19/Aug/19

Introduction (History of Java)

1. Variables
2. Methods
3. Condition statements
4. loop statements
5. Arrays
6. string functions
7. static
8. Non-static & JVM memory - final mock
9. class & object
10. programming
    - Factorial
    - Fibonacci series
    - Reverse a string
    - prime no
    - odd Even
    - pattern
11. Blocks
12. Constructor
13. pass by value
14. pass by reference
15. Composition
16. Inheritance
17. Method overloading - second mock
18. Method overriding
19. Typecasting
20. polymorphism
21. Abstract class
22. Interface
23. Package - pre final mock
24. Access Specifier
25. Encapsulation
26. Collection
27. Object class

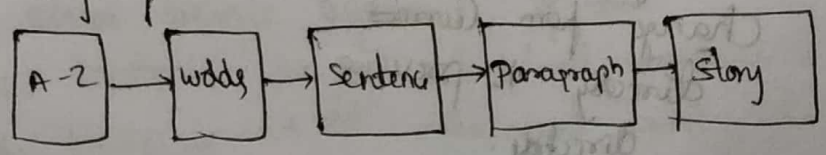
28. String Class
29. Exception handling
30. Threads
31. File handling - final mock -

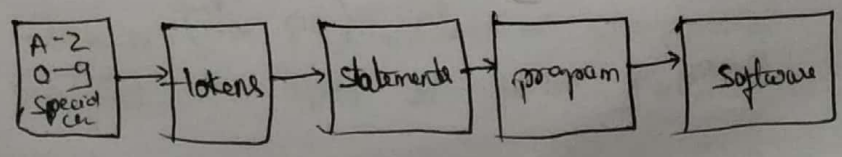
Scanned by CamScanner

History of Java

- James Gosling → founder of Java in the year 1991 and the Software was named as green talk which was developed by a team called 'green-team' later the software was renamed as OAK
  - OAK is a symbol of strength and it is a National tree of Germany
- there was a legal issue i.e there was already an Existing Company called OAK-technologies due to this in the year 1995. They change there slw name from OAK to Java
- James Gosling and his team went to an island of a coffe and the Coffee shop name was Java. hence they kept the slw name as Java.
- Since they Here of a Coffe they kept the Coffee mug as a logo of the Java Software

- What is language?
  - language is a medium to communicate between 2 Entities
  - ~~bitho~~
- What is programming language?
  - It is a medium to Communicate between a human and a machine, We have various programming languages i.e C, C++, Java, C# and python.
- How a language is build?





Softwares Required

Editors

+ Notepad
+ Editplus

Java Software

- Jdk (Java development kit)

Jdk 1.0	Jdk 1.8
1.1	1.9
1.2	1.10
1.3	1.11
1.4	1.12
1.5	1.13
1.6	
1.7	

Execution Area

* Cmd (Command prompt)

IDE (Integrated Development Environment)

Eclipse → SE (Standard Edition)
- ME (Micro Edition)
- EE (Enterprise Edition)

Netbeans

Command Used in Java

Javac	Java Compiler
Java	Java Interpreter
mkdir	make directory
cls	clear screen
cd	change directory
cd..	Change from Current directory to previous directory.

How to launch Notepad?

- On the system
- On the desktop screen left bottom corner click on Windows icon and in the search box type Notepad. Notepad icon will be displayed and double click on the icon.
- Notepad will be launched.

How to download Edit plus?

- On the system and launch the browser, type the URL http://IEBCLT.blogspot.in/.
- Click on mode on the web page - Java software
- Click on 'click here' under Edit plus link.
- Drop box page will be opened and click on direct download.

How to launch Command prompt?

- On the system, on the desktop click on Windows icon, type cmd, Command prompt icon will be displayed and double click on Command prompt icon.

How to download Java Software.

- On the System
  - Go to google type 'download jdk 1.8' and click on the first search result. which will take to the oracle download page, scroll & select accept licence agreement,
  - If the laptop is 32 bit, download windows x86
  - If the laptop is 64 bit, download windows x64
  - Right Click on (This pc), click on properties, where we can find the laptop is 32 bit & 64 bit.

Tokens

Token is a smallest unit of program.
In tokens we have

i) Identifiers
ii) Keywords
iii) literals
iv) Separators
v) Operators
vi) Comments

~~Keywords~~

Identifiers:

Identifiers is a name given to the java program

Keywords:

Keywords are the predefined words which have its own meaning. In Java we have 50 keywords as follows:

Abstract	for	Switch
Assert***	goto*	Synchronized
Boolean	if	this
Break	implements	throw
Byte	import	throws
Case	instanceof	transient
Catch	int	try
Char	interface	Void
Class	long	Volatile
Constant*	native	while
Continue	new	
default	package	
do	private	
double	protected	
Else	public	
Enum****	return	
Extends	short	
final	static	
finally	strictfp**	
float	super	

Literals

Literals are the values which is used in Java programming language to perform some operation.

1. Numeric literals → Integer = 9, 10, 20, 16
   → Decimal = 5.6, 71.36
2. Character literals → 'A' 'g' '$' '#' ' ' ~~'abxy'~~
3. String literal → "Hello" ~~"$#129abxy"~~ "a" " "
4. Boolean → true, false

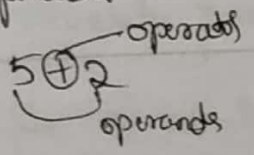
Separators

Separators are used to separate the given code

i) Braces { }
ii) Brackets [ ]
iii) parenthesis ( )
iv) semicolon ;
v) Comma ,

Operators

- Operators are the symbols which is used to perform some operation.
- In operators we have the following



Operator	Category	precedence
Unary	postfix	Expr++ Expr--
	prefix	++Expr --Expr +Expr -Expr ~!
Arithmetic	Multiplicative	* / %
	Additive	+ -
Shift	shift	<< >> >>>
Relational	Comparison	< > <= >= Instanceof
	Equality	== !=
Bitwise	Bitwise AND	&
	Bitwise Exclusive OR	^
	Bitwise Inclusive OR	|
logical	logical AND	&&
	logical OR	||
Ternary	ternary	? :
Assignment	Assignment	= += -= *= /= %= &= ^= |= <<= >>= >>>=

Comments

Comments are used to provide additional information, in Comments we have:

i) Single line Comment

Ex: // - - - - - -

ii) Block Comment

```
/* ______________
   ______________
   ______________
   ____________*/
```

Explain how a Java program will get Executed internally (Q)

Explain Java architecture.

- As a TE we will write the program in the Notepad & Editplus, it is also called as Source code. Which is in human readable format, Once after writing the program save it with an Extension of .Java.
- To Convert this human readable format into machine readable format we will go to Command prompt and perform two operations
  - i) Java c [Compilation]
  - ii) Java (Interpretation)
- .Java file will be given as an input for the Compiler it will check for
  - → Syntax
  - → Rules
  - → Translates from .Java file to .Class file
- .Class file is an intermediate Code which is in byte Code format, which cannot be understood neither by human, nor by machine.
- The .class file will be given as an input for the interpreter [Java] which reads line by line
  - → Execute (JVM)
  - → Translate from .class to binary format

Where the binary format will be understood by OS and gives the output.

- If we violate any syntax & rules we will get Compile Time Error (CTE)
- If we develop any abnormal statements like 1/0 we get Arithematic Exception i.e Run time Error (RTE)

JIT (Just in Time)

- JIT is whole responsible to convert .class file to binary format

JVM (Java Virtual machine)

- It is whole responsible to Execute the Java program
- It doesn't Exist physically, it is a virtual machine which comes into picture during Execution.

JRE (Java Runtime Environment)

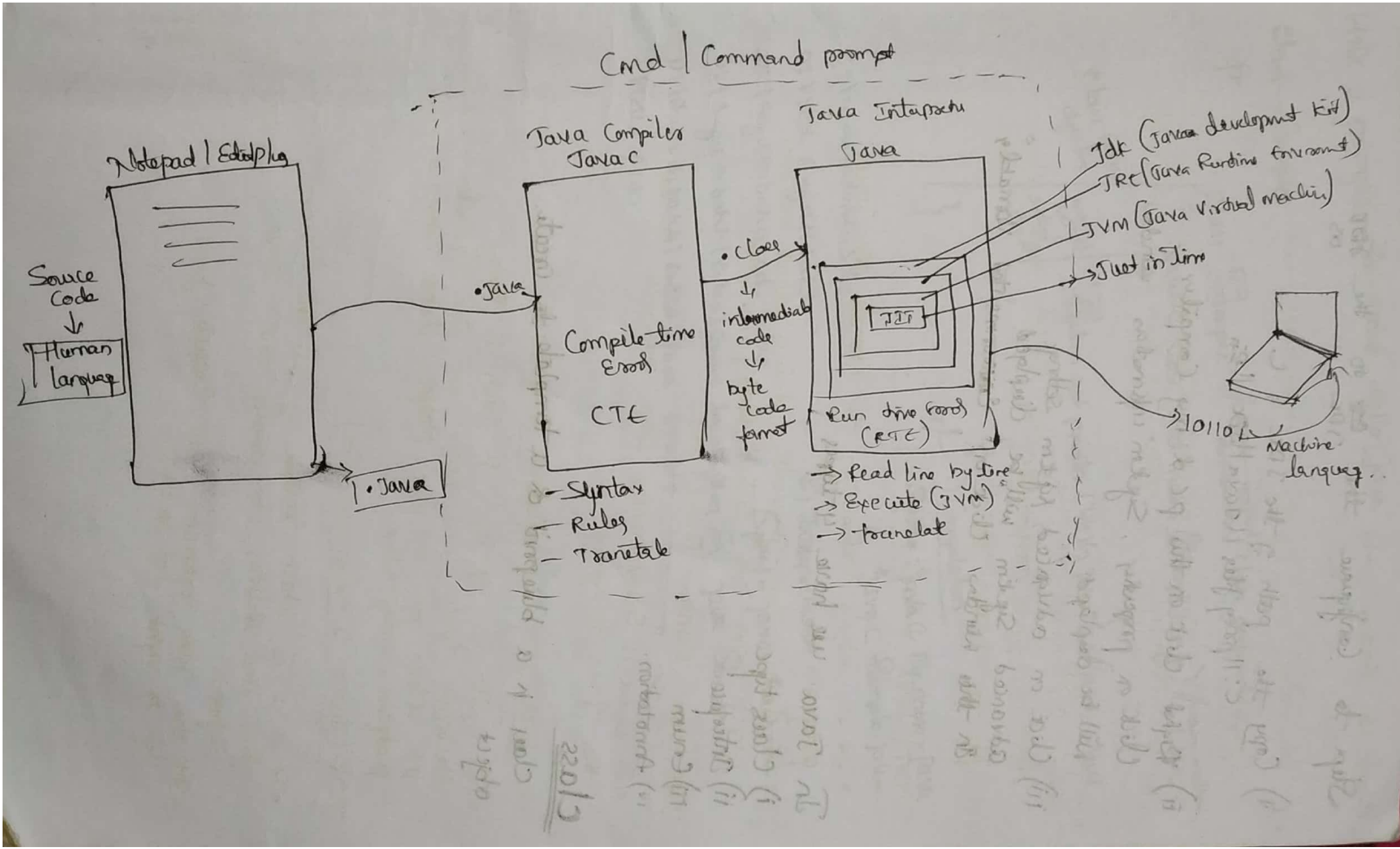
- It is an Environmental Setup provided to run the Java program

JDK (Why Java is platform Independent?)

- JRE will be installed in Each & Every Electronic device and hence we can Execute the .class file on any platform hence Java is platformes independent.

JDK (Java Development Kit)

- It is a kit which Consists of all the library files and utilities to develop the Java Software
- All the Java programe will Execute from left to right top to bottom.



Steps to Configure the JDK to the OS

i) Copy the path of the JDK & C:\Program files\Java\JDK\bin

ii) Right click on this pc of my Computer, Click on properties, System information window will be displayed.

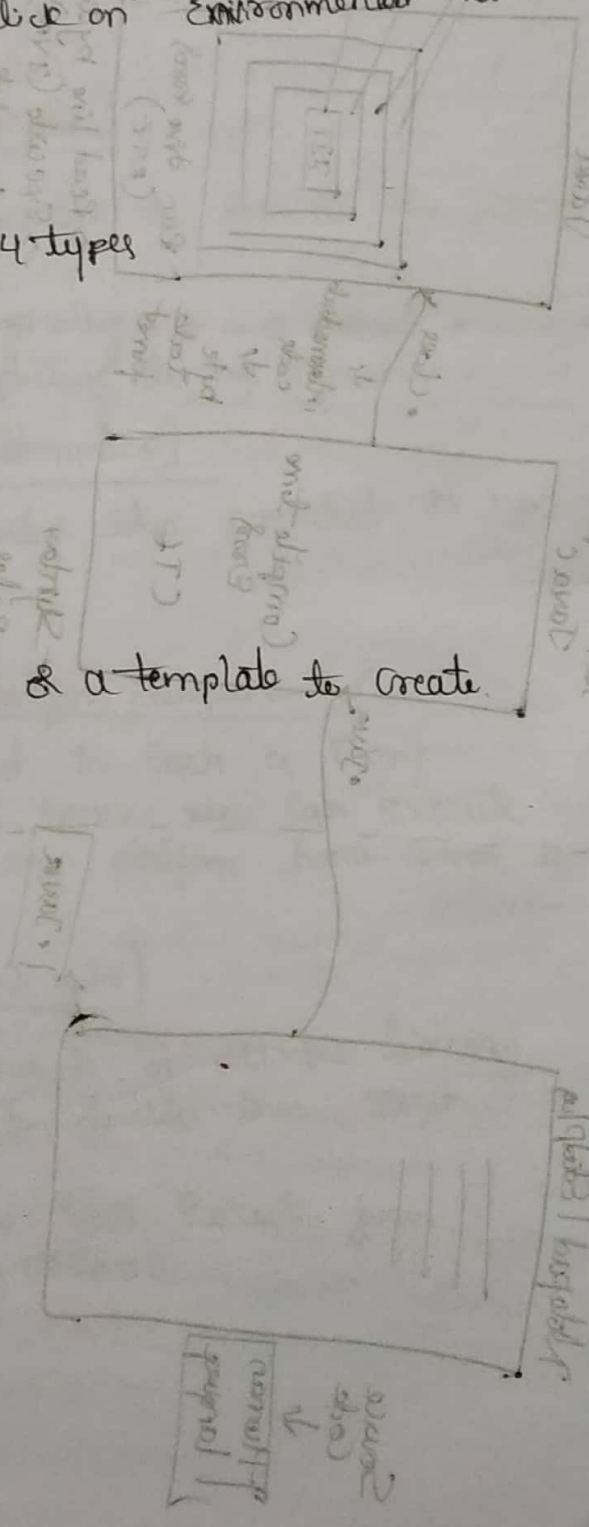
iii) Click on advanced system settings. Advanced System will be displayed In this window click on Environmental Variables

In Java we have 4 types

i) Class type
ii) Interface
iii) Enum
iv) Annotation.

Class

Class is a blueprint & a template to create object



Write a program to print "Hello java"

class Sample

```
class Sample // main method declaration  class declaration
{
    public static void main (String[] args) //main method declaration
    {
        System.out.println ("Hello java"); //print statement
    }
}
```

class body/definition

main method body/definition

1) Compilation

Syntax: javac file_name.java

Eg: javac Sample.java

```
C:\Users\admen>d:
D:\>cd Aug15thbatch
D:\Aug15thbatch>cd Intro
D:\Aug15thbatch\Intro>javac Sample.java
D:\Aug15thbatch\Intro>java Sample
Hello java
```

2) Interpretation

Syntax: java file_name

Eg: java Sample

Write a program to print the literals

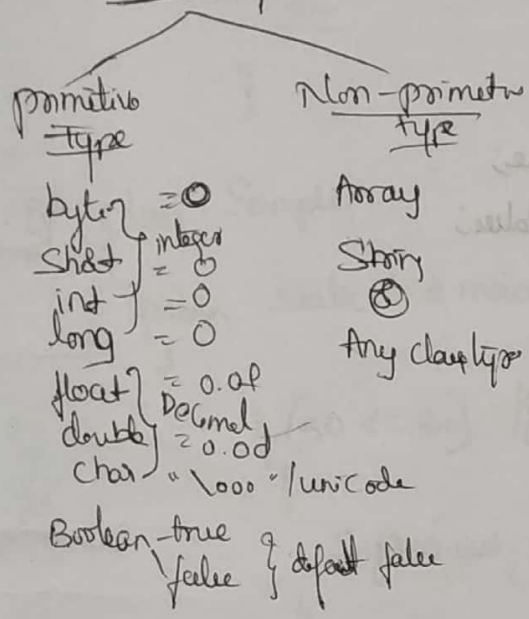
```
class Demo1
{
    public static void main (String[] args)
    {
        System.out.println (10);
        System.out.println ('A');
        System.out.println (true);
        System.out.println (20.56);
        System.out.println (20+20);
    }
}
```

```
class Demo
{
    public static void main (String[] args)
    {
        System.out.println (20+20);
        System.out.println ("the Value is "+20);
        System.out.println (20+" is the Value");
        System.out.println (20+20+" is the Value");
        System.out.println ("the Value is "20+20);
        System.out.println ("the Value is "+(20+20));
        System.out.println ("the Value is "+(½));
    }
}
```

Variable

- Variable is a named memory location which can store some value & data and it can change N number of times during Execution.
- Variable are of 2 type
  - i) primitive type
  - ii) Non-primitive type

Variable type



-2^{n-1} to $(2^{n-1}-1)$

It helps to calculate range of values it can store

	default value	bit	Size
byte	0	8	-128 to 127
short	0	16	-32768 to 32,767
Int	0	32	-2^{32} to $2^{32}-1$
long	0	64	
float	0.0f	32	
double	0.0d	128	
char	'\u000' / unicode	64	

for byte =

-2^{n-1} to $(2^{n-1}-1)$

-2^{8-1} to $(2^{8-1}-1)$

-2^7 to 2^7-1

-128 to 127

$\frac{15}{2}=$

$-2^{15}-1$ to $2^{15}-1$

Variable Declaration

Syntax: datatype Variable-name

```
int a;
   apple;
   -ab;
   $xyz;
```

Variable Initialization

Syntax:

Variable-name = Value;

```
a=10;
int a=10;
```

Variable utilization

```
System.out.println(a);
```

Copying the Value from one Variable to another Variable

```
int x=20;
int y=x
System.out.println(x);
System.out.println(y);
```

Variable Reinitialization

Syntax

```
datatype Variable-name = Value;
         Variable-name = new-Value;
```

```
int C=30;
    C=50;
System.out.println(c);
    C=80;
```

Program

```
class Sample1
{
    public static void main (String[] args)
    {
        int X;
        x=50;
        System.out.println("The Value is"+x);

        Char ch;
        ch=A;
        System.out.println("The Value is "+ch);

        boolean y;
        y=true;
        System.out.println("The Value is "+ y);
    }
}
```

29/08/19

Conditional Statements

I - Condition

Whenever we want to check the conditions, we go for Conditional Statements ie if Condition.

Syntax:

```
if (condition)
{
    ≡≡≡
}
```

Eg: Class Sample

```
{
    public static void main (String [] args)
    {
        if (50 <= 20) // false & true
        {
            System.out.println ("hi");
        }
    }
}
```

II - Else Condition

```
Class Sample;
{
    public static void main (String[] args)
    {
        String Expected = "home";
        String actual = "hey";
        if (Expected == actual) // false & true
        {
            System.out.println ("hi")
        }
        else
        {
            System.out.println ("bye");
        }
    }
}
```

If -Else-if (more-than 1 Condition)

Syntax:

```
if (condition)
{
    ==
}
else if (condition)
{
    ==
}
else
{
    ==
}
```

Ex:

```
class Sample1
{
    public static void main (String[] args)
    {
        if(10<=20)
        {
            System.out.println("if block");
        }
        else if(10==20)
        {
            System.out.println("Else if block");
        }
        else
        {
            System.out.println("both the Conditions are false");
        }
    }
}
```

Nested if else

Syntax:

```
if (condition)
{
    if (condition)
    {
    }
    else
    {
    }
}
else
{
}
```

Eg:

```
class Sample1
{
    public static void main (String[] args)
    {
        if (10<=20)
        {
            if (25==25)
            {
                System.out.println("nested if block");
            }
            else
            {
                System.out.println("nested else");
            }
        }
        else
        {
            System.out.println("else block");
        }
    }
}
```

Looping Statement

Whenever we want to perform some operation for multiple times we go for looping statement.

In looping statement we have 3 types, ↓

i) For loop
ii) While loop
iii) Do while

For loop

① for (initialization; Condition; inc/dec)
{

}

Eg Class Sample
{
public static void main (String[] args)
{
for (int a=1; a<=10; a++)
{
System.out.println ("Hi");
}
}
}

1<2=5
2<=5
3<=5
4<=5
5<=5

② for (int i=10; i>=1; i--)
{
System.out.println(a);
}

o/p
10
9
8
7
6
5
4
3
2
1

* Write a program to print from A to Z

```
class Alpha
{
    public static void main (String[] args)
    {
        for (char c=A; c<=Z; c++)
        {
            System.out.println (c);
        }
    }
}
```

* Write a program to print from Z to A in lower case

```
class Alpha1
{
    public static void main (String[] arg)
    {
        for (char A='z'; A>='a'; A--)
        {
            System.out.println (A);
        }
    }
}
```

* Write a program to print A to Z in a same line

```
class Alpha1
{
    public static void main (String[] arg)
    {
        for (char a=A, a<=Z; a++)
        {
            System.out.print (a);
        }
    }
}
```

```
class Sample1
{
    public static void main (String[] args)
    {
        for (char a='a'; a<='z'; a++)
        {
            System.out.println(a+" ");
        }
    }
}
```

o/p

a b c d e ... z

Write a program to print "hi" without using ;

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

Write a program for infinite for loop.

```
for (;;)
{
    System.out.println ("hi");
}
```

Note:

- In a for loop initialization condition and increment and decrement is not mandatory only for keyword, ( ) and ; is mandatory
- If we dont use increment and decrement for loop will again run for infinite

Variables are classified into Local Variables and Global Variables

Local Variables

Any Variable which is declared within the method is called as local Variable.

→ The scope of the local Variable is from the beginning of the method till the End of the method.

→ It Cannot be classified into Static and non-Static

→ It will not have default Values

→ Local Variable should be initialized before utilization. Else you will get Compile time Error.

```
class Demo
{
    public Static Void main (String[] arg)
    {
        int y;
        y=10;
        System.out.println(y);
    }
}
```

within this method

Global Variable: Any variable which is declared outside the method inside the class is called "Global Variable".

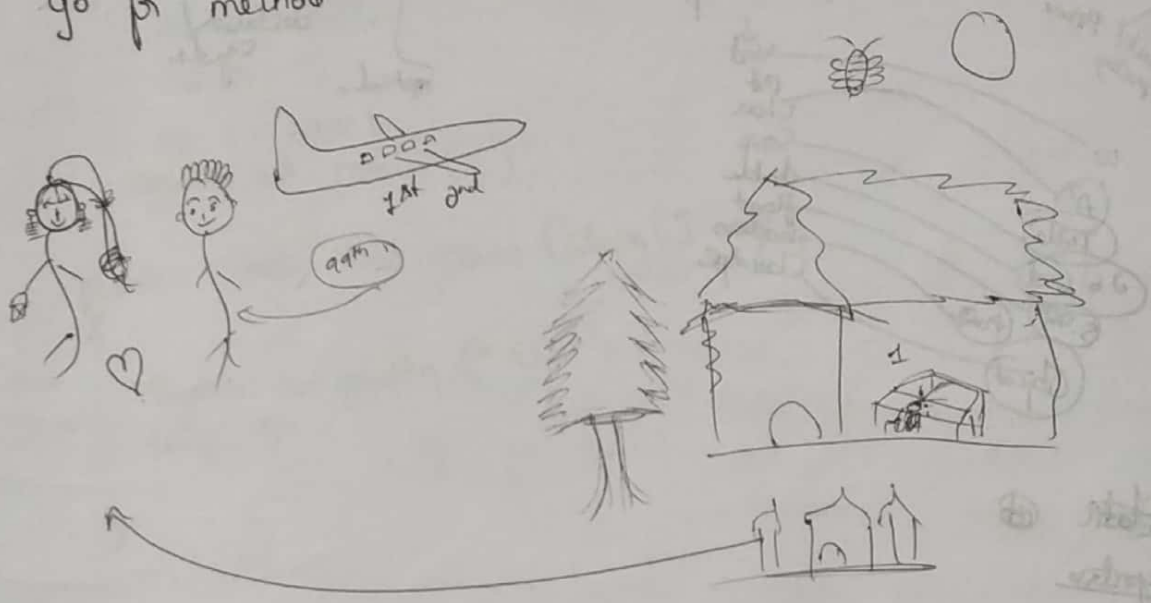
→ The scope of global variable is from beginning of the class till the end of the class
→ It can be classified into static and non static
→ It will have default values
→ Once global variables is declared immediately in the next line it cannot be initialised & re-initialized else you will get compile time error.

Class tester

```
{
    static int a=10;
    public static void main (String[] args)
    {
      a=80;
      System.out.println(a);
    }
}
```

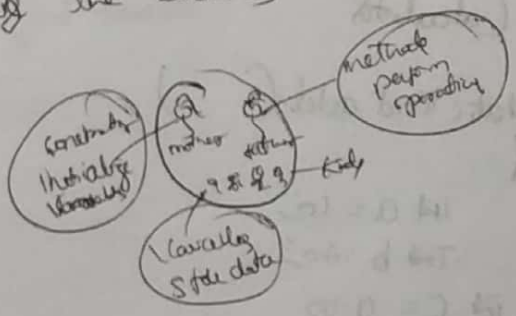
Method

- Method is a block of statements which will get Executed when Ever it is called & Invoked.
- Wherever we want to perform any operation, then we should go for method



- In a Class we have 3 members
  - i) Variables, which is used to store the data/values
  - ii) Methods, which are used to perform some operations
  - iii) Constructors, used to initialize the variables

```
Class Class_name
{
  Variable/data member
  Method/function members
  Constructor
}
```



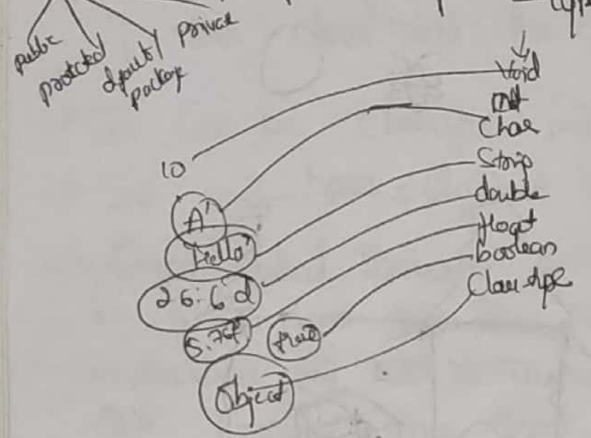
Variables and

method

Syntax ~~Static~~ (~~non static~~) Static ✓ non Static

Access Specifier modifier Return-type method name (argument) // method declaration Syntax.

(argument) – optinal.



~~Static~~

Syntax

datatype Variable-name

Eg: (int a)

(int b, double c)

```
Class Calculator
{
   Static Void add (  )
   {
       int a = 10;
       Int b = 20;
      int c = a+b
          10 + 20
   System.out.println (c);
  }
public Static Void main (String [] args)
{ System out println ("- - . Main Start");
  add ();
  System.out.println ("- - . Main End - -");
 }
}
```

Scanned by CamScanner

Write a program to multiply 2 numbers

```
Class Calculate
{
   Static Void mul()
   {
      int a = 10;
      int b = 5;
      int c = a*b;
      System.out.println(c);
   }
   public Static Void main (String[] args)
   {
      System.out.println("Yo");
      mul();
   }
}
```

Write a program to Calculate area of an circle using method

Final Variable: Any Variable which is declared with the keyword final is called as final Variable.

```
Class Demo
{
   Static Void area()
   {
      final double pi = 3.142;
      int r = 5
      double result = pi*r*r;
                    = 3.142 * 5 * 5;
      System.out.println(result);
   }
   public Static Void main (String() args)
   {
      S.op("--main start--");
      area();
      Sop("--main End--....");
   }
}
```

$\Delta^{le} = \frac{1}{2} \times b \times h$

Square $= a^2$

Rectangle $= w \times h$

Parallelogram $= b \times h$

Trapezoid $= \frac{1}{2}(a+b) \times h$

Circle $= \pi r^2$

Ellipse $= \pi ab$

Sector $= \frac{1}{2} r^2 \theta$

Method with parameters

- Whenever we want to provide input to the methods then we should go for method with parameters.
- parameters are always local to the method/block

Eg:

```
Class Demo
{
  Static Void add (int a , int b) // parameter
  {
    int c=a+b
    System.out.println(c);
  }
public Static Void main (String[] args)
  {
    S.o.p ("---- Main start ----");
    add (50,30);
    add (20,10);
    S.o.p ("---- Main Ends ----");

  }

}
```

6). class Sample1

```
{
    static void area (int r)
    {
        final double pi = 3.142;
        double result pi=3.142; = pi*r*r;
        System.out.println ("area of a circle is " + result);
    }
    public static void main (String [] args)
    {
        System.out.println ("**** Main start ****");
        area(5);
        area(6);
        System.out.println ("**** Main Ende ****");
    }
}
```

output

```
**** Main start ****
area of circle is 78.55
area of Circle 113.112
**** Main Ende ****
```

What is the difference between byte, short, int, long, float and double

⇒ The Size is the difference blw the primitive datatypes with the formula as follows. -2^{n-1} to $2^{n-1}-1$

Type	Size (in bit)	Range
Byte	8	-2^{8-1} to $2^{8-1}-1$ -128 to 127
Short	16	-32,768 to 32767
int	32	-2^{31} to $2^{31}-1$
long	64	-2^{63} to $2^{63}-1$
float	32	1.4E-045 to 3.4E+038
double	64	49e-324 to 1.8e+308
Char	16	
Boolean	1	0 to 65535 true & false

WAP with method with data type after adding 2 numbers

```
class Demo
{
  static int add ( )
  {
    int a=5;
    int b=5;
   int c= a+b;
   return c;
  }
  psvm(_____)
  {
  int x=add();
  S.o.p(x);
  }
}
```

Real time Example with method with return type

Krishna and Ramba joined IBM and Ramba was the topmost performer of the year and she was getting salary Every month, 15,000 Rs and since she was the topmost and the best Employee of the year she got 20,000 Rs bonus Was this bonus decision makers is the manager

WAP the total Earnings Earned by Ramba for this year.

Data	Operations
n_sal = 15,000	multiplication
bonus = 20,000	+
n_mon = 12	bonus

```
class Ibm
{
    static int account ( )
    {
        int m_sal = 15000;
        int n_mon = 12;
        int yearly_sal = m_sal * n_mon;
        return yearly_sal;
    }
    public static void main (String [] args)
    {
        int x = account ( )
        int bonus = 20000;
        int total = x + bonus;
        System.out.println (total);
    }
}
```

WAP to Calculate area of a Circle with method with return type

```
class Circle
{
    static double area1( )
    {
        final double pi = 3.142;
        int r = 5;
        double area1 = pi * r * r;
        return area1;
    }
    public static void main (String [] args)
    {
        double x = area( )
        System.out.println (x);
    }
}
```

WAP to find factorial of a given no.

```
class Sample
{
    public static void main (String [] args)
    {
        int i;
        int fact = 1;
        for (i=1; i<=5; i++);
        {
            fact = fact * i;
        }
    }
    S.o.p (fact);
}
```

Static

- Any member of the class declared with the keyword static is called as static member of the class.
- static is always associated with class
- static is one copy
- Whenever we want to access static members from one class to another class we should use

  Class_name.Variable_name

  (or)

  Class_name.method_name();
- All static members will be stored in "Static pool area"
- Variables and methods can be classified into static and non-static and Constructors is always non-static

class classname
{
Variable, method } → Static / Non static

Constructor → always Non static

```
class Sample
{
  static int a=10;
  static void add()
  {
    S.o.p("Hi");
  }
  public static void main(String [] args)
  {
    System.out.println(a);
    add();
  }
}
```

Note:

- We can develop N number of classes in a single file, those many .class files will get generated
- Whichever class is having main method that class name should be the file name.

```
class A
{

}
class B
{
  PSVM(  )
}
class C
{

}
```

→ class B.java

~~class Circle~~

{

~~static void area~~

Eg1 class Tester

{

static int x=80;

static void add()

{

int a=10;

int b=20;

int c=a+b;

System.out.println(c);

}

class Sample

{

public static void main(String[] args)

{

Tester.add();

S.o.p(Tester.x);

}

}

Eg2 class Circle

{

static void area()

{

final double pi=3.142;

int r=5;

double result = pi*r*r;

System.out.println(result);

}

}

class mainclass

{

public static void main(String[] args)

{

System.out.println("*** main start ***");

Circle.area();

System.out.println("*** main End ***");

}

}

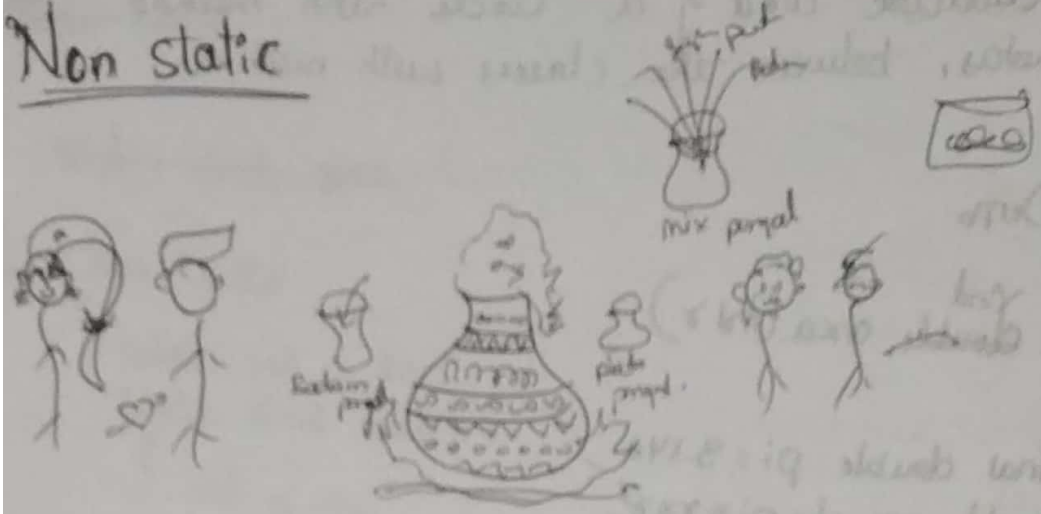
WAP to Calculate area of a Circle with method with parameters, between the classes with methods as static

⇒
```
class Demo
{
    static void area(int r)
    {
        final double pi = 3.142;
        double result = pi*r*r;
        System.out.println(result);
    }
}
class mainclass
{
    public static void main (String[] args)
    {
        System.out.println("*** main start ***");
        Demo.area(5);
        System.out.println("*** main End ***");
    }
}
```

WAP to Calculate area of the circle with method with return type bw the class

⇒
```
class Demo
{
    static double area()
    {
        final double pi = 3142;
        double r = 5;
        double result = pi*r*r;
        return result;
    }
}
class mainclass
{
    public static void main (String[] args)
    {
        System.out.println("*** main start ***");
        double y = Demo.area();
        System.out.println(y);
    }
}
```

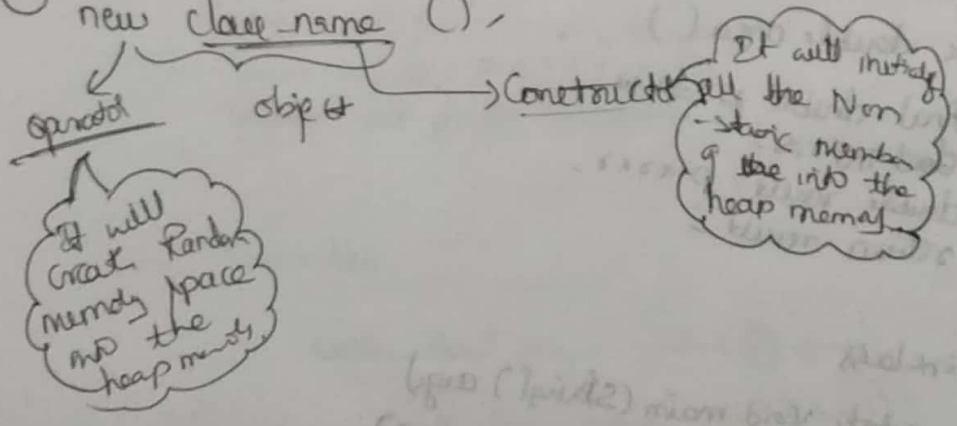
Non static



- Any member of the class declared without the keyword static is called as Non-static member of the class
- Non static is always associated with object
- Non static is multiple Copies
- Whenever we want to access from Non static do static we have to create object
  i.e Object.Variable_name;
  (2)
  Object.method_name;
- All the non-static member is stored in 'Heap memory'

Syntax for object Creation.

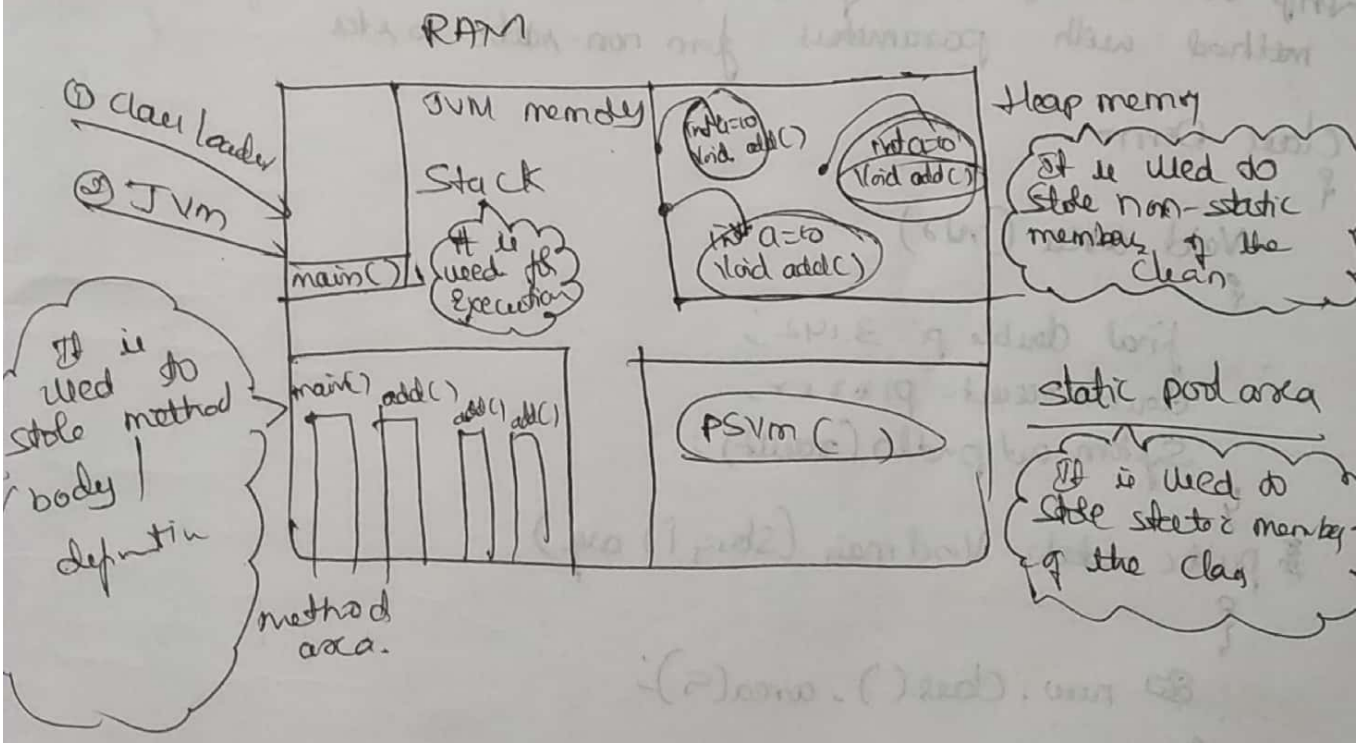
(1) new class_name ();



(2)

```
class Demo
{
    int a = 10;
    void add()
    {
        S.o.p("Hi");
    }
public static void main (String[] args)
    {
        new Demo().add();
        S.o.p(new Demo().a);
        new Demo().add();
    }
}
```

(3)



```
class Tester
{
    void area()
    {
        final double pi = 3.142;
        int r = 5;
        double result = pi*r*r;
        Sop(result);
    }
public static void main(String[] args)
{
    new Tester().area();
}
}
```

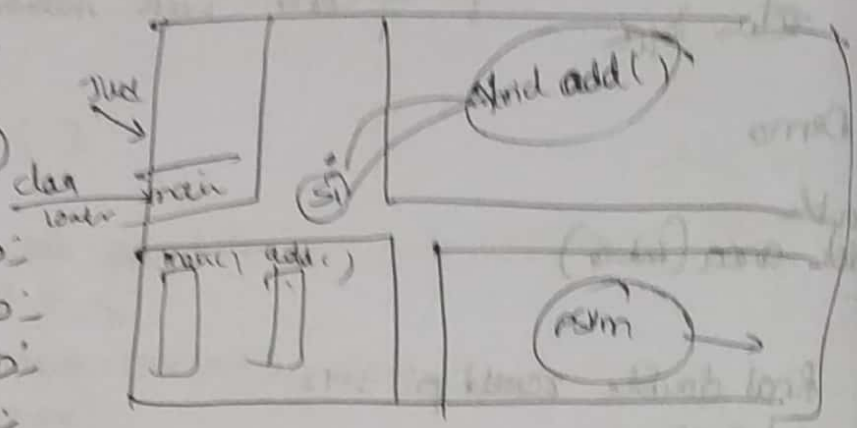
WAP to Calculate Area of a Circle with method with parameters from non-static to static

```
class Demo
{
    void area(int r)
    {
        final double pi = 3.142;
        double result = pi*r*r;
        System.out.println(result);
    }
public static void main(String[] arg)
    {
        new.class().area(5);
    }
}
```

WAP to calculate area of a circle with method with return type

```
class Demo
{
    double
    void area(int r)
    {
        final double result pi=3.142
        int r=5
        double result= pi*r*r;
        return result;
    }
    public static void main(String[] args)
    {
        double x= (area)new Demo().area();
        System.out.println(x);
    }
}
```

```
class Sample
{
  void add()
  {
    int a=10;
    int b=20;
    int c=a+b;
    S.o.p(c);
  }
public static void main(String[] arg)
  {
    new Sample S1=new Sample();
        S1.add();
       S1.add();
  }
}
```



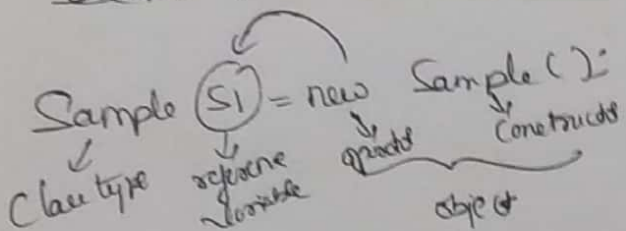
Reference Variable Declaration:

Syntax

```
class-name reference-variable;
Sample S1;
```

Reference Variable Initialization

```
Reference Variable = new Sample();
object
Sample S1 = new Sample();
```

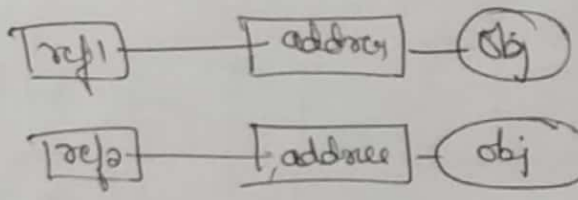


Def: • It is a special type of variable which is used to store object address
• A reference variable can hold two values
i) object address
ii) Null

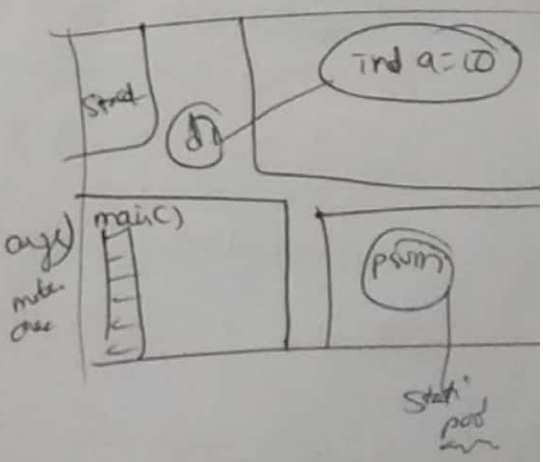
WAP to print the address of the object

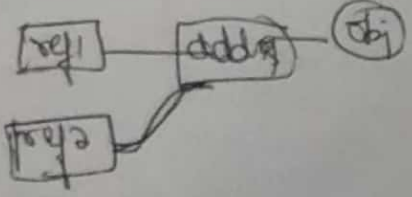
```
=> Class Sample1
{
    int a=10;
    public static void main (String[] args)
    {
        Sample s1= new Sample1();
        Sample s2= new Sample1();
        System.out.println (s1);
        System.out.println (s2);
    }
}
```

o/p Sample1@15db9742
Sample1@6d06d69c




```
=> Class Demo
{
    int a=10;
    public static void main (String[] args)
    {
        Demo d1 = new Demo();
        Demo d2 = d1;
        S.o.p(d1);
        S.o.p(d2);
    }
}
```



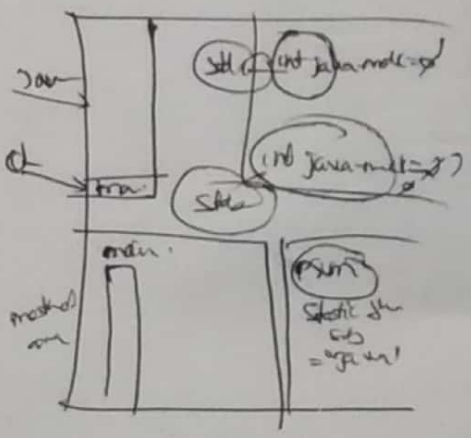


o/p Sample1@15db9742
Sample1@15db9742

Real time Example on static and Non-static

⇒ Keerthi joined Qspiders along with parvathi in the first mock of Java parvathi got 1, and keerthi got 2 and he was very much frustrated and he took the remock and got the mock rating as 1 WAP to parvathi and keerthi mock rating using static and Non-static concept.

```
Class Qspider
{
    int java_mock;
    Static String Sub_name = "java";
public static void main (String[] arg)
{
    Qspider std1 = new Qspdr;
    S.o.p(0)
      std1.java_mock = 1;
      S.o.p (std1.java_mock);
    Qspider std2 = new Qspider;
      std2.java_mock = 2;
      S.o.p (std2.java_mock);
    std2.java_mock = 1;
    S.o.p (std.javamock);
}
}
```



Eg: ① mobile_Cost
mobile_modle_name
mobile_Colour

② Car_Cost
Car_model_name
Car_type (petrol/d:)

③ School_name
school_grade
school_strength

④ tv_Cost
tv_brand
tv_type

⑤ bike_Cost
bike_brand
bike_Colour

⑥ Course_name
Univs_name
Exam_Cost

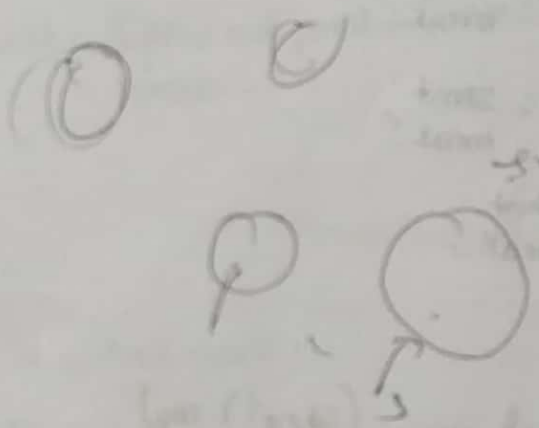
⑦ laptop_name
laptop_Cost
laptop_brand

⑧ home_name;
home_Cost
home_Colour

⑨ Emp_id
Emp_sal
Emp_grade

⑩ Cloth_Colour
Cloth_Cost
Cloth_Sz
(2 x 2 m)

Class KSRTC
{
int Busfair;
Static String Service provider = "KSRTC"
public Static Void main (String [] args)
{
KSRTC Hb = new KSRTC();
Hb. Busfair = 500;
KSRTC CM = new KSRTC();
CM. Busfair = 800;
S.o.p(Hb.Busfair);
S.o.p(CM. Busfair);



Conditional Statements

Switch Case

Switch Case is used for pattern matching based on the input it will match the Case and Execute that particular statement.

Syntax

```
Switch (charseq)
{
   Case charseq : stmt;
                  break;

   Case charseq : stmt;
                  break;

   Case charseq : stmt;
                  break;

   default : stmt
             break;

}
```

Eg

```
Public static Void main (String[] arg)
{ int input=2;
  Switch (input)
  {
    Case 1: S.o.p ("idly");
            break;
    Case 2: S.o.p ("Vada");
            break;
    Case 3: S.o.p ("Hot Hot masala dosa");
            break;
    default : S.o.p ("In valid input");
              break;

  }
}
```

Q2) Class Sample1

```
{
    public static Void main (String[] args)
    {
        Char grade = 'A';
        switch (grade)
        {
            Case 'A' : System.out.println ("FCD");
                       break;
            Case 'B' : System.out.println ("FC");
                       break;
            Case 'C' : System.out.println ("SC");
                       break;
            Case 'D' : System.out.println ("Go Home");
                       break;
            default : System.out.println ("invalid input");
                      break;
        }
    }
}
```

o/p

FCD

o/p → without break

FCD
FC

Write a program for the below Scenario

=> Shiva went to ICICI bank and he took a loan amount of 50,000 rupees and every month he paid an interest of 500 rupees and he gave 20,000 rs to his friend. This happened for 1 year but his friend repays the amount 25,000 after one year and Shiva Clear the loan after Exactly one year.

① Help to demonstrate the total interest paid by Shiva to the bank

② Help how much amount he has added as an interest to the bank when he received the money from his friend.

=> class Sample1
{
    loanin (int no_month)
    {
        int paid = 500 * no_month;
        return paid;
    }
    help (int amt)
    {
        int

```
class Shivam
{
static int Calinterest (int n-months)
{
    int iamt = 500;
    int result = n.months * inamt;
    return result;

}
static int loan( )
{

    return 50,000;

}
static int helpayment (int x)
{
    return x+5000;
}
public static void main (String[] args)
{
    int loanamt = loan();
    int fiendamt = loanamt-30,000;
    int Ramt = helpayment (fiendamt);
    int Iamt = Calintent (12);
    int remaingamt = 30,000;
    Totalamt = remamt + x;
```

Blocks

Java provides separate blocks called
- Static initialization block (SIB)
- Instance initialization block (IIB)

i) Static initialization block

→ Any block which is declared with the keyword static is called as static initialization block.

* SIB will get Executed "before main method"
* We can have 'N' no of SIB, the order of Execution is "sequential"

Syntax

```
Static {
        ==
        }
```

Eg:
```
class Sample
{
    Static int a;
    Static {
        S.o.p("SIB1");
    }
    Static {
        S.o p ("SIB2");
    }
    psvm (String[] ax)
    { S.o.p ("---ms"---);
      S.o p ("---m€---)
```

o/p SIB1
SIB2
___ ms---
___ me---

Note:
Developer will use SIB to initialize static member

1) We can Execute SIB without using main method in the Version of JDK 1.5 and before 1.5

2) Updated Version JDK 1.13 we Cannot Execute SIB without main method

3) We use SIB method to give path of selenium Jar

4) we can Use single line SIB in 1st statement of main method

IIB (Instance Initialization block)

- Any block which is declared without the keyword Static is called as IIB
- IIB will get Executed @ whenever the object is created
- We can have 'N' no of IIB, the order of Execution is Sequential
- It is used to initialize the non-static member.

Syntax

```
{
==
}
```

Eg:

```
class Tester
{
    {
        S.o.p ("IIB1");
    }
static
    {
        S.o.p ("SIB1");
    }
psvm (String[] args)
{
    S.o.p ("---ms---");
    new Tester();
    S.o.p ("---me---");
}
```

o/p

```
SIB1
---ms---
IIB1
---me---
```

Class and Object

→ class is a blueprint / a template to create object

What is object?

→ the object is an real-time Entity which has its own state and behavior

→ State: State defines what data it should hold.

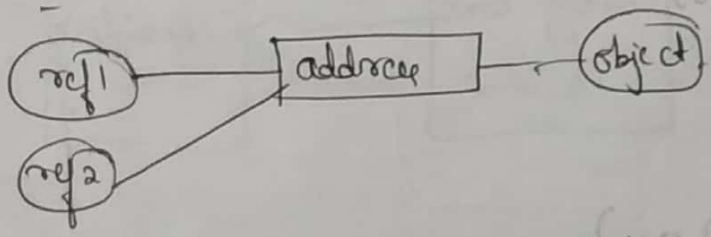
→ Behaviour defines the way it can behave

→ the Non Static Variables defines the state of the object.

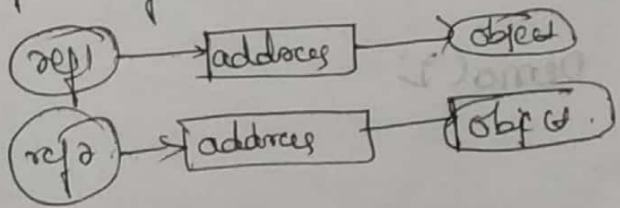
→ Whenever we create an object we will have both State & behaviour.

→ The object address will be stored in reference Variable.

→ Multiple reference Variable can hold single object address



⇒ Multiple reference Variables can hold multiple object address

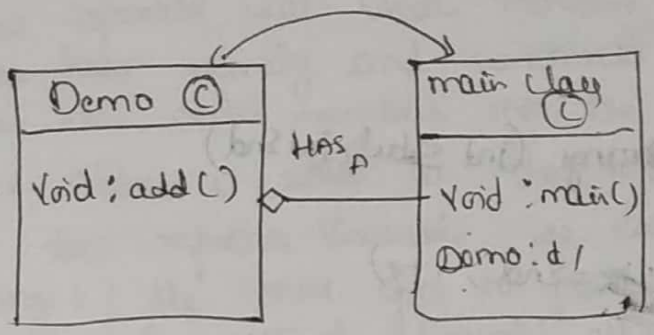


Eg: class Sample
{
int a = 10 //state
static add();
{ S.o.p(a+a); // behaviour
}
}

Composition: The class having a object of another class is called Composition.
* It has "HAS A" -

class Demo
{
void add()
{
int a = 10;
int b = 20;
S.o.p(a+b);
}
}
class mainclass
{ psvm(String[] arg)
{
Demo S1 = new Demo();
S1.add();
}
}

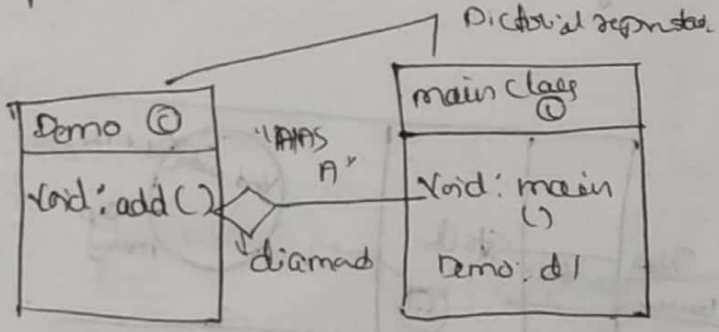
A "HAS A" relationship is a pictorial "relationship"



Composition: A class having an object of another class is called as "Composition".

Class Diagram: It is pictorial representation to represent the members of the class

* Composition is also called as "HAS A" relationship



WAP to print from the start and the End range using non-static method

```
Class Sample
{
    Static void printrange (int Start, int End)
    {
        for(int i=start ; i<=End; i++)
        {
            S.o.p(i)
        }
    }
public static void main (String[] arg)
  {
  Sample S1=new Sample();
       S1.printrange.(10,50);

  }
}
```

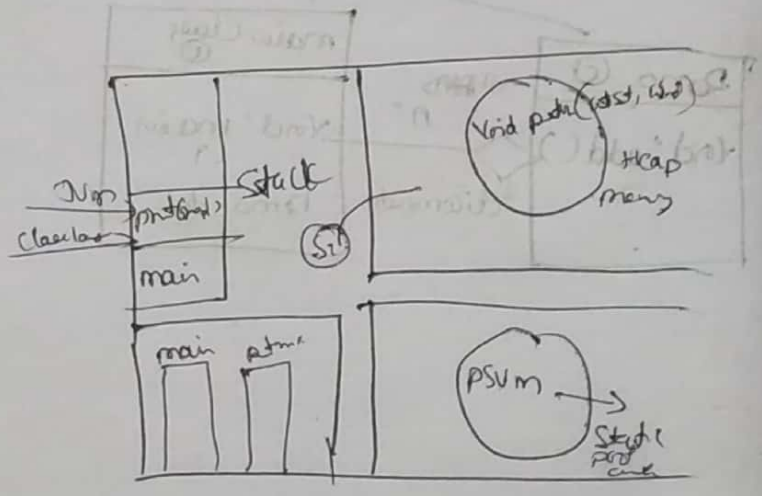


Fig the above program:

1) Once after writing the program in notpad & Edit plus as soon as we Compile the class loader will load all the static members into static pool area and main method declaration will be stored in static pool area, the method body will be stored in method area.

ii) JVM stack Executing from main method on the stack, the first statement is the object created, the Equal operator will work from right to left, Here new operator will Create random memory space into the heap memory and Constructor will initialize all the non static members into the heap memory.

iii) The object address is stored in the reference Variable & through that reference Variable we call the method print range() the control goes to print_range method from the main method, Executes all the statements in print range method and control comes back to the main method.

Constructor

Constructor is a special member & special method of the class which is used to initialize the data members & Variables

Syntax

```
class class_name
{
    class_name()
    {
    }
}
```

→ Constructor

Eg:

```
class Sample
{
    Sample()
    {
    }
}
```

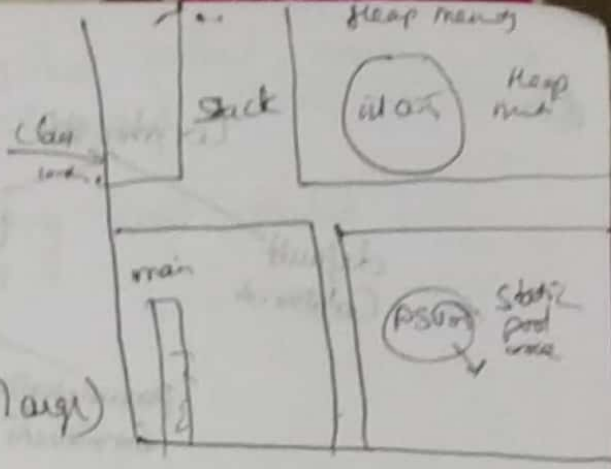
Rules

i) Constructor name should be same as Classname
ii) Constructor will not have return type
iii) Constructor will not return value
iv) Constructor is always non-static
v) Whenever the object is created, the Constructor will get invoked.

Class Sample
{
← Variable / data member { Static / Non-static
← methods / function
← Constructor → Non-static.

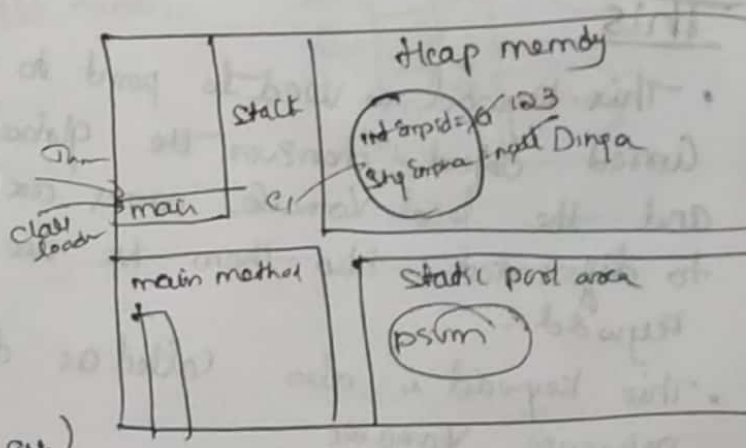
Eg: class Tester
{
Tester()
{
S.o.p("Hi");
}
public static void main(String[] args)
{
S.o.p("--- MS ---");
new Tester();
S.o.p("--- ME ---");
}
}

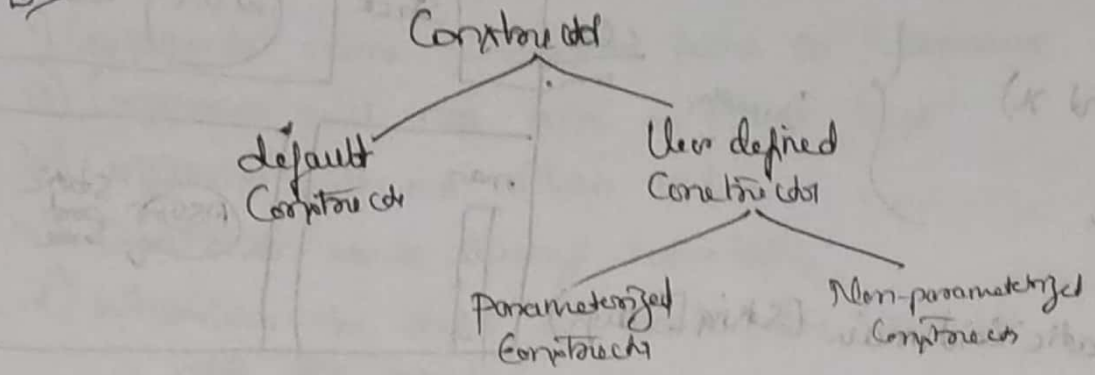
```
class Tester
{
    int a;
    Tester (int x)
    {
        a=x;
    }
    public static void main (String[] args)
    {
        S.o.p("---ms---");
        Tester t1 = new Tester;
        S.o.p(t1.a);
    }
}
```



KAP to initialize Empid, Empname through Constructor

```
class Employee
{
    int Empid;
    String Empname;
    Tester (int a, String b)
    {
        Empid = a;
        Empname = string;
    }
    public static void main (String[] args)
    {
        Employee e1 = new Employee (123, "Dinga");
        S.o.p(e1.empid);
        S.o.p(e1.Empname);
    }
}
```





Note:

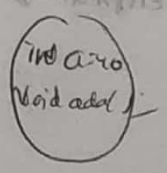
- Constructors are of two types Default Constructor and User defined.
- Once we develop User defined Constructor there will be no default Constructor.
- Each and Every class will have default Constructor

This

- This keyword is used to point to the Current Object, whenever the global variable and the local variable names are same to differentiate b/w them we use "This" keyword.
- This keyword is also called as default reference variable
- This keyword can be used only in the non-static context

```
⇒ class Demo
  { int x=80;
    void add()
    { sop(x);
      sop(this.x);
    }
  }
```

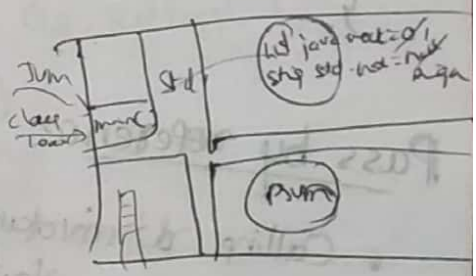
new Demo();



Note: A program will get Compiled without main method but it Cannot be interpreted

WAP to initialize java-mock rating and student name through Constructor

```
=> Class Ospidus
{
    int java-mock;
    String std-name;
    Ospidus (int java-mock, String std-name)
    {
        this.java-mock = java-mock;
        this.std-name = std-name;
    }
    public static void main (String() args)
    {
        Ospidus std1 = new Ospidus (1, "Dinga");
        S.o.p (std1.java-mock);
        S.o.p (std1.std-name);
    }
}
```



Pass by Value

Calling & invoking the method by passing primitive type of data is called as "pass by Value" or "Call by value"

Eg:
```
Class Tester
{
    Static void add(int a)
    {
        S.o.p(a+a);
    }
    Public static void main (String[] arg)
    {
        int x = 10;
        add(x);
    }
}
```

Pass by reference

- Calling & invoking the method by passing reference variable is called as pass by reference.
- pass by reference is also called as pass by Value

Eg:
```
Class Sample
{ int icecream = 10;
    Static void needicecream (Sample S2)
    {
        S.o.p (S2.icecream);
    }
public static void main (String[] arg)
{ sample S1 = new Sample();
    S.o.p (S1.icecream);
    needicecream (S1);

    }
}
```


```
Class TBMAunty
{
  void testEngg()
  {
    S.o.p("opening JB testEngg");
  }
}
Class Uncle
{
  psvm()
  {
    TBM_aunty a1 = new TBMaunty
    Dinga.needjobs(a1)
    Dagi.needjobs(a1);
  }
}
```

```
Class Dinga
{
  Static void needjobs(TBMaunty a2)
  {
    a2.testEngg();
  }
}
Class Dagi
{
  Static void needjobs(TBMaunty a3)
  {
    a3.testEngg();
  }
}
```

Interview Questions

1) What do you mean by System.out.println

Global reference Variable and Local reference Variable

Class Sample
{
Any reference Variable which is declared outside the method inside the class is called as global reference Variable

Any reference Variable which is declared within the method is called as Local reference Variable.

Eg:
```
Class Tester
{
Global reference Variable → Tester t1 = new Tester();
        void add()
        {
          Tester t2 = new Tester();
        }
}
```

Eg:
```
Class Sample
{
static int y=80;
static Sample S1 = new Sample();
}
Class Demo
{
  psvm (String[] args)
  {
    S.o.p (sample.y)
    S.o.p (Sample.S1);
  }
}
```

- System is a class
- Out is a Global Static reference Variable
- println is a non-static method of printStream class

```
class Sample // printStream
{
  void add()
  {   ad println

  }
}
```

```
class Tester // System
{
  static Sample S1 = new Sample();
  static printStream out = new printStream();
}
```

```
class Demo
{
  psvm (   )
  {
    Tester . S1 . add();
```

Tester → Class name; S1 → Global Static reference Variable; add() → non-static method of Sample

→ System.out.println();

② What is method overloading?

→ Developing multiple methods with the same name but Variations in arguments list is called as method overloading.

③ What is method overriding?

→ Developing a method in the Sub class with the same name and the Signature as in the Super class but different implementation in the Sub class is called as Method overriding.

④ What is Inheritance?

→ Inheriting the properties from one class to another class is called as inheritance.

⑤ What is Encapsulation?

→ Declare the data member as private and restrict the direct access outside the class and provide the indirect access through public services called getter() and setter() is called as Encapsulation.

Scanned by CamScanner

(6) What is abstraction?

→ Hiding the Complexity of the System and Exposing only the required functionality to the End User is called as abstraction.

(7) What is Polymorphism?

→ An object showing different behaviour at the different stages of it life cycle is called as Polymorphism.

(8) Resume points

(V) Good knowledge on method overloading
(VI) Good understanding on method overiding
i) Good understanding on static and non-static
ii) Good understading on Variables
iii) Good understanding on Constructors
iv) Good knowledge on pass by value & pass by reference

(9) What do you mean by public static void main(String[] arg)

⇒ public means application level access.

⇒ Here the main method should have application level access and hence it is declared as public

⇒ Main method is static because it should be loaded into the memory before Execution and it should be one copy.

⇒ Main method is declared as Void because it doesn't return any value

⇒ Main is the method name String[] args → are the parameters and it should be of string type where it receives the input in the form of string in Command line arguments

What is the difference b/w static and Non-static

Static	Non-static
i) Any member of the class declared with the keyword 'static' is called as static member of the class	i) Any member of the class which is not declared with the keyword 'static' is called as Non-static member of the class
ii) static is always associated with class	ii) Non-static is always associated with object.
iii) static is one copy	iii) Non-static is multiple copies
iv) Whenever we want to access static member from one class to another class we should use Classname.Variable-name; (or) Class-name.method-name;	iv) When ever we want to access from non-static to static we should access through object object.Variable-name; (or) object.method-name; (or) reference-Variable.Variable-name; (or) reference-Variable.method-name
v) Static members are also called as 'Class members'	v) Non-static members are also called as Instance members (or) object members.
vi) All the static members will be stored in static pool area	vi) All the non-static members will be stored in heap memory
vii) All the static members will get initialized into the static pool area when the class is loaded	vii) All the non-static members will be initialized in the heap memory whenever the object is created
viii) Java provides a separate block called SIB to initialize static members	viii) Java provides a separate block called IIB to initialize non-static members
ix) SIB will get executed before main method	ix) IIB will get executed whenever an object is created
x) We can have N no of SIB; the order of execution is sequential	x) We can have N no of IIB the order of execution is sequential

WAP to print Hello Java with " "

```
class Sample
{
    public static void main (String[] args)
    {
        System.out.println("\" hello java\" ");
    }
}
```

In how many ways you can declare main method
⇒ 4 ways

```
public static void main (String[] args)
static public void main (String[] args)
Static public Void main (String args[])
Public static void main (String... args)
```

Array

- Array is a linear data structure which is used to store homogeneous type of data.
- Drawbacks of array
  i) Size is fixed, and we can store only homogenous type of data.

① Array Declaration

datatype[] array-name;

int[] arr;

② Array Size initialization

array-name = new datatype(size);

arr = new int[];

③ Storing the value into array

Syntax

array-name[index] = value

arr[0] = 10;
arr[1] = 20;
arr[2] = 30;
arr[3] = 40;

④ Array utilization

System.out.println(arr[0]);
S.o.p(arr[1]);
S.o.p(arr[2]);
S.o.p(arr[3]);
S.o.p(arr[4]);

o/p
10
20
30
40

↓

Array Index out of bounds

⑤ To Calculat the length of the array

Syntax: array-name.length

S.o.p(arr.length) → o/p 4.

⑥ for (int i=0 ; i<arr.length ; i++)
{
S.o.p (arr[0])
}

⑦ Array Value re-initialization

Syntax

array-name(index) = new value;

arr[1] = 60;

⑧ If the array haven't store any data and if we try to print we get default value;

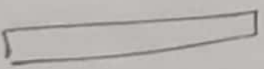
⑨ Array Declaration and store the Value directly

datatype [] array-name = {v1, v2, v3};

int[] arr = {10, 20, 30, 40};

⑩ Copying from one array to another array

int [] acc = arr;

arr 

Class Sample2

```
{
  public static void main (String[] arg)
  {
    String[] arr = new String [4];
    arr[0] = "padi";
    arr [1] = "Hot hot masala dosa";
    arr [2] = "gobi";
    arr (3) = "biryani";
    System.out.println ("*********** \n");
    System.out.println ("Index \t values");
    System.out.println ("*********** \n");
    for (int i = 0; i<arr.length; i++)
    { System.out.println (i + "\t" + arr[i]);
    }

  }

int [] abb = {10, 20, 30, 40, 50};
for (int i=0; i<abb.length; i++)
{
  System.out.println (abb[i]);

}
```

WAP to cal sum of digit
i/p → int[] arr = {10, 20, 30, 40}

```
Class sample
{
  public static Void main (String[] args)
  { int arr[] = {10,20,30,40}
    int sum = 0
    ~~int no~~;
    ~~no~~
    ~~while~~ if (arr[]=0)
      {
        ~~for(arr[]=1~~;
        for (int i = arr[0] ; i<arr[4] ; i++)
        {
          int rem = arr[i] % 10;
          Sum = Sum + rem;
          arr[i] = arr[i] / 10;
```

10)10(1
10
0

***

```
Class Sample
{
  public static Void main (String[] arg)
  {
    int arr[] = {10,20,30,40}
    int sum=0;
    if (arr[]!=0)
    {
      for (int i=0 ; i<arr[i] ; i++)
      {
        Sum = Sum+arr[i];
      }
      S.o.p (sum);
    }
  }
```

Method Overloading

Developing multiple methods with the same name but different in argument list is called as method overloading.

Variation in argument list means

i) Variations in data type
ii) Variation in the length of the argument
iii) Variation in the order of occurance of the argument

Rules

i) The method name should be same
ii) There should be variation in arguments list
iii) There is no restriction on access specifier, modifier ad ~~return~~ type
iv) We can overload both static and non-static method

1) Can we overload main method ?
=> Yes we can overload main method with variation in arguments list.

```
class Whatsapp
{
  static void send (int no)
    { s.o.p ("sending no" + no);
    }
  static void send (String msg)
    { s.o.p ("sending text msg" + msg);
    }
  static void send (int no, String msg);
    { s.o.p ("Send no & text msg" + no + " " + msg);
    }
  static void send (String msg, int no);
    { s.o.p ("send msg & no" + msg + " " + no);
    }
}
class Mainclass
{
  public static void main (String[] arg)
    {
      Whatsapp.send (123);
      Whatsapp.send ("Hello");
      Whatsapp.send (123, "hi");
      Whatsapp.send ("hi", 123);
    }
}
```

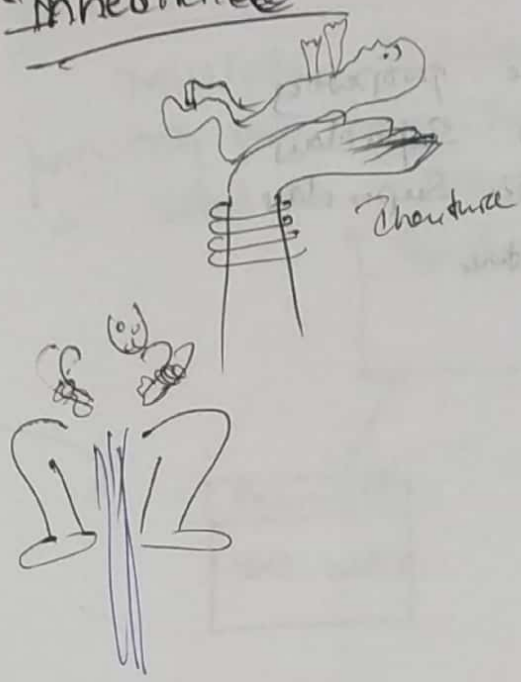
```
Class Gmail
{
  public void Compose (int no)
  {
    System.out.println ("Compose by sending no");
  }
  void Compose (String txt)
  {
    s.o.p("Compose by sending to mail");
  }
  void Compose (int no, String txt);
  {s.o.p ("Compose by sending" no, txt");
  }
  Void Compose (String text, int no);
  {s.o.p ("Compose by sendy "+text, +" " + no);
  }
}
Class Main
Class Gmail
{
  Public Static Void main (String [] args)
  {
    Gmail f1 = new Gmail();
      f1.Compose (123);
      f1.Compose ("Hi, hello");
      f1.Compose (12.5, "Cool");
      f1.Compose (12.5, "Cool");
      f1.Compose ("Cool", 123);
  }
}
```

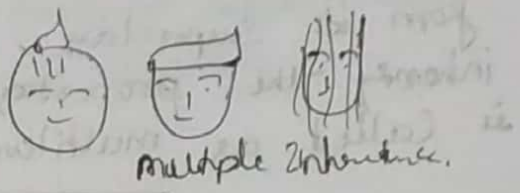
```
class MainClass
{
  public static void main (String[] ay)
  {
    SOP ("*** main start ***");
    main (123);
    main (123, "hello");
    main ("Cool", 456);
    S.o.p ("*** main Ende ***");
  }
public static void main (int arg)
{
  S.o.p ("main (int)");
}
public static void main (int a, String b);
{
  S.o.p ("main (int, String)");
}
public static void main (String b, int c);
{
  S.o.p ("main (String, int)");
}
}
```

o/p

```
main (int)
main (int, String)
main (String, int)
```

Inheritance

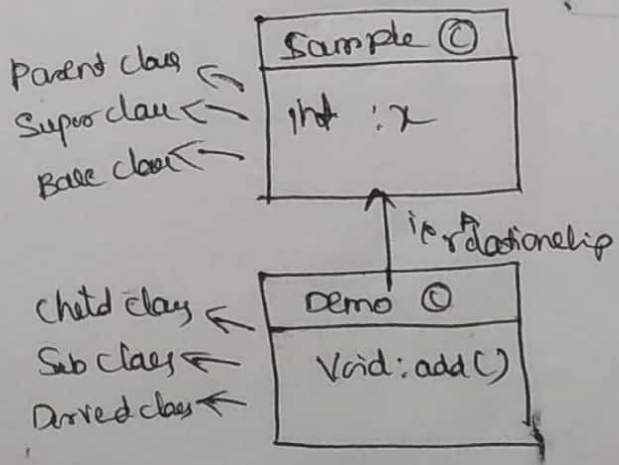




- Inheriting the properties from one class to another class is called as inheritance
- In inheritance we have 5 type
  - i) Single level inheritance
  - ii) Multi level inheritance
  - iii) Hierarchichal inheritance
  - iv) Multiple inheritance
  - v) Hybrid inheritance

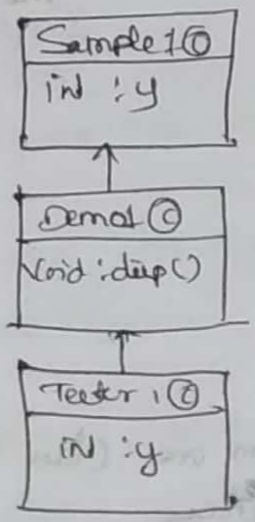
Single level inheritance

A Subclass inheriting the property from only one Super class is called as Single level inheritance



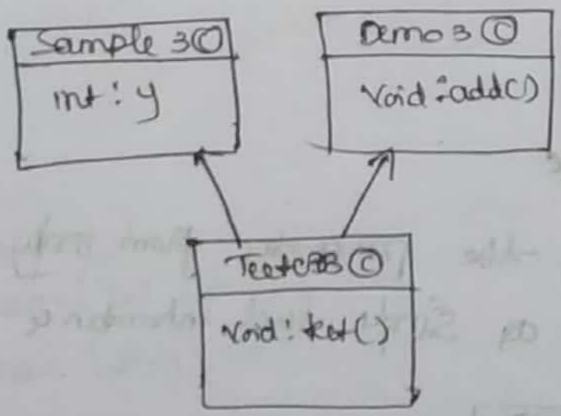
Multi level inheritance

A Subclass inheriting the properties from its Superclass, in turn Superclass inherits the properties from its Superclass - is called as multilevel inheritance



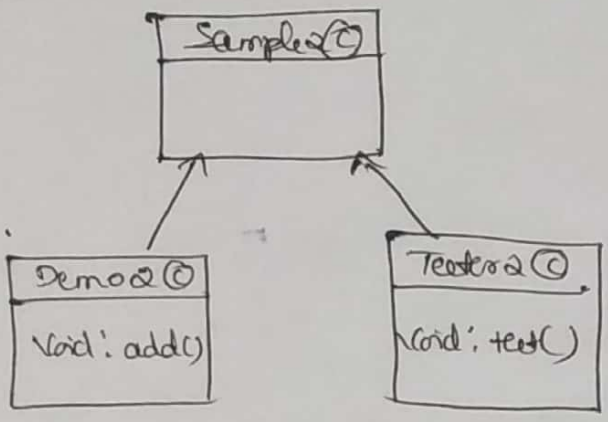
Multiple inheritance

A Subclass inheriting the properties from multiple Superclasses is called as multiple inheritance



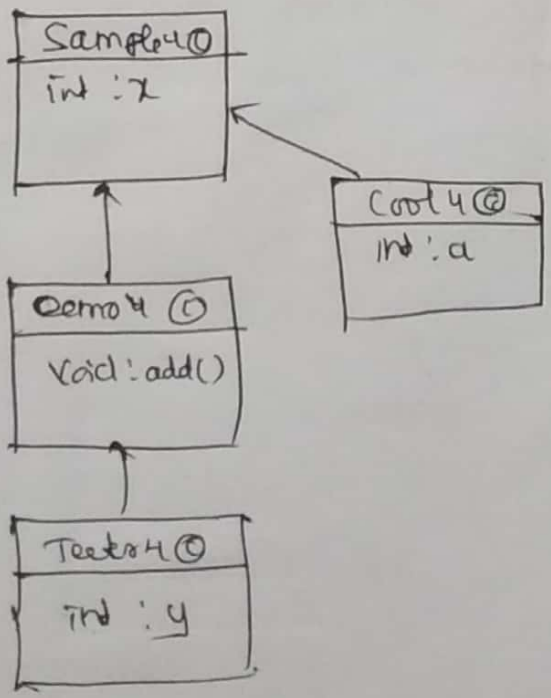
Hierarchical inheritance

Multiple SubClasses inheriting the properties from only one super class is Common Super class is called as Hierarchical inheritance



Hybrid inheritance

It is a Combination of single level, multi-level and hirarchical inheritance is called as hybrid inheritance.



Method overriding

Developing a method in the Sub class with the same name and signature as in the Super class with different implementation in the subclass is called as method overriding.

Rules

i) The method should be non-static
ii) there should be 'IS A' relationship
iii) the method name and signature should be same in the Sub class as in the Superclass

iv)

Why do we go for method overriding?

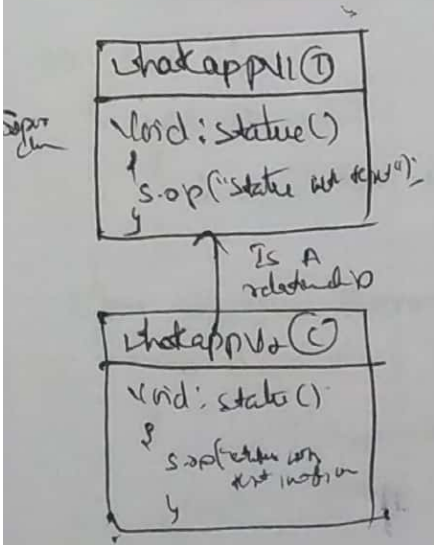
⇒ i) Whenever we want to provide new implementation for the old feature then we should go for method over-riding

Can we over-ride both static and non-static method?

⇒ No we cannot static method, we can over ride only non-static methods

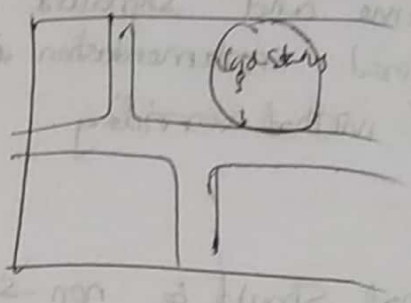
Can we over-load both static and non-static method?

⇒ Yes we can overload both static and non-static method.



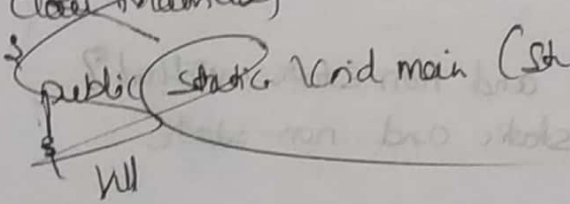
① Class Whatsapp V1

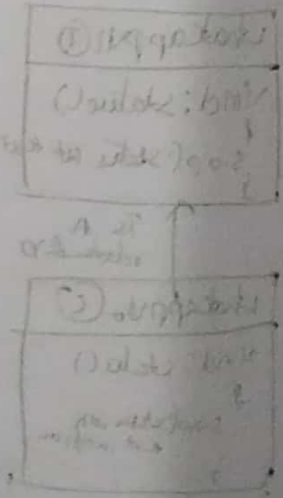
```
{
  void status()
  {
    S.op ("status with text");
  }
}
```



```
class Whatsapp V2 extends whatsapp V1
{
  void status()
  {
    S.op ("status with text, image & video");
  }
}
class mainclass
{
psvm (string[] args)
{
Whatsapp V2 Va = new Whatsapp V2()
Va.status();
}
}
```

② class ~~Mainclass~~





```
class Android_kitkat
{
    void Camera()
    {
        S.o.p("Back Camera...");
    }
}
class Android_lolipop extends Android_kitkat
{
    void camera()
    { S.o.p("front and back Camera...");
    }
}
class MainClass
{
    public static void main (String[] args)
    {
        Android lolipop1 = new Android();
        l1.Camera();
    }
}
```

Super

Super keyword is used only in the case of method over-riding, along with the sub-class implementation if we need Super class implementation then we should use `Super.method name()`

```
=> class Share it
   { void share()
     { S.o.p("share with bluetooth");
     }
   class shareit Extends Share it
   { void share()
     { S.o.p("share with QR code");
       Super.share();
     }
   }
```

Interview Questions

① What are Java features

i) It is simple language
ii) It is platform independent
iii) It is Compiled and interpreted
iv) It is robust
v) High performance
vi) Multi threaded

② Is Java object oriented programming language and why?

⇒ Yes Java is object oriented programming language because it Supports following Concepts.

i) Inheritance
ii) Polymorphism
iii) Abstraction
iv) Encapsulation
v) Class and object

and hence we call this Java as object oriented

③ Is Java 100% object oriented PL?

No, it is not 100% object oriented programming language because it Supports primitive data type.

④ Why main? In a program if we have SIB, IIB, Constructor and main method what is order of Execution

order
⇒ SIB
IIB
Constructor
Main method

What is the diff blw method overloading and overriding

Method Overloading	Method Overriding
i) Developing multiple methods with the same name but variation in arguments list is called as method overloading	i) Developing a method in subclass with same name & signature as in the superclass but different implementation in the subclass is called as Method method overriding
ii) We can overload both static and non-static method.	ii) We can override only non-static method.
iii) The method name should be same, there is no restriction on access specifier, modifier & return type etc	iii) The complete signature should be same in the subclass as in the superclass.
iv) Whenever we want to perform common task & operation then we should go for method overloading	iv) Whenever we want to provide new implementation then we should go for method overriding
v) In method overloading there is no rule that it should have 'IS A' relationship	v) Compulsorily there should be 'IS A' relationship
vi) Method overloading is an example for Compile time polymorphism	vi) method overriding is an example for run-time polymorphism.

Method	Constructor
i) Method is used to perform some operation.	i) It is used to initialize data members
ii) It is a block of statement which will get executed whenever it is called & invoked.	ii) Constructor will get executed whenever an object is created
iii) Method will have return type	iii) Constructor will not have return type
iv) Method can return some return some value	iv) Constructor cannot return any value
v) Method can be static & non-static	v) Constructor are always non-static
vi) Method should be developed explicitly	vi) In each and every class there will be a default constructor.
vii) Method name can be either class name or any name	vii) Constructor name should be same as class name
viii) Method can be inherited	viii) Constructor cannot be inherited
ix) Through methods we can do recursive calling	ix) Through constructor we cannot achieve recursive
x) We can declare method as final	x) We cannot declare constructor as final

What is the diff b/w this and Super keyword

This	Super
i) This keyword is used to point to the current object	i) 'Super' keyword In the case of method over-riding along with the sub-class implementation if we need the super class implementation then we will go for Super keyword.
ii) Whenever the local variable and global variable names are same, to differentiate b/w them we go for this keyword.	ii) ~~Whenever~~ In the case of method over-riding we go for Super keyword.
iii) This is a default reference variable	iii) Super keyword is not a default reference variable
iv) this keyword can be used only in the non-static context.	iv) Super keyword can be used only in the case of method over-riding.

What is the diff b/w Abstract class and Interface

Abstract Class	Interface
i) Any class which is declared with the keyword abstract is called as Abstract class	i) Interface is a java type which is by default abstract
ii) In abstract class we have 3 members	ii) In interface we have 2 members
iii) In abstract class we have constructors	iii) we dont have Constructs
iv) Variable in abstract class can be static and non-static	iv) Variables in interface is by default static and final
v) Methods can have access specifiers	v) Methods are by default public and abstract
vi) We can declare both abstract and concrete	vi) we can declare only abstract method
vii) we have to over-ride all the abstract methods in the sub-class but not in the concrete	vii) All the methods should be over-ridden

viii) Abstract class Extends Object class	viii) Through interface we Cannot achive Constructor Chain because it doesn't Support Constructor
ix) When we know ~~principal~~ partial implementation then we should go for abstraction class	ix) When we ~~don't~~ dont know 100% implementation then we should go for interface.

What is Constructor chaining?

→ A subclass Constructor calling its immediate Superclass Constructor, Superclass Constructor calling its immediate Super superclass Constructor is called as Constructor Chaining.

What is diff b/w Error and Exception:

Error	Exception
i) Error cannot be predicted	i) Exception can be predicted
ii) Errors can't occured due to the system configuration	ii) Exceptions are occured due to the mistakes done by the programmer
(ii) Error cannot be handled	iii) Exception can be handled.

Type Casting

Converting from one type to another type is called as type casting

2 types

i) Primitive type Casting

* Widening < Explicitly / Implicitly
* Narrowing — Explicitly

ii) class type Casting | Derived type Casting

* Upcasting < Explicitly / Implicitly
* Down Casting — Explicitly.

⇒ Primitive type Casting

Converting from one primitive data-type to another primitive data type is called as primitive type Casting.

1) Widening

- Converting from smaller primitive datatype to any of its bigger primitive data-type is called as widening.
- Widening can be done both implicitly and Explicitly

Implicitly

```
double x = 20;
S.o.p(x); o/p 20.0d
```

Explicitly

```
double x = (double) 20;
S.o.p(x) ; o/p 20.0d
```

```
float y = 40
S.o.p(y); o/p 40.0f
```

```
float y = (float) 40
S.o.p(y); o/p 40.0f
```

```
double z = 36.6f
S.o.p(z) o/p 36.6d
```

```
double z = (double) 36.6f
S.o.p(z) ; o/p 36.6d
```

ii) Narrowing

- Converting from bigger primitive data-type to any of its smaller primitive data-type is called as narrowing
- Narrowing should be done always Explicitly.

byte int shot long Float Double

```
int x = (int) 59.9d;
S.o.p(x) o/p 59
```

```
byte z = (byte) 30.38d;
S.o.p(z); o/p 30
```

```
shot y = (shot) 31.3f;
S.o.p(y); o/p 31
```

```
long x = (long) 66.6f;
S.o.p (x) o/p 66
```

Eg) class Sampler
{ public static void main (String[] arg)
{
double x=20; // widening
System.out.println(x);
int y = (int)20.56; // Narrowing
S.o.p(y);
}
}

o/p
20.0
20

Class typecasting / Derived type Casting

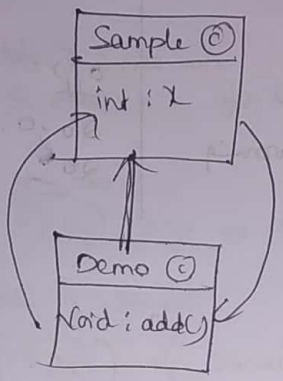
Converting from one class object to another class type is called as class typecasting / derived type Casting

i) Upcasting

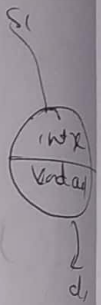
- Converting from Subclass object to Super class type is called as upcasting.
- Upcasting can be done both implicitly and Explicitly.

ii) Downcasting

- Converting from superclass object to subclass type is called as downcasting.
- Downcasting should be done always Explicitly

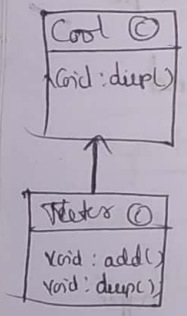



```
class mainclass
{
  psvm ( ——— )
  {
    Sample s1 = new Demo();
    S.o.p(s1.x);
    Demo d1 = (Demo) s1;
    S.o.p(d1.x)
    d1.add();
  }
}
```



Note:

- Without performing upcasting we cannot perform downcasting
- Direct downcasting is not possible




```
class Cool
{
  void disp()
  {
    S.o.p("Hi");
  }
}
class Tester
{
  void add()
  {
    S.o.p("Hello");
  }
}
```

```
class Main class
{
  psvm ( ——— )
  {
    Cool c1 = (Cool) new Tester();
    c1.disp(); //upcasting
    Tester t2 = (Tester) c1();
    t2.disp();
    t2.add(); // downcasting
  }
}
```

Scanned by CamScanner

Polymorphism

- An object showing different behaviour at different stages of its life cycle is called as polymorphism
- In polymorphism we have 2 types
  i) Compile time polymorphism.
  ii) Run-time polymorphism.
- poly means 'many', Mphism means 'forms'

Compile time polymorphism

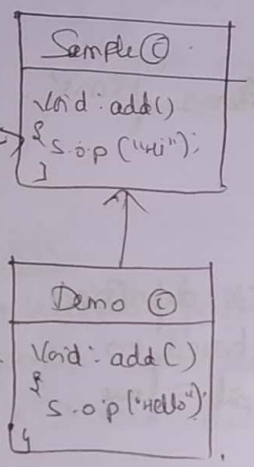
- The method declaration getting binded to its definition at the Compile time by the Compiler based on the arguments passed is called as Compile-time polymorphism.
- Once the method declaration get binded to its definition, it cannot be rebinded hence it is called as Static binding.
- Since the method declaration get binded to its definition at the Compile-time itself hence it is called as Early binding.
- Method overloading is an Example of Compile-time polymorphism.

Run-time polymorphism

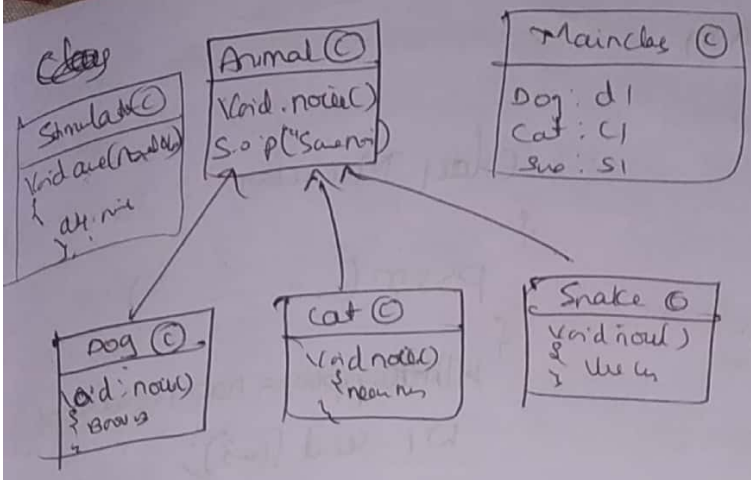
- The method declaration get binded to its definition at the run time by the JVM based on the object created is called as Run-time polymorphism.
- Once the method declaration get binded to its definition it can be rebinded hence it is called as dynamic binding

- Since the method declaration getting binded to it definition at the run time hence it is called as late binding
- Method over-riding is an Example of run-time polymorphism

Example of run time polymorphism

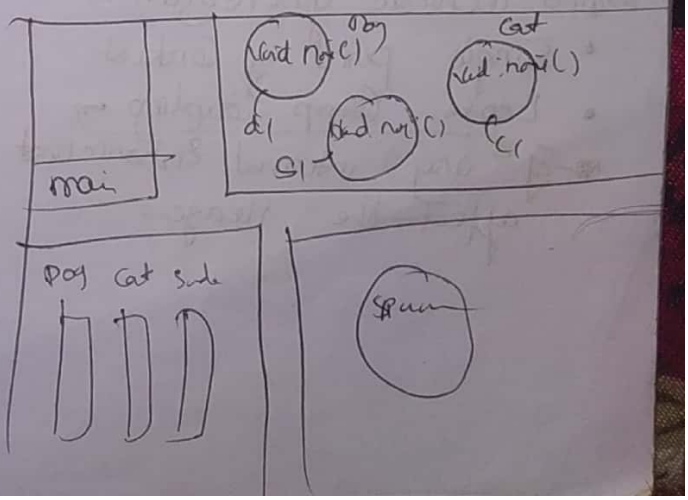



```
Class Sample
{
    void add()
    {
        s.o.p("Hi");
    }
}
Class Demo Extends Sample
{
    void add()
    {
        S o.p ("Hello");
    }
}
Class Mainclass
{
    public static void ma (String() arg)
    {
        Sample S1 = new Demo();
        S1.add();
    }
}
```




```
class Animal
{
  void noise()
  {
    S.o.p("some noise");
  }
}
class Dog Extends Animal
{
  void noise()
  {
    S.o.p("baow baow");
  }
}
class Cat Extends Animal
{
  void noise()
  {
    S.o.p("meow meow");
  }
}
class Snake Extends Animal
{
  void noise()
  {
    S.o.p("Bass hiss");
  }
}
class Stimulate
{
  Static void anim(Animal a4)
  {
    a4.noise();
  }
}
```

```
class Mainclass
{
  psvm(String[] args)
  {
    Dog d1 = new Dog();
    Cat c1 = new Cat();
    Snake s1 = new Snake();
    Stimulate.anim(d1);
    Stimulate.anim(c1);
    Stimulate.anim(s1);
  }
}
```



Compile-time polymorphism

```
class Whatsapp
{
  void send (int no)
  {
    S.o.p("sending no");
  }
  void send (String msg)
  {
    S.o.p("sending text msg");
  }
  void send(int no, String msg)
  {
    S.o.p("sending no & msg");
  }
  void send (String msg, int no)
  {
    S.o.p("send msg & no");
  }
}
```

```
class Mainclass
{
  psvm( ——— )
  {
    Whatsapp w1 = new Whatsapp();
    w1.send(123);
    w1.send("Hi");
    w1.send(126, "Cash");
    w1.send("Hi", 128);
  }
}
```

Note

- To achieve run time polymorphism we should have IS A relationship
  1. Method overriding
  2. Upcasting
  3. pass by reference

Q) Why do we go for run time polymorphism?

→ To achieve generalization

- Single point of Contact
- Loose ~~Comp~~ Coupling → If any internal Enhancement is done, it will not affect the usage.

① What is Method overloading with rules & Example
② What is method overriding with rules & Example
③ What is the diff blw static and Non-static
④ What is diff blw method overloading & overriding.
⑤ What is pass by reference with an Example
⑥ What is inheritance, Explain its types with an Exaple
⑦ What is type casting Example with an Exaple
⑧ What is polymorphism Exaple its types with Example.
⑨ All the above Questin 2 times.

Abstract Class

i) Concrete method :-
Any method which has both and definition is called as Concrete method.

Eg: Concrete method

```
void disp()
{
}
```

ii) Concrete class :- Any class which has only Concrete methods is called as Concrete class

Eg: Concrete class

```
class demo
{
    void disp()
    {
    }
}
```

iii) Abstract method :- Any method which is declared with the keyword abstract is called as abstract method.

Eg: abstract void cool();

iv) Abstract class :- Any class which is declared with the keyword abstract is called as abstract class.

Eg: abstract class Tester
{
}

v) If a class have an abstract method then the class should be declared as abstract, but Vice Versa is not true

Eg: abstract class Sample
{
abstract void disp();
}

vi) In an abstract class we can develop both concrete method and abstract method

Eg: abstract class Tester
{
abstract void cool();
void disp()
{ s.o.p("Hi");
}
}

**vii) We cannot Create Object for Abstract Class and Interface"

viii) We cannot declare abstract method as static, privat & final

ix) In abstract Class we will have Constructor

x) The class which provides the implementation for the abstract methods, the subclass is also called as implementation class.

xi) If any one of the abstract method is not over-ridden in the sub Class then the Subclass should be declared as Abstract

Eg: abstract class Demo1
{
abstract void disp();
abstract void test();
}
class Sample1 extends Demo1
{ void disp()
{ s.o.p("Hi");
}
void test()
{ s.o.p("Hello");
}
}

class Mainclass
{
public static void mai(String
{
Sample1 S1 = new Sample1
S1.disp();
S1.test();
}
}

Eg②

```
abstract class Testoo2
{
  abstract void cool();
  abstract void sam();
}
abstract class Samplee Extends Testoo2
{
  void cool()
  { s.o.p("Hello");
  }
  // abstract void sam() Virtually present //
}
class Demoo2 Extends Samplee
{
  void sam()
  { s.o.p("Hi");
  }
  public svm(——)
  {
    Demoo2 d2 = new Demoo2();
    d2.sam();
    d2.cool();
  }
}
```

Interface

i) Interface is a Java type
ii) Interface is a pure abstract body
iii) Interface is by default "abstract"
iv) In interface we have two members
  - Variable / data members
  - Methods / fun members
v) All the Variable in the interface are by default "static & final"
vi) All the methods in the interface are by default "public and abstract"
vii) Interface doesn't support Constructors
viii) Java provides a keyword called implements to inherit the properties from interface to class
ix) We cannot create object for Abstract class & interface

x) The class which provides the implementation for the abstract method, the subclass is also called as implementation class

xi) If any one of the abstract method is not Overriden in the subclass then the Subclass shall be declared abstract

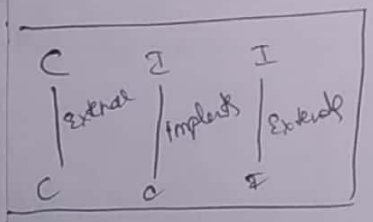
xii) From an interface to interface to inherit the properties we use the keyword 'Extends'

xiii) Each & Every class Extends object class, object class is the Supermost class in class type

xiv) Interface doesn't Extend any class, interface itself is the Supermost.

Syntax:

```
interface Sample
{
  Variable / data members
  method / fn mems
}
abstract interface Testco
{
  Static final int a=10;
  public abstract void disp();
}
```



① ~~class~~ interface

```
abstract interface Demo
{
public abstract void disp();
public abstract void cool();
}
class Sample implements Demo
{
  public void disp()
  {
    S.o.p("Hi")
  }
  public void cool()
  {
    S.o.p("Hello");
  }
}
class MainClass
{
  public static void main(String[]
  {
    Sample S1=new Sample();
    S1.disp();
    S1.cool();
```

```
② abstract interface Demo1
{
public abstract void test();
public abstract void sam();
}
abstract Class Test1 implements Demo1
{ public void test()
  { S.o.p("Hi");
  }
//public abstract void sam();
}
Class Sample1 Extends Test1
{
  public void sam()
  { S.o.p("Hello");
  }
  psvm (————————)
{
  Sample1 s1 = new Sample1();
    s1.test();
    s1.sam();
  }
}
```

```
③ interface Puma
{
  void shoes();
}
interface Nike Extends Puma
{
public abstract void bags();
public abstract void shoes()
}
Class Rajasthan implements Nike
{
  public void bags()
  {
    S.o.p("jingi jingi shoes");
  }
psvm (————————)
{
  Rajasthan r1 = new Rajasthan();
      r1.bags();
      r1.shoes();
  }
}
```

Abstract class and interface in development point of View

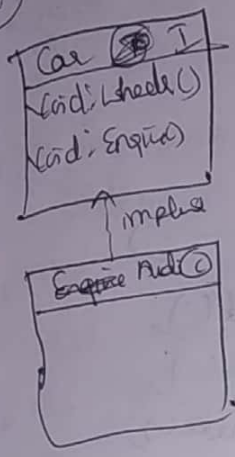
```
interface ChatboxRev
{
  void chat();
  void call();
  void ride();
  void color();
}
class Chatbox implements ChatboxRev
{
  E

  psvm(
  {
  }
}
```

```
Abstract class chatboxRev
{
abstract void chat();
abstract void call();
abstract void ride();
 void color()
   { s.o.p ("blue & white");
   }
class chatbox Extends chatboxRev
 {
   .
 }
}
```

Abstraction :-

- Hiding the Complexity of the System and Exposing only the required functionality to End user is called as abstraction
- To achieve abstraction declare all the Essential features in the interface and provide the implementation in the subclass
- Create a reference Variable of interface type and initialize that reference Variable with implementation class object. This is how we achieve abstraction.
- Through interface we can achieve 100% abstraction
- Through abstract class we can achieve upto 100%.

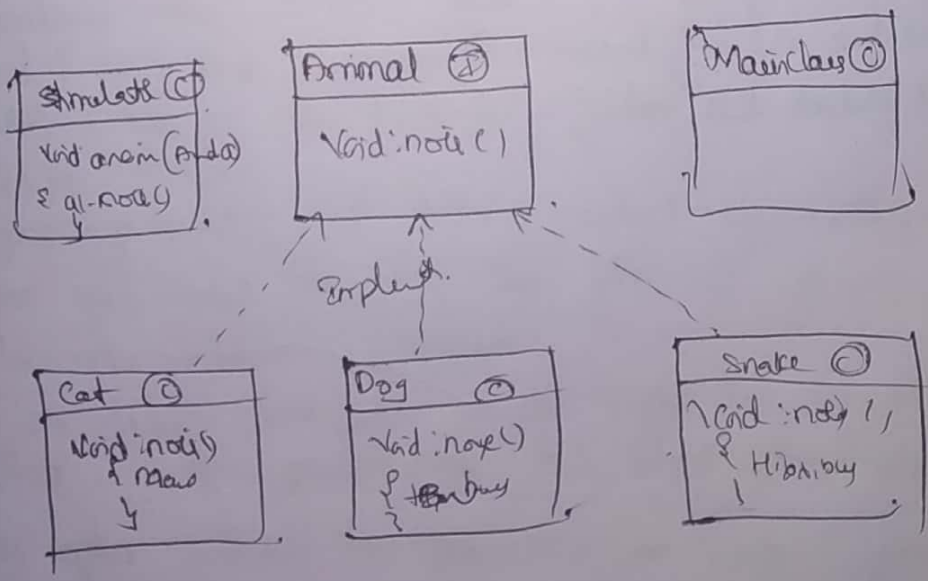
①




```
interface Car
{
    void wheels();
    void Engine();
}
class Audi implements Car
{
    void wheels()
    {
        S.o.p("Super wheels")
    }
    void Engine()
    {
        S.o.p("5000 CC");
    }
class Mainclass
{
    psvm (——————)
    {
        Car C1 = new Audi();
        C1.wheels();
        C1.Engine();
    }
}
```

→ Abstraction is achieved.

②



```
interface Animal
{
    void noise();
}
class Dog implements Animal
{
    S.o.p("Bau");
}
class Cat implements Animal
{
    S.o.p("meow");
}
class Snake implements Animal
{
    S.o.p("hiss");
}
class Simulator
{
    static void anim(Animal a1)
    {
        a1.noise();
    }
}
```

```
class MainClass
{
    psvm(
    {
        Cat c1 = new Cat();
        dog d1 = new dog();
        Snake s1 = new Snake();
        Simulator.anim(c1);
        Simulator.anim(d1);
        Simulator.anim(s1);
    }
}
```

- Java provides a keyword called import to import the files from one package to another package
- In any Java file the first statement should be package statement and it should be only one statement
- The second statement should be the import statement. We can have N no. of import statements.
- The 3rd can be any ~~any~~ java type

How to write a program in Eclipse

⇒ package naveen.English

Steps

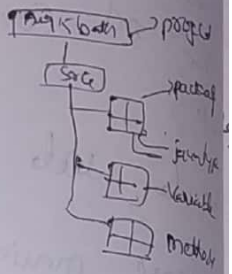
1. Download the Eclipse file
2. Right click on the downloaded zip file and Extract
3. Double click on the Extracted folder and double click on Eclipse icon.
4. A pop-up will come where select the workspace where you want to save the files.
5. Close the Welcome screen
6. In the Eclipse interface in the right top corner click on open perspective and select Java (default), ok
7. Once after selecting the perspective as java ~~project~~ package
8. Create a project → File — New - Java project - ~~filename~~ finish
9. Expand the project select Src
10. Create package → file - new - package - pkgname - finish

• Select the package file-new-class-classname-finish

Note:

• As soon as we write a program it will get compiled and dot class file will be generated, it will be stored in bin folder.

• To run the program click on play button.



Advantages of Eclipse

i) Easy to maintain the code
ii) It segregates .java & .class files
iii) Easy to identify defects.

Access Specifier

• It is used to restrict the access from one class to another class or from one package to another package

• In access specifier we have 4 types
  i) public
  ii) protected
  iii) default / package level
  iv) private

Public

• An member of the class declared with the keyword public is called as public access specifier.
• It can be accessed within the class
• It can be accessed within the package
• It can be accessed outside the package
• It can be accessed anywhere, application level

Protected

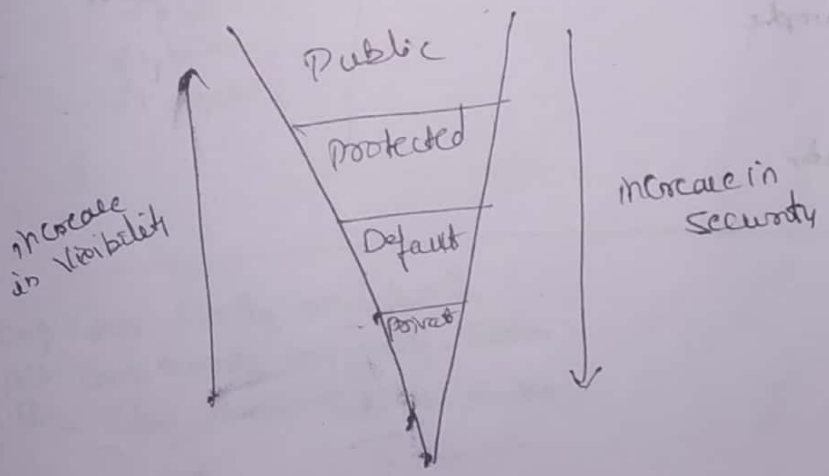
- Any member of the class declared with the keyword protected is called as protected member of the class.
- It can be accessed within the class
- within the package and outside the package with a relationship.

Default / package level

- Any member of the class, declared without any keyword as the access specifier is called as default access specifier
- It can be accessed within the class
- within the package.

Private

- Any member of the class, declared with the keyword private is called as private ~~members of the class~~ access specifier
- It can be accessed within the class



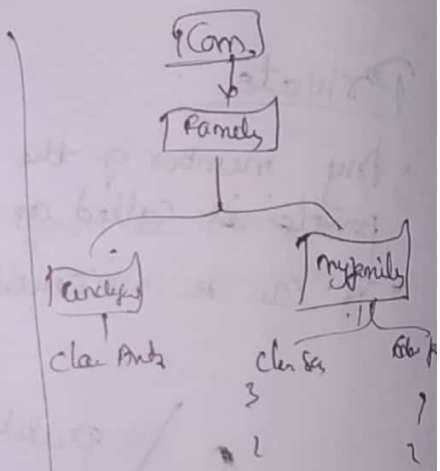
Note:-

- All the class members can have all four access specifiers
- The class can have only two access specifiers i.e public and default

public
default → class Demo { variable, method, Constructors → Public, private, protected, Constructors }

- If multiple classes are developed in a single Eclipse file the class which is declared as public only that class can have main method.

```
Public class Demo
{
    psvm(---)
    {
    }
}
class Sample
{
}
class Tester
{
}
```



```
package com.family.myfamily;
public class Father
{ private void atm()
  { s.o.p ("dad kaa atmaa");
  }
  void car()
  { s.o.p ("dad kaa caaa");
  }
  protected void bike()
  { s.o.p ("dad ka bike");
  }
  public void cycle()
  { s.o.p ("dad ka cycle");
  }
}
```

o/p

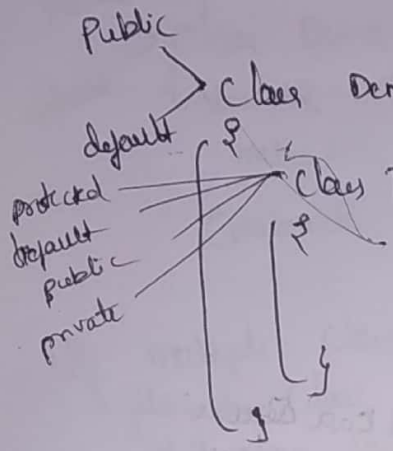
dad kaa caaa
dad kaa bike
dad ka cycle

```
package com.family.myfamily;
public class Son
{ psvm (String[] arg)
  { Father f1 = new Father();
    f1.car();
    f1.bike();
    f1.cycle();
  }
}
```

```
package com.family.unclefamily;
import com.family.myfamily.Father;
public class Aunty extends Father
{
  psvm (——— )
  { Aunty a1 = new Aunty();
    a1.bike();
    a1.cycle();
  }
}
```

Inner class

An class within a class is called as inner class, The inner class can have all 4 access specifiers, because it acts like the members of the Outer class



Final class

* Any class which is declared with the keyword final is called as final class
* Eg:- final class Demo
  {
  ==
  }

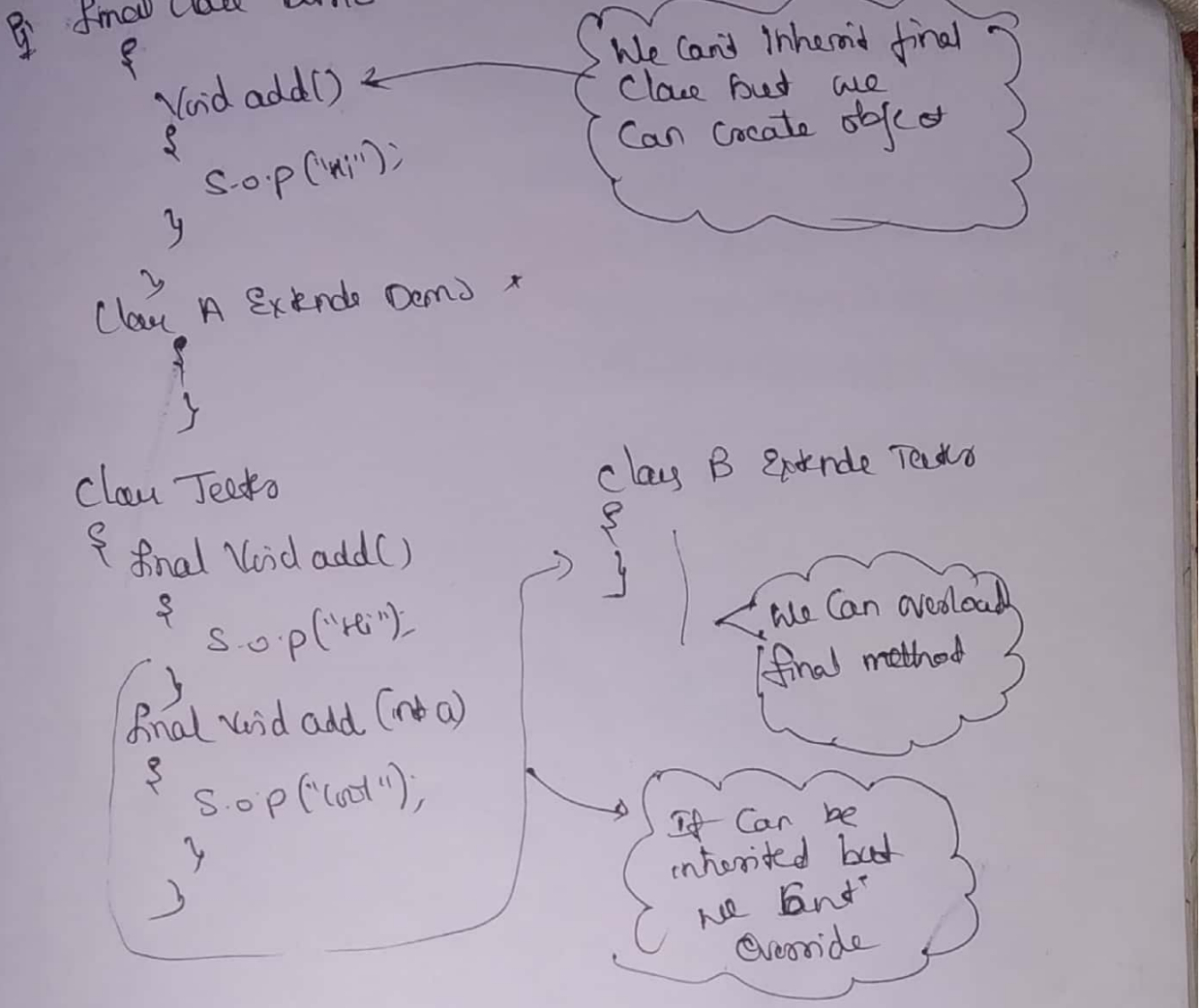
Final method

Any method which is declared with the keyword final is called as final method

Eg: final void disp()
{
==
}

Constructor cannot be declared as final.

Note:- 1) We can overload final method but we cannot override final method
2) We cannot inherit final class but we can create object of final class



Write a program to demonstrate String object without new operator will not allow duplicates and String object with new Operator

⇒
```
package Stringtopic;
public class Sample1
{
    public static void main (String[] args)
    {
        String s1="hii";
        String s2="hii";
        S.o.p (s1==s2);

        String s3=new String("hello");
        String s4=new String("hello");
        S.o.p (s3==s4);
    }
}
```

o/p
~~False~~ True
False

As a TE we should use Equals methods to Compare object values in string class.

⇒
```
public Stringtopic;
public class Sample1
{
    public static void main (String[] ay)
    {
        String s1=new String("hii");
        String s2=new String("hii");
        S.o.p (s1==s2);
        S.o.p (s1.Equals(s2));
    }
}
```

o/p = False
true

Diff b/w Comparison Operator and Equals ~~Operator~~ method

Comparison (==) Operator	Equals method
1) It will Compare object address	i) It will Compare object Values in String class
2) It is an operator	2) It is a non-static method
3) Comparison Operator Cannot be Overridden	3) Equals method Can be Overridden.

Object class

- Object class is a supermost class in java
- Object class belongs to java.lang package
- Each & Every package imports java.lang package by default.

In Object class we have the following methods

i) int hashCode()
ii) boolean Equals()
iii) Object clone()
iv) String toString()
v) Void notify()
vi) Void notify all()
vii) Void wait()
viii) Void finalize()

toString class

```
package String topic
public Class Sample1 Extends Object
{ public String toString()
   { return "hey it Sample1 obj";
   }
psvm(.   )
{ Sample1 S1=new Sample1();
   S.o.p (S1);
}}
```

publ. cla @...

public cla Sa Ext

{

}

PSV

{

Sa S1= n C

(S1)

- toString() is a non-static, non-final method of object class
- toString method will be inherited to each and every cl
- whenever we print a reference variable tostring metho will be invoked implicitly, which will return the fully Qualified path in the form of string
- Fully Qualified path means | Packagename.Classname@Hexadecim
- the signature of tostring method is public String toString

Hashcode()

⇒

```
packag stringtopic
Public Class Sample1
{
    public int hashCode()
    {
        return 123;
    }
Public static void main (String[] args)
{
    Sample S1 = new Sample1();
    System.out.println (S1.hashCode());
}
}
```

O/P: 123

```
S.hash())
S.o.p(S1)
```

7867.8

Hashcode()

- Hashcode() is a non-static, non final method of object class
- Hashcode() will be inherited to each & every class
- whenever we invoke the hashcode() it will return a unique integer number called hash number based on the object address.
- The hashnumber will be generated based on the hashing algorithm which is implemented internally.
- The Signature of hashcode method is public int hashcode

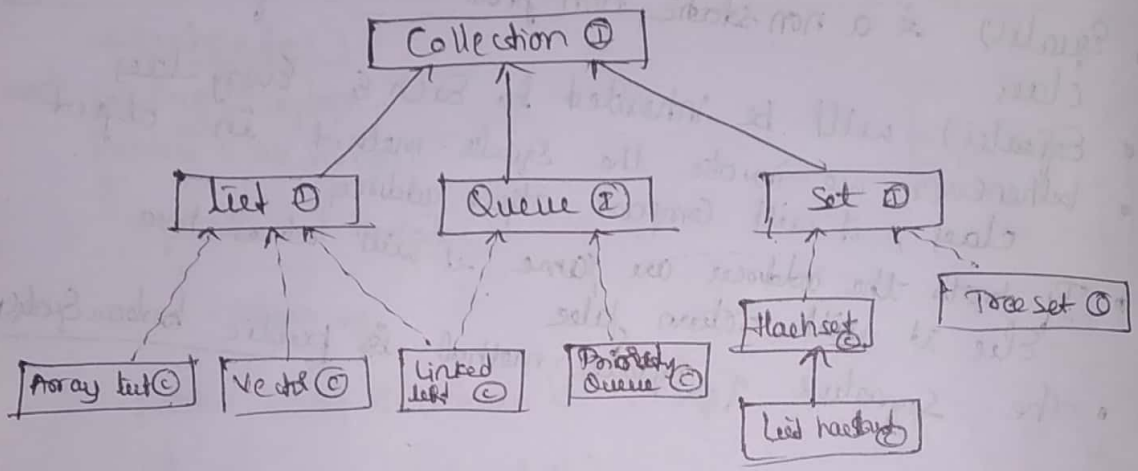
Equals()

```
package stringtopic;
public class Sample1
{
    public static void main (String[] args)
    {
        Sample1 s1=new Sample1();
        Sample1 s2=s1;
        System.out.println(s1.Equals(s2));
    }
}
```

- Equals() is a non-static, non-final method of object class
- Equals() will be inherited to Each & Every class
- Whenever we invoke the Equals method in object class, it will compare object addresses
- If both the addresses are same it will return true Else it will return false
- The signature of Equals method is public boolean Equals()

Collection

- Collection is an unified architecture which consists of classes and interfaces in order to overcome the drawback of array we will go for Collection.
- In Collection the size is dynamic and we can store heterogeneous type of data.
- All the collection classes and interfaces belongs to java.util package.



The default capacity in collection is 10 and it grows its size 50% with the below formula

$$\text{Capacity} = \text{Current Capacity} + \frac{3}{2} + 1$$

List

List is an interface it has 3 subclasses.

i) Arraylist
ii) Vector
iii) Linked list.

Array list

It is a class

```
package Collectiontopic
import java.util.ArrayList;
public class Sample18
{
    public static void main (String[] args)
    {
        ArrayList l1=new ArrayList();
        l1.add(10);
        l1.add(20.56);
        l1.add(10);
        l1.add('A');
        l1.add("hello");
        l1.add(null);
        S.o.p(l1);
    }
}
```

o/p

[10, 20.56, 10, A, hello, null]

WAP to add the element in a particular index.

```
public class Sample1
{
    public static void main (String[] args)
    {
        ArrayList l1=new ArrayList();
        l1.add(10);
        l1.add(20.56);
        l1.add(10);
        l1.add("hello");
        l1.add(null);
        l1.add(
        S.o.p("****b4****");
        S.o.p(l1);
        S.o.p("****a4****");
        l1.add(2, true);
        S.o.p(l1);
    }
}
```

→ | 10 | 20.56 | true | 10 | hello | null |

WAP to Copy form one Collection object to another Collection object.

```
=> public class Sample2
{
  psvm (String[] args)
  { ArrayList l1=new ArrayList();
    l1.add(10);
    l1.add(20);
    l1.add(30);

    ArrayList l2=new ArrayList();
    l2.add('A');
    l2.add('B');
    l2.add('C');

    s.o.p("**** b4 ****");      }   l1 = 1,2,3;
    s.o.p("l1 --->"+l1);        }   l2 = A,B,C;
    s.o.p("l2 --->"+l2);
    l1.add(l2)
    s.o.p("***a4***");          }   l1 = 1,2,3, A,B,C
    s.o.p("l1 ---->"+l1);       }   l2 = A,B,C
    s.o.p("l2 --->"+l1
```

=> Void addAll(int index, Collection)

```
=> psvm (    )
   {ArrayList l1=new ArrayList();
    l1.add(10);
    l1.add(20);
    l1.add(30);
    ArrayList l2=new ArrayList();
    l2.add('A');
    l2.add('B');
    l2.add('C');

    l1.addAll(1, l2);

    s.o.p(l1);
    s.o.p(l2);
```

Ex:

l1 = [10 | 20 | 30]
  0 1 2

added in 1st index

o/p

l1 = [1 | A | B | C | 10 | 20]

l2 = [A | B | C]

(63) WAP to remove the Element from the collection object.

```
=> psvm (String[] args)
{
    ArrayList l1 = new ArrayList();
    l1.add("goa");
    l1.add("banglde");
    l1.add("Mycode");
    l1.add("chennai");
    l1.add("salem");
    l1.remove(0); //remove by index value
    s.o.p(l1);
    l1.remove("salem"); //remove by value
    s.o.p(l1);
```

O/P
Banglde, Mycod, Chnai, salem
Banglde, mycod, chnai
&

(64) WAP to remove the duplicates in l1 wrt l2.

```
=> psvm (-->)
{
    ArrayList l1 = new ArrayList();
    l1.add(10);
    l1.add(20);
    l1.add(30);
    l1.add(40);
ArrayList l2 = new ArrayList();
    l2.add(30);
    l2.add(40);
    l2.add(50);
    l2.add(60);
    s.o.p(l1);
    s.o.p(l2);
    l1.removeAll(l2);
    s.o.p(l1);
    s.o.p(l2);
```

O/P

10	20	30	40
30	40	50	60
10	20		
30	40	50	60

removeAll will remove all the duplicate in l1 wrt l2

~~Remove All will remove all the duplicates~~

Retain All will retain all the duplicates in l_1 w.r.t l_2

~~⇒ psvm~~

(5) WAP to retain all duplicates l_1 w r t l_2

```
⇒ psvm ( ——— )
{
Arraylist l1 = new Arraylist();
l1.add(10);
l1.add(20);
l1.add(30);
l1.add(40);
Arraylist l2 = new Arraylist();
l2.add(30);
l2.add(40);
l2.add(50);
l2.add(60);
S.o.p(l1);
S.o.p(l2);
l1.retainAll(l2);
S.o.p(l1);
S.o.p(l2);
}
```

o/p

l_1 = [10 | 20 | 30 | 40]

l_2 = [30 | 40 | 50 | 60]

l_1 = [| | 30 | 40]

l_2 = [30 | 40 | 50 | 60]

(6) WAP to get the Elements one by one from collection

```
⇒ Arraylist l1 = new Arraylist();
l1.add(10);
l1.add(20);
l1.add(30);
l1.add(40);
Object o1 = l1.get(0);
S.o.p(o1);
Object o2 = l1.get(1);
S.o.p(o2);
Object o3 = l1.get(2);
S.o.p(o3);
```

```
for (int i=0; i<l1.size(); i++)
{
Object o1 = l1.get(i);
S.o.p(o1);
}
```

o/p

10
20
30
40

String	Array	Collection
String S1 = "Hello" S1.length()	int[] arr = {10,20,30}; arr.length	Arraylist l1 = new Array l1.add(10); S.o.p(l1.size());

Generic type Collection

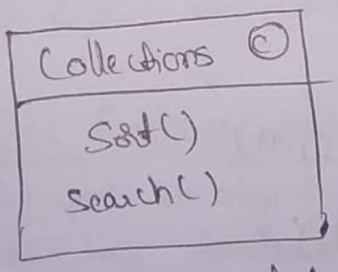
To achieve generic type of Collection we should use angular braces <>. Where we can store only homogenous type of data in one collection object

① WAP to demonstrate generic type of Collection.

```
psvm (————)
{ ArrayList<String> l = new ArrayList<String>();
  l.add("hello");
  l.add("hiii");
  l.add("bye");
  l.add("cool");
  S.o.p(l);
  for(int i=0; i<l.size(); i++)
  { Object o1=l.get(i);
    S.o.p(o1);
  }
}
```

o/p
hello
hiii
bye
cool

② WAP to sort the Collection objects



Collections is a class which belongs to java.util package and it has static methods like Sort(), search()

Sort() → sorting
Contain() → it will check for the content,

Contains()

Contains is a non static method of Arraylist class which checks the Element is present in the given Collection object, if it is present it will return true else it will return false

Eg:-

```
public class Sample
{
psvm(——)
{
ArrayList<String> l1=new ArrayList<String>();
l1.add("hello");
l1.add("cool");
l1.add("hii");
l1.add("Child");
Collections.sort(l1);
if (l1.contains("Child"))
{
  S.o.p("im Child");
}
else
{
  S.o.p("full child");
}
for(int i=0; i<l1.size(); i++)
{
  Object o1=l1.get(i);
  S.o.p(o1);
}
}
```

```
psvm(---)
{
ArrayList<String> l1 = new ArrayList<String>();
l1.add("Apple");
l1.add("Mango");
l1.add("Guava");
l1.add("Kiwi");

ArrayList<String> l2 = new ArrayList<String>();
l2.add("Apple");
l2.add("Mango");
l2.add("Chikku");
l2.add("Grapes");
for(int i=0; i<l1.size(); i++)
{
  if(l2.Contains(l1.get(i)))
  {
    S.o.p(l1.get(i));
  }
}
}
```

o/p
Apple
Mango
other than this
o/p
Guava
Kiwi

- In ArrayList class the Constructors are Overloaded as below

i) ArrayList Constructor with no parameter [ArrayList()]
→ Whenever an object is created it will invoke this Constructor with the default Capacity as 10.
~~→ ArrayList() with~~

ii) ~~ArrayList~~ (int initialCapacity)
→ This Constructor will be invoked with the User defined Capacity.

iii) ArrayList(Collection c)
→ It helps to Copy one Collection object to another Collection object

Prog to Copy from one Collection object to another without using addAll()

```
→psvm( ——— )
{
  ArrayList l1 = new ArrayList();
   l1.add("Apple");
   l1.add("mango");
   l1.add("Guava");
   l1.add("Kiwi");
  Sop(l1);
 ArrayList l2 = new ArrayList(l1);
   l2.add(10);
   l2.add(20);
  Sop(l2)
```

o/p

l1 = [Apple, mango, guava, kiwi]

l2 = [Apple, mango, guava, kiwi, 10, 20]

Vector

- Vector is a class which implements List interface

Prog on Vector

```
→psvm(String[] arg)
{ Vector l1 = new Vector(3);
  Sop("*** by Capacity ***" + l1.Capacity());
  l1.add("Apple");
  l1.add("mango");
  l1.add("Guava");
  l1.add("Kiwi");
  Sop("size is " + l1.size());
    Sop("*** ay Capa*** + l1.Capacity());
}
}
```

op

*** by Capacity 3
size is 4
*** ay Capacity 6

Queue

Queue is an interface which extends Collection interface and it implements 2 subclasses
i) Linked list
ii) priority Queue

Priority Queue

It is a class which implements Queue interface in priority Queue we have 2 methods to fetch the data ~~i) poll()~~

i) poll()
it will fetch the top most element of the Queue and reduce the size of the Queue by 1.

ii) peek()
It will fetch the top most element of the Queue and it will not reduce the size of the Queue by 1.

→ It is Autosorted with partial Output.

```
⇒ import java.util.priorityQueue;
  Public class Sample1
  {
     psvm (String[] args)
     {
        priorityQueue l1 = new PriorityQueue();
        l1.add(10);
        l1.add(28);
        l1.add(2);
        l1.add(5);
        S.o.p(l1);
        S.o.p("*** polling ***");
        S.o.p(l1.poll());
        S.o.p(l1);
        S.o.p("*** peeking ***");
        S.o.p(l1.peek());
        S.o.p(l1);
```

o/p
[5 10 28 2]
*** polling ***
5
[10 28 2]
*** peeking ***
10
[10 28 2]

Linked list

- It is a class which implements list and Queue interface
- Whenever we want proper order of insertion then we should go for linked list

```
import java.util.LinkedList;
public class Sample1
{
    psvm (String[] arg)
    {
        LinkedList ll=new LinkedList();
        ll.add(10);
        ll.add(10);
        ll.add(2.56);
        ll.add('A');
        ll.add(true);
        ll.add(null);
        S.o.p (ll);
        S.o.p (ll.get(0));
        S.o.p (ll);
        S.o.p (ll.poll());
        S.o.p (ll);
    }
}
```

o/p

[10 10 2.56 A true null]
10 //get(i)
[10 10 2.56 A true null]
10 //poll
[10 2.56 A true null]

Set

Set is an interface it has 3 subclasses

i) hashset()
ii) Linked hashset
iii) Tree set

⇒ import java.util.Hashset;
psvm (———)
{ Hashset l1 = new Hashset();
l1.add(10);
l1.add(10);
l1.add(2.56);
l1.add('A');
l1.add(true);
l1.add(null);
s.o.p(l1);
}

o/p
[2.56, null, A, 10, true]

⇒ import java.util.LinkedHashset;
psvm (———)
{ LinkedHashset l1 = new LinkedHashset();
l1.add(10);
l1.add(10);
l1.add(2.56);
l1.add('A');
l1.add(true);
l1.add(null)
s.o.p(l1);
}

o/p
[10, 2.56, A, true, null]

⇒ import java.util.Treeset;
psvm(———)
{ Treeset l1 = new Treeset();
l1.add(10);
l1.add(10);
l1.add(50);
l1.add(55);
l1.add(25);
l1.add(5);
s.o.p(l1);
}
}

o/p
[5, 10, 25, 50, 55]

For Each loop

- It is the loop it fetches the element one by one not upon the index

Syntax

for (datatype var name : arrayname / Collection type var)

Eg: ① char[] ar = {'A', 'B', 'C'};

```
for (char a1 : ar)
{ s.op (a1);
}
```

② LinkedHashSet l1 = new Linked();

```
l1.add(10);
l1.add(20.5);
l1.add('A');
for (object o1 : l1)
{ s.op (o1);
}
```

③ psvm(———)

```
{ TreeSet l1 = new TreeSet();
  l1.add(30);
  l1.add(5);
  l1.add(2);
  l1.add(40);
  for (object o1 : l1)
  { s.op(o1);
  }
}
```

	List			Queue	Set		
	ArrayList	Vector	LinkedList	Priority Queue	HashSet	LinkedHashSet	TreeSet
indexed	Yes	Yes	Yes	No	No	No	No
Size is dynamic	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Heterogen data	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Allows duplicate null	Yes	Yes	Yes	No	Yes	Yes	No
Allows duplicate	Yes	Yes	Yes	Yes	No	No	No
Order of insertion	Yes	Yes	Yes	No (auto sorted)	No	Yes	auto sorted
Sorted automatically	No	No	No	Yes (partially)	No	No	Yes (fully)

Wrapper class

- It is a in built class which belong to java lang package. For Each and Every primitive data type we have a Corresponding class called wrapper class
- It performs a mechanism called boxing and Unboxing

Boxing

- Converting from primitive data type to Corresponding Wrapper class object is called as boxing

Unboxing

Converting from wrapper class object to the Corresponding primitive data type is called as unboxing.

- Both boxing and unboxing happens implicitly

Primitive	Wrapper Class
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
char	Character
boolean	Boolean

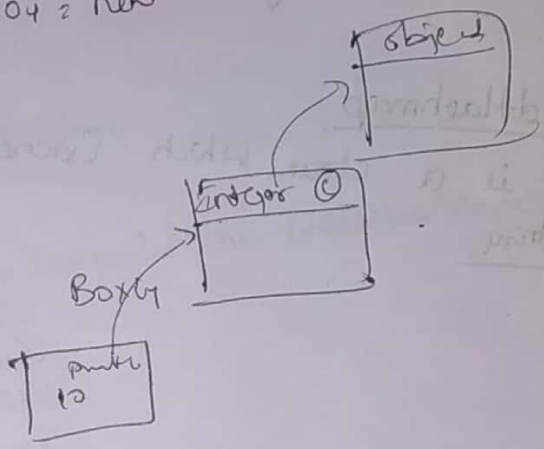
Eg:

```
Integer a1 = new Integer(10);
S.o.p(a1); //boxing o/p 10

int x = a1;
S.o.p(x); // o/p 10

Character c1 = new Character('A');
S.o.p(c1); o/p A

char c2 = c1;
S.o.p(c2); o/p A

Double d1 = new Double(20.16);
S.o.p(d1); o/p = 20.16

double x1 = d1;
S.o.p(x1); o/p 20.16
```

How the Elements are stored in the form of object in Collection

The primitive datatype will be boxed and then upcasted to the object class this is how the elements will be stored in the form of object.

```
ArrayList l1=new ArrayList();
l1.add(10);
l1.add(20.56); -> object o2= new Double(20.56);
l1.add('A'); -> object o3= new Character('A');
l1.add(true); -> object o4 = new Boolean(true);
Sop(l1);
```



Map is an interface which belongs to java.util package

- Map is used to store the element upon key & Value

Features of Map

- It is generic type
- It store upon key & Value
- It doesn't allow duplicate key
- It allows duplicate Value
- It has implemented 3 subclasses
  - i) HashMap
  - ii) LinkedHashMap
  - iii) TreeMap

HashMap

- HashMap is a class implements Map interface

Features

1)

vi) It will not follow order of insertion.

LinkedHashmap

It is a class which Extends Hashmap

Features

1)
2)
3)
4)
5)
6) It will follow order of insertion.

Treemap

It is a class which implements Map interface

Featur

1)
2)
3)
4)
5)
6) It is Autosorted upon keys

import java.util.HashMap;

{ psvm (——)

{ HashMap<String, Integer> l1 = new HashMap<String, Integer>();

l1.put ("Sneha", 123);

l1.put ("puneha", 125);

l1.put ("Sonehu", 452);

l1.put ("nehakka", 145);

l1.put ("appakaa", 141);

l1.put ("Sneha", 758);

S.o.p (l1);

}

}

o/p

⇒ import java.util.LinkedHashMap;

{ psvm (——)

{ LinkedHashMap<String, Integer> l1 = new LinkedHashMap<String, Integer>();

l1.put ("sneha", 123);

l1.put ("puneha", 125);

l1.put ("Sonehu", 452);

l1.put ("nehakka", 145);

l1.put ("appaka", 141);

l1.put ("sneha", 758);

S.o.p (l1);

}

}

⇒ import java.util.TreeMap;

{ TreeMap<String, Integer> l1 = new TreeMap<String, Integer>();

l1.put ("sneha", 123);

l1.put ("puneha", 125)

l1.put ("Sonehu", 452);

l1.put ("nehakka", 145);

l1.put ("appakaa", 145);

l1.put ("Sneha", 758);

S.o.p(l1);

}

}

o/p

appakka=145, nehakka=145, puneha=125, Sneha=758, Sonehu=452

What is Collection?

- Collection is an unified architecture which consist of all the classes and interfaces
- In order to overcome the drawback of array we will go for Collection.

- In Collection the size is dynamic
- It can store heterogeneous type of data.
- In Collection they have implemented a growable data structure which increases its size by 50%.
- Collection is an interface which belongs to java.util.p which should be imported Explicitly
- The Collection interface has 3 sub interfaces
  1) List
  ii) Queue
  iii) Set

List

- It is an interface which extends Collection interface then we will go for list type of Collection.
- Whenever we want to store upon index and allow duplicates then we should go for list type of Collection.
- List interface have 3 implementation classes
  i) ArrayList
  ii) Vector
  iii) LinkedList

Features of List

i) The size is dynamic
ii) It can store heterogeneous type of data
iii) It is indexed type
iv) It allows duplicates
v) Since the elements are stored upon index we can do random access
vi) It allows null.

ArrayList

- ArrayList is a class which implements List interface

Features of ArrayList

1.
2.
3.
4.
5.
6.
7. It increase its size by 50%
8. It is not Synchronized
9. Since it is not Synchronized the performance is fast

Vector

- Vector is a class which implements List interface

Features

1.
2.
3.
4.
5.
6.
7. It increase its size by 100%.
8. It is Synchronized
9. As it is Synchronized the performance is slow.

LinkedList

- ~~Linked list is a class which implements List~~
- Whenever we want proper order of insertion we should go for Linked List & data will be stored in random.

Features

1.
2.
3.
4.
5.
6.
7.
8. It implements both List & Queue interface, hence it can utilize both list & queue properties.

Queue

It is an interface which Extends Collection interface

When do we go for Queue type of Collection ?

- Whenever we want to maintain general Queue then we will go for Queue type of Collection.
- Queue interface have 2 implementing classes
  i) LinkedList
  ii) Priority Queue.

Priority Queue

- It is a class which is pure Queue type

Features of priority Queue

i) Size is dynamic
ii) Store heterogeneous type of data but homogenous for one object
iii) It is not indexed
iv) It allows duplicates
v) Since it is not stored upon index we cannot do random access.
vi) We cannot store null.
+++ vii) It is Auto sorted. (partially)

Set

Set is an interface which Extends Collection interface

When do we go for set type of collection ?

- When Ever we want to store the Elements without duplicates and not upon index then we should go for Set

- Set interface implements 3 subclass
  i) HashSet
  ii) LinkedHashSet
  iii) TreeSet.

Hashset

Features of Set

- Size is dynamic
- We can store heterogeneous type of data.
- It is not ordered
- It will not allow duplicates
- It allows null

Hashset: It is a class which implements Set interface

1.
2.
3.
4.
5.
6. It will not follow order of insertion.

LinkedHashSet

It is a class which extends Hashset.

1.
2.
3.
4.
5.
6. It will follow order of insertion

TreeSet: It is a class which implements Set interface

1.
2.
3.
4.
5. It doesn't allow null
6. It is Autosorted (fully).

How to write java prog in eclipse.

1. Once after selecting perspective to as java
2. Create a project

File → New → java project → project name → finish

3. Expand the project select "src"

File → New → package → package name → finish

4. Select the package

File → New → class → class name → finish

Note :- As soon as we write the prog it will get compile and .class file will be generated, it will be stored in bin folder.

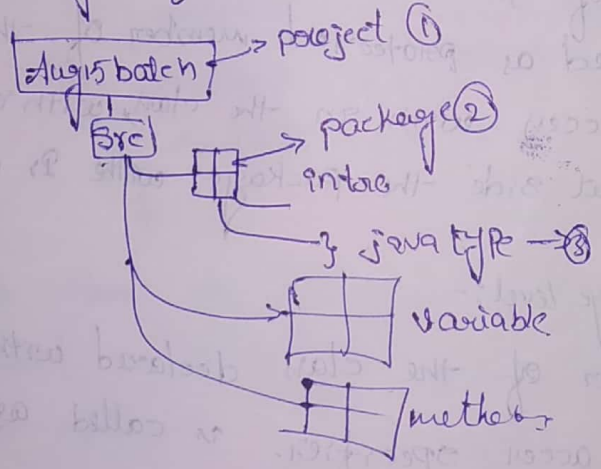
→ To Run the prog click on play button. you will get the out put

Advantages of eclipse

→ Easy to maintain the code.

→ It aggregates .class & .java files.

→ Easy identify defects.



Access specifier:

Access specifier is used to restrict the access from one class to another class (or) from one package to another package.

→ In Access specifier we have 4 types

1. public.
2. protected
3. Default / package level.
4. private

public :-

any member of the class declared with the keyword public is called as public access specifier.

→ It can be access with in the class
→ It can be access with in the package.
→ It can be access the outside the package.
→ It can be access any where, application level.

→ protected:

Any member of the class declared with the keyword protected is called as protected member of the class

→ It can be access with in the class, with in the package & Out side the package; with the IS a relationship

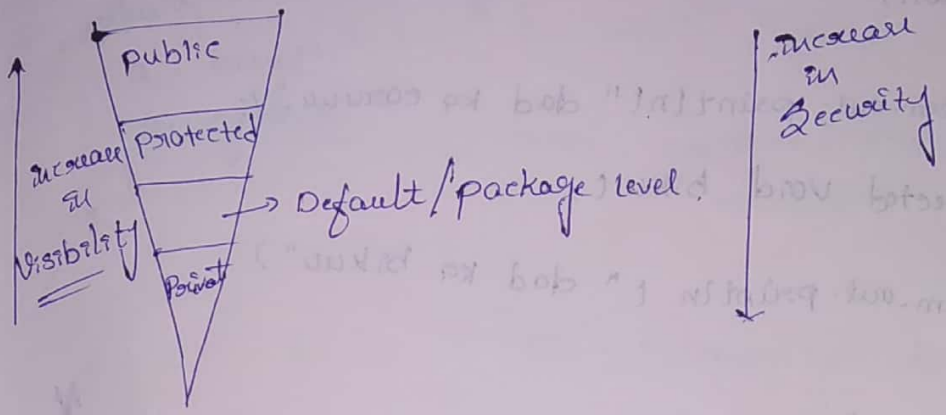
Default (or) package level:-

Any member of the class declared with any keyword. for the access specifier. is called as default access specifier.

→ It can be access with in the class
→ with in the package.

private :-
-> Any member of the class declared with the keyword private is called as private access specifier.
-> It can be accessed only with in the class

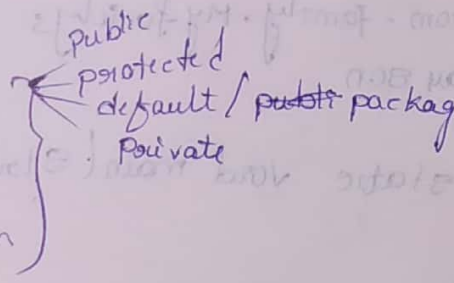
Note :-



Note :- All the class members can have all four access specifiers

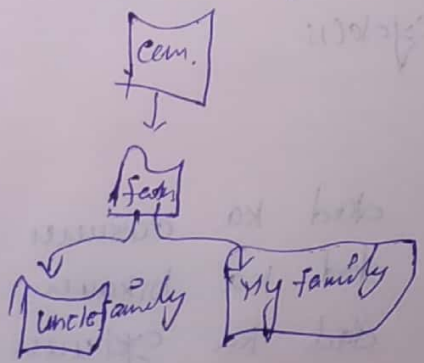
-> A class can have only two access specifier that is public & default

Ex :- public / default } class Demo
{
variable
Methe {
constructor
}



Note :-
-> if multiple classes are developed in single eclipse file the class which is declared as public. Only that class can have main method.

class Demo.
{
}
public class Sample.
{
psvm (-)
{
}
}
open testing
{



```
package com.family.MyFamily
public class father
{
private void atm()
{
System.out.println(" dad ka atmuuuu");
}
void car()
{
System.out.println(" dad ka caruuu ");
}
protected void biku();
{
system.out.println (" dad ka bikuu" )
}
public void cycle();
{
System.out.println(" dad ka cycluuu");
}
}
```

```
package com.family.MyFamily;
public class son
{
public static void main(String[] args)
{
father f1 = new father();
f1.car();
f1.biku();
f1.cycle();
}
}
```

Output:- dad ka caruuu
dad ka bikuuu
dad ka cycluuu

```
package com.family.uncle.family
import com.family.myfamily.father;
public class aunty extends father
{
public static void main(String[] args)
{
Aunty a1 = new Aunty();
a1.biku();
a1.cycle();
}
}
```

⇒ What is method overloading

③ ——— ∪ ——— Riding

③ What is polymorphism.

④ Inheritance

⑤ ~~Abstraction~~

⑥ Encapsulation

⑦ Collection :- Collection is an unified architecture which consist of classes and interfaces

WAP to check how many vowels are there in a String "~~Javanova~~".

```
⇒ class Demo
{
public static void main(String[] arg)
{
  int Count = 0;
  String S1 = "develop";
  for(i=0; i<7; i++)
  {
    ch = S1.charAt(i)
    if((ch=='a') || (ch=='e') || (ch=='i') || (ch=='o')
                 || (ch=='u'))
    {
      Count++;
    }
  }
}
```

WAP for bubble sort.

```
=> class Demo
{
  public static void main (String[] ay)
  {
    int [] arr = {10, 70, 80, 5, 9}

    for (i=0; i<arr.length; i++)
    {
      if (a[i]
      for (j=0; j<arr.length; j++)
      {
        if (a[j] < a[j-1])
        {
          temp = a[j];
          a[j] = a[j-1];
          a[j-1] = temp;
        }
        S.o.p (a[j] + " ");
      } for (i=0; i<arr.length; i++
      {
        S.o.p (arr[i] + " ");
      }
    }
  }
```

Broadridge

Quant - 20

Verbal - 20

Logical - 10

Puzzle - 1

Essay - 1 → First day of your college
Compare b/w school life & college life

Life is full of strange Experiences good and bad, happy and sad. Some of these Experience are just unforgettable. I can never forget my first day of college. It was a day of joy and excitement. But it turned to be a bad day. As i was new to Bangalore far from my parents i was nervous as well.

As soon as i entered the college gate i did not know what to do & where to go, i went to a senior and asked her the way to first year EC classroom. it was in the 3rd floor, i went near the classroom and i saw that Everyone are already present in the classroom. i was so scared to get in because i didn't know anyone there but i met a girl near the door and i got to know that she is from same hostel. we went inside and sat together in front bench.

As the lecture started i was only thinking about my home and my friends, as the class Ended i got to know that i have to wait till 4:30pm for college bus and i didn't knew the route by bus. As i had no friends over there i felt like crying being alone.

But one good thing happen because of my first day is the girl i met at the door is one of my closest friend now overall it was a good Experience getting in a unknown place between strange people and by the years passed those unknown people became very good friends

- Swap 2 numbers without using 3rd Variable.
- Swap _____ using 3rd Variable
- WAP to check given string is palindrome & not
- WAP to check given no is prime & not
- WAP to

(1) What is inheritance

↦ Inheriting the properties from one class to another class is called as inheritance

In inheritance we have diff types

i) Single level inheritance

↦ Subclass inheriting the properties from only one superclass

ii) Multilevel inheritance

→ Subclass inheritance properties from one Superclass & inturn Superclass inherits from its superclass

iii) Multiple inheritance

→ Subclass inheriting the properties from multiple superclass
- This is not possible in java through class because of diamond problem
iv) Hierarchical — JVM cannot develop & supercalling starts in the subclass constructor

Ex: multiple subclass inherits properties from one superclass

v) Hybrid

Combination of Single, multilevel & hierarchical.

(2) Why java is not 100% object Oriented programming language

→ Java supports primitive datatype hence java is not Oop language.

(3) What is method Overloading

Developing multiple methods with the same name but variation in argument list

(4) Method Overriding

Developing a method in the subclass with same name and signature as in the superclass but diff implementation in subclass.

⑤ Encapsulation

- It is one of the oops principle in Java, Java is by default Encapsulated
- Declare the data member as private and restrict the direct access outside the class and provide indirect access through public setters getters() & setters() is called as Encapsulation.

⑥ Polymorphism

Object showing diff behaviour in the diff stages of its life cycle is called polymorphism

poly means many

Morphism - shapes

2 type

i) Compile time polymorphism
ii) Run-time polymorphism.

Why java is oop langue?

⇒ Because it support the following Concepts

i) Inheritance
ii) Encapsulation
iii) polymorphism
iv) Abstraction

What is Abstraction

→ Hiding the Complexity of the System & Exposing only required functionality to the End user.

What is Exception handling?

⇒ The abnormal systems occurs in the program which terminates the programs Execution, handling those abnormal statement by using try & catch is called as Exception handling.

```
psvm (————————)
{
S.o.p ("Main start");
~~int i = 1/0~~
try {
    int i = 1/0;
  }
catch (~~Arithmetic~~ Exception e)
  {
    S.o.p ("handled");
  }
}
```

WAP to store user defined objects in collection.

```
⇒Class Mobile
{
  void call()
  {
     s.o.p("miss call ---");
  }
}
public class Sample1
{
  psvm (———)
  {
    ArrayList l1 = new ArrayList();
    l1.add(10);
    l1.add(new Mobile());
    l1.add(20.56);
    l1.add(
    mobile m1 = (mobile) l1.get(1);
    m1.call();
  }
}
```

O/P
miss call ---

WAP to check the given No is prime or not.

```
class Demo
{
  psvm(--)
  {
    int num=5;
    int copy=num;
    boolean flag=true;
    for(int i=2;i<5;i++)
    {
      if(num%i==0)
      {
        flag=false;
      }
    }
    sop(
    if(flag==true);
    {
      s.op("It is a palindrome");
    }
    else
    {
      s.op("Not a palindrome");
    }
  }
}
```

Encapsulation

- It is one of the Oops principle in Java.
- Java is by default Encapsulated
- We cannot declare any variable outside the class
- We cannot have any print statement outside the method
- Declare the data member as private and Restrict the direct access outside the class and provide indirect access through public services called getters() and setters() is called as Encapsulation.
- get is used to get the value, set is used to set the value
- It need not be get & set, it is an industrial convention to used get & set.

→
```
class Demo
{
private int x=10;
public int getx()
{
    return x;
}
public void setx()
{
    this.x=x;
}
}
```

```
class Mainclass
{
psvm(------)
{
Demo d1=new Demo();
S.o.p(d1.getx());
d1.setx(80);
S.o.p(d1.getx());
}
}
```

① package Collectionstopic;

class Mobile

{ private int pin=4561;
public int getpin()
return pin;

}
public void setpin (int pin)
{ this.pin = pin;

}

}
public class Sample1

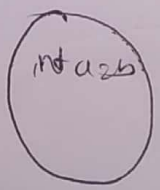
{
psvm (——)
{
Mobile m1 = new Mobile()
S.o.p (m1. getpin());
m1. setpin (1267);
S.o.p (m1. getpin());

}
}

Singleton Design pattern:

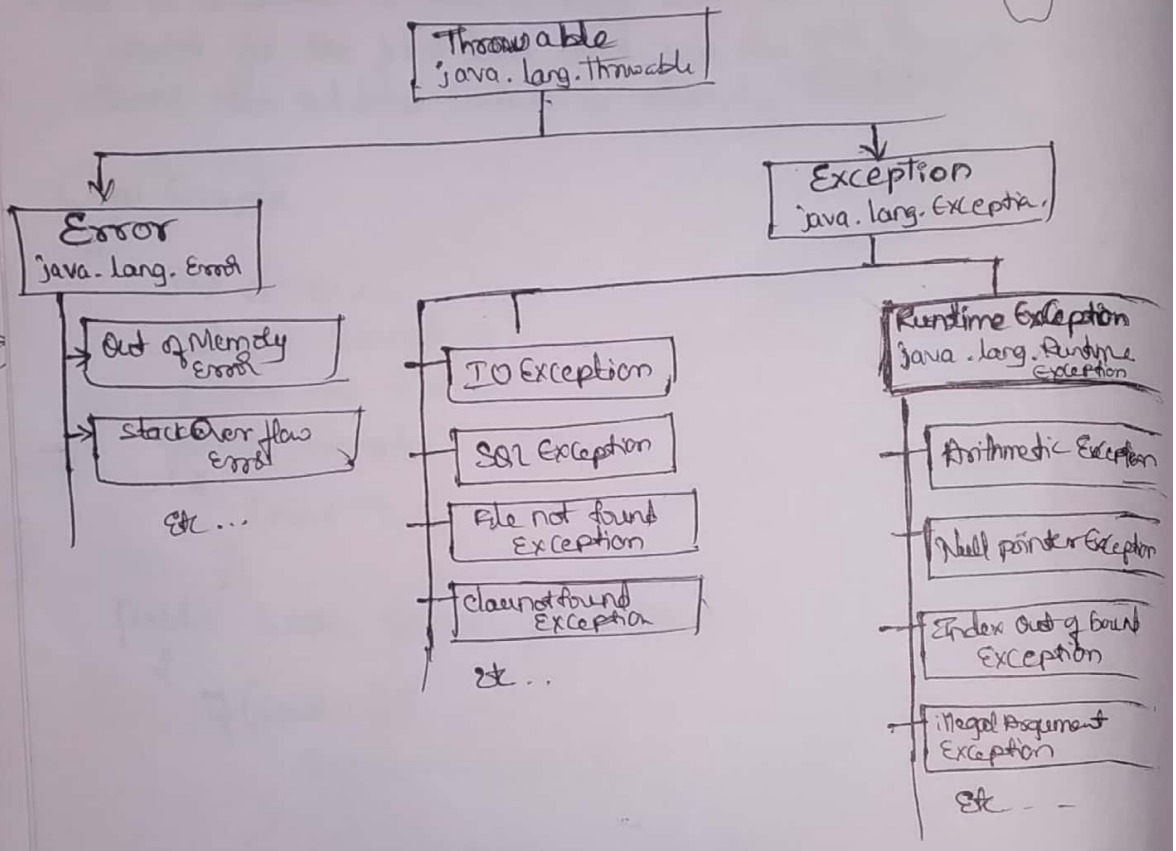
- Restricting the object creation by one through out life cycle of the application is called as singleton design pattern
- To achieve singleton design pattern declare the Constructor as private and initialize Count = 0
- Develop a public service called getter and setter where get is used to get the value, set is used to set the value.
- Put a Condition in such a way that the object Creates for the first time and from the 2nd time it return the address which is already created

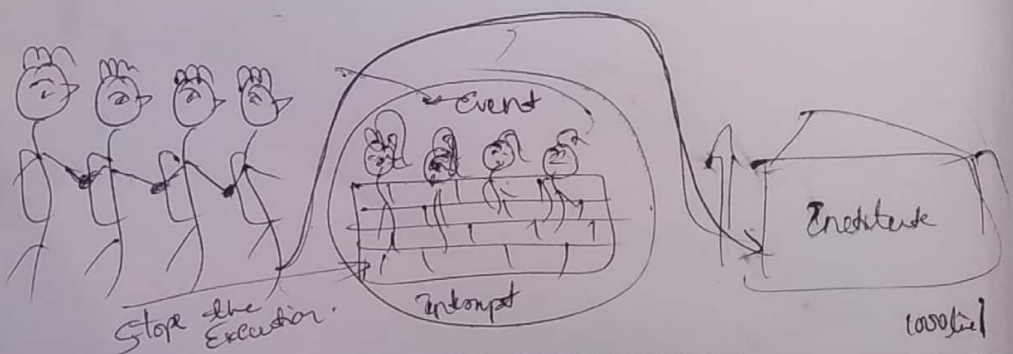
```
Class Sample
{
    int a=10;
    Static Sample s1;
    Static int Cnt=0;
    private Sample()
    {
      Count++;
    }
Public Static Sample getInstance()
{
   if(Cnt<1){
       s1=new Sample();
   }
Public static Void SetInstance(int y)
  {
    s1.a=y;
  }
}
```



Exception handling

- It is an Event triggered during the Execution of the program which interrupts the Execution and stops the Execution abruptly (or) Suddenly
- Handling this Event by using try and catch throws we called as Exception handling (or)
- All the abnormal statements should be developed in the try block and address them in the Catch block





11

during Execution of the program

Syntax for try and catch

① try {
}
catch (Exception class ref Variable)
{
}

② try {
}
catch( )
{
}
catch ( )
{
}

③ try {
try {
}
catch( )
{
}
catch ( )
{
}

④ try {
}
catch( )
{
}
finally {
}

⑤ try {
}
finally {
}

① WAP to handle a runtime Exception by try & Catch

```
⇒ psvm (String[] arg)
{
    System.out.println("***main start***");
    int[] a = {10, 20, 30};
    try {
        S.o.p(a[7]);
    }
    catch (ArrayIndexOutOfBoundException e)
    {
        S.o.p("handled....");
    }
    S.o.p("***main Ende***");
}
}
```

op
main start
handled....
main Ende

② ⇒ psvm (——)

```
{
    S.o.p("***main Start***");
    try {
        int i = 1/0;
    }
    catch (ArithmeticException e)
    {
        S.o.p("handled....");
    }
    S.o.p("***main Ende***");
}
}
```

```
psvm(String[] arg)
{
  Sample s1=null;
  try{
    s.o.p(s1.hashCode());
  }
  catch(NullPointerException e)
  {
    s.o.p("handled...");
  }
  s.o.p("***main Ende***");
 }
}
psvm(---)
{
  s.o.p("main start");
  try
  {
    int i=1/0;
  }
  catch(NullPointerException e)
  {
    s.o.p("handled---");
  }
  catch(ArithmaticException e)
  {
    s.o.p("Cool i am here");
  }
  S.o.p("main Ende");
}
```

o/p

main Start
Cool i am here
main Ende

Once the Exception occurs further statement of the try block will not get executed

Eg: class Tester

```
{
    psvm (——)
    {
        int[] arr={10,20};
        try{
            int i=1/0;
            S.o.p (arr[7]);
        }
        catch (ArithmeticException e)
        {
            S.o.p ("handled");
        }
    }
```

→ class Tester

```
{ psvm (——)
  {
    try{
      int i=1/0; // throw new ArithmeticException("/ by zero");
    }
    catch (ArithmeticException e)
    {
      S.o.p ("handled");
    }
}
```

psvm (———){
int [] arr = {10, 20, 30, 40, 10};
int a=10;
int b=0;
try{
int c = a/b;
S.o.p(arr[8]);
}
catch (Arithmetic Exception e)
{
S.o.p("handled");
}
}

'throw' is used to throw instance of throwable type

Note: ********
Once the Exception is addressed and if we try to re-address then we get Compile time Error.

⇒ psvm(———)
{
S.o.p("*** main start ***");
try
{
int i = 1/0;
}
catch (Exception e)
{
S.o.p("Catch i am here");
}
Catch(Arithmetic Exception e)
{
S.o.p("handled");
}
S.o.p("*** main Ende ***");
}

finally block

- It is a block in Exception handling which will get Executed mandatorily Even though the Exception is addressed or not addressed.
- The finally block should be developed Either with try finally or try catch finally

Eg: • Any database closing collection statements should be developed in the finally block.

- In Amazon 1Rs sale sale due to the load of on the Server the application got crashed where the database Closing collection statement was developed in the finally block. Which got Executed before terminating the application, which helps to protect the database from hackers

```
=> PSVm (String[] args)
{
    s.op("***main Start***");
    try
    {
        int i=1/0;
    }
    Catch (ArrayIndexOutOfBoundException e)
    { s.op("cool i am here");
    }
    finally
    {
        s.op("i am finally");
    }
    S.op("main End");
}
```

WAP to take dynamic input from the User

```
import.util.Scanner
Class Sample
{
  public static void main (String[] args)
  { Scanner sc = new Scanner (System.in);
    System.out.println("Enter the first number:");
    int fno = sc.nextInt();
    System.out.println("The number is" + fno);
    System.out.println("Enter the second number");
    int y = sc.nextInt();
    System.out.println("the number is");
  }
}
```

WAP to print the tables of 2

```
import.util.Scanner
Class Sample
{
  psvm (String[] args)
  {
    int no=1;
    for(int no=1; no<=10; no++)
    {
      System.out.println("Enter the no");
      Scanner sc = new Scanner(System.in);
      int no = sc.nextInt();
      for(int i=1; i<=10; i++)
      {
        System.out.println(no+"x"+i+"="+(no*i));
      }
    }
  }
}
```

WAP

1 10 2 9 3 8 4 7 5 6 6 5 7 4 8 3 9 2 10 1

Class Sample

```
{
  public static void main (String[] args)
  {
    int x=1;
    int y=10;
    for(int i=1; i<=10; i++)
    {
      System.out.print(x++ + " " + y-- + " ");
    }
  }
}
```

WAP

0 1 2 3 4 5 6 7 8 9 8 7 6 5 4 3 2 1 0

Class

WAP to find how many odd digits are present in the given array.

⇒ class Sample

```
{
    public static void main (String [] args)
    {
        int Count = 0;
        int [] arr = {1, 8, 3, 7, 10};
        for (int i=0; i<arr.length; i++)
        {
            if (arr[i] % 2 == 1)
            {
                Count++;
            }
        }
        S.o.p(Count);
    }
}
```

1) arr[0] % 2 == 1
   1 % 2 = 1 2)1(0
                0
                1
2) arr[1] % 2 == 1
   8 % 2 = 0 2)8(4
                8
                0
3) arr[2] % 2 == 1
   3 % 2 = 0 2)3(1
                2
                1
4) arr[3] % 2 == 1
   7 % 2 = 1 2)7(3
                6
                1
5) arr[4] % 2 == 1
   10 % 2 = 1 2)10(5
                 10
                 0

WAP to find how many Even digits are present in an given array.

⇒ class Sample

```
{
    public static void main (String [] args)
    {
        int Count = 0;
        int [] arr = {1, 8, 3, 7, 10};
        for (int i=0; i<arr.length; i++)
        {
            if (arr[i] % 2 == 0)
            {
                Count++;
            }
        }
        S.o.p(Count);
    }
}
```

WAP To print the sum of odd digits

```
Class Sample
{
  psvm (String[] arg)
  {
     int sum=0;
     int arr[] = {1, 8, 3, 7, 10}
     for (i=0; i<arr.length; i++)
     {
       if (arr[i] %2 == 1)
       {
          sum = sum + arr[i];
       }
       S.o.p (sum);
     }
  }
```

WAP to print the sum of Even digits

```
Class Sample
{
    public static void main(String [] arg)
    {
       int sum=0;
       int arr[]={1, 8, 3, 7, 10}
       for (int i=0; i<arr.length; i++)
       {
          if (arr[i] %2 == 0)
          {
             sum = sum + arr[i];
          }
          S.o.p (sum);
       }
    }
}
```

Write a program to reverse a given string "Java".

* understanding of charAt() and length() function

```
class string
{
  char At()
  {
  }
  int length()
  {
  }
}
```

```
String s2 = "Java";
int x = s2.length();
S.o.p(x);                    // o/p: 4
char ch = s2.charAt(3);
S.o.p(ch);                   // o/p = A
char ch1 = s2.charAt(0);
S.o.p(ch1);                  // o/p = 2
```

```
class Reversestring
{
  public static void main (String [] arg)
  {
    String s2 = "Java";
    String s3 = " ";
    for (int i = s2.length()-1; i>=0; i--)
    {
      s3 = s3 + s2.charAt(i);
    }
    S.o.p(s3);
  }
}
```

i) i = 4-1 = 3
   3 >= 0;
   s3 = " " + aaa
   i = -2

ii) i = 2
   2 >= 0;
   s3 = a + v = av

iii) i = 1
   ~~2 >= 0~~
   1 >= 0
   s3 = av + a = ava

iv) i = 0
   0 >= 0
   s3 = ava + j = avaj

s3 = avaj

② WAP to check if the given string is a palindrome & not

⇒ Class palindrome
{
psvm (——)
{
String S1 = "MOM";
String S2 = "";
for(int i = S1.length()-1; i>=0; i--)
{
S2 = S2 + S1.charAt(i);
}
if (S2.Equals(S1))
{ S.o.p("string is a palindrome");
}
else
{ S.o.p ("string is not a palindrome")
}
}
}

* The job of toCharArray() method is to convert string to character array
* toCharArray() method is a non-static method of String class

③ WAP to reverse a string without using string function and check if its a palindrome & not

⇒ Class Reverse
{
psvm(——)
{
String S1 = "Java";
String S2 = "";
char[] arr = S1.toCharArray();
for(int i = arr.length-1; i>=0; i--)
{ S2 = S2 + arr[i];
}
S.o.p(S2);
if (S1.Equals(S2));
{ S.o.p ("It is a palindr")
}
else
{ S.o.p ("Not palin")
}
}

arr	J	a	V	a
	0	1	2	3

i) i = 4-1 = 3
3>=0
S2 = "" + arr[3]
S1 = "" + a = a
ii) i = 2
2>=0
S2 = a + arr[2]
S2 = avr = av
iii) i = 1
1>=0
S2 = av + arr[1]
= av + a = ava
iv) i = 0
0>=0
S2 = ava + arr[0]
= avaj
i = -1
v) -1 >= 0 ✗
S2 = avaj

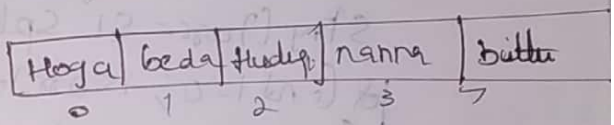
split() is a non static method of string class
split() method is used to split the string based on the given input

```
class Sting
{
    String[] split()
    {
    }
}
```

④ Reverse a string using split function

⇒ class Reverse

```
{ psvm(---)
  { string s1 = "Hoga beda Hudugi nanna bittu";
    string[] arr = s1.split(" ");  → split based on space
    for(int i = arr.length-1; i>=0; i--)
    { s.o.p(arr[i] + " ");
    }
  }
```



O/p bittu nanna hudugi beda hoga

⇒ I/p = Hoga beda Hudugi nanna bittu
O/p = Hoga adeb Hudugi annan bittu

```
class String
{
  psvm(---------)
  {
    String s1 = "Hoga beda Hudugi nanna bittu";
    String[] arr = s1.split(" ");
    for(i=0; i<arr.length; i++)
    {
      if(i%2 != 0)
      {
        String s2 = arr[i];
        String s3 = "";
        for(int j = s1.length()-1; j>=0; i--)
        {
          s3 = s3 + s2.charAt(i);
        }
      }
    }
  }
```

```
            S.o.p(s2+" ");
        }
        else
        {
            s.o.p(arr[i] + " ");
        }
    }
}
```

⑤ WAP to reverse only Even positions in the given story "Hoga beda hudugi nanna bittu".

⇒

```
class Reverse_String
{
    public static void main(String[] arg)
    {
        String s1="Hoga beda hudugi nanna bittu";
        String[] arr=s1.split(" ");
        for(int i=0; i<arr.length; i++)
        {
            if(i%2==0)
            {
                String s3=arr[i];
                String s4="";
                for(int j=s3.length()-1; j>=0; j--)
                {
                    s4=s4+s3.charAt(j);
                }
                s.o.p(s4+" ");
            }
            else
            {
                s.o.p(arr[i]+" ");
            }
        }
    }
}
```

⑥ class Rev
{
psvm (———)
{
String S4 = "Hoga beda hudugi nanna bittu";
String [] arr = S4. Split (" ");
for (int j=0 ; j < arr.length ; j++)
{ if (j % 2 != 0)
{ String S1 = arr[j];
String S2 = "";
for (int i = S1.length() - 1; i >= 0; i--)
{
S2 = S2 + S1.charAt(i);
}
S.o.p (S2 + " " + S2.length() + " a ");
}
else
String S3 = arr[j];
S.o.p (S3 + " " + S3.length() + " ");
}
}
}

o/p : Hoga4 adeb 4 hudugi 6
annan 5 bittu 5

⑦ WAP to print how many palindrome are present in a given string

→ class Count_palin
{
psvm (———)
{
int Countp = 0;
int Countp = 0;
String S1 = "mom amma hogo dad appa hogo";
String [] arr = S1. Split (" ");
for (int j=0 ; j < arr.length ; j++)
{ String S3 = arr[j];
String S4 = S" ";
for (int i = S3.length-1 ; i >= 0 ; i--)

```
    S4 = S4 + S3.charAt(i);
  }
  if (S3.equals(S4))
    countp++;
  else
    countnp++;
 }
 S.o.p("the no of palindrome is " + countp);
 S.o.p("the no of non palindrome is " + countnp);
}
```

o/p: The no of palindrome is 4
The no of palindrome is 2

* Array is a data structure, but array is a class

```
class Array
{
  char() sort()
  {
  }
}
```

```
java.util
  -> class Array
  {
    char() sort()
    {
    }
  }
```

WAP to program to sort an array without using bubble sort

```
=> import java.util.Arrays;
public class Demo1
{
  public static void main(String[] args)
  {
    char()
```

WAP to check 40 is present in an array /// linear Search

```
⇒ int find = 40;
  int [] arr = {10, 20, 30, 40, 50};
  boolean flag = true;
  for (int i=0 ; i<arr.length ; i++)
  {
      if (arr[i] == find)
      {
          flag = false;
      }
  }
  if (flag == false)
  {
    S.o.p ("present");
  }
  else
  {
    S.o.p ("Not present");
  }
}
```

WAP to find the character 'A'.

```
⇒ char find = 'A';
  char [] arr = {'x', 'c', 'A', 'D'};
  boolean flag = true;
  for (int i=0; i<arr.length; i++)
  {
      if (arr[i] == find)
      {
          flag = false;
      }
  }
  if (flag == false)
  {
    S.o.p ("present");
  }
  else
  {
    S.o.p ("not present");
  }
}
```

⇒ class Sample

```
{
public static void main(String[] arg)
{
    char[] abb={'x','i','o','a'};
    char find ='o';
    search (abb, find);
}
static void search (char[] arr, char find)
{
    boolean = true;
    for(int i=0; i<arr.length; i++)
    {
        if (arr[i] == find)
        {
            flag = false;
        }
    }
    if(flag==false)
    {
        S.o.p("present");
    }
    else
    {
        S.o.p ("Not present");
    }
}
```

WAP to swap two numbers using 3rd Variable.

```
psvm (— )
{
    int a=10;
    int b=20;
    int temp=a; //10
    a=b //20
    b=temp //10
    S.o.p(a)
    S.o.p(b)
}
```

WAP a swap two nos without 3rd Variable

```
{
    int a=10;
    int b=20;
    a= a+b; //
    b= a-b;
    a= a-b;
    S.o.p(a);
    S.o.p(b);
}
```

(a+b)
= 30
30-20
= 10
a= 30-10
= 20

WAP to convert a string to an integer

```
public static void main (String[] arg)
{
    String s1="123";
    int x= Integer.parseInt(s1);
    S.o.p(x);
}
```

Primitive datatype	Wrapper class
int	Integer
byte	Byte
short	Short
long	Long
float	Float
double	Double
char	Char

class Integer

```
{
    static void parseInt( )
    {
    }
}
```

inbuilt

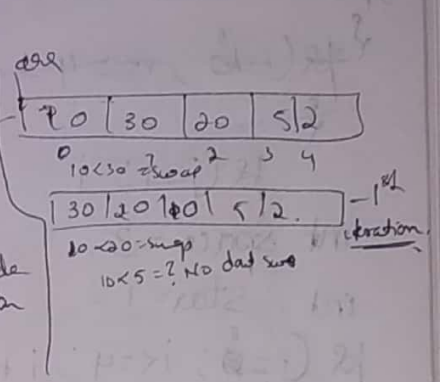
WAP to sort an array in

Scanned by CamScanner

WAP to sort an array in ascending order using Bubble sort

```
=> public static void main (String[] args)
{
    int [] arr = {10, 30, 20, 5, 2};
    int n = arr.length;
    for (int i = 0; i<n; i++)
    {
        for (int j = 1; j<n; j++)        // n-i will Can use for Code optimization
        {
            if (arr[j]<arr[j-1])
            {
                int temp arr[j]
                arr[j] = arr[j-1];
                arr[j-1] = temp;
            }
        }
    }
    for (int i=0; i<arr.length; i++)
    {
        S.o.p (arr[i] + " ");
    }
}
```

This loop should run for 4 times if n=5 (n-1) means (n-1) = 4 iterations



=> Descending Order

```
psvm (———)
{
    int [] arr = {10, 30, 20, 5, 2};
    int n = arr.length;
    for (i=0; i<n-1; i++)
    {
        for (j=1; j<n-i; j++)        // n-i => Code Optimization
        {
            if (arr[j-1] < arr[j])
            {
                int temp = arr[j];
                arr[j] = arr[j-1];
                arr[j-1] = temp;
            }
        }
    }
    for (int i=0; i<arr.length; i++)
    {
        s.o.p (arr[i] + " ");
    }
}
```

WAP to print a pattern

```
   *
  * *
 * * *
* * * *
```

```
public static void main (String[] arg)
{
for(i=1; i<=4; i++)
{
    for(j=1; j
int space = 3;
int star = 1;
for (i=1; i<=4; i++) //row
{
  for(j=1; j<=space; j++) //will print space
  {
    S.o.p (" ");
  }
  for(k=1; k<=star; k++) // print Star (*)
  {
    S.o.p("*");
  }
  S.o.pln();
  space--;
  star++;
}
```

_	_	_	*
_	_	*	*
_	*	*	*
*	*	*	*

To print

```
    1
  1 2
1 2 3
1 2 3 4
```

```
int space=3;
int star=1
for(i=1; i<=4; i++)
{
  for(j=1; j<=space; j++)
  {
    S.o.p(" ");
  }
  for(k=1; k<=star; k++)
  {
    S.o.p(k);
  }
  S.o.pln();
  space--;
  star++;
}
```

To print

```
    1
  2 2
3 3 3
4 4 4 4
```

```
int space=3;
int star=1
for(i=1; i<=4; i++)
{
  S.o.p(" ");
  for(j=1; j<=space; j++)
  {
    S.o.p(" ");
  }
  for(k=1; k<=star; k++)
  {
    S.o.p(i);
  }
  S.o.pln();
  space--;
  star++;
}
```

print
4
4 3
4 3 2
4 3 2 1

```
int space = 3;
int star = 1;
for (i=1; i<=4; i++)
{
    for (j=1; j<=space; j++)
    {
        S.o.p(" ");
    }
    for (k=1; k<=star; k++)
    {
        S.o.p(i);
    }
    S.o.pln();
    space--;
    star++;
}
```

print
4
3 3
2 2 2
1 1 1 1

```
=> int space = 3,
int star = 1
for (i=4; i>=1; i--)
{
    for (j=1; j<=space; j++)
    {
        S.o.p(" ");
    }
    for (k=1; k<=star; k++)
    {
        S.o.p(j);
    }
    S.o.pln();
    space--;
    star++;
}
```

print
a
b b
c c c
d d d d

```
int space = 3;
int star = 1;
for (i=1; i<=4; i++)
{
    for (j=1; j<=space; j++)
    {
        S.o.p(" ");
    }
    for (char ch='a'; ch<=star; ch++)
    {
        S.o.p(ch);
    }
    S.o.pln();
    space--;
    star++;
}
```

print
a
a b
a b c
a b c d

```
=> int space = 3;
int star = 1;
for (i=1; i<=4; i++)
{
    char ch = 'a';
    for (j=1; j<=space; j++)
    {
        S.o.p(" ");
    }
    for (k=1; k<=star; k++)
    {
        S.o.p(ch);
        ch++;
    }
    S.o.pln();
}
```

print
d
d c
d c b
d c b a

```
int space=3;
int star=1;
for(i=0;i<=4;i++)
{
  char ch='d';
  for(j=1;j<=space;j++)
  { s.o.p(" ");
  }
  for(k=1;k<=star;k++)
  { s.o.p(ch);
    ch--;
  }
  sop();
}
space--;
star++;
```

print
a
b c
d e f
g h i j

```
int space=3;
int star=1;
char ch='a'; //Outside the for loop
for(i=1;i<=4;i++)
{
  for(j=1;j<=space;j++)
  { s.o.p(" ");
  }
  for(k=1;k<=star;k++)
  { s.o.p(ch);
    ch++;
  }
  sopln();
  space--;
  star++;
}
```

print
d
c c
b b b
a a a a

```
int space=3;
int star=1;
```

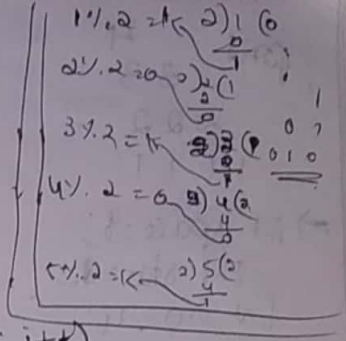
print
1
2 3
4 5 6
7 8 9 10

```
int space=3;
int star=1;
int no=1; //outside
for(i=0;i<=4;i++)
{
  for(j=1;j<=space;j++)
  { s.o.p(" ");
  }
  for(k=1;k<=star;k++)
  { s.o.p(no);
    no++;
  }
  sopln();
  star++;
  space--;
}
```

print
10
9 8
7 6 5
4 3 2 1

```
int space=3;
int star=1;
int no=10;
for(i=1; i<=4; i++)
{
  for(j=1; j<=space; j++)
  {
    S.o.p(" ");
  }
  for(k=1; k<=star; k++)
  {
    S.o.p(no);
    no--;
  }
  S.o.pln();
  star++;
  space--;
}
```

print
1
0 1
0 1 0
1 0 1 0




```
⇒ int space=3;
  int star=1;
  int no=1;
  for(i=1; i<=4; i++)
  {
    for(j=1; j<=space; j++)
    {
      S.o.p(" ");
    }
    for(k=1; k<=star; k++)
    {
      S.o.p(no%2);
      no++;
    }
  space--;
  star++;
  }
```

print
d
c c
b b b
a a a a

```
⇒ int space=3;
  int star=1;
  char ch='d';
  for(i=1; i<=4; i++)
  {
    for(j=1; j<=space; j++)
    {
      S.o.p(" ");
    }
    for(k=1; k<=star; k++)
    {
      S.o.p(ch);
    }
    ch--; // outside for loop
    Sop();
    Space--;
    Star++;
  }
```

print
a
b b
c c c
d d d d

```
⇒ int space=3;
  int star=1;
  char ch='a';
  for(i=1; i<=4; i++)
  {
    for(j=1; j<=space; j++)
    {
      S.o.p(" ");
    }
    for(k=1; k<=star; k++);
    {
      S.o.p(ch);
    }
    ch++;
  Sopln();
  Space--;
  Star++;
  }
```

print
4
3 3
2 2 2
1 1 1 1

```
=> int space = 3;
   int star = 1;
   int no = 4;
for (i=1; i<=4; i++)
{
   for (j=1; j<=space; j++)
   {
      S.o.p(" ");
   }
   for (k=1; k<=star; k++)
   {
      S.o.p(no);
   }
   no--; // outside
   space--;
   star++;
}
```

print
*
* *
* * *
* * * *

```
=> int space = 3;
   int star = 1;
for (int i=1; i<=4; i++)
{
   for (j=1; j<=space; j++)
   {
      S.o.p(" ");
   }
   for (k=1; k<=star; k++)
   {
      S.o.p("* ");
   }
   space--;
   star++;
}
```

pro

j
j a
j a v
j a v a

```
=> int space = 3;
   int star = 1;
   String s1 = "java";
for (i=1; i<=4; i++)
{
   for (j=1; j<=space; j++)
   {
      S.o.p(" ");
   }
   for (k=0; k<=star; k++)
   {
      S.o.p(s1.charAt(i));
   }
spa
```

print
*
* * *
* * * * *
* * * * * * *

```
=> int space = 3;
   int star = 1;
for (i=1; i<=4; i++)
{
   for (j=1; j<=space;
   {
      S.o.p(" ");
   }
   for (k=1; k<=star; k++)
   {
      S.o.p("* ");
   }
   S.o.p();
   space--;
   star = star+2;
}
```

print

```
int space = 0;
int star = 7;
for(i=1; i<=4; i++)
{ for(j=1; j<=space; j++)
  { S.op(" ");
  }
  for (k=1; k<=star; k++)
  { S.op("* ");
  }
  space++;
  star = star-2;
}
```

print

```
⇒int space = 3;
int star = 7
for (int i=1, i<=4, i++)
{ for (int j=1; j<=space; i++)
  { S.op(" ");
  }
  for (int k=1; k<=star; k++)
  { S.op("*");
  }
  S.op();
  space--;
  star = star+2;
}
int space1 = 1
int star1 = 1
for (int i=1; i<=4; i++)
{ for (j=1; j<=space1; j++)
  { S.op(" ");
  }
  for (k=1; k<=star; k++)
  { S.op("* ");
  }
  S.op();
  space++;
  star1 = star1-2;
}
}
```

print

```
⇒int space = 0;
int star = 3;
for (i=1; i<=3; i++)
{ for(j=1; j<=space; j++)
  { S.op(" ");
  }
  for(k=1; k<=star; k++)
  { S.op("*
```

*WAP to generate fibonacci Series of first 10 nos (or) within the range 100

0 1 1 2 3 5 8 13 21 34 55 89

⇒ Class Sample1

```
{
    public static void main (String[] args)
    {
        int fib1=0;
        int fib2=1;
        int fib3;
        S.o.p (fib1+" "+fib2);
        for(int i=1; i<=10; i++)
        {
            fib3= fib1+fib2;
            S.o.p (fib3);
            fib1=fib2;
            fib2 = fib3;
        }
    }
}
```

⇒ WAP to remove the nos in an array

```
int arr[] = {10, 20, 0, 30, 20, 40};
for (int i = 0; i <= 5; i++)
{
    if (arr[i] != 0)
    {
        for (int j = i+1; j <= 5; j++)
        {
            if (arr[i] == arr[j])
            {
                arr[j] = 0;
            }
        }
    }
}
for (int i = 0; i <= 5; i++)
{
    if (arr[i] != 0)
    {
        S.o.p (arr[i]);
    }
}
```

⇒ WAP to remove the string duplicates from an array.

```
String arr[] = {"hi", "hello", "hi", "bye", "bye"};
for (int i = 0; i <= 4; i++)
{
    if (arr[i] != null)
    {
        for (int j = i+1; j <= 4; j++)
        {
            if (arr[i].Equals(arr[j]))
            {
                arr[j] = null;
            }
        }
    }
}
for (i = 0; i <= 4; i++)
{
    if (arr[i] != null)
    {
        S.o.p (arr[i]);
    }
}
```

Prime no^r

```
Class Sample
{
  psvm (String [] arg)
  {
    boolean flag=true;
    int num=6;
    int copy=num;
    for (int i=2 ; i<6; i++)
    {
      if (num%i==0)
      {
        flag=false;
      }
    }
    if (flag==true)
    {
      S.o.p("is a palindrome ");
    }
    else
    {
      S.o.p ("Not a palindrome");
    }
  }
```

WAP to print only prime no^r from 1 to 100

```
→ Class Sample
  {
    psvm (————)
    {
      for (int i= 2; i<=100; i++)
      {
        ~~int no =i;~~
        boolean flag=true;
        for (int j= 2; j<i ; j++)
        {
          if (no%i ==0)
          {
            flag =true;
          }
        }
        if (flag==true)
        {
          S.o.p ("prime no is "+ i);
        }
      }
    }
```

Comman line program

```
Class Sample
{
 public static Void main (String [] args)
 {
   for (int i=0; i<args.length; i++)
   {
      s.o.p (args [i]);
   }
 }
}
```

in Eclipse

Run
run Configuration
Arguments -> type here.