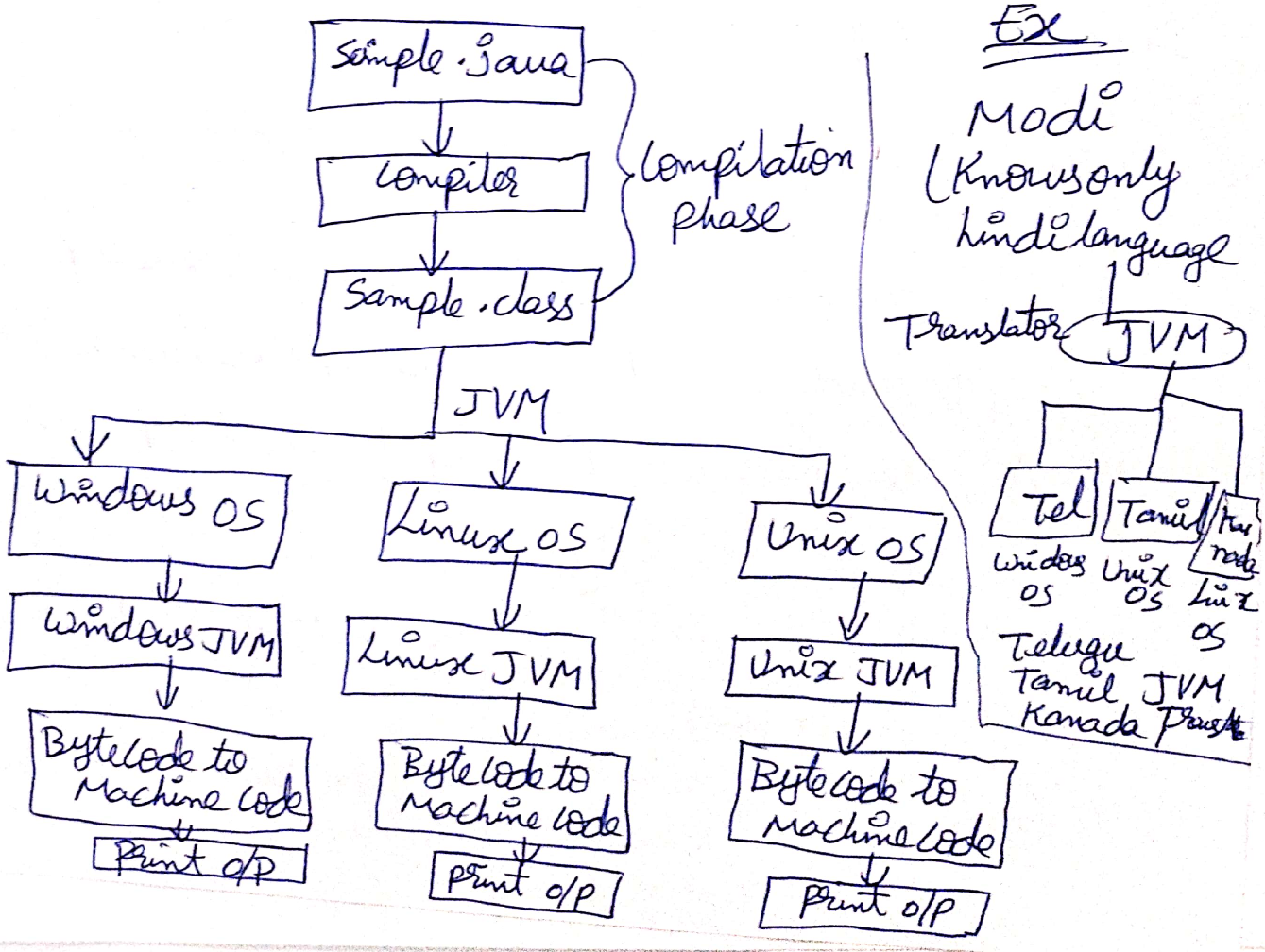
↓

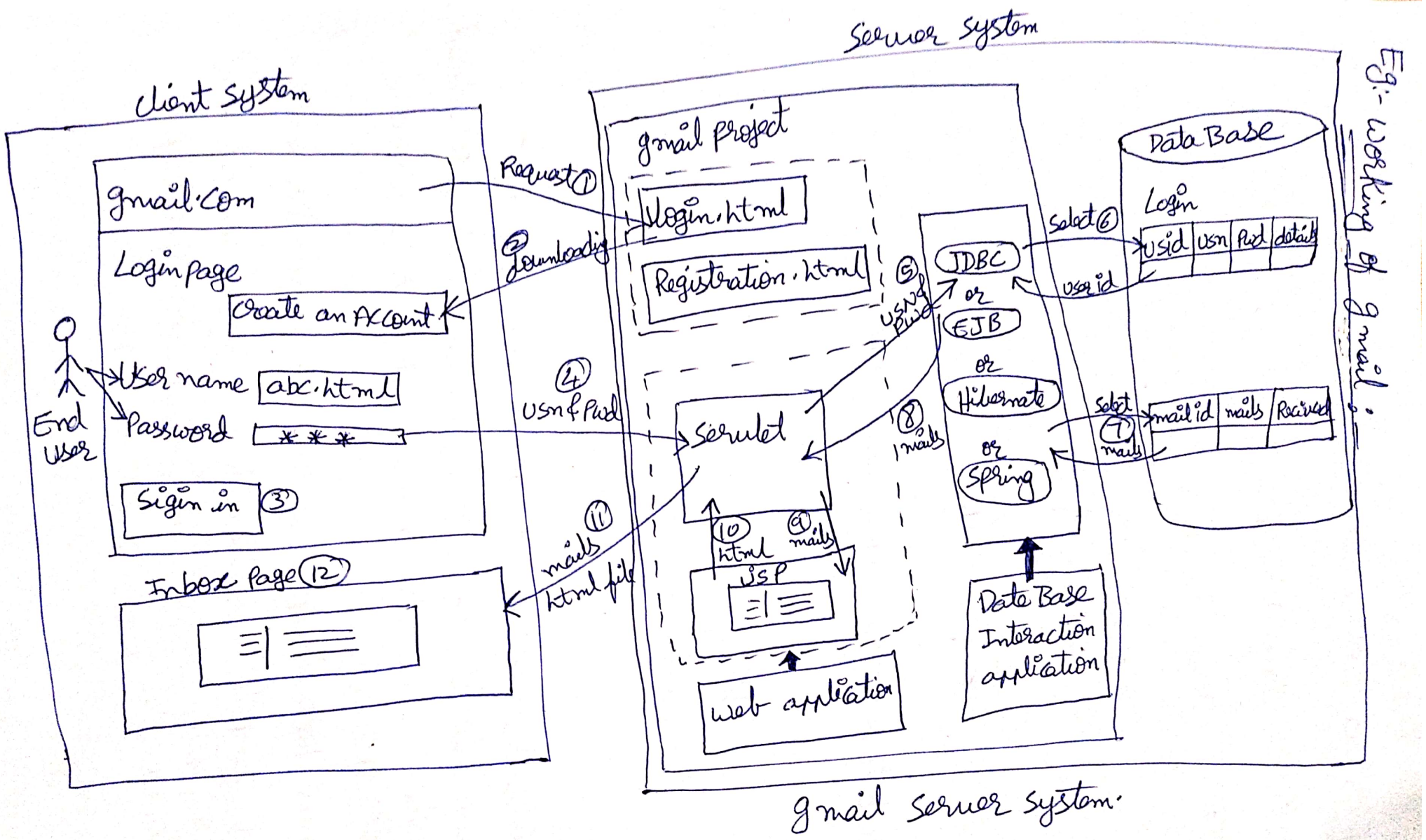
Operating system (every OS has its own JVM translator).

→ So, Java is a Interpreter based language.

→ Java application can be run on any OS. bytecode is converted to machine code by any OS. so, we can say Java is a platform independent language.

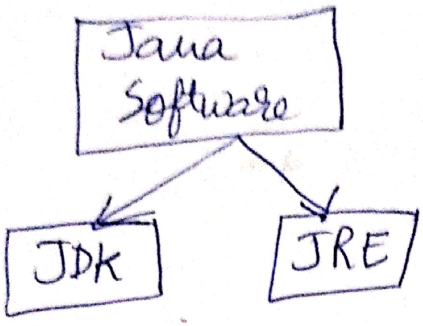
→ Java software (JVM) is platform dependent.



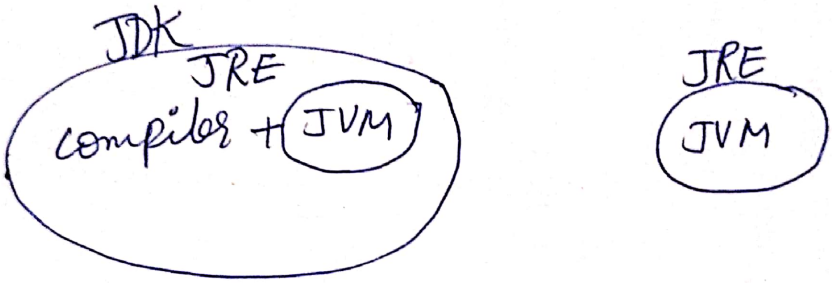

Server system
Client system
gmail project
Data Base
gmail.com
Login page
Create an Account
Request ①
Downloading ②
Login.html
Registration.html
Login
uid | usn | Pwd | details
End User
User name
abc.html
Password
* * *
Sign in ③
usn & Pwd ④
Servlet
JDBC
or
EJB
or
Hibernate
or
Spring
usn & Pwd ⑤
select ⑥
user id
select ⑦
mails
⑧ mails
⑨ mails
⑩ html
JSP
⑪ mails
html file
Inbox Page ⑫
mail id | mails | Received
Data Base Interaction application
web application
gmail server system.
Eg:- working of gmail:-
9

Types of Java softwares :-

1. JDK : Java Development Kit
2. JRE : Java Runtime Environment



difference between JDK, JRE and JVM :-

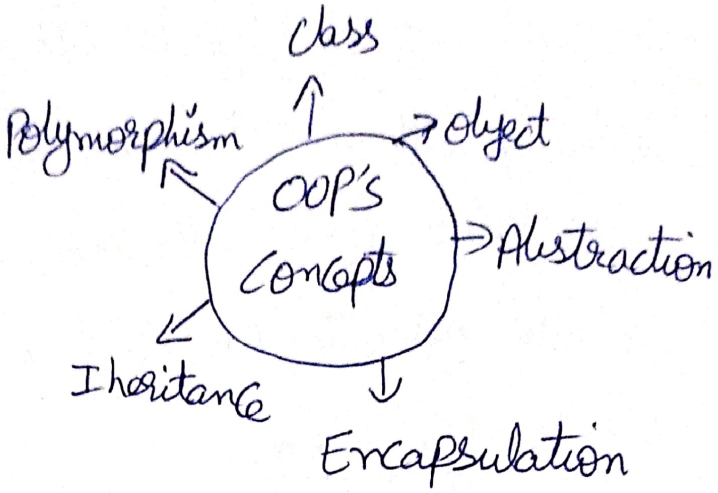


→ JVM is subset of JRE, & JRE is subset of JDK. When we install JDK, JRE is also installed automatically. JRE s/w is available as a seperate pack, so we can install JRE alone.

→ JDK has both compiler & JVM. So using JDK we can develop, compile and execute new Java applications & also modify, compile and execute already developed applications.

OOP'S Concepts :-

1. Class
2. Object
3. Abstraction
4. Encapsulation
5. Inheritance
6. Polymorphism



1. **Class :-** A class is a blue print / instance from which an object is created.

(or)

Class is a collection of methods & variables

(or)

Class is a collection of objects.

(or)

Each class is a collection of data & functions that manipulate the data.

→ The data components of a class are called data fields and function components are called member functions / member variables.

⇒ The class models the state and behavior of a real-world object. class is a template or a blue print which does not occupy any memory location. Data declare exact form and nature. Code operates on that Data.

Class defination :-

```
class className
{
  type instance_Variable 1;
  type instance_Variable 2;
  ....
  type instance_Variable N;
type methodname1(Parameter-list)
{
  // body of method
}
  //.....
type methodnameN(Parameter-list)
{
  // body of method
}}
```

→ Class is the starting Point for your program. applets don't require a main() method at all.

Class syntax :-

```
class <class_name>
{
    fields/variables   → Instance Variable.
    methods
}
```

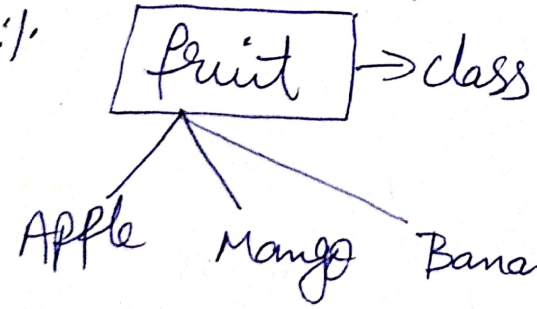
Eg:-

```
class student
{
int rno;
int marks;
void total();
```

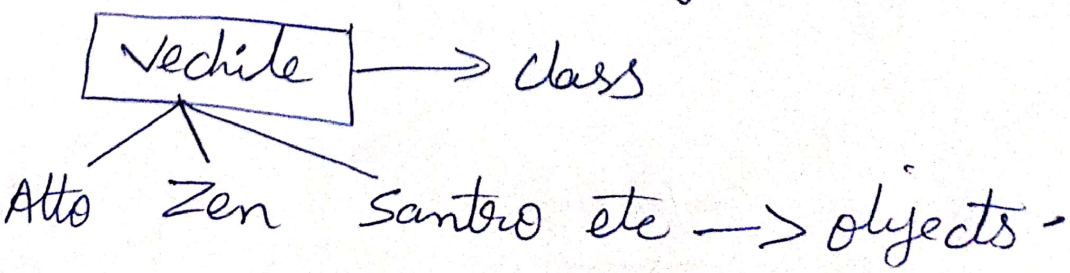
Class is represented as

class name
Data fields
m v/m f

Eg:1.



Eg:2.



Object :- object is an instance of a class; the object represents real world entity.

→ memory for class variables are allocated only after object is created.

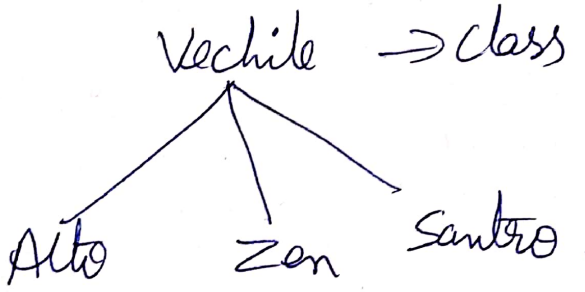
Ex: an object of type Box;

```
Box mybox = new Box();
```

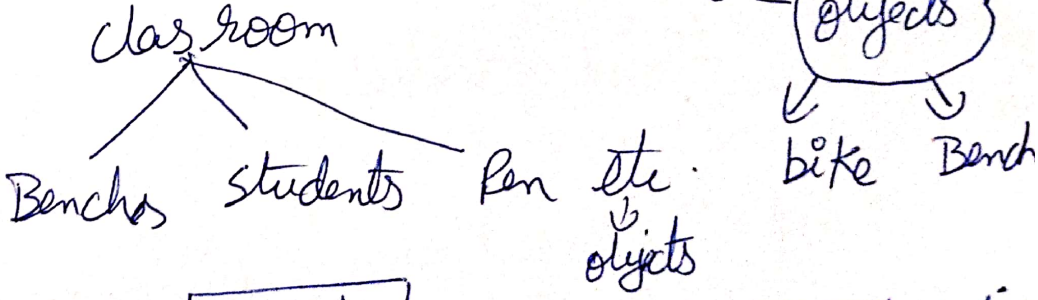
this statement combines the two steps just described.

```
Box mybox; // declare reference to object.
mybox = new Box(); // allocate a Box object.
```

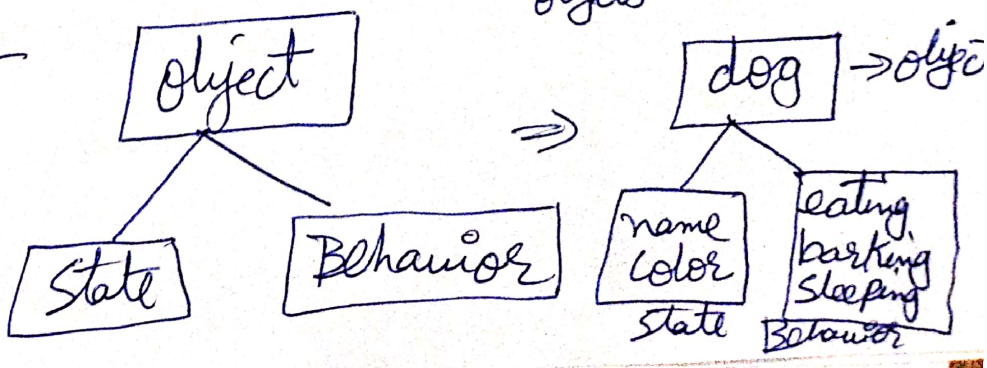
Ex:- 1.



Eg: 2.



Eg 3:-



Object syntax :-

Class_name objectname = new classname();

Eg:- Student x = new Student();

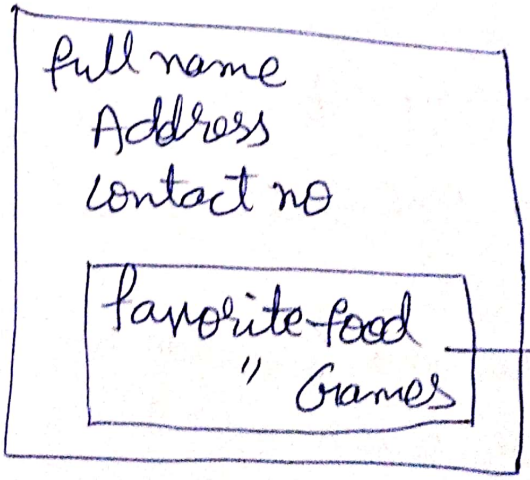
S. roll no;
S. marks;
S. total();

3. Abstraction :- (Bulb) :- hide bulb's internal function and only shows light.

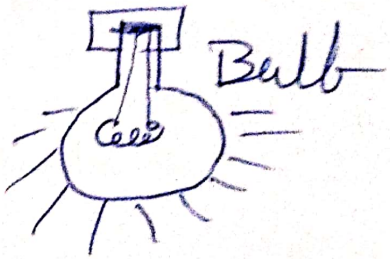
⇒ Abstraction shows only essential information and hides unnecessary information.

⇒ the main purpose is hiding unnecessary details from the user.

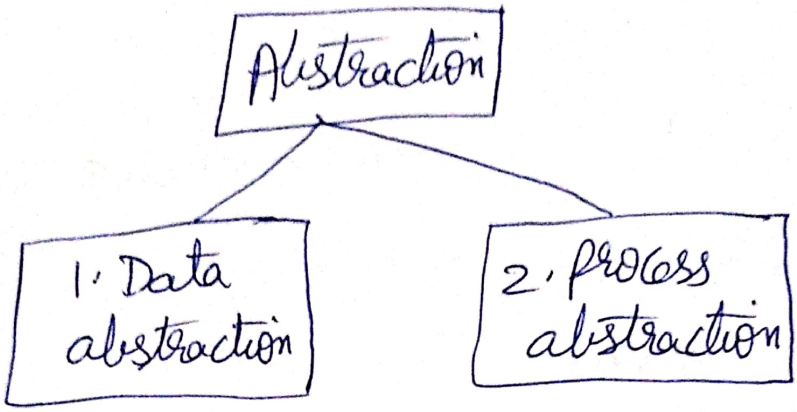
Ex:- Bank application



→ these are not need for bank application



⇒ Fetching necessary information & removing unnecessary information is called abstraction.



1. Data abstraction :- the process of hiding the internal representation of data objects & exposing the necessary information.

2. Process abstraction :- hiding the implementation details of a process & providing simplified interface for executing it.

4. Encapsulation :- (code | data) Capsule (tablet).

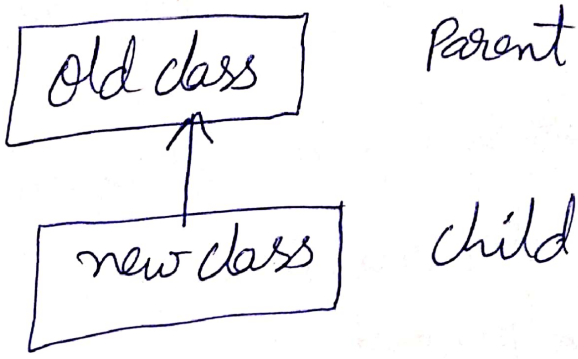
Combining / wrapping / binding variables and methods under one unit is called Encapsulation.

(or)

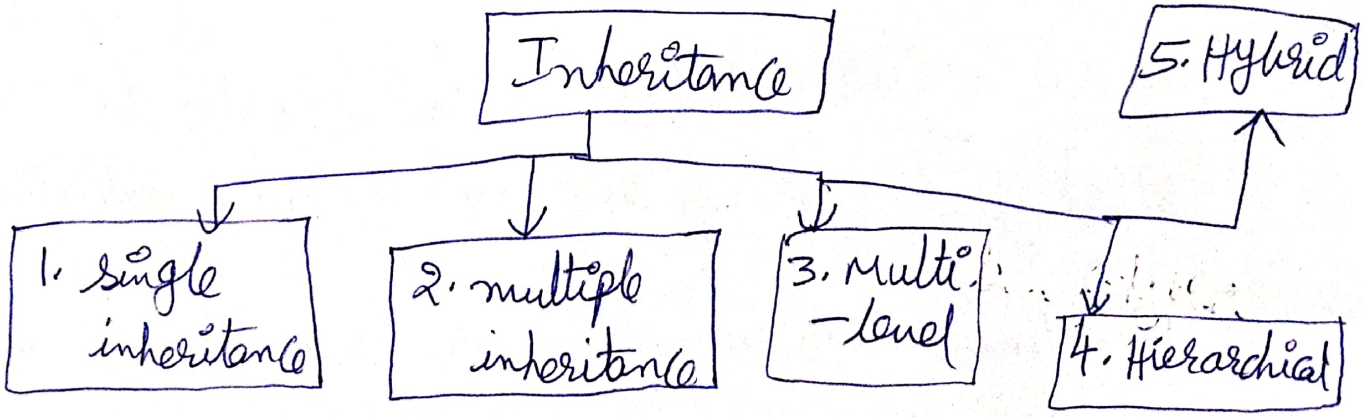
wrapping of data and function within the structure is called Encapsulation. (or)

Binding code & data together into a single unit.

5. Inheritance :- the process of creating a new class from old class (or)
the process of Accessing the properties from old class to new class is called Inheritance.
→ old class is called Base class/Parent class/super class.
→ new class is called Derived class/child class/Sub class



⇒ Re-Using the properties from one class to another class.
Ex:- human beings are inherited from parent or grand parent features.

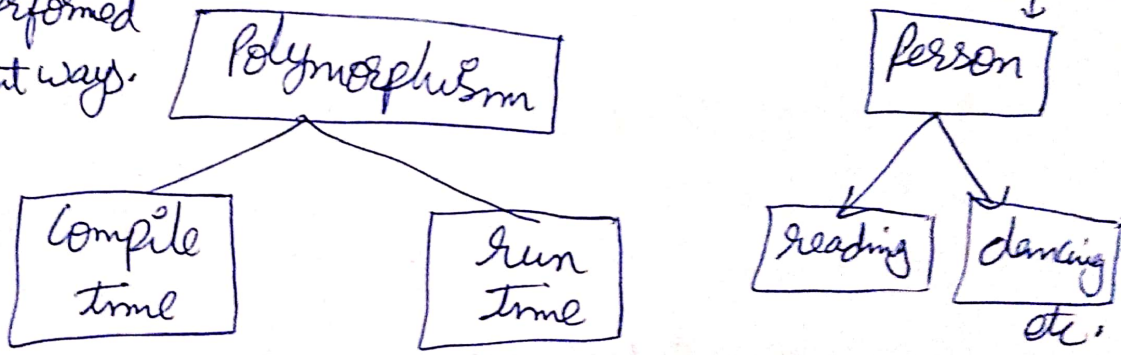


1. Single Inheritance
2. multiple "
3. multi-level "
4. Hierarchical "
5. Hybrid ".

6. Polymorphism :- the word polymorphism taken from Greek. Poly means Many; Morphs means forms.

⇒ Representing one thing in many forms(ways) is called Polymorphism.

⇒ 1 task performed by different ways.

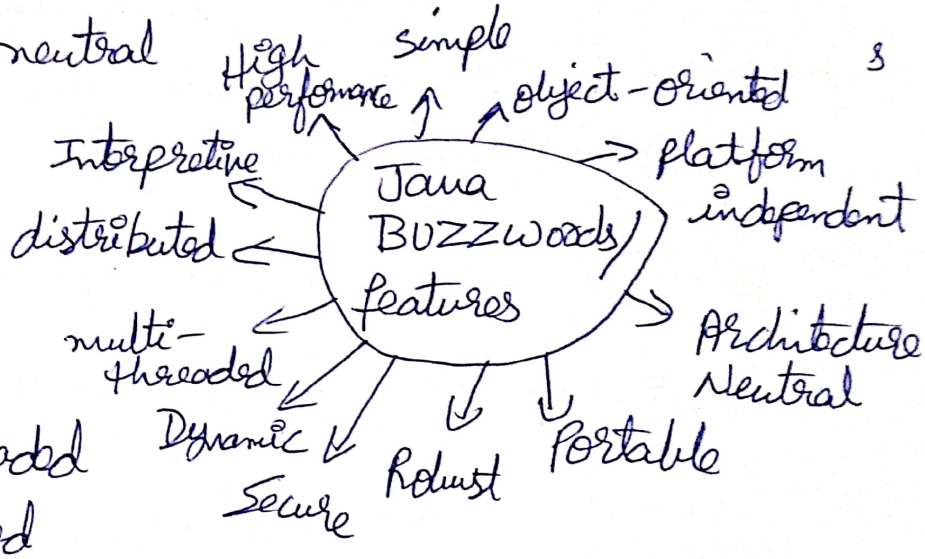


1. Compile-time :- which method should be called during compile time Ex:- method overloading

2. Run-time :- which method should be called during run time. Ex:- method overloading.

→ Java features/BUZZ words :-

1. Simple
2. object-oriented
3. platform independent
4. Architecture neutral
5. Portable
6. Robust
7. Secure
8. Dynamic
9. multi threaded
10. Distributed
11. Interpretive
12. High performance



1. Simple :- Java is a simple language, because many Syntaxes from C, C++ are similar (same).
→ Java omits confusing concepts, Pointers in C & method overloading & multiple inheritance in C++.
→ Execution time of Java is less.

2. object-oriented :- execution depends on objects.
→ it supports various object-oriented features such as data Encapsulation, inheritance, Polymorphism. in Java Everything is object.

3. Platform independent :- "WORA" (Write Once Run Anywhere). Java programs can be executed in any OS (operating system).

4. Architecture neutral :- it is same as platform independent. we can execute Java programs on any configuration, on any OS, on any H/W (hardware).

5. Portable :- we can move program from one hardware to another. easy to carry.

6. Robust :- means strong.

⇒ In programming languages like C, C++ the memory management is done explicitly by the User. that means User allocates (or) de-allocates memory. where as in Java it automatically done using Garbage collection.

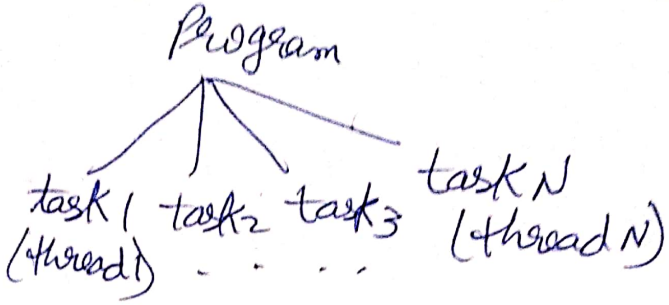
7. Secure :- JVM security manager is there for protection.

⇒ JAVA Authentication & Authorization ~~security~~ service (JAAS) will take care about the security mechanisms.

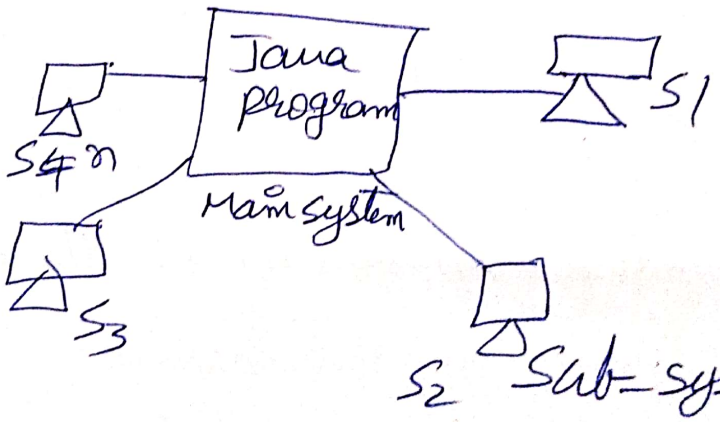
⇒ we had Access specifiers to secure.

8. Dynamic :- class file is loaded into memory during the runtime. memory is allocated during execution time.
→ Java is a dynamic programming language.

9. Multi-thread :- Program is divided into number of tasks. each task is called as thread. Java can execute multiple threads simultaneously.



10. Distributed :- Java is a distributed language. Java program can be executed in many systems simultaneously. By Java we can develop network application



11. Interpretive :- Java is both compiler & interpretive language.

12. High-performance :- Java is a high performance language because it is platform independent, simple etc.

TC3: Data Types, Variables, Scope and life time of Variables:-

Data Types :- These are used to declare Variables to store values.

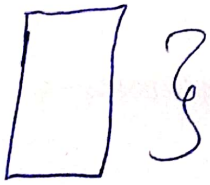
Data types are used to store data temporarily in computer through a program.

⇒ In real world we have different types of data like integer, floating-point, character, boolean and string.

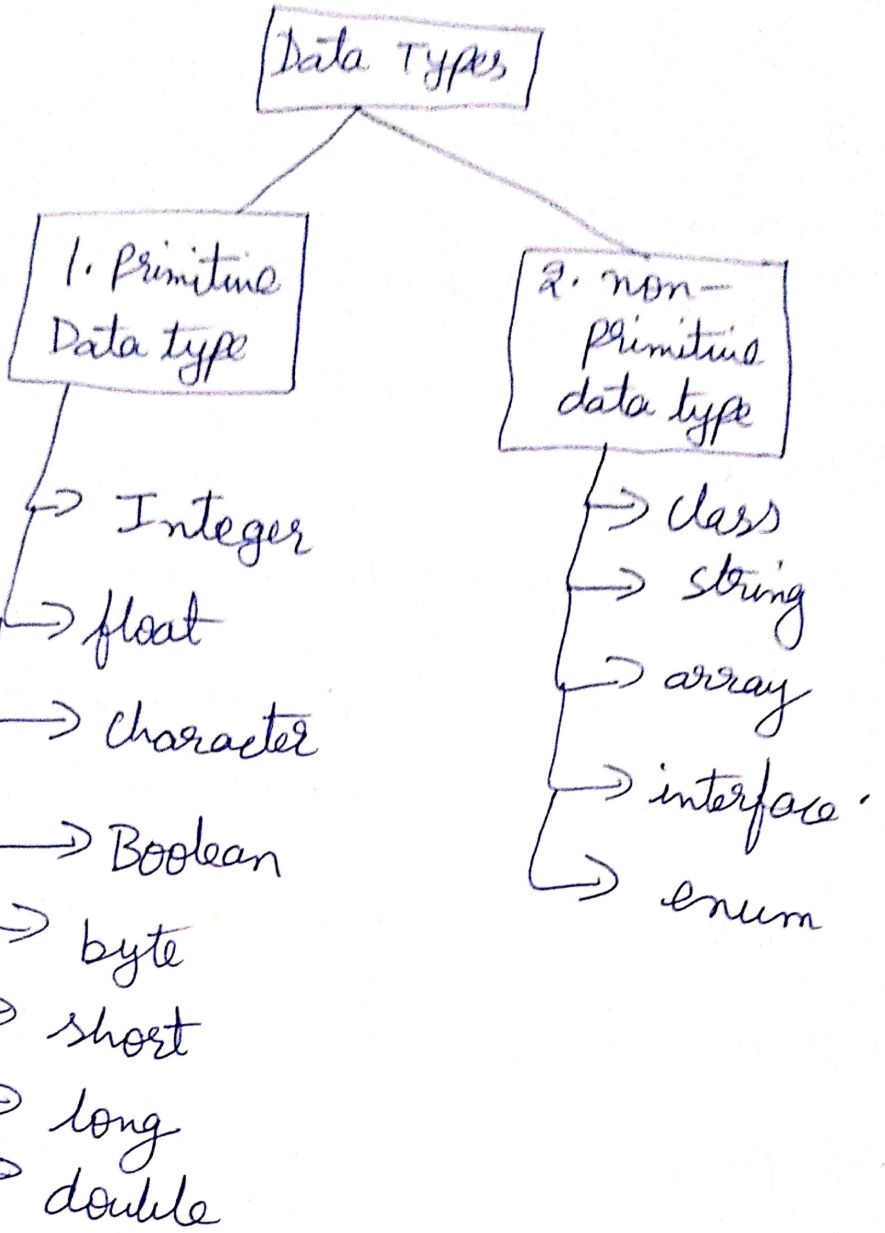
⇒ to store all these types of data in program to perform business required calculation and validations we must use data types concept.

⇒ Data type is something which gives information about,

1. size of memory location and range of data that can be accommodated inside the location.

 ← size of memory location

2. Possible legal ~~location~~ operations those can be performed on that location.



1. Primitive data types: The single valued data formats are known as primitive data types or basic data types or Fundamental data types.
Eg:- int, float, char, Bool etc.

1. Integer data type :- it represents the numeric data without decimal point.

integer Types :-

1. byte - 1 Byte (8 bits)
2. short - 2 Bytes
3. int - 4 Bytes
4. long - 8 Bytes

⇒ byte and short data types are used for stream-data, which means used for multimedia data.

⇒ int data used in normal programming.

⇒ long is used to hold largest integer values like phoneNo, accNo, cardNo,.. etc

2. float :- it represents numeric data with decimal point.

Types :-

1. float ⇒ 4 Bytes
2. double ⇒ 8 Bytes.

Eg :- float e = 12.34F;

3. ∮ double :- used to hold largest scientific calculated values.

byte = 1byte (8 bits) ; short = 2 bytes (16 bits)

$2^8 = 256$

$256/2 = 128$

Range = -128 to +127

$2^{16} = 65536$

$65536/2 = 32768$

Range = -32768 to +32767.

4. char :- it consists single valued characters represents in single quotes.

Eg:- 'K', 'h', ...

Type :- char — 2 Bytes

5. Boolean :- it represents in the form of true or false.

boolean — 1 bit

2. Non-primitive data types :- the grouped value data formats are known as non-primitive data types or Refrential datatypes.

Eg: class, interface, array, enum.

String :- is a class and non-primitive datatype or Referential data type; ~~or~~ String is a collection of words or characters.

Name	width	Range
long	64	$-9,223,372,036,854,775,808$ to $9,223,372,036,854,775,807$ (or) -2^{63} to $2^{63}-1$
int	32	-2^{31} to $2^{31}-1$
short	16	$-32,768$ to $32,767$
byte	8	-128 to 127
double	64	$4.9e-34$ to $1.8e+308$
float	32	$1.4e-045$ to $3.4e+038$.

Variables :- The variable is the basic unit of storage in Java program. It is a combination of an identifier, a type, and an optional initializer all variables have a scope, which defines their visibility, and a life time.

⇒ In Java, all variables must be declared before they can be used. During program execution we can modify the data. Variable is a name that used to store data value. A variable may contain letters, digits & underscore symbol.

Syntax:- type identifier [= value][, identifier [= value]...];
(type → name or interface of class; identifier → name of variable.)

(or)

Syntax:- data type <identifier>
int a; → variable name;

defining a variable:-

<Access specifier> <Modifier> <datatype> <variable name>
Public static void/int x = 10;
= <value>;

Public static int x=10;

Ex:- int a, b, c; // declares three int's a, b, c.
int d=3, e, f=5; // declare three more ints
byte z=22; // initialize z.
double pi = 3.14159; // declares an approximation of pi
char x = 'x'; // the variable x has the value 'x'.

Constants :- Constants are also known as literals.

Ex:- an integer constant is int n = 100;
↳ (literal)

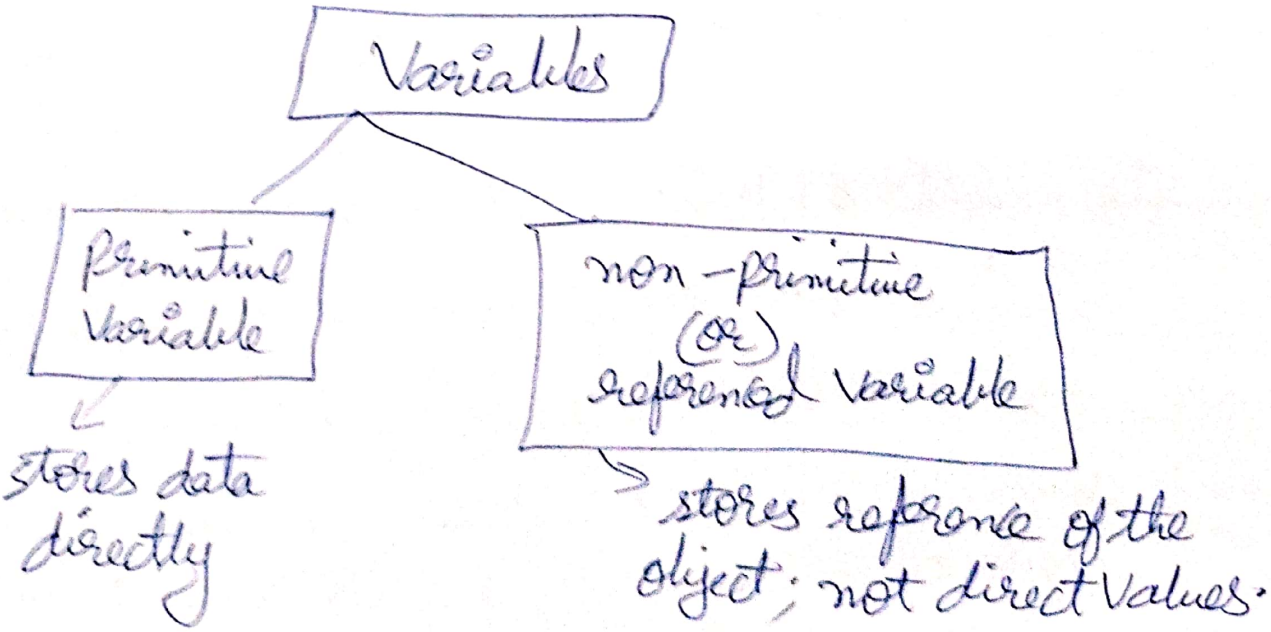
⇒ we have different type of constants like integer, floating - type, character, string etc.

integer :- int i = 100;
byte b = 10;
short s = 1;
long l = 1000;

float :- float f = 3.14f
double d = 187.786d;

Char :- char c = 'U XXX'

String: String s = "welcome to JAVA";



Rules of Variables:-

1. Variable name should not start with digit
2. Keywords should not be used as Variable name.
3. Variable name should not contain any special Symbol except underscore.

The Scope and Lifetime of Variables:- all Variables used till now have been declared at the start of the main() method. Java allows Variables to be declared within any block. A block is begun with an opening curly brace '{' and ended by a closing curly brace '}'. A block defines a scope.

```
{
  <= Scope
}
```

⇒ each time you start a new block, you are creating Scope determines what objects are visible to other parts of your program. it also determines lifetime of those objects
Scope : global and local.

⇒ The scope begins with its openning curly brace ("{")
Parameters are included within method's scope.
Variables declared inside a scope are not visible to code that is defined outside that scope.

⇒ When you declare a variable within a scope, you are localizing that variable and protecting it from unauthorized access and code.

⇒ Scopes can be nested. Ex: each time you create a block of code, you are creating a new, nested scope. That means that objects declared in the outer scope will be visible to code within the inner scope; the reverse is not true. Objects declared within the inner scope will not be visible outside it.

Effect of nested scopes:-

```
// Demonstrate block scope.
class Scope {
public static void main(String args[]) {
int x; // Known to all code within main.
x = 10;
if(x == 10) { // start a new scope.
```

```
int y = 20;  // Known only to this block
             // x any y both Known here.
System.out.println("x and y:" +x+" " +y);
x = y * 2;
}
// y = 100; // Error! y not Known here
// x is still Known here.
System.out.println("x is" +x);
}}
```

⇒ x is declared at start of main ()'s scope. within y block, y is declared. y is only visible to other code within its block.

y = 100; is commented out

⇒ variables are created when their scope is entered and destroyed when their scope is left.

⇒ if a variable declaration includes an initializer, then that variable will be reinitialized each time the block in which it is declared is entered.

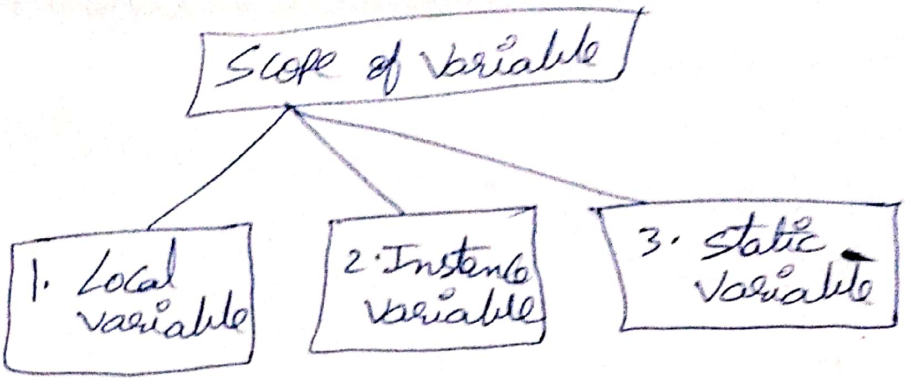
Ex:-

```
// Demonstrate lifetime of a variable.
class Lifetime {
public static void main(String args[]) {
int x;
for(x=0; x<3; x++)
{
  int y=-1; // y is initialized each time
            block is entered.
  System.out.println("y is: "+y); // this
                                   always prints -1
  y=100;
  System.out.println("y is now: "+y);
}}}
```

O/P:-

```
y is: -1
y is now: 100
y is: -1
y is now: 100
y is: -1
y is now: 100
```

⇒ y is always re-initialized to -1 each time the inner for loop is entered.



Life time of a variable is :- it is the time period between variable creation & destruction.

Eg:- B.Tech 4 years is life time of a variable.

Scope:- it is the region in which it can accessible.

Eg:- B.Tech : 2-1 semister. its life time completes after I semister in 2nd year. but variable exists still 4 years.

1. Local variable:- it is declared inside the body of the method. You can use this variable only within that method not other method.

→ local variable gets life only when its method is called & its variable creation stmt is executed. it is destroyed automatically once method execution is completed. Eg:-

```
class A
{
m1()
{int a;
}
m2()
{
}
}
```

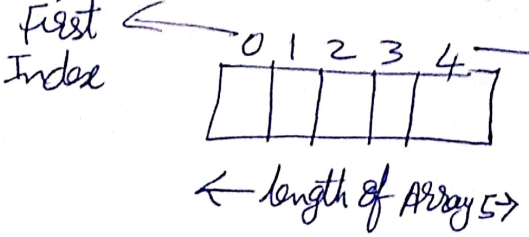
2. Instance variable :- it is declared inside the class but outside the body of the method.
⇒ when object is created, it is also created; it is destroyed when object is destroyed.

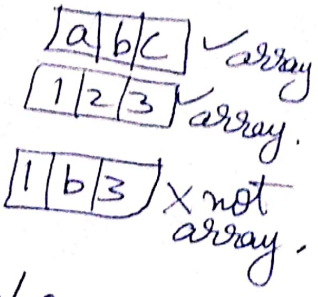
Ex:-
```
class A
{
int a;
m1()
{
}
m2()
{
}
}
```

3. Static variable :- A variable which is declared as static is called static variable.
⇒ when object is created, it is loaded.

Eg:-
```
class A
{
static int a;
m1()
{
}
m2()
{
}
PSVm()
{
}
}
```

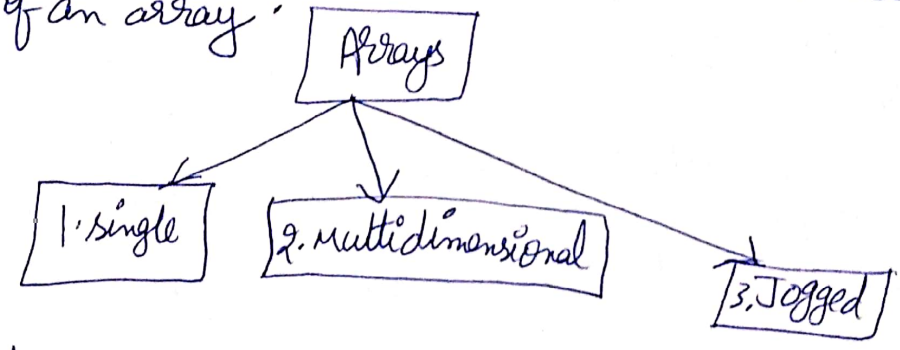
Array:- it is a collection of similar type of data, it is fixed in size, you can't increase the size of array at run time. it stores the value on the basis of index value.





Advantages:- One variable can store multiple value.

disadvantage:- size limit ⇒ once we declare the array there is no chance to increase and decrease the size of an array.



1. **Single:-** we can store single values either row or column

int[] a; int a[]; int []a;

Single Array is also called as one-dimensional array

⇒ to create an array, you first must create an array variable of the desired type. The general form of 1-D array declaration is:

type var-name[];

type declares the base type of the array.

Ex:- declare an array named month: int month[];

array-var = new type[size];

⇒ type specifies the type of data being allocated,
size specifies the number of elements in array
array-var is array variable

month = new int[12];

(or) type arr-name = new type[size];

⇒ Arrays can be initialized when they are declared.

Ex:-

```
class AutoArray{
    PSVm()
    {
      int month[]={31,28,31,30,31,30,31,31,30,31,30,31};
      SOPln("April has "+month[3]+" days!");
    }}
```

O/P:- the O/P prints the number of days in April.

⇒ Java array indexes start with zero, so the number of days in April is month[3] or 30.

2 Ex:- Avg of a set of numbers.

```
// Average an array of values.
class Average {
    PSVm()
    {
```

```
double nums[] = {10.1, 11.2, 12.3, 13.4, 14.5};
double result = 0;
int i;
for(i=0; i<5; i++)
result = result + nums[i];
S.o.p(n/"Average is" + result/5);
}
}
```

O/P:- Average is : 12.3

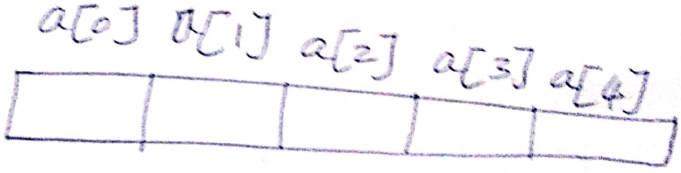
memory allocation of array :-

a = new int[size]

a = new int[5]

↓

memory allocation operator



Combining declaration & initialization :

```
int a[] = new int[5]
int a[0] = 10;
int a[1] = 20;
(or)
int a[10] = {10,20,30...}
```

2. Multi-dimensional / 2D-array:- multidimensional arrays are actually arrays of arrays. To declare a 2-D array variable, specify each additional index using another set of square brackets.

Eg:- int twoD[][] = new int[4][5];

This allocates a 4 by 5 array and assigns it to 2-D.

Declaration:- int a[][];

int[][] a;

int [][]a;

initialization:-

a= new int[3][4]

(or)

int a[][] = new int[3][4]

a[0][0]=10; a[0][1]=20;

(or)

int[][] a= {10,20,30...}

Pgm:-

```
// Demonstrate a 2-D array.
class TwoDArray {
psvm()
{ int twoD[][]= new int[4][5];
  int i, j, k=0;
```

```
for(i=0; i<4; i++)
 for(j=0; j<5; j++)
 {
   twoD[i][j]=k;
   k++;
 }
 for(i=0; i<4; i++)
 {
   for(j=0; j<5; j++)
   s.o.p(twoD[i][j]+" ");
   s.o.pln();
 }}}
```

o/p:-

```
 0 1 2 3 4
 5 6 7 8 9
10 11 12 13 14
15 16 17 18 19
```

→

```
int twoD[][]= new int[4][];
twoD[0] =new int[5];
twoD[1] = new int[5];
twoD[2]= new int[5];
twoD[3]= new int[5]
```

⇒ //Manually allocate differing size of second dimensions

```
class TwoDAgain {
 PSVm()
```

```
int twoD[][] = new int[4][];
    twoD[0] = new int[1];
    twoD[1] = new int[2];
    twoD[2] = new int[3];
    twoD[3] = new int[4];
    int i, j, k=0;
    for(i=0; i<4; i++)
    for(j=0; j<i+1; j++)
    {
      twoD[i][j]=k;
      k++;
    }
      for(i=0; i<4; i++)
      {
        for(j=0; j<i+1; j++)
        S.o.p(twoD[i][j]+" ");
      }}} S.o.pln();
```

o/p:

```
0
1 2
3 4 5
6 7 8
```

⇒ 3-D Array:

```
// Demonstrate a 3-DA.
class threeDMatrix
{
  PSVm()
  {
```

```
int threeD[][][]=new int[3][4][5];
  int i,j,k;
  for(i=0; i<3; i++)
   for(j=0; j<4; j++)
   for(k=0; k<5; k++)
  threeD[i][j][k] = i*j*k;
   for(i=0; i<3; i++)
    {
     for(j=0; j<4; j++)
      {
       for(k=0; k<5; k++)
       s.o.pln(threeD[i][j][k]+" ");
       s.o.pln();
      }
     s.o.pln();
}}}
```

op :-

```
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0

0 1 2 3 4
0 2 4 6 8
0 3 6 9 12
0 0 0 0 0

0 2 4 6 8
0 4 8 12 16
0 6 12 18 24
```

Alternative Array Declaration Syntax:- Here is a second form that may be used to declare an array:

type[] var-name;

Square brackets follow the type specifier.

```
int a1[] = new int[3];
int[] a2 = new int[3];
```

```
char twod1[][] = new char[3][4];
char[][] twod2 = new char[3][4];
```

3. Jagged Arrays (or) Array of Arrays:- it is an array with a variable number of columns in each row as different

Ex:-

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 2 & 3 & & \\ 1 & & & \\ 1 & 2 & 3 & \end{bmatrix}$$

→ Operators topic next →

Console input and output:-

1. BufferedInputStream:- it adds functionality to another input stream-namely, the ability to buffer the input & to support the mark & reset methods.

2. BufferOutputStream :- the class implements a buffered output stream.

3. BufferedReader :- Reads text from a character-input stream, buffering characters so as to provide for the efficient reading of characters, arrays and lines.

4. BufferedWriter :- writes text to a character-output stream, buffering characters so as to provide for the efficient writing of single characters, arrays and lines.

5. ByteArrayInputStream :- it contains an internal buffer that contains bytes that may be read from the stream.

6. ByteArrayOutputStream :- this class implements an output stream in which the data is written into a byte array.

7. CharArrayReader :- this class implements a character buffer that can be used as a character-input stream.

8. CharArrayWriter :- this class implements a character buffer that can be used as an ~~character-input stream~~ writer.

9. Console :- Methods to access the character-based console device, if any, associated with the current Java Virtual machine.

10. DataInputStream :- A data input stream lets an application read primitive Java data types from an underlying input stream in a machine-independent way.

11. DataOutputStream :- A data output stream lets an application write primitive Java data types to an output stream in a portable way.

12. File :- An abstract representation of file and directory pathnames.

13. FileInputStream :- it obtains input bytes from a file in a file system.

14. FileOutputStream :- it used for writing data to a file or to a FileDescriptor.

15. FileReader :- Convenience class for reading character files.

16. FileWriter :- Convenience class for writing character files.

17. FilterInputStream :- it contains some other input stream, which it uses as its basic source of data, possibly transforming the data along the way or providing additional functionality.

18. FilterOutputStream :- this class is the superclass of all classes that filter output streams.

19. FilterReader :- Abstract class for reading filtered character streams.

20. FileWriter :- Abstract class for writing filtered character streams.

21. InputStream :- this abstract class is the superclass of all classes representing an input stream of bytes.

22. InputStreamReader :- it is a bridge from byte streams to character streams; it reads bytes and decodes them into characters using a specified charset.

21. ObjectInputStream :- it deserializes primitive data and objects previously written using an ObjectOutput stream.

22. ObjectOutputStream:- It writes primitive data types and graphs of Java object to an output stream.

23. OutputStream:- It is the superclass of all classes representing an output stream of bytes.

24. OutputStreamWriter:- It is a bridge from character streams to byte streams: characters written to it are encoded into bytes using a specified charset.

25. PrintWriter:- prints formatted representations of objects to a text-output stream.

26. RandomAccessFile:- instances of this class support both reading and writing to a random access file.

27. StreamTokenizer:- It takes an input stream and parses it into "tokens," allowing the tokens to be read one at a time.

Operators:- Operators are special symbols that perform specific operations on one, two, or three operands and then return a result. All binary operators except for the assignment operators are evaluated from left to right; assignment operators are evaluated right to left

Operators	precedence
Postfix	Exp ++, Exp --
Unary	++Exp, --Exp, +Exp, -Exp ~ !
multiplicative	* / %
additive	+ -
shift	<< >> >>>
relational	< > <= >= instanceof
equality	== !=
bitwise AND	&
bitwise exclusive OR	^
~~bitwise~~ Logical AND	&&
Logical OR	||
ternary	? :
assignment	= += -= *= /= %= &= ^= |= <<= >>= >>>=

1. Arithematic : +, -, *, /, %.
2. Relational : <, >, <=, >=, ==, !=
3. logical : && (AND), || (OR)
4. Assignment : =, +=, -=, *=, <<=, >>=
5. Unary : increment ++
   decrement --
6. Ternary : ? :
7. Bitwise : & AND, ^ OR
8. Shift : <<, >>

1. Arithematic :-

+ → Add → c = a+b;
- → sub → c = a-b;
* → mul → c = a*b;
/ → div → c = a/b;
% → mod → c = a%b;

2. Relational :-

< → lessthan → a<4
> → greaterthan → b>10
<= → lessthan equalto → b<=10
>= → greaterthan equalto → a>=5
== → equal to → x==100
!= → not equalto → m!=8

3. Logical :- && ⇒ And operator → 0 && 1
   || ⇒ OR operator → 0 || 1

4. Assignment :-
   = → is assigned to → a = 5

5. Unary :-
   1. increment → ++ increment by 1 → ++i or i++
   2. decrement → -- decrement by 1 → --i or i--

6. special operators :-
   1) instance of — determining whether the object belongs to particular class or not.
      instance of operator is used
      ∈
      & .dot
      instance
      Ex: Ganga instance of River.
      obj Ganga is an obj of class River

→ to access the instance variable and methods of class objects

Ex :- customer.name → // accessing the name of the customer
customer.ID → // accessing the ID of the customer

Conditional operator :-
is " ? : " the syntax of conditional operator.

condition ? Expression1 : Expression2
Where the Exp1 denotes the True condition.
Exp2 denotes the False condition.
Ex :- a > b ? true : false.

Arithmatic operator:- the arithmatic operators are used to perform basic arithmatic operations. the operands used for those operators must be of numeric type. the boolean type operands cannot be used with arithmatic operators.

Pgm:-

```
/* Arithmatic operators */
class ArithOperDemo
{
  psvm()
  {
    s.o.pln("\n performing arithmatic operations");
    int a=10, b=20, c;
    s.o.pln("a="+a);      →a=10
    s.o.pln("b="+b);      → b=20
     c=a+b;
  s.o.pln("\n Addition of two numbers in"+c);  →30
    c=a-b;
  s.o.pln("\n Subtraction of two no. is"+c);  →10
  c=a*b;
  s.o.pln  ——————→200
  c=a/b;
}}s.o.pln  "  →0
```

Relational:- used to denote some condition, those operators establish the relation among the operators.

Pgm

```
/* Relational operators */
import java.io.*;
class RelOper
{
psvm( )
{
int a,b;
a=10;
b=20;
if(a>b)
{
s.o.pln("a is greater");
}
else
{
s.o.pln("b is greater");
}}}
```

logical:- used to combine two operators.

Pgm:

```
/* Boolean */
import java.io.*;
class LogicalOper
{
psvm("string args[])
{
```

```
boolean oper1, oper2;
    oper1 = true;
    oper2 = false;
boolean ans1, ans2;
ans1 = oper1 & oper2;
ans2 = oper1 | oper2;
S.o.println("the oper1 is "+oper1); →true
S.o.println("the oper2 is"+oper2); →false
S.o.println("the oper1 & oper2 is"+ans1); →false
S.o.println("the oper1 | oper2 is"+ans2); →true.
}
}
```

1 * 0 → 0

1 + 0 → 1

Expressions:- it is a combination of operators and operands in a specific manner.

Ex:

```
c = a + b * c; // Expression with arithmetic operators.
(a>b) && (b<c); // Expression with logical operators.
```

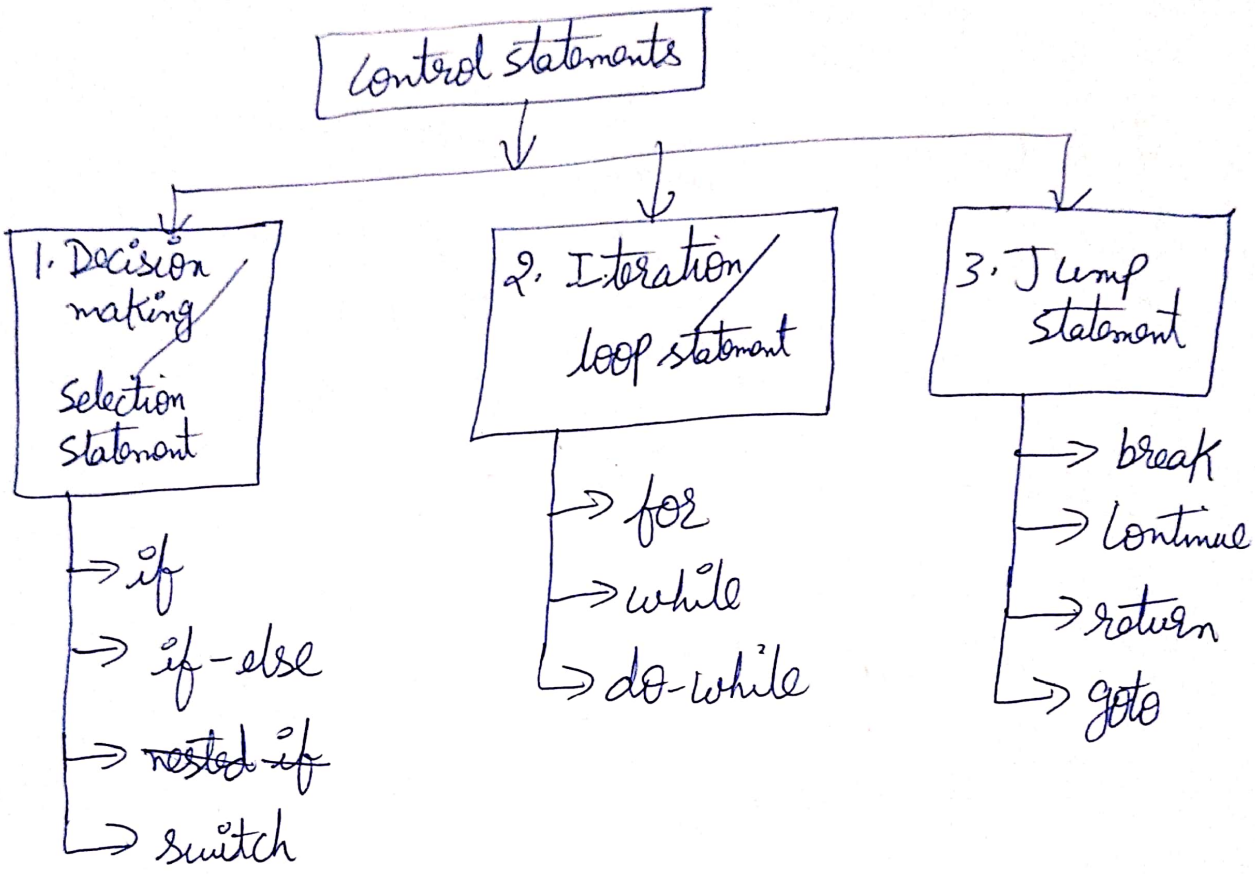
A + B = C

operators.

operands

Control statements :- Control flow :- conditional statements :-

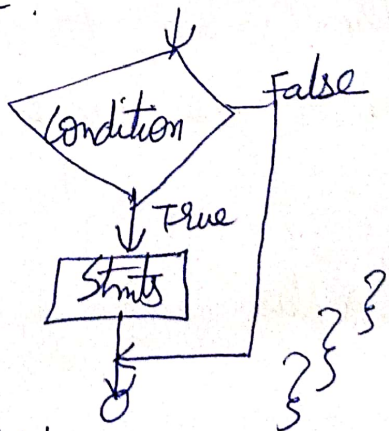
Control statements used to control the flow of code.



1. **if statement :-** this statement used to test the condition. it is also called as

general form :-

```
if(condition) statement 1;
(or)
if (condition)
{
  stmts;
}
```



Eg :-

```
class
{
 Psvm()
 {int age=20;
 if(age>18)
 {
 S.o.println("eligible for voting");
 }
 }
}
```

→ if condition is true the compiler immediately execute the if block stmts otherwise skip condition & pgm.

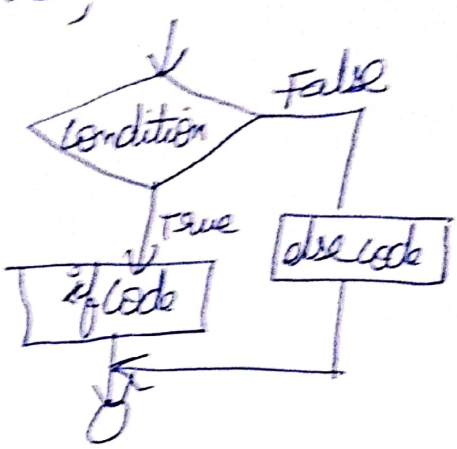
2. if-else :- used to tests the condition.

general form:

```
if (condition) stmt 1;
else stmt 2;
```

(or)

```
if (condition)
{
stmt 1;
}
else
{
stmt 2;
}
```



Ex:-

```
void applyBrakes()
{
if(isMoving){
currentSpeed--;
}
else
{
System.err.println("The bicycle has already
                    stopped!");
```

⇒ if condition is true, the compiler will ~~be~~ execute if code otherwise else block code executed.

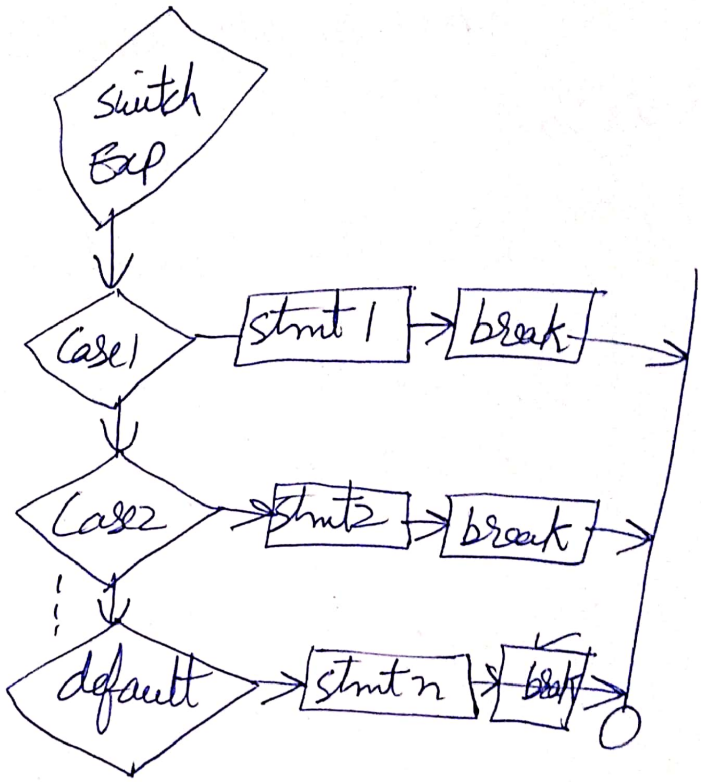
Nested if:- it represents the if block with another if block. (A class within another class is called nested).

Switch:- it executes one statement from multiple conditions.

A switch works with byte, short, char, & int primitive datatypes. it also works with enumerated types.

Syntax:-

```
switch (Exp)
{
  case value 1:
      Stmt 1;
      break;
  case value 2:
      stmt 2;
      break;
  :
  default :
}
```



Ex:-

```
class SwitchDemo
{
  PSVm ()
  { int month = 8;
    Switch (month)
    {
      case 1: s.o.pln("Jan");
              break;
      case 2: s.o.pln("Feb");
              break;
      :
      default: s.o.pln ("Invalid month");
               break;
}}}
```

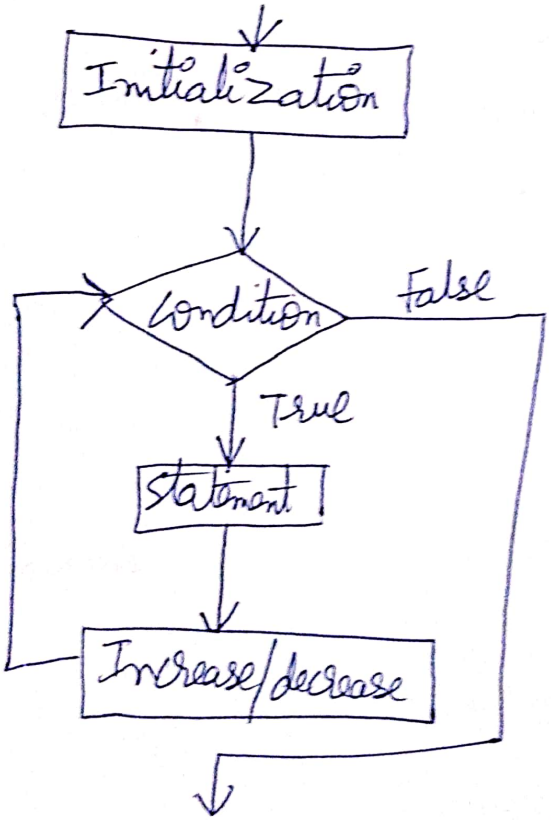
O/P:- August.

loops:- it execute a block of lines while a particular condition is true.

for loop:- loops are used to execute a set of instructions/ functions repeatedly when some conditions become true.

→ **syntax:-**

```
for( initialization ; condition; increase/decrease)
{
  statement 1;
} statement 2;
```



Eg:-

```
class Eg
{
  PSVm()
  {
    int i;
   for(i=0; i<=10; i++)
   {
    S.O.Pln(i);
}}}
```

2.

```
class factorial
{
 PSVm()
{
 int i, fact=1, n=5;
 for(i=1; i<=5; i++)
 { fact = fact*i;
 }
 S.O.Pln("factorial Values in" ;+
                    fact);
}}}
```

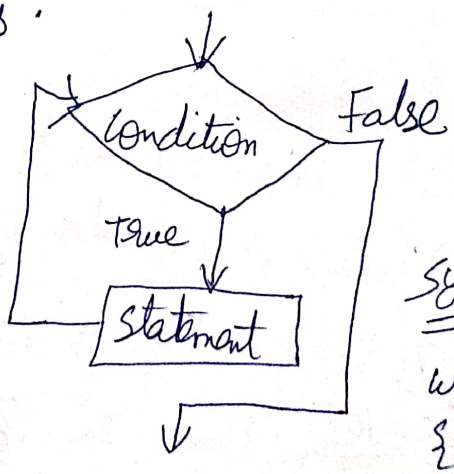
⇒ first the compiler checks the initialization after then it will check the condition, the condition is true the compiler immediately executes inner block statement.

⇒ when using the for statement, we need to remember that
  ⇒ the 'initialization' expression initializes the loop; it executed once, as the loop begins.
  ⇒ when the 'Termination' expression evaluates to false the loop terminates.
  ⇒ The 'increment' expression is invoked after each iteration through the loop; it is perfectly acceptable for this expression to increment or decrement a value.

2. **while statement** :- it is used to iterate a part of the program several times.

Syntax :-

```
while(condition)
{ statement 1;
  statement 2;
  increase/decrease;
}
```



Syntax :-

```
while(expression)
{ statement(s)
}
```

⇒ The while loop statement continually executes a block of statements while a particular condition is true.

⇒ The while statement continues testing the expression and executing its block until the expression evaluates to false.

Ex:- Print the values from 1 to 10.

```
class whileDemo
{
PSVm()
{int count=1;
while(count<11)
{
S.O.Pln("count is:"+count);
count++;
}}}
```

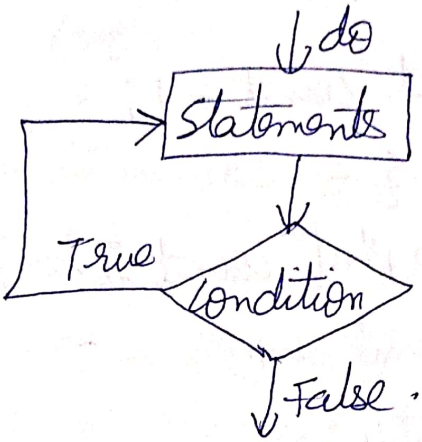
Eg2:- Pgm:-

```
class Ex
{
PSVm()
{
int i=1;
while(i<=10)
{S.O.Pln(i);
i++;
}}}
```

Eg3:-

```
class Factorial
{
PSVm()
{int i=1, fact=1, no=5;
while(i<no)
{
fact = fact * i;
i++;
}
S.O.Pln("Factorial value", +fact);
}}
```

3. do-while :- it is used to iterate a part of the program several times. In Java do-while is executed at least once because condition is checked after loop body.



Syntax :- 1.

```
do
{
  stmt 1;
  stmt 2;
  increase/decrease;
}
while (condition).
```

(or)

Syntax 2:-

```
do {
   statement(s)
} while(expression);
```

⇒ the difference between do-while and while is that do-while evaluates its expression at the bottom of the loop instead of the top. the statements within the do block are always executed at least once

⇒ in do-while loop the compiler first process the do statements after that it will check the compiler condition is true the compiler will execute do statement.

Eg:- 1. Pgm:-

```
class DoWhile
{
 P.SVm( )
 {int count = 1;
 do {
 S.o.pln("count is:" + count);
  count ++;
 }
 while (count <= 11);
}}
```

Ex:- 2:-

```
class do while
{
 PSVm( )
 {int count = 1, i=0;
 do
 { i= i+1;
 S.o.pln("the value of i", +i);
  count ++;
 }
 while (count <= 5);
}}
```

Ex 3:-

```
class dowhile
{
PSVm( )
{ int i = 1;
 do
{ S.o.pln(i);
 i ++;
}
while (i <= 10);
}}
```

Jumping statements :-

1. Break :- it is a pre-defined keyword, it is used to Break the current loop.

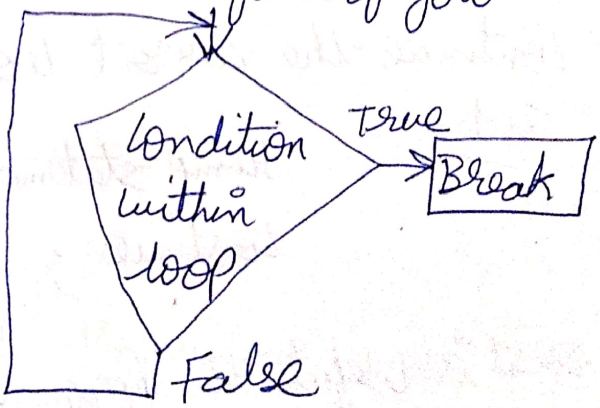
→ we can force immediate termination of a loop, by passing the conditional, when we apply a break statement, the loop is terminated & the program control resumes at the next statement.

Uses of break :-

1. it terminates a statement sequence in ~~which~~ switch statement.
2. it can be used to exit a loop.
3. it can be used as a civilized form of goto.

Syntax :-

```
Jump-statement ;
    break ;
```



Ex:-

```
class Break
{
 PSVm()
{
for(int i=1; i<=10; i++)
{ if(i==5)
  {
   break;
  }
 S.O.pln(1);
}}}
```

Ex:-

```
class BreakWhile
{
 PSVm()
 {
  int i=1;
 while (i<=10)
 { if (i==5)
 { i++;
 }break;
 }
 S.O.pln(i);
}}} i++
```

Continue statement :- it is a keyword, used to continue the current loop.

Syntax :- Jump statement;
continue;

⇒ it is useful to force early termination. it continue running the loop.

Ex:-

```
class Continuetx
{
psvm()
{
for(int i=1; i<=10; i++)
{
if(i==5)
{
continue; //it will skip to the test stmt
}
s.o.println(i);
}}}
```

o/p:-

```
1
2
3
4
6
7
8
9 10
```

(5)

2) loop is this continued when it reaches to 5.

Eg2:

```
class Continuewhile
{
psvm()
{
int i=1;
while(i<=10)
{
if(i==5)
{
i++
continue; //skip the test condition
}
s.o.println(i)
i++;
}}}
```

difference between Break & continue:-

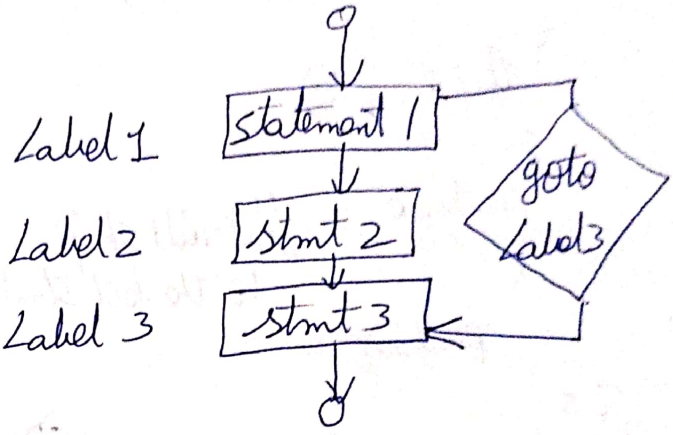
Break	continue
1. the break is used to terminate the Execution.	1. the continue is not used to terminate the execution.
2. it breaks iteration.	2. it skips the iteration.
3. Break is used in loops & switch.	3. continue is used only in loop.

goto statement :- this stmt provides an Un-conditional Jump from the goto to a labeled stmt in the same function.

⇒ the compiler Jumps from one stmt to another stmt.

Syntax :-

```
goto Label;
.....
Label: statement;
```



Ex :-

```
class goto
{
  PSVm()
  {
  int count=0;
  loop: count++;
S.O.Pln("count in"; +count);
while(count ≤ 5)
  {
S.O.Pln(count);
  goto : loop;
  }
}}}
```

op:

```
0
0
1
2
2
```

Return statement :- it is a keyword, it indicates a function should return a value. the value is either Positive (+ve) or negative (-ve).

Syntax :- Return Label;

TYPE Conversion :- Converting a value from one datatype to another data type is called Type Conversion or Implicit Conversion or Automatic Conversion.

→ it is used to convert small data type to large data type.

→ data loss is not there.

→ it doesn't convert numeric data type to character or boolean.

⇒ (1,2,3,4) —not converted→ X (a, b, c, d) —not converted→ X (True/false)

→ when you assign value of one data type to another, if the data types are compatible then Java will perform the conversion automatically.

Byte → Short → int → long → float → double.

Small —convert→ to —convert→ large

Ex:-
```
class Test
{
PSVM()
{
int i=100; //automatic type conversion
long l=i;
float f=l;
S.O.Pln(" "+i); → 100
S.O.Pln(" "+l); → 100
S.O.Pln(" "+f); → 100.0
}
}
```

⇒ we can assign a value of one type to a variable of another type. if the two types are compatible, then Java will perform the conversion automatically.

Ex:- it is always possible to assign an int value to a long variable; not all types are compatible, not all type conversions are implicitly allowed. there is no conversion defined from double to byte.

⇒ But it is possible for conversion between incompatible types. to do so, you must use a cast, which performs an explicit conversion between incompatible types.

Java's Automatic Conversions:-

When one type of data is assigned to another type of variable, an automatic type conversion will take place if the following two conditions are satisfied:

1. The two types are compatible.
2. The destination type is larger than the source type.

→ When these two conditions are met, a widening conversion takes place. Ex:- the int type is always large enough to hold all valid byte values, so no explicit cast statement is required.

For widening conversions, the numeric types, including integer and floating-point types, are compatible with each other; the numeric types are not compatible with char or boolean. (1,2,3) $\xrightarrow{X}$ (a,b,c) or (T/F).
Also, char and boolean are not compatible with each other. (a,b,c) $\xrightarrow{X}$ (True/False).

Java also performs an automatic type conversion when storing a literal integer constant into variables of type byte, short, or long.

Type Casting (or) Casting Incompatible types:-

the conversion is done explicitly by the User.

here we want to assign a value of larger data type to smaller data type.

→ at type conversion some data may be lost.

Double → float → long → int → short → byte.

larger data —convert→ to —convert→ smaller data.

Ex:-

```
class Simple
{
PSVm()
{
Float f=10.5f;
int a=f; //compiletime error
int a=(int)f;
S.O.Pln(f); → 10.5
S.O.Pln(a); → 10
}
}
```

→ the Automatic type conversions are helpful, they will not fulfill all needs. Eg:- if we want to assign an int value to a byte variable.

→ this conversion will not be performed automatically, because a byte is smaller than an int. This kind of conversion is sometimes called narrowing conversion; since you are explicitly making the value narrower so that it will fit into the target type. To create a conversion between two incompatible types, you must use a cast. A cast is simply an explicit type conversion.

general form:- (target-type) value

Here, target-type specifies the desired type to convert the specified value to.

Ex:-

```
int a;
byte b;
//.....
b=(byte) a;
```

→ A different type of conversion will occur when a floating-point value is assigned to an integer type; truncation. As integers do not have fractional components, so when a floating-point value is assigned to an integer type, the fractional component is lost.

Pgm:-

```
class Conversion
{
 PSVm()
 {
  byte b;
  int i=257;
  double d=323.142;
  S.o.pln("\n Conversion of int to byte.");
  b=(byte) i;
  S.o.pln("i and b" +i+" "+b);
  S.o.pln("\n Conversion of double to int.");
 i=(int)d;
  S.o.pln("d and i "+d+" "+i);
  S.o.pln("\n conversion of double to byte.");
  b= (byte) d;
  S.o.pln("d and b "+d+" "+b);
}}
```

o/p:

```
Conversion of int to byte.
    i and b  257  1
Conversion of double to int
  d and i  323.142  322
Conversion of double to byte.
  d and b  323.142  67
byte b=50;
  b=(byte)(b*2);
```

The Type promotion Rules:

⇒ In addition to the elevation of bytes and shorts to int, Java defines several type promotion rules that apply to expressions.

1. all byte and short values are promoted to int;
2. than, if one operand is a long, the whole expression is promoted to long. if one operand is a float, the entire expression is promoted to float. if any of the operands is double, the result is double. the following pgm demonstrates how each value in the expression gets promoted to match the second argument to each binary operator:

```
class promote {
Psvm() {
byte b=42;
char c='a';
short s=1024;
int i=50000;
float f=5.67f;
double d=.1234;
double result = (f*b)+(i/c)-(d*s);
S.o.pln((f*b)+" + "+(i/c)+" - "+(d*s));
} S.o.pln("result="+result);
}
double result=(f*b)+(i/c)-(d*s);
```

⇒ in the first sub expression, f * b, b is promoted to a float and the result of the sub expression is float. next, in sub exp i/c, c is promoted to int, and the result is of type int. then, in d * s, the value of s is promoted to double, and the type of the sub expression is double.

Enumerated types:- An enumerated datatype is another user-defined data type, which provides a way for attaching names to numbers, thereby increasing of the code. It is a legacy collection interface. which is used to iterate through legacy collections. it has the following methods.

nextElement() returns value of an elements.

hasMoreElements() returns true if elements are present.

the syntax is similar to struct statement.

Eg:-

```
enum shap {circle, square, triangle};
enum color {red, blue, green, yellow};
```

⇒ "attaching names to numbers"

Concepts of classes, objects simple java programs:

Concepts of classes: → class is a collection of data & functions that manipulate the data.

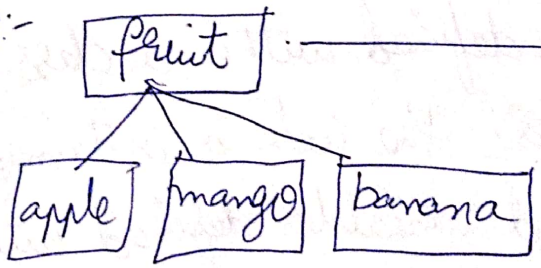
class:- A class is a blueprint or prototype from which objects are created. A class that models the states and behavior of a real-world object.

⇒ class is a template or a blue print which does not occupy any memory location. A class consists members (data and methods). when you define a class, you declare its exact form and nature. it specifies the data it contains and code operates on data.

⇒ A simple class may contain only code or only data, most real-world classes contain both.

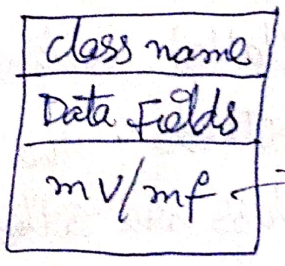
⇒ A class code defines the interface to its data.

⇒ A class is declared by use of class keyword.

Eg:- 

→ class.

Class represented as



Syntax:-

```
class <class name>
{
  m.v/m.f;
}
```

→ data components of class are called data Fields & function components as member functions & variables.

Class defination :-

```
class classname {
  type instance-variable 1;
  type instance-variable 2;
  //....
  type instance-variable N;
  type methodname 1(parameter-list) {
    // body of method
  }
  type methodname 2(parameter-list) {
    // body of method
  }
  //...
  type methodname N(parameter-list) {
    // body of method
  }
}
```

→ the data, or variables, defined within a class are called instance variables. the code is contained within methods. the methods and variables defined within a class are called members of the class.

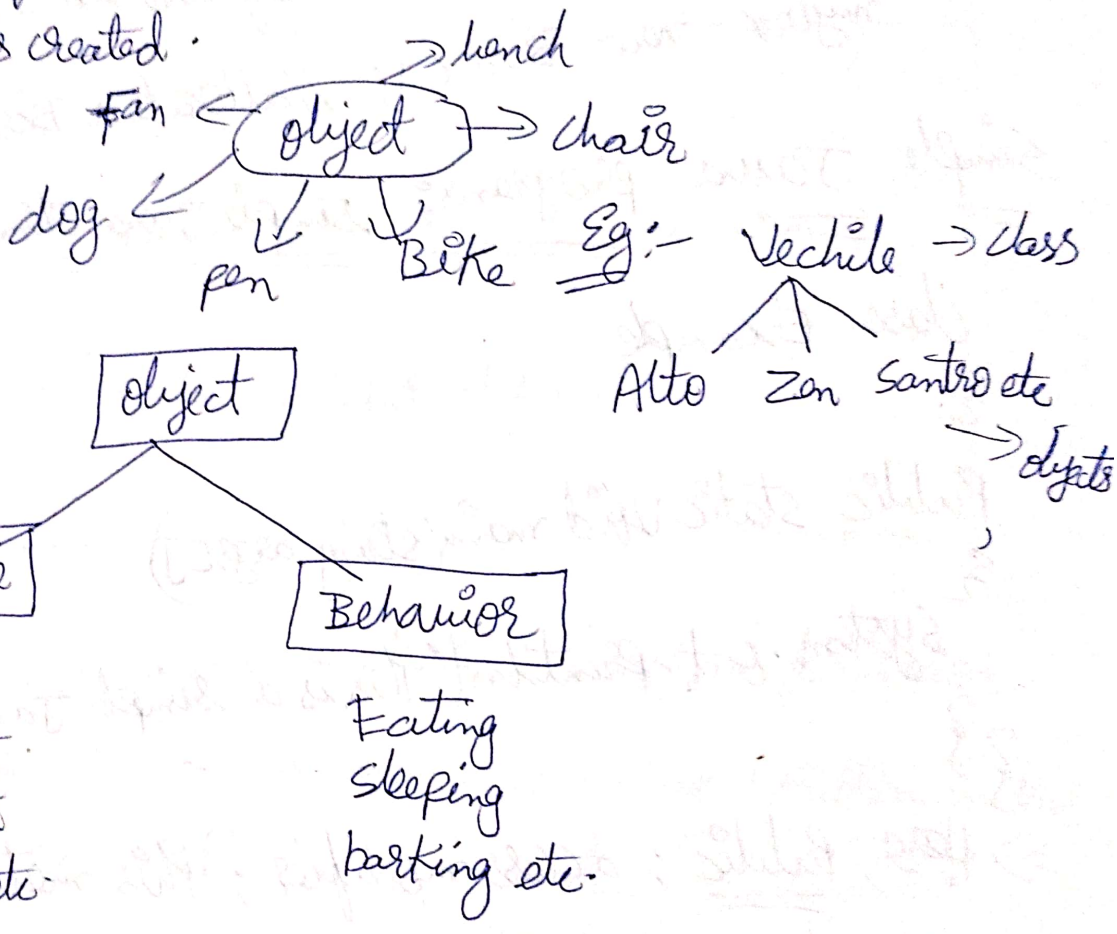
→ all methods have general form as main().

→ most methods will specified as static or public.

→ Java classes do not need to have a main() method.
→ class is the starting point of your program.

object :- object is an instance of a class, the object represents the real-word entity. the object has state and behavior.

→ memory for class variables are allocated only after object is created.



Syntax:- classname objectname = new classname();

ex:- Student s = new Student();
s.rollno;
s.marks;
s.total();

→ In Java, all class objects must be dynamically allocated.

Ex: an object of type Box;

Box mybox = new Box();

→ this combines the two steps.

Box mybox; // declare reference to object

mybox = new Box(); // allocate a Box object.

Simple Java programs: simple Java stand alone pgm:

```
class Example
{
Public static Void main(String args[])
{
  System.out.println("This is a simple Java program");
}}
```

→ Here, Public : access modifier; this method can be accessed by any one outside the class.

static: static allows the main() method to be called without initiating any instance for the class.

Void: tells the compiler that main() does not return anytype.

String args[]:- declares a parameter named args, which is an array of instances of class string.

args:- takes the arguments for a command line.

simple Java program :- structure :- (or)

Documentation section
package statements section
import statements section
class defination

class
{
Public static void main(String args[])
{
// main method defination
}}

1. Documentation section :- it provides the information about the source program. everything written in this section is written as comment.
2. Package section :- it consists name of the package by using the keyword "Package".
3. import section :- all required Java API can be imported by this import statement.

4. Class section :- this section contains the defination of the class. this class normally contains the data & method manipulating the data.

Writing single Java program :-

```
/* This is my 1st Java program */
import java.io.*;
class FirstProgram {
    public static void main(String args[])
    {
        System.out.println("Hello");
    }
}
```

O/P :- Hello.

→ any Java program can be written using simple text editor like notepad. the file name should be same as the name of the class.

→ the extension of file name should be

.Java → Firstprogram.Java

→ the o/p of this program can be generated on command prompt using the commands.

→ Javac → compiles to check errors

Javac — Firstprogram.Java ↵

⇓

checks errors

→ Java filename → execute output.

Java — Firstprogram ↵

class:- it is the first statement in Java application it contains collection of member variables and member functions.

→ the name of the class & name of your Java program should be same. the class defination should be written in within the curly brackets '{ }'.

Public :- it is a Access specifier, it is used for accessing the values anywhere.

static :- it indicates the compiler to check the main method only once.

void :- it says return type of a value.
void = returns empty value
int = returns value.

Main() :- it is a method or function.

String args[] :- it indicates storing all type of arguments in array.

. (dot) :- it is used for accessing the value from anywhere.

out :- it connects to output console.

println :- used to print the correct text.

'{' :- statements starting block

'}' :- statements ending block.

';' :- end of line / statement

Constructors, methods, Access control:-

Constructor:- it is used to create objects of a class. it is a special member method, it will automatically executed by the JVM.

→ Constructors are mainly created for initializing the object. Constructors are mainly used to eliminate default values by userdefined values.

Syntax:-
```
class Name
{
}
```
≡

Ex:- `Rectangle r = new Rectangle();` (Rectangle() ↓ Constructor)

Eg:-
```
class A
{
int a;
Float b;
char ch;
String str;
boolean bl;
}
```

default Constructor →

// before using constructor

Default values.

int → 0
Float → 0.0
char → one space
String → null
boolean → False

parametrized Constructor →

// after using constructor:

```
A()
{
a=10;
b=20.0;
ch='m';
str="java";
boolean="true";
}
```

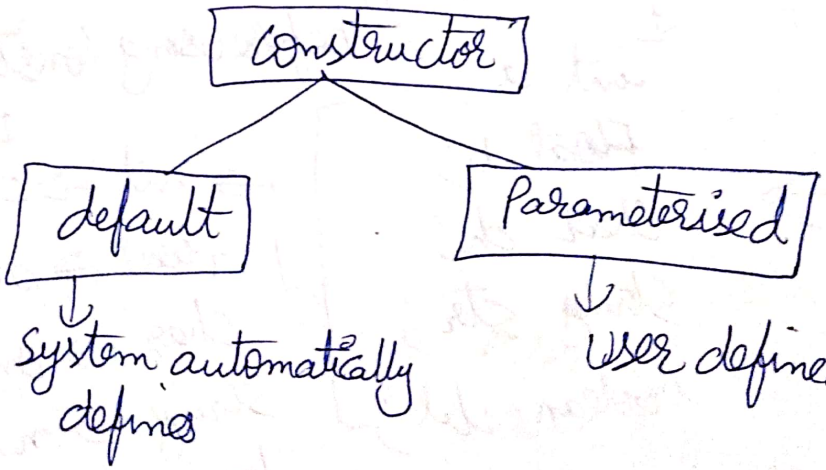
10 int
0.0 20.0 float
m char
null java String
False True boolean

→ A class contains constructors that are invoked to create objects from the class blueprint.

→ Constructor declarations look like method declarations- except that they use the name of the class and have no return type.

→ A constructor initializes an object immediately upon creation.

Rules:-
1. it is called automatically when the object is created.
2. its name must be similar to name of the class.
3. it should not return any value.



default constructor is called when an instance is created for a class.

Ex:-

```
class demo{
 int x;
 int y;
 float z;
 demo(){
 x=1;
 y=2;
 }z=3;
 void display(){
 S.O.Pln("values of x,y and z are: "+x+" "+y+" "+z);
}}
class demomain
{
 PSVm()
 {
 demo d1=new demo(); // this is a class for the above
                        default constructor.
 d1.display();
}}
```

Parameterized constructor :- user defines values.

```
class demo{
int x;
int y;
float z;
demo(int x1, int y1, int z1){
x=x1;
y=y1;
z=z1;
}
void display(){
S.O.pln("values of x,y and z are:"+x+" "+y+" "+z);
}}
class demomain{
PSVm(){
demo d1=new demo(1,2,3); // this is a call for the above parameterized constructor
d1.display();
}}
```

Method:- Method is a block of code which only runs when it is called.

⇒ Method is a sub-block of a class that is used for implementing logic of an object's operation.

Rule: logic must be placed only inside a method, not directly at class level. if we place logic at class level compiler throws error.

⇒ method must be declared within class followed by paranthesis '( )'.

Syntax:-

```
class classname
{
declaration of m.v;
declaration of method.f;
{
=
}
}
```

Methods uses:-

1. decrease Line of code
2. Re-add
3. Repeat

Ex:-

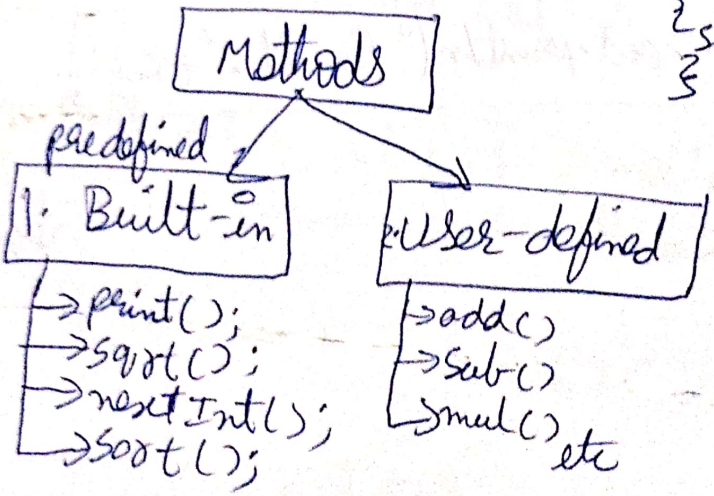
```
class Myclass
{
static void mymethod()
{
=
}
}
```

mymethod → name of the method.

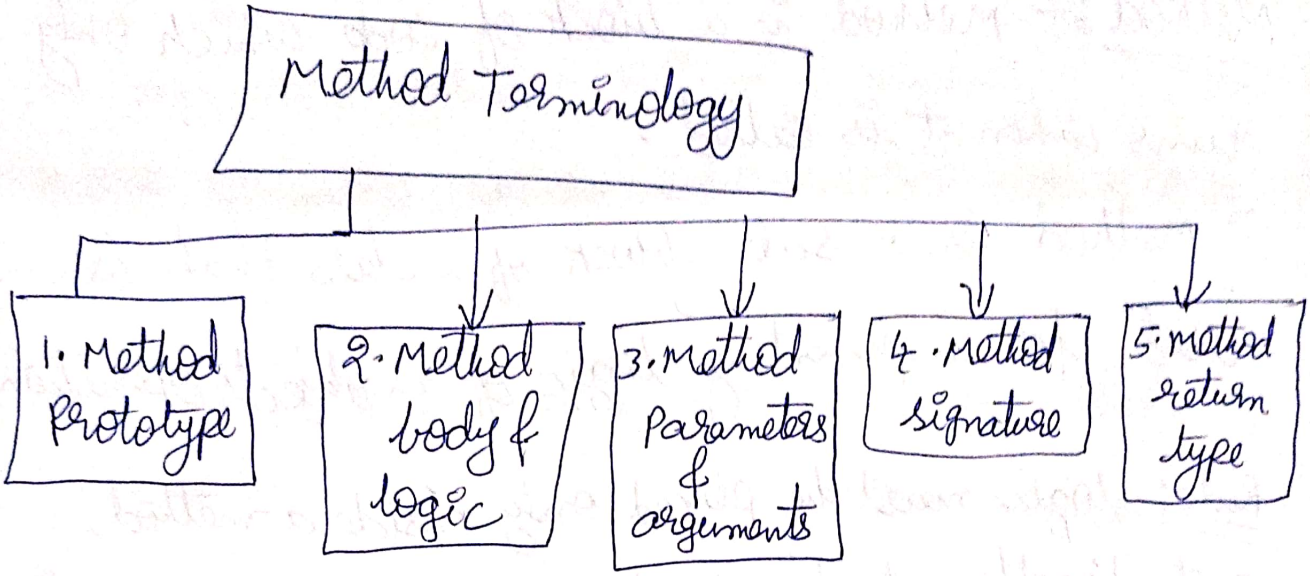
Syntax 2:-

```
returntype methodname(parameters)
{stmts
}
```

Ex: m1()
{a=b+c;
}



→ topic goto Pg:49.



1. Method prototype :- the head portion of the method is called method prototype.

2. Method body & logic :- the " { } " region is called method body, and the statements placed inside method body is called logic. { ==> logic }

Eg:-

```
static void add(int a, int b)
{
    int c=a+b;
    System.out.println("Result:"+c);
}
```

method prototype

method body

method logic

3. Method parameters & arguments :-

The variables declared in method parenthesis " ( ) " are called parameters. We can define method with 0 to n number of parameters.

→ The input values passing to parameters are called arguments.

Ex :-

```
Void add (int a, float b)
{
}
```

method parameters

method arguments.

```
add (50, 60.0f);
```

method arguments

```
add (60.0f, 50);
add (50, 70.0);
```

✗ not arguments, it not followed parameter oder & type.

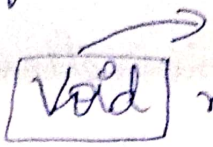
4. Method signature :- the combination of [method name + parameter list] is called method signature.

Eg :-

```
Void add (int a, int b)
{
}
```

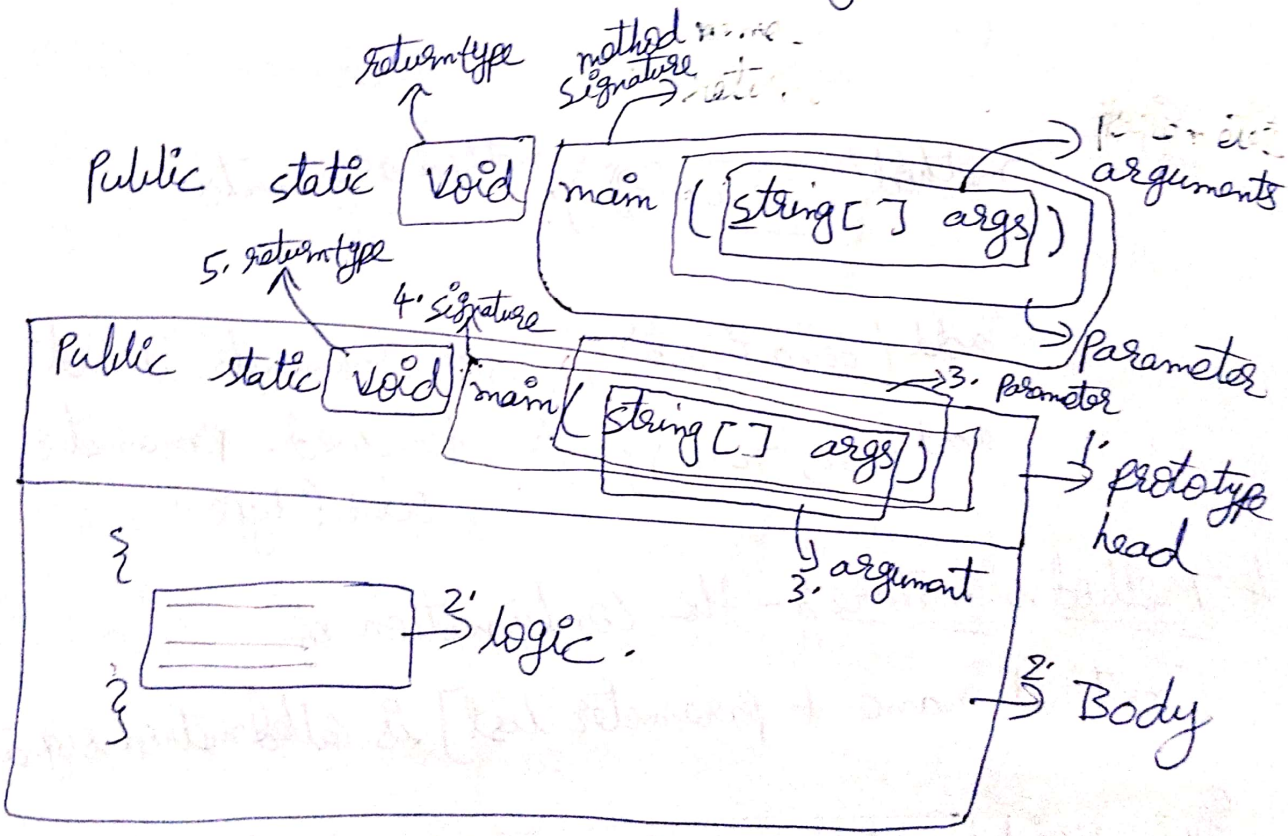
→ method signature.

5. method return type :- the data type keyword that is placed before method name is called method return type.



→ it tells the compiler, JVM & developer about the type of the variable is returned from this method after its execution.

→ if we don't want to return a value for a method we must use void keyword.



→ A method is a set of code which is referred to by name and can be called (invoked) at any point in a program simply by utilizing the method's name. Think of a method as a sub-program that acts on data and often returns a value.

→ Each method has its own name. When that name is encountered in a program, the execution of the program branches to the body of that method. When the method is finished, execution returns to the area of the program code from which it was called, and the program continues on to the next line of code.

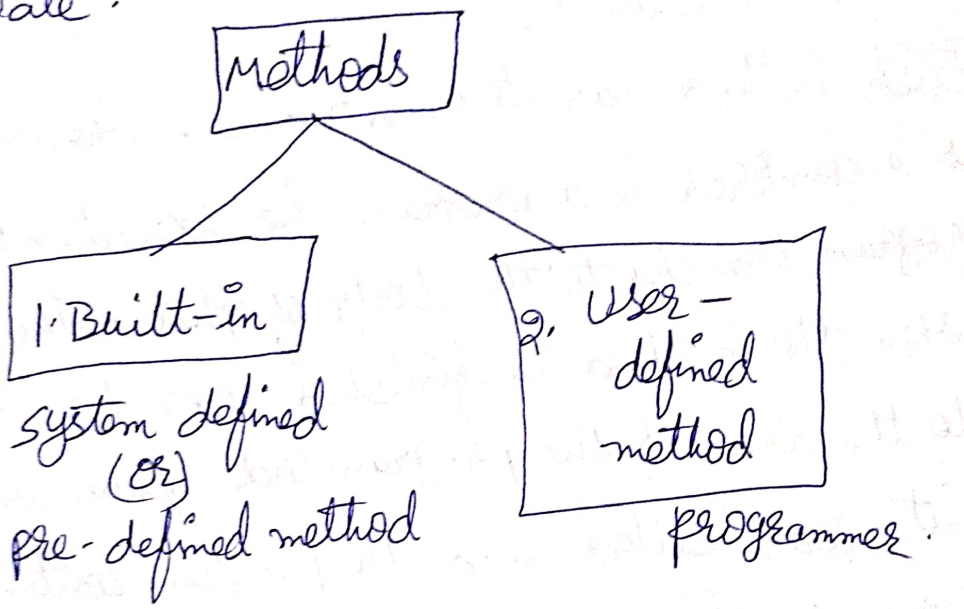
There are two basic types of methods:

1. Built-in: Build-in methods are part of of the compiler package, such as

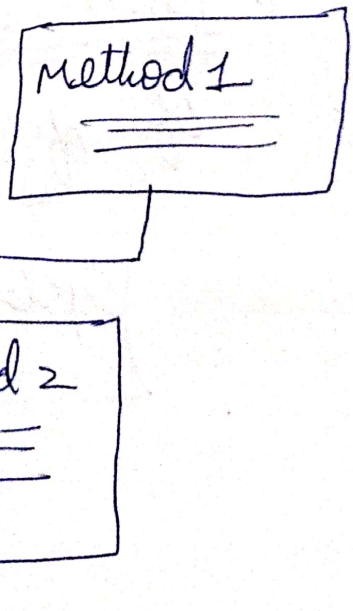
`System.out.println()` and

`System.exit(0)`.

2. User-defined :- User-defined methods are created by you, the programmer. These methods take-on names that you assign to them and perform tasks that you create.




```
Public static void main(String[] args)
{
  statement;
  method1();
  statement;
  method2();
  statement;
}
```



⇒ there are 3 types of methods

Using → 1. name of the method.
2. Return type of the method
3. Parameter passes to the method.

1. Using method:-

```
class A
{
 int x=10;
 void display()
 {
  sopln(x);
 }
 PSVm()
 {
  A oa=new A();
    oa.display(); →10
    oa.x=200;
    oa.display(); →200
 }
}
```

2. Using return type:-

```
class A
{
 int x=10;
 int display()
 {
  sopln(x);
  return x;
 }
 PSVm()
 {
  A oa=new A();
    oa.x=20;
  int a=oa.display();
    oa.x=200;
    oa.display();
 }
}
```

3. Using Parameter Passing :-

```
Class A
{ int i;
  int display(int x)
   { i=x;
     return i;
   }
   PSVm()
   {
     A oa=new A();
     int c=a.display(100);
     Sopln(c);
   }
}
```

Access control :- / specifiers / modifiers :-

The keywords which define accessibility permissions are called access modifiers. Access specifiers are applied before data members or data methods. These are used to say where to access & where not to access.

→ Java supports 4 accessibility levels to define accessibility permissions at different levels.

1. only within the class.
2. only within the package.
3. outside the package but only in subclass by using the same subclass object.
4. from all places of project.

we had 4 types of **Access specifiers**

- 1. Public
- 2. private
- 3. protected
- 4. default

1. Public :- Public members can be accessed anywhere.

→ it is used for accessing the member variables & member functions inside or outside class defination.

→ those members can be accessible from all places of Java application.

Syntax : Public :
statements ;

Note :- these code can be seen by all members/People/Public.

2. Private :- private members can be accessed only within the class in which they are declared.

→ these are used for accessing the member variable values only inside class or within the class.

Syntax :- private :
statements ;

Note :- the code can be seen by only class members.

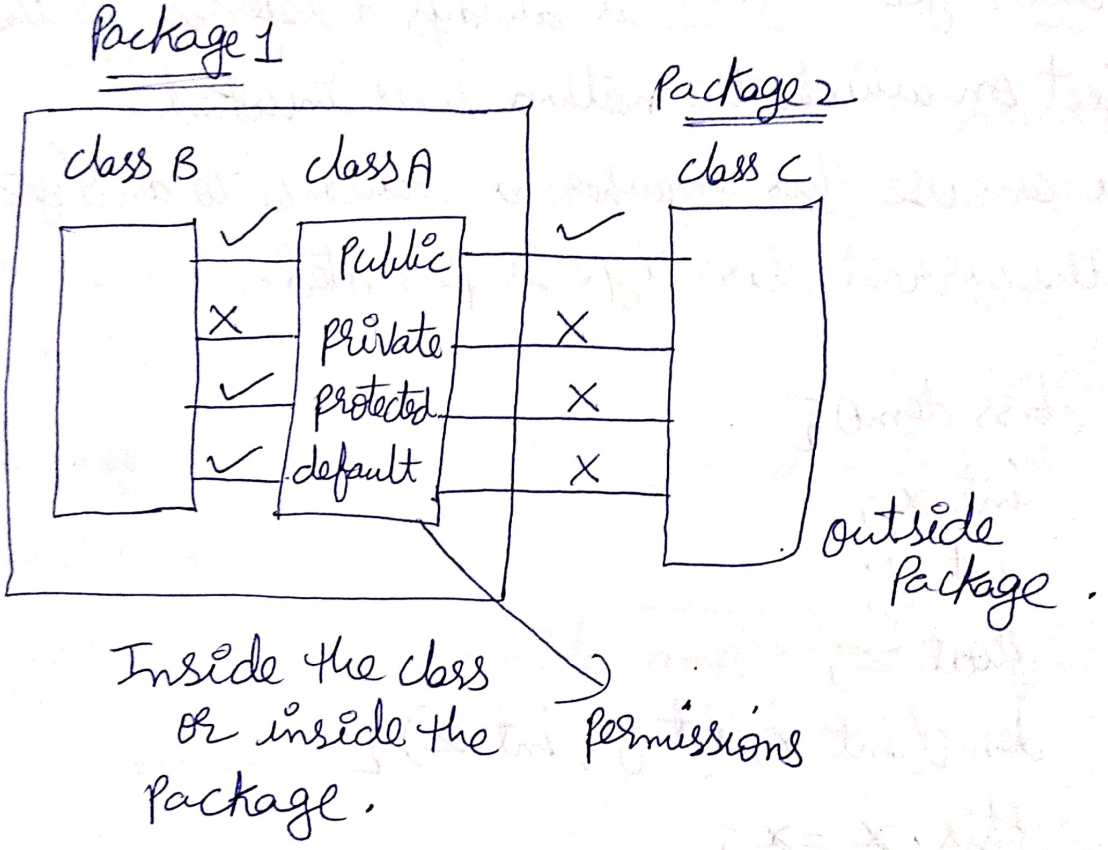
3. protected :- these are used for accessing the member variables & member functions only inside the class.

Syntax : protected :
statements ;

Note :- Only we can see the code.

4. default :- if Public, Private, protected keywords are not written system consider default specifier

⇒ default members are accessed by the method in their own class & in sub-classes of same package.



This Keyword, Garbage collection :-

This Keyword :- **this reference :-** sometimes a method will need to refer to the object that invoked it. To allow this, Java defines the this keyword.
this can be used inside any method to refer to the current object. this is always a reference to the object on which the method was invoked.
You can use this anywhere a reference to an object of the current class type is permitted.

Ex:-

```
class demo{
    int x;
    int y;
    float z;
    demo(int x, int y, int z){
    this.x = x;
    this.y = y;
    this.z = z;
}
void display(){
s.o.pln("values of x, y and z are: "+x+" "+y+" "+z);
}}
class demomain {
```

→

```
P.S.Vm() {
demo d1 = new demo(1,2,3);   //this is a call for the above parameterized constructor.
d1.display();
}
}
```

o/p:-

Values of x, y and z are: 1 23

→ to differentiate between local & instance variables we have used this keyword in the constructor.

→ this is a keyword used for accessing the return values inside the constructor. this is a non static keyword.

→ it refers current object inside a method or constructor.

Syntax:

```
this.Variablename;
```

Ex:-

```
class A
{
}
A r=new A();
```

→ for every class a unique reference id is generated, that unique reference is refered by object. this keyword is also refer to the reference id of the class

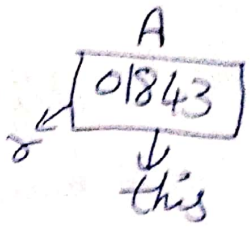
Ex:-

```
class A
{
psvm()
{
  A r=new A();
  sop(r);
}}
```

o/p:- 01843

Ex2:

```
class
{
void show()
{
sop(this);
}
psvm()
{
A r=new A();
sop(r);
r.show();
}}
```



o/p:- 01843 → r
01843 → this

⇒ this is used to call default constructor, as well as parameterized constructor.

⇒ we cannot use this keyword in static members, it leads to compile time error; since it is non-static variable.

Ex:-

```
class Ex
{
int x=10;
int y=20;
void m1()
{
S.O.Pln(this.x);
S.O.Pln(this.y);
}
public static void main(String args[])
{
S.O.Pln(this.x);
S.O.Pln(this.y);
}}
```

this is a non-static keyword ✓

this is a non-static keyword
Here, static is present - throws Error.

Garbage collector :- In Java when an object is no longer in use (or) when an object is un-referenced, then garbage collector automatically destroys the object & release memory of that object.

⇒ Since objects are dynamically allocated by using the new operator, objects are destroyed and their memory released for later re-allocation.

⇒ In some languages, such as C++, dynamically allocated objects must be manually released by use of a delete operator. Java takes a different approach. it handles de-allocation for you automatically. the technique that accomplishes this is called garbage collection.

⇒ it works like this: when no references to an object exist, that object is assumed to be no longer needed, and the memory occupied by the object can be reclaimed. there is no explicit need to destroy objects as in C++.

⇒ the finalize() method sometimes an object will need to perform some action when it is destroyed

Ex:- If an object is holding some non-Java resource such as a file handle or window character font, then you might want to make sure these resources are freed before an object is destroyed.

⇒ to handle such situations, Java provides a mechanism called finalization. By using finalization, you can define specific actions that will occur when an object is about to be reclaimed by the garbage collector.

⇒ to add a finalizer to a class, you simply define the finalize() method. the Java run time calls that method whenever it is about to recycle an object of that class. inside the finalize() method you will specify those actions that must be performed before an object is destroyed. the garbage collector runs periodically, checking for objects that are no longer referenced by any running state or indirectly through other referenced objects. Right before an asset is freed the Java run time calls the finalize() method of the object. general form:-

```
protected void finalize(){
} // finalization code here
```

Here, keyword protected is a specifier that prevents access to finalize() by code defined outside its class.

Overloading methods and constructors :-

Overloading methods :- Java supports overloading methods and Java can distinguish between methods with different signatures. methods within a class can have the same name if they have different parameter list.

⇒ Suppose that you have a class that can use calligraphy to draw various types of data (strings, integers, etc) and that contains a method for drawing each data type. it is cumbersome to use a new name for each method —

Ex:- drawString, drawInteger, drawFloat, etc. in Java programming language, you can use the same name for all the drawing methods but pass a different argument list to each method. Thus, the data drawing class might declare four methods named draw, each of which has a different parameter list.

Ex:->

Ex:-

```
Public class DataArtist {
...
  Public void draw(String s) {
  }....
  Public void draw(int i) {
  }....
  Public void draw(double f) {
  }...
   Public void draw(int i, double f) {
   .....
  }}
```

overloaded methods are differentiated by the number and the type of the arguments passed into the method. in the code sample, draw(String s) and draw(int i) are distinct and unique methods because they require different argument types. you cannot declare more than one method with the same name and the same number and type of arguments, because the compiler cannot tell them apart. the compiler does not consider return type when differentiating methods, so you cannot declare two methods with the same signature even if they have different return type.

Overloading Constructors:

We can overload constructor methods

```
class Box {
  double width;
  double height;
  double depth;
    // this is the constructor for Box.
  Box(double w, double h, double d) {
    width = w;
    height = h;
    depth = d;
  }
    // Compute and return volume
  double volume() {
    return width * height * depth;
  }
}
```

As you can see, the Box() constructor requires three parameters. That all declarations of Box objects must pass these arguments to the Box() constructor.

Ex:- the following statement is currently invalid:

```
Box ob = new Box();
```

Since Box() requires three arguments.

```
/* Here, Box defines three constructors to initialize
the dimensions of a box various ways */
class Box {
  double width;
  double height;
  double depth;
  // constructor used when all dimensions specified
  Box(double w, double h, double d) {
    width = w;
    height = h;
    depth = d;
  }
  // constructor used when no dimensions specified
  Box() {
    width = -1; // use -1 to indicate
    height = -1; // an uninitialized
    depth = -1; // box
  }
  // constructor used when cube is created
  Box(double len) {
    width = height = depth = len;
  }
  // compute and return volume
  double volume() {
```

```
    return width * height * depth;
  }
}
class OverloadCons{
public static void main(){
// create boxes using the various constructors.
Box mybox1 = new Box(10,20,15);
    Box mybox2 = new Box();
    Box mycube = new Box(7);
    double vol;
    // get volume of first box
   vol = mybox1.volume();
   S.o.println("volume of mybox1 is "+vol);
     // get volume of second box
   vol = mybox2.volume();
    S.o.println("volume of mybox2 is "+vol);
      // get volume of cube
   vol=mycube.volume();
  S.o.println("volume of mycube is"+vol);
}}
```

o/p :-

```
Volume of mybox1 is 3000.0
Volume of mybox2 is -1.0
Volume of mycube is 343.0
```

Method overloading

1. overloading
2. Over riding

Eg:- Stack container.

1. add some extra water → overflow/water

full of water

Overload

Pulled down because it already filled with full of water.

Eg:- 2. in class full students present. there is no place for other students. So, no extra student will not enter inside

2. Over-ride:

no water — if we expect some water we won't get.

Eg2: if no one in class, if we go near the class and call to someone no one will come out. because empty class.

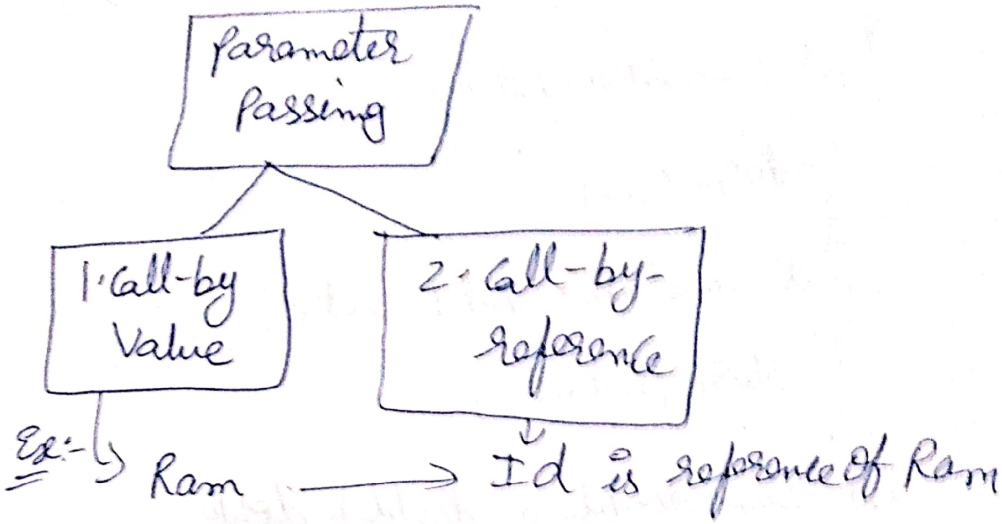
Overloading: is the concept of defining multiple methods with same name but with different signature.

Eg:- group of students with same name and same initial But Id is different. with the help of Id we can assign right marks to right student.

→ overloading ex: Same name but different signature.

```
class Test
{
 int Sum(int a, int b)
 {
  return (a+b);
 }
 int Sum(int a, int b, int c)
 {
  return (a+b+c);
 }
int Sum(double a, double b, int c)
{
 return (a+b+c);
}
 int Sum(int a, double b)
 {
  return (a+b);
 }
 PSVM ()
 {
  Test t=new Test();
  S.o.pln(" Sum is "+t.Sum(10,20));
   "   (   "        "   (5,2,3));
   "        "           (1,2.2));
   "        "           (1.2,1.3,5));
}}
```

Parameter Passing :- In general there are two ways that a computer language can pass an argument to a subroutine.



⇒ the first way is call-by-value. in this method copies the value of an argument into the formal parameter of sub routine. Changes made to the parameter of sub routine have no effect on argument.

2. Call-by reference :- a reference to an argument (not the value of the argument) is passed to the parameter. Changes made to parameter will effect the argument.

⇒ Java uses both approaches, depending upon what is passed.

⇒ when you pass a simple type to a method it is passed by value; what occurs to parameter that receives the argument has no effect outside the method.

ex:-

```
class Test {
void meth(int i, int j) {
  i *= 2;
  j /= 2;
}}
class CallByValue
{
  PSVM()
  {
    Test ob = new Test();
    int a=15, b=20;
    S.O.Pln("a and b before call: "+a+" "+b);
    ob.meth(a, b);
    S.O.Pln("a and b after call: "+a+" "+b);
}}
```

o/p:-

a and b before call: 15 20

a and b after call: 15 20

→ the operation occur inside meth() have no effect on values a and b used in the call. Values not change to 30 and 10.

→ when we pass an object to method, the situation changes dramatically, because objects are passed by reference.

Eg:-

```
class Test {
  int a, b;
  Test(int i, int j) {
    a = i;
    b = j;
  }
  // Pass an object
  void meth(Test o)
  {
    o.a *= 2;
    o.b /= 2;
  }
}
class CallByRef
{
  PSVm()
  {
    Test ob = new Test(15, 20);
    SOPln("ob.a and ob.b before call: " + ob.a + " " + ob.b);
    ob.meth(ob);
    SOPln("ob.a and ob.b after call: " + ob.a + " " + ob.b);
  }
}
```

O/P:- ob.a and ob.b before call: 15 20
ob.a and ob.b after call: 30 10

→ the actions inside meth() have affected the object used as an argument.

Eg 2:-

```
A()  → caller method
{
  B()  → called method
}
B()
{
}
```

Static fields and Methods :- Java's static keyword becomes important when we want to define a class member that will ~~depe~~ used independently at class level. when a class member is declared static it can be accessed without creating any objects of its class. both member methods and fields (variables) can be declared static.

Ex :- static member is Java's main() method.
the main() method is declared as static because it must be called before any object exist.

Java static Data members or Fields :- Data members or fields of a Java class declared static are called class members. there exists only one copy of static fields or class members no matter how many objects of the class are finally created. A static field also called as class variable comes into existence when the Java class is initialized. Data members declared as static are essentially global variables. when objects of its class are created they share the same copy of static field. ⇒

Ex:-

```
class staticField {
static int objectCount=0;
Public staticField(){
objectCount++;
}
Public String toString(){
return new String("there are "+objectCount+"object of class "+
                  this.getClass().getName());
}
Public static String numOfObjects(){
return new String("there are "+objectCount+"object of class "+
                  staticField.class.getName());
}}
Public class staticFieldDemo{
PSVm()
{
sopln("staticField.numofObjects());
staticField s1=new staticField();
//  s2=  //  ();
//  s3=  //  ();
sopln(s1);
// (s2);
// (s3);
}}
```

O/P: there are 0 objects of class staticField
// 3 objects
// 3 //
// 3 //

Java static methods :- used to implement class behaviors that are not affected by the state of any instance. they are private and cannot be accessed Publicly. Restrictions :-

1. they can only call static methods within from them.
2. Can only access data members or fields.
3. Cannot refer this and super.

⇒ Java static fields are called class Variables likewise static methods are called class methods. to initialize static variables, we usually declare a static block which gets executed exactly once, when the class is first loaded. it is always invoked without reference to a particular object. there will be a compile-time error if you try to invoke static method using the keyword this or super.

⇒ it cannot access instance variables directly.

Ex:-

```
class staticMethod {
int instVar = 10;
static int staticVar = 20;
Public static void staticMethod() {
System.out.println(instVar); //error
S.O.Pln(staticVar);
```

```
static Method smObject = new staticMethod();
    S.O.println(smObject.instVar);
}}
Public class staticMethodDemo{
    PSVm(){
    staticMethod.staticMethod();
}}
```

Recursion :- it is a process in which a function calls itself directly or indirectly.

Java supports recursion. Recursion is the process of defining something in terms of itself.

recursion is the attribute that allows a method to call itself.

⇒ recursion is the computation of the factorial of a number.

⇒ the factorial of a number N is the product of all whole numbers between 1 and N.

Ex:- 3 factorial is 1X2X3, or 6.

⇒ a factorial is can be computed by use of a recursive method:

Or:-

```
class Factorial {
        // this is a recursive function
    int fact(int n){
     int result;
     if(n==1) return 1;
     result = fact(n-1) * n;
    return result;
}
class Recursion{
   PSVm(){
   Factorial f=new Factorial();
  S.o.println("Factorial of 3 is "+f.fact(3));
      "        "  4 is "+f.fact(4));
      "        "  5 is "+f.fact(5));
}
```

O/P:- Factorial of 3 is 6
" 4 is 24
" 5 is 120.

Exploring string class :- string is a collection of similar type of characters.

Building strings :- string is a sequence of characters.

→ many languages implements strings as character arrays. Java Implements strings as object of type string.

→ when you create a string object, you are creating a string that cannot be changed. the characters that comprise that string. once string object is created.

→ each time you need an altered version of an existing string, a new string object is created that contains the modifications. the original string is left unchanged.

→ string & string Buffer classes are defined in java.lang. they are available to all programs automatically. Both are declared final, which means that neither of those classes may be sub-classed.

Exploring string class:- String class supports several constructors. To create an empty String, you call the default constructor.

Eg:- String s = new String();

⇒ to create a string initialized by an array of characters, use the constructor:

String(char chars[])

ex:-

```
char chars[] = {'a', 'b', 'c'};
String s = new String(chars);
```

⇒ this constructor initializes s with the string "abc".

Sub range of a character array:

String(char chars[], int startIndex, int numChars)

⇒ startIndex specifies the index at which the sub-range begins & numChars specifies number of characters to use

ex:-

```
char chars[] = {'a', 'b', 'c', 'd', 'e', 'f'};
String s = new String(chars, 2, 3);
```

⇒ String :- same character as another string object

String (String strObj)

⇒ strObj is a string object

Str Length :- it is the number of characters it contains.
To obtain this value, call the length() method.

Eg :- int length()

```
char chars[] = {'a', 'b', 'c'};
String s = new String(chars);
SOPln(s.length());
```

O/P 3.

CharAt :- it extract a single character from a string, you can refer directly to an individual character via charAt() method.

General form :- char charAt(int where)

getChars() :- more than 1 character at a time, you can use getChars() method.

Eg :- Void getChars(int sourceStart, int sourceEnd, char target[], int targetStart)

equals():- to compare two strings for equality use equals().

eg:- sy:- boolean equals(object str)

Here, str is the string object compared with the invoking string object.

equalsIgnoreCase(): it compares two strings, A-Z to be same as a-z.

sy:- boolean equalsIgnoreCase(String str)

compareTo():- to know 2 strings are identical. for sorting application, you need to know which is less than, equal to, or greater than the next. A string is less than another if it comes before the other in dictionary order. A string is greater than other it comes after the other in dictionary order.

sy:- int compareTo(String str)

indexOf():- two methods that allow you to search a string for a specified character of sub-string.

⇒ indexOf(): searches for first occurrence of character

2. lastIndexOf():- Searches last occurrence of character.

1. Syn:- String substring(int startIndex)

2. Syn:- String substring(int startIndex, int endIndex)

replace():- it replaces all occurrences of one character with another.

Syn:- String replace(char original, char replacement)

StringBuffer constructors:- it defines 3 constructors.

1. StringBuffer()
2. StringBuffer(int size)
3. StringBuffer(String str)

→ the default constructor (the one with no parameters) reserves room for 16 characters without reallocation.

→ the 2nd version accepts a String argument that sets the initial contents of StringBuffer object & reserves room for 16 more characters without reallocation.

StringBuffer allocates room for 16 additional characters when no specific buffer length is requested, because reallocation is a costly process in terms of time.

By allocating room for a few extra characters, StringBuffer reduces the number of reallocations that take place.

length() and capacity() :- current length of a StringBuffer can be found via length(), while total capacity can be found through capacity().

Syntax :- int length()
int capacity()

ensureCapacity :- if you want to pre allocate room for a certain number of characters after a StringBuffer has been constructed, you can use ensureCapacity() to set the size of the buffer.

Syn :- Void ensureCapacity(int capacity)

append() :- it concatenates the string with other. it has overloaded versions for all the built-in types and for object.

Syntax:- StringBuffer append(String str)
StringBuffer append(int num)
StringBuffer append(Object obj)

String.valueOf() :- is called for each parameter to obtain its string representation.

insert() :- inserts one string into another

Sy:- StringBuffer insert(int index, String str)
StringBuffer insert(int index, char ch)
StringBuffer insert(int index, Object obj)

reverse() :- You can reverse the characters within a StringBuffer using reverse().

Sy:- StringBuffer reverse()

delete() and deleteCharAt() :-

Syntax:-

StringBuffer delete(int startIndex, int endIndex)
StringBuffer deleteCharAt(int loc)

replace():-

Sy:- StringBuffer replace(int startIndex, int endIndex, String str)

endIndex-1 is replaced.

* UNIT-I completed :: * ::

* Extra or Missing topics :- *

declaring string :- a) string s1;
s1 = "hello";
b) string s1 = "hello";

methods of string class:-

1. concat():- used to concatinate 2 strings.

Eg:- string s1 = "hello";
string s2 = "world";
string s3 = s1.concat(s2);

o/p:- hello world.

2. length():- string s1 = "hello"; → 5
int len = s1.length();

o/p:- 5

3. charAt():-

ex:- String s = "hello";
char ch = s.charAt(1);
o/p:- e

4. strcmp():- compares 2 strings
if s1 > s2 +ve
if s1 < s2 -ve

Eg:- s1 = Hi
s2 = Hi ⇒ it returns 0 value because both values are equal.

→ Applets:- to develop web supportive applications (System Browser)

→ Servlet & JSP:- to develop web applications (internet Server System)

⇒ Bytecode:- Bytecode is an intermediate format that can only be understandable by JVM.

→ JVM:- Java Virtual machine - is a software that executes Java byte codes by converting byte codes into machine language of the current operating system's understandable format.

Two file extensions :-

1. ".java" extension file it called source code which is developed by developer.
2. ".class" extension file it is called compiled code contains bytecode which is generated by compiler from source code.

⇒ Javac :- it is a compiler which is used to check the errors of source code (we written code).

to compile :- `javac space filename.java`

checks errors.

to Execute :- `java space filename`

gives output.

⇒ compilers convert sourcode into bytecode.

⇒ JVM executes these bytecodes. byte codes convert into ML (Machine Language) of current OS.

→ developer :- is responsible of developing, compiling & executing Java application. in developer system we must install both compiler & JVM.

→ client :- executes only Java applications. client computer just installing JVM is sufficient

→ Binary files :- are command files by default stored in a folder called "bin". they have commands to be executed in sequence. Java binary files have command to compile and execute Java program.

→ Library files :- are program files by default stored in a folder called "lib". they have logic, used to develop another applications.
→ Java library files used in developing new Java applications and applets.

⇒ Path :- Used by OS to identify binary files. Path is a mediator between developer and (compiler, JVM) to inform softwares binary files path.

⇒ classpath :- used by compiler and JVM to identify Java library files. classpath is a mediator between developer and (compiler, JVM) to inform the library files path those are used in our source code.

⇒ Path environment :- variable is inbuilt available in windows OS. we must update it with our software path value in windows