Unit-2

JAVA

Ques:- Why Java does not have pointer?
Since pointer allows the programmer/ user to play with the memory. It causes harm to the system. To avoid this, JAVA does not allow, anyone to play with the memory, java does not entertain eavesdropper to break the security.

Ques:- What is the diff. b/w function & method?
Func. in 'C' language is used as a procedural form but method in OOPs based language (JAVA, C++, Sea shark) is used in bottom-to-top based language.

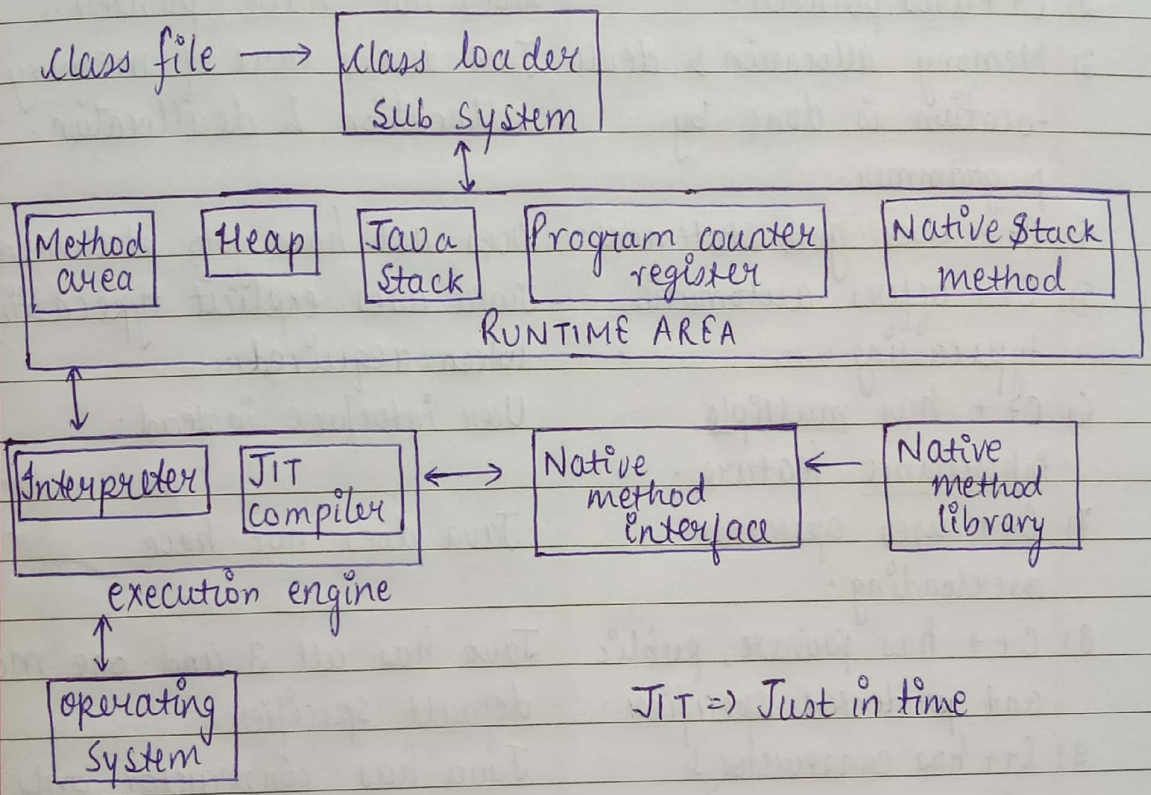
Ques:- What is JDK and JRE?
JDK stands for Java Development Kit which is used to develop Java programs whereas JRE stands for Java Runtime Environment which gives an environment / platform to run the JAVA program.

Ques:- What is the role of JVM in Java?
JVM stands for Java virtual machine. Main role of JVM is to run the object code of Java (byte code) → A compiled version of Java source file. (abc.java)

(abc.class)
↳ byte code

JVM manages the deallocation of memory when variables or objects are not in use. Garbage collector uses the algorithm "mark and sweep" to manage the memory.

Component of JVM Architecture



JIT => Just in time

Ques:- How interpreter & JIT compiler work together?
Interpreter & compiler are responsible for converting the byte code into machine code (native code). Just in time compiler is a part of JVM which increases the speed of repetitive execution in the program.

Ques:- Difference b/w C++ and Java.

C++	Java
1) Not a pure oops based language because C++ does not require to create object always.	Java needs atleast one class and then object to run the program.
2) C++ uses pointer.	Does not have pointer.
3) Memory allocation & deall-ocation is done by programmer.	JVM takes care of memory allocation & deallocation.
4) C++ uses goto statement.	Does not have goto statement.
5) C++ offers automatic typecasting.	Java uses explicit typecasting when required.
6) C++ has multiple inheritance feature.	User interface instead.
7) C++ uses operator overloading.	Java does not have.
8) C++ has private, public and protected specifier.	Java has all 3 and one more default specifier.
9) C++ has constructor & destructor.	Java has constructor only.

Ques:- Diff b/w #include and import.

In any programming language, there are default library methods or functions which helps us to execute the program without any hurdle.

In C language, #include brings the header files to our current program to execute it.

Similarly in Java, we have import command to call all libraries and their methods to execute the program.

Default keywords or reserved keywords: Those keywords which are already defined in Java library.
Eg:- int, double, scanner, println, public, static, void, for, while.

Importing class in Java

Java library
↓
package
↓
class & interface
↓
Methods

eg:- import java.lang.system;
import java.io.*;
import java.util.*;

where, import = An instruction
java = A library
lang, io, util } = package
system = class

Java uses package which is kind of directory that contains a group of related classes and interfaces.

H.W.

Ques:- When Java imports the library files, will the size of program increase? Justify it.

Ques:- While including the files in C/C++, why the size of program increases? [we copy the files]

Program (Hello.java)

```
class Hello
{
public ← public static void main(String [] args)
                                  ↳ string array as parameter
  {
  System.out.println("Hello");
  }
}
```

↑ does not require any object to invoke

publicly available to JVM

System → class, out → static variable, println → method, "Hello" → printable statement.

out / in / err

JVM class the main() by invoking Hello.main() because it is a static method.

Mathop.java

```
class Mathop
{
  psvm (String [] args)
  {
    int x = 5, y = 25;
    int z = x + y;
1. → Sopln ("Add=" + z); → 30
```

2 → Sopln("SUM =" + (x+y)); → 30
3 → Sopln("SUM =" + x+y); → 525
4 → sopln("Total =" + x+y+2); → 52530
5 → Sop("SUM ="); → SUM = 30
6 → Sop(x+y);

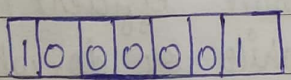
Naming Convention =>

Class name always starts with capital letter
Eg: 'Hello', Mathop, etc.
Methods name follows camelCase style. Eg:- indexOf()
charAt(), lastIndexOf(), valueOf(), sqrt().
Constant variable can be represented by all caps.
Eg:- PIE, MAX, MIN
All reserved keywords are in small letters.
Eg. public, static, void, int.

Data type =>

	size
int	4byte
char	2byte
float	4byte
double	8byte
byte	1byte
String	A class
long	8byte
short	2byte
boolean	1bit

ASCII → 8-bit



A = 65

ASCII is a part of UNICODE.
↳ ☺ â

UNICODE is a universal representation of any language used across the globe. Eg:- symbolic language, chinese / japanese script, latin language, etc.

Ranges from 0 to 65536

Operator => 1) Arithmetic operator - +, -, *, /, %
2) Logical operator - &&, ||, !
3) Relational operator - &, |, <, >, <=, >=, ==, !=
4) Bitwise - &, |, ^, ~, <<, >>
5) Unary - ++, --
6) Shorthand - +=, -=, *=, /=
7) Assignment - '='
8) Ternary :- (conditional) exp1 ? exp2 : exp3
Relational TRUE FALSE

Conditional =>
1) if 2) if-else 3) Nested if-else
4) Ladder if 5) Ladder else 6) Switch
7) Nested switch

Control st =>
1) for 2) do-while 3) while

Q1. If-else or switch, which is more efficient?
Q2. Which is more complex, while or do-while?
Q3. Which will perform faster, for or while?
Q4. At what condition, we must use do-while loop only?

Q5. Display 10 nos. randomly from range of 15 - 25

(int)(10 * Math.random() % 25) + 15

Q6.
```
+
+ +
+ + +
+ + + +
+ + + + +
+ + + + + +
```

Q7. Table Output

2	3	4	5
4	6	8	10
6	9	12	15
8	12	16	20
10	15	20	25

Q8. What is the actual diff. in writing exit() & break func. in switch?

Q9. Diff. in
System.exit(0); → termination
System.exit(1); → signifies error.
System.exit(-1);

Q1. Switch is more efficient as it tests expressions based only on a single integer, enumerated value or String object.

Q2. do-while

Q3. for will be better as i will be out of scope after the loop, while the i will stick around in 'while' loop case.

Q4. In condition where we want our loop to execute atleast once.

Ques:- Find the distance b/w 2 points (20,20) and (40,30)

2) Estimate the time needed for a stone to fall from the height of 3000 mts. Assume $g = 9.8 m/s^2$

3) Given the temperature, print whether water will be in solid, liquid or vapour form.

4) Find if a given pt. lies in a given circle or not.

5) Ram pays Shyam Rs. 10000/- everyday. Shyam pays Ram Rs 1/- on first day, Rs. 2/- on second Rs 4/- on third, Rs 8/- on third ----. On which day the total amount received by Ram is greater than the total amount he has paid. [18th day]

Java Program Skeleton

```
class MyEx                    class.
{   static int z;        (instance or
instance ← int x, y;          class variable)
    void display()
    {
      sop("Display the screen");
    }
     psvm(string args[])
     {
        MyEx obj = new MyEx();
          obj.display();
     }
}
```

The process of creating object using class name is called instantiation.
Class variable are those variables which are not instance variable & should have prefixed with static.

```
static int z;
```

Similarly instance methods & class methods are :
those methods which are invoked by object.
Class methods are those which are prefixed with static and does not require any object to be invoked. Eg :- object.method() → instance
classname.method() → class method.

```
obj.display();        Eg :- MyEx.name();
```

func. vs. Method

1)

```
Fnc
void main()
{
  display();
  pf("I am in main");
}
void display()
{
  pf("I am in display");
}
```

2) In Java

```
class f1
{
  static void display()
  {
    sop("I am in display");
  }
  psvm(String[] args)
  { sop("I am in main");
    display();
  }
```

3) In Java

```
class f2
{ void display()
  {
    sop("I am in display");
  }
    psvm(——)
    {
      f2 obj = new f2();
      obj.display();
    }
```

→ In ② we have used a static keyword. Static is a very powerful tool in Java which does not need any object to invoke any method in Java.

Ques:- WAP to accept some values like height, width and length using a method input(), calculate the volume of a solid box along with the surface area using another method compute(); finally display the result using a method display().

[Box.java]

```
public class Box
{
    void input double length, width, height;
        void input( double l, double w, double h)
        {
            length = l;
            width = w;
            height = h;
        }
    void compute( )
    {
        double volume, surface;
            volume = length * width * height;
            surface = 2 * ((length * width) + (width * height)
                        + ( length * height));

    }
    void display ( )
      {
        Sopln ( "volume=" + volume);
        Sopln ("surface area =" + surface);
      }
    psvm (String [ ] args)
{   Box obj = new Box( );
    obj. input ( 10.0, 5.0, 3.0);
     obj. compute ( );    obj. display ( );
}}
```

The above program is logically correct but syntactically wrong because in compute() section the 'vol' & 'surface' variables are defined only for that method. we cannot use them in another method viz. display() until it is declared with public access specifier.

Ques:- WAP to take the input as radius in order
a) to find area of a circle if radius is being passed by main() through method invokation using parameter.
b) Introduce the concept of method overloading.

Method overloading is a concept where no. of methods will be there with the same name different no. and types of parameters.

```
public class circle
 {
    void radius()
    { Sop("No parameter"); }
      void radius(int n)
      {
        Sop("one parameter = "+n);
      }
      void radius(int n, int m)
       { Sop("two parameter = "+(m*n));
         }
```

```
void radius( double x)
{ sop ("double type = "+x); }

void radius( double x, float y, int z)
{ sop(" 3 parameters = " + (x+y+z));
  }
psvm (String[] args)
{
  Circle obj = new Circle();
       Obj.radius();
       Obj.radius(1);
       obj.radius(2,3);
       Obj.radius(2.5);
       obj.radius(2.0, 4.2f, 3);
}}
```

Scanner class

When a scanner class receives the input from the keyboard, it breaks the input into several pieces, called **tokens**. These tokens can be retreived from the scanner object using fol. methods :-

1) next() → To read a string
2) nextLine() → To read a string (A Line)
3) nextInt() → " " an Integer
4) nextDouble() → " " a double
5) nextFloat() → " " " float
6) nextLong() → " " " long
7) nextByte() → " " " byte
8) next_____() " " " character

Prog ⇒

```
import java.util.Scanner;
import java.util.*;
class Ex
{
  psvm(--)
  {
    Scanner sc = new Scanner(System.in);
     Sop("Enter id, name, salary");
     int id = sc.nextInt();
     String name = sc.next();
     float sal = sc.nextFloat();
    Sop("The value entered are:");
    Sop("id=" + id + "name=" + name);
  }
}
```

Ques:- WAP to accept length & width of a rectangle using 2 methods:
(1) by using getData()
(2) by using direct assignment & finally find the area of rectangle for 2 different values.

```
class Rect
{
   int l, w;
   void getData(int x, int y)
   {    l = x, w = y;
        }
     int rectArea()
     {
        return(l*w);
     }
```

```
psvm (String[] args)
{
  int a1, a2;
  Rect obj1 = new Rect();
  Rect obj2 = new Rect();
   obj1.l = 15;
   obj1.w = 10;
    a1 = (obj1.l) * (obj1.w);
  sop("Area1 = "+a1);

  obj2.getData(12,20);
   a2 = obj2.rectArea();
   sop("Area2 = "+a2);
}
```

Ques:- WAP to compare the area of room when the size of the room is already mentioned but customer wants some specific size of the room. Result should display whether the room size is smaller or larger.

Constructor

The meaning of constructor is to initialize the value to the instance variable. It can be declared inside the class with the same name as class has in the format of function declaration. It does not return anything, any type, any value, not even void.

1) →
```
class Rectangle
{ int l, w;
  Rectangle(int x, int y) → Constructor.
  { l = x;
    w = y;
  }
  int rectArea()
  { return l*w;
  }
  psvm(String args[])
{
                                → parameterized.
  Rectangle obj1 = new Rectangle (15, 10);
  Sop("Area =" + obj1.rectArea());
}}
```

2)
```
class Rectangle
{
  int l, w;
  Rectangle (int x, int y)
  {
    l = x, w = y;
  }
```

```
int rectArea()
{
  return l*w;
}

class RectangleArea                    [file name will be
{                                       RectangleArea.java as
  psvm(String[] args)                   it contains main()]
  {
    Rectangle r1 = new Rectangle(12,20);
      int area = r1.rectArea();
      sop("Area=" + area);
  }
}
```

Ques:- When is a constructor called?

a] before creating the object

b) after " " "

c) during creation of "

d) object is not required.

Ques:- Differentiate b/w constructor & methods [5].

Ques:- Can we overload constructor?

Ques:- WAP to create an object and invoke the fou. methods:

1) parameterized constructor to take input or ~~initiali~~ initialize the instance variables.

2) find the max and avg

3) display the name, roll, age, m1, m2, m3 (for 3 subject marks), max, avg.

data member:

1) Name 2) Roll, age, 3) m_1, m_2, m_3 4) max, avg.

```
class mark
{
    int roll, age, m1, m2, m3, max;
    double avg; String name;
    mark(i
```

Ques:- Define

Date

bike_no

Member

Ques:- W

acce

If p

a

→ F

i

→ U

Ques:

Ques:- Define a class bike_rent with the foll. description

Data member ⇒

bike_no, mobile, name, govt_ID, days, charge

Member method ⇒

input()

compute(), display()

Charges

5 days or less ⇒ 500/days

next 5 days ⇒ 400/day

Rest day ⇒ 200/day

Ques:- WAP to check the status of age of a person by accepting the value of name & age using constructor.

If person <= 30, display it as 'young'

age <= 500, middle age.

age >= 50, old age

→ Further extend this question by using method for input, not constructor.

→ Use 2 types of constructor - default & parameterized to initialize name & age.

Ques:- WAP to take name & age from command line argument and display the same condition as mentioned above.

Command Line argument

class ex

{

```
psvm (String[] args)
{
  int a = Integer.parseInt(args[0]);
    String name = args[1];
  sop ("Name = " + name + "Age = " + a);
}
```

How to run

```
C:\ javac    ex.java
C:\ java     ex  ←┘   ERRR
C:\ java     ex 20    Raja ↵
                ↓       ↓
             args[0]  args[1]
```

O/P: Name = Raja
Age = 20

Array
having
availab
packet
Drawbo
of array in

Advantag
array

Declar
of
(or de

de

Ques:-h

Array means group of elements with similar type having limited no. of elements. Eg:- No. of students (/members) available in the classroom, No. of pens in a packet.

Drawback of array in C - 1) Limited no. of elements.
2) Garbage value if elements are not filled up

Advantage of array in JAVA - 1) Java takes care of array size when elements are not filled in.

⇒

Declaration of array (or definition) - Declaration means write the variable name with data type.
Eg. int a;

Definition means assign a value to the declared variable.
Eg:- a = 100;

int a[ ]; or int [ ] a;

1) int [ ] a = new int [5];
2) int [ ] a = new int [ ]{ 5, 6, 8, 15, 70};
3) String [ ] str = new String[Math.max(5, 15)];

a[ ] =>

-11	20	100	77	19
0	1	2	3	4

print / display all the array elements.

Ques:- WAP to accept no. of elements in an array and print all those elements with the same order.

```
for (i=0; i< a.length; i++)
  {
    a[i] = sc.nextInt();
  }
for (i=0; i< a.length; i++)
  {
    sop(a[i]);
  }
```

Ques:- Input your friends & make a plan to visit Mussoorie with some committment. The program should display the conversation among friends.

Ques:- WAP to make 2 diff. arrays with some limited no of arrays & create one more array with no element. The program should display all the elements of each array and the last array might contain / merge of 2 arrays element.

Ques:- String strArr[] = new String[]{"Autumn", "Summer", "Spring", "winter"};

1) for (int i=0; i< strArr.length; i++)
   sop(strArr[i]);
   String strArr[2] = NULL;

2) for (string val : strArr)
   sop(val);

↓ any variable name matching data type with the purposes for.

o/p => 2
loop, know
loop takes
pass thro

vat
val
val
val

Array

Ex:- clas
{

o/p => 2] → This is an extended version of 'for' loop, known as foreach loop. It means foreach loop takes the very first element in an array and pass through each element till the end of the array

val[0] = Autumn.
val[1] = Summer
val[2] = Spring
val[3] = Winter.

Array & Array-List =>

Ex:-
```
class Arrex
{ psvm(String[] x)
  {
    int[] n = {15, 16, 17, 40, 50};
    int[] num = new int[]{5, 6, 8, 7, 9};
    int p[];
    p = new int[5];
    for(int i=0; i<5; i++)
      p[i] = i*2;
      for (int i: n)
        Sop(i+" ");

    for(int k: num)
      Sop(k+" ");

    for(int d: p)
      Sop(d+" ");
```

for each loop

```
for (int i = 0 ; i < 5 ; i++)
 {
    sop(n[i]);
    sop(p[i]);
    sop(num[i]);
 }
```

2)

Array-list => Is a dynamic array which can store a sequence of values (or object) whose size can change dynamically. Array-list can have homogenous elements (say values), or heterogenous elements (say object).
An arraylist can grow & sink as needed.
Array list class supplies methods for many common tasks such as inserting, removing element.
Elements can be added (or deleted) from any position (front, back or elsewhere).

eg1 :-

```
class ArrList
 { psvm(--)
   {
     ArrayList<Float> al = new ArrayList<>();
        al.add(3.2);
        al.add(4.2);
        al.add(5.75);
     for (float f : al)
        sop(f);
```

2) class ArrList
{ psvm(—) → wrapper class.
{ ArrayList<String> al = new ArrayList<>();
al.add("INDIA");
al.add("China");
al.add("Nepal");
al.add(0, "Korea");
Sop(al.get(1)); //INDIA
Sop(al.remove(0));
Sop(al.contain("Korea")); → // False
(Return true / False)
al.clear();
for(String S: al)
Sop(S);
}}

Methods -

1) boolean add(value)
boolean add(index, value)
2) void clear()
3) int indexOf(value)
4) Object get(index)
5) Object remove(index)
6) Object set(index, value)
7) int size()
8) boolean addAll(list)
boolean addAll(index, list)
9) boolean contains(value)
10) boolean equals(list)
11) int lastIndexOf(value)
12) boolean remove(value)

2) class ArrList

```
{
    psv m (—)        // wrapper class.
    {
        ArrayList <String> al = new ArrayList <x>;
            al.add("INDIA");
            al.add("China");
            al.add("Nepal");
            al.add(0, "Korea");
            sop(al.get(1));        //INDIA
        sop(al.remove(0));
        sop(al.contain("Korea"));  // (Return true / False) → // false
            al.clear();
        for (String S: al)
            sop(S);
    }
}
```

Methods -

1) boolean add(value)
   boolean add(index, value)
2) void clear()
3) int indexOf(value)
4) Object get(index)
5) Object remove(index)
6) Object set(index, value)
7) int size()
8) boolean addAll(list)
   boolean addAll(index, list)
9) boolean contains(value)
10) boolean equals(list)
11) int lastIndexOf(value)
12) boolean remove(value)

boolean removeAll (list)
13) boolean retainAll (list)
14) sublist (from, to)
15) boolean isEmpty ()
16) void trimToSize ()

String

Mutable String (can be changed)
eg - StringBuffer

Immutable String (cannot be changed)
eg - String

```
String S1 = "Hello";
S1 = "Hello! DITU";
S1 = "Hello" + "DITU";
```

String is a collection of characters which is a sequence of characters.
String is a class in Java as well as a variable.
String can be defined in 2 ways -

Variable {
```
String S1 = "Hello";
String S2 = "Hello";
```

object {
```
String S3 = new String ("Hello");
String S4 = new String ("Hello");
```

Methods

1) charAt
2) compar
3) concat
4) endsW
5) equal
6) index
7) last
8) rep

eg:- pu

variable {

object

1)

2

classmate
Date
Page

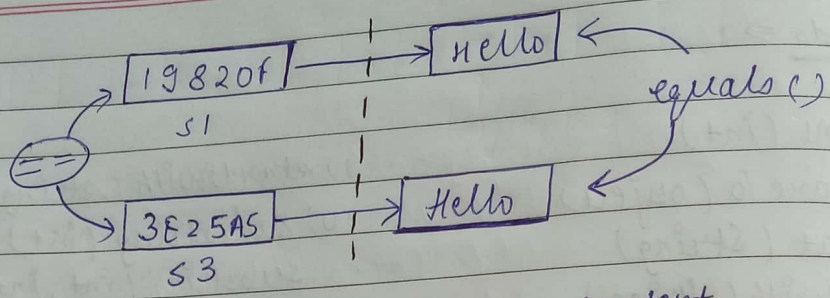
Methods =>

1) charAt (int)
2) CompareTo (object)
3) concat (String)
4) endsWith (String)
5) equals (object)
6) indexOf (char)
7) lastIndexOf (char)
8) replace (char)
9) startsWith (string)
10) substring (int)
    substring (int, int)
11) valueOf ()
12) toLowerCase(), to UpperCase()
13) trim()
14) String[ ] split (delimin-iter)

eg:- public class strImp {
    psvm (—) {
variable { String s1 = "Hello DIT";
           String s2 = "Hello";

object { String s3 = new String ("Hello");
         String s4 = new String ("Hello");

1) if (s1 == s3) sop ("same"); → Ans.
   else sop ("Not same");
2) if (s1.equals (s3)) sop ("same");
   else sop ("not same");
3) if (s1 == s2) sop ("same");
   else sop ("not same");
4) if (s3.equals (s4))
   sop ("same");
   else
   sop ("Not same");
}



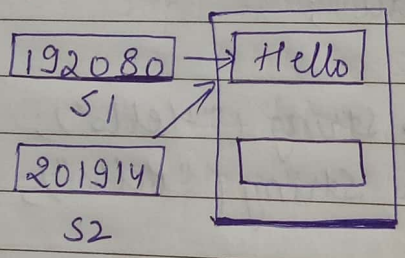
Ref. var. comparison.

content comparision

5) if (s4 == s3) sop("same");
else sop("Not same");

6) if (s1.equals(s2)) sop("same");
else sop("Not same");

String Constant Pool



String s1 = "Hello";
String s2 = "Hello";

Checks the value of variable in string constant-pool, if same value is assigned.

String Bu
modified
updation
do not
assignn

eg:-

strin

A Stri
It
may
pro
→ Lon
le

String Buffer => It is a mutable string that can be modified at any point of time. In the case of updation of new value to the String Buffer, we do not need to create any new object for the assignment of updated value.

eg:- String s = "Hello";
s2 = s + "Hi"; } => s is updated

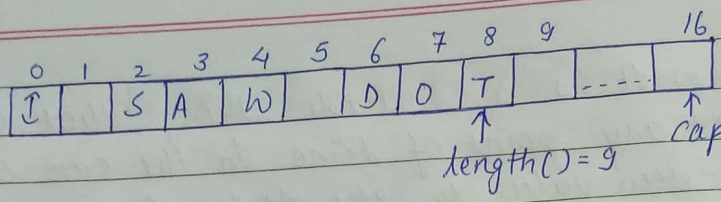
StringBuffer sb = new StringBuffer("Hello");
sb.append("Hi"); => sb is

A StringBuffer class is a part of java.lang package. It contains a memory block called buffer which may not contain a string. A string buffer object provides more flexibility in terms of memory.
→ Length of the string may not be the same as the length of the Buffer (refer to capacity()).

Initialisation

StringBuffer sb = new StringBuffer(); } capacity is
" " = " " ("Hello"); } 16 default
" " = " " (30); // capacity now is 30

length() of a string returns total no. of characters in the string.
capacity() of a string returns the max. no. of character that can be inserted in the string.



ex:- class StrBuf
{
psvm(—)
{
String S = new Buffer("Greet for");
StringBuffer sb = new StringBuffer(S);
int p = S.length();
int q = sb.capacity();
Sop(sb.append("Greet"));
sb.insert(5, "Java"); → Greet Java for Greet
Sop(sb);
Sop(sb.reverse());
sb.delete(0,5);
sb.~~re~~delete(charAt(7));
sb.replace(8,13, "%IT");
}}

Ques:- Accept a string of a line and convert all the first letter of each word into uppercase & Last character of each word to uppercase.

Ques:- Convert each first & last letter of ~~a~~ each word into their next succeeding character.

Append() m
end of ano

1) append (s
StringBuff
Sb.app
Sop(

2) insert (int

StringBuff
Sb.
So

3) delete (i
character
end in
String

4) repl
rep

String

String Buffer Methods

Append() method is used to add the string to the end of another string.

1) append (str) →

```
StringBuffer sb = new StringBuffer("Hello");
sb.append("Java!");
Sop(sb);          # HelloJava!
```

2) insert (int offset, str) → inserting a string at a specified index position

```
StringBuffer sb = new StringBuffer("Welcome to the java");
sb.insert(15, "world of");
Sop(sb);
```

3) delete (int begIndex, int endIndex) → deletes the character from the specified index to another specified end index.

```
StringBuffer sb = new StringBuffer("Hello Java");
sb.delete(1, 4);
Sop(sb);          # Ho Java
```

4) replace (int begIndex, int endIndex, str)
replace the & character in a substring of the sequence.

```
StringBuffer sb = new StringBuffer("Hello");
sb.replace(1, 3, "Java");
Sop(sb);
          # HJavalo.
```

Reverse Method:

reverse()

It is used to reverse the characters of a string.

```
sb = new sb("Hello");
sb.reverse();
Sop(sb);        # olleH
```

→
```
main()
{
    String s = "Hello";
    for(int i = s.length()-1; i>=0; i--)
        Sop(s.charAt(i));
}
```

setCharAt(int) → replace a character with another character at a specified index.

```
StringBuffer sb = new sb("welcome");
sb.setCharAt(2,'l');
Sop(sb);        # welcome.
```

setLength(int) → Sets the length of a string buffer variable upto a specified range. It truncates, if the length of character exceeds the boundary limit range.

StringBuffer sb = new SB ("Welcome to Java");
sb.setLength(7);
Sop(sb);
# welcome.

capacity() - i) It returns the current capacity of the buffer.

ii) default capacity of buffer is 16.
iii) if the no. of character increases from its current capacity, it ↑ the capacity by
(old capacity * 2) + 2
eg:- (34 * 2) + 2 = 70
(16 * 2) + 2 = 34

StringBuffer sb = new StringBuffer("A"); → #17
16 # Sop ("default capacity = " + sb.capacity());
sb.append("Hello");
16 # Sop("Hello capacity = " + sb.capacity());
sb.append("Java is my favourite language");
34 # Sop("sb.capacity=" + sb.capacity());

Difference b/w String & String Buffer

	String	String Buffer.
1)	Read only and immutable string.	Mutable & can be modified.
2)	Not faster comparatively.	Faster in some cases (concatenation)
3)	Having fixed length.	Provision to change their length.
4)	A new object is created everytime.	Not req. to create a new object everytime, so

5) A general approach. Advanced approach.

changes can be reflected in the same object.

Static => It is a keyword in JAVA which can be categorised into 3 types -

1) Static block
2) " method
3) " variable

1) Static method => To call a static method you do not need to create any object for invocation. We can call a static method as follows -

Syntax- classname.methodname();

s_method.java ← File name.

```
class Sample
{
static double sum (double x, double y)
{
    return (x+y);
}
}
class s_method
{psvm ()
{ double ans = Sample.sum (10.0, 15.25);
    sop("SUM=" + ans);
}}
```

2) Static variable => A variable which can be prefixed with static variable for any kind of variable (instance variable). It is also known as class variable.

eg:-
```
class s_test
{
    static int x = 50;
    static void access()
    {
        sop("x = " + x); }}
class s_demo
{ psvm(--)
  { s_Test.access();
  }}
```

Static variable will always be used inside the static method.

Que: Diff. b/w static variable & instance variable.

1) For the class variable, a single copy of a static variable is shared b/w all the objects.
2) For instance variable, a separate copy is available to each object.
3) Instance variable is created in the object memory (heap memory).
4) Class variable is stored on method area.

3) Static block - It is a block of statement declared as a 'static'.
JVM executes a static block on a highest priority basis.

1) Class Test
{ static {
sop("I am in static block");
} psvm(-)
{
Sop(" static main() method");
}
}

2) → A program without static main()
Class Test2
{
static
{ sop(" only static block"); }
}

⇒ Compilation is okay
⇒ Execution is not possible, produces ERROR.

3) → without ~~main()~~ main(), also run
Class Test3
{ static {
sop("static block");
System.exit(0);
}}

⇒ Execution run in Java 6 & below.

'JVM execute Java program in foll. order'
1) Static block
2) Static main()
3) instance method

1) instance method - A method which acts upon the instance variable.
To call the instance method, an object is needed.
Instance method is categorised into 2 types -
→ getter method (Accessor method) ⇒ return or get the value
→ Setter method (mutator ") ⇒ Set the value

eg :-
```
class Person
{ String name;
  int age;
void setName(String n)
 { name = n;
 }

 void setAge(int a)
 { age = a;
 }

 String getName()
 { return name;
 }

 int getAge()
 { return age;
 }}
 class P_Demo
{
  Person P = new Person();
   p.setName("Raju");
    p.setAge(20);
  Sop("Your Name =" + p.getName());
  Sop("Your Age =" + p.getAge());
}}
```

instance variable → In instance variable each object gets a separate copy of instance variable

```
class Test_i
{
    int x = 10;
    void display()
    { Sop(x); }
}
class Demo_i
{
    psvm(--)
    {
        Test_i P1 = new Test_i();
        "      P2 = "    "  ();
        ++P1.x;
        Sop("x in P1"); p1.display();
        Sop("x in P2"); P2.display();
    }
}
```

class variable (or static variable)

```
class Test_c
{
    static int x = 10;
    static void display()
    { sop(x); }
}
class Demo_c
{
    psvm()
    { Test_c T1 = new Test_c();
      "      T2 = "    "  ();
      ++T1.x; ⇒ ++Test_c.x;
      Sop("x in T1"); T1.display();
      Sop("x in T2"); T2.display();
```

All the Objects share the same copy of static variable

Access specifier	Access Modifier
public	final, abstract
Private	static, strictfp
protected	volatile, transient
default	

Access Modifier: visibility of field.

	Public	Protected	Default	Private
same class	✓	✓	✓	✓
Subclass in same package	✓	✓	✓	X
other class in same package	✓	✓	✓	X
Subclass in other package	✓	✓	X	X
No-Subclass	✓	X	X	X

package JavaApplication

- prog1.java
- prog2.java
- <subclass>
  - prog3.java
  - prog4.java
- prog5.java
- ....

Introduction to OOPs

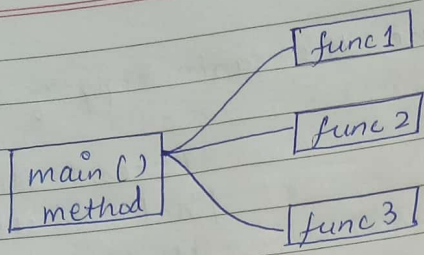


Fig 1: Procedural approach

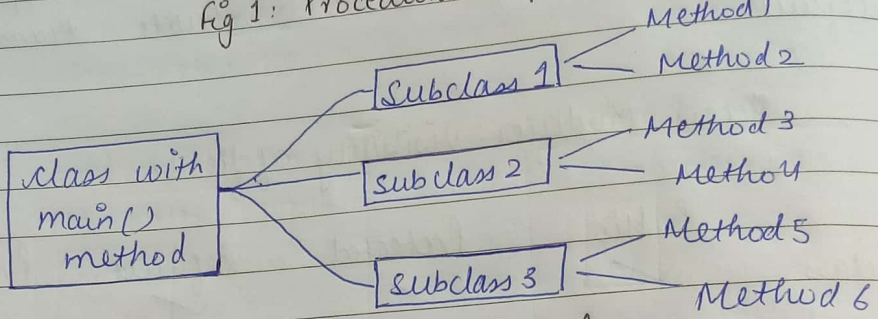


Fig 2: Object oriented approach.

Procedural Approach

A programmer uses procedures or functions (also known as modules) to perform a separate sub task. So a ~~C~~ program contains generally several funcs. which are invoked & controlled ~~for~~ ≠ from a main().

A class is a module which itself contains data & methods to achieve the task with the help of objects.

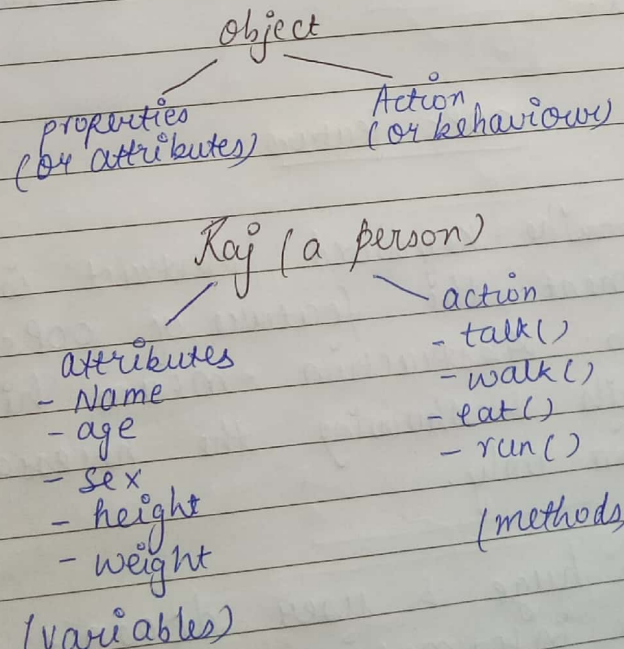
The main() task is divided into several classes or subclasses, further divided into several methods.

Drawback — we cannot use the code again & again. It violates the reusability technique.

OOPs features =>

1) class / object
2) Abstraction
3) Encapsulation
4) Polymorphism
5) Inheritance

Object → An object is anything that really exists physically & it has distinguished features from other. Eg :- table, car, ball, dog, person.
The objects are not as our imagination. Plan, our thoughts, etc.

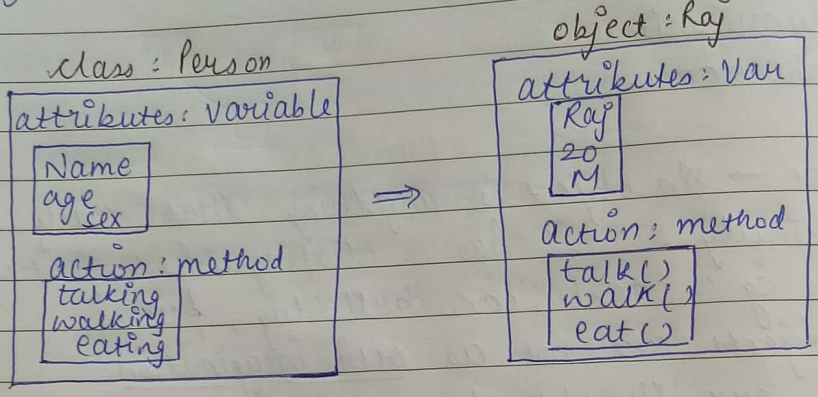


Some objects may have similar attributes & similar actions; such objects belong to same category called a 'class'.

eg. classname - person

objects - raj, ravi, sita, krishna

class person does not exist physically but objects - raj, ravi --- exist physically.



Class is a container which contains data & methods.

OOPs feature

1) Abstraction → The keyword 'abstract' is used to implement the feature of OOPs for abstraction. Abstraction means hiding the details & showing the necessary information only.

Data is huge & user doesn't need the entire information. User requires only some part of the available data.

(HAS - A)

2) Inheritance => It means to inherit the properties from base class.

Base class —inherit→ Derived class

Super class / Parent class — Sub class / child class

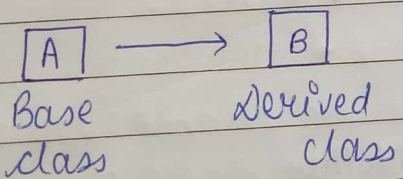
It creates a new class from existing class. New class will aquire all the features of existing class.

A —inherit→ B

```
class A
{
  ___
  ___
};
class B extends A
{
  ___
  ___
};
```

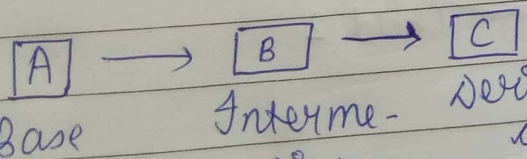
Types of Inheritance

1) Single Inheritance



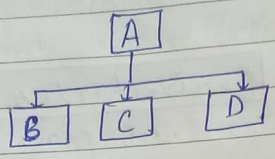
Base class — Derived class

2) Multi level

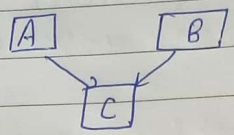


Base Class — Interme-diatory class — Derived class

3) Hierarchical Inheritance

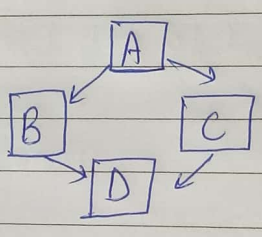


4) Multiple

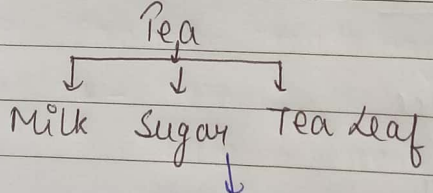


(Through interface)

5) Hybrid (Through interface)

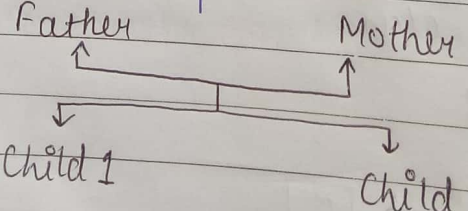


Q1.



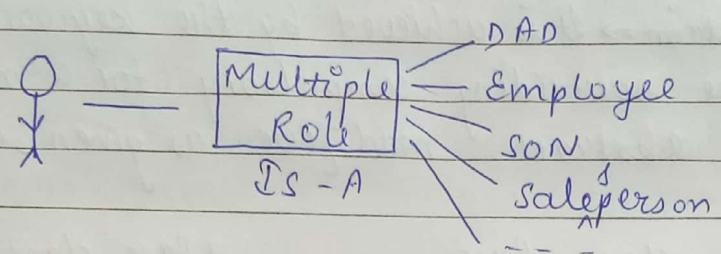
Multiple

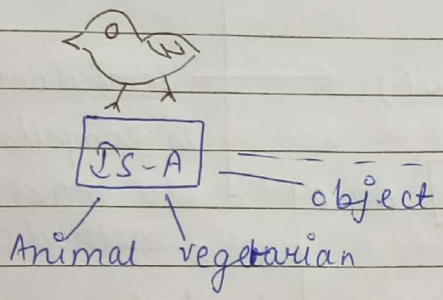
Q2.



(IS-A)

3) Polymorphism ⇒ Ability to perform several forms (tasks) by an entity. Polymorphism encourages 'extendibility' which means an object or a class can be extended.





4) Encapsulation ⇒ It is a single unit which binds data & methods ~~it a single~~ together.

Variables of a class will be hidden from other classes & can be accessed only through the methods of their current class (also known as data hiding).

Encapsulation	Data Hiding.
It is a goal	Data hiding is one way to achieve the encapsulation.

Data hiding is protecting the data from outside world. It hides the implementation & make the data private.

N.B. Data Hiding is not possible without Encapsulation. Encapsulation is possible without Data Hiding.

Abstraction => It is achieved by the keyword abstract. To make overriding compulsory for a method, we use abstract modifier as given below

```
abstract class Base
{
  abstract void draw();
}
class Derived extends Base
{
  //var...
  void draw()
  {
    //body
  }}
```

Base class

Semicolon(;) is compulsory to make a method abstract.

Derived class

It is a keyword to derive a class from the base class

We cannot create any object of abstract class. Eg -

```
Base b = new Base();
```

ERROR

In the same order we cannot create any constructor for the abstract class.

```
abstract class Base
{
  abstract void fun();
}
class Derived extends Base
{ void fun()
  { sop ("Derived class , fun() method");
  }}

class BOMain
{ psvm (--)
  }
```

1) `Base b = new Base();` ✗ (abstract class cannot contain any obj.)
2) `Base b = new Derived();` ✓
3) `Derived d = new Base();` ✗✗✗✗
4) `Derived d = new Derived();` ✓
5) `b.fun();` } ✓ same result
6) `d.fun();` } ✓

eg 2 Using constructor.

```
abstract class Base
{
  Base()
#1{ sop("Base constructor called");
  }
  abstract void funE();
}

class Derived extends Base
{
#2 Derived() { sop("Derived c'tor (constructor) called"); }
```

```
    void fun()
#3  { sop("Derived fun()"); }
}
class Test
{ psvm(--)
  {
    Derived d= new Derived();
  }
}
```

O/P => #1
#2

3) final, abstract => final is a keyword in Java which acts like a constant, but it is not & cannot be extended further.

```
abstract class Base3
{
  final void fun()
  { sop("base class, final method");
}}
class Derived3 extends Base3
   { //empty }

class Main BD3
 { psvm(--)
  { Base b = new Derived();
      b.fun();
 }
}
```

Q1. Is it possible to create abstract and final class?
Q2. Is it possible to have an abstract method in a final class?

Polymorphism

Method Overloading	Method Overriding
within same class, same method's name having diff. signature	with different classes, same method's name having diff. signature

Signature of methods means - order of parameter, number & types.

Method overloading =>

```
Long Math.max (Long a, long b)
int  ----     (int a, int b)
```

Method overriding =>

```
class Emp
{
  getSal()
  {
  --
  }}
```

```
class Sales
{ getSal()
  { --
  }}
```

```
class Program
{ getSal()
  { ---
  }}
```

Polymorphism can be performed by 2 methods

Static Binding (or Early Binding)

→ It is performed during compile-time by the compiler.

→ Binding for static, private and final method is done at compile time because they cannot be overriden further & can be accessed only by the local class (object)

Dynamic binding (or Late Binding)

* STATIC BINDING TAKES PLACE DURING COMPILE TIME & DYNAMIC ↓ RUN-TIME

⇒ // static

```
class Vehicle
{ void display()
  { Sop("Vehicle is super");
  }}
Class Car extends Vehicle
  {
    Void color()
    {
      Sop("My black Car");
  }}
    psvm(—)
  {
```

```
Car obj = new Car();
    obj.color();
    obj.display();
}}

//dynamic
class vehicleName
{
 void display()
{
 Sop ("vehicle detail is super");}}
class CarNew extends VehicleNew
{
 void display()
  { Sop("car detail in subclass");}
   void color()
   {
    Sop(" My Govt. car");
   }
 {psvm(--)
   VehicleNew ob = new CarNew();
       ob.display();
   CarNew obj = new VehicleNew();
         obj.display();
 }}
```

Compiler is not able to resolve the call (or binding) at compile time. Method overriding is one such example.

Inheritance

The mechanism of deriving a new class from an old one is called inheritance.
Inheritance ≠ (new / derived class) acquires all the features of existing class (——)

Single Inheritance =>

```
class Room
{
 int height, length;
  Room(int l, int h)
   {
     height = h;  length = l; }
     int area()
   {
     return height * length;
  }}
   class BedRoom extends Room
    {
       int width;
      BedRoom( int l, int h, int w)
      {
        super(l, h);
         width=w;
      }
      int vol()
     {
     return  length * width * height;
      }
```

public (contains main m
class Room_Bed
{
p svm(--)
{
BedRoom bd=
Sop ("Are
Sop ("Vol
}}

'Super' is
the construct

public (contains main method)

```
class Room_BedRoom
{
  p svm(--)
  {
   BedRoom bd = new BedRoom (14, 10, 16);
     Sop ("Area=" + bd.area());
     Sop ("Volume=" + bd.vol());
  }
```

'Super' is a keyword in JAVA used to encounter the constructor of super class.