CORE JAVA

BY
Mr.VENKATESH SIR

SATYA TECHNOLOGIES

VT

19/2/15

Core Java

What is Java:-

Java is a Technology and it has two things

1) programming language
2) platform

Java is called as programming language because by using java we can write programs

programs are divided into two categories

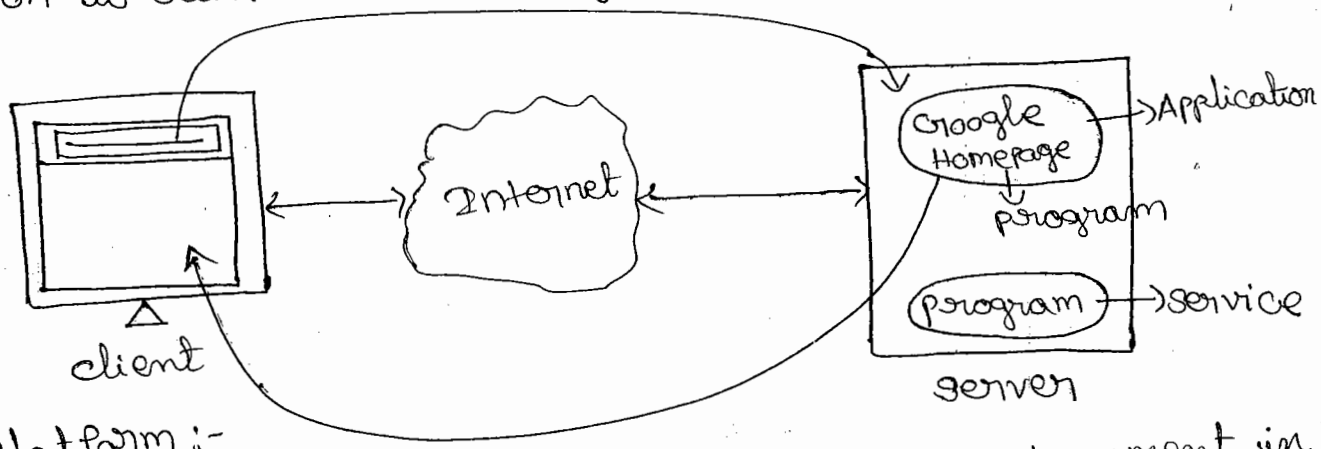
1) Applications
2) services

Application:-

an application is a program in which we interact on the desktop

service:-

a service is a program and it is used by an application to run under operating system, browser and server



Platform:-

It can be a software or Hardware environment in which program runs.

Demo.C (unicode)
↓
compiler
↓
Demo.exe (bit code)
↓
windows
↓
output

Demo.java (unicode)
↓
compiler
↓
Demo.class (byte code)

windows JVM	linux JVM	solaries JVM
↓	↓	↓
windows	linux	solaries
↓	↓	↓
o/p	o/p	o/p

→ Java is a platform independent because programs written in java language can be executed on any platform

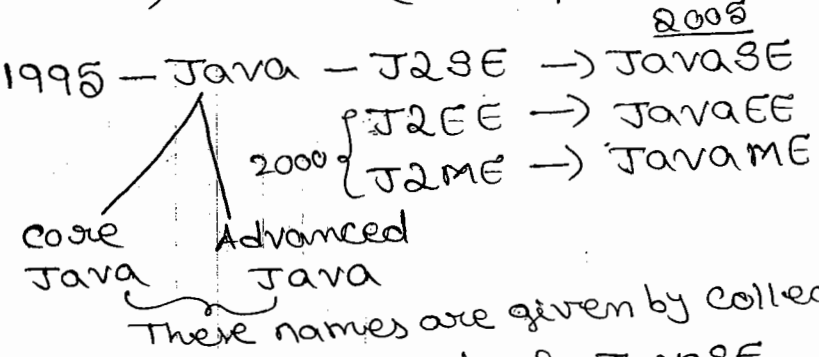
→ Java virtual machine is not platform independent because windows JVM is specific to windows platform Linux JVM is specific to linux platform, solaries JVM is specific to solaries platform etc.

→ Java virtual machine is a part of java runtime Environment

→ Java Runtime environment is called Java platform

→ There are 4 Java platforms

1) JavaSE (Java platform, standard Edition)
2) JavaEE (Java platform, Enterprise Edition)
3) JavaME (Java platform, micro Edition)
4) JavaFX (Java platform, FX)



→ Core java is a part of JavaSE

→ Advanced java is a part of javaSE & javaEE

Core Java

1) Java Fundamentals
2) object oriented programming (OOP)
3) Interfaces & packages
4) String Handling & wrapper classes
5) Exception Handling
6) Java Streams & serialization
7) networking programming
8) collection Framework & Generics
9) multithreading & synchronization

10) Inner classes
11) Abstract window Toolkit (AWT), Event Handling
Applets & Swing
12) New Features (from JDK 1.0 to JDK 1.8)

Java Fundamentals:-

1995 "Java" was developed by James Gosling, Patrick Naughton, Ed Frank, Chris Warth & Mike Sheridan at SUN microsystems (now owned by oracle corporation)

Java slogan

"Write once, Run anywhere (WORA)"

Identifiers:- 20/2/15

→ Identifier is a word and it is used to identify variable, method, class, interface, package .... etc.

→ It can be formed by using alphabets, digits, (_) symbol and ($) symbol

Rules to declare an Identifier:-

1) It must begans with alphabet, (_) symbol (or) $ symbol
2) The length of the Identifier is not limited
3) It should not contains special symbols other than (_) & ($) symbols.
4) It should not contain whitespace characters (spacebar, Tab, & Enter keys)

Examples:-

1) demo (✓)
2) anddemo (X)
3) demo2 (✓)
4) DEMO (✓)
5) Demo (✓)
6) -args (X)
7) _args (✓)
8) $args (✓)
9) #args (X)
10) net income (X)
11) net_income (✓)
12) net*income (X)
13) _$_ ✓
14) _$args ✓
15) _args5$ ✓

Keywords:-

→ A set of words reserved by language itself and those words are called keywords

→ There are total 50 keywords in Java including strictfp, assert & enum

→ "strictfp" keyword added in JDK 1.2 version in 1998

→ "assert" keyword added in JDK 1.5 version in 2002

→ "enum" keyword added in JDK 1.5 version in 2004

→ JDK stands for Java development kit and it is called as Java software.

→ constant and "goto" keywords presently not in use. ↳ "const"

→ Keyword can not be used as an identifier

1) int x = 5;

2) int enum = 10; X
Error because enum is a keyword

3) int void = 5; X Error
bcoz void is a keyword

→ All keywords must be written in lower case letters otherwise compiler time error occurs.

DataTypes:-

→ Datatype that determines what value variable can hold and what are the operations can be performed on variable.

→ In Java datatypes are divided into two categories

1) primitive datatypes
2) Reference datatypes

1) primitive Datatypes:-

→ primitive datatypes are predefined datatypes and those are named by keywords

→ There are 8 primitive datatypes.

These are divided into 4 sub categories.

1) Integers
2) Floating point numbers
3) characters
4) Boolean

	Datatype	Memory size	Range
Integers	byte	1 Byte (8 bits)	-2^7 to 2^7-1
	short	2 Bytes (16 bits)	-2^{15} to $2^{15}-1$
	int	4 Bytes (32 bits)	-2^{31} to $2^{31}-1$
	long	8 Bytes (64 bits)	-2^{63} to $2^{63}-1$
Floating point numbers	float	4 Bytes (32 bits)	-3.4×10^{-38} to 3.4×10^{38}
	double	8 Bytes (64 bits)	-1.7×10^{-308} to 1.7×10^{308}
Character	char	2 Bytes (16 bits)	0 to 65535
Boolean	boolean	1 bit	true/false

Reference datatypes:-

Arrays, strings, classes, Interfaces .... etc

21/2/15

Literals:-

A literal is a source code representation of a fixed value.

In java literals are divided into 6 categories.

1) Integer literals:-

5, 7, 587, 0, -2, -57, ... etc.

2) Floating point literals:-

5.2, 7.9, 6.578, 0.009, -2.57 ... etc.

3) Character literals:-

'a', 'A', '\n', '\t' ... etc

4) String Literals :-

"Hi", "Hello", "\n", "A" .... etc.

5) Boolean literals :-

true, false

6) object Literals :-

null.

Note :-

true, false and null are not keywords.

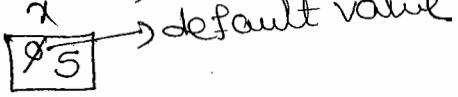
Variables :-

a variable is a container which contains data

Declaration :-

Syntax :- Datatype variable;

Ex :- int x;



Assignment :-

syntax :-

variable = literal;

Ex :- x = 5;

Intialization :-

syntax :- DataType variable = Literal;

Ex :- int x = 5;

primitive DataType	defaultvalue
1) byte	0
2) short	0
3) int	0
4) long	0
5) float	0.0f
6) double	0.0
7) char	blankspace (ASCII value is 0)
8) boolean	false.

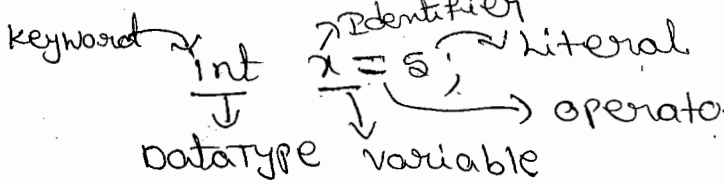
→ Long literal must be suffixed with l (or) L because int type is default in integers category

byte a = 5; (✓)
byte b = 130; (✗)
short c = 130; (✓)
short d = 40000; (✗)
int e = 40000; (✓)
int f = ____; (✗)
↓
out of int range

long g = ____ (✗)
↓
out of int range and within long range
long h = 40000; (✓)
long i = ____ l (or) L (✓)
↓
out of int range with in long range
long j = ____; (✗)
↓
out of long range.

→ Every float literal must be suffixed with f or F because double type is default in floating point numbers category.

float a = 3.45; (✗)
float b = 3.45 f (or) F; (✓)
double c = 3.45; (✓)



operators:-

an operator is a special symbol which operates on data.

Types of operators:-

1) unary operator
2) Binary operator
3) Ternary operator

unary operator:-

an operator that operates on only one operand is known as unary operator.

ex:- a++, ++a, a--, --a, !a, ... etc

Binary operator :-

an operator that operates on two operands is known as binary operator.

Ex:- a+b, a-b, a*b, a/b, a%b, a<b, a>b, ... etc.

Ternary operator :-

an operator that operates on three operands is known as ternary operator

Ex:- conditional operator (? :)

```
int a=5, b=8;
int c=(a>b)? a : b;
```

c ↓ 8

true

false

statements :-

1) selection statements :-
   i) If statement
   ii) if else statement
   iii) if else if.... else statement
   iv) nested if statement
   v) switch statement

2) Iteration statements (LOOPS) :-
   i) while loop
   ii) do while loop
   iii) for loop
   iv) Enhanced for Loop (or) for each loop
   v) nested loops.

3) Jump statements :-
   i) break statement
   ii) break LABLE statement
   iii) continue statement
   iv) continue LABLE statement
   v) return statement

In Java:-

```
int a=5;
System.out.println(a);
```

class — object Reference (or) Reference variable — method

In C :-

```
int a=5;
printf("%d", a);
```

→ System is a class and it is belongs to java.lang package.

Java Notations:-

1) Package and subpackage:-

all small letters subpackages are separated with dot(.) symbol.

Ex:- i) java.lang
ii) java.awt.event
iii) Javax.swing

2) class and Interface:-

Each word first letter is capital no space between words

Ex:- i) System
ii) StringBuffer
iii) NullPointerException } classes
iv) Runnable
v) Autocloseable } Interfaces

→ nouns are classes and adjectives are interfaces

3) variable and method:-

second word onwards first letter is capital. no space between words. (but not recommended for future use we are using second word 1st letter capital)

Examples:-

i) name } variables
ii) empno }
iii) main()
iv) compareTo()
v) equalsIgnoreCase() } methods

import java.lang.system; :-

This statement imports only system class in our Java program

import java.lang.*; :-

This statement imports all classes and all interfaces of java.lang package into our Java program.

Note:-

java.lang package is a default package which is imported automatically in every java program

Arrays:-

an array is a collection of similar data elements

Declaration:-

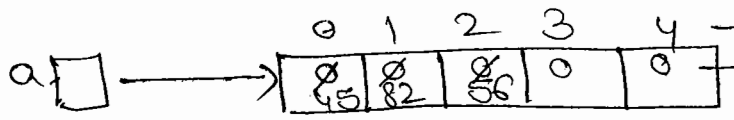
Ex:- int a[] = new int[5];

(or)

int[] a = new int[5];

(or)

int []a = new int[5];

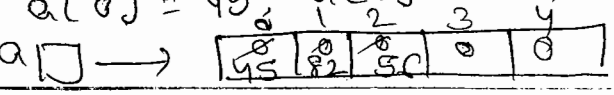


→ Array index always begins with 0 (zero) and ends with size−1

→ new is called as dynamic memory allocation operator and it allocates the memory.

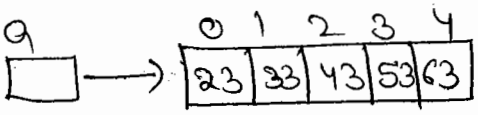
Assignment Ex:-

a[0] = 45; a[1] = 82; a[2] = 56;



Initialization example:-

```
int a[] = {23, 33, 43, 53, 63};
```



Accessig Array elements by using For loop:-

```
for(int i=0; i<a.length; i++){
    system.out.println(a[i]);
}
out of loop
```

false

o/p:-

```
0 0<5  23
1 1<5  33
2 2<5  43
3 3<5  53
4 4<5  63
5 5<5 X
```

C/C++:-

```
char ch[10] = "welcome"
```

Java:-

```
char ch[10] = "welcome"; X
String s = "welcome" ✓
```

23/2/15

Object oriented programming:-

Java is an object oriented programming language.

OOPL:-

a language that supports all the principles of an object oriented programming is known as an OOPL.

object oriented principles:-

1) Encapsulation
2) Abstraction
3) Inheritance
4) Polymorphism

To use the above principles in a Java programming we need the following language constructs

1) class
2) object

class:-

It is a collection of variables and methods

syntax:-

```
class 'classname
{
    Datatype  variable1;
    Datatype  variable2;
      : :       : :
    ReturnType methodname(arg1, arg2, --)
    {
       ==
    }
  : : : :
}
```

Example:- Emp.java

```
class Emp
{
    int empNo;
    String name;
    float salary;          } variables
    void accept().
    {
       ==
    }                      } methods
    void display()
    {
       ==
    }
}
```

101 "____" 5000.00
102 "____" 7000.00
103 "____" 5500.00

→ class will not occupy memory
→ files occupies memory

object:-

an Instance of a class

syntax:-

```
classname objectReference = new constructor();
```

↑ object.

Example:-

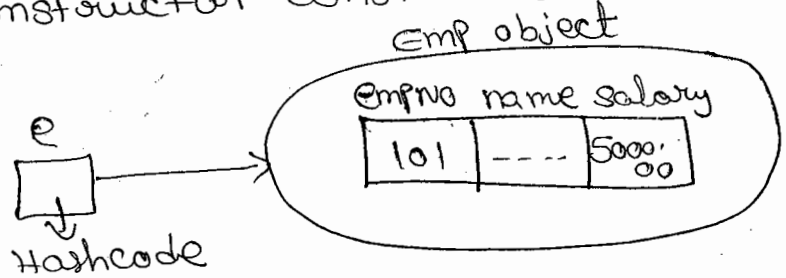
```
Emp e = new Emp();
```

new:-

It is called as dynamic memory allocation operator and it allocates the memory to variables at runtime

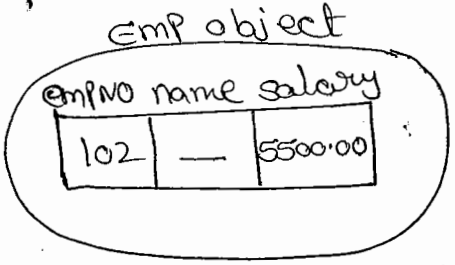
constructor:-

constructor constructs the object



Without object reference:-

new Emp(); → Anonymous object (or) unreferenced object



→ every object occupies memory
→ object contains data where as object reference contains hashcode

24/2/15

Java program structure:

```
class Demo
{
    public static void main(String args[])
    {
        System.out.println("welcome");
    }
}
```

Declaration Rules to a source file (.java file) :-

1) a source file can have only one public class
2) a source file can have any number of non-public classes
3) If the source file contains public class then the file name must match with public class name
4) If the source file does not contain public class then no naming restrictions to a file name

Examples :-

1) B.java

```
public class A
{
  ==
}
```
X

2) B.java

```
class A
{
  ==
}
```
✓

3) A.java

```
public class A
{
  ==
}
public class B
{
  ==
}
```
X

4) A.java

```
public class A
{
  ==
}
class B
{
  ==
}
```
✓

5) C.java

```
class A
{ ==
}
class B
{ ==
}
```
✓

To compile :-

C:\>javac demo.java ↵

↓

demo.class

To Run :-

C:\> java demo ↵

↓

output

→ javac and java are called jdk tools
→ jdk stands for Java development kit and it is called as java software
→ All JDK tools are the part of bin folder in JDK

temporary path setting :-

c:\> set path = "%path%"; c:\program Files\Java\jdk1.8.0\bin;

- set path → It is a command it is used to set the path
- "%path%" → append mode
- c:\program Files\Java\ → Location of java s/w
- jdk1.8.0\bin; → java s/w
- bin; → It contains JDK tools → JavaC, Java are JDK tools.

versions are updated in the following ways :-

1.6.0 – 11
1.6.0 – 12 (little bit change)
1.6.1 (some changes)
1.7.0 (more changes)
2.0.0 (more & more changes)

permanent path setting :-

Right click on my computer → select properties → select Advanced Tab → click on Environment variables button → click on new button under user variables

variable name: Path

variable value: c:\program Files\Java\jdk1.8.0\bin;

→ click on OK button → click on OK button → click on OK button

→ user variables are specific to user where as system variables are common to all users

→ use edit button to append the path

Demo.java

```
class Demo
{
  ==
}
```

c:\>javac Demo.java

↓

Demo.class

c:\>java Demo↵

↓

output

B.java

```
class A
{
  ==
}
```

c:\>javac B.java↵

↓

A.class

c:\>java A↵

↓

output

c:\>java B↵

Error

→ The java compiler generates .class file for every class in a source file

↳ .java file

→ a class that contains main method only can be used to execute the program.

Example:-

D.java

```
class A
{
  ==
}
class B
{
  ==
}
class C
{
  ==
  public static void main(String args[])
  {
    ==
  }
}
```

compilation and Execution

javac D.java↵

↓

A.class

B.class

C.class

c:\>java C↵

↓

output

dir *.java → It shows java files

dir *.class → It shows .class files

Example:-

C.java

```
class A
{
    public static void main(String args[])
    {
        System.out.println("core java");
    }
}
class B
{
    public static void main(String args[])
    {
        System.out.println("Advanced Java");
    }
}
```

compilation:-

```
c:\>javac c.java
```

↓

A.class
B.class

Execution:-

```
c:\>java A
  core java
c:\>java B
  Advanced Java
c:\>java C
  Error
```

25/2/15

variables :-

a variable is a container which contains data.

Types of variables :-

1) Instance variables
2) class variables
3) Local variables

1) Istance variables :-

→ a variable that is defined as a member of class is known as an Instance variable.

→ Istance variables memory allocated whenever an object is created.

→ Instance variables are stored in Heap Area

2) class variables :-

→ a variable that is defined as a static member of a class is known as class variable.

→ class variables memory allocated whenever class is loaded.

→ class variables are stored in method Area.

3) Local variables :-

→ a variable that is defined inside a method is known as a Local variable.

→ Local variables memory allocated whenever method is called.

→ Local variables are stored in Stack Area.

NOTE 1:-

Local variables can not be static

NOTE2:-

No global variables in Java (outside the class)

Example:-

```
int a=5; X
static int b=10; X        } -> global variables
class Demo
{
    int c=15;             -> Instance variable
    static int d=20;      -> class variable
    public static void main(String args[])   -> class method
    {
        int e=25;         -> Local variable
        static int f=30; X
    }
}
```

Execution priority:-

1) class variables
2) static blocks
3) public static void main(String args[]) i.e main method

new:-

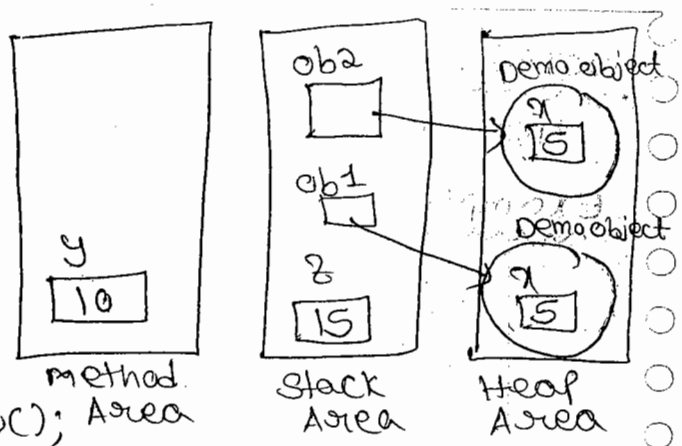
new is called as dynamic memory allocation operator and it allocates the memory to instance variables

=> Primitive Type variable contains data whereas Reference Type variable contains Hash code

-> only one copy of class variable exists for all objects

-> separate copy of Instance variable exists for every object

```
class Demo
{
    int x=5;
    static int y=10;
    PSVM(String args[])
    {
        int z=15;
        Demo ob1=new Demo();
        Demo ob2=new Demo();
    }
}
```



primitive type variable

Reference type variable

local variable

object reference is also called as reference variable

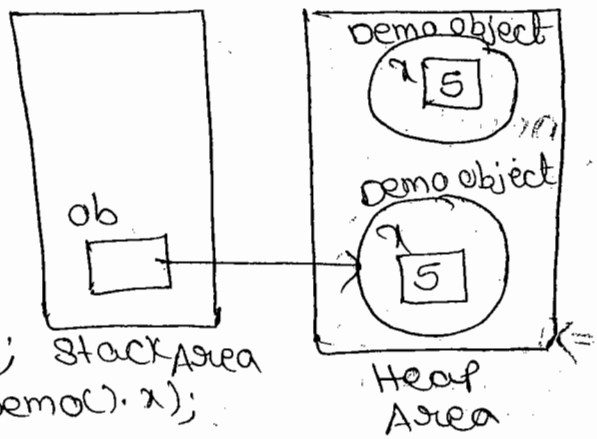
-> There are 2 ways to access Instance variables

1) By using object
2) By using object Reference

-> use object to access Instance variable if it is required only one time

-> use object reference to access Instance variable If it is required more than one time

```
class Demo
{
    int x=5;
    PSVM(String args[])
    {
        Demo ob=new Demo();
        System.out.println(ob.x);
        System.out.println(new Demo().x);
        System.out.println(ob.x);
    }
}
```



26/2/15

There are 4 ways to access class variables

1) Directly
2) By using class name
3) By using object
4) By using object reference

Example:-

```
class Demo
{
    static int x=5;
    public static void main(String args[])
    {
        System.out.println(x);
        System.out.println(Demo.x);
        System.out.println(new Demo().x);
        Demo ob = new Demo();
        System.out.println(ob.x);
    }
}
```

→ The following 2 ways are not recommended to access class variable

1) By using object
2) By using object reference

→ If we use above two the memory waste so as a programmer we should take care about memory and output

→ access class variable directly if it is present in the same class

→ use class name to access class variable if it is present in another class

```
class Test
{
    static int x=5;
}
class Demo
{
    static int y=10;
    public static void main(String args[])
    {
        System.out.println(x); X Error
        System.out.println(Test.x); ✓
        System.out.println(y); ✓
        System.out.println(Demo.y); ✓
    }
}
```

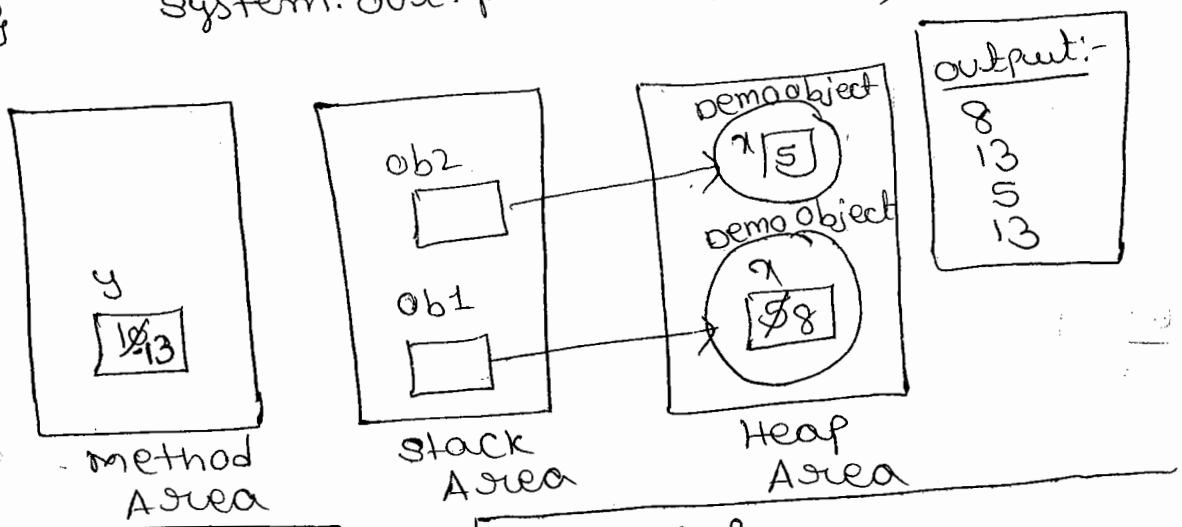
→ There is only one way to access local variable that is directly

```
class Demo
{
    public static void main(String args[])
    {
        int x=5;
        System.out.println(x);
    }
}
```

→ use class name to access class variable whenever both class variable and local variable names are same

Ex:-
```
class Demo
{
    static int x=5;
    public static void main(String args[])
    {
        int x=10;
        System.out.println(x); => 10
        System.out.println(Demo.x); => 5
    }
}
```

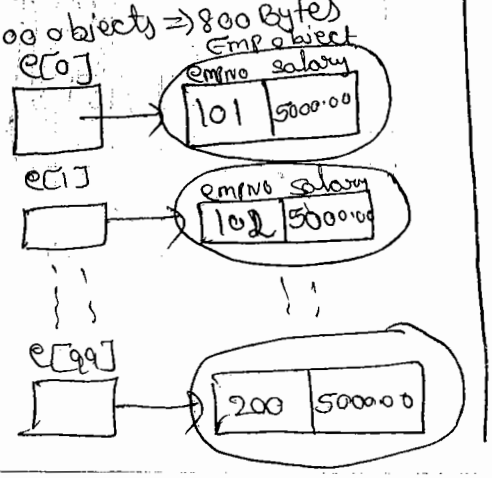
```
class Demo
{   int x=5;
    static int y=10;
    public static void main(String args[])
    {
        Demo ob1 = new Demo();
        Demo ob2 = new Demo();
        ob1.x = ob1.x+3;
        ob1.y = ob1.y+3;
        System.out.println(ob1.x);
        System.out.println(ob1.y);
        System.out.println(ob2.x);
        System.out.println(ob2.y);
    }
}
```




```
class Emp
{   int empno;
    Float salary;
}
```

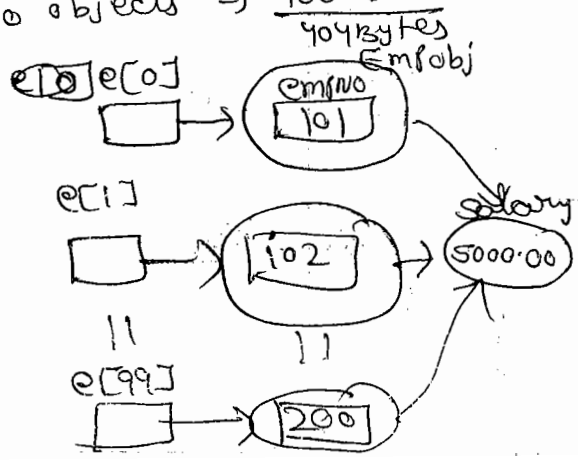
Here memory is wasted

100 objects => 800 Bytes




```
class Emp
{   int empno;
    static Float salary;
}
```

class variable => 4 bytes
100 objects => 400 Bytes
404 Bytes



→ use Instance variables if the values are different for all objects
→ use class variables if the values are same for all objects
→ use local variables to perform the task

27/2/15

Garbage :-

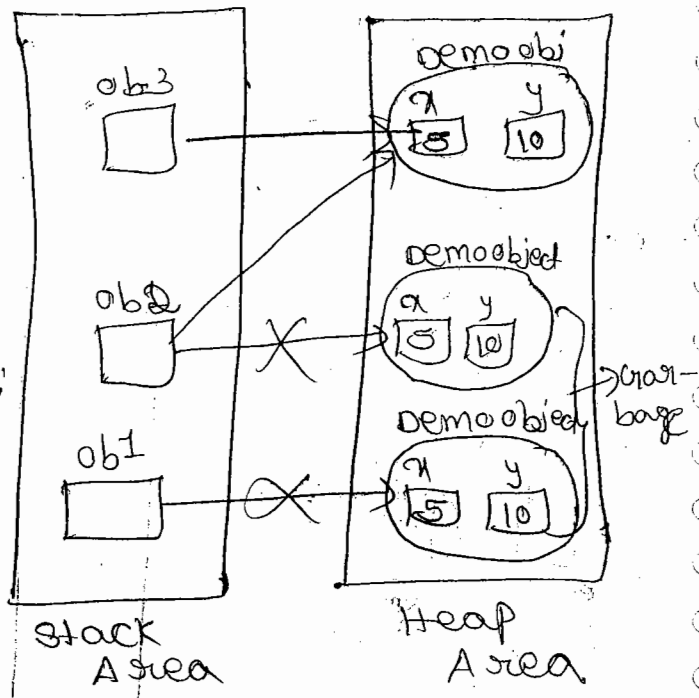
Garbage means unused objects.

Garbage collector :-

→ It is a JVM component and it collects the garbage whenever cpu gets free time because Garbage collector priority is least priority.
→ priority number is 1.
→ It is also possible to call garbage collector explicitly by using gc() method of java.lang.System class

Example:-

```
class Demo
{
    int x = 5;
    int y = 10;
    psvm(String args[])
    {
        Demo ob1 = new Demo();
        Demo ob2 = new Demo();
        Demo ob3 = new Demo();

        ob1 = null;

        ob2 = ob3;
    }
}
```



→ Local variables must be Intialized before access otherwise compile time error occurs bcoz They donot get default values.

Example:-

```
class Demo
{
    int x;
    static int y;
    public static void main(String args[])
    {
        int z;
        Sopln(new Demo().x); => 0
        Sopln(Demo.y); => 0
        Sopln(z); => Error
    }
}
```

Instance and class variable gets Default value 0.

Example:-

```
class Demo
{
    int x=5;        → Instance variable
    static int y=10;  → class variable
    psvm(String args[])   → method parameter
    {
        int z=15;   → Local variable
        for(int i=1; i<=10; i++)   → Block variable
        {
            ==
        }
    }
}
```

→ method parameters & Block variables are also called local variables

Variable	life
1) Instance variable	object (LLS-4)
2) class variable	class (LLS-5)
3) method parameter	method (LLS-3)
4) Local variable	method (LLS-2)
5) Block variable	Block (less life span)

28/2/15

```
class Demo
{
    int x=5;  -> Instance variable
    public static void main(String args[])
    {
        x++;
        x+=5;
        System.out.println(--x);
    }
}
```

O/P:- Error (Instance variable we can not access directly)

Example:-

```
class Demo
{
    static int x=5;
    public static void main(String args[])
    {
        x++;
        x+=5;
        System.out.println(--x);
    }
}
```

O/P:- 10

x 5 6 11 10

Methods :-

a group of statements into a single logical unit is called as methods

-> methods are used to perform the task

-> methods are divided into 4 categories

1) methods with arguments & with return value
2) methods with arguments & without return value
3) methods without arguments and with return value
4) methods without arguments and without return value

1) Methods with arguments and with return value :-

Syntax :-

ReturnType methodName(arg1, arg2, ----);

ReturnType → primitive data type (or) reference data type

{ ≡≡≡ } Task code

arg1, arg2 → primitive Types (or) Reference Types.

Examples :-

1) int max(int a, int b)
{ ≡≡ }

2) int cube(int a)
{ ≡≡ }

3) int[] sort(int[] a)
{ ≡≡ }

4) String append(String S1, String S2)
{ ≡≡ }

Example :-

```
class Demo
{
    int max(int a, int b)
    {
        if(a>b)
            return a;
        else
            return b;
    }
    psvm(String args[])
    {
        Demo ob = new Demo();
        int x = ob.max(56,23);
        System.out.println(x);
    }
}
```

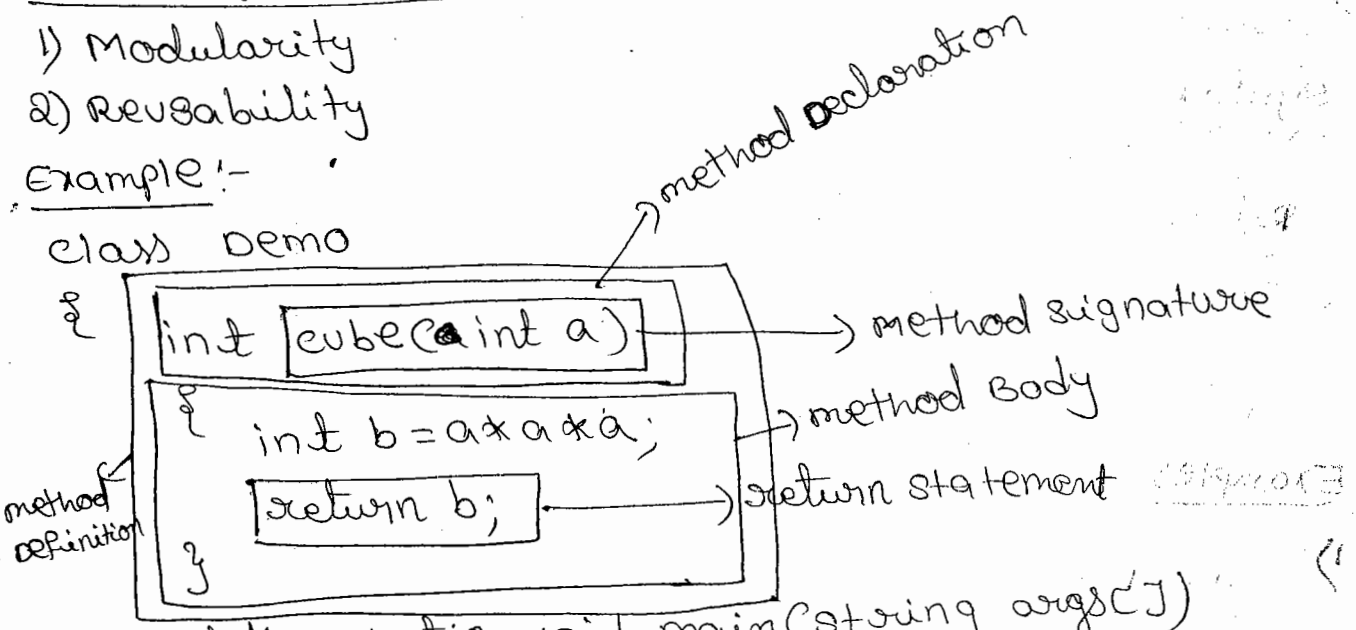
int x=a;

int a=56;
int b=23;

Advantages of methods:-

1) Modularity
2) Reusability

Example:-




```
class Demo
{
    int cube(int a)
    {
        int b = a*a*a;
        return b;
    }
    public static void main(String args[])
    {
        Demo ob = new Demo();
        int x = ob.cube(5);
        System.out.println(x);
    }
}
```

method call statement

mon 12/2/15

2) methods with arguments and without return value type:-

Syntax:-

```
ReturnType methodName(arg1, arg2, .....)
{
    ---------
    ---------
    ---------
}
```

ReturnType → Empty data type

(arg1, arg2, ...) → primitive Types (or) Reference Types

Task code

Examples:-

1)
```
void add(int a, int b)
{
    =
}
```

2)
```
void max(int a, int b, int c)
{
    =
}
```

3) void sort(int[] a)
{
==
}

4) void append(String s1, String s2)
{
==
}

Example:-

```
class Demo
{
    void max(int a, int b, int c)        // Instance method
    {
        if((a>b) && (a>c))
            System.out.println(a);
        else if(b>c)
            System.out.println(b);
        else
            System.out.println(c);
    }
    public static void main(String args[])
    {
        Demo ob = new Demo();
        ob.max(23,45,82);
    }
}
```

↳ by using object reference we are calling to instance method

Example:-

```
class Demo
{
    void add(int a, int b)
    {
        sopln(a+b);
    }
    psvm(String args[])
    {
        new Demo().add(45,62);
    }
}
```

→ with object we are calling to add()

3) methods without arguments and with return value:-

Syntax:-

ReturnType (primitive or referenced datatype) methodName()
{
    ≡≡ } task code
}

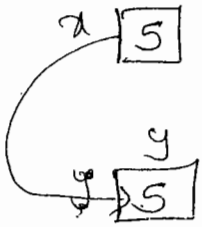
Examples:-

1) int getInt()
{
    ≡≡
}

2) int[] getArray()
{
    ≡≡
}

3) String getMessage()
{
    ≡≡
}

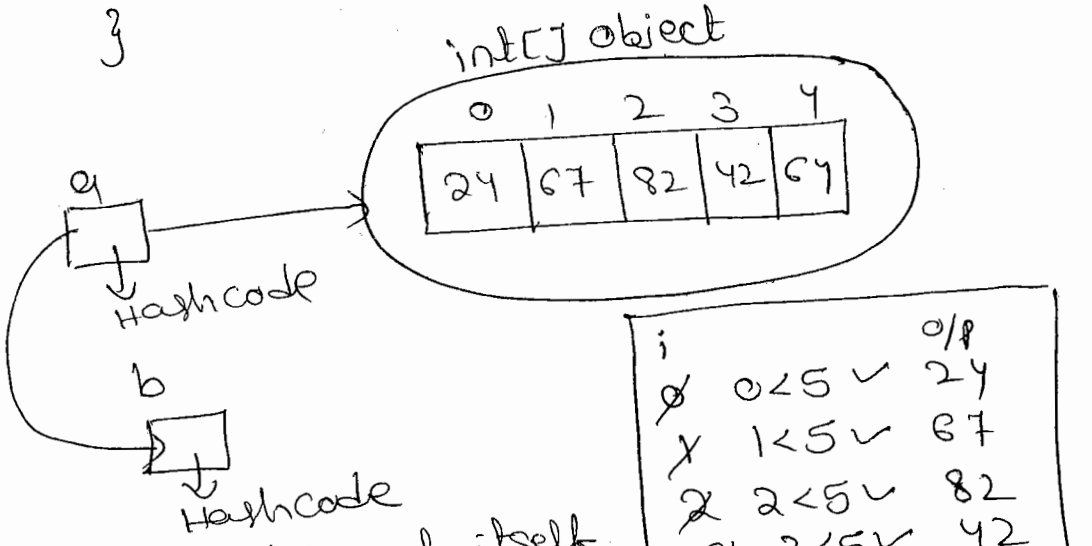
Example:-




```
class Demo
{
    int getInt()
    {
        int x=5;
        return x;
    }
    public static void main(String args[])
    {
        Demo ob = new Demo();
        int y = ob.getInt();
        System.out.println(y);
    }
}
```

→ Array itself an object in java
→ primitive type contains Data and Reference Type contain Hashcode

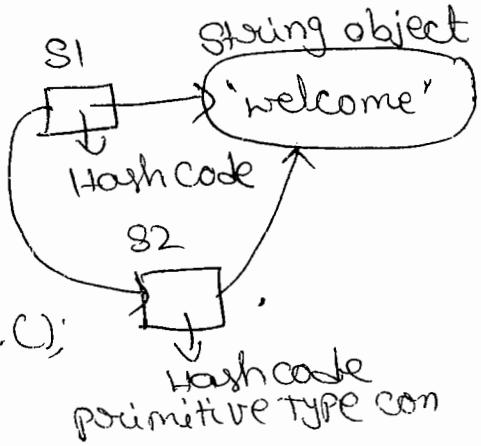
```
class Demo
{
    int[] getArray()
    {
        int a[] = {24, 67, 82, 42, 64};
        return a;
    }
    psvm(String args[])
    {
        Demo ob = new Demo();
        int[] b = ob.getArray();
        for(int i=0; i<b.length; i++)
        {
            sopln(b[i]);
        }
    }
}
```



i	o/p
0	0<5 ✓ 24
1	1<5 ✓ 67
2	2<5 ✓ 82
3	3<5 ✓ 42
4	4<5 ✓ 64
5	5<5 X

→ Every String literal itself an object of String class.

```
class Demo
{
    String getMessage()
    {
        String S1 = "welcome";
        return S1;
    }
    psvm(String args[])
    {
        Demo ob = new Demo();
        String S2 = ob.getMessage();
        Sopln(S2);
    }
}
```



4) methods without Arguments and without return value:-

Syntax:-

→ Empty data type void

```
ReturnType methodName()
{
    ==
}
```

Examples:-

```
void display()
{
    ==
}
```

```
void add()
{
    ==
}
```

Example:-

```
class Demo
{
    void display()
    {
        sopln("Hello");
    }
    PSVM(String args[])
    {
        Demo new
        Demo ob = new Demo()
            ob.display();
    }
}
```

3/3/15

Method overloading :-

If two or more methods with the same name and with different parameters list ~~and~~ then it is said to be method overloading.

In method overloading return type can be same (or) different

There are 3 ways to overload methods

1) Different in no. of arguments
2) Different in data types
3) Different in order of arguments

1) Different in no. of arguments

Example:-

```
void add(int a, int b){ }
void add(int a, int b, int c){ }
```

2) Different in data types:-

Example:-

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

3) Different in order of arguments :-

Example:-

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

Program to demonstrate method overloading:-

```
class Demo
{
    void add(int a, int b)
    {
        System.out.println(a+b);
    }
    void add(int a, int b, int c)
    {
        sopln(a+b+c);
    }
    psvm(String args[])
    {
        Demo ob = new Demo();
        ob.add(45, 23, 55);
        ob.add(42, 46);
    }
}
```

Method overriding:- (overriding)

If two or more methods with the same name and with the same parameters list then it is said to be method overriding

Example:-

```
void add(int a, int b){}
void add(int x, int y){}
```

NOTE:-

methods can't be overriden in the same class because of ambiguity to call

Ex:-

```
class Demo
{
    void add(int a, int b)
    {
        sopln(a+b);
    }
    void add(int x, int y)
    {
        sopln(x+y);
    }
    psvm(String args[])
    {
        Demo ob = new Demo();
        ob.add(2,3);
    }
}
```

✗ compiler gets confusion ambiguity to call.

```
class Demo
{   int a=5;
    void show()
    {   sopln(ob.a); => Error because ob is a local reference variable and it can not be accessed outside the main method
    }
    psvm(String args[])
    {   Demo ob = new Demo();
        ob.show();
    }
}
```

this keyword :-

It is called as an object reference or reference variable because it ~~referen~~ refers an object

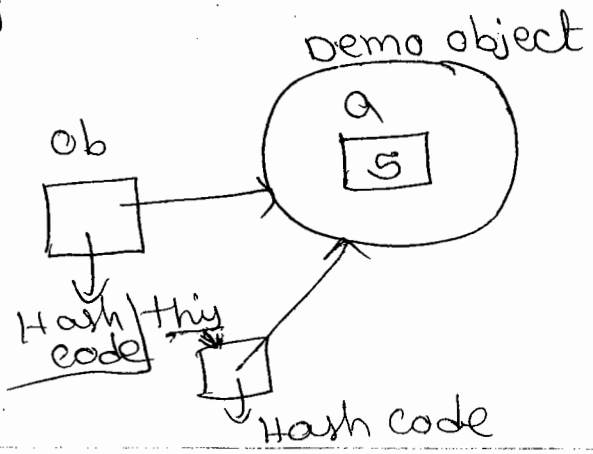
It always refers current object

Example1 :-

```
class Demo
{   int a=5;
    void show()
    {   sopln(a);
    }
    psvm(String args[])
    {   Demo ob= new Demo();
        ob.show();
    }
}
```

implicitly calls this keyword

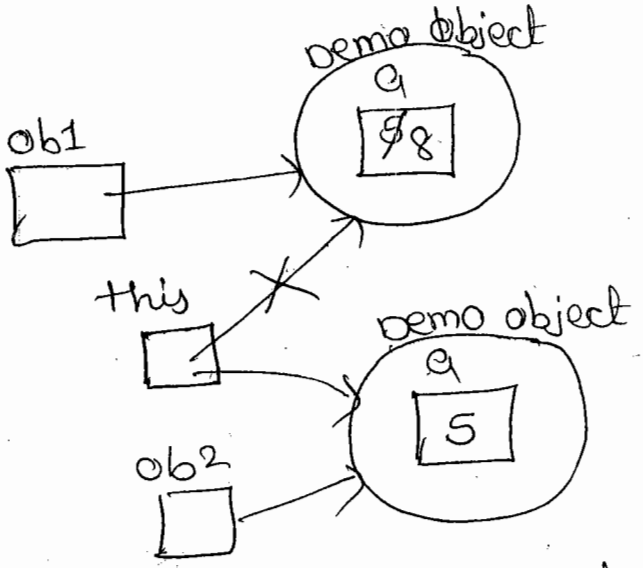
this.a



Example2:-

```
class Demo
{
    int a=5;
    void show()
    {
        sopln(a);
    }
    psvm(String args[])
    {
        Demo ob1 = new Demo();
        Demo ob2 = new Demo();
        ob1.a = ob1.a+3;
        ob1.show();
        ob2.show();
    }
}
```

implicitly → this.a



→ this keyword explicitly required to access an Instance variable whenever both Instance variable and local variable names are same

Example3:-

```
class Demo
{
    int a=5;
    void show()
    {
        int a=10;
        sopln(a); =>10
        sopln(this.a); =>5
    }
    PSVM(String args[])
    {
        Demo ob=new Demo();
        ob.show();
    }
}
```

-> local variable
-> local variable we can print in one way i.e directly

O/P:- 10
5

Example4:-

```
class Demo
{
    void show()
    {
        sopln("core java");
        print(); >> this.print -> implicitly
    }
    void print()
    {
        sopln("Advanced java");
    }
    PSVM(String args[])
    {
        Demo ob=new Demo();
        ob.show();
    }
}
```

Class method :-

a method that is defined with static keyword is known as class method.

There are 4 ways to access class methods.

1) Directly
2) by using object
3) By using object reference } → not recommended to use.
4) By using class name.

→ Access class method directly if it is present in the same class

→ use class name to access class method if it is present in ~~other~~ another class

Example :-

```
class Demo
{
    static void show()
    {
        sopln("welcome");
    }
    psvm(String args[])
    {
        show();
        Demo.show();
        new Demo().show();
        Demo ob = new Demo();
        ob.show();
    }
}
```

o/p :- welcome
welcome
welcome
welcome

→ use Instance methods if the class contains an Instance variables otherwise use class methods

→ use class method if the class does not containe instance variable

4/3/15

Static method does not refer this keyword in any way (Implicitly & explicitly both not possible)

Example:-

```
class Demo
{   int x=5;
    static void show()
    {   sopln(x); => Error because static method doesn't refer this keyword.
    }
    psvm(String args[])
    {   Demo ob = new Demo();
        ob.show();
    }
}
```

Static keyword:-

It is called as modifier because it modifies the behaviour of a variable, block, method and class

→ By using static keyword we can create class variables, Static Intialization blocks, class methods and nested top level classes. (one type of Inner class)

Constructors:-

a constructor is a special method which has same name as the class name and which has no return type.

→ constructor is called automatically whenever an object is created

→ constructors are used to Intialize Instance variables

→ constructors are two types
  1) Default constructors (without arguments)
  2) parameterized constructors (with arguments)

1) class Demo
{
  Demo()
  { == ✓
  }
}
constructor

2) class Demo
{
  demo()
  { == ✗
  }
}

3) class Demo
{
  void demo()
  { == ✓
  }
}

4) class Demo
{
  void Demo()
  { == ✓
  }
}

methods.

Example1:-

```
class Demo
{
    Demo()
    {
        sopln("core Java");
    }
    psvm(String args[])
    {
        new Demo();
    }
}
```

Example2:-

```
class Demo
{
    Demo()
    {
        sopln("core Java");
    }
    Demo(String s)
    {
        sopln(s);
    }
    psvm(String args[])
    {
        new Demo();
        new Demo("Advanced Java");
    }
}
```

Constructor overloading

Demo obj → not called bcoz i.e not a object

5/3/15

this();- this();

This statement calls default constructor of current class

this(.....);

This statement calls parameterized constructor of current class.

NOTE:-

this(); (or) this(.....); must be a first statement in a constructor.

Example1:-

```
class Demo
{   Demo()
    {  sopln("core Java");
    }
    Demo(String s)
    {
        this();
        sopln(s);
    }
    psvm(string args[])
    {  new Demo ("Advanced Java");
    }
}
```

o/p:- core Java
Advanced Java

Example2:-

```
class Demo
{   Demo()
    {  this("Advanced Java");
       sopln("core Java");
    }
    Demo(String s)
    {  sopln(s); }
    psvm(String args[])
    {
       new Demo();
    }
}
```

o/p:- Advanced Java
core Java

Instance variable Intialization without constructor

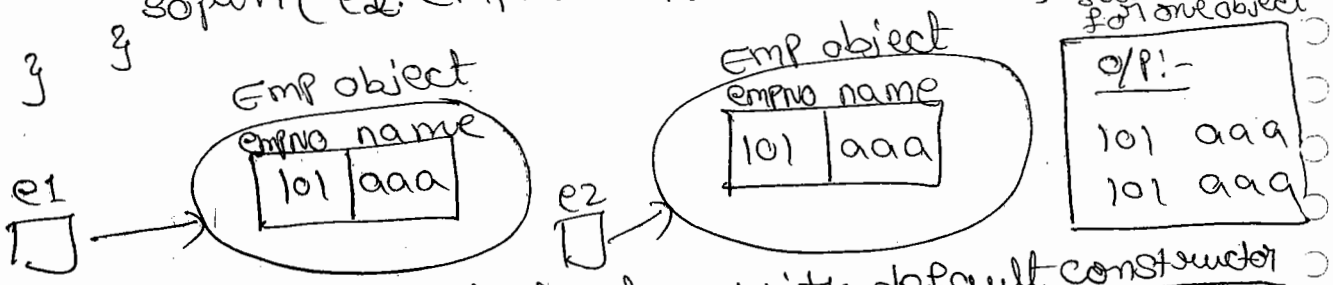
Example:-

```
class Emp
{
    int empNO = 101;
    String name = "aaa";
}
class Demo
{
    psvm(String args[])
    {
        Emp e1 = new Emp();
        Emp e2 = new Emp();
        sopln(e1.empNO + "\t" + e1.name);
        sopln(e2.empNO + "\t" + e2.name);
    }
}
```

This is not the right way to Intialize Instance variable if more then one object is present means do not go to this intialization. Intialization zation.

our requirement of instance variable is value is different for all objects but here we are getting same o/p for all objects we req different data so this is not the right way.

This is way of Intialization suitable only for one object



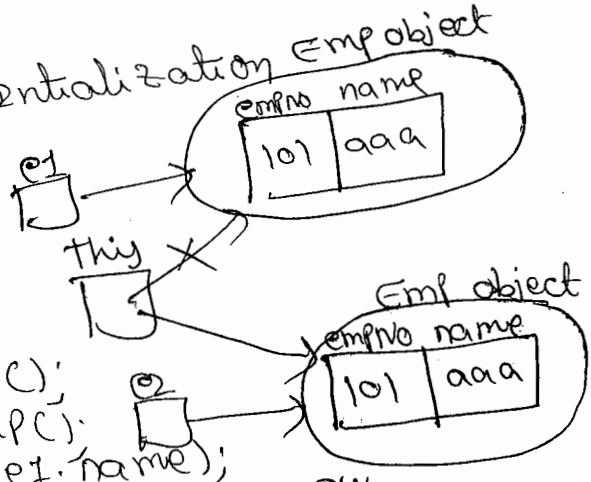
o/p:-
101 aaa
101 aaa

Instance variable Intialization with default constructor

```
class Emp
{
    int empNO;
    String name;
    Emp()
    {
        this.empNO = 101;
        this.name = "aaa";
    }
}
class Demo
{
    psvm(String args[])
    {
        Emp e1 = new Emp();
        Emp e2 = new Emp();
        sopln(e1.empNO + "\t" + e1.name);
        sopln(e2.empNO + "\t" + e2.name);
    }
}
```

Implicitly → this. this.

Intialization



o/p:- 101 aaa
101 aaa

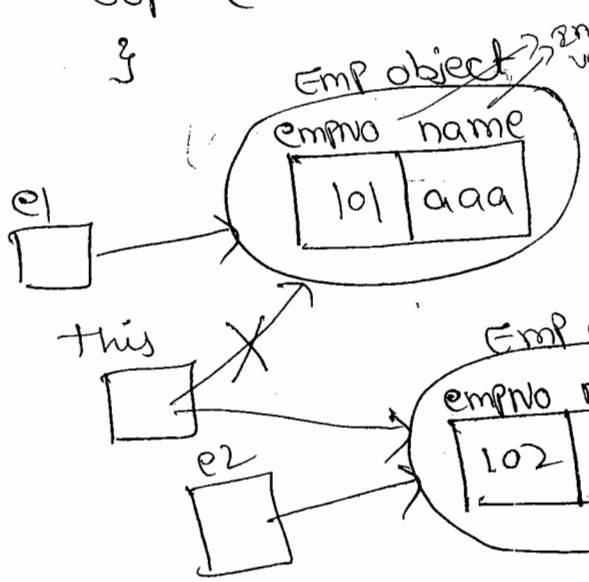
In the above example allocation of memory to Instance variables and calling constructor both will be done at a time whenever object is created

The above two examples are suitable if only one object is present

Instance variable Intialization with Parameterized constructor :

If the local variables & instance variables names are same then use this. implicitly explicitly

```
class Emp
{
    int empno;
    String name;
    Emp(int empno, String name)        // local variables
    {
        this.empno = empno;
        this.name = name;
    }                                  // Intialization. This is not assignment This is Intialization
}
class Demo
{
    psvm(String args[])
    {
        Emp e1 = new Emp(101, "aaa");
        Emp e2 = new Emp(102, "bbb");
        soplin(e1.empno+"\t"+e1.name);
        soplin(e2.empno+"\t"+e2.name);
    }
}
```



local variable
empno name
101 aaa

after coping local variables to object then local variables are vanished

local variable
empno name
102 bbb

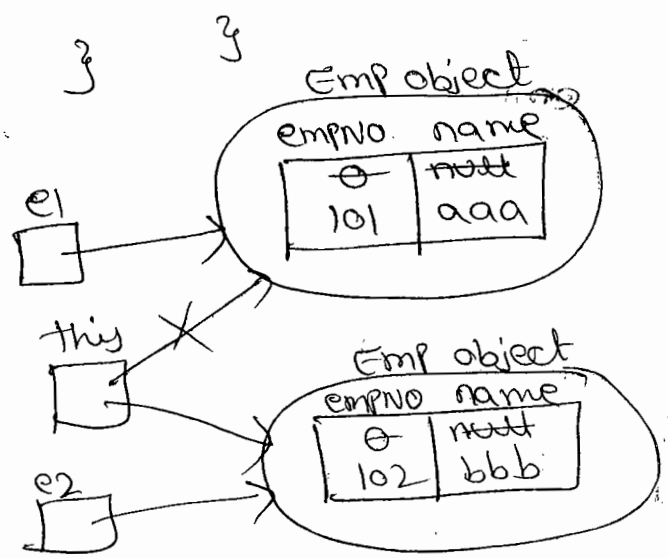
o/p:- 101 aaa
102 bbb

The above example is suitable for one object also and many objects also

Instance variable assignment with method:-

```
class Emp
{
  int empno;
  String name;
   void set(int empno, String name)
    {
       this.empno = empno;
       this.name = name;
    }
}
class Demo
{
   psvm(String args[])
    {
      Emp e1 = new Emp();
      e1.set(101, "aaa");
      Emp e2 = new Emp();
       e2.set(102, "bbb");
    sopln(e1.empno + "\t" + e1.name);
    sopln(e2.empno + "\t" + e2.name);
    }
}
```

} It is called as an Assignment.



→ The above example system implicitly provides one default Constructor to initialize 0 and null values to Instance variables

6/3/15
Fri

NOTE: System Implicitly provides one default constructor If the class does not contain any other constructor. then only

Example1:-

```
class Demo                    ✓valid (system provides default constructor)
{
  psvm(String args[])
   {
     new Demo();
   }
}
```

Example2:-

```
class Demo                    X Invalid (system does not provides parameterized constructor)
{
  psvm(String args[])
   {
     new Demo("hello");
   }
}
```

Example3:-

```
class Demo
{
   Demo(String s){}          valid.
   psvm(String args[])
   {
     new Demo("hello");
   }
}
```

Example4:- **

```
class Demo
{
    Demo(String s){}        X Invalid
    psvm(String args[])
    { new Demo();
    }
}
```

Block:-

→ a group of statements between open { and close } is called as block.

→ Blocks are also called Intialization Blocks. because Block also can be used to initialize instance variables.

→ one class can have any number of blocks and all those blocks are executed from top to bottom order whenever object is created.

→ Blocks are executed before constructor only.

Example:-

```
class Demo
{   Demo()
    { sopln("constructor"); }
    {
        sopln("Block1");
    }
    psvm(String args[])
    {
        sopln("main method");
        new Demo();
    }
    {
        sopln("Block2");
    }
}
```

O/P:-

```
main method
Block1
Block2.
Constructor.
```

Constructors, blocks are same thats why we are not using "blocks

Static Blocks:-

→ a block with static keyword is called as static block

→ static blocks are also called as static intialization blocks because it can be used to intialize static variables.

→ one class can have any number of static blocks and all those blocks are executed from top to bottom order whenever class is loaded.

→ static blocks are executed before main method only. Example

```
class Demo
{
    static
    { sopln("static block1");
    }
    psvm(String args[])
    { sopln("main method");
    }
    static
    { sopln("static block2");
    }
}
```

o/p:- static block1
static block2
main method

NOTE:-
static blocks are used in real time to load libraries

Differences between method and constructor.

Method	Constructor
i) a group of statements into a single logical unit is called as method.	i) a constructor is a special method which has ~~no~~ same name as class name and which has no return type.
ii) method can have same name as the class name or) different name	ii) constructor must be same name as a class name.
iii) method must contain return type.	iii) constructor must not contain return type.
iv) methods are used to perform the task	iv) constructors are used to Intialize Instance variables.
v) It must be explicitly called	v) constructor called automatically whenever object is created
vi) method can be called any no. of times with one object	vi) constructor can be called only one time with one object

Object oriented principles:-

1) Encapsulation
2) Abstraction
3) Inheritance
4) Polymorphism

Encapsulation:-

Binding of variables and methods into a class is known as an encapsulation
(or)
Encapsulation is a language construct that facilitates binding of variables with methods and those methods operating on same variables.

7/3/15

Abstraction:-

providing necessary information without including background details or)
providing necessary information and hiding unnecessary information.

Inheritance:-

creating a new class from an existing class is called as Inheritance.

Polymorphism:-

The ability to take more than one form

Poly means many and morphism means forms.

Inheritance:-

creating a new class from an existing class is called as Inheritance.

→ In Inheritance existed class is said to be super class and new class is said to be subclass.

→ USE Inheritance in the following situations

1) To add new features to existed class
2) To modify features of existed class

Advantages of Inheritance:-

1) Reusability
2) Extendability
3) modularity

Example:-

Student.java → superclass

```
class Student
{ int rollno;
  String name;
  String address;
  int landline;
  int landlineno;
  void accept()
  { =
  }
  void display()
  { sop(rollno);
    sop(name);
    sop(address);
  } } sop(landlineno);
```

Demo.java

```
class Demo
{
  psvm(S ar)
  {
    Student s = new Student2();
    s.accept();
    s.display();
  }
}
```

Student2.java

```
class Student2 extends Student
→ subclass
{
  int mobileno;
  String mail2d;
  void accept()
  { =
  }
  void display()
  { sop(rollno);
    sop(name);
    sop(mobileno);
    sop(mail2d);
  } }
```

→ In the above example new features are mobileno and mailId. reusable features are rollno and name
→ modified features are accept() & display()
→ Removed features are address and landLineNo.

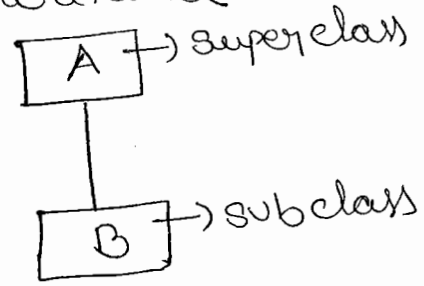
9/3/15 monday.

Types of Inheritance:-

1) Single Inheritance
2) Multiple Inheritance
3) Multilevel Inheritance
4) Hierarchical Inheritance
5) Multipath Inheritance
6) Hybrid Inheritance

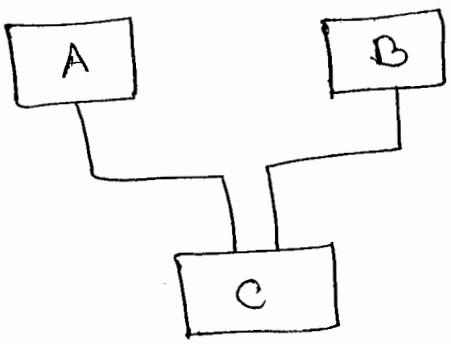
1) Single Inheritance:-

derivation of a class from only one superclass is called single Inheritance.

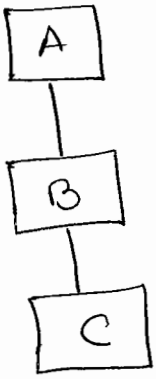


2) Multiple Inheritance:-

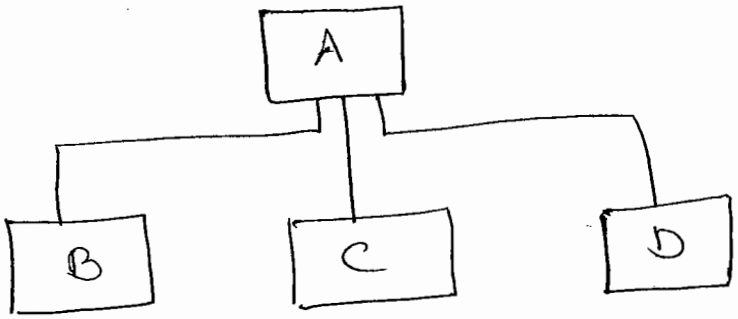
derivation of a class from more than one super class is called as multiple Inheritance



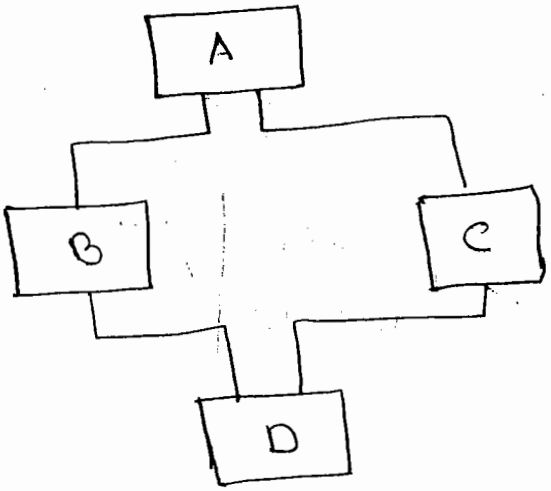
3) Multilevel Inheritance:-
derivation of a class from sub class is called as multilevel Inheritance.



4) Hierarchical Inheritance:
derivation of several classes from only one super class is called as Hierarchical Inheritance.



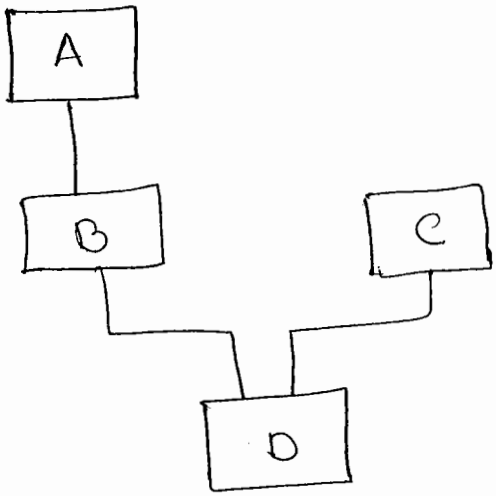
5) Multipath Inheritance:-
derivation of a class from more than one sub-class, those subclasses get Inherited from the same super class is called as multipath Inheritance.



A to D two paths are there via B & C thats why it is multipath

6) Hybrid Inheritance:-

Derivation of a class involving more than one form of Inheritance. is called as Hybrid Inheritance.



NOTE1:-

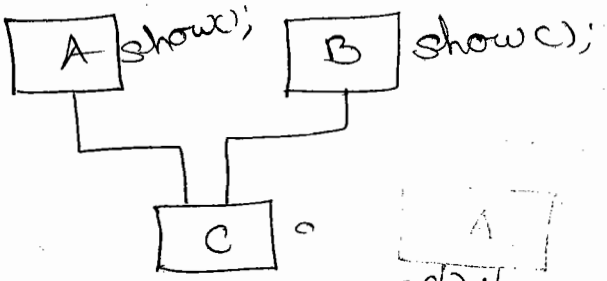
multipath Inheritance is also one example of Hybrid Inheritance

NOTE2:-

java doesnot support multiple Inheritance, multipath Inheritance and Hybrid Inheritance.

→ java doesn't support multiple Inheritance because of ambiguity

Example:-



C ob = new C();

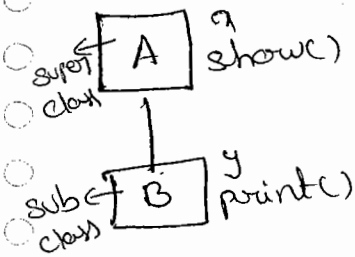
ob.show(); ⇒ this statement gets ambiguity to call either A class show() or B class show().

→ java doesn't support multipath Inheritance and Hybrid Inheritance because both types contain multiple Inheritance.

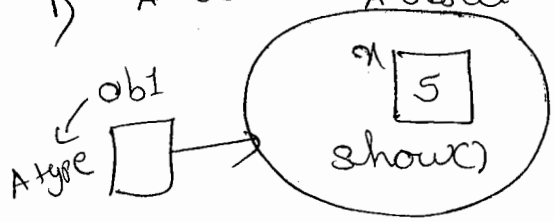
Example:-

```
class A
{
    int x=5;
    void show(){
    { soln(" A class");
     }
}
class B extends A
 {
     int y=10;
      void print()
      { soln(" B class");
       }
      psvm(String args[])
      {
          B ob = new B();
          Soln(ob.x);
          Soln(ob.y);
          ob.show();
          ob.print();
      }
}
```

op:- 5
10
A class
B class



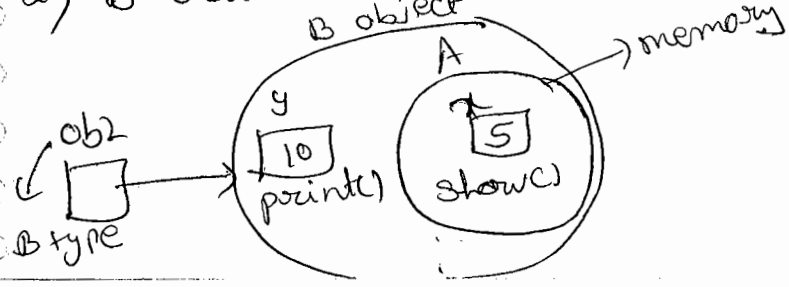
1) A ob1 = new A();



Here y & print() does not have memory how can we call

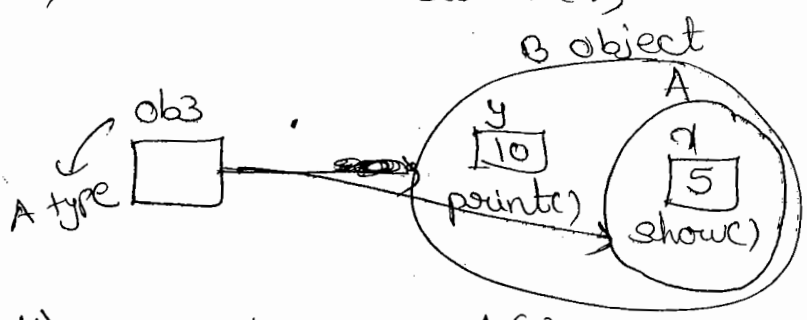
S.o.P(ob1.x); ✓
ob1.show(); ✓
S.o.P(ob1.y); X
ob1.print(); X

2) B ob2 = new B();



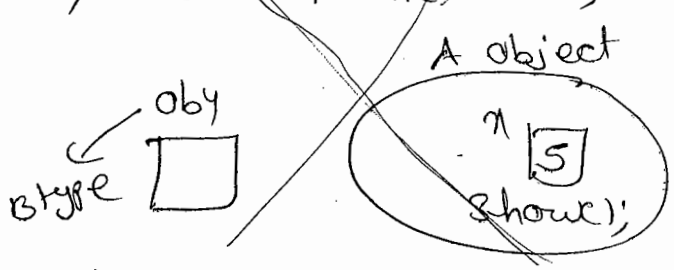
S.o.P(ob2.x); ✓
ob2.show(); ✓
S.o.P(ob2.y); ✓
ob2.print(); ✓

3) A ob3 = new B(); ✓



S.o.p(ob3.x); ✓
ob3.show(); ✓
S.o.p(ob3.y); X
ob3.print(); X

4) B ob4 = new A();



This is not possible, Illegal error occurs.

→ whenever ~~a~~ Superclass object is created then memory allocated to only superclass members.

→ whenever subclass object is created then memory allocated to both superclass members and subclass members

→ Superclass object reference refers both superclass object and subclass object but it can access only superclass members

A ob = ——— ; new A(); or new B();

~~→ subclass object refers only sub class~~

→ subclass object reference refers only subclass object but it can access both superclass members and subclass members

B ob = ——— ; new B();

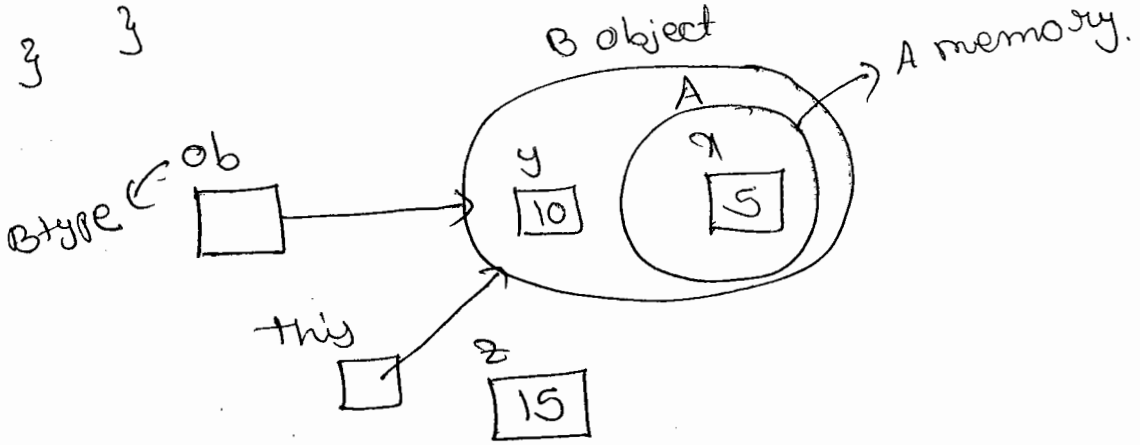
Example:-

```
class A
{  int x=5;
}
class B extends A
{  int y=10;
   void print()
   {  int z=16;
      System.out.println(x);
      System.out.println(y);
      System.out.println(z);
   }
public static void main(String args[])
{  B ob = new B();
   ob.print();
}  }
```

implicitly.

this.x

this.y



10/3/15 TUE

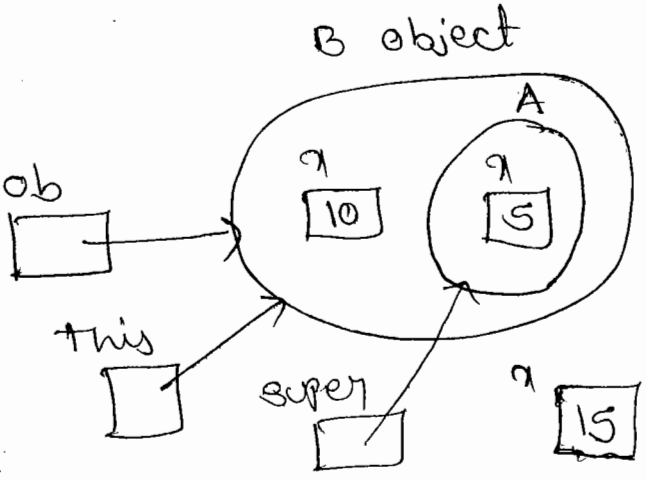
Super Keyword :-

It is called as an object reference or reference variable because it refers super class memory.
It is explicitly required to access super class member (variable or method) whenever both super class member and sub class member names are same

Example1 :-

```
class A
{   int x=5;
}
class B extends A
{
    int x=10;
    void show()
    {   int x=15;
        system.out.println(x);
        system.out.println(this.x);
        system.out.println(super.x);
    }
    public static void main(String args[])
    {
        B ob = new B();
        ob.show();
    }
}
```

o/p :-
15
10
5



Example2:-

```
class A
{
    void print()
    {
        System.out.println("A class");
    }
}
class B extends A
{
    void print()
    {
        System.out.println("B class");
    }
    void show()
    {
        print();
        this.print();
        super.print();
    }
    public static void main(String args[])
    {
        B ob = new B();
        ob.show();
    }
}
```

print(); → this.print (implicitly.)

main method also static method

o/p:-
B class
B class
A class

NOTE:- ***
Static method does not refer this and super keywords many where (implicitly & explicitly)

super(); :-
This statement calls default constructor of super class

super(....); :-
This statements calls parameterized constructor of super class

NOTE1:-

super(); implicitly comes in every constructor

NOTE2:-

this();/this(....);/super();/super(....); must be a first statement in a constructor.

Example:-

```
class A
{
    A()
    {
        system.out.println("A class");
    }
}
class B extends A
{
    B()
    {
        super();   <-- implicitly
        System.out.println("B class");
    }
    public static void main(String args[])
    {
        new B();
    }
}
```

O/P:- A class
B class

Example:-

```
class A
{   A(String s)
    {   sopln(s);
    }
}
class B extends A
{   B(String s)
    {   super();   <-- implicitly
        sopln(s);
    }
    psvm(String args[])
    {   new B("core java");
    }
}
```

Error: because there is no default constructor in A class.

There are 3 solutions for above example

1) writing default constructor in A class

Example:-

```
class A
{
    A()
    { system.out.println("java");
    }
    A(String s)
    { system.out.println(s);
    }
}
class B extends A
{
    B(String s)
    { System.out.println(s);
    }
    psvm(String args[])
    { new B("core java");
    }
}
```

implicitly → super();

o/p:- java
corejava

2) Removing parameterized constructor from super class

```
class A
{}
class B extends A
{
    B(String s)
    { sopln(s);
    }
    psvm(String args[])
    { new B("core java");
    }
}
```

implicitly → super();

o/p:- core java

3) writing super(.....); in subclass constructor

Example:-

```
class A
{   A(String s)
    {   sopln(s);
    }
}
class B extends A
{
    B(String s)
    {
        super(s);
        sopln(s);
    }
    psvm(String args[])
    {   new B("core java");
    }
}
```

o/p:- core java
core java

Example:-

```
class A
{   A(String s){}
}
class B extends A
{
}
```

Error: because there is no default constructor.

11/3/15

```
Class A
{
    A()
    {
        this("JavaME");
        System.out.println("Core Java");
    }
    A(String s)
    {
        System.out.println(s);
    }
}
class B extends A
{
    B()  --> implicitly super();
    {
        System.out.println("Advanced Java");
    }
    B(String s)
    {
        this();
        System.out.println(s);
    }
    public static void main(String args[])
    {
        new B("JavaEE");
    }
}
```

o/p: JavaME
CoreJava
AdvancedJava
JavaEE

Final Keyword:-

It is called as modifier because it modifies the behaviour of a variable, method and class
It is used to prevent a value of the variable, method overriding and Inheritance

Example:- 1) final int x = 5;
x++ X Error because final values can not be changed.

→ final keyword can be applied to Instance variables, class variables and local variables.

→ final variable must be Intialized otherwise compile time error occurs.

Example:-

```
class Demo
{
    int x;
    public static void main(String args[])
    {
        System.out.println(new Demo().x);
    }
}
```

O/P:- 0

Example:-

```
class Demo
{
    final int x;
    public static void main(String args[])
    {
        System.out.println(new Demo().x);
    }
}
```

O/P:- Error; because final variable must be Intialized before access otherwise ~~compiletime~~ error occurs

Example:-

```
class Demo
{
    final int x;
    void show();
    {
        x=0;
    }
    psvm(String args[])
    {
        Demo ob = new Demo();
        ob.show();
        System.out.println(ob.x);
    }
}
```

→ Can not assign a value to a final variable

→ It is called as an assignment

Error because final variable must be Intialized.

Example:-

```
class Demo
{
  final int x;
   Demo()
   {
      x=0;
   }
   psvm(String args[])
   {
      Demo ob = new Demo();
      System.out.println(ob.x);
   }
}
```

This is constructor so valid method is different constructor is different

⇒ It is called Intialization

o/p:- 0

Example:-

```
class A
{
   final void show()
      {
         sopln("A class");
      }
}
class B extends A
{
      void show()
      {
         sopln("B class");
      }
}
```

Error because final methods can not be overridden

method overriding.

Example:-

```
final class A
{
}
class B extends A
{
}
```

✗ Error because final classes can not be Inherited.

Access modifiers :-

Access modifiers are also called access specifiers because they specifies access permission to variables, method, class and Interfaces ... etc.

There are 4 access modifiers in java

1) private
2) protected
3) public
4) no name. (No Access modifier)

noname is a default (AM) in a package
noname is a default in a class
public is a default in an Interface

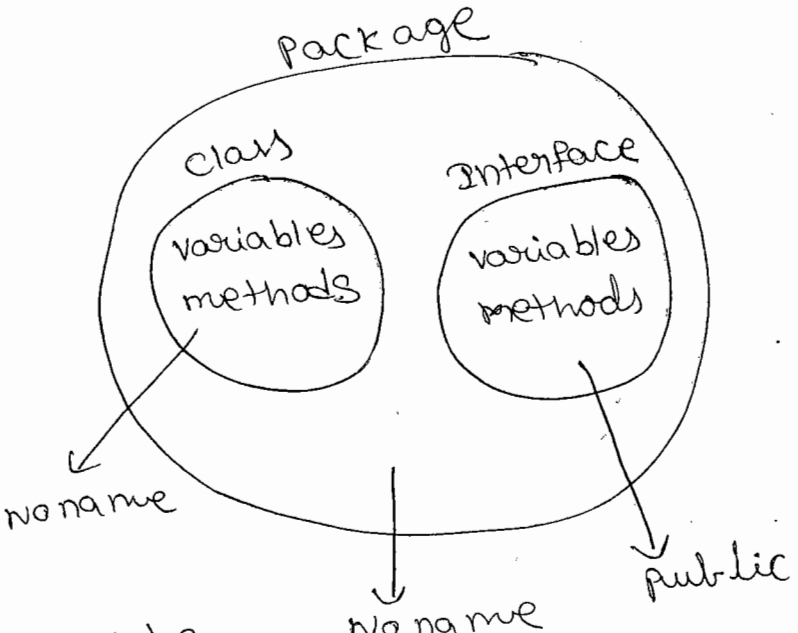
Access modifier (AM)

Package :-

Package is a collection of subpackages, classes and Interfaces

Example :-

```
Package abc;
noname ← class A
       {
noname ← int a = 5;
       }
noname ← interface B
       {
public ← int b = 10;
       }
```



→ In java class can not be private and can not be protected

```
private class A{}  X
protected class B{}  X
public class C{}  ✓
class D{}  ✓
```

→ Interface also can not be private and can not be protected.

```
private interface A{} X
protected interface B{} X
public interface C{} ✓
class D{} ✓
```

```
Package abc;
public class A
{
    int a=5;
    private int b=10;
    protected int c=15;
    public int d=20;
}
class B extends A
{
    ==
}
class C
{
    ==
}
```

scope

Package ← a → A B C → 3 classes

class ← b → A → 1 class

package & subclass ← c → A B C D → 4 classes

(E is outside the package) ← d → A B C D E → 5 classes

visible to

```
Package xyz;
class D extends A
{
    ==
}
class E
{
    ==
}
```

NO restriction

12/3/15

Relationships in Java :-

1) "IS-A" Relationship
2) "HAS-A" Relationship

"IS-A" Relationship VS "HAS-A" Relationship

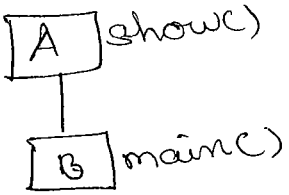
IS-A Relationship

i) IS-A Relationship refers Inheritance

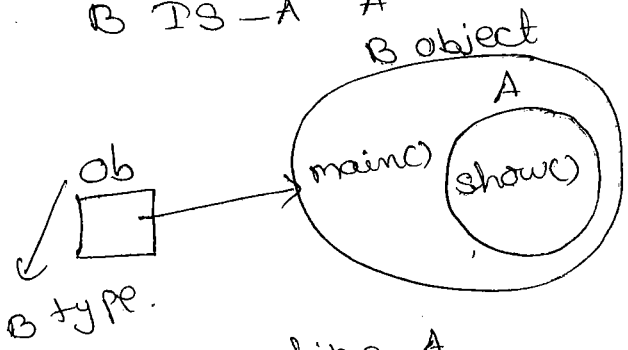
Example:-

```
Class A
{
    void show()
    {
        S.o.P("welcome");
    }
}
Class B extends A
{
    PSVM(String args[])
    {
        B ob = new B();
        ob.show();
    }
}
```

Here only one object created



B "IS-A" A



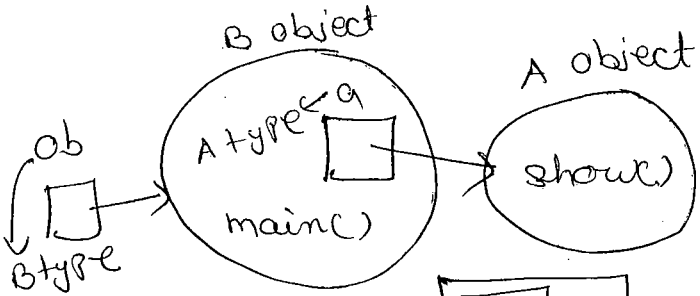
B working like A

In class A If we want to call so method we have to create A class object but here we are calling with B class object why bcoz there is a relation b/w B & A-

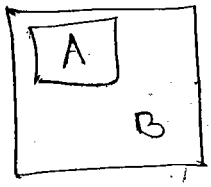
"HAS-A" Relationship

i) HAS-A Relationship refers Composition

```
Class A
{
    void show()
    {
        Sop("welcome");
    }
}
Class B
{
    A a = new A();
    PSVM(String args[])
    {
        B ob = new B();
        ob.a.show();
    }
}
```



B "HAS-A" A



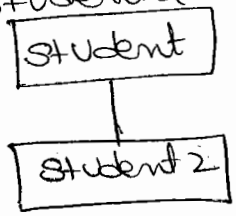
B class has a A class.

We can create objects outside the main also

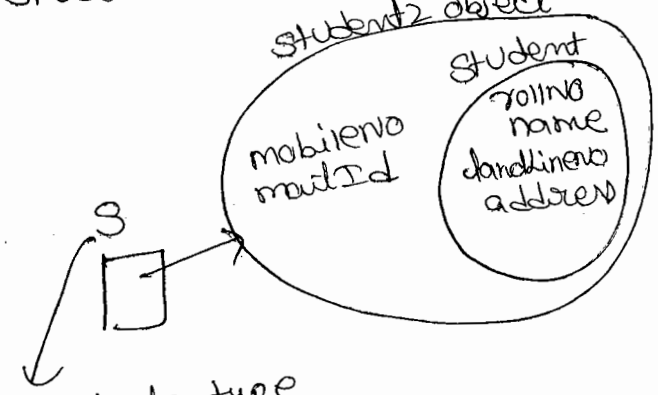
Here a is the member of B and a referes class A

In the above example two objects created

```
class Student
{ int rollno;
  String name;
  int landlineno;
  String address;
}
class Student2 extends Student
{ int mobileno;
  String mailId;
}
Student2 s = new Student2();
```



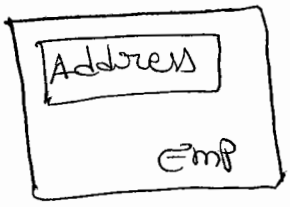
Student2 "IS-A" Student

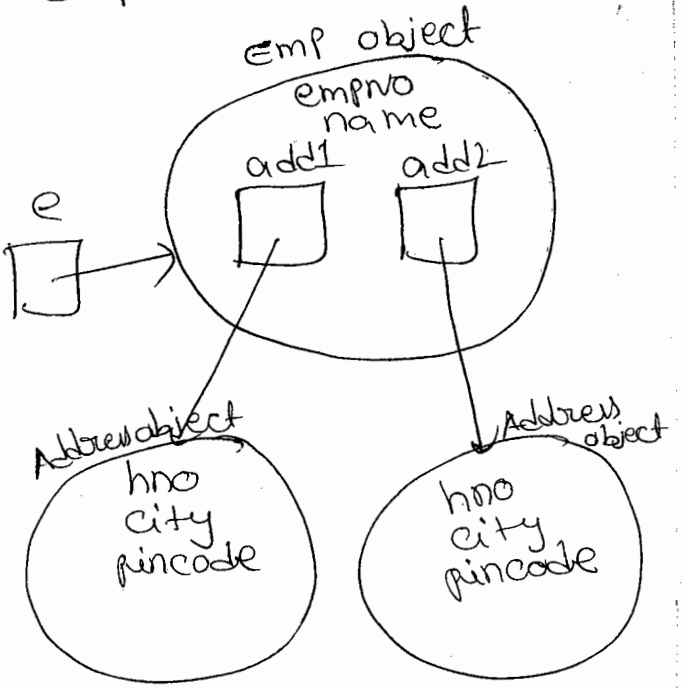


Student2 type

Student2 s = new Student2();

```
class Address
{ String hno;
  String city;
  int pincode;
}
class Emp
{ int empno;
  String name;
Address add1 = new Address();
Address add2 = new Address();
}
Emp e = new Emp();
```





Emp "HAS-A" Address.

Polymorphism:-

The ability to take more than one form

Poly means many and morphism means forms.

There are 2 types of polymorphism

1) Compile time polymorphism (static polymorphism)
2) Runtime polymorphism (dynamic polymorphism)

1) Compile time polymorphism:-

The binding of method call statement with method definition is done at compile time is known as compile time polymorphism.

It is also called as static polymorphism or) static binding. (or) early binding

Ex:- method overloading.

method overloading:-

If two or more methods with the same name and with different parameters list then it is said to be method overloading.

The parameter list difference can be a no. of arguments, datatypes or order of an arguments

~~not~~ In method overloading return type can be same or different

Example:-

```
class Demo
{   int max(int a, int b)
    {   if(a>b)
            return a;
        else
            return b;
    }
    int max(int a, int b, int c)
    {   if((a>b)&&(a>c))
            return a;
        else if(b>c)
            return b;
        else
            return c;
    }
    psvm(String args[])
    {   Demo ob = new Demo();
        int x = ob.max(23,45);
        sopln(x);
        int y = ob.max(45,88,34);
        sopln(y);
    }
}
```

method definitions

method call statements

Runtime polymorphism:-

Binding of method call statement with method definition is done at runtime is known as runtime polymorphism It is also called as dynamic polymorphism or dynamic binding. (or) late binding.

Ex:- method overriding.

Method overriding:-

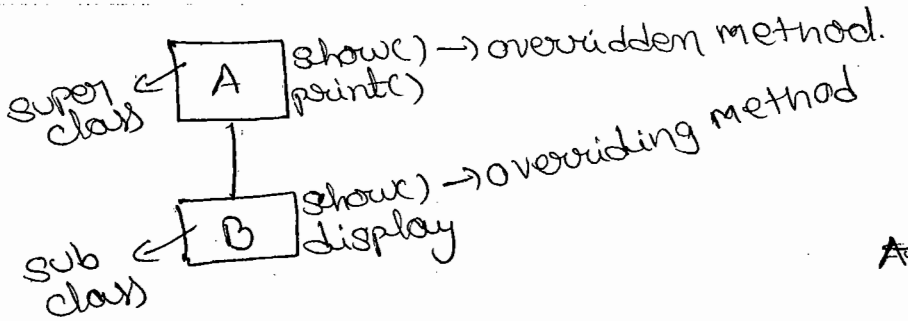
If two or more methods with the same name and with the same parameters list then it is said to be method overriding.

methods can not be overidden in the same class because of ambiguity.

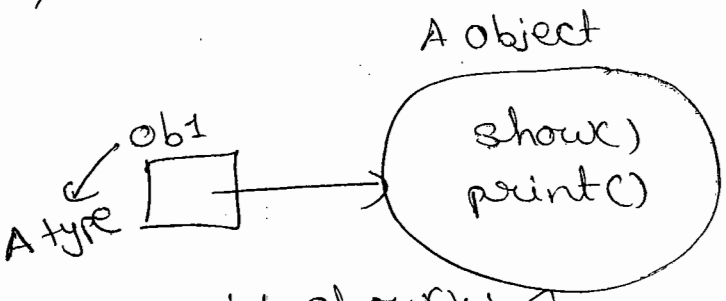
methods can be overridden in only in Inheritance

13/3/15

```
class A
{
    void show()
    { soln("A class");
    }
    void print()
    { soln("print() method");
    }
}
class B extends A
{
    void show()
    { soln("B class");
    }
    void display()
    { soln("display() method");
    }
psvm(String args[])
{
    B ob = new B();
    ob.show();
    ob.print();
    ob.display();
}
```

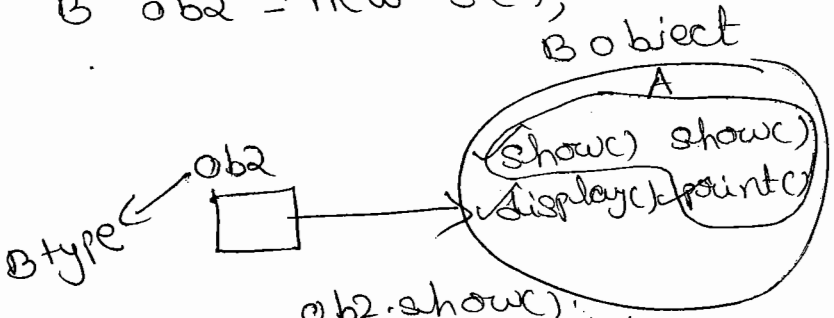


1) A ob1 = new A(); ✓



ob1.show(); ✓
ob1.print(); ✓
ob1.display(); X

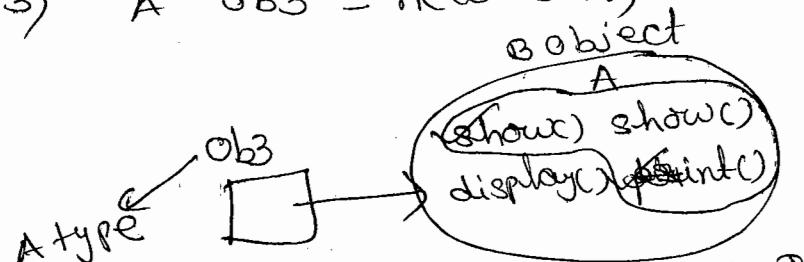
2) B ob2 = new B(); ✓



ob2.show(); ✓
ob2.print(); ✓
ob2.display(); ✓

It calls B class show().

3) A ob3 = new B(); ✓



ob3.show(); ✓ → It prints B class show() it is overriding A class by B class
ob3.print(); ✓
ob3.display(); X

4) B ob4 = new A(); X

NOTE!-

According to Sun microsystem specification no compile time polymorphism in java because objects are created by using new operator (new is called as Dynamic memory allocation operator)

Method overloading vs method overriding

method overloading

i) If two or more methods with the same name and with different parameters list then it said to be method overloading

ii) methods can be overloaded in the same class and also it can be overloaded in Inheritance

iii) In method overloading method name must be same and parameters list must be different

iv) In method overloading return type can be same or different

v) In method overloading access modifiers can be same or different

vi) final methods can be overloaded

vii) static methods can be overloaded

method overriding

i) If two or more methods with the same name and with the same parameters list then it is said to be method overriding.

ii) methods can not be ~~overridden in same class~~ because of ambiguity it ~~must~~ can be overridden in Inheritance only

iii) In method overriding method names must be same and parameters list also must be same.

iv) In method overriding return type must be same except covariant Return type.

v) In method overriding access modifiers can be same (or) can be less restrictive. (must not be more restrictive)

vi) final methods can not be overridden because final keyword used to prevent method overriding

vii) static methods can not be overridden because static members are not a part of an object

viii) private methods can't be overloaded.

viii) private methods can not be overridden because private members are can not be inherited.

14/3/15

Covarient Return Type:-

Java permits subclass type as a return type while overriding a method. this is known as covarient return Type.

```
class A
{
   int get()
   {
   ==
   }
}
class B extends A
{
   float get()
   {
   ==
   }
}
```

X

```
class A
{
   A get()
   {
   ==
   }
}
class B extends A
{
   B get()
   {
   ==
   }
}
```

A is superclass
B is subclass

Here B is a subclass of A ✓

Covarient Return Type.

that's why Here B as return type valid i.e covarient Return type

private → no name → protected → public
↓ more Restrictive Access modifier

↓ Less Restrictive Access modifier

public → protected → no name → private
↓ Less Restrictive Access modifier

↓ more Restrictive Access modifier.

→ In super class private is there ~~sub~~ overridden is not possible (Overriding)

```
class A
{
  protected void show(){}
}
class B extends A
{
  protected void show(){}
    or
    public
}
```

→ In super class protected is there in ~~max~~ subclass both protected and public are ~~possible~~ valid.

→ In super class noname access modifier is there in sub class noname, protected and public are valid.

1)
```
class A
{
  final void show()
  {
    ==
  }
}
class B extends A
{
  void show()
  {
    ==
  }
}
```
X

2)
```
class A
{
  final void show()
  {
    ==
  }
}
class B extends A
{
  final void show()
  {
    ==
  }
}
```
X

→ final method can not be overridden. because final keyword is used to prevent overriding

3)
```
class A
{
  void show()
  {
    ==
  }
}
class B extends A
{
  final void show()
  {
    ==
  }
}
```
✓

In super class there is no final so we can overridden but subclass final is there so we can not overridden in further classes.

```
class A
{
    void show() → overridden method
    {
        sopln("A class");
    }
}
class B extends A → This method belongs to A class
{                   This method have two definitions
    void show() → overriding method
    {
        sopln("B class");
    }
    psvm(String args[])
    {
        A ob = new B();
        ob.show();
    }
}
```

1)
```
class A
{
    static void show()
    {
        =
    }
}
class B extends A
{
    void show()
    {
        =
    }
}
```
→ It is not valid bcoz static method can not be overridden

2)
```
class A → It is method hiding.
{
    static void show()
    {
        =
    }
}
class B extends A
{
    static void show()
    {
        =
    }
}
```
valid but not overriding

3)
```
class A
{
    void show()
    {
        =
    }
}
class B extends A
{
    static void show()
    {
        =
    }
}
```
It is not valid bcoz overriding method can not be static

program to demonstrate method hiding:-

```
class A
{   static void show()
    {   sopln(" A class");
    }
}
class B extends A
{
    static void show()
    {   sopln(" B class");
    }
    PSVM(String args[])
    {   A ob = new B();
        ob.show();
    }
}
```

O/p : A class

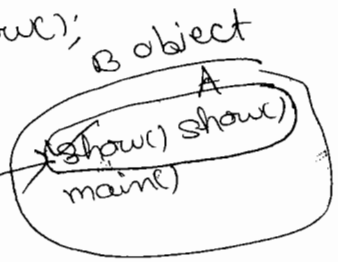
It is not an overriding

This concept is called method hiding because one method is available to reference and one more method is hidden

→ In the above example one method is available to object reference and one more method is hidden. this is known as method hiding.

```
class A
{   void show()
    {   =
    }
}
class B extends A
{   void show()
    {   =
    }
    PSVM(String args[])
    {   A ob = new B();
        ob.show();
    }
}
```

valid (it is an overriding)




```
class A
{   static void show()
    {   =
    }
}
class B extends A
{   static void show()
    {   =
    }
    PSVM(String args[])
    {   A ob = new B();
        ob.show();
    }
}
```

valid but not overriding.

1) class A
{
private void show()
{ = }
}
class B extends A
{
void show()
{ = }
}

this method is different B class method is different why bcoz private scope is within the class only.

valid but not overriding

2) class A
{
private void show()
{ = }
}
class B extends A
{
private void show()
{ = }
}

valid but not overriding.

3) class A
{
void show()
{ = }
}
class B extends A
{
private void show()
{ = }
}

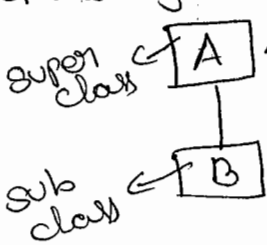
✗ not valid becoz in super class noname so subclass must be noname, protected, public

16/3/15

Upcasting :-

assigning an object (or) object reference of subclass to superclass type is known as upcasting

Upcasting will be done by system implicitly



1) A ob = new B(); ✓ valid upcasting

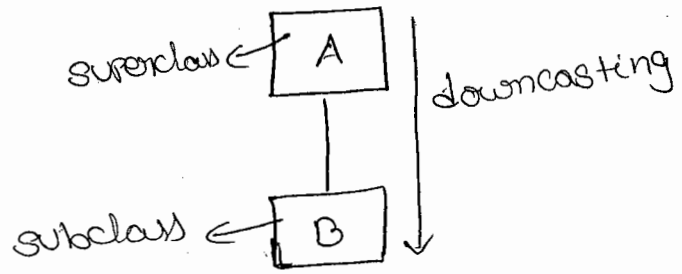
2) B ob1 = new B();
A ob2 = ob1; ✓ valid upcasting

Downcasting :-

assigning an object or object reference of superclass to subclass type is known as downcasting

→ downcasting must be done by programmer explicitly

→ Downcasting always needs upcasting



1) `B ob = new A();` ✗ Invalid downcasting (There is no B memory)

2) `A ob1 = new A();` ✓
`B ob2 = ob1;` ✗ Invalid downcasting (There is no B memory we have to assign)

3) `B ob = (B)new A();` ✗ [Here no upcasting]
↳ Explicit type conversion

4) `A ob1 = new A();` ✓
`B ob2 = (B)ob1;` ✗ Invalid downcasting [Here no upcasting]

5) `A ob1 = new B();` ✓ valid upcasting [Here Explicit type conversion not present]
`B ob2 = ob1;` ✗ Invalid downcasting

6) `A ob1 = new B();` ✓ valid upcasting
`B ob2 = (B)ob1;` ✓ valid downcasting

→ upcasting required first then only downcasting possible and Explicit type conversion required then only downcasting valid

→

abstract class :-

a class that is declared with abstract keyword is known as an abstract class

→ abstract class can have only abstract methods, only non-abstract methods (concrete methods) or both abstract and non-abstract methods

abstract method :-

a method which has no body is called as an abstract method

abstract method must be declared with abstract keyword in java otherwise compile time error occurs

Example :- abstract void show();

concrete method :-

a method with body is called as concrete method

Examples :- 1) void show()
{

}

concrete methods

2) void print(){}

↓

It is also called null body method

→ If the class contains an abstract method then the class must be declared with abstract keyword otherwise compile time error occurs.

1) abstract class A ✓
{
abstract void show();
void print(){}
}

2) abstract class A ✗
{
void show();
}

3) abstract class A ✓
{ abstract void show();
abstract void print();
}

4) class A ✗
{ abstract void print();
}

5) abstract class A ✓
{
void show(){}
void print(){}
}

→ abstract classes can not be Instanciated
→ abstract class can be Inherited into another class by using extends keyword
→ whenever abstract class is Inherited then all abstract methods of an abstract class must be overridden in a subclass or) subclass must be declared with abstract keyword otherwise compile time error occurs.

17/3/15

Example:-

```
abstract class A
{
   abstract void show();
    void print()
    {
      system.out.println("print() method");
    }
}
class B extends A
{
     void show()
     {
       system.out.println("show() method");
     }
     void display()
     {
       system.out.println("display() method");
     }
     public static void main(String args[])
     {
          B ob = new B();
            ob.show();
            ob.print();
            ob.display();
     }
}
```

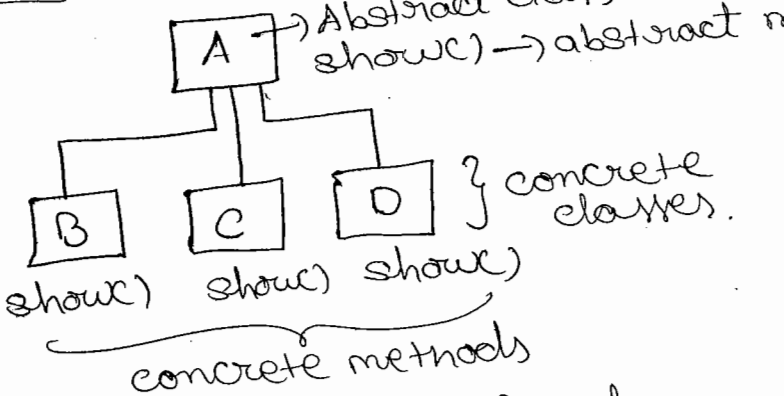
1) A ob1 = new A(); ✗ abstract class can not be Instanciated

2) B ob2 = new B(); ✓ → It is valid
ob2.show(); ✓
ob2.print(); ✓
ob2.display();

3) A ob3 = new B(); ✓ → By using this we can access only A class members
ob3.show();
ob3.print();
ob3.display(); ✗ → we can not access B members

4) B ob4 = new A(); ✗ abstract class can not be Instanciated
↓ Downcasting not possible.

→ abstract method allows to declare a method in abstract class and allows to define in subclasses



→ abstract method can not be final
→ abstract method can not be static
→ abstract methods can not be private. because private methods can not be overridden but abstract methods must be overridden.
→ abstract class can not be final
→ abstract class can have constructors and those constructors are called whenever objects are created to subclass

```
abstract class A
{
    A()
    {
        system.out.println("constructor");
    }
}
class B extends A
{
    public static void main(String args[])
    {
        new B();
    }
}
```

Implicitly

```
B() { super(); }
```

→ abstract class can have static members

```
abstract class A
{
    static void show()
    {
        system.out.println("java");
    }
}
class B extends A
{
    psvm(String args[])
    {
        show();
        A.show();
        B.show();
    }
}
```

In the above example show() inherited into B class

→ abstract class can have main method also

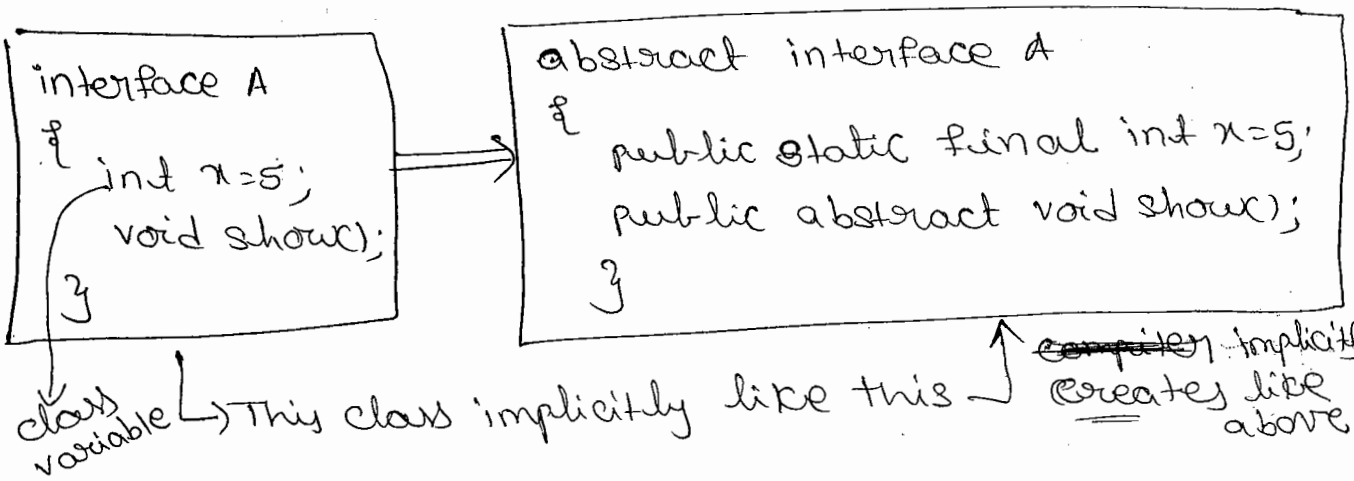
```
abstract class B
{
    psvm(String args[])
    {
        soplm("welcome");
    }
}
```

Interfaces :-

It is a collection of public static final variables and public abstract methods

→ In Interface all variables are implicitly public static final and all methods are implicitly public abstract

→ Interface itself Implicitly abstract



→ Interfaces can not be Instanciated

→ Interface can be Inherited into a class by using implements keyword

→ whenever Interface is Inherited into a class then all methods of an interface must be overridden in a subclass or) subclass must be declared with ~~abstract~~ abstract keyword otherwise compile time error occurs.

→ Interface also can be Inherited into another interface by using extends keyword

19/3/15

Example:-

```
interface A
{   int x=5;
    void show();
}
```

implicitly

```
public static final int x=5;
public abstract void show();
```

```
class B implements A
{   int y=10;
    void show() → access modifier rule is same or less restrictive so here no name in above public there so public in subclass.
    {   sopln("show() method");
    }
    void print()
    {   sopln("print() method");
    }
    Psvm(String args[])
    {   sopln(A.x);
        B ob = new B()
        sopln(ob.y);
        ob.show();
        ob.print();
    }
}
```

program does not compile

```
interface A
{
    int x=5;
    void show();  → implicitly public abstract void show();
}
class B implements A
{
    int y=10;
    public void show();  → Here also must place public access modifier.
    {
        sopln("show() method");
    }
    void print()
    {
        sopln("print() method");
    }
psvm(String args[])
    {
        sopln(A.x);
        B ob = new B();
        sopln(B.ob.y);
        ob.show();
        ob.print();
    }
}
```

1) A ob1 = new A(); X

2) B ob2 = new B(); ✓

s.o.p(ob2.x); ✓ valid not recommended

s.o.p(ob2.y); ✓

ob2.show(); ✓

ob2.print(); ✓

3) A ob3 = new B(); ✓
S.o.p(ob3.x); ✓
S.o.p(ob3.y); X B member we can not access with A reference
ob3.show(); ✓
ob3.print(); X

4) B ob4 = new A(); X downcasting
downcasting is possible first upcasting required

1) class A
{ ═══ ✓
}
class B extends A
{ ═══
}

2) interface A
{ ═══ ✓
}
class B implements A
{ ═══
}

3) interface A
{ ═══ ✓
}
interface B extends A
{ ═══
}

4) class A
{ ═══ X not valid illegal combination
}
interface B ——— A
{ ═══
}
Here class feature different & interface feature different

NOTE:-
class can not be Inherited in Interface
→ Interfaces are Introduced in java to achieve multiple Inheritance

```
class A
{
==
}
class B
{
==
}
interface C
{
==
}
interface D
{
==
}
```

```
class Demo extends A,B{} X
class Demo implements C,D{} ✓
class Demo extends A implements C{} ✓
class Demo extends A implements C,D{} ✓
interface Demo extends C,D{} ✓
```

→ ∴ a class Inheriting not possible bcoz java does not support multiple Inheritance.

abstraction:-

providing necessary Information and hiding unnecessary Information

→ In java abstraction can be implemented by using abstract class and Interface

→ abstract class supports 0 to 100% abstraction whereas Interface supports only 100% abstraction.

20/3/15

Encapsulation:-

Encapsulation is a language construct that facilitates binding of variables with methods and those methods operating on same variables

Program to demonstrate Encapsulation:-

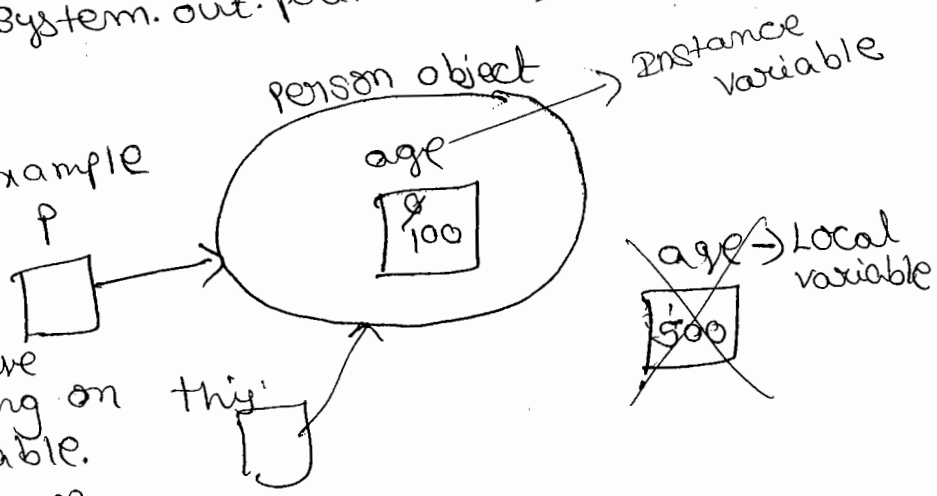
```
class person
{
    private int age;          // -> Instance variable
    void setAge(int age)      // -> local variable
    {
        if (age > 100)
            this.age = 100;
        else if (age < 0)
            this.age = 0;
        else
            this.age = age;
    }
    int getAge()
    {
        return age;
    }
}

class demo
{
    public static void main(String args[])
    {
        person P = new person();
        P.setAge(500);
        int x = P.getAge();
        System.out.println(x);
    }
}
```

whenever Instance variable and local variable names are same then this keyword explicitly required

Ex:- capsule

O/P:- 100

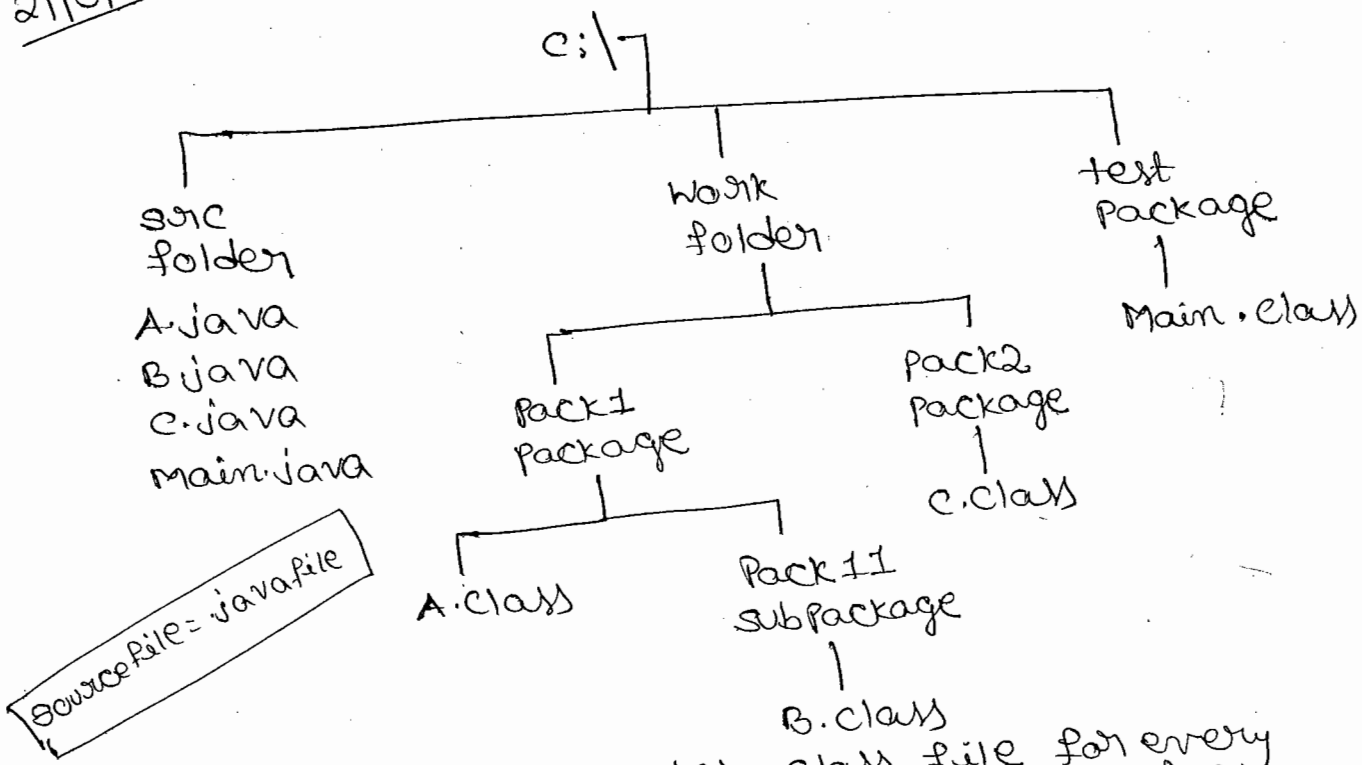
-> In the above example age variable bound with setAge & getAge methods and there methods operating on same age variable. this is known as an encapsulation



Packages:-

a package is a collection of subpackages, classes and interfaces

21/3/15



→ The java compiler generates .class file for every class in a source file (and every interface also)

→ The java compiler also generates folder for every Package and subpackage in a source file

create folders using command prompt:-

```
c:\> md src
c:\> md work
c:\> cd src
c:\>src> start notepad A.java
```

cannot find A.java
Do you want to creat click [OK] to creat new file

md → making directory
cd → changing directory

A.java

```
Package pack1;
public class A
{   public void add(int a, int b)
    {
        System.out.println(a+b);
    }
}
```

class → this method should be accessible to main.class that's why public

Compilation:-

```
c:\src>javac -d c:\work A.java
```

-d → It omits current directory

c:\work → Destination Directory

```
c:\src> start notepad B.java
```

B.java

```
package pack1.pack11;
public class B
{   public void cube(int a)
    {
        sopln(a*a*a);
    }
}
```

package → pack1.pack11; → subpackage

Compilation:

```
c:\src>javac -d c:\work B.java
c:\src> start notepad c.java
```

c.java

```
package pack2;
public class C
{   public void max(int a, int b)
    {   if(a>b)
            sopln(a);
        else
            sopln(b);
    }
}
```

Compilation:-

```
c:\src>javac -d c:\work c.java
c:\src> start notepad main.java
```

~~Main.java~~

import pack1.*;

This statement imports all classes and all Interfaces of pack1 package into our java program. Here * indicates all classes and all Interfaces

import pack1.A;

This statement imports only A class of pack1 package into our java program.

Main.java:-

```
Package test;
~~Package~~
import pack1.*;
import pack1.pack11.*;
import pack2.*;

class main
{
    psvm(String args[])
    {
        A ob1 = new A();
        ob1.add(23,45);
        B ob2 = new B();
        ob2.cube(5);
        c ob3 = new cc();
        ob3.max(24,88);
    }
}
```

Compilation and Execution:-

```
c:\src>set classpath="%classpath%.";
c:\src>javac -d c:\main.java
c:\src>cd..
c:\>java test.main
```

c:\work; → It is to inform to compiler that all packages are exist under work folder → append mode.

test → package name. main → class name.

68
125
88

23/3/15

javap :(using this command we can get the profile of a class)

It is called as JDK tool and it is used to get the profile of a class or an Interface.

Ex:- use the following to get the profile of A class

```
c:\work>javap pack1.A
```

```
Compiled from "A.java"
public class pack1.A extends java.lang.Object{
        public pack1.A();  → constructor
        public void add(int,int);  → method
}
```

↳ It is a profile of A class

It shows the class contains how many methods and how many constructors

java.lang.object is class is a superclass for all java classes

→ In java each and every class either directly or Indirectly derived from java.lang.object class

1) class Demo [extends object] ↑ implicitly
{ ⇒ }

2) class A [extends object] ↑ implicitly
{ ⇒ }
class B extends A
{ ⇒ }

If we want to see string class profile use the following

```
c:\>java java.lang.String
```

This class is in java.lang package

javadocumentation:-
It contains package description, classes description, Interface description, methods description .... etc.

```
file:///E:/jdk-7u45-apidocs/docs/api/index.html
```
→ api documentation

javadoc:-
It is called as JDK tool and it is used to create documentation

Ex:-
```
c:\src>javadoc *.java
```

open index.html from src folder

String-Handling:- In java the strings can be handled in the following ways.
There are four string related classes to handle strings

1) java.lang.String
2) java.lang.StringBuffer
3) java.lang.StringBuilder
4) java.util.StringTokenizer

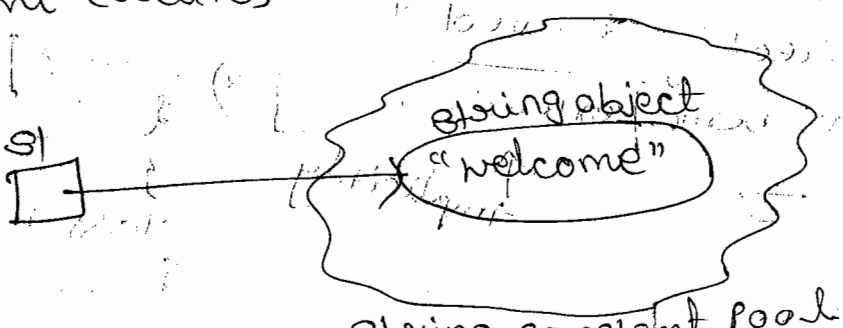
All are string classes.

→ every string literal itself an object of string class

Ex:- "welcome" → It is an object of string class

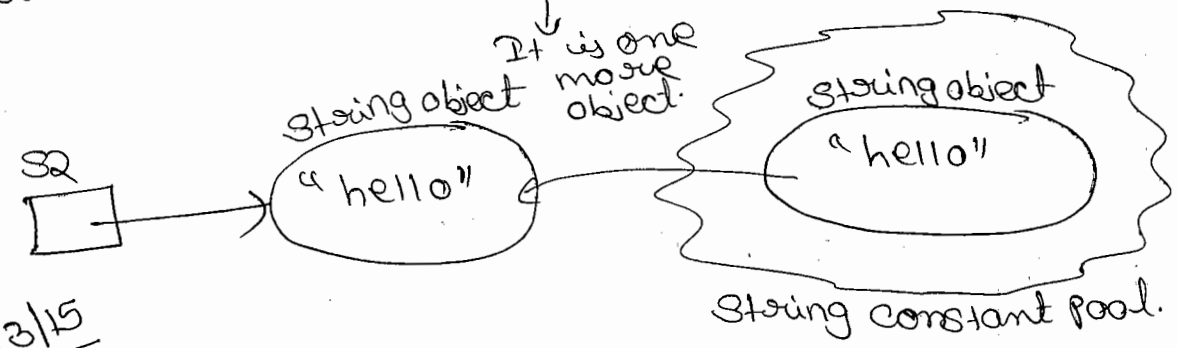
```
String s1 = "welcome";
```
→ with this we can save memory.

This statement creates one object in string constant pool



`String s2 = new String("hello");` → literal itself object

This statement creates one object in String constant pool and one more object outside the pool.



24/3/15

profile of java.lang.String class:-

public final class String extends Object

`javap java.lang.String`

constructors:-

public String();
public String(String);
public String(char[]);

`public String(byte[]);`
→ It is used to convert byte array into a string.

→ It is used to convert char[] array to a String

methods:-

public int length();
→ It returns no. of chars in a given String

public char charAt(int);
→ It returns char at specified index

public byte[] getBytes();
→ It is used to convert String to byte array

public boolean equals(Object);
public boolean equalsIgnoreCase(String);
public String concat(String);
public String toLowerCase();
public String toUpperCase();

Public String trim();

→ It is used to remove extra spaces at the beginning of the text and end of the text

public String toString();

→ It returns string representation of string object data

public char[] toCharArray();

→ It is used to convert string to char array

NOTE:-

Whenever object or object reference is passed as an argument to a method then `toString()` method is called implicitly

→ String class toString() returns content of string object whereas object class toString() method returns ~~class name at~~ classname@Hashcode-In-Hexadecimal format

How many classes are there in java.lang package

Example:-

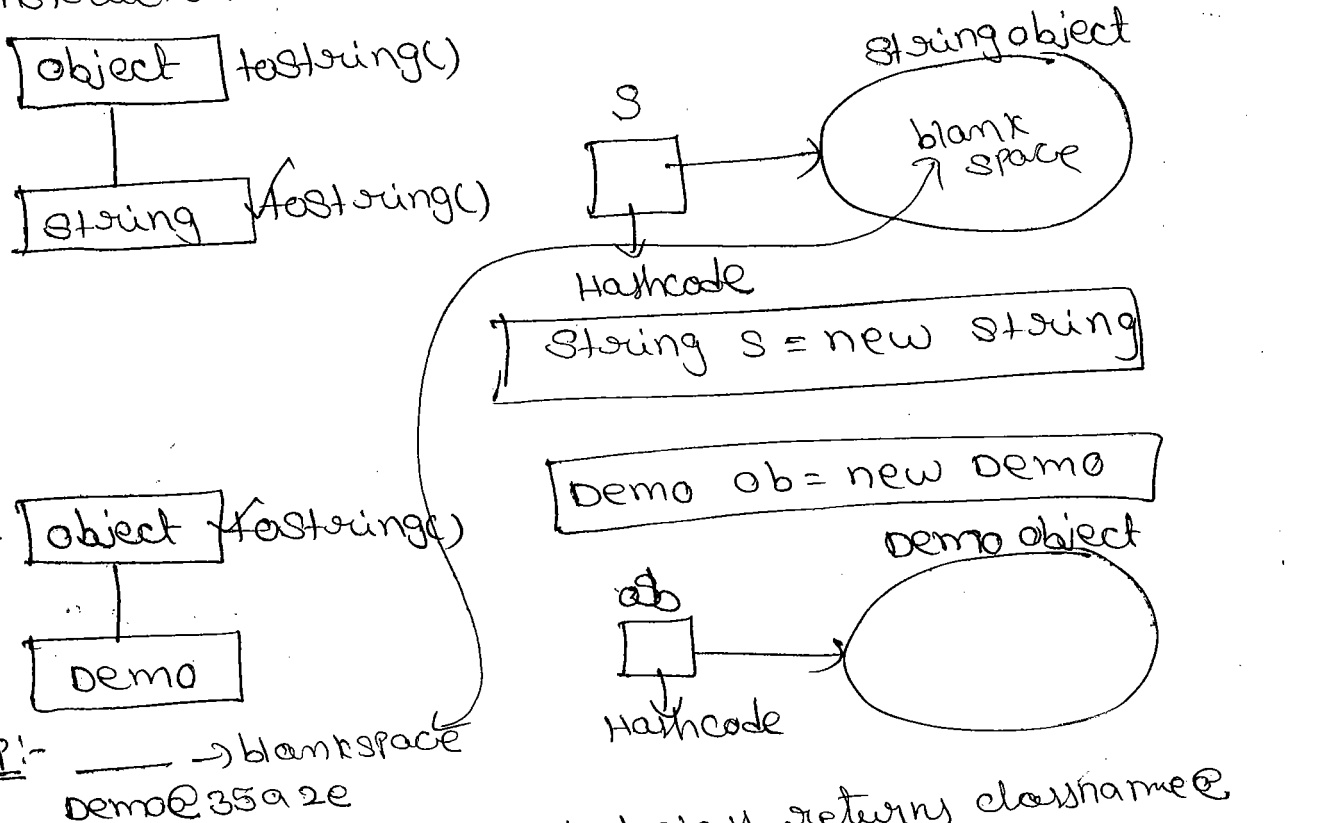
```
class Demo
{
    public static void main(String args[])
    {
        String s = new String();   → It calls default constructor of String class
        Sys.sopln(s);              → s.toString();
        Demo ob = new Demo();
        sopln(ob);                 → ob.toString();
    }
}
```

It is defined in library we are accessing

Whenever object (or) object reference is passed in sopln() then implicitly toString() method is called.

implicitly

The above program ~~contents~~ contains default String() constructor.



→ tostring() method of object class returns classname@Hexa-decimal-format

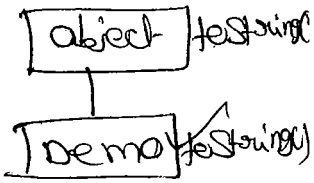
→ tostring() method of String class returns content of String object

→ The object class tostring() method can be overridden by programmer to change the definition

Example:-

```
class Demo [extends object]  → Implicitly This super class (object class)
{
    public String tostring()   object class contains tostring that why we are overriding Here
    { return "welcome";
    }
    psvm(String args[])
    { Demo ob = new Demo();
      sopln(ob);  → Here Demo class to string called
    }
}
```

sopln(ob) → ob.tostring() ↳ implicitly



o/p:- welcome

```
class Demo extends String
{ =                    X Error because String is a final class
}
```

Example:-

```
class Demo
{
    psvm(String args[])
    {
        byte b[] = {97, 98, 99, 100, 101};   // It is converting byte[] array to String
        String s1 = new String(b);
        sopln(s1);
        char ch[] = {'h', 'e', 'l', 'l', 'o'};   // It is converting char[] to String
        String s2 = new String(ch);
        sopln(s2);
    }
```

→ ~~This~~ There are already defined in library abroad built in Here we are using If we declare like that no error bcause every java.lang is default package in above class. String class is related to java.lang package.

o/p:- abcde
hello.

Example:-

```
boolean b = s1.equals(s2);
sopln(b);
```

↓↓

we can write those two lines in a single line

```
sopln(s1.equals(s2));
```

Example:-

```
class Demo
{
    psvm(String args[])
    {
        String s = new String("hello");
        byte b[] = s.getBytes();
        for(int i=0; i<b.length; i++)
        {
            sopln(b[i]);
        }
        char ch[] = s.toCharArray();
        for(int i=0; i<ch.length; i++)
        {
            sopln(ch[i]);
        }
    }
}
```

Here we are converting String to byte array

logic inbuilt

String to character array.

O/P:-
```
104
101
108
108
111
h
e
l
l
o
```

Example:-

```
class Demo
{
    psvm(String args[])
    {
        String s = new String("hello");
        int n = s.length();
        sopln(n);
        char ch = s.charAt(2);
        sopln(ch);
    }
}
```

O/P:- 5
l

h e l l o
0 1 2 3 4 → index

NOTE:-

String constant pool does not allow duplicates

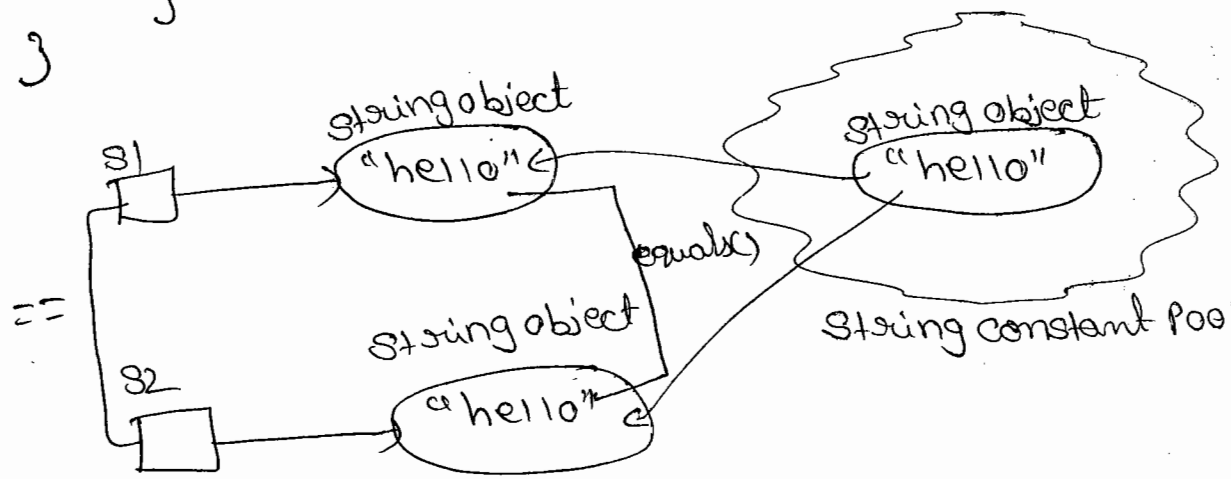
→ equals() of string class compares the contents of string objects whereas equals operator compares the Hashcode

(==)

Example:-

```
class Demo
{
    psvm(String args[])
    {
        String s1 = new String("hello");
        String s2 = new String("hello");
        sopln(s1.equals(s2));
        sopln(s1==s2);
    }
}
```

o/p:- true
false

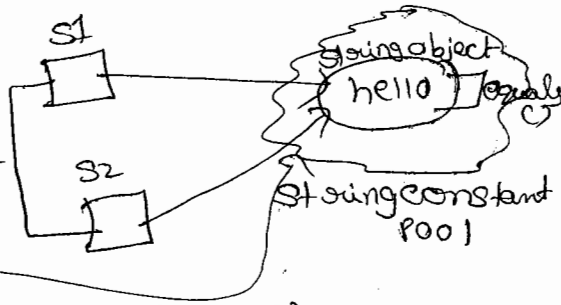


25/3/15.

→ equals() method of object class compares the Hash-code

Example:-

```
class Demo
{
    psvm(String args[])
    {
        String s1 = "hello";
        String s2 = "hello";
        System.out.println(s1.equals(s2));
        System.out.println(s1==s2);
    }
}
```



o/p:- true
true

both the hashcodes are same so:- true.

Example:-

```
class Demo
{
  public static void main(String args[])
  {
    String s1 = new String("hello");
    String s2 = new String("HELLO");
    System.out.println(s1.equals(s2));
    Sopln(s1.equalsIgnoreCase(s2));
  }
}
```

o/p: false
true

Example:-

```
class Demo
{
  public static void main(String args[])
  {
    String s1 = new String("Taj");
    String s2 = new String("mahal");
    String s3 = s1.concat(s2);
    System.out.println(s3);
  }
}
```

o/p: Tajmahal

s3 contains hashcode but implicitly s3.toString() thats why o/p:- Tajmahal

Example:-

```
class Demo
{
  PSVM(String args[])
  {
    String s1 = new String("welcome");
    String s2 = s1.toUpperCase();
    Sopln(s2);
  }
}
```

o/p:- WELCOME

```
class Demo
{
    psvm(String args[])
    {
String s1 = new String("   welcome to java   ");
        String s2 = s1.trim();
        System.out.println(s1);
        System.out.println(s2);
    }
}
```

O/P:- welcome to java
welcome to java

Differences b/w String and StringBuffer

String	StringBuffer
i) The object of String class is immutable	i) The object of StringBuffer class is mutable
ii) methods of String class are not synchronized	ii) methods of StringBuffer class are synchronized

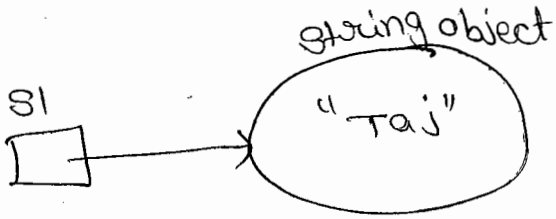
immutable object:-

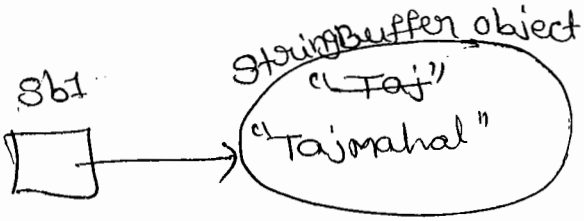
It means the value of an object can not be changed **mutable:-** The value of an object can be changed

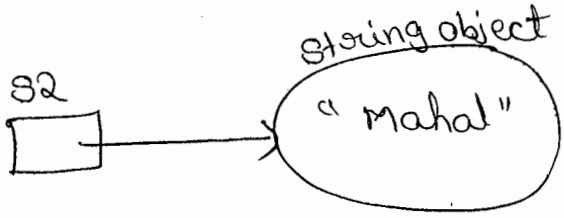
Example:-

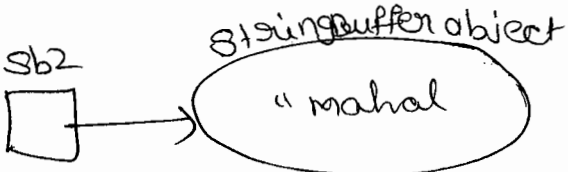
```
class Demo
{
    psvm(String args[])
    { String s1 = new String("Taj");
      String s2 = new String("mahal");
       s1.concat(s2);
       sopln(s1);
StringBuffer sb1 = new StringBuffer("Taj");
StringBuffer sb2 = new StringBuffer("mahal");
        sb1.append(sb2);
    } }  sopln(sb1);
```

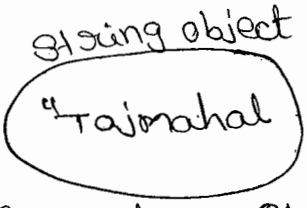
O/P:- Taj
Tajmahal











Here S2 concat to S1 so Here another string object is created the result is stored another object with out reference

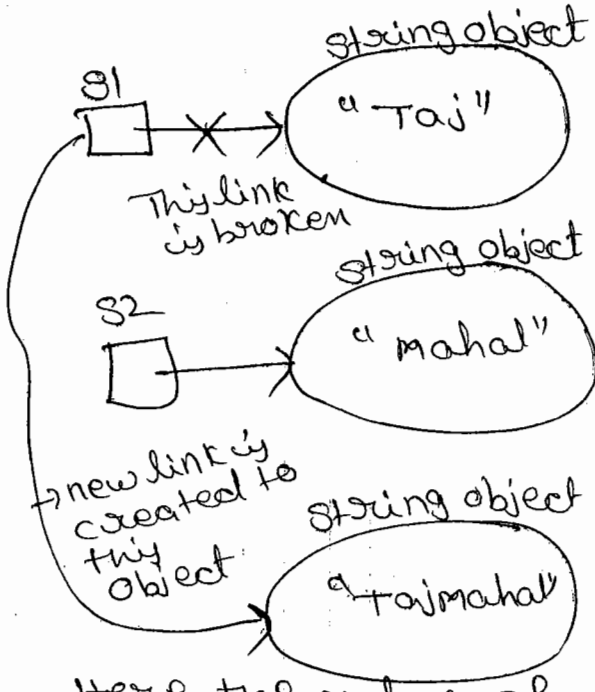
→ Here the value of object can not be changed thats why one more object created and result stored in ~~tweed~~ that object

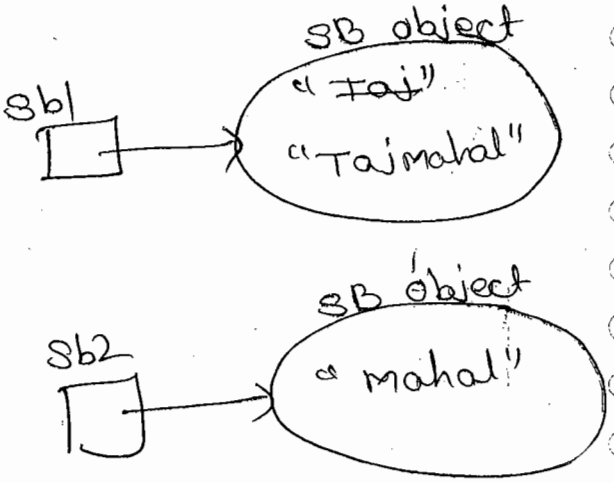
Here Sb2 append to Sb1 so Tajmahal result is stored in Sb1 object

Example:-

```
class Demo
{
    psvm(String args[])
    {
        String S1 = new String("Taj");
        String S2 = new String("mahal");
        S1 = S1.concat(S2);
        Sopln(S1);
StringBuffer Sb1 = new StringBuffer("Taj");
StringBuffer Sb2 = new StringBuffer("mahal");
    Sb1 = Sb1.append(Sb2);
        Sopln(Sb1);
    } }
```

o/p:- Tajmahal
Tajmahal





Here the value of object not changed the reference changed.

Synchronization!-

It is a mechanism that allows to access a shared resource only one Thread at a time

Thread!-

a Thread is a piece of code that executes independently

difference between string, StringBuffer and StringBuilder

String	StringBuffer	StringBuilder
i) immutable	i) mutable	i) mutable
ii) not synchronized	ii) synchronized	iii) not synchronized

StringTokenizer class

It allows an application to break a string into Tokens (words)

java.util.StringTokenizer

constructor!-

public StringTokenizer(String);

Methods:-

public boolean hasMoreTokens();

→ It returns true if the token is present, otherwise returns false

public String nextToken();

→ It returns current token and moves the cursor to next token if the token is present

public int countTokens();

→ It returns no. of tokens.

Example:-

```
import java.util.*;
class Demo
{  psvm(String args[])
   {  String s = " welcome to java";
StringTokenizer st = new StringTokenizer(s);
    int n = st.countTokens();
     sopln(n);
   }
}
```

O/P: 3

for creating object constructor required.

javac Demo.java
java Demo

Example:-

```
import java.util.*;
class Demo
{  psvm(String args[])
   {
String s = " welcome to java";
StringTokenizer st = new StringTokenizer(s);
      while(st.hasMoreTokens())
       {  sopln(st.nextToken());
}}
}
```

26/3/15

Command line arguments :-

The arguments that are passed at the command prompt are called command line arguments

```
c:\>java Demo welcome to Java
           0     1   2   -> index
           command line arguments
```

These arguments are received by main method

while executing the java program what are we passed at the command line are called command line arguments.

These arguments are received by main method

Example:-

```
class Demo
{
    psvm(String args[])   -> command line arguments are received here
    {                        -> If we pass integers also it will treat like a string
        for(int i=0; i<args.length; i++)
        {
            sopln(args[i]);
        }
    }
}
```

```
for(String s : args)
{
    sopln(s);
}
```

o/p :-

```
c:\>javac Demo.java
    java Demo welcome to Java
    welcome
      to
      Java
    java Demo 10 20 30 40
                "10" -> String type
      10
      20
      30
      40
```

Ex:-

```
class Demo
{
    psvm(String args[])
    {
        sopln(args[0]+args[1]);
    }
}
```

o/p: e:\>java Demo.java
e:\>java Demo Tajmahal
Tajmahal
e:\>java Demo 10 20
1020
↓ This we convert 30 so we need wrapper class concept

Wrapper classes :-

Each of Java's 8 primitive data types has a class and those classes are called wrapper classes because they wrap the data into an object

Primitive Data type	Wrapper class
1) byte	1) Byte
2) short	2) Short
3) int	3) Integer
4) long	4) Long
5) float	5) Float
6) double	6) Double
7) char	7) Character
8) boolean	8) Boolean

→ All wrapper classes are the part of java.lang package

java.lang.Integer

constructor :- (constructor class name & constructor name must match.)

```
public Integer(int);
```

→ It is used to convert primitive data type to reference data type → It is used to convert int to Integer

Method :-

```
public int intValue();
```

→ It is used to convert reference data type to primitive data type → It is used to convert Integer to int

Primitive data type to Reference data type

```
int x = 5;
```

x → 5 (primitive data type)

```
Integer ob = new Integer(x);
```

(ob: Reference data type) → Integer object: 5

Reference data type to primitive data type

```
Integer ob = new Integer(5);
```

(ob: Reference data type) → Integer object: 5

```
int x = ob.intValue();
```

x → 5 (primitive datatype)

→ whenever the object is created that time constructor called

→ some of the applications x value can not pass so we have to convert object value we need constructor in above example

Auto boxing:-

The process of converting primitive data type to the corresponding Reference data type is known as Auto boxing

int to Integer
char to character
} corresponding only possible

Example:-

```
int x = 5;
↓
primitive data type

Integer ob = x; ⇒ Auto boxing
↓
Reference data type.
```

(without constructor we are converting)

Auto unboxing:-

The process of converting Reference data type to the corresponding primitive data type is known as Auto unboxing

(without method)

This is only possible wrapper classes that to corresponding classes ex:- Integer to int, character to char

```
Integer ob = new Integer(5);
↓
Reference data type

int x = ob; ⇒ Auto unboxing.
↓
primitive data type
```

→ Both Auto boxing and Auto unboxing features added in JDK 1.5 version in 2004

→ Automatically both will done boxing in the box retriving the box unboxing (without method) OUT constructor that is the reason there Introduced

java.lang.Integer:-

Methods:-

```
public static String toHexString(int);
public static String toOctalString(int);
public static String toBinaryString(int);
```

Example:-

```
class Demo
{
    psvm(String args[])
    {
        int a=30;
        sopln(a);
        String s = Integer.toOctalString(a);
        sopln(s);
        sopln(Integer.toOctalString(a));
        sopln(Integer.toHexString(a));
        sopln(Integer.toBinaryString(a));
    }
}
```

(There are 5 lines we can write in single line)

$30_{(10)}$ → Decimal

8 | 30
3 — 6 ↑

$36_{(8)}$ → Octal

16 | 30
1 — 14 ↑

10 → a
11 → b
12 → c
13 → d
14 → e
15 → f

$1e_{(16)}$ → Hexadecimal.

2 | 30
2 | 15 — 0
2 | 7 — 1
2 | 3 — 1
1 — 1 ↑

$1110_{(2)}$ → Binary

```
int a = 30;
    └→ decimal literal
int b = 036;
    └→ ocal literal
int c = 0x1e;
    └→ Hexadecimal literal
int d = 0X1e;
int e = 0b1110;
    └→ Binary literal;
int f = 0B1110;
```

Decimal to octal
$30_{(10)} \longrightarrow 36_{(8)}$

octal to decimal
$36_{(8)} \longrightarrow 30_{(10)}$

NOTE:-
Binary literal feature added in JDK 1.7 version in 2011

java.lang.Integer:-

Method:-

```
public static int parseInt(String)
                throws NumberFormatException.
```

→ It is used to pass integer from given string

Example:-

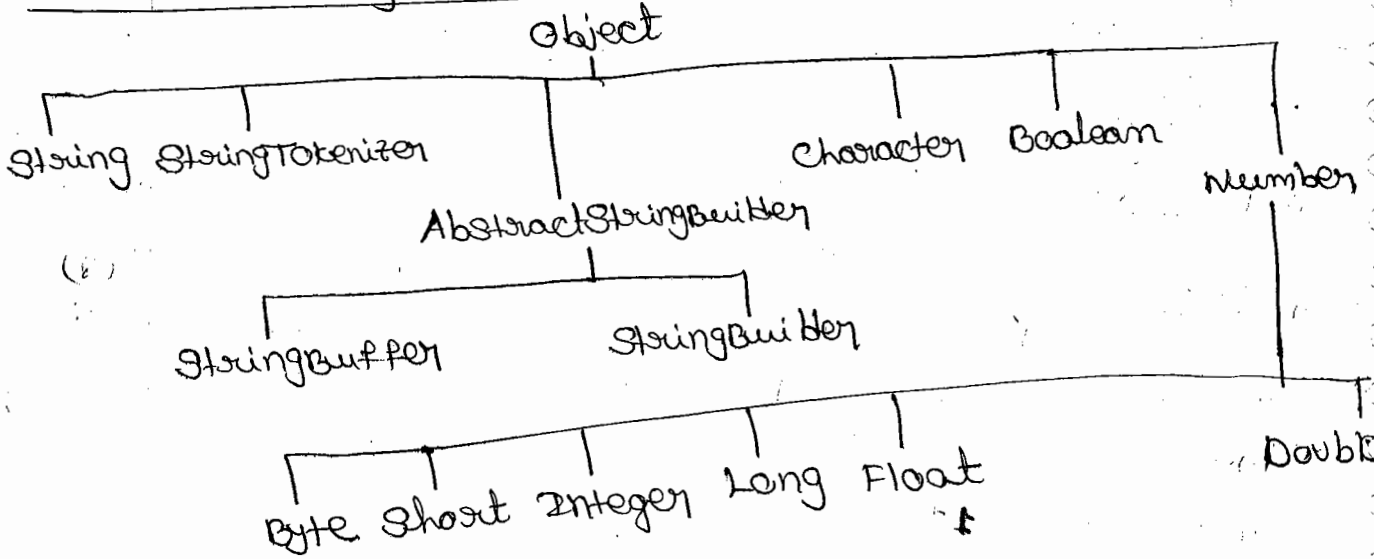
```
class Demo
{
    public static void main(String args[])
    {
        int x = Integer.parseInt(args[0]);
        int y = Integer.parseInt(args[1]);
        System.out.println(x+y);
    }
}
```

(both arguments are received here; and passed to first argument)

o/p: c:\>javac Demo.java

```
java Demo 10 20
30
c:\> java Demo Taj mahal
NumberFormatException.
```

The Hierarchy of String Related classes and wrapper classes



Example:-

```
class Demo
{
    psvm(String args[])
    {
        int a = Integer.parseInt(args[0]);
        int b = Integer.parseInt(args[1]);
        System.out.println(a+b);
    }
}
```

"10" → String
↓ → parsing
10 → int

```
c:\>java Demo 10 20↵
30
c:\>java Demo Taj mahal↵
```
→ Runtime error
NumberFormatException

23/3/15

Exception Handling

Errors:-

- A programming mistake is said to be an error
  There are three types of errors
  1) compile time errors (syntax errors)
  2) Run time errors (Exceptions)
  3) Logical errors (program logic mistake)

Exception:-

→ Exception means Run time error
→ only Run time errors are called Exceptions
→ All errors are not Exceptions
→ All Exceptions are classes in java
→ In Exception Handling we use the following keywords

try, catch, throw, throws & finally

Logical errors

not be shown wrong o/p comes
If we want to perform addition mistakenly (-) → substraction Place then unexpected o/p comes

The syntax of try and catch blocks:-

```
try
{
   ==  } task code
}catch(ExceptionClassName objectReference)
{
   ==  } message(user friendly error message)
}
```

→ The try block must be associated with atleast one catch block or finally block

→ whenever Exception occured in a program then the object of related Exception class is created by JVM. and passed to Exception Handler (catch blocks)

→ catch block is called as Exception Handler and it Handles the Exception

→ After Handling Exception catch block code is Executed

new ArrayIndexOutOfBoundsException();
↳ created by JVM.

```
try
{
   ==
}catch(ArrayIndexOutOfBoundsException ae)
{
   ==
}
```

Exception Handler → catch block

Size	Range	
5	0 to 4	5 (ArrayIndexOutBoundsException)
4	0 to 3	4 (")

Example!-

```
class Demo
{
  public static void main(String args[])
  {
     try{
       int a = Integer.parseInt(args[0])
       int b = Integer.parseInt(args[1]);
       int c = a/b;
       sopln(c);
   }catch(ArrayIndexOutofBoundsException ae)
   {system.err.println
    sopln(" Please pass two arguments ");
   }
   catch(NumberFormatException ne)
   {
     system.err.println(" Please pass two
                          integers only ");
   }
   catch(ArithmeticExceptron ae)
   { system.err.println(" Please Pass second
                        argument except zero");
   }
  }
}
```

O/P!- c:\>javac Demo.java
c:\>java Demo
Please Pass two arguments
c:\>java Demo abc xyz
Please Pass two integers only
c:\>java Demo 10 0
Please pass second argument except zero
c:\>java Demo 10 2 5 (O/P)

24/3/15

There are two types of Exceptions

1) Checked Exceptions
2) Unchecked Exceptions

Checked Exceptions:-

The Exception classes which are derived from java.lang.Exception class are called checked Exceptions

→ They donot include java.lang.RuntimeException class and all its subclasses

→ All checked exceptions must be handled by programmer explicitly otherwise compile time error occurs

→ The java compiler checks try and catch blocks for this kind of Exceptions

→ All application specific Exceptions are comes under this category

Unchecked Exceptions:-

The Exception classes which are ~~derived~~ derived from java.lang.RuntimeException class are called unchecked Exceptions

→ All unchecked Exceptions are handle by system Implicitly

→ The java compiler does not check try and catch blocks for this kind of Exceptions

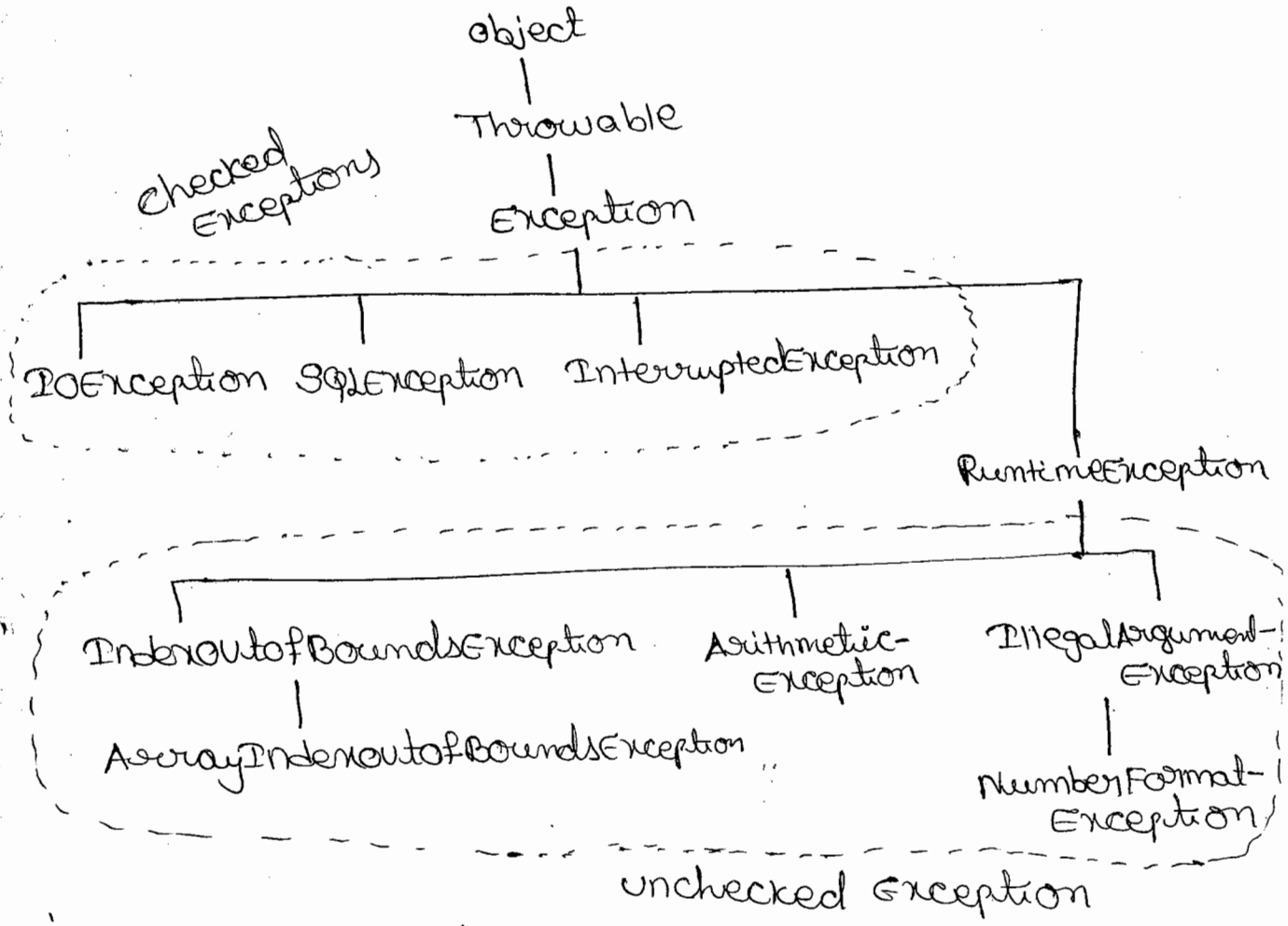
→ All general Exceptions are comes under this category

→ Handling unchecked Exceptions are optional.
by programmer.

→ main is not a library method

→ we can no ~~I can~~ library
we can not write open { } for library method.

The Hierarchy of Exception classes:-



java.lang.Integer:-

Method:-

```
public static int ParseInt(String)throws
                    NumberFormatException;
```

→ It is used to parse int from given String

```
class Demo
{ public static void main(String args[])
    { int a = Integer.ParseInt(args[0]);
    }
}
```

→ The above program compiles successfully because ParseInt() method throws Unchecked Exception

→ All Unchecked Exceptions are Handled by system Implicitly

java.io.FileInputStream

Constructor:-

Public FileInputStream(String) throws FileNotFoundException.
(String → file name)

→ It opens a file for reading if the file is exist, otherwise FileNotFoundException will be thrown

c:\>javac demo.java
unreported exception java.io.FileNotFoundException must be called
→ It is called as compile time error

```
import java.io.*;
class Demo
{  public static void main(String args[])
   {
FileInputStream fis = new FileInputStream(args[0]);
   }
}
```

→ The above program will not compile because FileInputStream constructor throws checked Exception.

→ All checked exceptions must be handled by programmer explicitly

```
import java.io.*;
class Demo
{  psvm(String args[])
   {  try{
FileInputStream fis = new FileInputStream(args[0]);
   }catch(FileNotFoundException fe)
   {  system.err.println(fe);
   }
}
```

This catch block handles FileNotFoundException

fe.toString()

c:\> javac Demo.java
⟶ compiled successfully

c:\> java Demo Demo.java
⟶ Executed successfully

c:\> java Demo abc.txt
java.io.FileNotFoundException
↓
It is runtime error.

```
import java.io.*;
class Demo
{
    psvm(String args[])
    { try{
FileInputStream fis = new FileInputStream(args[0]);
}catch(Exception e)
{
System.err.println(e);
}
}
}
```

This catch block handles all Exceptions because all Exceptions are derived from Exception class.

e.toString()

→ Here Throwable class toString() method → implicitly is called

Identify
⇒ If the class is derived from Exception class that is checked Exception
↳ must be derived from Exception class

→ If the class is derived from RuntimeException class that is unchecked Exception

→ whenever Exception occurs JVM creates a object

26/3/15

Program to create and handle checked Exception

~~Exception~~:-

Example:-

```
class negativenumberException extends Exception
{
}
class Demo
{
void cube(int a)throws negativenumberException
    {   if(a>0)
          System.out.println(a*a*a);
        else
    throw new negativenumberException();
    }
psvm(String args[])
    {   try{
        int x = Integer.parseInt(args[0]);
          Demo ob = new Demo();
            ob.cube(x);
    }catch(negativenumberException ne)
    {   System.err.println(ne);
    }
}
```

receiver here with that

extracts here then converts int

Cube method is instance method create object

Here Throwable class toString() called

ne.toString()

implicitly

with that class name comes.

This type programs we can not use feature to understand throws purpose this program

Compilation & Execution

```
c:\>javac Demo.java
c:\>java Demo 5
  125
c:\>java Demo -3
    negativenumberException
```

Program to create unchecked Exception :-

```
class negativeNumberException extends RuntimeException
{
}
                    Here NNE is predefined
class Demo
{
void cube(int a)throws negativeNumberException
      { if(a>0)
          sopln(a*a*a);
        else
throw new negativeNumberException();
 }
 psvm(String args[])
 { int x = Integer.parseInt(args[0]);
     Demo ob = new Demo();
      ob.cube(x);
   }
}
```

This is unchecked Exception so no need to handle with try, catch because system handles implicitly.

→ In the above example system implicitly provides try and catch blocks to handle negativeNumber-Exception because it is unchecked Exception

"throw" keyword :-

It is used to pass an object of Exception class to a catch block

"throws" keyword :-

It is used to apply an Exception to a method and also used to handle Exceptions

→ In the above two examples negative negativeNumber-Exception applied to cube method

→ If throws keyword there then Exception

There are two ways to handle Exceptions

1) By using try and catch blocks
2) By using throws clause

if we want to display user friendly message we have to write try & catch only handle means we use throws.

1) By using try and catch blocks :-

Example:-

```
psvm(String args[])
{
    try{
     int x = Integer.parseInt(args[0]);
     Demo ob = new Demo();
      ob.cube(x);
    }catch(NegativeNumberException ne)
    {
       System.err.println(ne);
    }
}
```

2) By using throws clause

cube method is throwing Exception here main method also throwing Exception at last system provies one catch block that catch block handles Exception.

Example:-

```
psvm(String args[])throws NegativeNumberException
 {
      int x = Integer.parseInt(args[0]);
      Demo ob = new Demo();
      ob.cube(x);
 }
```

-> In this approach try and catch blocks are provided by system implicitly

finally block :-

It is used to perform cleanup activities
cleanup activities are closing a file, closing a socket, closing a database connection ..... etc

-> use try & catch blocks to display user friendly error msg
-> use throws clause to handle Exception.

NOTE:-
finally block is executed even Exception occurs in a program

→ By compilation we can Identify whether it is checked Exception or unchecked Exception

→ At the time of compilation if we got unreported that is checked Exception

→ By using library we can check whether it is checked or unchecked Exception

26/3/15

JDK, JRE, JVM & JIT

JDK

JDK stands for java development kit (These all are developed in C language)

It contains JRE, development Tools (javac, javap, java, javadoc .... etc), additional libraries and supporting files for development tools.

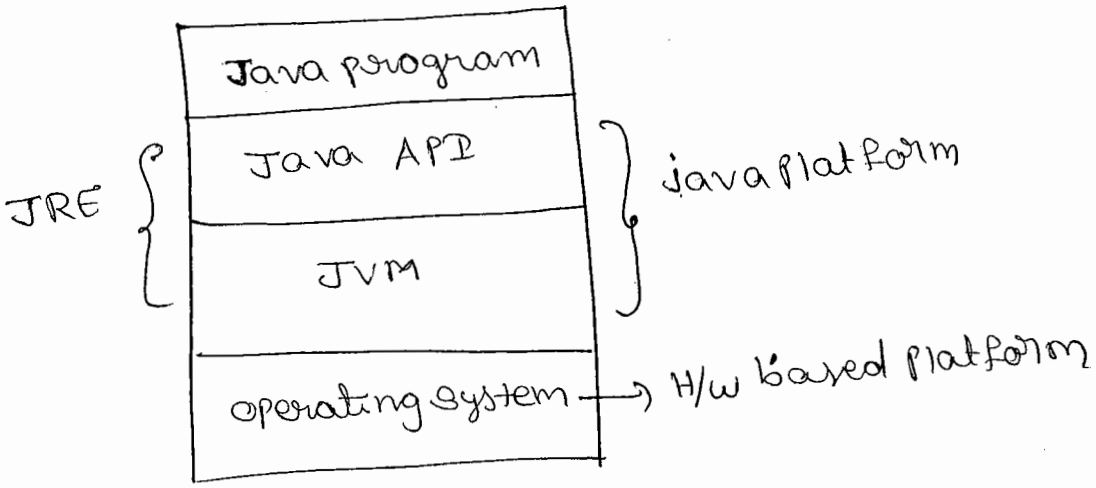
JDK is called as java software

→ All development tools are the part of bin folder in JDK

JRE:-

JRE stands for java Runtime Environment

→ It is called as java platform it contains JVM and class libraries (java api) (Java API)



→ Java Platform consists of Java API and JVM
→ Java API (Application programming Interface)
→ Java API is called as Java library and it is included in Java program at Runtime

There are two jre's
1) public jre
2) private jre.

Difference between public jre and private jre

Public jre	private jre
c:\Program Files\Java\jre6 is called as public jre (outside the JDK)	c:\Program Files\Java\ jdk1.6.0_11\jre is called as private JRE (Inside the JDK)
→ Public JRE used by browser to run Java Applets	→ private JRE used by operating system to run Java applications
→ public JRE has a registory with operating system	→ private JRE has no registory with operating system

JVM :-

JVM stands for Java virtual machine

JVM contains 5 components
1) Class Loader
2) Byte code verifier
3) Garbage collector
4) Security manager
5) Execution engine

1) Class Loader:-

It is a JVM component and it loads the classes from the following locations

1) C:\Program Files\Java\jdk1.6.0_11\jre\lib

It contains predefined classes

2) C:\Program Files\Java\jdk1.6.0_11\jre\lib\ext

It is suitable for project files

3) class path locations

It is suitable for applications

2) Byte code verifier:-

It verifies the byte code instructions which are loaded from server system

3) Garbage collector:-

It is a JVM component and it collects the garbage whenever CPU gets free time because garbage collector priority is least priority (priority no. is 1)

It is also possible to call garbage collector explicitly by using gcc() method of java.lang.System class

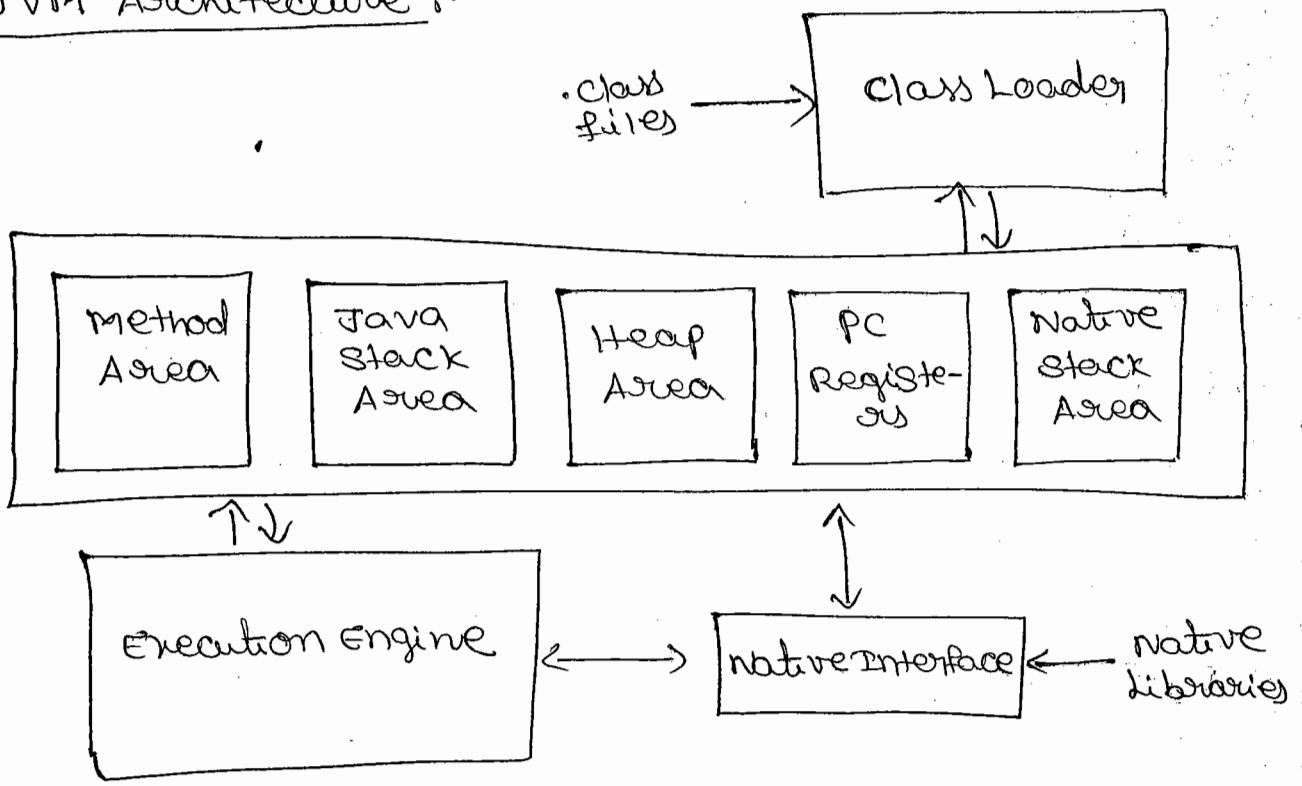
4) Security manager:-

It is a JVM component and it provides security service

5) Execution Engine:-

It contains Interpreter and JIT compiler. both Interpreter & JIT compiler converts byte code instructions to bit code to run under operating-system

JVM Architecture :-



There are 5 memory blocks in jvm

1) Method Area
2) Java Stack Area
3) Heap Area
4) PC Registers (Program counter Registers)
5) Native Stack Area

Method Area!-

It contains class variables and method definitions

Java Stack Area!-

It contains local variables and method call statements

Heap Area!-

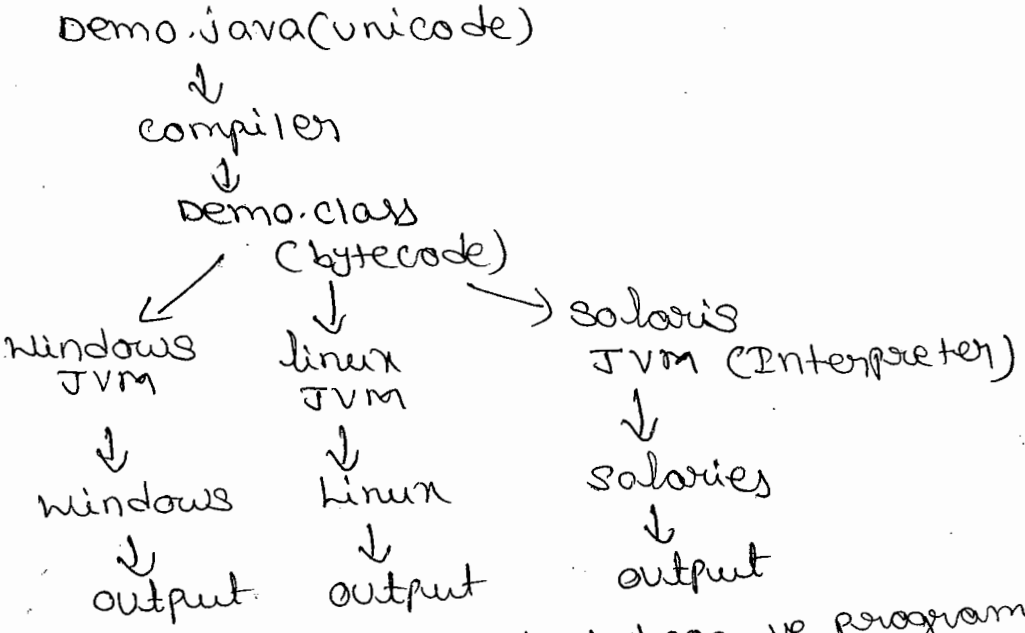
It contains Instance variables and objects

PC Registers!-

Program counter Registers. It contains byte code instructions order to be executed on JVM.

Native Stack Area!-

It contains native members. (non Java member)
eg!- C, C++



→ Java is a platform Independent because programs written in java language can executed on any platform.

→ Java virtual machine is not a platform independent because windows JVM is specific to windows, linux JVM is specific to linux, solaris JVM is specific to solaries - etc.

JIT compiler:- (JustInTime compiler)

→ It is introduced in JDK 1.1 version to improve the performance of JVM.

→ JIT compiler identifies the repeated code instructions, converts those instructions from byte code to bit code, put into cache memory and loads several times as needed.

→ JIT is a part of JVM, JVM is the part of JRE and JRE is a part of JDK.

Static imports:-

This feature allows to avoid class name with static members.

Syntax:- import static package-name.sub-package-name.ClassName.*;

→ Here * indicates all static members.

eg:- import static java.lang.Integer.*;

It means all static members of Integer class can be accessed without classname.

```
import static java.lang.Integer.*;
class Demo
{
   psvm(String args[])
   { int x = parseInt(args[0]);
     int y = parseInt(args[1]);
     System.out.println(x+y);
} }
```

→ ~~The~~ This feature added in JDK 1.5 version in 2004

enumerations:-

This feature allows us to create new datatype.

→ In order to use this feature enum keyword added in JDK 1.5 version in 2004.

Example:-

```
enum Day
{
   MON, TUE, WED, THU, FRI, SAT, SUN
}
class Demo
{
   public static void main(String args[])
   {
      Day d = Day.MON;
      System.out.println(d);
} }
```

Note: All enumerations literals are implicitly static

varArgs:- It means variable Arguments.

this feature allows to pass '0' to 'n' number of arguments to a method.

example:-

```
class Demo
{
  void show(int...a)        (3-dots)
  {
    for(int b:a)  → enhanced for loop (or) for each loop
    { System.out.println(b);
  } }
  psvm(String args[])
  {
    Demo ob = new Demo();
      ob.show(23,24,65,34);
      ob.show(4323);
      ob.show(345,389,967,478,932);
  }
}
```

→ It is called as varArgs

o/p:- 23
24
65
34
4323
345
389
967
478
932

→ Both varArgs and for each loop features added in JDK 1.5 version in 2004.

Strings in switch statement:-

This features allows to pass string literals in a switch statement.

→ This feature added in JDK 1.7 version in 2011.

eg:-

```
class Demo
{
  psvm(String args[])
  { switch(args[0])
    {
    case "mon": System.out.println("Monday");
          break
    case "tue": System.out.println("Tuesday");
          break
```

```
default : System.out.println("Invalid");
}
}
```

Underscores in numeric literals :-

This feature allows to write any no. of underscore symbols in int literals and floating point literals.

eg:-
```
class Demo
{
    psvm(String... args)
    {
        int mobileNo = 99_85_97_46_92
        System.out.println(mobileNo)
    }
}
```

→ This feature added in jdk 1.7 version in 2011

Java Streams

A stream is a flow of data from source to destination

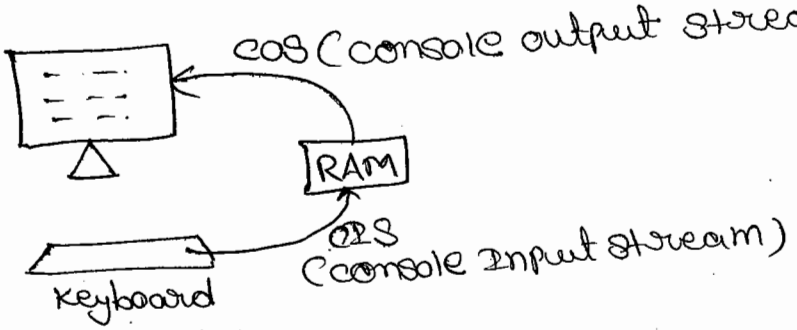
→ A source can be a keyword, file, client, server — etc

→ A destination can be a monitor, client, server, file — etc

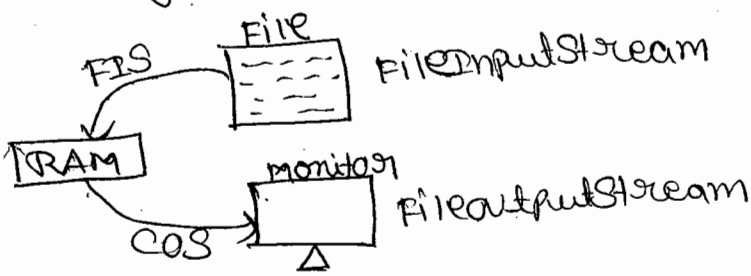
→ In java streams all divided into 3-categories.

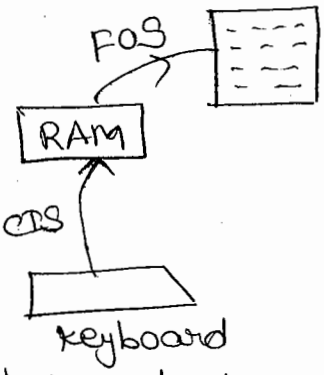
1) console Input/output streams
2) File Input/output streams
3) network Input/output streams

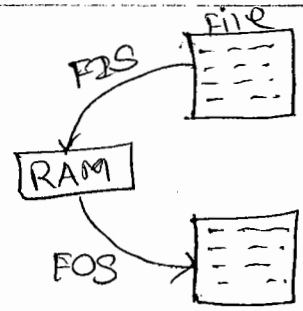
1)

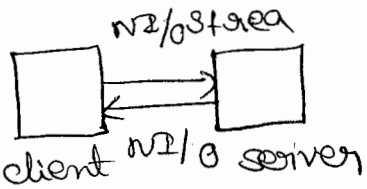


2)



3) 

4) 

5) 

predefined streams!-

1) in
2) out
3) err

1) in!- in is an object reference of java.io.InputStream class

2) out!- out is an object reference of java.io.printStream class.

3) err!- err is an object reference of java.io.printStream class

4) → in, out and err are static members of java.lang.System class.

→ Relationship between system class and InputStream class is "HAS-A" relationship

Note!- system class "HAS-A" Input-stream class and printstream class.

java.lang.system!-

object Reference!-

public static final java.io.InputStream in;
public static final java.io.printStream out;
public static final java.io.printStream err;

java.io.printStream

methods!-

```
public void print(boolean);
public void print(char);
public void print(int);
public void print(long);
public void print(float);
public void print(double);
public void print(char[]);
public void print(String);
public print void print(Object);
public void println();
            println(boolean);
            println(char);
            println(long)
            println(int);
            println(float);
            println(double);
            println(char[]);
            println(String);
            println(Object);
```

→ System.out.print("Taj");
System.out.print("Mahal");

o/p:- TajMahal.

→ System.out.println("Taj");
System.out.println("mahal");

o/p:- Taj
mahal.

→ System.out.println("Taj");
System.out.println();
System.out.println("mahal");

o/p:- Taj

mahal.

java.io.InputStream:-

method:-

public int read(byte[])throws IOException;

→ It is used to read data from keyboard.

Task:-①

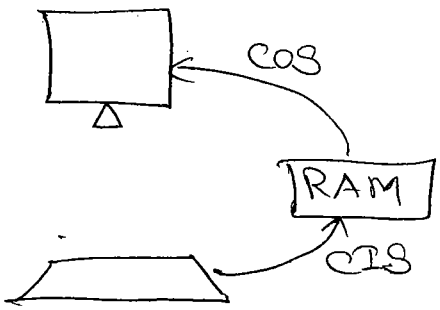
C:\> javac demo.java↵

C:\> java Demo↵

Enter any number: 15↵

15

Task diagram:-



program to demonstrate predefined streams:-

```
import java.io.*;
class demo
{
  psvm(String args[])
  {
    try{
      byte b[] = new byte[10];
    System.out.print (" Enter any number: ");
     System.in.read(b);
     String s1 = new String(b);
     String s2 = s1.trim();
     int x = Integer.parseInt(s2);
      System.out.println(x);
    }catch(IOException ie)
     {
       System.err.println(ie);
     }
  }
}
```

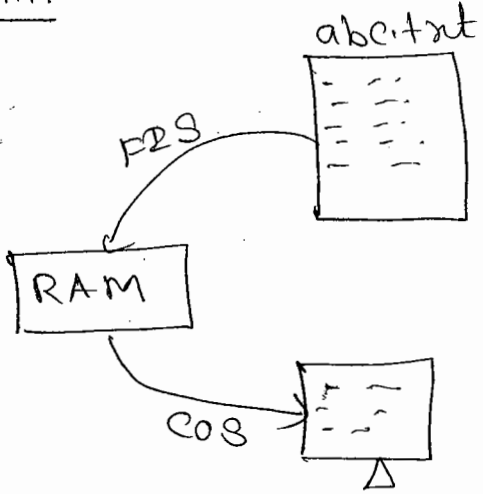
Task 2

c:\s javac Readdemo.java↵

c:\s java Readdemo abc.txt↵

content of abc.txt file

Diagram:-



Steps to develop the above application:-

1) open a file for reading
2) Find out file size
3) Allocate memory ~~according~~ in a RAM according to filesize
4) Read data from file
5) write data to a monitor
6) close the file.

30/3/15

java.io.FileInputStream

dir *.java

Constructors:-

public FileInputStream(String) → filename
    throws FileNotFoundException;

→ It opens a file for reading if the file is exist, otherwise FileNotFoundException will be thrown

Methods:-

public int read(byte[]) throws IOException.

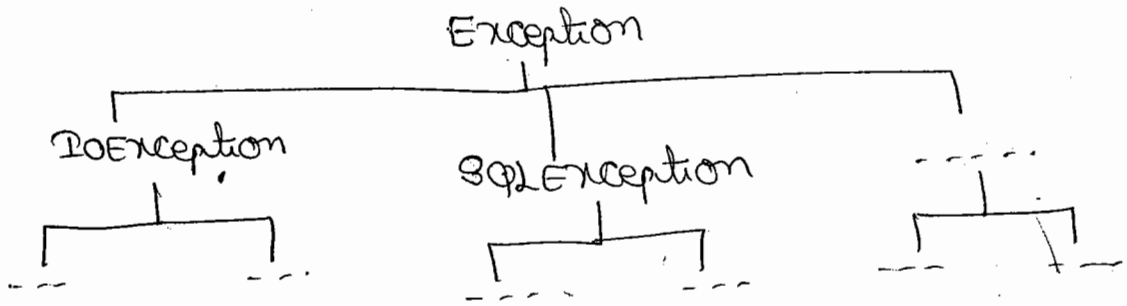
public native int available() throws IOException.

→ It returns file size

public void close() throws IOException

program to Read Data from file By using FileInputStream class:

```
import java.io.*;
class ReadDemo
{   public static void main(String args[])
    {   try{
FileInputStream fis = new FileInputStream(args[0]);
        int n = fis.available(); → Instance method accessing statement
        byte b[] = new byte[n];
        fis.read(b);
      String s = new String(b);
       System.out.println(s);
      }catch(Exception e)
      {  System.err.println(e);
      }
    }
}
```



1) try
{ ==
}catch(Exception e)
{ ==
}

→ This catch block handles Exception class and all its sub classes. it means all Exceptions because all Exceptions are derived from Exception class

2) try
{ ==
}catch(SQLException se)
{ ==
}

→ This catch block handles SQLException and it all its subclasses

3) try
{ ==
}catch(IOException ie)
{ ==
}

→ This catch block handles IOException and all its sub classes.

4) try
{ ==
}
catch(IOException ie)
{ ==
}
catch(SQLException se)
{ ==
}

→ This catch block handles IOException and all its subclasses

This catch block handles SQLException and all its subclasses

alternative to a way to 4th example is 5th one.

5) try
{ ≡
}
catch(IOException | SQLException e)
{ ≡
}

This catch block handles IOException, SQLException and subclasses of IOException and SQLException. This concept is called as handling multiple exceptions with single catch block. It is introduced in JDK 1.7 version in 2011

Program to demonstrate finally block :-

```
import java.io.*;
class ReadDemo
{
  psvm(String args[])
  {
    FileInputStream fis = null;
    try{
  fis = new FileInputStream(args[0]);
    int n = fis.available();
    byte b[] = new byte[n];
     fis.read(b);
    String s = new String(b);
    System.out.println(s);
    }catch(Exception e)
    {
      System.err.println(e);
    }
    finally
    {
        try{
          fis.close();
          }catch(Exception e)
          {
            System.err.println(e);
          }
  } } }
```

try with Resource statement:-

Syntax:-

```
try(resource)
{
  ==
}
```

This feature allows to write resource with try block. a java class that implements java.io.Closable interface only can be used as resource with try block

→ a class that implements ~~java.io.Closable interface~~ java.io.Closeable ~~can~~ Interface can only used as a resource. This resource automatically closed even Exception occurs in a program

→ This concept is an alternative way to a finally block

→ This feature Introduced in JDK 1.7 version in 2011

program to demonstrate try with resource statement

```
import java.io.*;
class ReadDemo
{
  psvm(String args[])throws Exception
  {
  try(FileInputStream fis = new FileInputStream
                    (args[0])){
        byte b[] = new byte[fis.available()];
            fis.read(b);
      System.out.println(new String(b));
    }   }
  }
```

→ The above example FileInputStream is closed automatically even Exception occurs in a program

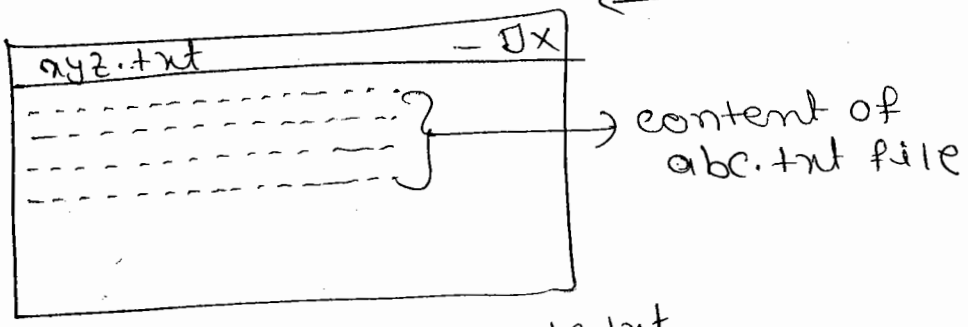
`javac -version` → type in cmd to check java version

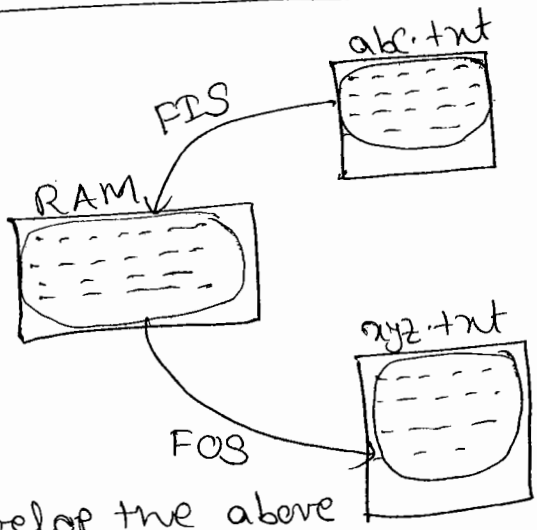
Task

c:\>javac copyDemo.java ↵

c:\>java copyDemo abc.txt xyz.txt ↵

c:\>java notepad xyz.txt ↵





Steps to develop the above application:-

1) open a file for reading
2) find out file size
3) allocate memory in a RAM according to file size
4) Read data from file
5) open a file for writing
6) write data to a file
7) close both the files

library:-
java.io.FileOutputStream:-

constructors:-
public FileOutputStream(String) ← filename
throws FileNotFoundException;

→ It opens a file for writing

public FileOutputStream(String, boolean) throws FileNotFoundException;
(String → file name; boolean → append mode)

→ It opens a file for appending.

Methods:-

public void write(byte[]) throws IOException;

public void close() throws IOException;

NOTE:-

In write mode existed data erased

Program to copy data from one file to another file

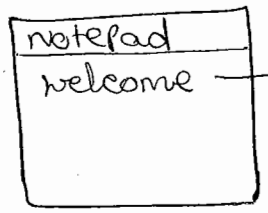
```
import java.io.*;
class copyDemo
{
 public static void main(String args[])
  {  try{
FileInputStream fis = new FileInputStream(args[0]);
     int n = fis.available();
     byte b[] = new byte[n];
        fis.read(b);
FileOutputStream fos = new FileOutputStream(args[1],true);
      fos.write(b);
      }catch(Exception e){
        {
           System.err.println(e);
        }
     }
}
```

Difference between System.out and System.err

System.out	System.err
→ It is used to display output message	→ It is used to display error message.
→ This stream can be redirected.	→ This stream can not be redirected.

```
class Demo
{
    psvm(String args[])
    {
        System.out.println("welcome");
        System.out.println("Hello");
    }
}
```

```
c:\>javac Demo.java
c:\>java Demo
    welcome
    Hello.
c:\> java Demo > a.txt
    Hello
c:\> start notepad a.txt
```



→ out - messages only displayed or redirected to txt file.

Types of streams

1) byte Streams
2) character streams

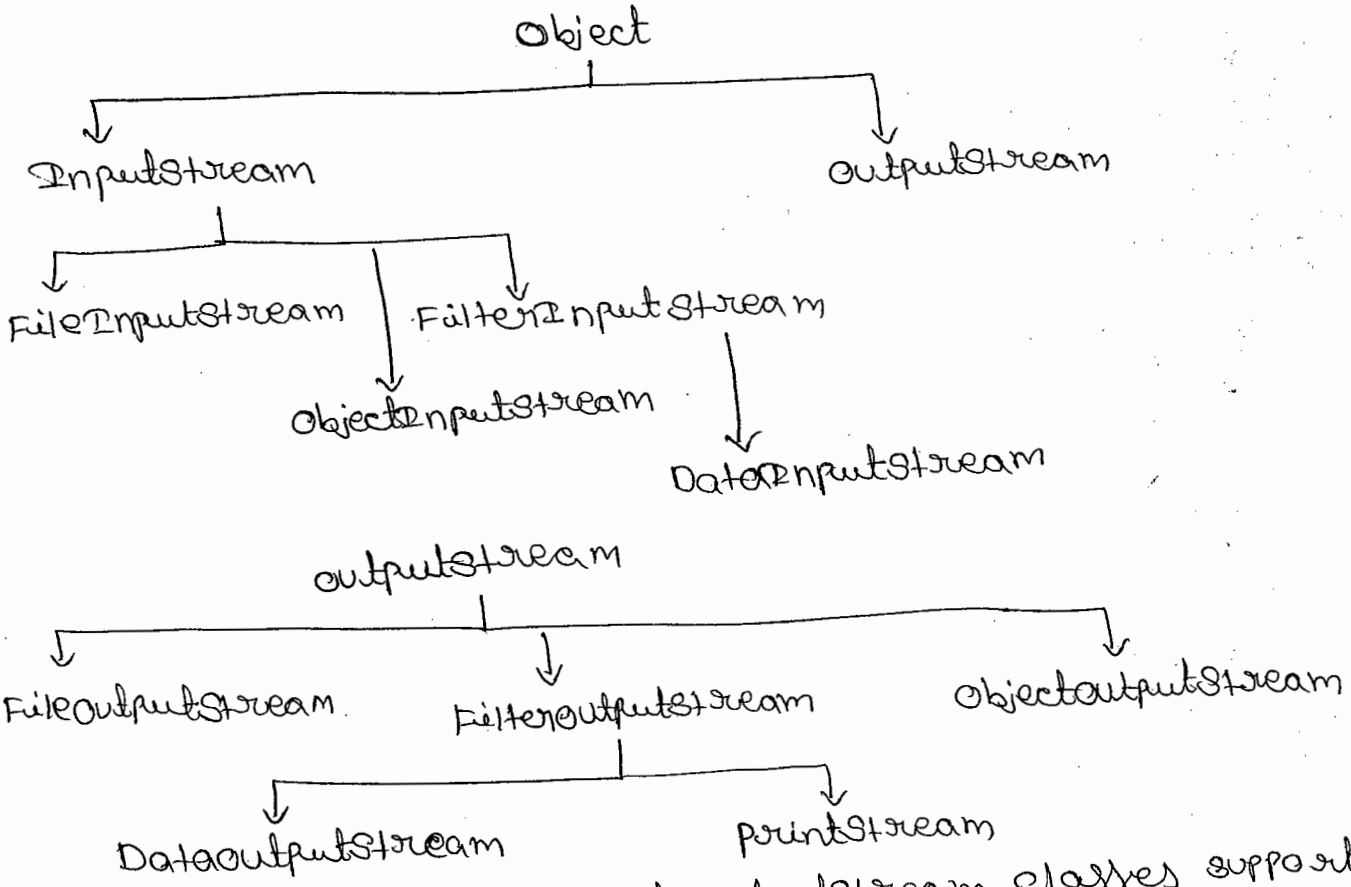
1) byte streams :-

These streams handle text, image, graphics, animation, audio, video — .etc.

2) Character Streams :-

These streams handle only text

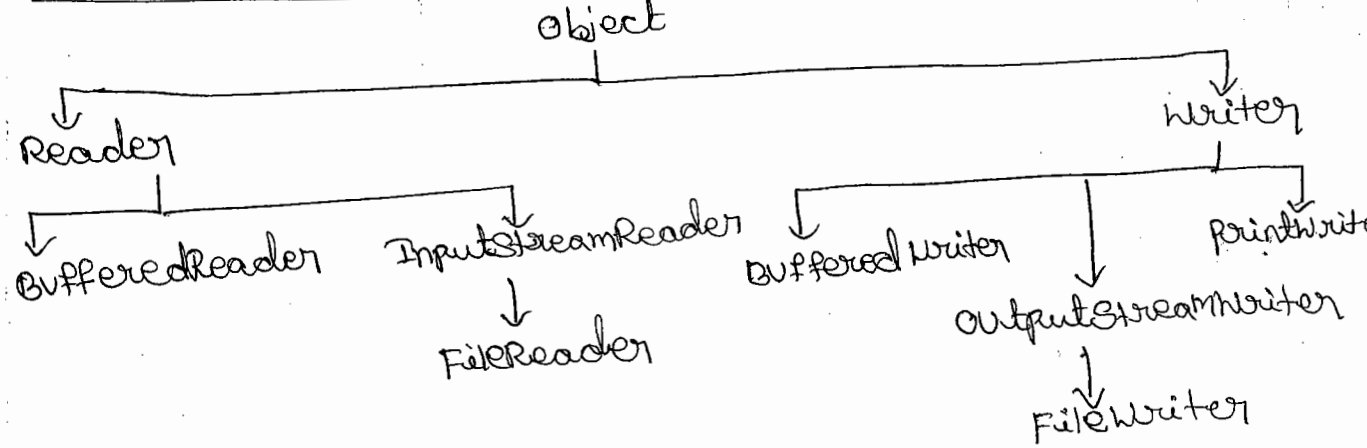
The Heirarchy of byte stream classes



→ DataInputStream and DataOutputStream classes support all primitive datatypes and strings.

→ ObjectInputStream and ObjectOutputStream classes support objects. eg(empobj, studentobj)

The Hierarchy of CharacterStream classes



ObjectStreams:-

1) ObjectInputStream
2) ObjectoutputStream

1) ObjectInputStream:-

It is used to read object from file or network.

2) ObjectoutputStream:-

It is used to write object to a file or network.

java.io.ObjectInputStream

Constructor:-

public ObjectInputStream(InputStream) throws IOException;

Methods:-

public final object readObject() throws IOException, ClassNotFoundException; → for reading object

public void close() throws IOException;
↳ to close the object

java.io.ObjectoutputStream:-

constructor:-

public ObjectoutputStream(outputStream) throws IOException;

Methods:-

public final void writeObject(object) throws IOException;

public void close() throws IOException;

1/4/15

Serialization:-

It is a process of converting object into series of bits

In java object must be serializable to do the following operations

1) writing object to a file
2) Reading object from file
3) writing object to a network
4) Reading object from network

class must Implements java.io.Serializable Interface to make Serializable object

java.io.Serializable Inteface is called as marker Interface, Tag Interface, or Empty Interface because it does not contain members

program to demonstrate serialization or) program to demonstrate object streams

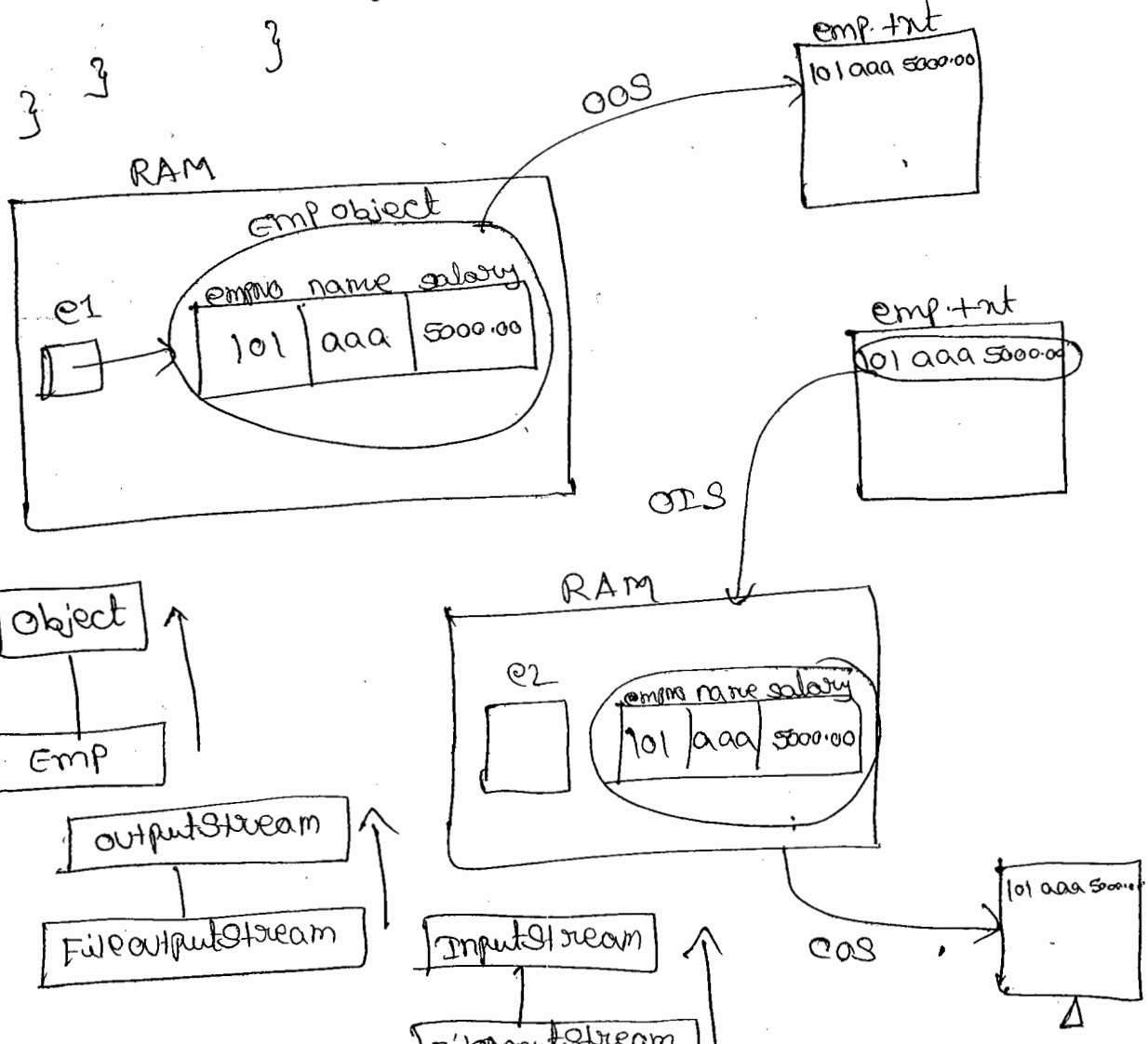
---

```
import java.io.*;
class Emp implements Serializable
{   int empNO = 101;
    String name = "aaa";
    float salary = 5000.00f;
}
class Demo
{   public static void main(String args[])
    {   try{
            Emp e1 = new Emp();
FileOutputStream fos = new FileOutputStream("emp.txt");
ObjectOutputStream oos =
            new ObjectOutputStream(fos);
```

⇒ It is used to convert the object into series of bits

If we want to convert bits then implements serializable

```
        oos.writeObject(e1);
        oos.close();
        fos.close();
FileInputStream fis = new FileInputStream("emp.txt");
ObjectInputStream ois = new ObjectInputStream(fis);
    Emp e2 = (Emp)ois.readObject();
System.out.println(e2.empNo + "\t" + e2.name + "\t" +
                                        e2.salary);
        ois.close();
        fis.close();
      }catch(Exception e)
          {
            System.err.println(e);
          }
      }
}
```



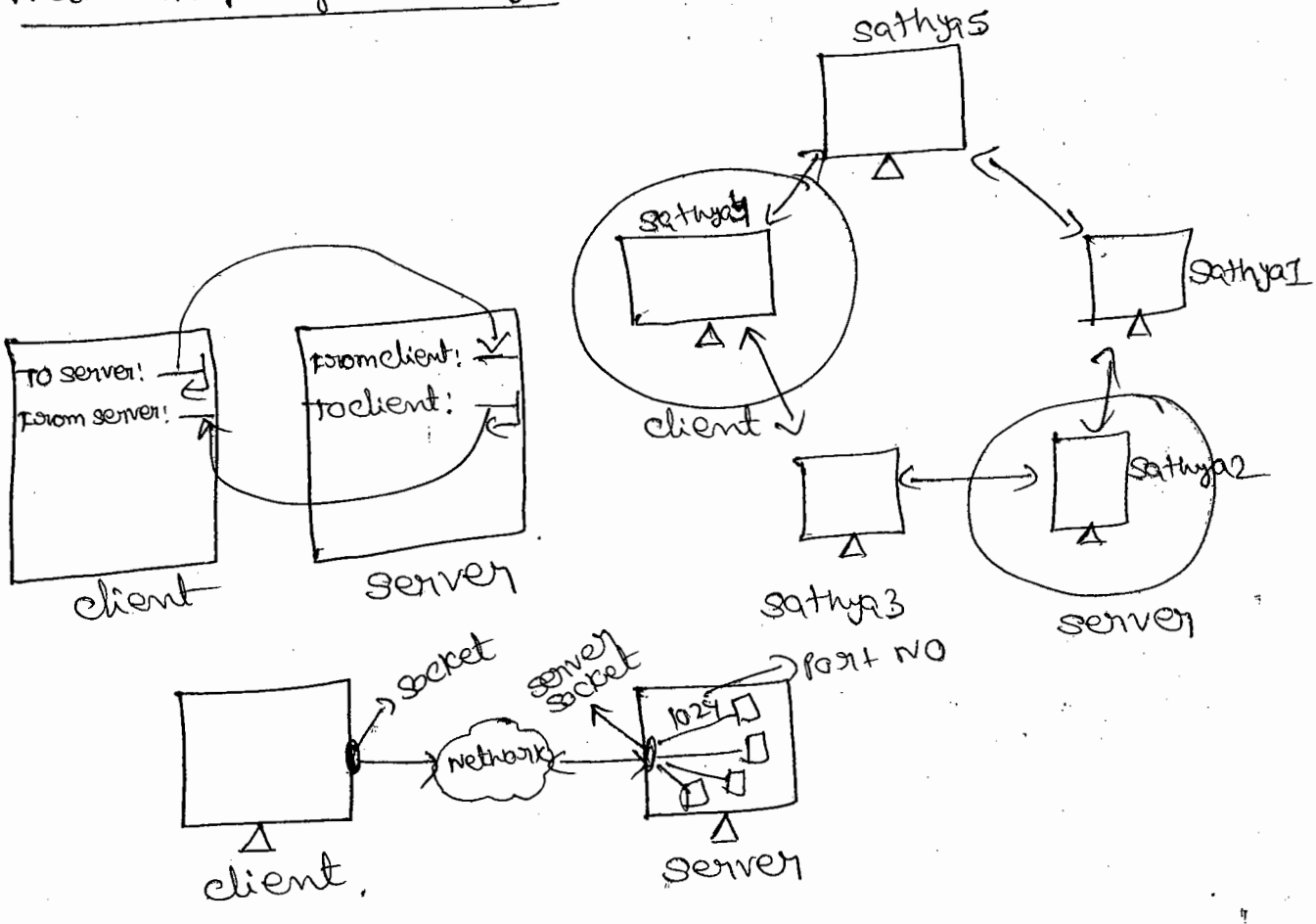
transient keyword :- (It is used to prevent serialization)

It is called as modifier and it is used to prevent serialization. it is used in real time with passwords, pin numbers, security code — etc

Example :-

```
transient String password;
```

Network programming :-



Socket :-

It is a connection end at client side

Server socket :-

It is a connection end at server side

Port no :-

It is used to Identify the service

0 to 1023 Reserved ports

1024 to 65535 Free ports.

Examples:-

http port no. is 80
ftp port no. is 21
telnet port no. is 23
---- etc.,

http protocol transfers hypertext
ftp protocol transfers files
telnet protocol allows remote login

→ Reserved port numbers can not be changed whereas free port numbers can be changed

Steps to develop client application:-

1) create a socket object with server address & port number
2) create an output stream that can be used to send information to the server
3) create an input stream that can be used to receive Information from the server.
4) Do Input and output operations
5) close a socket

Steps to Develop server application:-

1) create a server socket object with port number
2) call accept() method to receive request from the client
3) create an input stream that can be used to receive Information from client
4) create an output stream that can be used to send Information to the client
5) Do input and output operations
6) close a server socket

Library :-

java.net.socket :-

constructor :-

server Address.

public Socket(String, int) throws UnknownHostException, IOException.

→ Port No.

→ It is used to create a socket at client side.

Methods :-

public InputStream getInputStream() throws IOException;

public outputStream getoutputStream() throws IOException;

public synchronized void close() throws IOException;

java.net.ServerSocket :-

constructor :-

public ServerSocket(int) throws IOException;

→ It is used to create a socket at server side

Methods :-

public Socket accept() throws IOException.

public void close() throws IOException;

c:\> javac client.java

c:\>

Program :-

```
import java.io.*;
import java.net.*;
class client
{
  public static void main(String args[])   // command line arguments received by main method
  {
      try{
      String add = args[0];
      int port = Integer.ParseInt(args[1]);
net package{ Socket s = new Socket(add, port);   // creates socket object
  OutputStream os = s.getoutputStream();   // creates o/p stream sending information
IO pkg{ InputStream is = s.getInputStream();   receiving
    byte b1[] = new byte[100];
    byte b2[] = new byte[100];
    while(true)
    {
    System.out.print("To server :");   // displaying.
     System.in.read(b1);   // reads the data from keyboard
      os.write(b1);   // send the data to server
      is.read(b2);   // receiving data from server
    String s1 = new String(b2);   // converting bytearray to string
     String s2 = s1.trim();   removing extra spaces
     sopln("From server:" + s2);
    }
   }catch(Exception e)
     {
       System.err.println(e);
     }
}
}
```

```
import java.io.*;
import java.net.*;

class server
{
   psvm(String args[])
   {
     try{
      int port = Integer.parseInt(args[0]);
  ServerSocket ss = new ServerSocket(Port);
     while(true)
      {
        Socket s = ss.accept();
 InputStream is = s.getInputStream();
 OutputStream os = s.getOutputStream();
    byte b1[] = new byte[100];
    byte b2[] = new byte[100];
    while(true){
     {     is.read(b1);
           String s1 = new String(b1);
           String s2 = new s1.trim();
       sopln("From client :" +s2);
       System.out.print("to client:");
            System.in.read(b2);
             os.write(b2);
   } }
     }catch(Exception e)
      {
        System.err.println(e);
      }
} }
```

C: client side

c:\>java client localhost 2000

To server : Hello

From server : Hi

To server : How r U?

From server : fine

To server :

first server executed

server side

c:\>java server 2000

From client : Hello

To client : Hi

From client : How r U?

To client : fine

Collections Framework

a set of collection classes and interfaces is called as collections Framework. (or) a set of data structures related classes and interfaces is called as collections Framework

Collection :-

a collection is an object that represents group of objects

Advantages of collections Framework :-

1) Reduces programming efforts
2) Increases programming speed and quality
3) allows Interoperability among unrelated APIs.

a set of data structures related classes and interfaces is also called as collections Framework.

Examples of data structures :-

1) stack
2) queue
3) Tree
4) Graph

Data Representation methods:-

1) Array Representation (Linear Representation or Linear list)
2) Linked Representation (Linked list)
3) Indirect Addressing
4) Simulating pointers

Techniques:-

1) Sorting
2) Searching
3) Hashing.

→ all collection classes and Interfaces are the part of java.util package

→ java.util Package classes interfaces are divided into two categories

1) Collections Framework Collections
2) Legacy Collections.

1) Collections Framework Collections:-

JDK 1.2 and above versions ~~etc~~ collection classes and interfaces are called collections Framework Collections.

2) Legacy collections:-

JDK 1.0 and 1.1 versions collection classes and interfaces are called legacy collections.

① Collections Framework Collections:-

There are divided into 3 sub categories

1) core collection Interfaces
2) General purpose Implementations
3) more utility collections.

1) Core collection interfaces :-

There are the foundation of collections framework

1) Collection
2) List
3) Set
4) Map
5) SortedSet
6) SortedMap
7) NavigableSet
8) NavigableMap
9) Queue
10) Deque

} Interfaces

1) Collection :-

It is a root interface in collections Hierarchy

2) List :-

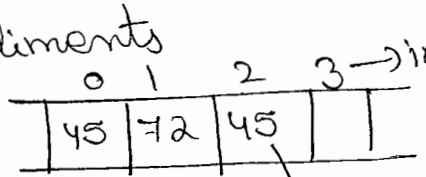
It extends collection Interface and it maintains sequences

3) Set :-

It extends collection Interface and it maintains sets

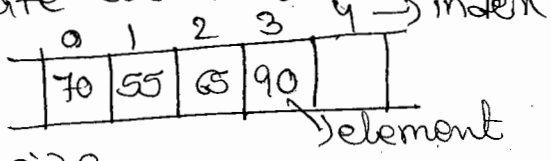
Differences between List and Set

List	Set
1) It maintains sequences	1) It maintains sets
2) It allows dup duplicate elements	2) It does not allow duplicate elements



size : 3
(or)
No. of elements : 3

capacity : 4



size
(or)
No. of elements : 4

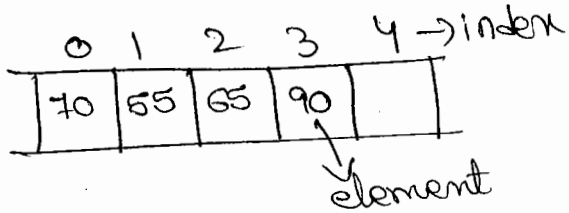
capacity : 5

Map:-

It is a two dimensional collection and it maintains data as a key/value pairs

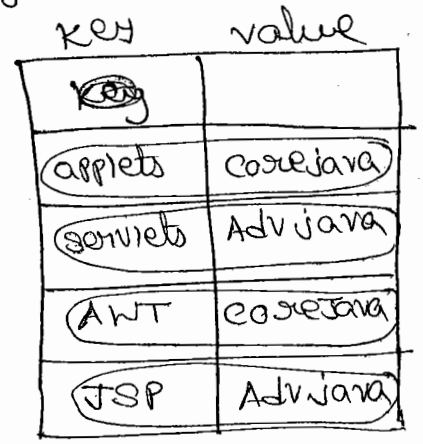
Differences between a Set and Map

Set	Map
1) It is a one dimensional collection	1) It is a two dimensional collection
2) It is a index based collection	2) It is a key based collection
3) It does not allow duplicate elements	3) It does not allow duplicate keys. (values may be duplicated)



size (or) no. of elements : 4

capacity : 5



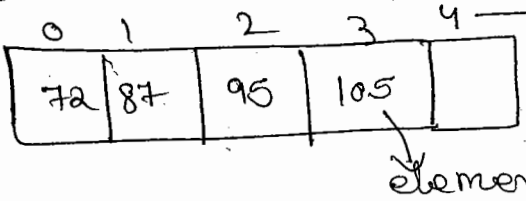
size: (or) no. of key/value pairs : 4

capacity : 6

5/4/15

SortedSet:-

It extends Set Interface and it maintains Sorted Sets



SortedMap:-

It extends Map Interface and it maintains Sorted Maps

Here only keys are sorted

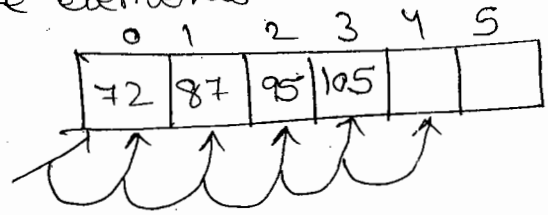
Sortedmap

key	value
applets	corejava
JSP	Adv.java
Servlets	Adv.java
String	corejava

7) Navigableset:-

It extends SortedSet Interface and it is used to navigate elements of a set

Ex:-



8) navigablemap:-

It extends SortedMap Interface and it is used to Navigate key value pairs of a map

key	value
applets	corejava
JSP	Adv.java
Servlets	Adv.java
String	corejava

9) Queue:

It is called as First in first out list because it allows Insertion at Rare end and Deletion at front end

10) Deque:-

It is called as double ended queue because It allows both Insertion and deletion at both the ends (front end and rare end)

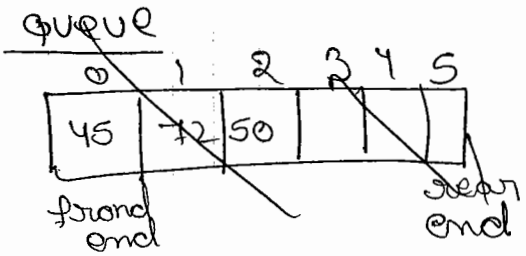
differences between queue and deque

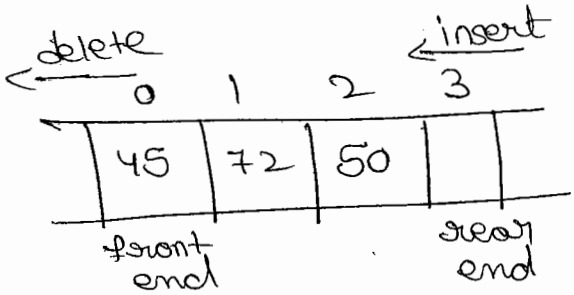
queue

1) It is a first in first out list

2) It allows Insertion at ~~rear~~ rear end only

3) It allows Deletion at front end only



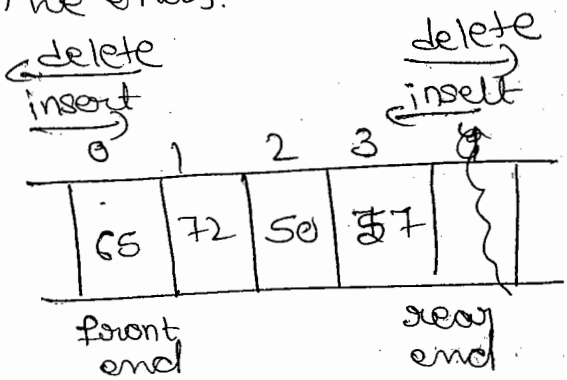


deque

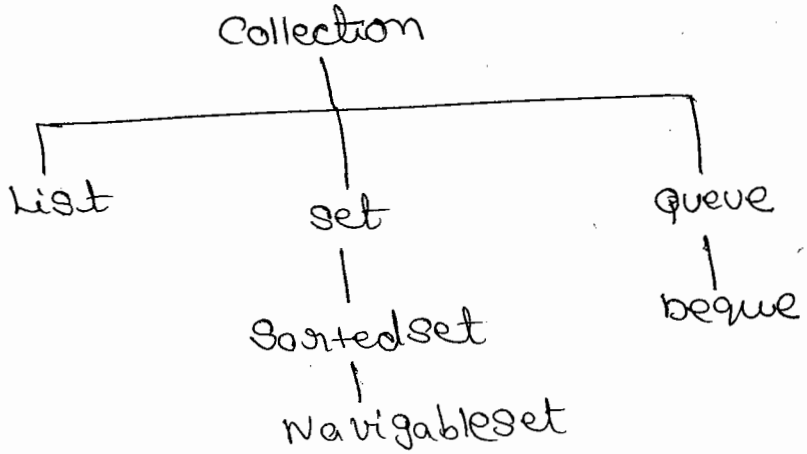
1) It is not a first in first out list

2) It allows Insertion at both the ends

3) It allows deletion at both the ends.



The Hierarchy of one dimensional collections



The Hierarchy of two dimensional collection

Generalpurpose Implementations :-

core collection Interfaces implementation classes are called generalpurpose Implementations

1) Arraylist
2) LinkedList
3) HashSet
4) LinkedHashSet
5) TreeSet
6) HashMap
7) LinkedHashMap
8) TreeMap
9) Priorityqueue
10) ArrayDeque

Arraylist :-

It is an implementation of linear list datastructure

It supports all the operations of linear list data structure

Library :-

java.util.Arraylist

Constructor :-

public Arraylist();

Methods:-

```
public int size();
public object get(int);          <-- index
public object set(int, object);  <-- element
public boolean add(object);
public void add(int, object);
public object remove(int);
public boolean remove(object);
```

It is used to modify an element at specified index

It is used to insert an element at specified index.

program to demonstrate arraylist:-

```
import java.util.*;
class Demo
{
    psvm(String args[])
    {
        ArrayList al = new ArrayList();
        al.add(23);
        al.add(99);
        al.add(82);
        al.add(1, 72);
        System.out.println(al);
    }
}
```

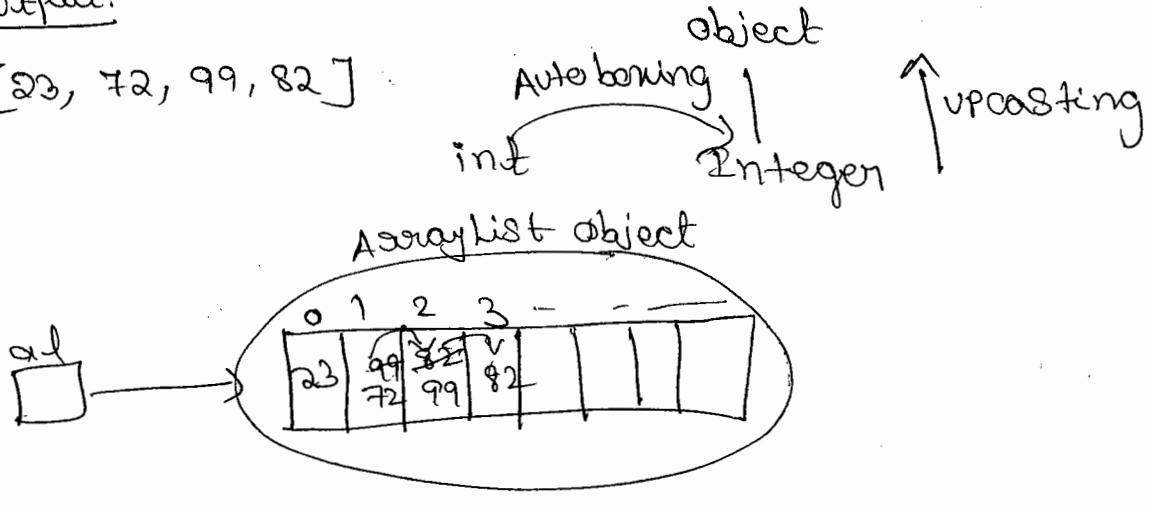
Auto boxing and upcasting

al.toString

Implicitly

output:-

[23, 72, 99, 82]



In the above example a Abstract collection class toString() method is called.

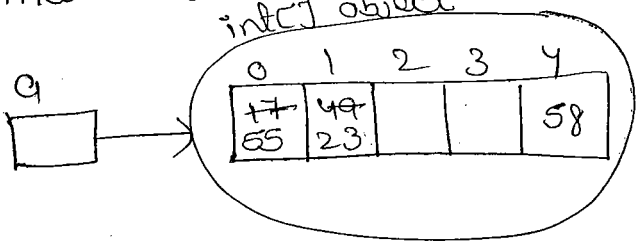
Differences between Arrays and collection @?)

Arrays and & ArrayList

Arrays	Collections (Arraylist)
1) It is a collection of similar data elements	1) Arraylist is a collection class
2) The size of Array can not grow and can not ~~shrunk~~ shrank	2) The size of collection can grow and can shrank dynamically as needed.

Array

```
int a[] = new int[5];
```

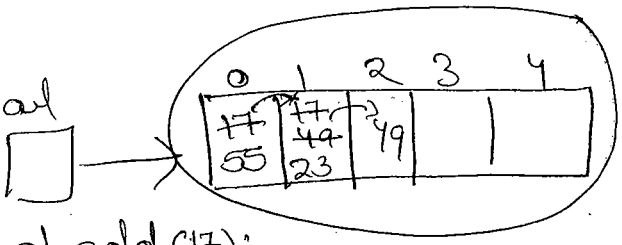



```
a[0] = 17;
a[1] = 49;
a[0] = 55;
a[4] = 58;
a[1] = 23;
for(int i=0; i<a.length; i++)
{
    S.o.P(a[i]);
}
```

O/P
55
23
0
0
58

Arraylist:-

```
Arraylist al = new Arraylist();
```




```
al.add(17);
al.add(49);
al.add(0,55);
al.add(4,58); => Error
al.set(1,23);
S.o.P(al);
```

output:-

[55, 23, 49]

LinkedList:-

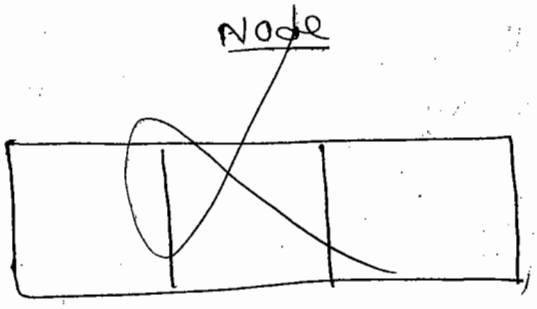
It is an implementation of double linked list data structure.

→ It supports all the operations of double linked list data structure

Linked list is a collection of nodes.

→ In double linked list each node contains 3 parts

1) ~~previous node address~~
2)



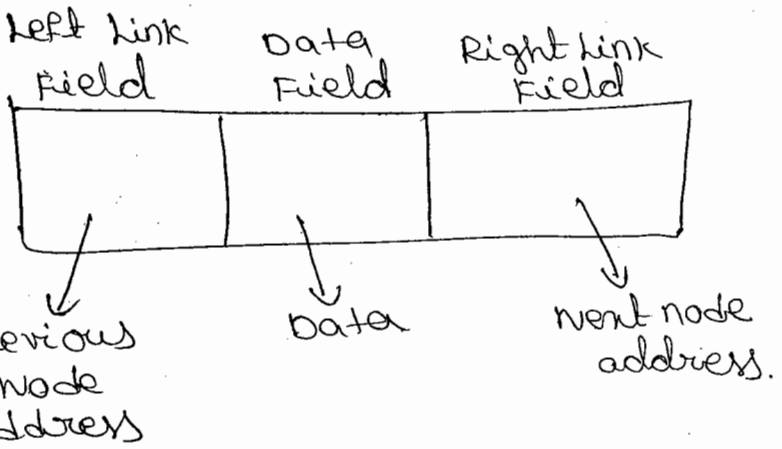
1) Left link field
2) Data field
3) Right link field

→ Left link field contains previous node address

→ Data field contains data

→ Right link field contains next node address

Node



java.util.LinkedList:-

Constructor:-

```
public LinkedList();
```

Methods:-

```
public object getFrist();
public object getLast();
public object removeFirst();
public object removeLast();
public void addFirst(Object);
public void addLast(object);
```

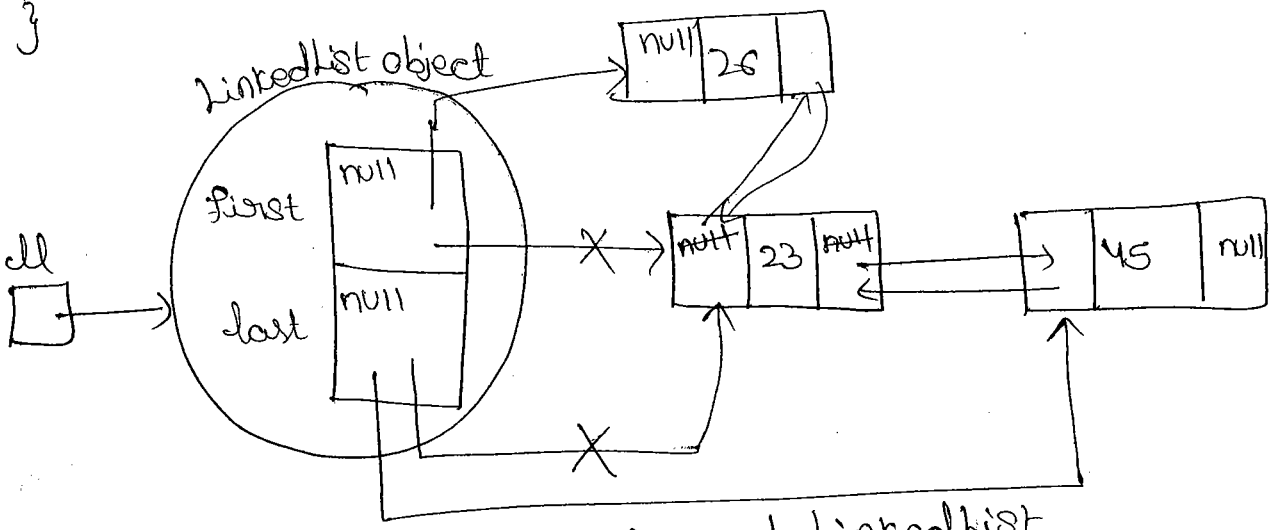
→ It is equivalent to addLast() method.

public boolean remove(Object);
public void add(int, object);

→ It is used to insert an element at specified index.

Program to Demonstrate LinkedList :-

```
import java.util.*;
class Demo
{
    psvm(String args[])
    {
        LinkedList ll = new LinkedList();
        ll.add(23);
        ll.add(45);
        ll.addFirst(26);
        System.out.println(ll);
    }
}
```



Differences b/w ArrayList and LinkedList

ArrayList	LinkedList
1) It is a linear list Data Structure	1) It is a double linked list data structure
2) It occupies less memory	2) It occupies more memory because data stored in nodes
3) In ArrayList insertion and deletion operations require shuffling of data	3) In linked list it doesn't require shuffling of data

Hashset:-

It is an implementation of Hashing technique with Array representation

Hashing:-

Hashing is a technique which allows insertion, deletion and find in a constant average time

Ex:- Dictionary.

Hashset does not allow duplicate elements because it implements Set Interface

program to demonstrate Hashset

```
import java.util.*;
class Demo
{
    psvm(String args[])
    {
        Hashset hs = new Hashset();
        hs.add(23);
        hs.add(45);
        hs.add(99);
        hs.add(54);
        soln(hs);
    }
}
```

o/p:- can not be predicted.

LinkedHashset:-

It is an implementation of Hashing technique with Linked representation

```
import java.util.*;
class Demo
{
    psvm(String args[])
    {
LinkedHashset lhs = new LinkedHashset();
        lhs.add(23);
        lhs.add(45);
        lhs.add(99);
        lhs.add(54);
        soln(lhs);
} }
```

o/p:- [23 45 99 54]

Treeset:-

It is an implementation of Binary search tree technique with linked representation.

Binary search tree:-

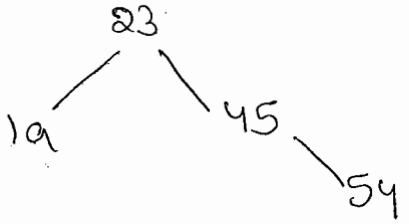
a Binary tree is called as Binary search tree if it follows the following rules.

→ if the element is less than root element then it must be left subtree

→ if the element is greater than root element then it must be right subtree.

Program to Demonstrate the Treeset:-

```
import java.util.*;
Class Demo
{
    PSVM(String args[])
    {
        Treeset ts = new Treeset()
        ts.add(23);
        ts.add(45);
        ts.add(19);
        ts.add(54);
        SOPln(ts);
    }
}
```



o/p:-

[19, 23, 45, 54]

6/4/15

Differences b/w Hashset, LinkedHashset and Treeset

Hashset	LinkedHashset	Treeset
i) It is an implementation of Hashing technique with array representation	i) It is an implementation of Hashing technique with linked representation.	i) It is an implementation of Binarysearch technique with linked representation.
ii) It occupies less memory	ii) It occupies more memory because data stored in nodes	ii) It occupies more memory because data stored in nodes.
iii) Hashset is unordered set and unsorted set	iii) LinkedHashset ordered set	iii) It is sorted set

Hashmap:-

It is an implementation of Hashing technique with array representation.

It is two dimensional collection class and it maintains data as a key value pairs

program to Demonstrate Hashmap:-

```
import java.util.*;
class Demo
{
  public static void main(String args[])
  {
    Hashmap hm = new Hashmap();
    hm.put("servlets", "Adv.Java");
    hm.put("applets", "core Java");
    hm.put("swing", "core Java");
    hm.put("jsp", "Adv. Java");
    Sopln(hm);
  }
}
```

O/p:- Can not be predicted.

Differences b/w Hashset and Hashmap

Hashset	Hashmap
1) It is a one dimensional collection	1) It is a two dimensional collection
2) It is a index based collection	2) It is a key based collection
3) Hashset does not allow duplicate elements	3) Hashmap does not allow duplicate keys.

LinkedHashMap:-

It is an implementation of Hashing technique with linked representation.
It is also two dimensional collection and it maintains data as a key value pairs

program to demonstrate LinkedHashmap

```
import java.util.*;
class demo
{  psvm(String args[])
   {
LinkedHashmap lhm = new LinkedHashmap();
    lhm.put("servlets", "Adv.Java");
    lhm.put ("applets", "core.Java");
    lhm.put ("swing", "core.Java");
    lhm.put("jsp", "Adv.Java");
      sopln(lhm);
   }
}
```

o/p:- { servlets = Adv.Java, applets = core Java,
swing = core.Java, JSP = Adv.Java }

TreeMap:-

It is an implementation of BinarySearchTree technique with linked representation.

→ It is also two dimensional collection class and it maintains data as a key value pairs

program to demonstrate tree map:-

```
import java.util.*;
class Demo
{
    psvm(String args[])
    {
     TreeMap tm = new TreeMap();
      tm.put("servlets", "Adv.Java");
      tm.put("applets", "core Java");
      tm.put("swing", "core Java");
      tm.put("jsp", "Adv.Java");
       sopln(tm);
    }
}
```

o/p:- { applets = coreJava, jsp = Adv.Java,
servlets = Adv.Java, Swing = coreJava }

→ Hashmap is an unordered and an sorted map

→ LinkedHashmap is ordered map

→ TreeMap is sorted Map (only keys are sorted)

priorityqueue:-

priorityqueue is a queue which allows Insertion and deletion based on priorities

Library:-

java.util.priorityqueue

constructor:-

public priorityqueue();

methods:-

public boolean offer(object);

→ It is used to insert an element into a priorityqueue

public object poll();

→ It is used to remove an element from priorityqueue.

```
import java.util.*;
class Demo
{
    psvm(String args[])
    {
    PriorityQueue pq = new PriorityQueue();
        pq.offer("javase");
        pq.offer("javaee");
        pq.offer("javame");
        pq.offer("javafx");
        System.out.println(pq);
    }
}
```

Object but we are passing String it is valid upcasting

Object ↑ upcasting

String

int we pass i.e Autoboxing.

Arraydeque:-

It is an implementation of Doubleended queue data-structure with array representation.

→ It allows Insertion and deletion at both the ends

java.util.ArrayDeque

Constructor:-

```
public ArrayDeque();
```

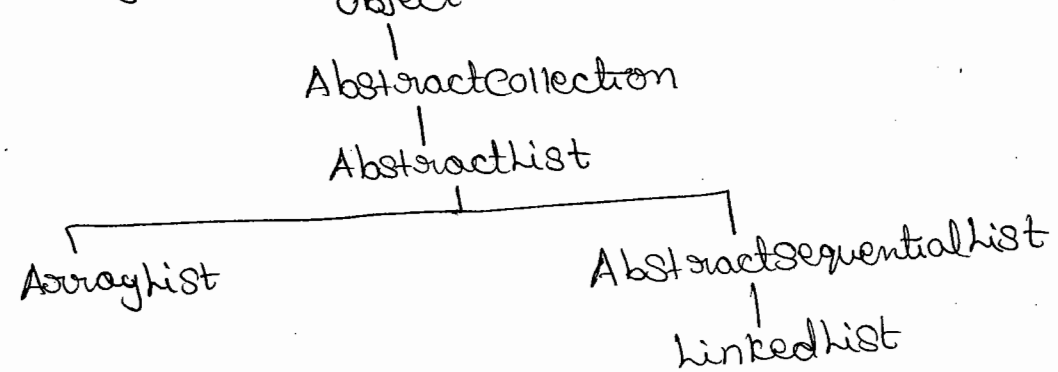
methods:-

```
public void addFirst(Object);
public void addLast(Object);
public Object removeFirst();
public Object removeLast();
public boolean add(Object);
```

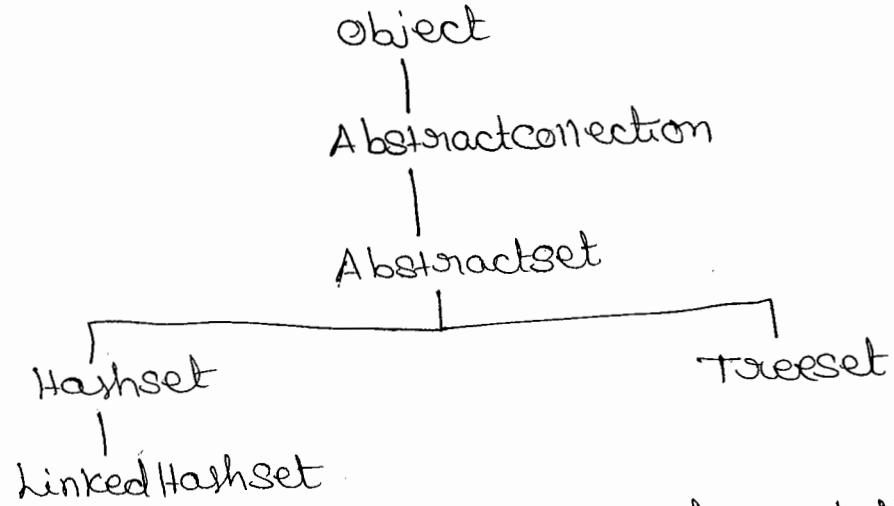
→ It is an equivalent to addLast() method.

```
import java.util.*;
class Demo
{
    psvm(String args[])
    {
    ArrayDeque ad = new ArrayDeque();
            ad.add(23);
            ad.addFirst(43);        o/p:- [88,43,23,21,92]
            ad.addLast(21);
            ad.addFirst(88);
            ad.addLast(92);
        }       sopln(ad);
}
```

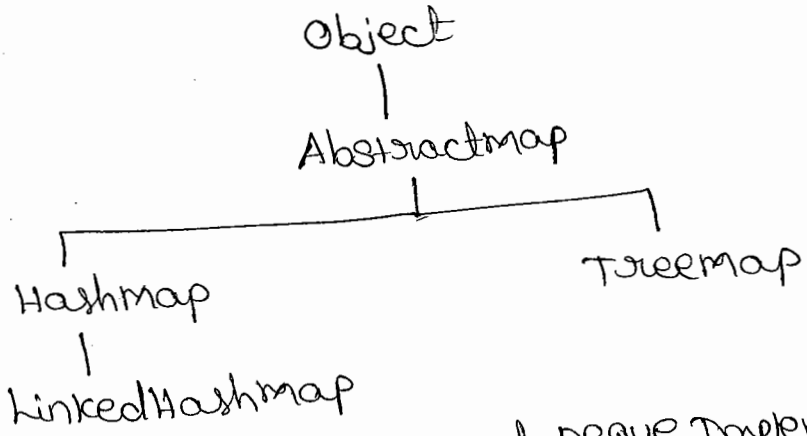
The Hierarchy of List implementation classes:-



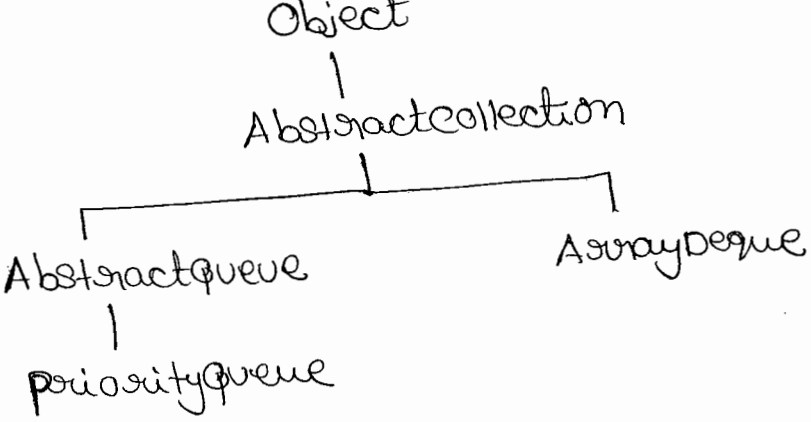
The Hierarchy of set implementation classes:-



The Hierarchy of map Implementation classes:-



The Hierarchy of queue and Deque Implementation classes



3) More utility collections:-

1) Iterator (Interface)
2) ListIterator (Interface)
3) Arrays (class)
4) Collections (class)
5) Scanner (class)

1) Iterator (Interface)

It is used to Iterate elements of any collection

java.util.Iterator

Methods:-

public abstract boolean hasNext();

→ It returns true if the element is present, otherwise returns false

public abstract Object next();

→ It returns current element and moves the cursor to next element if the next element is present

public abstract void remove();

→ It removes current element

java.util.AbstractList

Method:-

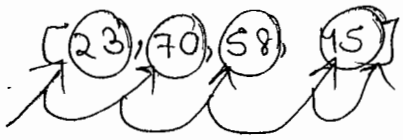
public Iterator iterator();

→ It is used to get the reference of Iterator.

Program to demonstrate Iterator:-

```
import java.util.*;
class Demo
{   psvm(String args[])
    {  ArrayList al= new ArrayList();
       al.add(23);
       al.add(70);
       al.add(58);
       al.add(45);
```

```
        System.out.println(al);
        Iterator i = al.iterator();
        while(i.hasNext())
        {
          System.out.println(i.next());
        }
    }
}
```




```
⇒ while(i.hasNext())
  {
    int x = (Integer)i.next(); ⇒ Downcasting &
    Sopln(x+5);                  Auto unboxing.
  }
```

7/4/15

ListIterator:-

It is used to Iterate elements of list implementation classes only (ArrayList & LinkedList)

java.util.ListIterator

Methods:-

```
public abstract boolean hasNext();
public abstract object next();
public abstract boolean hasPrevious();
public abstract object previous();
public abstract void remove();
```

→ It is used to remove current element

```
public abstract void set(Object);
```

→ It is used to modify current element

```
public abstract void add(Object);
```

→ It is used to add an element at current position

java.util.AbstractList

methods:-

public ListIterator listIterator(); ⌐ index to start
public ListIterator listIterator(int); ⌙

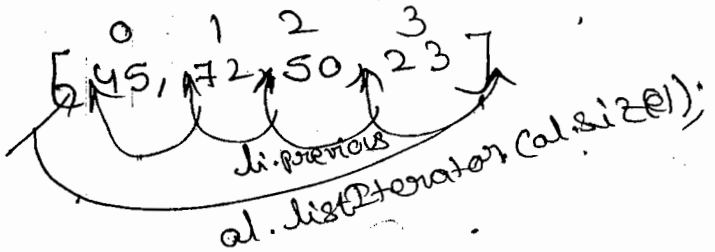
→ The above two methods are used to get the reference of ListIterator

program to demonstrate ListIterator:-

```
import java.util.*;
class Demo
{
    psvm(String args[])
    {
        ArrayList al = new ArrayList();
        al.add(45);
        al.add(72);
        al.add(50);
        al.add(23);
        sopln(al);
ListIterator li = al.listIterator(al.size());
        while(li.hasPrevious())
        {
            s.o.p(li.previous())
        }
    }
}
```

↳ we are passing one more size i.e 4 so cursor move to last

O/P:




```
23
50
72
45
```

Differences b/w Iterator and ListIterator

Iterator	ListIterator
1) It is used to Iterate elements of any Collection	1) It is used to Iterate elements of ArrayList and LinkedList only
2) Iterator allows to Itera-te elements only in forward direction	2) ListIterator allows to Iterate elements in both forward and backward directions.
3) Iterator supports only remove operation while Iterating elements	3) ListIterator ~~allows~~ supports add, remove and set operations while Itera-ting elements

3) Arrays :- (class)

It contains several methods that are used to perform on ~~Arrays~~ arrays.

java.util.Arrays

methods:-

```
public static void sort(int[]);
public static void sort(char[]);
public static void sort(Object[]);
public static int binarySearch(int[], int);
public static boolean equals(int[], int[]);
public static String toString(int[]);
```

element to search

→ It returns string representation of array elements

Program to demonstrate Arrays class

```
import java.util.*;
class Demo
{
    psvm(String args[])
    {
        int a[] = {23, 45, 81, 40, 34};
        String s1 = Arrays.toString(a);
        sopln("Before Sorting ------");
        sopln(s1);
        Arrays.sort(a);
        String s2 = Array.toString(a);
        sopln("After sorting ------");
        sopln(s2);
    }
}
```

4) Collections:- (class)

It contains several methods that are used to perform on collections

java.util.Collections

methods:-

```
public static void sort(List);
public static int binarySearch(List, Object);
public static void reverse(List);
```

element to search

program to demonstrate Collection class:-

```
import java.util.*;
class Demo
{ psvm(String args[])
  { ArrayList al=new ArrayList();
    al.add(23);
    al.add(99);
    al.add(25);
    al.add(65);
    System.out.println(al);
    Collections.reverse(al);
    System.out.println(al);
```

~~o/p:- [23 99 25 65]~~

```
    Collections.sort(al);
      sopln(al);
    Collecton.reverse(al);
      sopln(al);
  } }
```

5) Scanner:-

It is used to read the data from keyboard, file & network

java.util.Scanner

Constructor:-

public Scanner(InputStream);

methods:-

public int nextInt();

public float nextFloat();

program to Demonstrate Scanner class

```
import java.util.*;
class Demo
{
  public static void main(String args[])
   {
    Scanner s = new Scanner(System.in);
    System.out.print("Enter any number:");
          int x = s.nextInt(); -> Instance method
          System.out.println(s); so obj ref.
   }
}
```

O/P:- Enter any number: 33
33.

Legacy Collections:-

JDK 1.0 and 1.1 versions collection classes and Interfaces are called legacy collections.

1) Enumeration(Interface)
2) Vector(class)
3) StringTokenizer(class)
4) Dictionary(Abstract class)
5) Hashtable(class)
6) Stack(class)
7) Random(class)
8) Date(class)

Enumeration:-

It is similar to Iterator Interface
↳

Differences between Enumeration and Iterator:-

Enumeration	Iterator
1) It is a legacy collection	1) It is a Collections Framework collection
2) It does not allow any other operation while Iterating elements	2) It allows remove operation while Iterating elements

2) Vector:-(Class)

It is similar to Arraylist class

Differences between vector and Arraylist

Vector	ArrayList
1) It is a legacy collection	1) It is a collection framework collection
2) methods of vector class are synchronized	2) methods of Arraylist class are not synchronized

3) Stringtokenizer:-

It allows an application to break a string into tokens

4) Dictionary 9/4/15

It is similar to map Interface

→ It is a two dimensional collection and it maintains data as a key/value pairs

→ Dictionary is a legacy collection whereas map is a collections framework collection.

5) Hashtable:-

It is similar to Hashmap

It is an implementation of Hashing technique with array representation

It is a two dimensional collection and it maintains data as a key/value pairs because it extends Dictionary class.

Differences between Hashtable and Hashmap

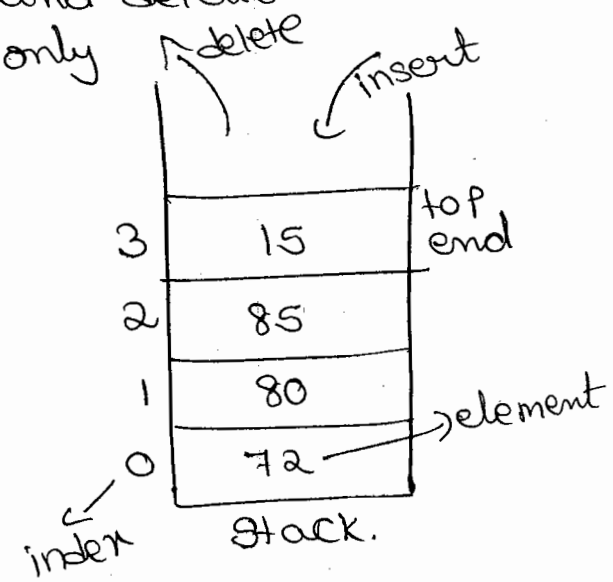
Hashtable	Hashmap
1) It is a legacy collection	1) It is a collections Framework collection.
2) Hashtable doesn't allow null keys & null values	2) Hashmap allows one null key and many null values
3) Methods of Hashtable are synchronized	3) Methods of Hashmap are not synchronized.

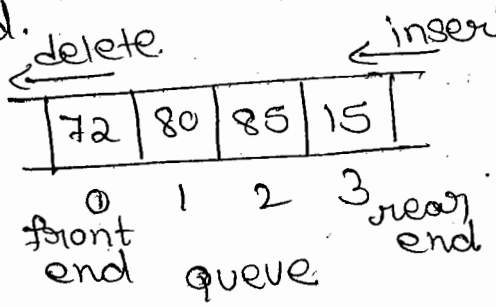
6) Stack (Class):-

It is called as Last in First out list

Differences between Stack and Queue

Stack	Queue
1) It is a last in first out list	1) It is a first in first out list.
2) Stack is a legacy collection	2) Queue is a collections Framework collection.
3) Stack allows both insertion and deletion at top end only	3) Queue allows insertion at rear and deletion at front end.





java.util.Stack

constructor:-

public Stack();

methods:-

public Object push(Object);

→ It is used to insert an element into a stack

public synchronized Object pop();

→ It is used to remove an element from stack

Both are instance methods we have to create object for stack class

```
import java.util.*;
class Demo
{
    psvm(String args[])
    {
        Stack s = new Stack();
        s.push(72);
        s.push(80);
        s.push(85);
        s.push(15);
        System.out.println(s);
        sopln(s.pop());
        sopln(s);
    }
}
```

o/p:- [72, 80, 85, 15]

15

[72, 80, 85].

7) Random :-

It is used to get Random integers, floating point numbers and boolean values

java.util.Random

Constructor:-

public Random();

Methods:-

public int nextInt();

→ It returns one number within int range

public int nextInt(int);

→ It returns one number b/w 0 and specified number

public boolean nextBoolean();

public float nextFloat();

```
import java.util.*;
class Demo
{
    psvm(String args[])
    {
        Random r = new Random();
        sopln(r.nextBoolean());
        sopln(r.nextInt(100));
    }
}
```

O/P: can not predict

8) Date :-

It is used to get system date and time

Java.util.Date :-

Constructor:-

public Date();

Methods:-

```
public int getYear();
public int getMonth();
public int getDate();
public int getHours();
public int getMinutes();
public int getSeconds();
```

```
import java.util.*;
class Demo
{
  psvm(String args[])
    {
      Date d=new Date();
  sopln(d.getHours()+":"+d.getMinutes()+":"
                +d.getSeconds());
    }
}
```

Generics

```
import java.util.*;
class Demo
{
    psvm(String args[])
    {
        Arraylist al=new Arraylist();
          al.add(92);
          al.add(23);
          al.add(42);
          al.add(82);
          al.add("abc");
          sopln(al);
    }
}
```

o/p:- [92,23,42,82, abc]

→ In the above example Arraylist object is unsafe because it allows any type of data.

To write type safe programs we need generics in java

Generics

Generics:-

This feature introduced in JDK 1.5 version (in 2004) to write type safe programs.

Syntax to create an object of Generic class

```
ClassName <ReferenceDataType> ObjectReference = new
                    constructor<ReferenceDataType>();
```

Example:-

```
import java.util.*;
class Demo
{
    psvm(String args[])
    {
ArrayList<Integer> al=new ArrayList<Integer>();
        al.add(92);
        al.add(23);
        al.add(42);
        al.add(82);
      //al.add("abc"); => Error
        sopln(al);
    }
}
```

→ In the above example Arraylist object is typesafe because it allows only Integers

predefined Generic classes :-

1) ArrayList<E>
2) LinkedList<E>
3) HashSet<E>
4) HashMap<K,V>
5) TreeMap<K,V>

} If the class names like this these classes are Generic classes.
→ we can also create our own Generic classes

..... etc.,

Here E means element, K means key V means value.

advantages of Generics :-

1). It allows to write typesafe programs
2) It doesn't require typecasting

10/4/15

1) class Linearlist
{ int x; } only for integers
}

```
Linearlist ll=new Linearlist();
   ll.set(57);✓
   ll.set("abc");X
```
→ Typesafe

Advantage of above program :-

Typesafe because it allows only integers

Disadvantage :-

The above class only for integers

2) class Linearlist
{ object o; } For all types
}

```
Linearlist ll=new Linearlist();✓
   ll.set(23);✓
   ll.set("abc");✓
```
→ Unsafe

advantage of above program :-

The above class for all data types

Disadvantage :-

It is unsafe. because it allows any type of data.

3) class LinearList<T>

```
{
    T t;
}
```

} for all types

typesafe

```
LinearList <Integer> ll1 = new LinearList<Integer>(); ✓
    ll.set(23); ✓
    ll.set("abc"); X
```

```
LinearList <String> ll2=new LinearList <String>(); ✓
    ll2.set(23); X
    ll2.set("abc"); ✓
```

```
LinearList ll3 = new LinearList(); ✓
    ll3.set(23);
    ll3.set("abc"); ✓
```

} unsafe

→ not recommended no use

Advantages :-

1) The above class for all data types
2) It is typesafe because first object allows only integers and 2nd object only allows strings

Program to demonstrate user defined Generic class:-

```
class LinearList<T>
{
    T x;
    void set(T x)
    {
        this.x=x;
    }
    T get()
    {
        return x;
    }
}
class Demo
{
    public static void main(String args[])
    {
LinearList<Integer>l1=new LinearList<Integer>();
                l1.set(23);
                int y=l1.get();
                Sopln(y);
LinearList<String>l12=new LinearList<String>();
                l12.set("abc");
                String s=l12.get();
                Sopln(s);
    }
}
```

Multithreading

OOP

Multithreading:-

Execution of more than one thread at a time it is known as multithreading

thread:-

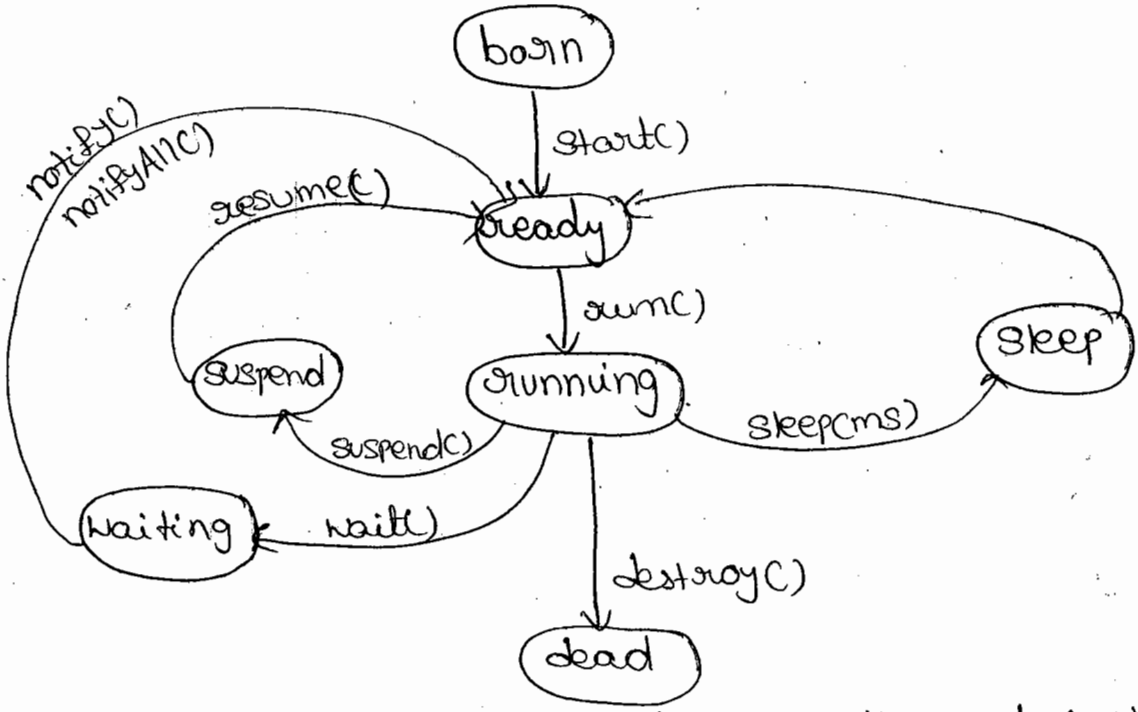
a Thread is a piece of code that executes independently.

-) every program contains atleast one Thread that is main Thread.

-) There are two ways to create new Thread

1) By extending java.lang.Thread class
2) By ~~extending~~ implementing java.lang.Runnable interface

Thread States:- (or) Life cycle of a Thread



-) new Thread will boorn whenever Thread class constructor is called

-) Thread comes to a ready state whenever start method is called

-) Thread comes to a running state whenever run() method is called.

-> Thread comes to a sleep state whenever sleep method is called.
-> Sleeping Thread wakes up automatically after time period
-> Thread comes to a suspend state whenever suspend() method is called.
-> suspended Thread comes to a running state whenever resume method is called
-> Thread comes to waiting state whenever wait method is called.
-> waiting Thread comes to a running state whenever notify() (or) notifyAll() method is called.
-> Thread will die whenever destroy() method is called.

11/4/15

java.lang.Thread

Fields (Variables)

```
public static final int MIN_PRIORITY; (1)
public static final int NORM_PRIORITY; (5)
public static final int MAX_PRIORITY; (10).
```

Constructors:-

```
public Thread();
public Thread(Runnable);
```

we can create Thread in 2 ways so 2 constructors
1) extends
2) Implementing

Methods:-

```
public static native Thread currentThread();   (Rt)
public static native void sleep(long) throws InterruptedException;   (milliseconds)
public synchronized void start();
```

If (int) (secs)

```
public void run();
public void destroy();
public final void suspend();
public final void resume();
public final void setpriority(int);
public final int getpriority();
public final void setname(String);
public final String getname();
```

→ every program contains atleast one Thread that is main Thread.
→ default name of the main Thread is main
→ default priority of the main Thread is 5

program to get current Thread or main Thread information:-

```
class demo
{
    psvm(String args[])
    {
        Thread t = Thread.currentThread();
        sopln(t.getname());
        sopln(t.getpriority());
        t.setname("mythread");
        t.setpriority(Thread.MAX_PRIORITY);
        sopln(t.getname());
        sopln(t.getpriority());
    }
}
```

// in this program by default main Thread is the thread

o/p:- main
5
mythread
10

Program to Demonstrate sleep method:-

```
class Demo
{
    psvm(String args[])
    {
        try{
            for(int i=1; i<=10; i++)
            {
                sopln(i);
                Thread.sleep(1000);
            }
            catch(Exception e)
            {
                System.err.println(e);
            }
        }
    }
```

→ sleep method throws Exception that why try & catch.

~~Program to create multithreaded~~
steps to create multithreaded application while extending java.lang.Threadclass

---

Step1:-
create a class that extends java.lang.Threadclass

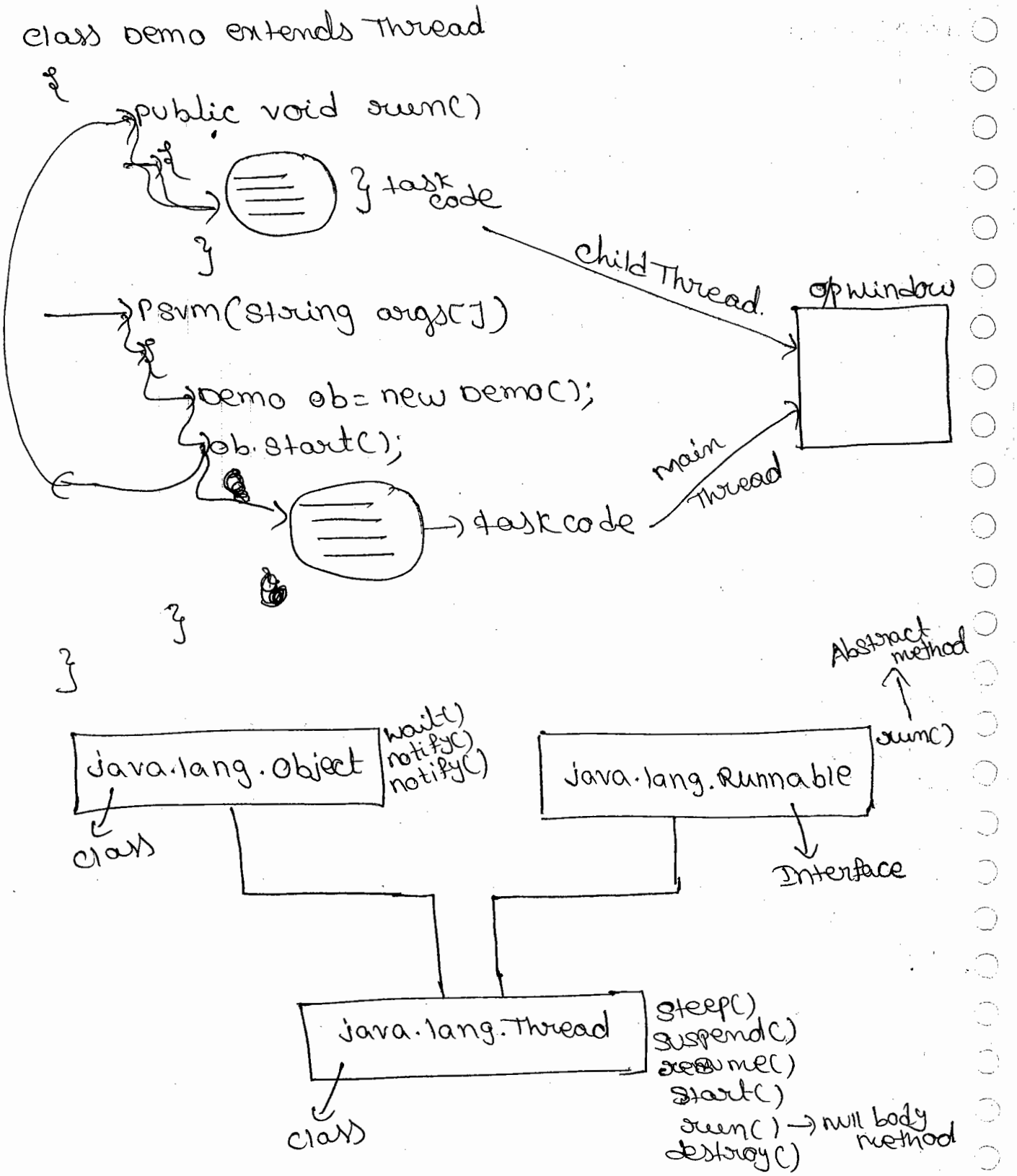
Step2:-
override run method

Step3:-
write main method

Step4:-
create an object of current class

Step5:-
call start method

NOTE:-
start() method calls run method.


class Demo extends Thread
{
public void run()
{
task code
}
psvm(String args[])
Demo ob = new Demo();
ob.start();
task code
child Thread
op window
main Thread
}
}
java.lang.Object
wait()
notify()
notify()
class
java.lang.Runnable
Abstract method
run()
Interface
java.lang.Thread
sleep()
suspend()
resume()
start()
run() → null body method
destroy()
class

```
class Demo extends Thread
{
    public void run()
    {
        try{
        for(int i=1; i<=10; i++)
        {
        sopln(" child Thread : " + i);
                Thread.sleep(1000);
         }
       }catch(Exception e)
        {
          System.err.println(e);
         }
  }
  psvm(String args[])
  {
      try{
       Demo ob = new Demo();
        ob.start();
        for(int i=1; i<=10; i++)
        {
        sopln(" main Thread: " + i);
                Thread.sleep(1000);
          }
          }catch(Exception e)
           {
            System.err.println(e);
           }
  } }
```

Program to demonstrate suspend() & resume() methods:-

```
class Demo extends Thread
{
    public void run()
    {
        try{
        for(int i=1; i<=10; i++)
        {
System.out.println("child Thread: "+i);
                Thread.sleep(1000);
        }
      }catch(Exception e)
      {
        system.err.println(e);
      }
}
  psvm(String args[])
  {
      try{
       Demo ob= new Demo();
        ob.start();
       for(int i= 1; i<=10; i++)
        {
       sopln("main Thread :"+i);
        Thread.sleep(1000);
        if(i==5)
          ob.suspend(); // child Thread suspended
        if(i==10)                    if we want
          ob.resume();
        }
       }catch(Exception e)
        {
          system.err.println(e);
  } } }
```

```
class Demo extends Thread
{
    public void run()
    {
        for(int i=1; i<=10; i++)
        {
        System.out.println("child Thread");
        }
    }
    psvm(String args[])
    {   Demo ob = new Demo();
        ob.start();
        sopln("main Thread");
    }
}
```

o/p:- can not be predicted

Program to demonstrate join() method:-

```
class Demo extends Thread
{   public void run()
    {   for(int i=1; i<=10; i++)
        {
        sopln("child Thread");
        }
    }
    psvm(String args[])
    {   try{
        Demo ob= new Demo();
        ob.start();
        ob.join(); // This statement puts the current Thread into waiting state until child thread joins.
        sopln("main Thread");
        }catch(Exception e)
        { System.err.println(e);
    } } }
```

run() method is in Thread class we are overriding that method in Demo by using Demo class

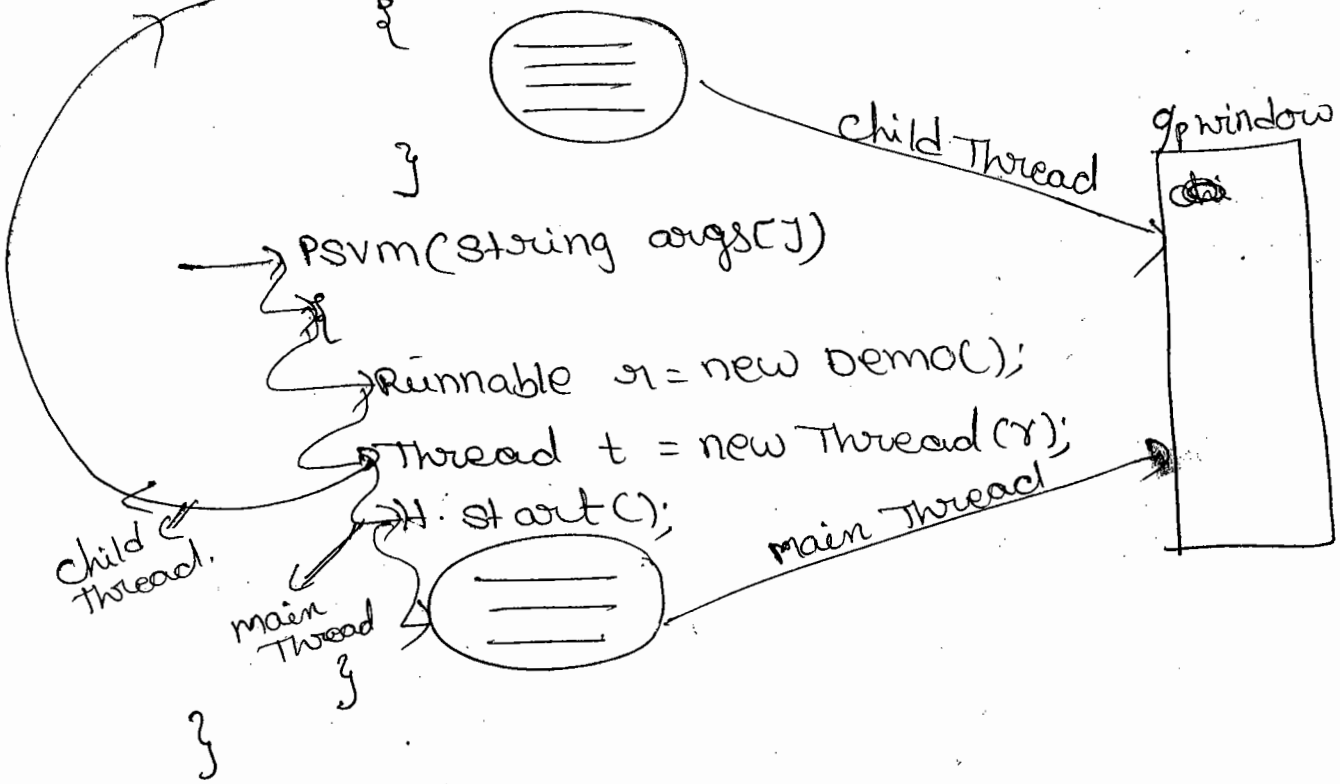
13/4/15

Steps to create multiThreaded application by implementing java.lang.Runnable Interface

1) create a class that implements Runnable interface
2) override run()
3) write main method
4) create an object of current class and assign to runnable reference
5) create an object of Thread class by passing Runnable reference in Thread class constructor.
6) call start method.

Example:-

```
class Demo implements Runnable
{
    Public void run()
    {
    }
    PSVM(String args[])
    {
        Runnable r = new Demo();
        Thread t = new Thread(r);
        t.start();
    }
}
```



```
class Demo implements Runnable
{
    public void run()
    {
        for(int i=1; i<=10000; i++)
        {
            Sopln("child Thread");
        }
    }
    Psvm(String args[])
    {
        Runnable r = new Demo();
        Thread t = new Thread(r);
        for(int i=1; i<=10000; i++)
        {
            Sopln("main Thread");
        }
    }
}
```

Synchronization:-

It is a mechanism which allows to access a shared resource only one Thread at a time

There are two ways to synchronize the code

1) Synchronizing a method
2) Synchronizing a block of code

1) Synchronizing a method:-

Syntax:-

```
synchronized ReturnType methodName(arg1, arg2, --)
{
    ==========
}
```

2) Synchronizing a block of code:-

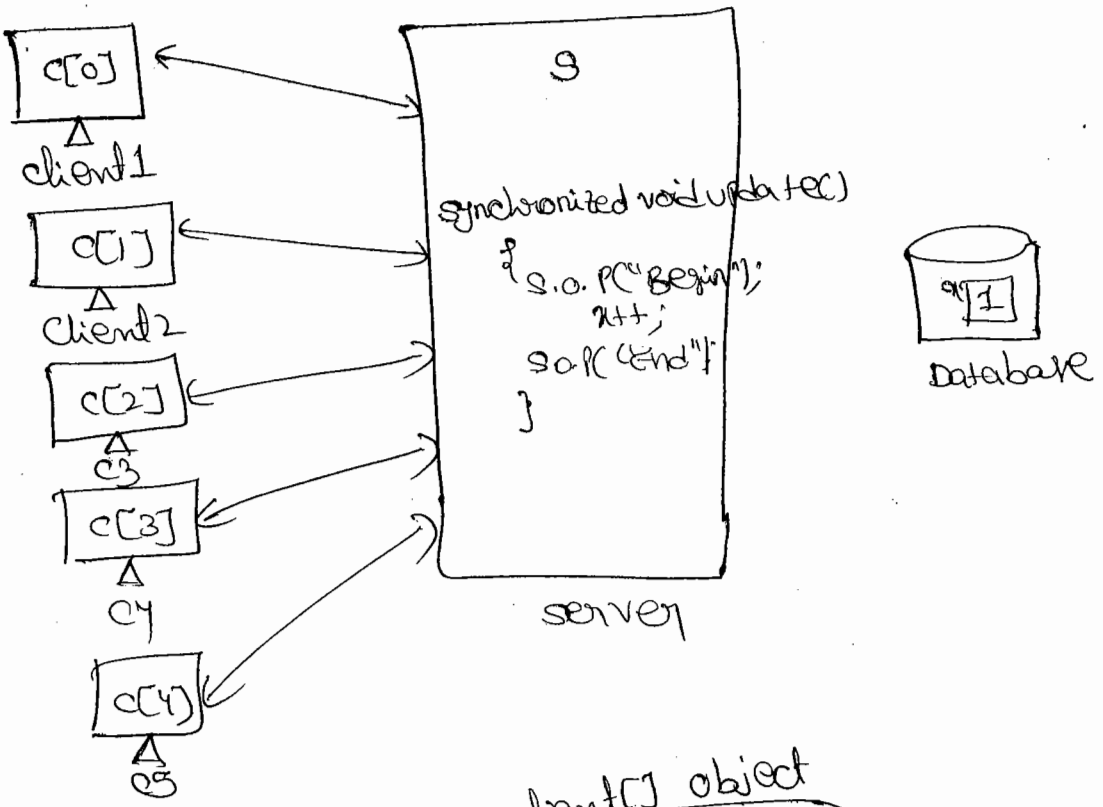
Syntax:-

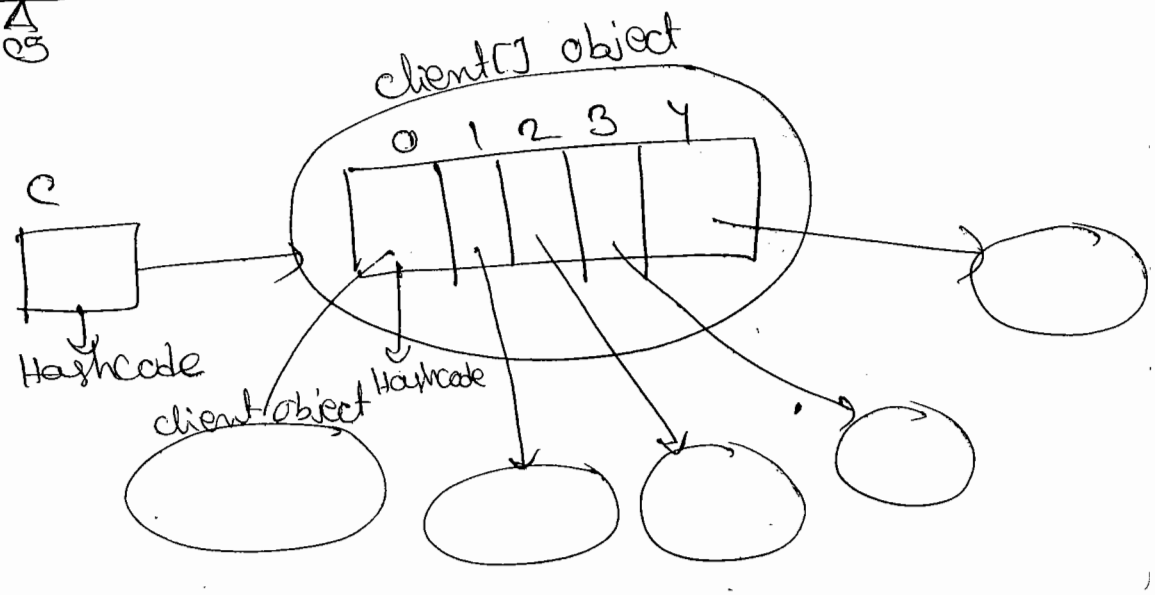
```
ReturnType MethodName(arg1, arg2,.....)
{
        ========
        ========
        synchronized(object)
        {
            ========
            ========
        }
        ========
        ========
}
```

program to demonstrate synchronization:-

```
class Server
{
    int x=1;
    synchronized void update()
    {
        System.out.println("Begin");
        x++;
        sopln("End");
    }
}
class Client extends Thread
{
    Server s;
    Client(Server s)
    {
        this.s = s;
    }
    public void run()
    {
        for(int i=1; i<=10; i++)
        {
            s.update();
        }
    }
}
```

```
class Demo
{
    psvm(String args[])
    {
        Server s=new Server();
        Client c[]=new Client[5];
        for(int i=0; i<c.length; i++)
        {
            c[i]=new Client(s);
            c[i].start();
        }
    }
}
```





java.lang.Object

Methods:-

```
public final native void notify();
public final native void notifyAll();
public final void wait() throws
                        InterruptedException;
```

program to demonstrate wait and notify() methods

```
class Demo extends Thread
{   int total;
    public void run()
    {   for(int i=1; i<=5; i++)
        {   total = total+i;
        }
        synchronized(this)
        {   notify();
        }
    }
public static void main(String args[])
{   try{
        Demo ob = new Demo();
        ob.start();
        synchronized(ob)
        {   ob.wait();
        }
        Sopln(ob.total);
    }catch(Exception e)
    {   System.err.println(e);
    }
}
```

NOTE:-
Both wait and notify methods must be used in synchronized area otherwise IllegalMonitorStateException will be thrown.

14/4/15

Inner classes

→ In java classes are divided into two categories

1) Top Level classes
2) Inner classes

1) Top Level class-

a class that is defined under package is known as Top Level class.

Example1:-

A.java → Top Level class

```
class A
{
  void show()
  {
    s.o.p("welcome");
  }
}
```

B.java → Top Level class

```
class B
{
  psvm(String args[])
  {
    A ob = new A();
    ob.show();
  }
}
```

Example2:-

A.java

```
package abc;                          // TOP Level class
public class A
{
  public void show()
    {
      S.o.P("welcome");
    }
}
```

B.java

```
class B                               // TOP level class
{
  psvm(String args[])
    {
      A ob = new A();                 // new A().show.
      ob.show();
    }
}
```

C.java

```
class C                               // TOP level class
{
  psvm(String args[])
    {
      abc.A ob = new abc.A();
      ob.show();
    }
}
```

(or)

```
A a = new A();
a.new B().show();
```

(or).

```
new A().new B().show();
```

2) Inner classes:-

a class that is defined in another class is known as Inner class

There are 4 types of Inner classes in java.

1) Member class
2) Nested TOP Level class
3) Local class
4) Anonymous class

1) Member class:-

a class that is defined as a member of another class is known as member class

Example:-

```
class A  -> TOP Level class(outer class)
{
    class B -> member class (Inner class)
    {
        void show()
        {
            soplln("welcome");
        }
    }
}
class Demo -> TOP Level class
{
    psvm(String args[])
    {
        A a = new A();
        A.B b = a.new B();   -> syntax
        b.show();
    }
}
```

2) Nested Top Level class!-

a class that is defined as a static member of another class is known as nested top level class

Example1-

```
class A
{
    static class B --> Nested Top Level class(Inner class)
    {
        void show()
        {
            sopln("welcome");
        }
    }
}
class Demo
{
    psvm(String args[])
    {
        A.B ob = new A.B();
        ob.show();
    }
}
```

Example2!-

```
class A
{
    static class B
    {
        static void show()
        {
            sopln("welcome");
        }
    }
}
class Demo
{
    psvm(String args[])
    {
```

```
        A.B.show();
    }
}
```

NOTE:-
member class and nested Top level class can have all access modifiers

3) Local class:-

a class that is defined inside method is known as Local class.

Example:-

```
class A
{ void print()
  {
    class B ---> Local class(Inner class)
    {
      void show()
      { soln("welcome");
      }
    }
    B ob = new B();
    ob.show();
  }
}
class Demo
{
  psvm(String args[])
  { A ob = new A();
    ob.print();
  }
}
```

4) Anonymous class!-

It is a one type of Inner class which has no name.

It is always subclass of a class (or) an Interface.

Example!-

```
interface A
{
    void show();
}
class Demo
{
    psvm(String args[])
    {
        A ob = new A()
        {
            public void show()
            { soln("welcome");
            }
        };
        ob.show();
    }
}
```

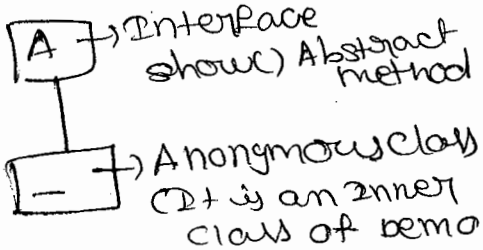
This program contains 3 classes Aclass, Demo class, Anonymous class (no name)
↳ Demo$1.class.

→ Anonymous class.

```
class — implements A
{
    public void show()
    { s.o.p("welcome");
    }
}
A ob = new — ();
ob.show();
```

c:\>javac Demo.java

A.class
Demo.class
Demo$1.class



Demo → class main()

java.awt.Color:-

Object Reference:-

```
public static final java.awt.Color WHITE;
public static final java.awt.Color LIGHT_GRAY;
                    java.awt.Color DARK_GRAY;
                    java.awt.Color BLACK;
                    java.awt.Color RED;
                    java.awt.Color PINK;
                    java.awt.Color ORANGE;
                    java.awt.Color YELLOW;
                    java.awt.Color GREEN;
                    java.awt.Color MAGENTA;
                    java.awt.Color CYAN;
                    java.awt.Color BLUE;
```

Constructor:-

```
public Color(int, int, int);
```

Red, Green, Blue → 0 to 255

Applet program:-

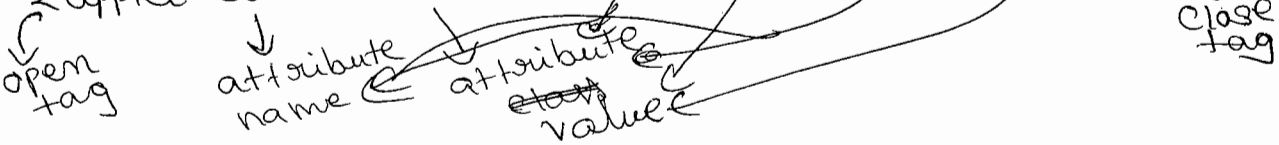
```
import java.awt.*;
import java.applet.*;
public class Demo extends Applet
{
    public void paint(Graphics g)
    {
        g.drawString("welcome",100,100);
    }
}
```

javac Demo.java → compilation.

in web browser c:\demo.html

→ Applet must be embeddddd in Html to run in browser

```
<applet code=Demo.class width=400 height=400> </applet>
```



java.awt.Component:-

Method:-

public void setBackground(Color);

```
import java.awt.*;
import java.applet.*;
public class Demo extends Applet
{
    public void init()
    {
        setBackground(Color.YELLOW);
    }
    public void paint(Graphics g)
    {
        g.setColor(Color.BLUE);
        g.fillOval(50,50,50,50);
    }
}
```

Example:-

```
import java.util.*;
import java.awt.*;
import java.applet.*;
public class Demo extends Applet
{
    public void init()
    {
        setBackground(Color.YELLOW);
    }
    public void paint(Graphics g)
    {  try{
        Random r = new Random();
        for(int i=1; i<=100; i++)
        {
```

Applets

java supports two types of programs

1) application
2) Applets

1) Application:-

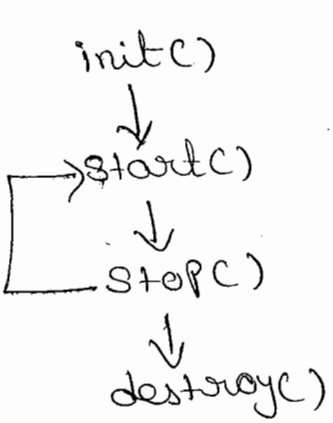
an application is a java program which runs under operating system.

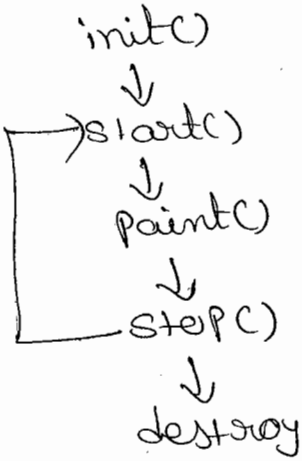
2) Applet:-

an Applet is a java program which runs under web browser.

Applets do not have main method because applets run under ~~now~~ browser.

Applet has a life cycle to under web browser.





open:-

init()
↓
start()
↓
paint()

deactivation:

stop()

Activation:

start()
↓
paint()

close

stop()
↓
destroy()

→ All life cycle methods are the part of ~~java.Applet~~ java.applet.Applet class.

→ every applet must extends java.applet.Applet class to derive life cycle.

java.applet.Applet:-

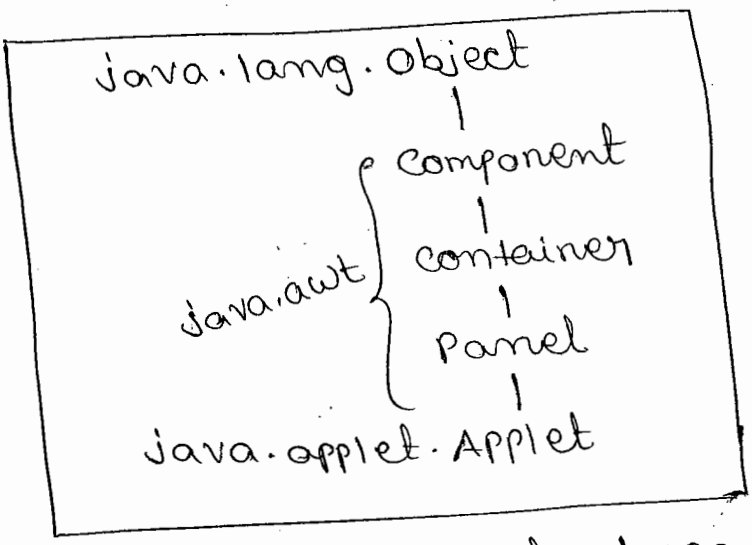
Methods:-

```
public void init();
public void start();
public void stop();
public void destroy();
```

java.awt.Component:-

Method:-

```
public void paint(Graphics);
```



→ every Applet must be public because it should be accessible to browser to create an object

java.awt.Graphics:-

Methods:-

```
public abstract void setColor(Color);
public abstract void drawOval(int, int, int, int);
public abstract void fillOval(int, int, int, int);
"  "  "  drawString(String, int, int);
```

x-coordinate, y-coordinate, width, Height

Class

text, x-coordinate, y-coordinate

```
g.setColor(new Color(r.nextInt(255), r.nextInt(255),
                                    r.nextInt(255)));
g.fillOval(r.nextInt(700), r.nextInt(400), 50, 50);
        Thread.sleep(50);
      }
        }catch(Exception e)
        {
          System.err.println(e);
        }
  }  }
```

→ Applets can also be executed by using ~~applet viewer~~ appletviewer at the command prompt

```
c:\>appletviewer demo.html
```

Swing

a swing is a set of classes and interfaces that are used to develop graphical user interfaces.

AWT VS Swing

AWT (Abstract window Toolkit)	Swing
1) AWT components are heavy weight components	1) Swing components are light weight components
2) AWT look and feel is different in different platforms.	2) Swing look and feel is same in all platforms
3) AWT look and feel can not be changed by programmer	3) Swing look and feel can be changed by programmer.
4) AWT components can not display ~~intro~~ images	4) Swing components can display images.
5) AWT components developed in C, C++ languages also	5) Swing components are developed in java language only

Swing classes and Interfaces are the part of ~~javax.~~ javax.swing.package.

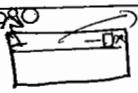
javax.swing package classes and interfaces are divided into 5 categories.

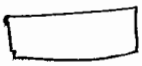
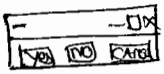
1) Containers
2) Components
3) Layout managers
4) Menu components
5) utility classes & Interfaces

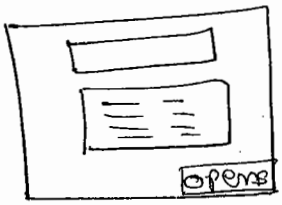
1) Containers:-

container is a swing class that contains component it can also be called as component because one container can be placed on another container

Container classes list

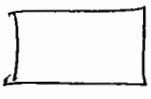
1) JFrame 

2) JPanel 
3) JDialog 

4) JFileChooser 

2) Components:-

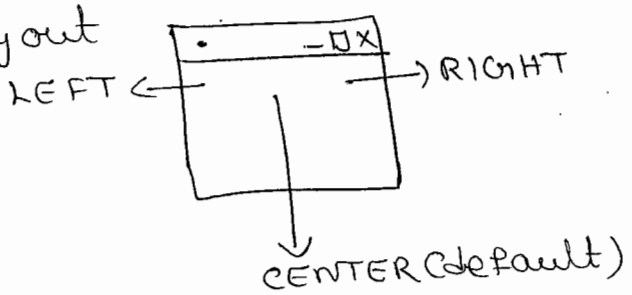
a component is ~~the swing class~~ a swing class that can be placed on container.

1) JButton [OK]
2) JLabel username
3) JTextField [abc]
4) JPasswordField [***]
5) JTextArea 
6) JCheckBox ☑ —
7) JRadioButton ◉ —
8) JToggleButton [on/off]
9) JComboBox [▾]
10) JList 

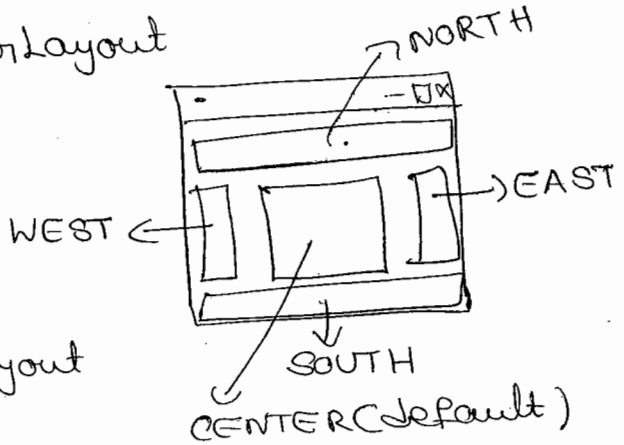
3) Layout Managers:-

These are AWT classes.
These are used to arrange components on container

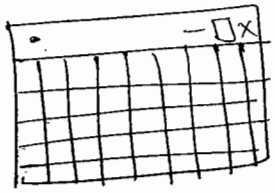
1) java.awt.FlowLayout



2) java.awt.BorderLayout



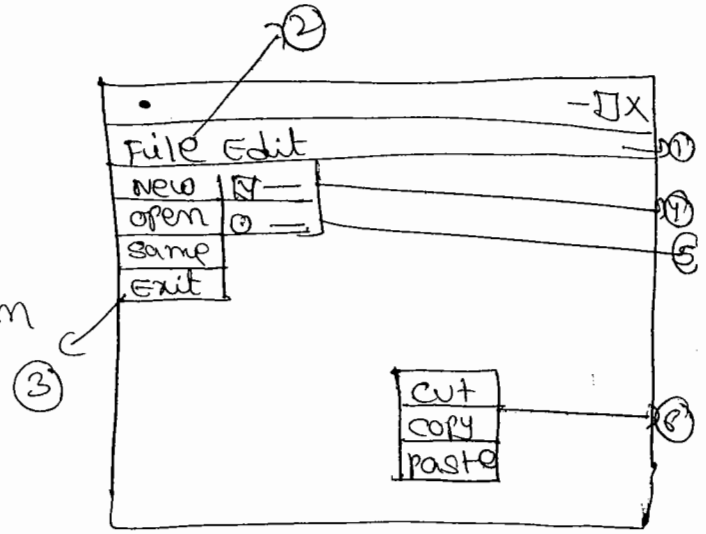
3) java.awt.GridLayout



4) Menu components:-

These are used to create menus

1) JMenuBar
2) JMenu
3) JMenuItem
4) JCheckBoxMenuItem
5) JRadioButtonMenuItem
6) JPopupMenu.



5) utility classes and interfaces:-
There are helper classes used to set the properties like colours, fonts, margins, width, height ..... etc

1) java.awt. container
2) java.awt. Component
3) javax. swing. JComponent
4) java.awt. Color
5) java.awt. Font
6) java.awt- Layoutmanager (interface)

16/4/15

steps to develop graphical user Interfaces:- (window applications)

step1:-
⊛ import the packages which are required

step2:-
create a class that extends container class and implements one or more ⊛ Event listeners Interfaces

step3:-
Declare the components which are required on container

step4:-
write constructor.

step5:-
set the layout to a container

step6:-
Instanciate components which are already declared.

step7:-
Register Event handlers with components

step8:-
add components to a container

step9:-
set the container size

Step10:-
set the container visible mode.

Step11:-
write main method

Step12:-
create an object of current class

Step13:-
override event handlers

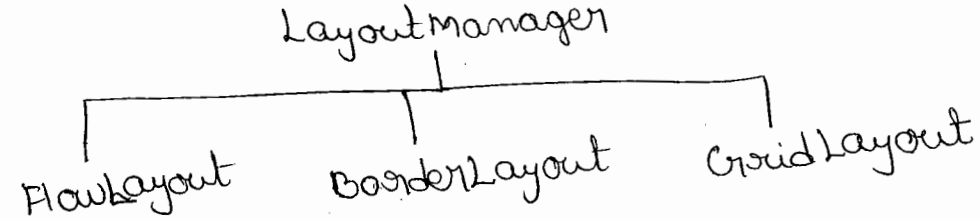
Step14:-
write task code

The following method used to set the layout

java.awt.Container

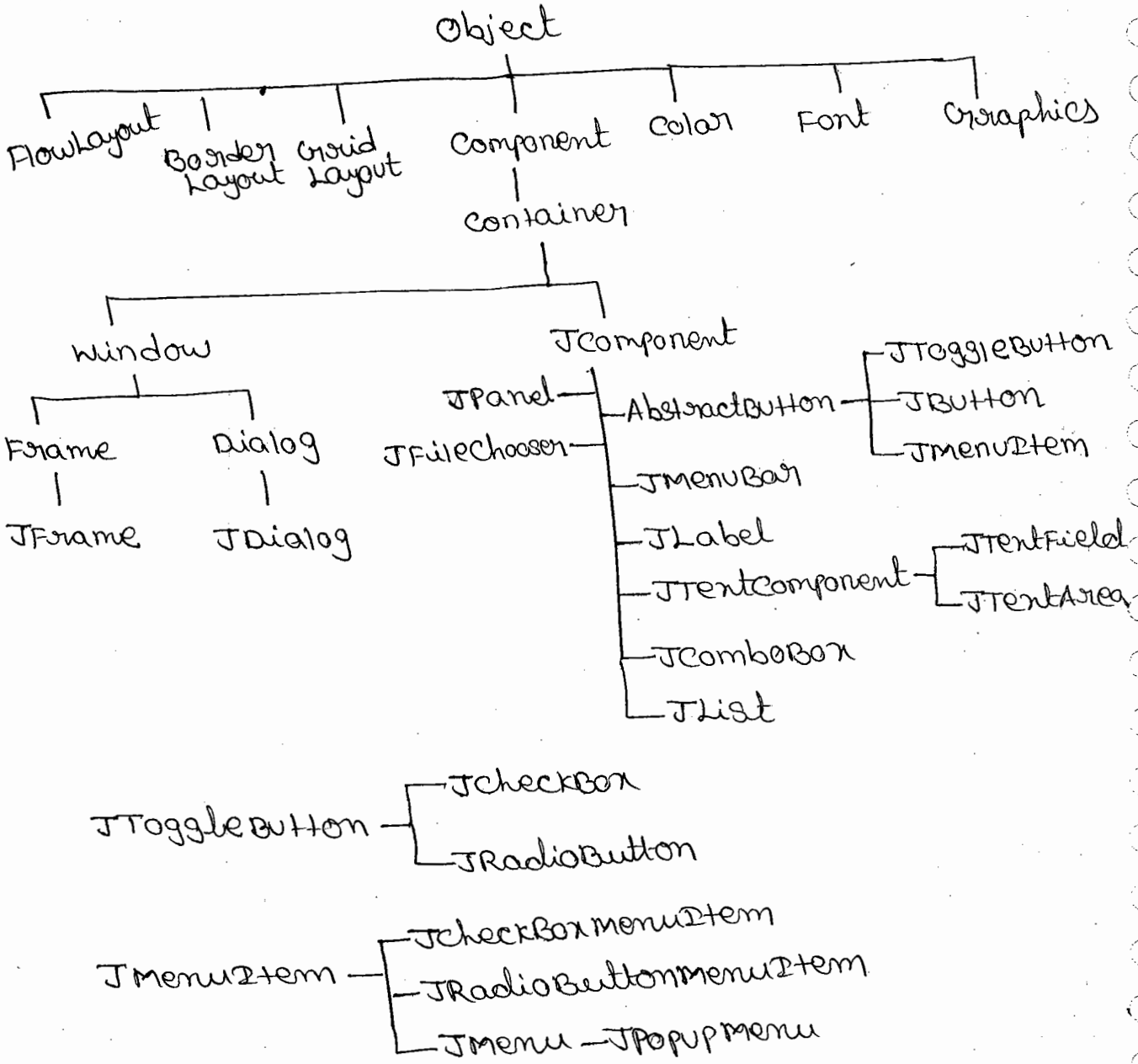
Method:-
public void setLayout(Layoutmanager);

Container	Default Layout
1) JFrame	BorderLayout
2) JPanel	FlowLayout
3) JDialog	BorderLayout



The Hierarchy of swing classes:-

The Hierarchy of Swing classes:-



java.awt.Window:-

Methods:-

public void setSize(int, int);
(width, height)
→ It is used to set the container size

public void setVisible(boolean);
→ It is used to set the visible mode.

```
import javax.swing.*;
class Demo extends JFrame
{
    Demo()
    {
        setSize(400, 400);
        setVisible(true);
    }
    public static void main(String args[])
    {
        new Demo();
    }
}
```

→ This class is under that ab package

→ These methods are in window class.

~~Instanciating components~~ :-

Instantiating components :-

1) `JButton jb1 = new JButton();` □

2) `JButton jb2 = new JButton("Exit");` Exit

3) `JLabel jl = new JLabel("username");` username

4) `JTextField jtf1 = new JTextField();` |

5) `JTextField jtf2 = new JTextField(20);` |

6) `JTextArea jta = new JTextArea(10, 20);`

..... etc

java.awt.Container

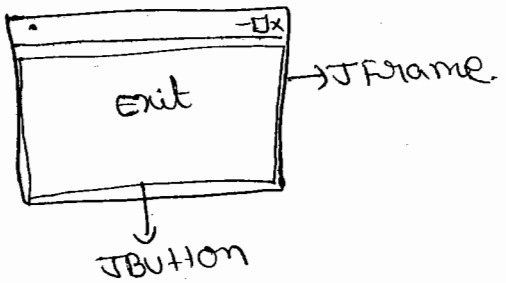
method :-

public Component add(Component);

→ It is used to add a component to a container.

```
import javax.swing.*;
class Demo extends JFrame
{
        JButton jb;
        Demo()
        {
            jb = new JButton("Exit");
            add(jb);
            setSize(400,400);
            setVisible(true);
        }
    public static void main(String args[])
    {
        new Demo();
    }
}
```

O/P:-

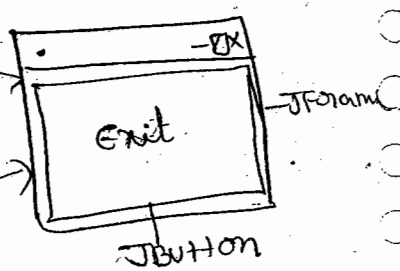


Setting Layouts to a Container:-

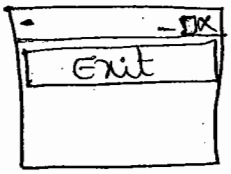
Examples:-

1) without setLayout() method

2) setLayout(new BorderLayout());

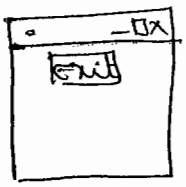


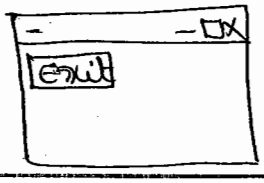
3) setLayout(new BorderLayout(BorderLayout.NORTH));



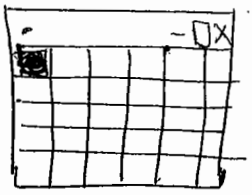
4) setLayout(new FlowLayout());

5) setLayout(new FlowLayout
(FlowLayout.LEFT));



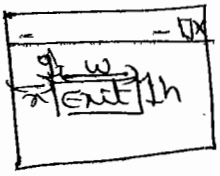


6) setLayout(new GridLayout(5,5));



7) setLayout(null);

jb.setBounds(50, 100, 80, 30);



--- etc.

Event Handling

Event Handling:-

It is a mechanism and it is implemented by using ~~event~~ Event Delegation model

Event Delegation model:-

It has 3 components

Example

1) Event source → javax.swing.JButton

2) Event object → java.awt.event.ActionEvent

3) Event Handler → public abstract void actionPerformed(ActionEvent);

This method belongs to java.awt.event.ActionListener interface.

→ In the above example whenever JButton is clicked & actionevent object created, passed to actionperformed method and actionperformed method is executed.

java.lang.System

method:-

public static void exit(int);

- 0 ⇒ Normal Termination
- 1 ⇒ Abnormal Termination

→ It is used to exit the application

javax.swing.AbstractButton:-

method:-

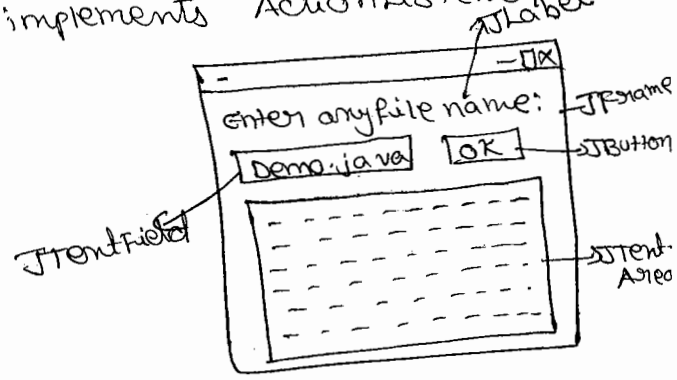
public void addActionListener(ActionListener);

→ It is used to register event handler with component

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
class Demo extends JFrame implements ActionListener
                                          ↓
                                     Event Listener
{
    JButton jb;
    Demo()
    {
        setLayout(new FlowLayout());
        jb = new JButton("Exit");
        jb.addActionListener(this);
        add(jb);
        setSize(400, 400);
        setVisible(true);
    }
    psvm(String args[])
    {
        new Demo();
    }
Public void actionperformed(ActionEvent ae)
    {
        System.exit(0);
    }
}
```

17/4/15

```
import java.io.*;
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
class Demo extends JFrame implements ActionListener
{
    JLabel jl;
    JTextField jtf;
    JButton jb;
    JTextArea jtA;
     Demo()
     {
         setLayout(new FlowLayout());
     jl = new JLabel("Enter any file name");
        jtf = new JTextField(20);
        jb = new JButton("OK");
        jta = new JTextArea(30,50);
        jb.addActionListener(this);
        add(jl);
        add(jtf);
        add(jb);
        add(jta);
        setSize(400,400);
        setVisible(true);
     }
public static void main(String args[])
     {
         new Demo();
     }
public void actionPerformed(ActionEvent ae)
     {  try{
FileInputStream fis = new FileInputStream(jtf.getText());
   byte b[] = new byte[fis.available()];
```



To add images

```
JButton jb=new JButton("OK",
    new ImageIcon("c:/but.GIF"));
```

```
fis.read(b);
String s = new String(b);
jta.setText(s);
}catch(Exception e)
{ system.err.println(e);
} }
}
```

more than one button

```
if(ae.getSource()==jb1)
{ =
}
else if(ae.getSource()==jb2)
{ =
}
else
{ =
}
```

new features :- (Last topic for core java)

Year	version	new features
1995	JDK 1.0	Intial Language
1997	JDK 1.1	1) Inner classes 2) new Event model
1998	JDK 1.2	no changes at Language Level
2000	JDK 1.3	no changes at Language Level.
2002	JDK 1.4	1) Assertions
2004	JDK 1.5	1) Enhanced for loop (or) for each loop 2) Auto boxing & unboxing 3) Covarient Return Type. 4) Annotations 5) Enumerations(enum) 6) Generics 7) Static imports 8) VarArgs.
2006	JDK 1.6	no changes at Language Level
2011	JDK 1.7	1) Binary Literals 2) underscores in numeric literals 3) Strings in switch statement 4) Handling multiple Exceptions with single catch block

5) try with resource statement

2014 JDK1.8 1) Lambda Expressions
2) new Date & time API.

New Features:-

1) Assertions:-

an Assertion is the condition that must be true during program execution
Assertions are used to Identify logical errors
In order to use assertions "assert" keyword and java.lang.AssertionError class Introduced in JDK1.4 version.
By default assertions are disabled.
To enable assertions use the following

```
c:\>java -enableassertion Demo
        (or)
c:\>java -ea Demo
```

```
import java.util.*;
class Demo
{
    psvm(String args[])
    {
        Scanner s = new Scanner(System.in);
System.out.print("Enter any number b/w 1 and 10: ");
        int x = s.nextInt();
        assert((x>=1)&&(x<=10));
          Sopln(x);
    }
}
```

Execution:-

```
c:\>javac Demo.java
c:\>java Demo
 Enter any number b/w 1 and 10: 82
 82
```

```
C:\ java -ea demo
Enter any number between 1 and 10: 82
Exception in thread "main" java.lang.AssertionError
        at demo.main(
```

2) Annotations:-

Annotations are meta tags that are used to pass some additional information to compiler and server about method, class and interface

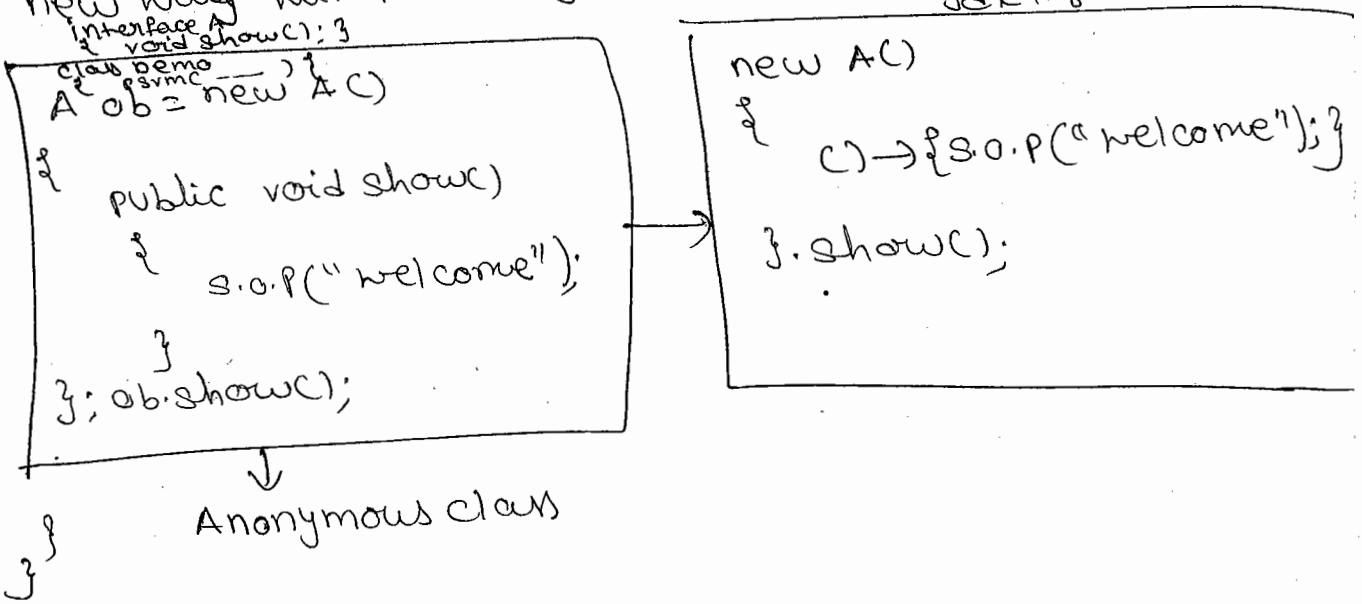
```
class A
{
    void show()
    {
        system.out.println("A class");
    }
}
class B extends A
{
    @override
    void shw()
    {
        system.out.println("B class");
    }
    psvm(String args[])
    {
        A ob = new B();
        ob.show();
    }
}
```

→ In the above example @override annotation informs the compiler "it is an overriding method"

→ All annotations are starts with '@' symbol

3) Lambda Expressions :-

This feature allows to write anonymous class in a new way with annonymous method.



4) new Date & time API :-

1) java.time.LocalDate
2) java.time.LocalTime
3) java.time.LocalDateTime

} classes.

```
import java.time.*;
class Demo
{
    psvm(String args[])
    {
    LocalDate ld = LocalDate.now();
    System.out.println(ld);
    LocalDate ld2 = ld.plusWeeks(10);
      sopln(ld2);
    LocalDate ld3 = ld.plusDays(100);
      sopln(ld3);
    }
}
```

Practice

1) notes
2) Java, 2 Complete Reference
3) Sun Java Tutorial
4) SCJP by Kathy Sierra.

1) venkatesh.mansani@yahoo.com
2) javadoubts@yahoo.com