

CORE JAVA

BY

Mr. VENKATESH SIR

**SATYA
TECHNOLOGIES**

SRI RAGHAVENDRA XEROX

Software Languages Material Available

Beside Bangalore Ayyangar Bakery, Opp. C DAC, Ameerpet, Hyderabad.

Cell: 9951596199

WT

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

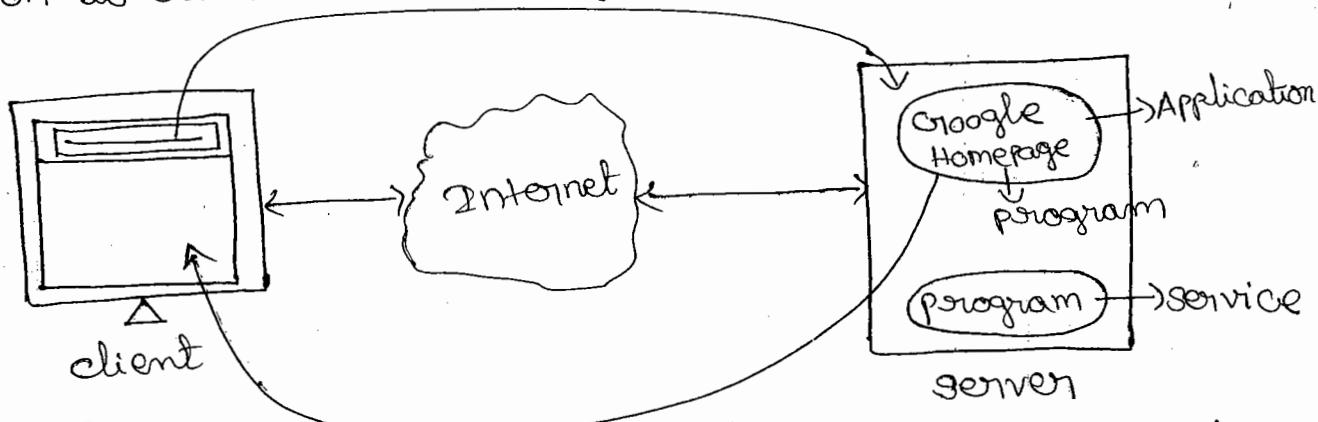
2) services

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 under operating system, browser and server



Platform:-

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

demo.java (unicode)

demo.c (unicode)

↓
Compiler

↓
demo.exe (bitcode)

↓
Windows

↓
Output

↓
Compiler

↓
demo.class (byte code)

↓
Windows JVM

↓
Windows

↓
O/P

↓
Linux JVM

↓
Linux

↓
O/P

↓
Salaries JVM

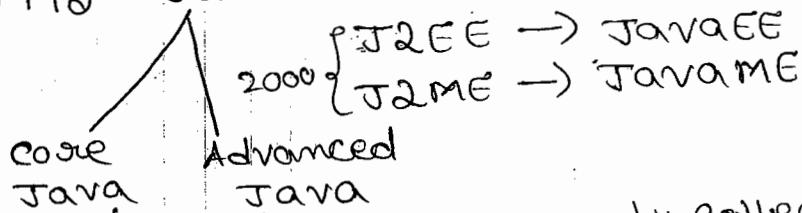
↓
Salaries

↓
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, solaris JVM is specific to solaris 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)
2008

1995 - Java - J2SE → JavaSE



These names are given by colleges, Institutes

- 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 begins 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 white space characters (Space bar, Tab, & Enter keys)

Examples! -

- | | | |
|--|-------------------|-------------------|
| 1) demo(<input checked="" type="checkbox"/>) | 6) -args(x) | 12) net*income(x) |
| 2) anddemo(x) | 7) -args(v) | 13) -\$- ✓ |
| 3) demo2 (<input checked="" type="checkbox"/>) | 8) \$args(v) | 14) -\$args ✓ |
| 4) OEMO(<input checked="" type="checkbox"/>) | 9) #args(x) | 15) -args5\$ ✓ |
| 5) Demo (<input checked="" type="checkbox"/>) | 10) net income(x) | |
| | 11) net-income(v) | |

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 2004
- "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
 - 3) int void=5; X Error
 beoz void is a keyword
- All keywords must be written in lower case letters otherwise compiler time error occurs.

Data Types:-

- Datatype that determines what value variable can hold and what 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

| <u>Data type</u> | <u>Memory size</u> | <u>Range</u> |
|--|--------------------|---|
| Integers byte short int long | 1 byte (8 bits) | $-2^7 \text{ to } 2^7 - 1$ |
| | 2 bytes (16 bits) | $-2^{15} \text{ to } 2^{15} - 1$ |
| | 4 bytes (32 bits) | $-2^{31} \text{ to } 2^{31} - 1$ |
| | 8 bytes (64 bits) | $-2^{63} \text{ to } 2^{63} - 1$ |
| Floating point numbers float double | 4 bytes (32 bits) | $-3.4 \times 10^{-38} \text{ to } 3.4 \times 10^{38}$ |
| | 8 bytes (64 bits) | $-1.7 \times 10^{-308} \text{ to } 1.7 \times 10^{308}$ |
| char etc | 2 bytes (16 bits) | 0 to 65535 |
| Boolean | 1 bit | true/false |

Reference data types:-

Arrays, strings, classes, interfaces --- etc

21/215

Literals:-

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

In java literals are divided into 6 categories.

1) Integer literals:-

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

2) Floating point literals:-

num num num
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; x → default value
85

Assignment :-

Syntax:- variable = literal;
Ex:- x = 5;

Initialization :-

Syntax:- datatype variable = literal;
Ex:- int x = 5; x [5]

| primitive datatype | default value |
|--------------------|--------------------------------|
| 1) byte | 0 |
| 2) short | 0 |
| 3) int | 0 |
| 4) long | 0 |
| 5) float | 0.0f |
| 6) double | 0.0 |
| 7) char | blank space (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; (✓)

long g = ; (X)

out of int range and
within long range

byte b = 130; (X)

long h = 40000; (✓)

short c = 130; (X)

long i = L or l (✓)

short d = 40000; (X)

out of int range with
in long range

int e = 40000; (X)

long j = ; (X)

int f = ; (X)

 out of int
 range

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

float a = 3.45; (X)

float b = 3.45f or l F; (✓)

double c = 3.45; (✓)

keyword int ^{Identifier}
x = 5; ^{Literal}
 ^{operator}
datatype variable

operators:-

an operator is a special symbol which operates on

data.

Types of operators:-

1) Unary operator

2) Binary operator

3) Ternary operator

Unary operators:-

an operator that operates on only one operand is known

as unary operator.

e.g:- att, tta, 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;  
          ↓  
          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 LABEL statement
- iii) continue statement
- iv) continue LABEL statement

v) return statement

In Java:-

```
int a=5;  
System.out.println(a);  
          ↓   ↓   ↓  
  class  object  method  
      Reference  
      (obj)  
Reference  
variable
```

In C :-

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

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

Java Notations:-

1) Package and sub package:-

all small letters sub packages 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 } classes
 iii) NullPointerException
 iv) Runnable } Interfaces
 v) AutoCloseable }

→ 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 letter
word 1st letter capital)

Examples:-

- i) name } variables
 - ii) empno } variables
 - 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:-

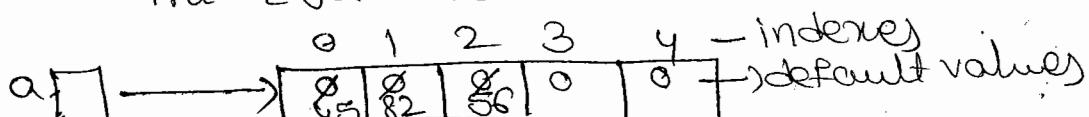
Ex:- int a[5] = new int[5];

(or)

int[] a = new int[5];

(or)

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



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

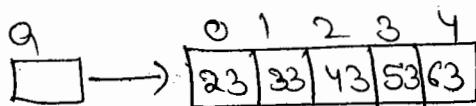
→ new is called as dynamic memory allocation operator and it allocates the memory.

Assignment Ex:-

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

Initialization Example:-

int a[5] = {23, 33, 43, 53, 63};



Accessing Array Elements by using For loop:-

for (int i=0; i<a.length; i++) {

 System.out.println(a[i]);

 }

out of loop.

C/C++ :-

char ch[10] = "welcome"

Java :-

char ch[10] = "welcome"; X

String s = "welcome" ✓

O/P:-

x 0 < 5 23
y 1 < 5 33
z 2 < 5 43
x 3 < 5 53
y 4 < 5 63
z 5 < 5 X

23/2/15

Object oriented programming :-

Object oriented programming language.

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();

{
 :: ::

} :: ::

 void display();

{
 :: ::

} :: ::

methods

101 " _____ " 5000.00
102 " _____ " 7000.00
103 " _____ " 5500.00

- class will not occupy memory
- file occupies memory

Object:-

An instance of a class

Syntax:-

classname objectReference = new constructor();

object
↑

Example:-

Emp e = new Emp();

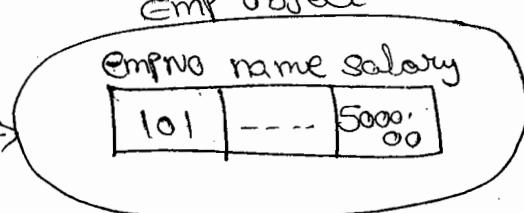
new:-

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

constructor:-

constructor constructs the object

Emp object



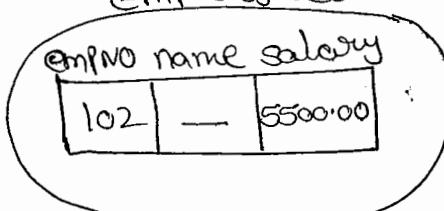
HashCode

without object reference:-

new Emp(); → Anonymous object

Emp object

unreferenced object



- every object occupies memory
- object contains data whereas 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):

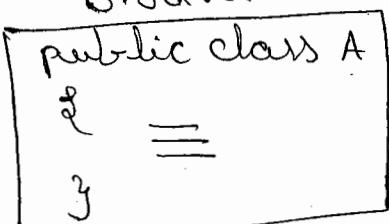
1) A source file can have only one public class.

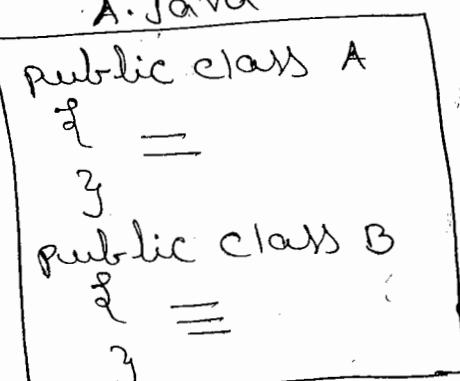
2) A source file can have any number of non-public classes.

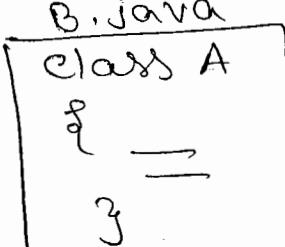
3) If the source file contains public class then the file name must match with public class name.

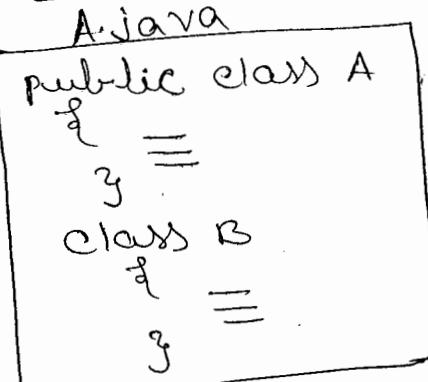
4) If the source file does not contain public class then no naming restrictions to a file name.

Examples:-

1) 

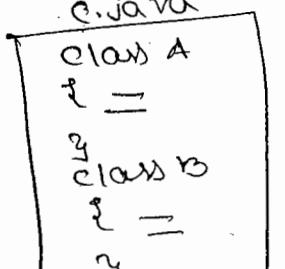
2) 

2) 

4) 

To compile :-

c:\>javac demo.java
↓
demo.class

5) 

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;

↓ ↓
It is a command append mode
it is used to set mode
the path

Location of java\w
↓
java\w 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 whereas system

variables are common to all users

variables are common to all users

→ use edit button to append the path

→ click on edit button

→ click on edit button

```

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:-
 multi
 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

26/2/15

variables :-

a variable is a container which contains data.

Types of variables :-

1) Instance variables

2) class variables

3) Local variables

1) Instance variables :-

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

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

Execution priority:-

1) class variables 3) main method i.e)

2) static blocks

3) public static void main(String args[])

new:-

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

→ Primitive type variable contains data whereas

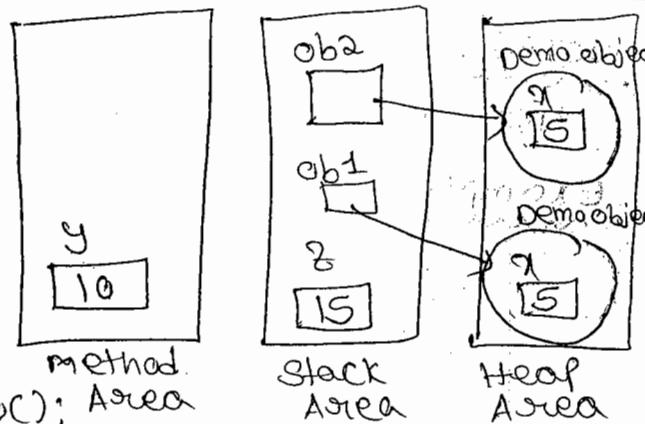
Reference type variable contains Hash code

→ 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;
    PS VM(String args[])
    {
        primitive type variable
        int z=15;
        Demo obj1=new Demo();
    }
}

```



Reference type variable,

Demo ob1 = new Demo(); → local variable
Demo ob2 = new Demo(); → 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

```

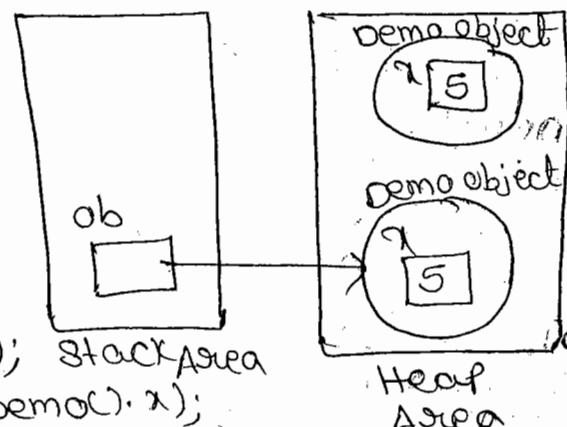
class Demo
{

```

```

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

```



26/2/19

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

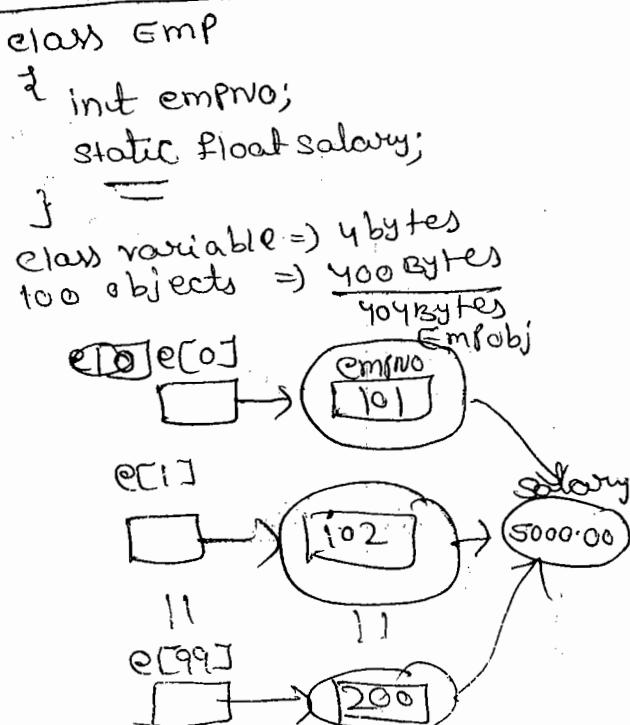
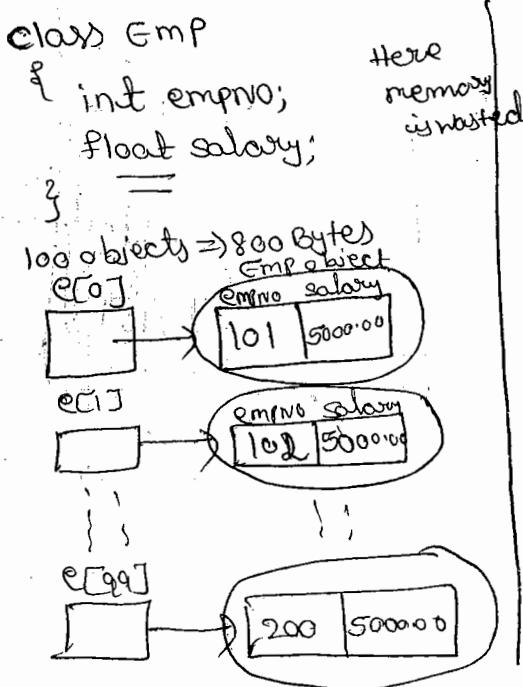
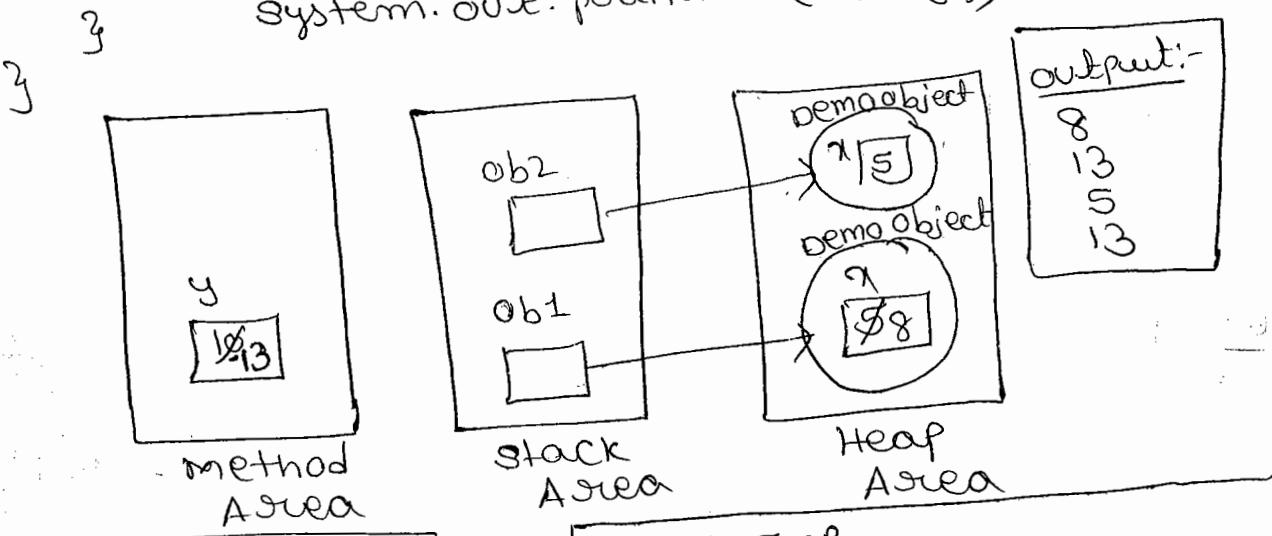
Ex:- class Demo

```
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;
        ob2.y = ob1.y+3;
        System.out.println(ob1.x);
        System.out.println(ob1.y);
        System.out.println(ob2.x);
        System.out.println(ob2.y);
    }
}

```



- 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 java.lang.System class

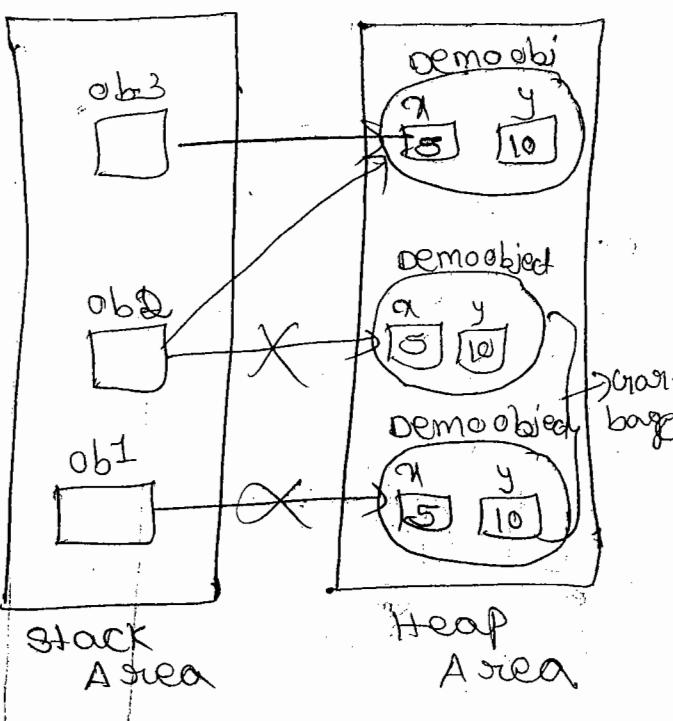
Example:-

```
class Demo
{
    int x=5;
    int y=10;
    public void main(String args[])
    {
        Demo obj1=new Demo();
        Demo obj2=new Demo();
        Demo obj3=new Demo();
    }
}
```

```
ob1=null;
```

```
ob2=ob3;
```

}



- Local variables must be initialized before access
- otherwise compile time error occurs bcoz they don't get default values.

Example:-

```

class Demo           Instance and class variable gets
{
    int x;           default value 0.

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

```

Example:-

```

class Demo           Instance variable
{
    int x=5;          Class variable
    static int y=10;   method parameter
    public static void main(String args[])
    {
        int z=15;      Local variable
        for(int i=1; i<=10; i++)
        {
            ==
        }
    }
}

```

- method parameters & block variables are also called local variables

Variable

- 1) Instance variable
- 2) Class variable
- 3) Method parameter
- 4) Local variable
- 5) Block variable

life

- object (L1S-4)
- class (L1S-5)
- method (L1S-3)
- method (L1S-2)

Block (less life
1 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

$\boxed{5} \neq 10$

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

→ methods with arguments & with return value

 1) methods with arguments & without return value

 2) methods with arguments & with 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 :-

- Primitive Data TYPE
- Reference Data TYPE

Return Type methodName(arg1, arg2, ...)

{ primitive types
| (or)
} Reference Types

○ Examples:-

②) int max(int a, int b) 2) int cube(int a)

三

```

3) int[] sort(int[] a) 4) String append(String s1,
   {                           String s2)
     {
       =}
     }
   }
 }
```

Example:-

class demo

{ int max(int a, int b)

$\{ \quad \text{if } c > b \}$

return a;

else
 return b;

2
Jewell

```
psvm(string argc)
```

```
{ demo ob = new Demo();
```

int a=ob. max(5c, 23);

System.out.println();

```
int a=56;  
int b=23;
```

Advantages of methods:-

- 1) Modularity
- 2) Reusability

Example:-

class Demo

{

 int cube(int a)

 method declaration

 { int b = a * a * a;

 method signature

 | return b;

 method body

}

 return statement

public static void main(String args[])

{

 Demo ob = new Demo();

 int x = ob.cube(5);

 method call
 statement

 System.out.println(x);

}

 mon2/2/15

2) methods with arguments and without return

Syntax:-

 Empty Data type

 ReturnType methodName(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) methods without arguments and with return value:-

Syntax:- primitive (or) referenced
 datatype datatype

Retwintype methodname()
{ == } task
 code
}

Examples:-

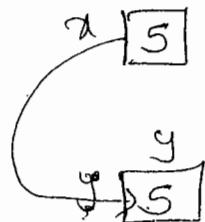
1) int getInt() 2) int[] getArray() 3) String getMessage()

{ == } { == } { == }
} } } }

Example:-

class Demo

{ int getInt()
 { int x=5;
 return x; }



3) public static void main(String args[])

{ Demo ob = new Demo();
 int y = ob.getInt();
 System.out.println(y);
}

}

→ Array itself an object in java

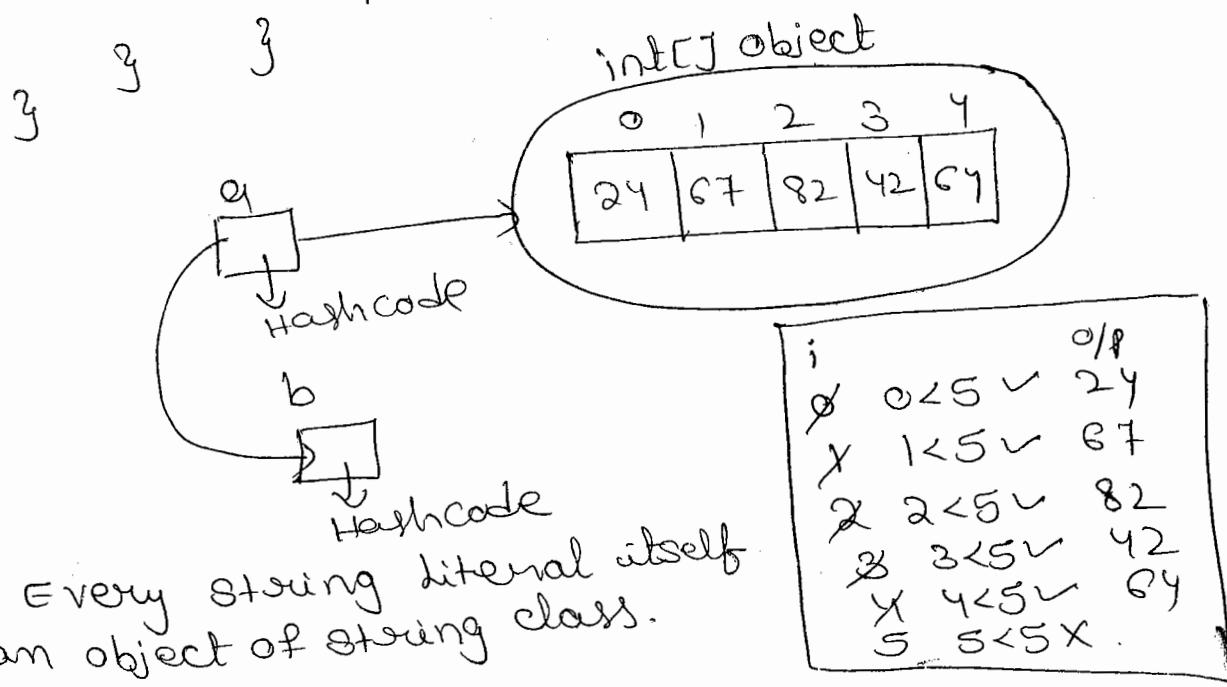
→ primitive type contains data and Reference Type
contain Hashcode

```

class Demo
{
    int[] getArray()
    {
        int a[] = {24, 67, 82, 42, 64};
        return a;
    }

    public static void main(String args[])
    {
        Demo ob = new Demo();
        int[] b = ob.getArray();
        for(int i=0; i<b.length; i++)
        {
            System.out.println(b[i]);
        }
    }
}

```



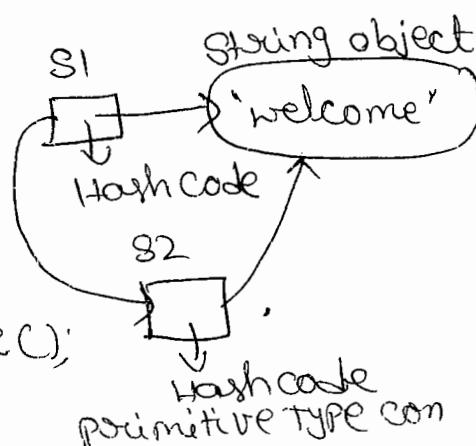
→ Every string literal itself
an object of String class.

```

class Demo
{
    String getMessage()
    {
        String s1 = "welcome";
        return s1;
    }

    public static void main(String args[])
    {
        Demo ob = new Demo();
        String s2 = ob.getMessage();
        System.out.println(s2);
    }
}

```



v) methods without Arguments and Without return value:-

Syntax:- → Empty data type void

Return type method name()
{
 ≡
 }

Example:-

void display() void add()
{
 ≡
 }

Example:-

```
class Demo
{
    void display()
    {
        System.out.println("Hello");
    }

    public static void main(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)
    {
        System.out.println(a+b+c);
    }

    public static void main(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 them 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 overridden in the same class because of ambiguity to call

```
class Demo
{
    void add(int a, int b)
    {
        System.out.println(a+b);
    }
}
```

```
void add(int x, int y) { }
{ System.out.println(x+y); }
}

public static void main(String args[])
{
    Demo ob = new Demo();
    ob.add(2, 3);
}
```

X Compiler gets confusion
ambiguity to call.

```
2 class Demo
```

```
3 { int a=5;
```

```
4 void showC)
```

```
5 { System.out.println(a); } => Error because ob is local  
reference variable and
```

it can not be accessed
outside the main
method

```
6 public static void main(String args[])
```

```
7 { Demo ob = new Demo(); }
```

```
8 ob.show();
```

```
9 }
```

this keyword :-

It is called as an object reference or reference
variable because it refers to an object
and always refers current object

Example1:-

```
class Demo
```

```
{ int a=5;
```

```
void showC)
```

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

```
}
```

```
public static void main(String args[])
```

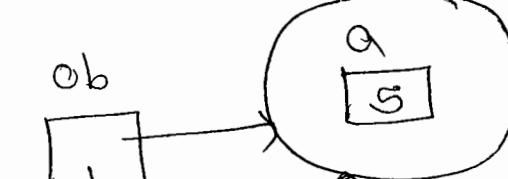
```
{ Demo ob = new Demo(); }
```

```
ob.show();
```

```
}
```

implicitly calls this keyword

↓
this.a

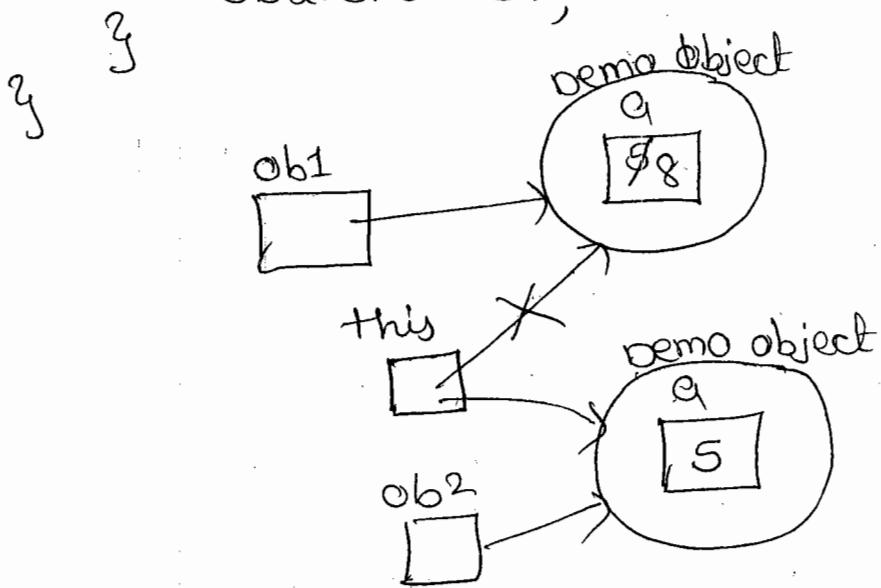


Hash(this
code)

HashCode

Example2:-

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



→ `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 showC()
    {
        int a=10;           local variable → local variable we can
        System.out.println(a); → print in one way i.e
        System.out.println(this.a); = 5   directly
    }
}
psvm(String args[])
{
    Demo ob = new Demo();
    ob.showC();
}

```

O/P:- 10
5

Example4:-

```

class Demo
{
    void showC()
    {
        System.out.println("Core Java");
        point(); ~ implicitly [this.point]
    }
}

void point()
{
    System.out.println("Advanced Java");
}

psvm(String args[])
{
    Demo ob = new Demo();
    ob.showC();
}

```

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
- 4) By using class name.

not recommended
to use.

→ 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()
    {
        System.out.println("welcome");
    }
    public static void main(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 contain instance variable

4/3/15

- Static method does not refer this keyword in anywhere
(Implicitly & explicitly both not possible)

Example:-

class Demo

{ int x=5;

static void show()

{ System.out.println(x); } *⇒ Error because static method
doesn't refer this keyword.*

}

public static void main(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 Initialization blocks, class methods and nested top level classes. (One type of Inner class)

Constructors:-

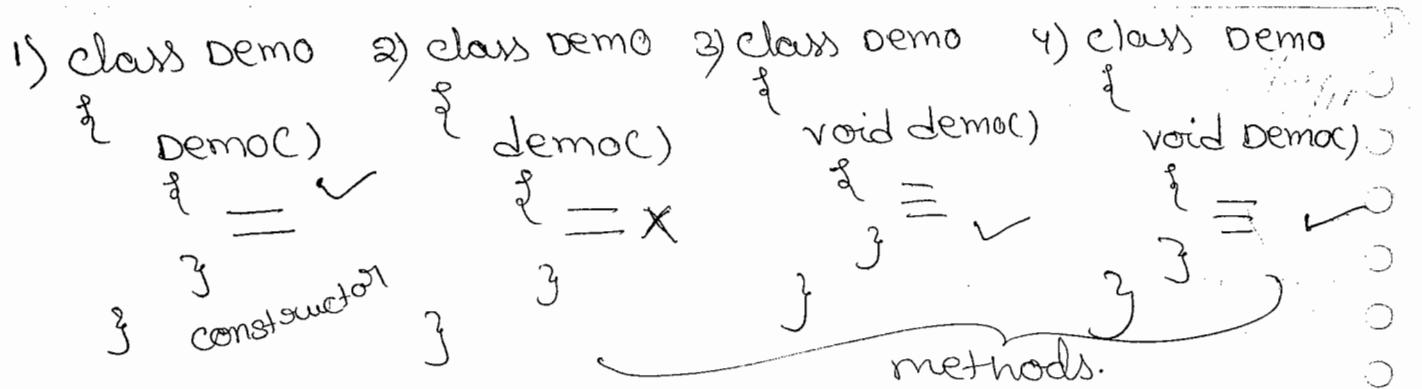
- A constructor is a special method which has same name as the class name and which has no return type.

→ constructor is called automatically whenever an object is created

→ constructors are used to initialize instance variables

→ Constructors are two types

- 1) Default Constructors (without arguments)
- 2) Parameterized Constructors (with arguments)



Example 1:-

```

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

Example 2:-

```

class Demo
{
    demo()
    {
        System.out.println("core Java");
    }
    demo(String s)
    {
        System.out.println(s);
    }
    public static void main(String args[])
    {
        new Demo();
        new Demo("Advanced Java");
    }
}
  
```

constructor overloading

demo obj → not called becoz 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 either a constructor.

Example1:-

```
class demo
{
    demo()
    {
        System.out.println("Core Java");
    }
    demo(String s)
    {
        this();
        System.out.println(s);
    }
    public static void main(String args[])
    {
        new demo("Advanced Java");
    }
}
```

O/p:- Core Java
Advanced Java

Example2:-

```
class demo
{
    demo()
    {
        this("Advanced Java");
        System.out.println("Core Java");
    }
    demo(String s)
    {
        System.out.println(s);
    }
}
```

psvm(String args[])

{
new demo();

O/p:- Advanced Java
Core Java

Instance variable Initialization without constructor

Example:

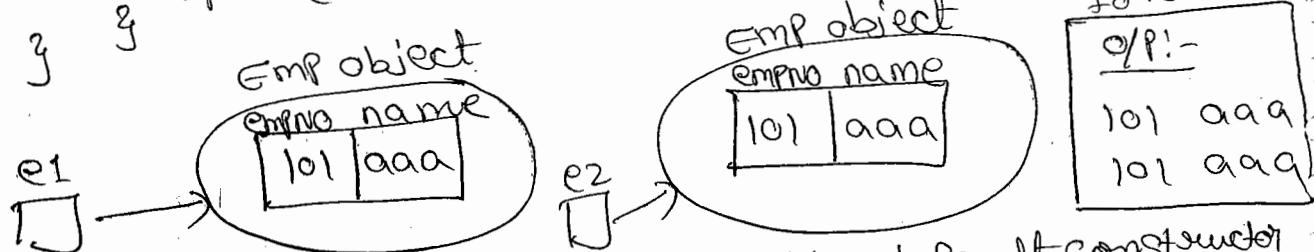
This is not the right way to Initialize Instance variable if more than one object is present means it not go to

class Emp

```
{ int empno=101; } Initialization this enti-  
string name="aaa"; } zation.  
               requirement of
```

play demo

psvm(string args[]);
Emp e1 = new Emp();
Emp e2 = new Emp();
but here we are getting same obj for all objects
we are different data so this is not the right



Instance variable Initialization with default constructor

class Emp

```
{ int empno;  
    string name;  
    EmpC)
```

Implicitly →  empNO = 101;
 name = "aaa";

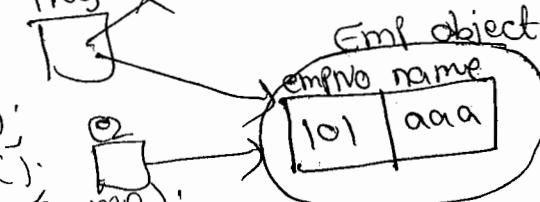
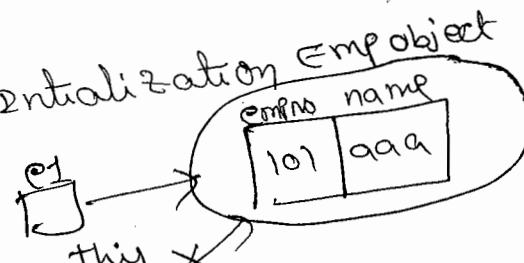
class Demo

{ symmetric arg(J)}

```

{ Emp e1 = new Emp();
  Emp e2 = new Emp();
  System.out.println(e1.empno + " " + e1.name);
  System.out.println(e2.empno + " " + e2.name);
}

```



91. - 201 a.a.a
101 a.a.a.

- 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 Initialization with Parameterized constructor :

```
Class Emp
```

```
{ int empno;
```

```
String name;
```

```
Emp(int empno, String name)
```

```
{ this.empno = empno; }
```

this.name = name; } Initialization
This is not assignment this is
Initialization

```
    }
```

```
class Demo
```

```
{ public static void main(String args[])
```

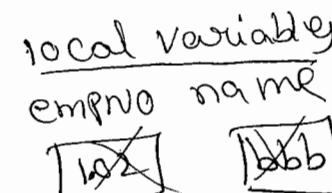
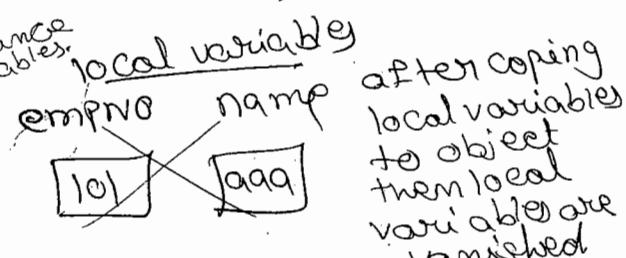
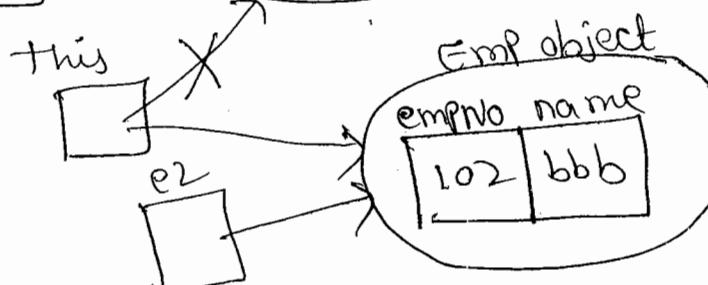
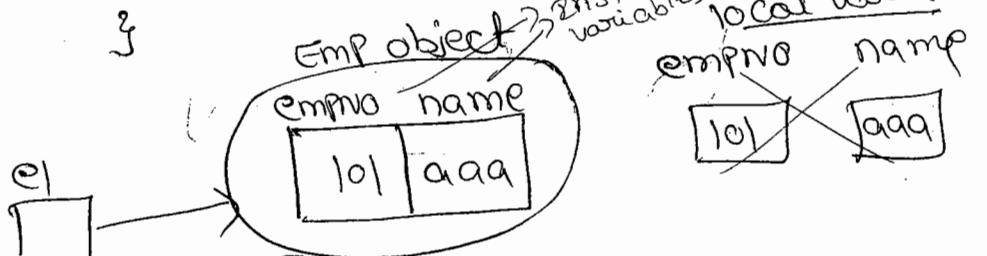
```
{ Emp e1 = new Emp(101, "aaa");
```

```
Emp e2 = new Emp(102, "bbb");
```

```
System.out.println(e1.empno + " " + e1.name);
```

```
System.out.println(e2.empno + " " + e2.name);
```

```
}
```



Output:- 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; } it is called as an assignment.
        this.name = name; }
```

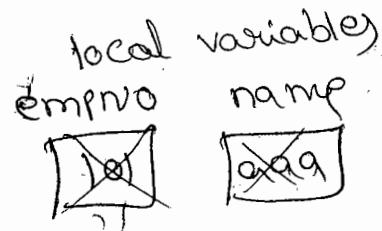
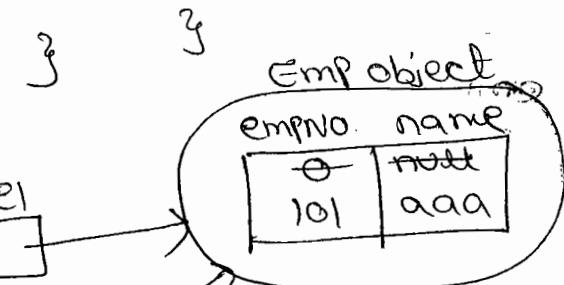
```
}
```

```
class Demo
```

```
{ public static void main(String args[])
{
    Emp e1 = new Emp();
    e1.set(101, "aaa");
    Emp e2 = new Emp();
    e2.set(102, "bbb");
```

```
    System.out.println(e1.empno + " " + e1.name);
```

```
    System.out.println(e2.empno + " " + e2.name);
```



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

6/3/15

for

NOTE: System implicitly provides one default constructor if the class does not contain any other 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           ✗ invalid (system does not provide
{
    psvm(String args[])
    {
        new Demo("Hello");
    }
}
```

Example3:-

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

Example:-

```

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 initialization 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");
    }

    Block- { sopln("Block1"); }

    psvm(String args[])
    {
        sopln("main method");
        new Demo();
    }

    Block- { sopln("Block2"); }
}

```

O/P:-

main method
Block1
Block2.
Constructor.

Constructors, blocks
are same that
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 initialization
- because it can be used to initialize 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.

Class Demo

```
static
{
    sopm("static Block1");
}
psvm(String args[])
{
    sopm("main method");
}
static
{
    sopm("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

- i) a group of statements into a single logical unit is called as method.
 - ii) method can have same name as the class name or different name
 - iii) method must contain return type.
 - iv) methods are used to perform the task
 - v) it must be explicitly called

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

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

7/3/15

Abstraction:

Abstraction:- providing necessary information without including details (all)

providing many background details (1)

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

class Student

```
int rollno;
String name;
String address;
int landline;
int landlineno;
```

```
void accept()
```

```
{ = }
```

```
void display()
```

```
{
```

```
SOP(rollno);
SOP(name);
SOP(address);
```

```
}
```

```
SOP(landlineno);
```

student2.java

class student2 extends student

→ subclass

```
int mobileno;
```

```
String mailId;
```

```
void accept()
```

```
{ = }
```

```
)
```

```
void display()
```

```
{
```

```
SOP(mobileno);
```

```
SOP(name);
```

```
SOP(mobileno);
```

```
SOP(mailId);
```

```
}
```

```
}
```

- In the above example new features are mobileNo & mail2d & reusable features are rolling 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

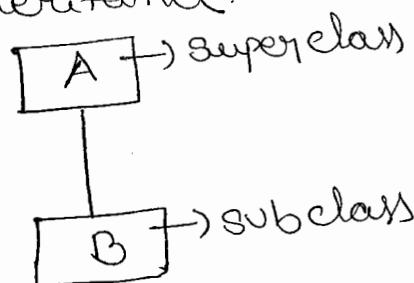
4) Hierarchical Inheritance

*5) Multipath Inheritance

*6) Hybrid Inheritance

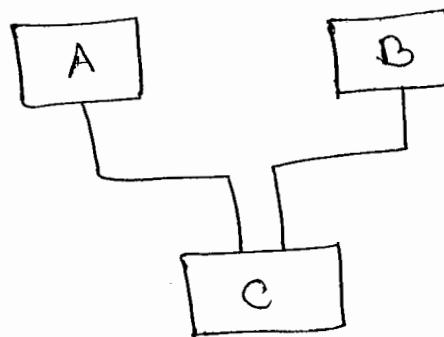
1) Single Inheritance:-

Derivation of a class from only one superclass is called single inheritance.



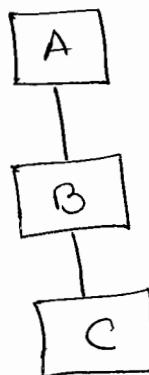
2) Multiple Inheritance:-

Derivation of a class from more than one superclass is called as multiple inheritance



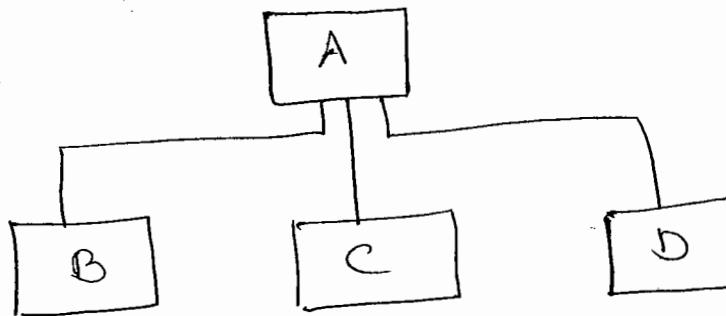
3) Multilevel Inheritance:-

Derivation of a class from sub class is called as multilevel inheritance.



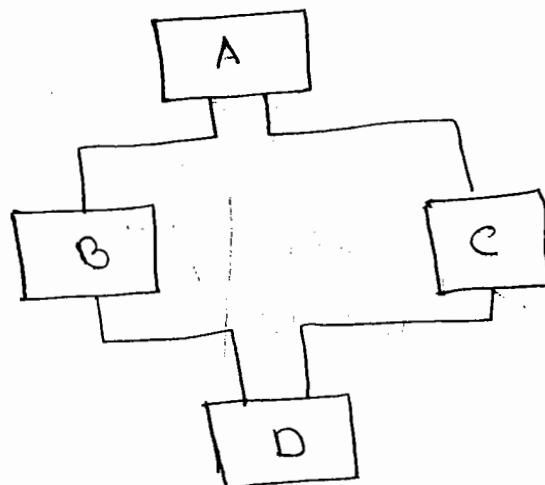
4) Hierarchical Inheritance:

Derivation of several classes from only one super class is called as Hierarchical Inheritance.



5) Multipath Inheritance:-

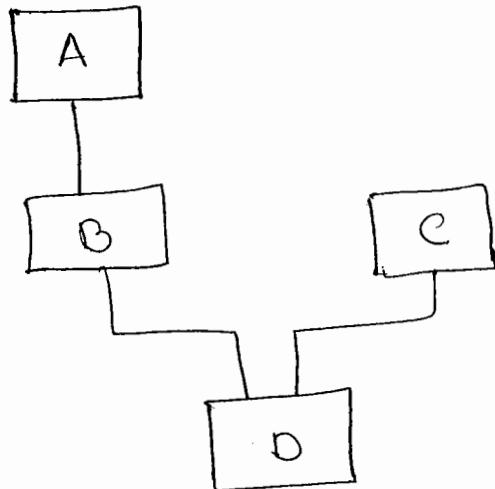
Derivation of a class from more than one super class, those subclasses get inherited from the same super class is called as multipath inheritance.



A to D two paths
are there via
B & C that's why
it is multipath

6) Hybrid Inheritance:-

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



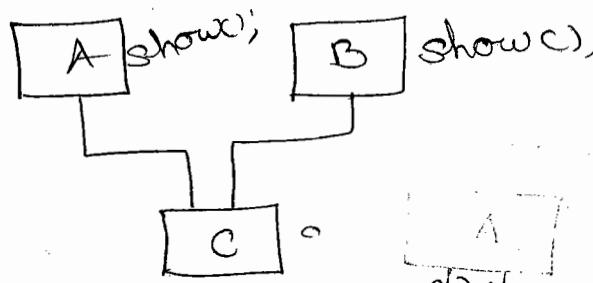
NOTE1:-

Multipath Inheritance is also one example of Hybrid Inheritance.

NOTE2:-

Java doesn't support multiple inheritance, multipath inheritance and Hybrid Inheritance because Java doesn't support multiple inheritance because of ambiguity.

Example:-



c ob = new C();

ob.showC();

→ This statement gets ambiguity to call either A class showC() or B class showC().

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

Example:-

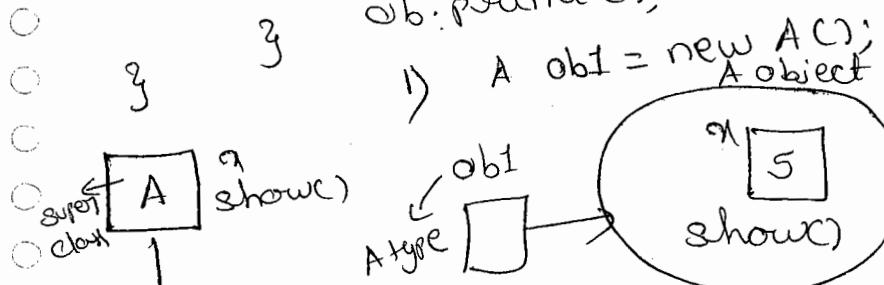
```

class A
{
    int x=5;
    void showC()
    {
        System.out.println("A class");
    }
}

class B extends A
{
    int y=10;
    void printC()
    {
        System.out.println("B class");
    }
}

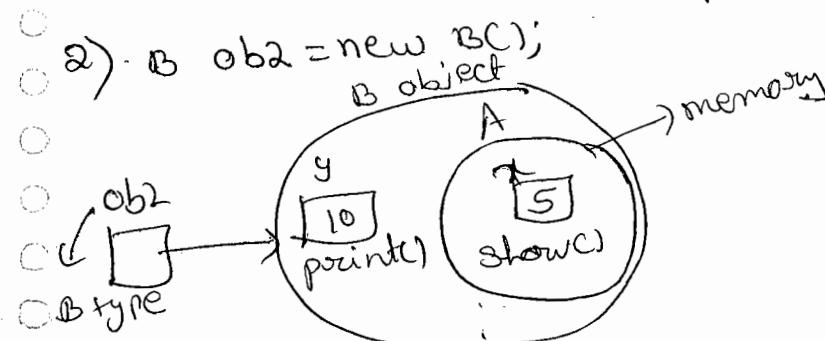
public class Main
{
    public static void main(String args[])
    {
        B ob = new B();
        System.out.println(ob.x);
        System.out.println(ob.y);
        ob.showC();
        ob.printC();
    }
}
  
```

Op:-
5
10
A class
B class



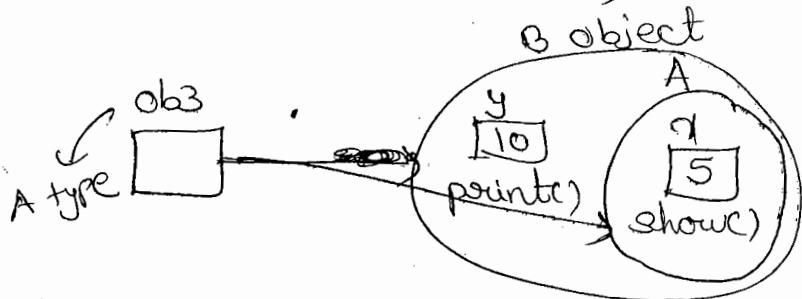
Here y & print() does
not have memory
show can we call

s.o.p(ob1.x); ✓
ob1.showC(); ✓
s.o.p(ob1.y); X
ob1.print(); X



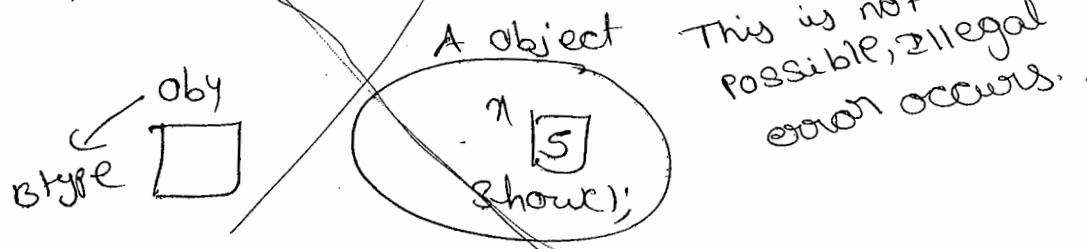
s.o.p(ob2.x); ✓
ob2.showC(); ✓
s.o.p(ob2.y); X
ob2.print(); X

3) $A \ ob3 = \text{new } B();$ ✓



$s.o.p(ob3.x);$ ✓
 $ob3.show();$ ✓
 $s.o.p(ob3.y);$ ✗
 $ob3.pointc();$ ✗

4) $B \ ob4 = \text{new } A();$



→ 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

→ Super class object reference refers both super class object and subclass object but it can access only super class members

$A \ ob = \underline{\hspace{2cm}};$ $\text{new } AC();$
 or
 $\text{new } BC();$

→ ~~sub class object refers only sub class~~

→ sub class object reference refers only sub class object but it can access both super class members and sub class members

$B \ ob = \underline{\hspace{2cm}};$ $\text{new } BC();$

Example:-

Class A

{ int x=5;

}

Class B extends A

{ int y=10;

void point()

{ int z=15;

System.out.println(x);

System.out.println(y);

System.out.println(z);

}

public static void main(String args[])

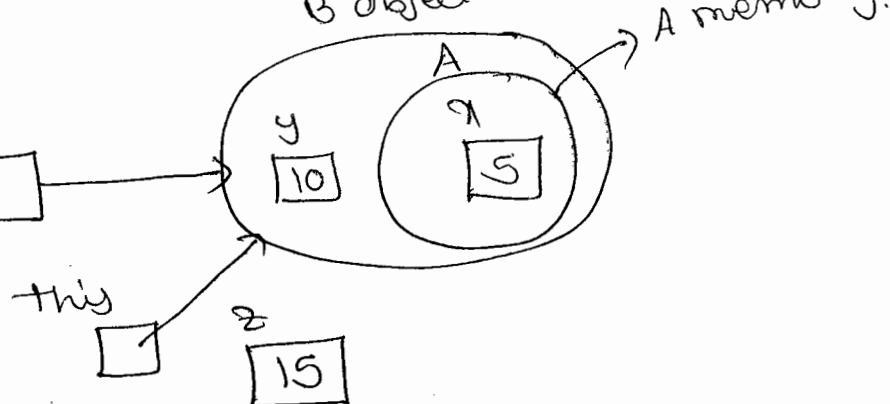
{ B ob = new B();

ob.point();

}

B type

ob



10/3/15 true

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.

Example!:-

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();
```

```
}
```

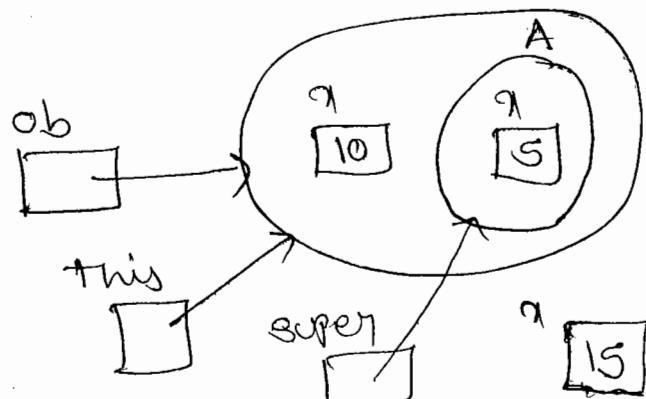
O/P!:-

15

10

5

B object



Example:-

```
class A
{
    void point()
    {
        System.out.println("A class");
    }
}

class B extends A
{
    void point()
    {
        System.out.println("B class");
    }

    void showC()
    {
        point();           // implicitly
        ↓
        this.point();
    }

    public static void main(String args[])
    {
        B ob = new B();
        ob.showC();
    }
}
```

o/p:-
B class
B class
A class

NOTE:- ***

Static method does not refer this and super keywords
anywhere (implicitly & explicitly)

super(); :-

This statement calls default constructor of super class

super(...); :-

This statement 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()

↓ implicitly

→ super();

{ System.out.println("B class");

}

public static void main(String args[])

{ new B();

}

OP:- A class
B class

Example:-

class A

{ A(String s)

{ sopln(s);

}

class B extends A ↓ implicitly

{ B(String s)

→ super();

{ 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)
    {
        super();
        System.out.println(s);
    }

    public static void main(String args[])
    {
        new B("core java");
    }
}
```

implicitly

o/p:- java
corejava

2) Removing parameterized constructor from super class

o/p:- corejava

```
class A
{
}

class B extends A
{
    B(String s)
    {
        super();
        System.out.println(s);
    }

    public static void main(String args[])
    {
        new B("core java");
    }
}
```

implicitly

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

Example:-

```
class A
{
    A(String s)
    {
        System.out.println(s);
    }
}
```

class B extends A

```

{
    B(String s)
    {
        super(s);
        System.out.println(s);
    }
}

public class B
{
    public static void main(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 initialized 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 initialized before access otherwise compile time error occurs

Example:-

```
class Demo
{
    final int x;
    void show()
    {
        x=0; // It is called as assignment
    }
    public static void main(String args[])
    {
        Demo ob = new Demo();
        ob.show();
        System.out.println(ob.x);
    }
}
```

can not assign a value to a final variable

Error because final variable must be initialized.

Example:-

```
class Demo
{
    final int x;
    Demo()
    {
        x=0;  $\Rightarrow$  It is called Initialization
    }
}
psvm(String args[])
{
    Demo ob = new Demo();
    System.out.println(ob.x);
}
```

This is constructor so valid
method is different constructor is
different

O/P:- 0

Example:-

```
class A
{
    final void show()
    {
        System.out.println("A class");
    }
}

class B extends A
{
    void show()
    {
        System.out.println("B class");
    }
}
```

Error because
final methods can not be
overridden

method overriding.

Example:-

```
final class A
{
}

class B extends A
```

X 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)

No name is a default in a package

No name is a default in a class

No name is a default in an Interface

Access modifier (AM)

Package:-

Package is a collection of subpackages, classes and interfaces

Example:-

```
package abc;
```

```
    class A {
```

```
        int a = 5;
```

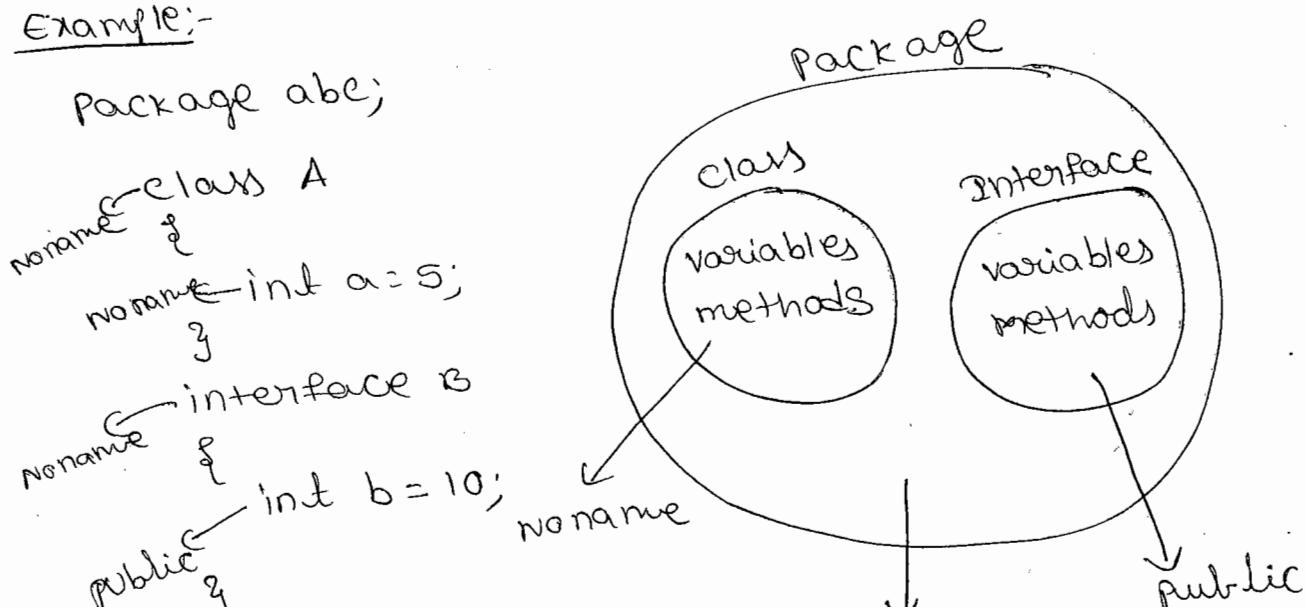
```
    interface B {
```

```
        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

scope

package int a=5; → A B C → classes
class ← private int b=10; → A → class
package ← protected int c=15; → A B C D
subclasses ← public int d=20; → A B C D E
(even outside the package) ↓ classes

visible to

Package xyz;

class D extends A

class E
↓
↓ classes

no subclasses Class B extends A

↓
↓
↓
↓

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 to Inheritance

Example:-

Class A

Here only one object created.

{ void showC)

{ System.out.println("welcome");

}

Class B extends A

{ public void sum(String args[]){

{ B ob = new BC();

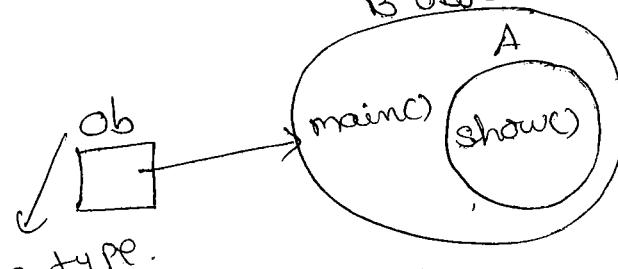
ob.showC();

}



B "IS-A" A

B object



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 to Composition

Class A

{ void showC()

{ System.out.println("welcome");

}

Class B

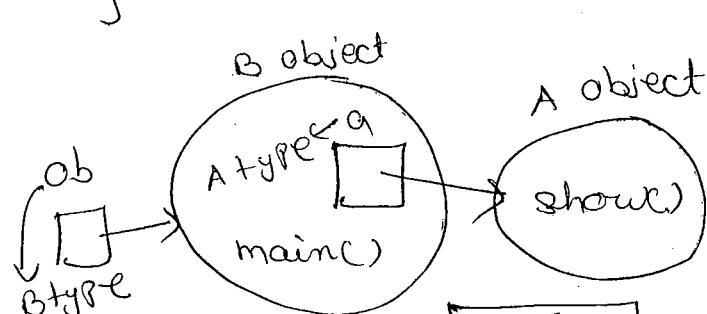
{ A a = new AC();

public void sum(String args[]){

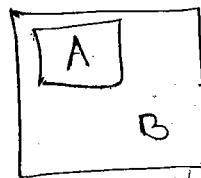
{ B ob = new BC();

ob.a.showC();

}



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 refers class A
In the above example two objects created

class student

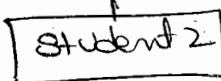
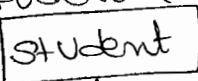
```
{ int rollno;  
  string name;  
  int landlineNo;  
  string address;
```

=

class student2 extends student

```
{ int mobileno;  
  string mailId;
```

student2 s = new student2;



student2 "IS-A" student

student2 object



mobileno
mailId

s
student2 type

student2 s = new student2();

class Address

```
{ string hno;  
  string city;  
  int pincode;
```

=

class EMP

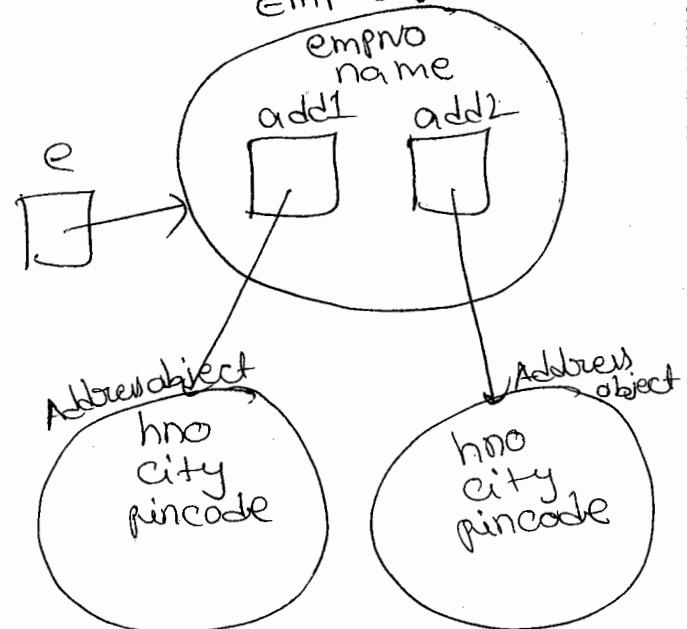
```
{ int empno;  
  string name;
```

Address add1 = new Address();
Address add2 = new Address();

=

EMP e = new EMP();

EMP object



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 them it is said to be method overloading.

The parameter list difference can be a no. of arguments, datatypes or order of arguments met 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;
    }
}

```

method definition

```

else if(c>b)
    return b;
else
    return c;
}

psvm(String args[])
{
    Demo ob = new Demo();
    int x = ob.max(23, 45);
    System.out.println(x);
    int y = ob.max(45, 88, 34);
    System.out.println(y);
}

```

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 them it is said to be method overriding.

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

methods can be overridden in only in Inheritance

13/3/15

Class A

```
{ void show()
  {
    System.out.println("A class");
  }
  void print()
  {
    System.out.println("print() method");
  }
}
```

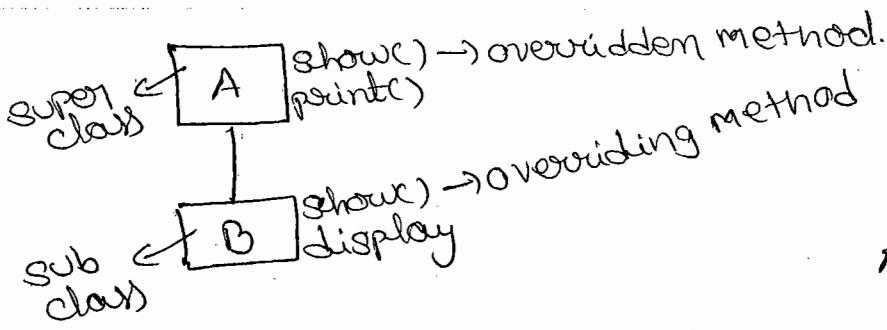
Class B extends A

```
{ void show()
  {
    System.out.println("B class");
  }
  void display()
  {
    System.out.println("display() method");
  }
}
```

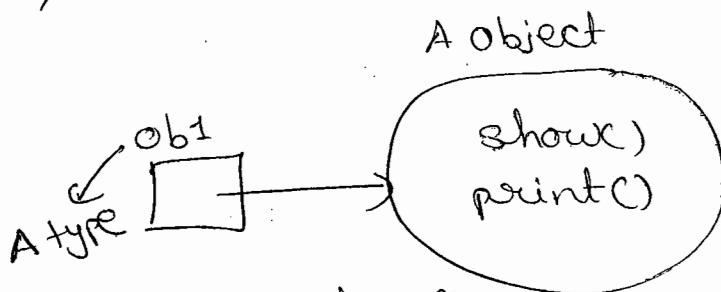
psvm(String args[])

```
{
  B ob = new B();
  ob.show();
  ob.print();
  ob.display();
}
```

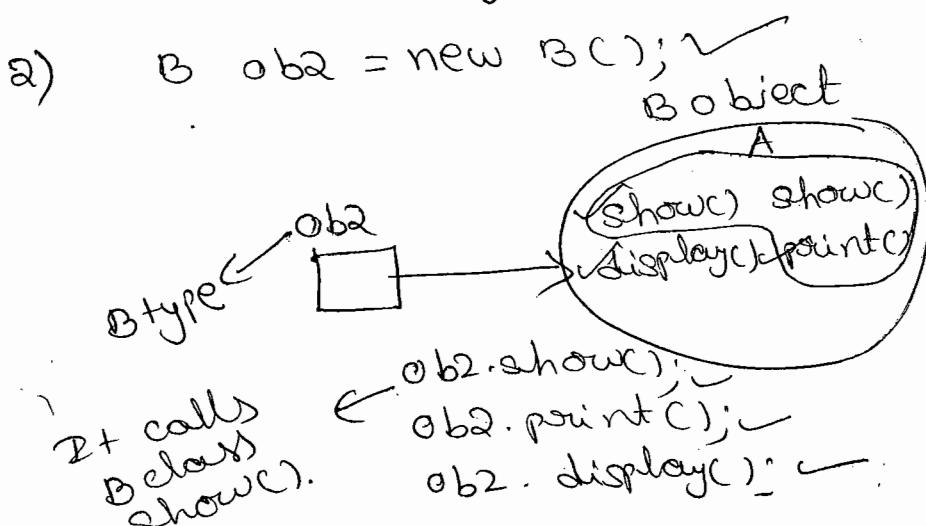
```
{ }
```



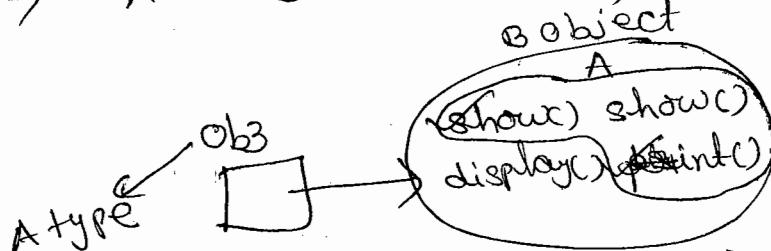
1) $A\ obj1 = \text{new } A(); \checkmark$



2) $obj1.show(); \checkmark$
 $obj1.point(); \checkmark$
 $obj1.display(); X$



3) $A\ obj3 = \text{new } B(); \checkmark$



$obj3.show(); \checkmark$
 $obj3.point(); \checkmark$
 $obj3.display(); X$

It ~~call~~ points B class
 show() it is overriding
 A class by B class
 Action ~~at~~ 6
 Below.

4) $B\ obj4 = \text{new } A(); X$

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 them 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 can't 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.

14/3/15

Covariant Return Type:-

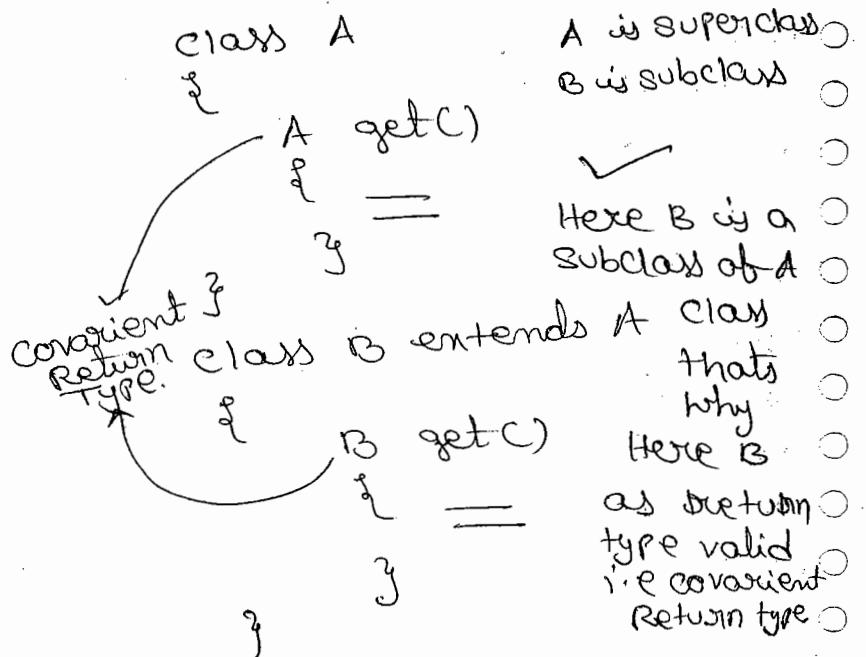
Java permits subclass type as a return type while overriding a method. This is known as covariant return type.

Class A

```
int get()  
{  
      
}
```

Class B extends A

```
float get()  
{  
      
}
```



private → no name → protected → public
↓
more
restrictive
access
modifier

public → protected → no name → private
↓
less
restrictive
access
modifier

→ In super class private is there but overridden is not possible (overriding)

class A

```
{ protected void show() }
```

→ In super class protected
is there in sub class
both protected and public
are possible valid.

class B extends A ✓

```
{ protected void show() }  
or  
public
```

→ 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 X

```
{ void show() }
```

{ =

{ }

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

2) class A

```
{ final void show() }
```

{ =

}

class B extends A X

```
{ final void show() }
```

{ =

{ }

3) class A

```
{ void show() }
```

{ =

{ }

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

class B extends A

```
{ final void show() }
```

{ =

{ }

```
class A
{
    void show() → overridden
    method
}
```

```
{    System.out.println("A class");
}
```

```
} class B extends A → This method belongs to A class
{
    void show() → overriding
    method
}
```

```
{    System.out.println("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

It is not
valid bcoz
overriding
method can
not be static

This method belongs to A class
This method have two
definitions

2) Class A → It is method
hiding.

```
{    static void show()
}
```

valid
but not
overriding

class B extends A

```
{    static void show()
    {
        =
    }
}
```

3) Class A

```
{    void show()
    {
        =
    }
}
```

X

class B extends A

```
{    static void show()
    {
        =
    }
}
```

Program to demonstrate method hiding:-

class A

```
{ static void showC()
    { System.out.println("A class"); }
}
```

class B extends A

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

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.

(E)

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 showC() valid
    { = }
}
```

class B extends A

```
{ void showC()
    { = }
}
public static void main(String args[])
{
    A ob = new B();
    ob.showC();
}
```

class A

```
{ static void showC()
    { = }
}
```

class B extends A

```
{ static void showC()
    { = }
}
```

```
public static void main(String args[])
{
    A ob = new B();
    ob.showC();
}
```

A type

```
ob
```

1) class A

{ private void show()

{ =

this method
is different
B class method
is different

class B extends A

why bcoz
B's private
scope is

void show()

{ =

valid within
the class
but not
overriding only.

3

obj
exec

2) class A

{ private void show()

{ =

3

class B extends A

{ private void show()

{ =

3 3

✓ valid but not
overriding.

3) class A

{ void show()

{ =

X not valid bcoz

in super class noname so
sub class must be noname,
protected, public

class B extends A

{

private void show()

{ =

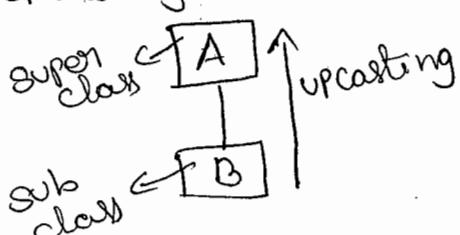
3 3

16/3/15

Upcasting :-

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

upcasting will be done by system implicitly



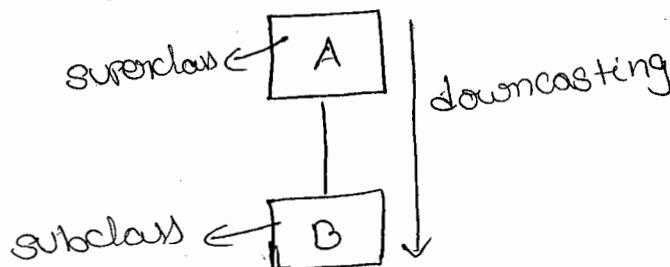
1) A ob = new B(); ✓ valid upcasting

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) $B \text{ ob} = \text{new } AC();$ X Invalid downcasting (There is no B memory)
 - 2) $A \text{ ob1} = \text{new } AC();$ ✓ $B \text{ ob2} = \text{ob1};$ X Invalid downcasting (There is no B memory we have to assign)
 - 3) $B \text{ ob} = (B)\text{new } AC();$ X [Here no upcasting]
↳ Explicit type conversion
 - 4) $A \text{ ob1} = \text{new } AC();$ ✓ $B \text{ ob2} = (B)\text{ob1};$ X Invalid downcasting [Here no upcasting]
 - 5) $A \text{ ob1} = \text{new } BC();$ ✓ valid upcasting [Here Explicit type conversion not present]
 $B \text{ ob2} = \text{ob1};$ X Invalid downcasting
 - 6) $A \text{ ob1} = \text{new } BC();$ ✓ valid upcasting
 $B \text{ ob2} = (B)\text{ob1};$ ✓ 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, or 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()` 2) `void print()`

It is also called null body method

3) `=====`

→ 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();
}`

3) `abstract class A`

{ `abstract void show();
 abstract void print();`

4) `class A`

{ `abstract void print();`

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

5) `abstract class A`

{ `void show();
 void print();`

- abstract classes can not be instantiated
- abstract class can be inherited into another class by using extends keyword → object creation.
- 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");
    }

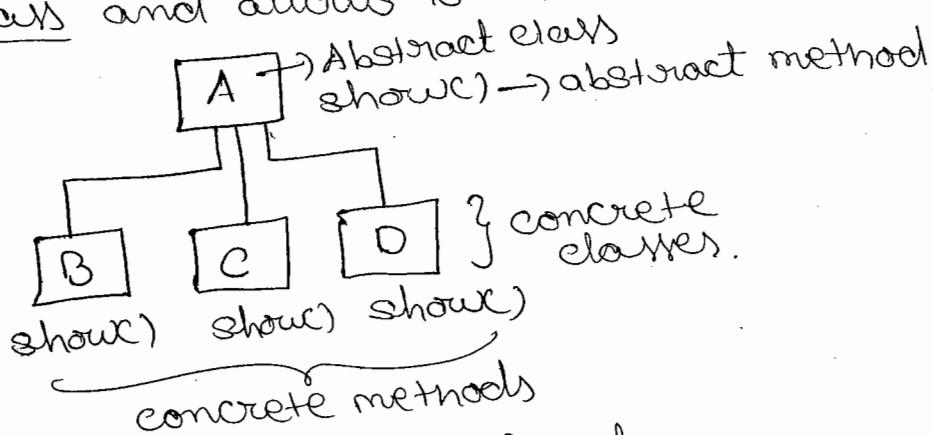
    void display()
    {
        System.out.println("display() method");
    }

    public static void main(String args[])
    {
        B ob = new B();
        ob.show();
        ob.print();
        ob.display();
    }
}

```

- 1) A ob1 = new A(); X abstract class can not be instantiated
- 2) B ob2 = new B(); ✓ It is valid
 ob2.show(); ✓
 ob2.point(); ✓
 ob2.display();
- 3) A ob3 = new B(); ✓ → By using this we can access only A class members
 ob3.show();
 ob3.point();
 ob3.display(); X → we can not access B memory
- 4) B ob4 = new A(); X abstract class can not be instantiated
 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
 Implicitly

{
 B()
 super();
 public static void main(String args[]){
 new B();
 }
}

}{

→ 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[])

{
 sopln("welcome");

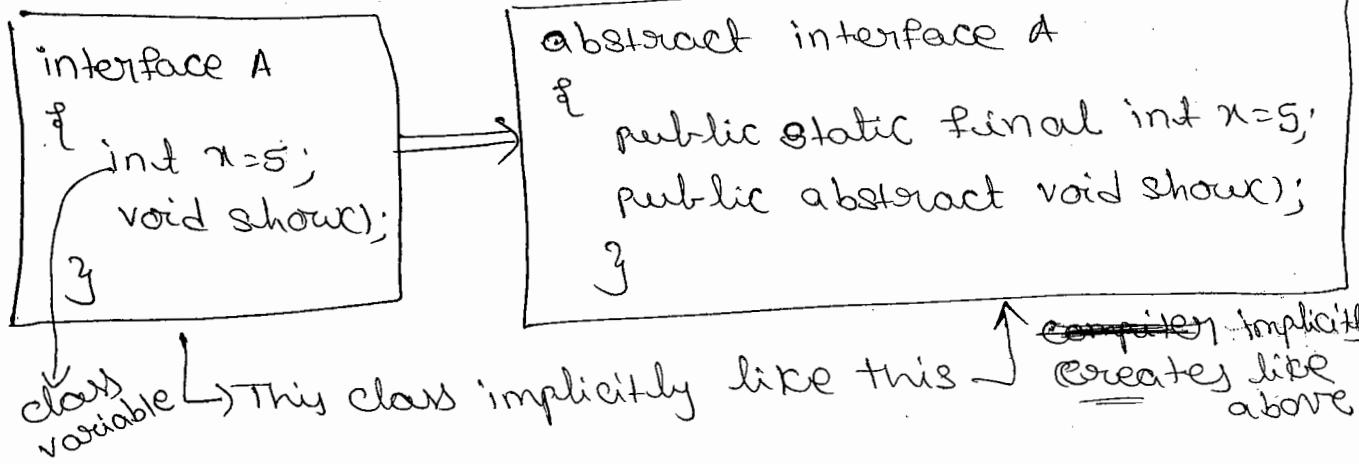
}{
 }{

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 instantiated
- 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 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(); }  
public static final int n=5;
public abstract void show();
```

✓ implicitly

class B implements A

```
{ int y=10;
  public void show() → access modifier rule is
  same or less restrictive
  so here
  { sopln("show() method");
    }
  void point();
  { sopln("point() method");
    }
```

psvm(String args[])

```
{ sopln(A.x);
  B ob = new B();
  sopln(ob.y);
  ob.show();
  ob.point();
```

} program does not compile

interface A

{ int x=5;
void show(); } *implicitly public static final abstract void show();*

class B implements A

{ int y=10;
public void show(); } *Here also must place
public access
modifier.*
{ sopl("show() method"); }

25 void print()
{ sopl("print() method"); }

psvm(string args[])

{ sopl(A.x);
B ob = new B();
sopl(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 obj = new B(); ✓

S.o.P(obj.x); ✓

S.o.P(obj.y); X B member we can not access with A reference

obj.show(); ✓

obj.print(); X

4) B obj = 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

{ =

}

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;
        }
    }

```

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

ex:- capsule

```

    {
        int getAge()
        {
            return age;
        }
    }

```

q.p:- 100

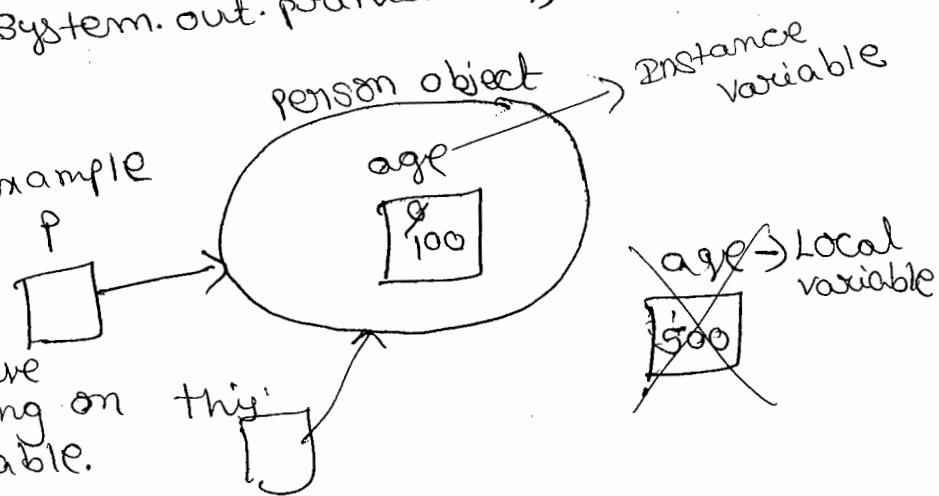
Class Demo

```

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

```

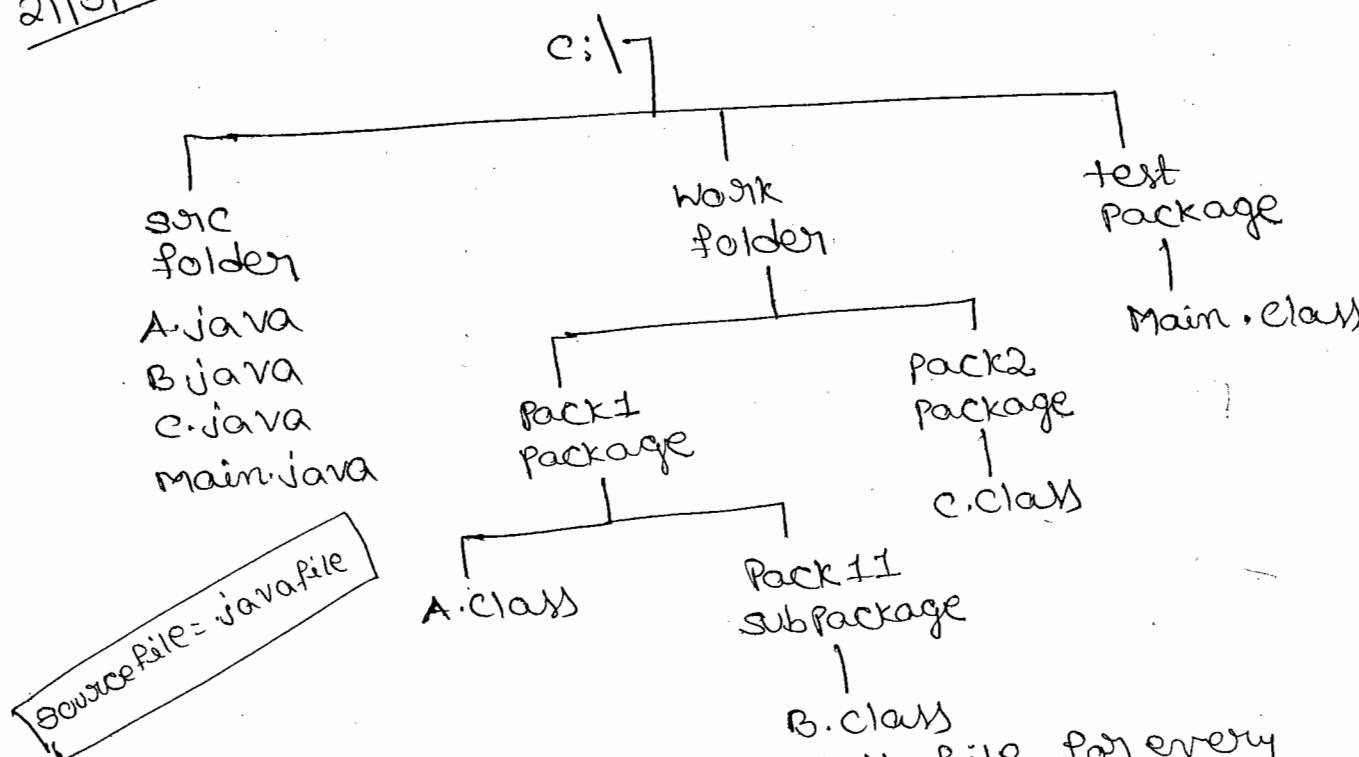
→ In the above example
 age variable bound with setAge & getAge methods and there methods operating on this same age variable.
 this is known as an encapsulation



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 create click **OK**
new file

md → making directory

cd → changing directory

A.java

```

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

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

compilation:-

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

↓
It omits
current
directory

↳ destination
directory

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

B.java

```

C package Pack1.Pack11;
C public class B {
C     {
C         public void cube(int a) {
C             System.out.println(a*a*a);
C         }
C     }
C }
```

compilation:

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

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

C.java

```

C package Pack2;
C public class C {
C     {
C         public void max(int a, int b) {
C             if(a>b)
C                 System.out.println(a);
C             else
C                 System.out.println(b);
C         }
C     }
C }
```

compilation:-

```
c:\src>javac -d c:\work e.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
```

```
{    public void main(String args[])
```

```
{        A ob1 = new A();
```

```
        ob1.add(23, 45);
```

```
B ob2 = new B();
```

```
ob2.cube(5);
```

```
C ob3 = new C();
```

```
ob3.man(24, 88);
```

```
}
```

```
}
```

Compilation and Execution:-

c:\src>set classpath=%classpath%;

c:\src>javac -d c:\ main.java

c:\src>cd..

c:\>java test.main

↳ package name. class name.

68

125

88

23/3/15

c:\work

↓ It informs computer that all packages are exist under work folder
→ append mode.

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 class is a superclass for all

Java classes

→ In Java each and every class either directly or indirectly derived from java.lang.Object

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

Javadoc documentation:-

It contains package description, classes description, etc.

Interface description, methods description - etc.

file:///E:/Jdt-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.

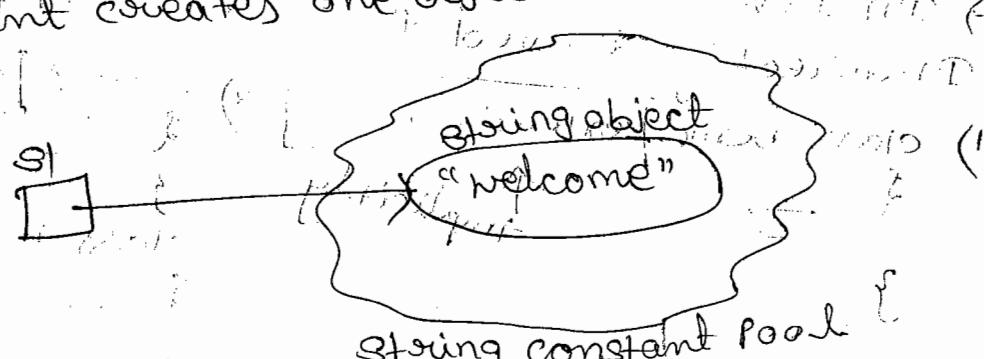
→ every string literal itself an object of string class

Ex:- "welcome" → It is an object of string class with this we can save memory.

String s1 = "welcome";

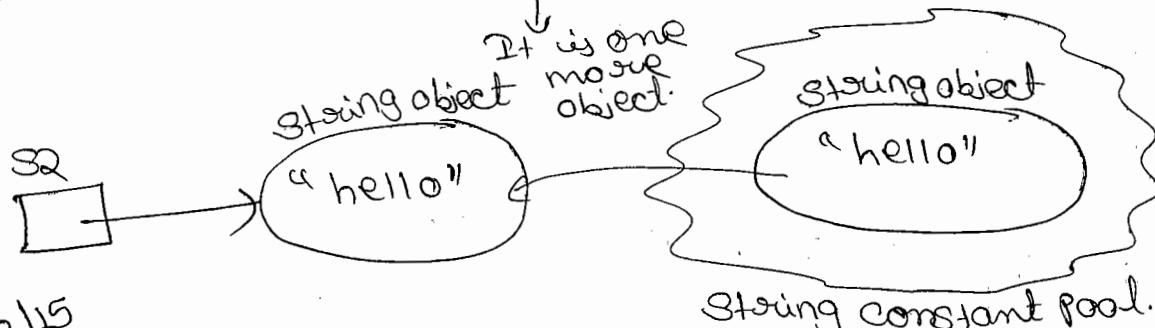
This statement creates one object in string constant pool.

Similarly,



`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`

constructors:-

`public String();`

`public String(String);`

`public String(char[]);`

`java.lang.String`

`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:-

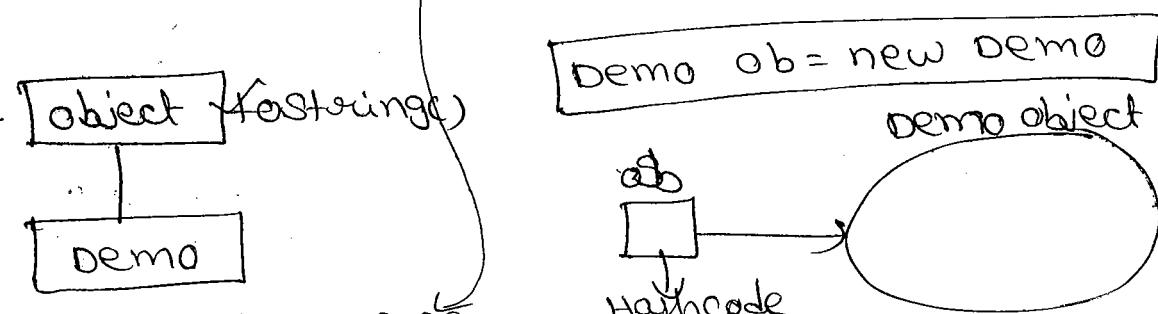
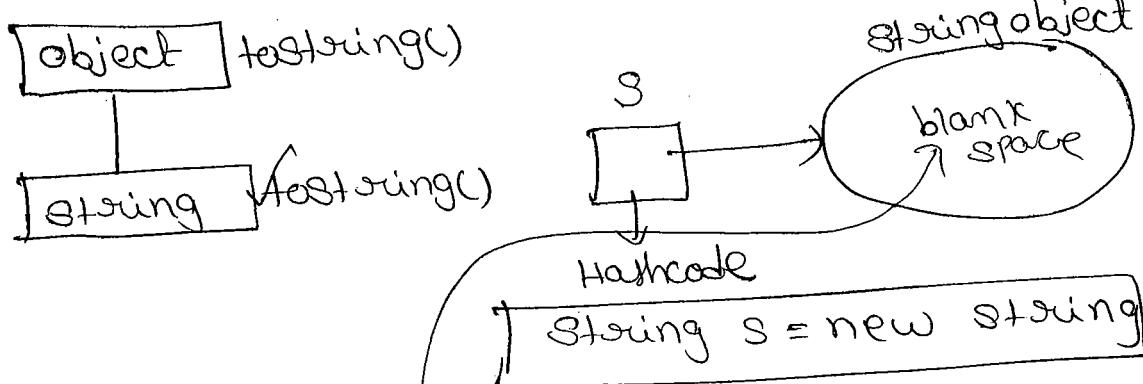
class Demo

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

It is defined in library we are accepting.

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

The above program contains default String() constructor.



O/p:- → blank space
Demo@35a2e

- `toString()` method of `Object` class returns class name
Hexa-decimal format
→ `toString()` method of `String` class returns content of string object
→ `toString()` method of `Demo` class can be overridden by programmer to change the definition

Example:-

```
class Demo extends Object {  
    public String toString() {  
        return "welcome";  
    }  
    public static void main(String args[]) {  
        Demo ob = new Demo();  
        System.out.println(ob);  
    }  
}
```

Implicitly
This super class (`Object` class)
`Object` class contains `toString` that why we are overriding here

Implicitly
`ob.toString()`, `Object.toString()`

O/p:- welcome

class demo extends String
{} = X because String is a final class

Example:-

```
class Demo
{
    public static void main(String args[])
    {
        byte b[] = {97, 98, 99, 100}; } It is converting
        String s1 = new String(b); } byte[] array to
        System.out.println(s1);
        char ch[] = {'h', 'e', 'l', 'l', 'o'}; } It is
        String s2 = new String(ch); } converting
        System.out.println(s2);
    }
}
```

→ There are already defined in library
already built in. Here we are using
O/P:- abcde If we declare like that no error because
hello. every java.lang is default package
in above class. String class is related
to java.lang package.

Example:-

```
boolean b = s1.equals(s2),
System.out.println(b);
```

↓
we can write those two
lines in a single line
System.out.println(s1.equals(s2));

Example:-

class demo

{ psvm(String args[])

logic inbuilt

{ Here we are converting String s = new String("hello");
byte b[] = s.getBytes();

String to byte array for(int i=0; i<b.length; i++)

{ sopm(b[i]);

} char ch[] = s.toCharArray();
for(int i=0; i<ch.length; i++)

{ sopm(ch[i]);

}

}

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();

sopm(n);

char ch = s.charAt(2);

sopm(ch);

}

O/P:- 5

hello
0'1234→index

NOTE:-

String constant pool does not allow duplicates

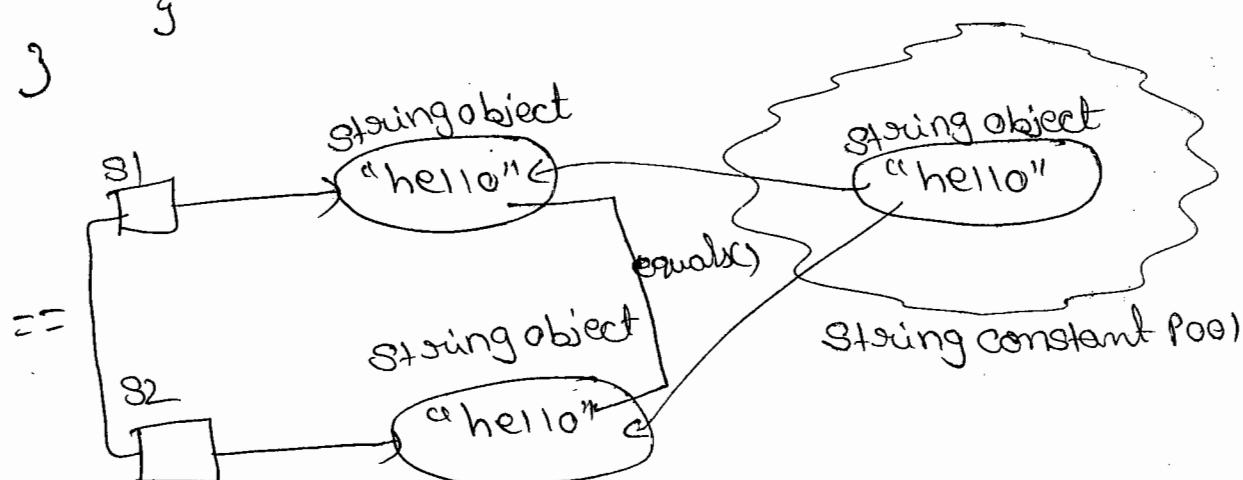
→ equals() of String class compares the contents of string objects whereas equals operator compares the Hashcode

↓
(==)

Example:-

```

class Demo
{
    public static void main(String args[])
    {
        String s1 = new String("hello");
        String s2 = new String("hello");
        System.out.println(s1.equals(s2));           o/p:- true
        System.out.println(s1==s2);                  o/p:- false
    }
}
  
```



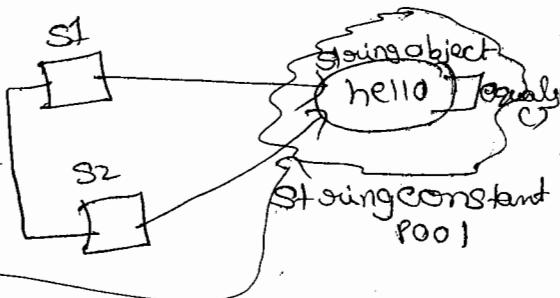
25/3/15

→ equals() method of Object class compares the Hashcode

Example:-

```

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



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()
that's 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
```

```
{  
    public static void main(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 i) The object of StringBuffer class is immutable

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:-

O/P:- Taj
Tajmahal

```
class Demo
```

```
{  
    public static void main(String args[]){  
        {
```

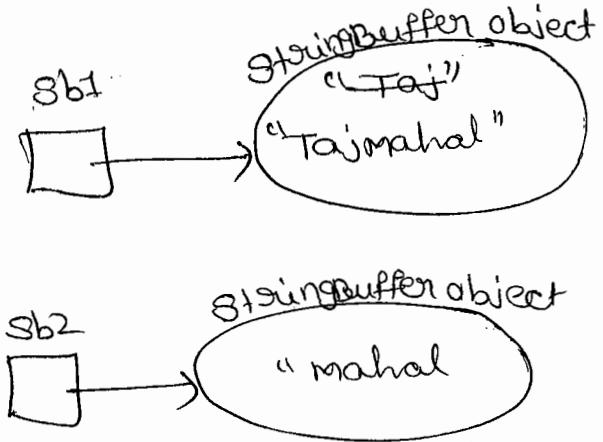
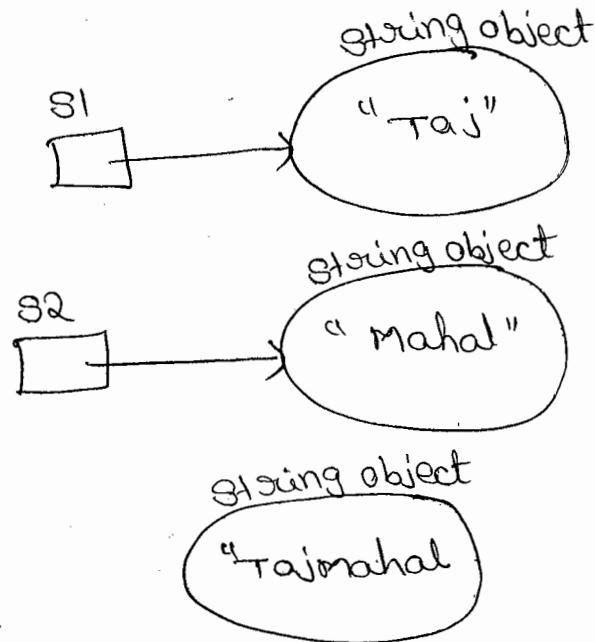
```
            String s1 = new String("Taj");  
            String s2 = new String(" mahal");
```

```
s1.concat(s2);
```

```
sopln(s1);
```

```
StringBuffer sb1 = new StringBuffer("Taj");  
StringBuffer sb2 = new StringBuffer(" mahal");  
sb1.append(sb2);
```

```
} } sopln(sb1);
```



Here sb2 append to sb1
so Tajmahal result is
stored in sb1 object

- 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 that's why one more object created and result stored in that object

Example:-

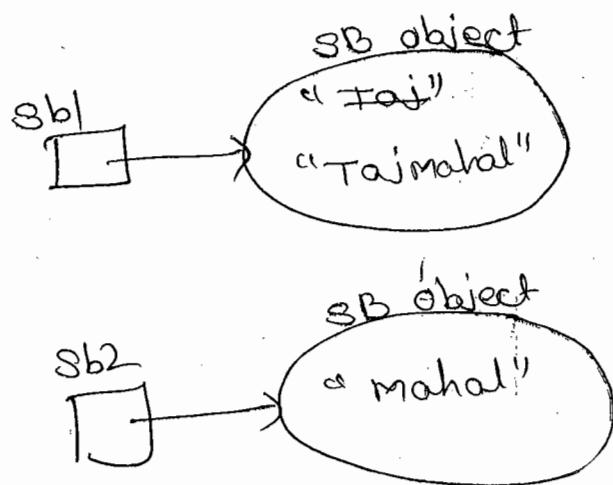
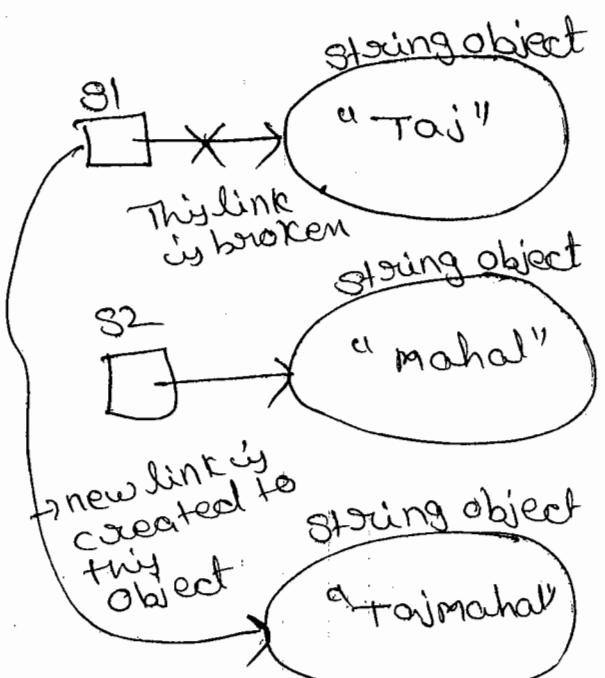
class demo

```

class demo
{
    public void psvm(String args[])
    {
        String s1 = new String("Taj");
        String s2 = new String("Mahal");
        s1 = s1.concat(s2);
        System.out.println(s1);
        StringBuffer sb1 = new StringBuffer("Taj");
        StringBuffer sb2 = new StringBuffer("Mahal");
        sb1.append(sb2);
        System.out.println(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

- i) immutable
- ii) not synchronized

StringBuffer

- i) mutable
- ii) synchronized

StringBuilder

- i) mutable
- 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
```

```
{ public static void main(String args[])
```

```
{ String s = " welcome to java";
```

```
 StringTokenizer st = new StringTokenizer(s);
```

```
 int n = st.countTokens();
```

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

```
}
```

```
}
```

O/P: 3

for creating
object constructor
required.

javac Demo
java Demo

Example:-

```
import java.util.*;
```

```
class Demo
```

```
{ public static void main(String args[])
```

```
{ String s = " welcome to java";
```

```
 StringTokenizer st = new StringTokenizer(s);
```

```
 while(st.hasMoreTokens())
```

```
{ System.out.println(st.nextToken());
```

```
}
```

```
}
```

Output:

26/3/15

Command line arguments :-

short/long

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

c:\>java Demo welcome to Java

0 1 2 → index are received by main method

 |
command line arguments

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 {
    public static void main(String args[]) {
        for (int i = 0; i < args.length; i++) {
            System.out.println(args[i]);
        }
    }
}
```

command line arguments are received here if we pass integers also it will treat like a string

O/P:-

c:\>javac Demo.java
 c:\>java Demo welcome to Java
 welcome
 to
 Java
 Java Demo 10 20 30 40

10 → string type

Ex:- class Demo

```
class Demo {
    public static void main(String args[]) {
        System.out.println(args[0] + args[1]);
    }
}
```

O/P c:\>java Demo.java
 c:\>java Demo Taj Mahal

Taj Mahal

c:\>java Demo 10 20

10 20

↓ 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
- 2) short
- 3) int
- 4) long
- 5) float
- 6) double
- 7) char
- 8) boolean

| Wrapper Class |
|---------------|
|---------------|

- 1) Byte
- 2) Short
- 3) Integer
- 4) Long
- 5) Float
- 6) Double
- 7) Character
- 8) Boolean

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

java.lang.Integer

constructor:- constructor 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; 
 New primitive data type

Integer ob = new Integer(x);
 Reference data type ob  Integer object

Reference data type to primitive data type

Integer ob = new Integer(5);
 NOT Integer object



int x = ob.intValue();
 ↓ primitive datatype 

- whenever the object is created that time constructor called
- some of the applications a 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

Example:-

```
int n=5;
```

↓
primitive data type

Integer ob=n; ⇒ Auto boxing

↓
Reference data type. (without constructor we are converting)

int to Integer ↗ corresponds
char to character ↗ only
possible

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 n=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

retrieving the box unboxing (without method) with out constructor that is the reason these introduced

java.lang.Integer:-

Methods:-

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

Example:-

Class Demo

{ public void main(String args[])

{ int a=30;

System.out.println(a);

There is string s = Integer.toOctalString(a);

lines System.out.println(s);

we can write in System.out.println(Integer.toOctalString(a));

single System.out.println(Integer.toHexString(a));

System.out.println(Integer.toBinaryString(a));

}

30 \rightarrow Decimal

$8 \overline{) 30}$
3 - f \uparrow

36(8) \rightarrow Octal

$16 \overline{) 30}$
1 - 14 \uparrow

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

1e(16) \rightarrow Hexadecimal.

$2 \overline{) 30}$
2 - 15 - 0
 $2 \overline{) 15}$
2 - 7 - 1
 $2 \overline{) 7}$
2 - 3 - 1
1 - 1 \uparrow

1110(2) \rightarrow binary

int a = 30;

↳ decimal literal

int b = 036; \downarrow
↳ octal literal

int c = 0x1e; \downarrow
↳ hexadecimal literal

int d = 01e; \uparrow

int e = 0b1110; \downarrow
↳ binary literal;

int f = 0B1110; \uparrow

decimal to octal

$30_{(10)}$ \rightarrow 36(8)

octal to decimal

$36_{(8)}$ \rightarrow $30_{(10)}$

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[]) } both arguments are received here

{ int x = Integer.parseInt(args[0]); and passed first argument to

int y = Integer.parseInt(args[1]);

System.out.println(x+y);

}

O/P: c:\\$javac Demo.java

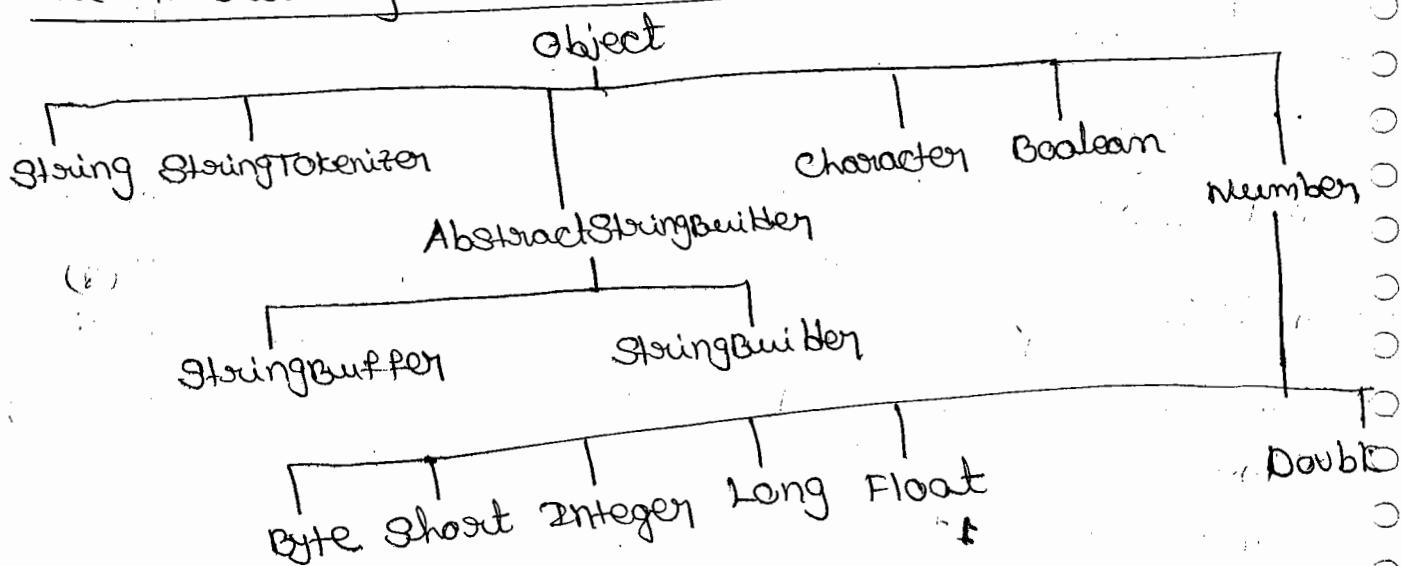
Java Demo 10 20

30

c:\\$ java Demo Taj mahal

NumberFormatException.

The Hierarchy of String Related classes and wrapper classes



Example:-

class demo

{
 public static void main(String args[])

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

e:\>java demo 10 20
30

a 10 " → string

↓ → parsing

10 → int

e:\>java demo taj mahal
NumberFormatException

Exception Handling

23/3/15

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

→ Exception means Run time errors are called Exceptions

→ only Run time errors are called Exceptions

→ All errors are not Exceptions

→ All Exceptions are classes in java

→ 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

→ subtraction

addition mistakenly (-) place them

unexpected o/p comes

We want to perform addition mistakenly (-) place them

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 at least one catch block or finally block

→ Whenever Exception occurred in a program then the object of related Exception class is created by JVM and passed to Exception Handler (catch block)

→ catch block is called as Exception Handler and it handles the exception

→ After Handling Exception catch block & code is Executed

~~new ArrayIndexOutOfBoundsException()~~

try

{ ≡

} catch(ArrayIndexOutOfBoundsException ae)

↳ created by JVM.

~~size~~

catch block

Exception Handler

size

5

Range

0 to 4

5 (ArrayIndexOutOfBoundsException)

4

0 to 3

4 ("")

Example:-

```
class Demo
{
    public static void main(String args[])
    {
        int a = Integer.parseInt(args[0])
        int b = Integer.parseInt(args[1]);
        int c = a/b;
        System.out.println(c);
    }
    catch(ArrayIndexOutOfBoundsException ae)
    {
        System.out.println("Please pass two arguments");
    }
    catch(NumberFormatException ne)
    {
        System.out.println("Please pass two integers only");
    }
    catch(ArithmaticException ae)
    {
        System.out.println("Please pass second argument except zero");
    }
}
```

Output:-

```
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 (y)
```

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 don't 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 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

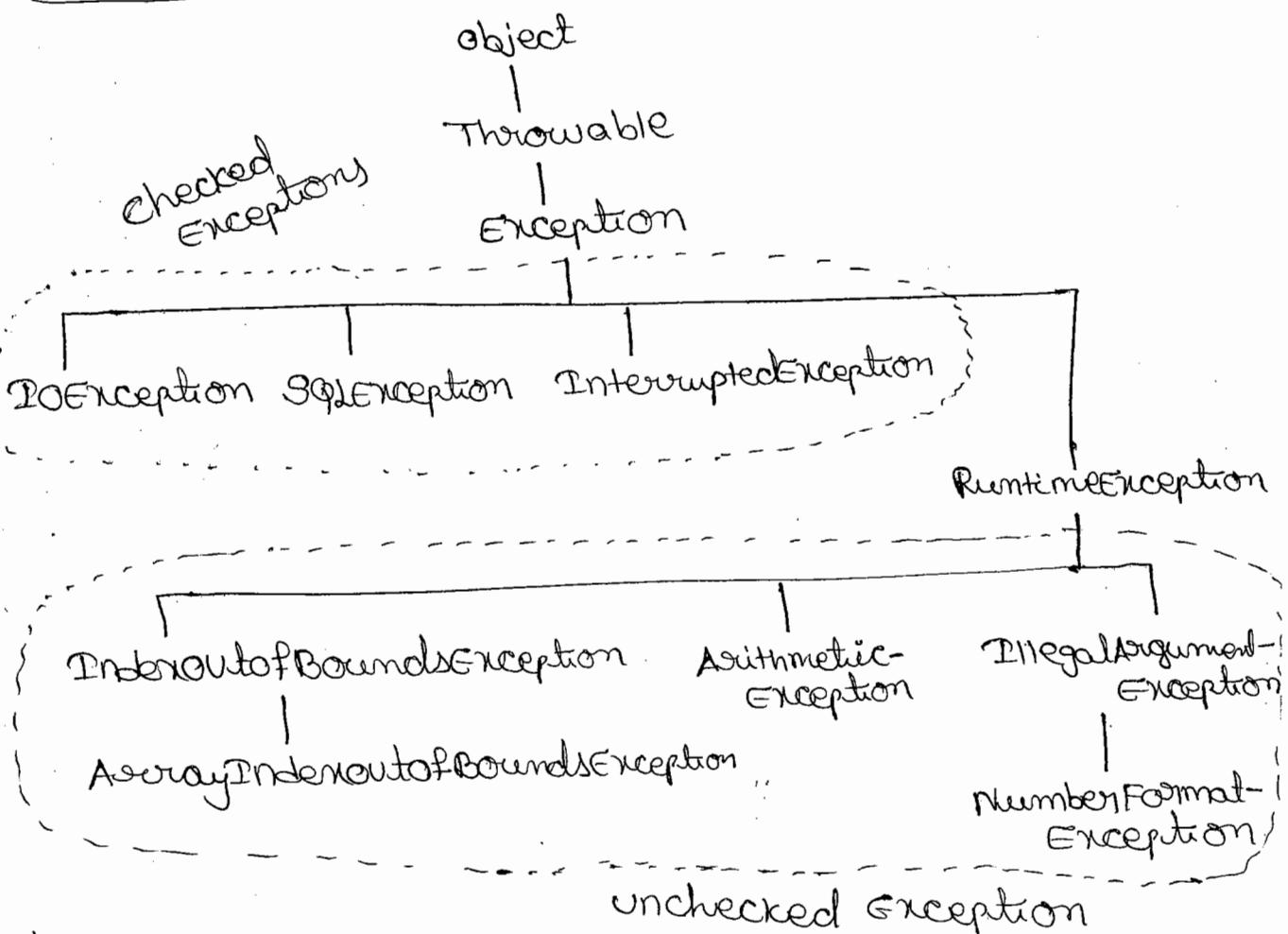
→ Handling unchecked exceptions are optional by programmer.

→ main is not a library method

→ ~~we can no~~ library

→ we can not write open {} for library methods

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.

→ 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 caught~~
import java.io.*;
↳ It is called as compiletime error

```
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
{
    public static void main(String args[])
    {
        try
    }
```

```
        FileInputStream fis = new FileInputStream(args[0]);
        catch(FileNotFoundException fe)
    }
```

↳ system.out.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
{
    public static void main(String args[])
    {
        try
        {
            FileInputStream fis = new FileInputStream(args[0]);
            catch(Exception e)
            {
                System.out.println(e);
            }
        }
    }
}
```

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

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

- Here Throwable class testing() 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[]) → 5 receiver here
with that

```
{ for{
```

Cube method is instance method int x = Integer.parseInt(args[0]);

create object Demo ob = new Demo();
ob.cube(x);

→ extracts here
then converts int

→ Here Throw-
able class
testing() called

```
} catch(negativeNumberException ne)
```

```
{ System.out.println(ne);
```

Compilation & Execution

↑ implicitly
with
that
class
name
comes.

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

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

```
c:\>java demo 5
```

125

```
c:\>java demo -3
```

negativeNumberException

```

C Program to create unchecked Exception :-
C class NegativeNumberException extends RuntimeException
C {
C     {
C         Here NNE is predefined
C     }
C     class Demo
C     {
C         void cube(int a) throws NegativeNumberException
C         {
C             if(a>0)
C                 System.out.println(a*a*a);
C             else
C                 throw new NegativeNumberException();
C         }
C     }
C     public static void main(String args[])
C     {
C         int x = Integer.parseInt(args[0]);
C         Demo ob = new Demo();
C         ob.cube(x);
C     }
C }

```

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 ~~to~~ ^{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

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.out.println(ne);
    }
}
```

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

2) By using throws clause

cube method is throwing exception here main method also throwing exception at last system provides one catch block

Example:-

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

that catch block handles exception

→ 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 exception that is checked exception
- By using library we can check whether it is checked or unchecked exception

26/3/15

JDK, JRE, JVM & JIT

These all are developed in C language

JDK

JDK stands for Java Development Kit

It contains JRE, development tools (javac, javaP, Java, javadoc... etc), additional libraries and supporting files for development tools.

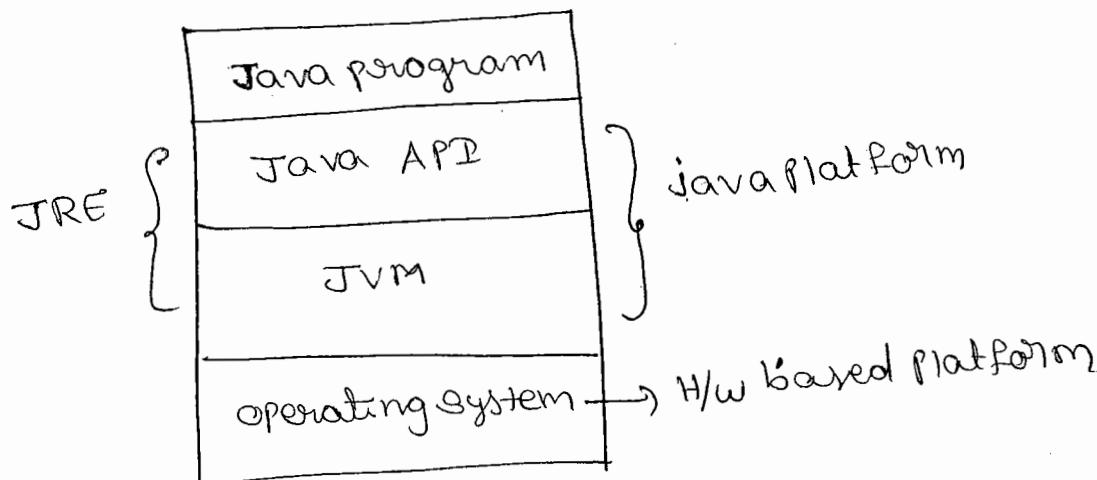
JDK is called as Java software

→ 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 (javaapi) (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

e:\program files\java\jres
is called as public JRE
(outside the JDK)

→ public JRE used by browser
to run Java Applets

→ public JRE has a
registry with operating
system

private JRE

e:\program files\java
JDK1.6.0_11 JRE is called
as private JRE
(inside the JDK)

→ private JRE used by
operating system to run
Java applications

→ private JRE has no
registry 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 gc() method of java.lang.System class

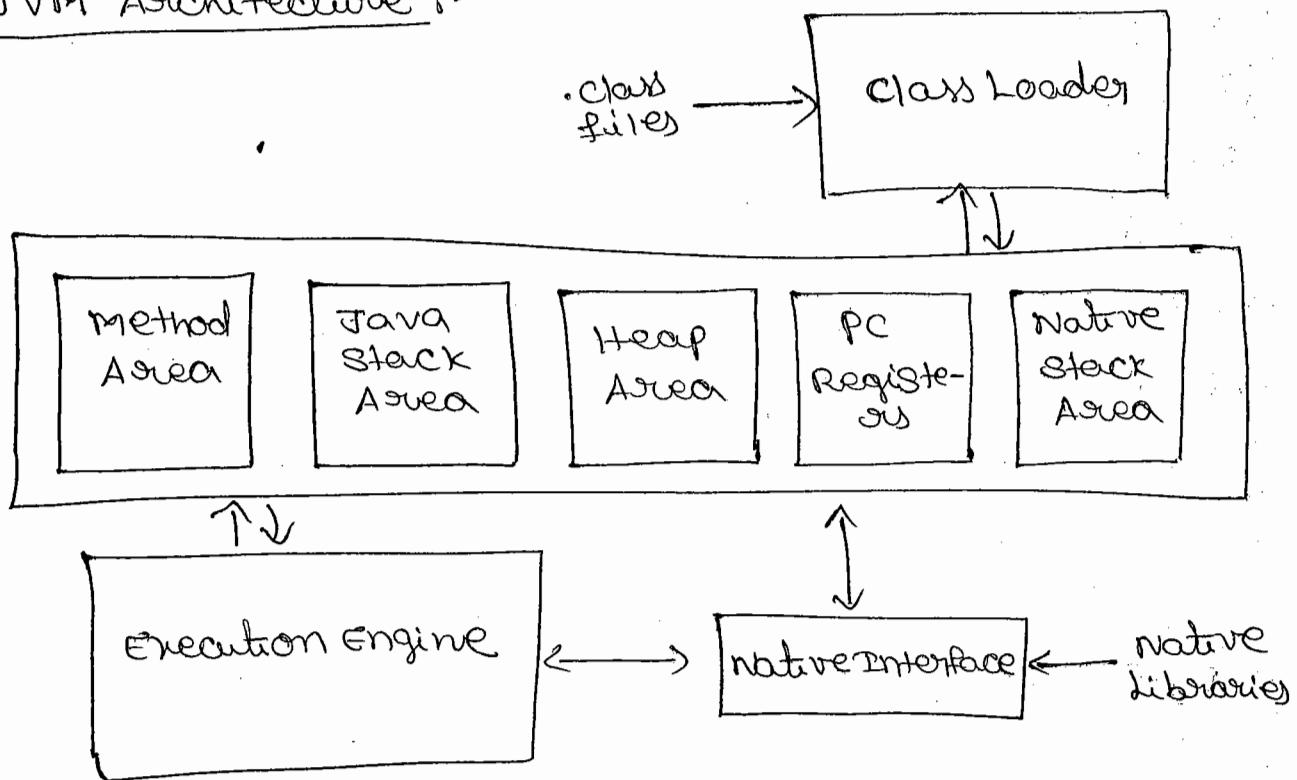
4) Security manager:-

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:-

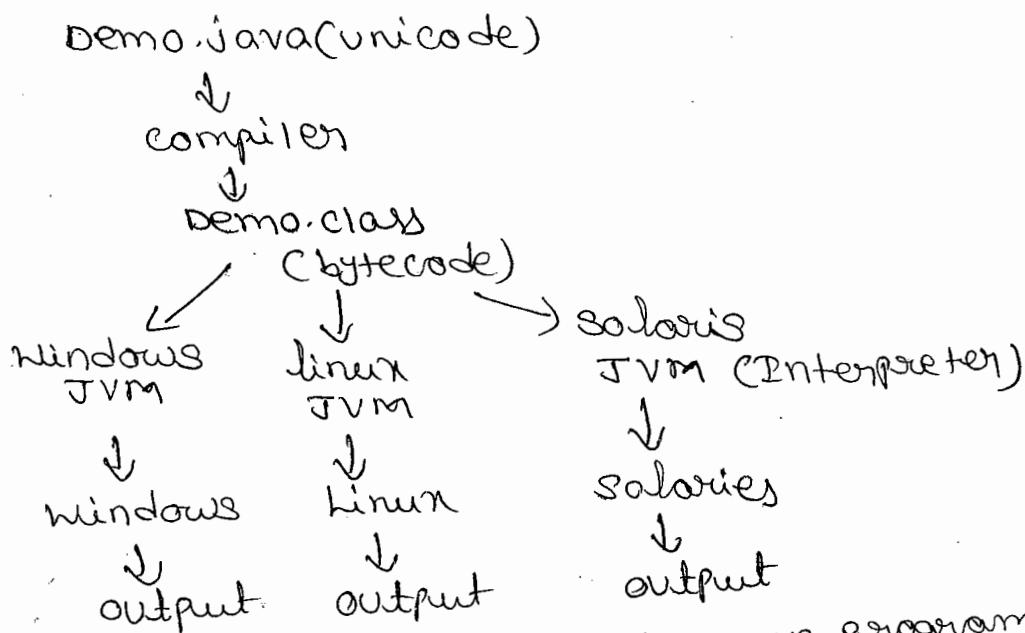
It contains instance variables and objects

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 be 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 Solaris - etc.

JIT compiler:- (JustInTime compiler)

- JIT compiler:- (Just In Time 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, puts into cache memory and loads several times as needed.

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

static imports:

static imports:-
This feature allows to avoid class name with static members.

Syntax:- import static package-name . sub-package-name .
 classname ;

C → here * indicates all static members.

Eg:- `import static java.lang.Integer.*;`

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

```
import static java.lang.Integer.*;
```

```
class Demo
```

```
{  
    public static void main(String args[])
```

```
    {  
        int x = parseInt(args[0]);
```

```
        int y = parseInt(args[1]);
```

```
        System.out.println(x+y);
```

```
}  
}
```

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

O/P:- 23 → Both varargs and for each loop features
24 added in jdk 1.5 version in 2004.
65
34
4323
345
389
967
478
932

Strings in switch statement:-

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

→ This feature added in jdk 1.7 version in 2011.

Eg:- class Demo

```
=
{
    public static void main(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.

e.g:- class Demo

```
{ public static void main (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

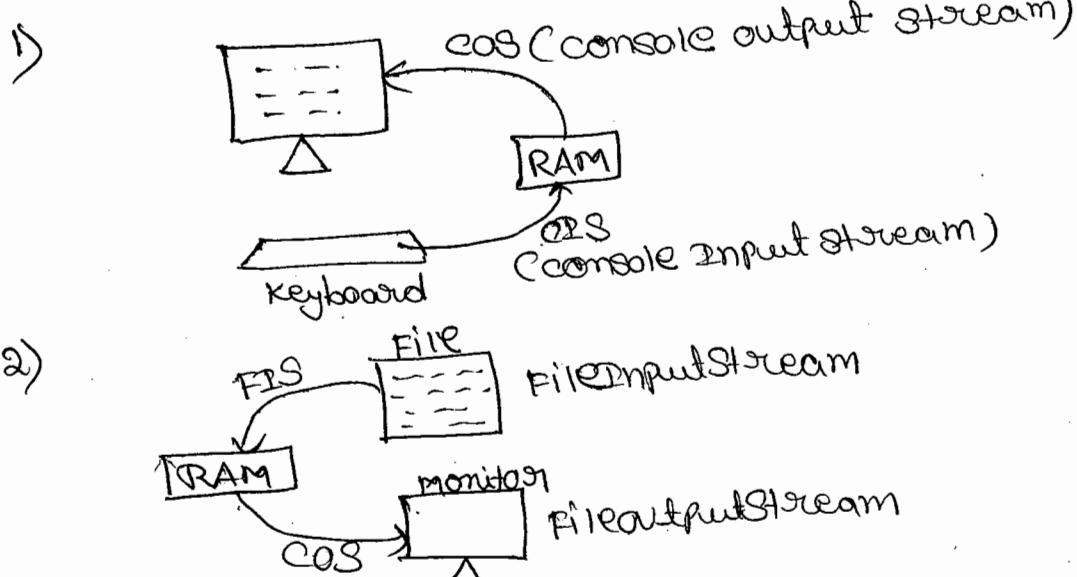
A Stream is a flow of data from source to destination

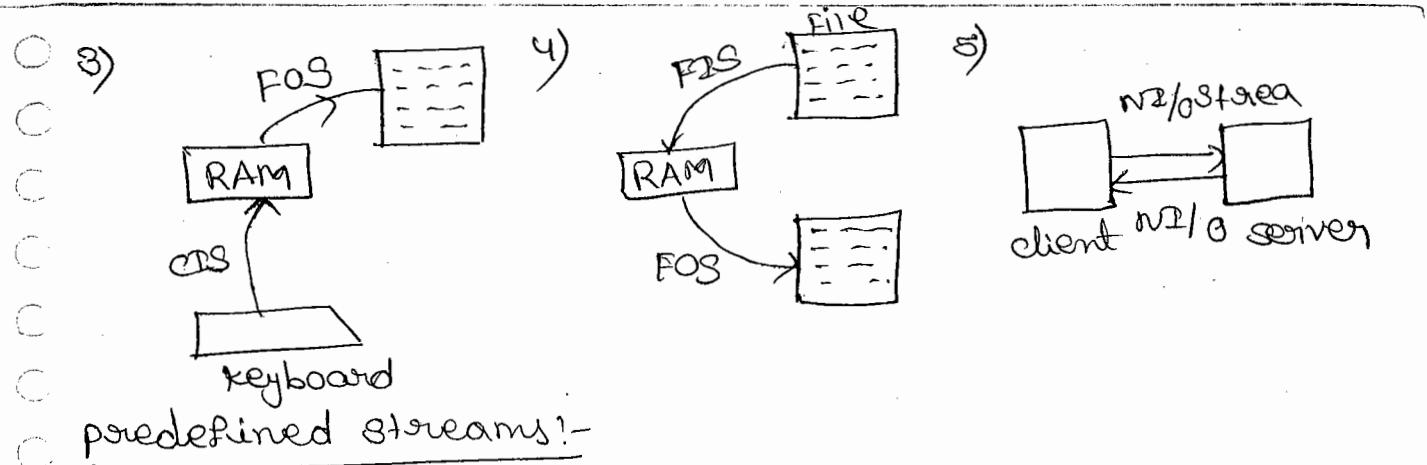
→ A source can be a keyword, file, client, server — etc

→ 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





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.OutputStream class.

3) err:- err is an object reference of java.io.OutputStream class.

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

→ Relationship between system class and InputStream class.

class is "HAS-A" relationship

Note:- system class "HAS-A" InputStream class and

OutputStream class.

java.lang.System:-

object reference:-

```
public static final java.io.InputStream in;
```

```
public static final java.io.OutputStream out;
```

```
public static final java.io.OutputStream err;
```

java.io.OutputStream

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.

```

C → system.out.println("Taj");
C   system.out.println("mahal");
C
C   O/P:- Taj
C   mahal.
C
C → system.out.println("Taj");
C   system.out.println();
C   System.out.println(" mahal ");
C
C   O/P:- Taj
C   mahal.

```

Java.io.InputStream:

method:-

public int read(byte[]) throws IOException;
 → It is used to read data from keyboard.

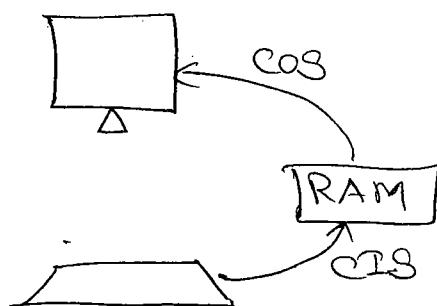
Task:-①

```

C:|> javac demo.java
C|> java Demo
C|          ↴
C| Enter any number: 15
C|          ↴
C|      15

```

Task diagram:-



Program to demonstrate predefined streams:-

```
import java.io.*;
class demo
{
    public static void main(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.out.println(ie);
        }
    }
}
```

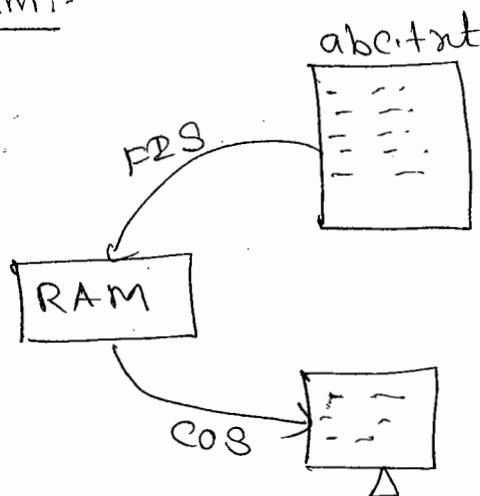
Task 2

c:\>javac Readbemo.java

c:\>java Readbemo abc.txt

==== } content of
==== } abc.txt file

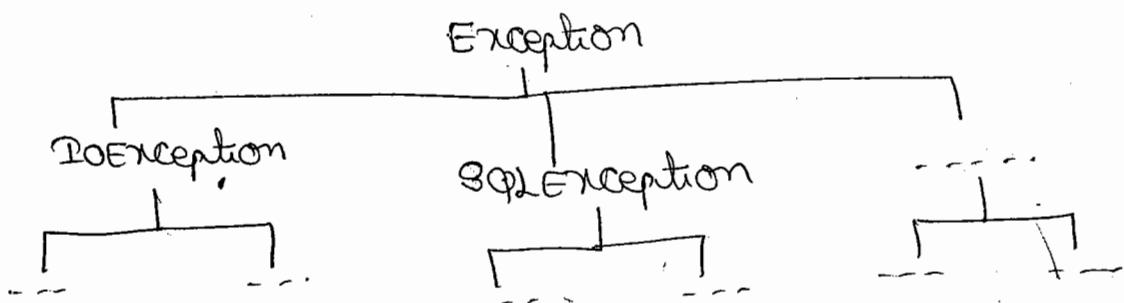
Diagram:-



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 d1*.*.java
Constructor:- filename
 public FileInputStream(String)
 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
 byte b[] = new byte[n];
 fis.read(b);
 String s = new String(b);
 System.out.println(s);
 } catch(Exception e)
 {
 System.out.println(e);
 }
 }
 }



- 1) `try` → This catch block handles
`{ = }` exception class and all its
`} catch(Exception e)` sub classes. It means all
`{ = }` exceptions because all exceptions
`}` are derived from exception class
- 2) `try` → This catch block handles SQLException
`{ = }` and all its subclasses
`} catch(SQLException se)`
`{ = }`
- 3) `try` → This catch block handles IOException
`{ = }` and all its subclasses.
`} catch(IOException ie)`
`{ = }`
- 4) `try` → This catch block handles IOException and
`{ = }` all its subclasses
`} catch(IOException ie)`
`{ = }`
`} catch(SQLException se)` This catch block handles
`{ = }` SQLException and all its subclasses
`}`

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

O 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

O Program to demonstrate finally block :-

O import java.io.*;

O class ReadDemo

O {

 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.out.println(e);

 }

 finally

 try {

 fis.close();

 } catch(Exception e)

 {

 System.out.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.Closeable interface can be used as resource with try block.

- a class that implements ~~java.io.Closeable interface~~ can only used as ~~java.io.Closeable~~ interface. 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  
{  
    public static void main(String args[]) throws Exception  
{  
        FileInputStream fis = new FileInputStream  
            (args[0]);  
        byte b[] = new byte[fis.available()];  
        fis.read(b);  
        System.out.println(new String(b));  
    }  
}
```

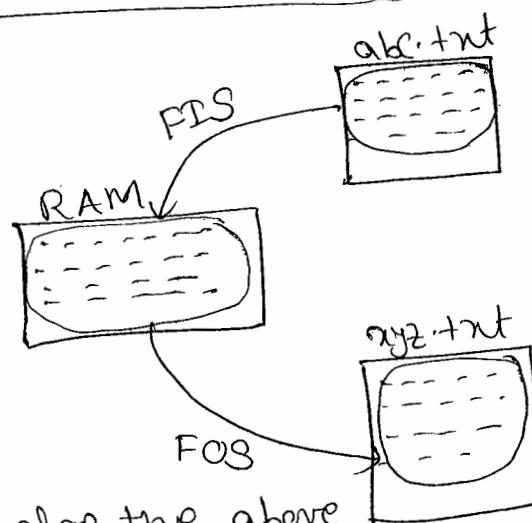
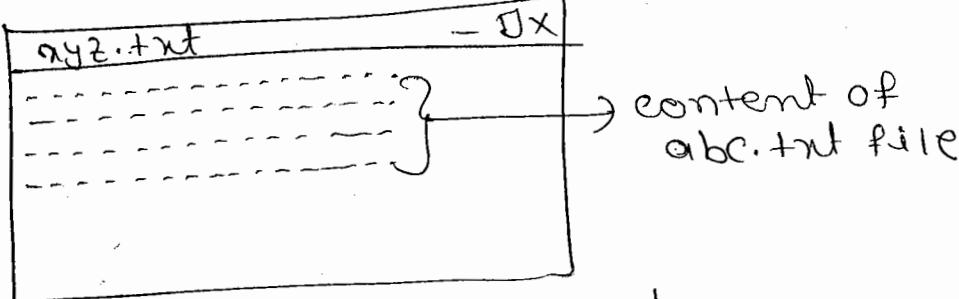
- The above example `FileInputStream` is closed automatically even exception occurs in a program.

`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 filename, boolean append)` → append mode
throws FileNotFoundException;

→ It opens a file for appending.

Methods:-

`public void write(byte[] b) 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.out.println(e);
        }
    }
}
```

Difference between System.out and System.err

System.out

- It is used to display output message
- This stream can be redirected.

class Demo

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

c:\>javac Demo.java

c:\>java Demo

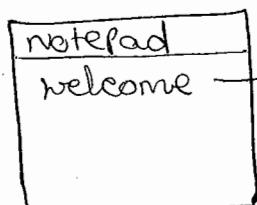
welcome

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

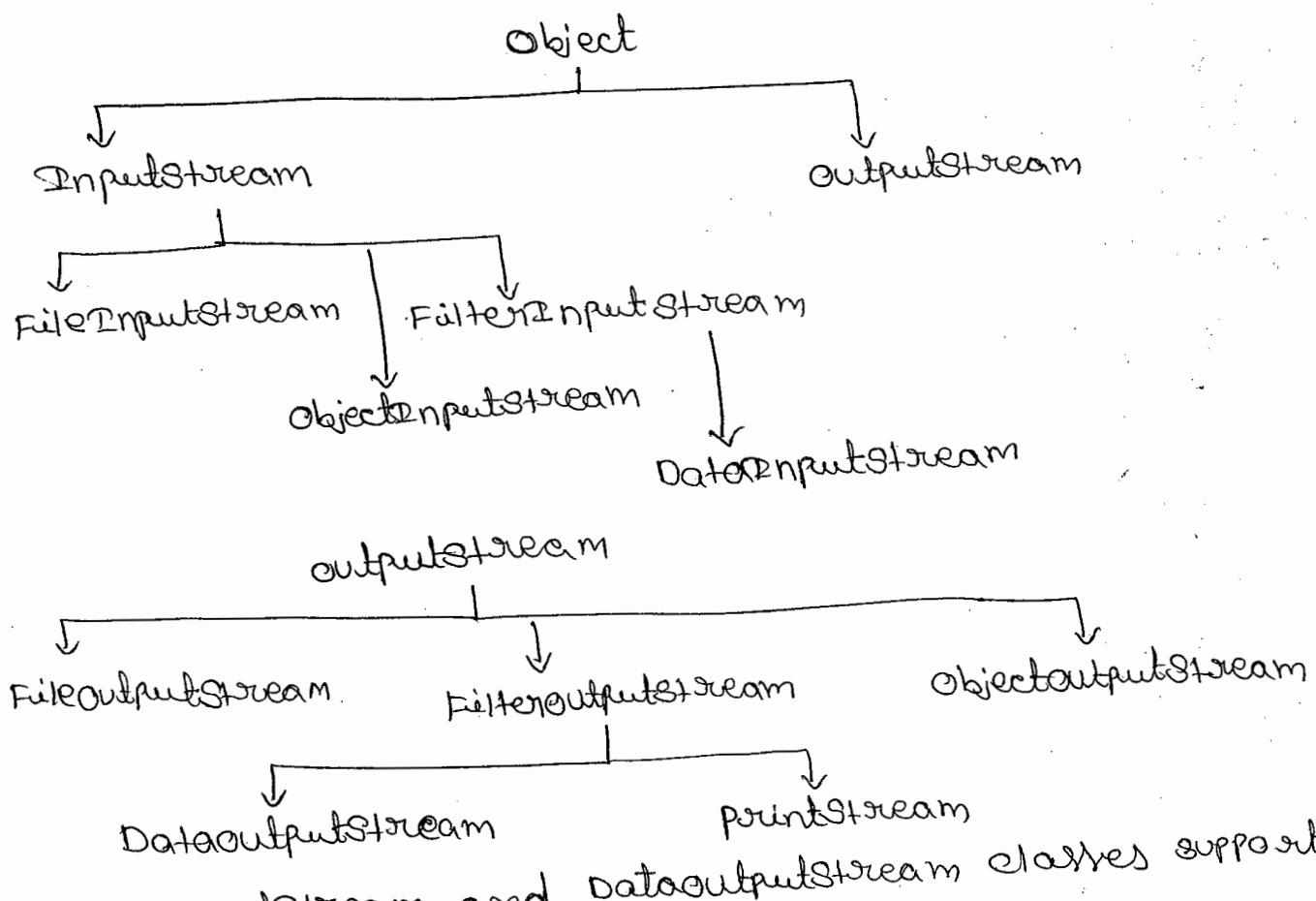
1) byte streams:-

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

2) Character streams:-

These streams handle only text

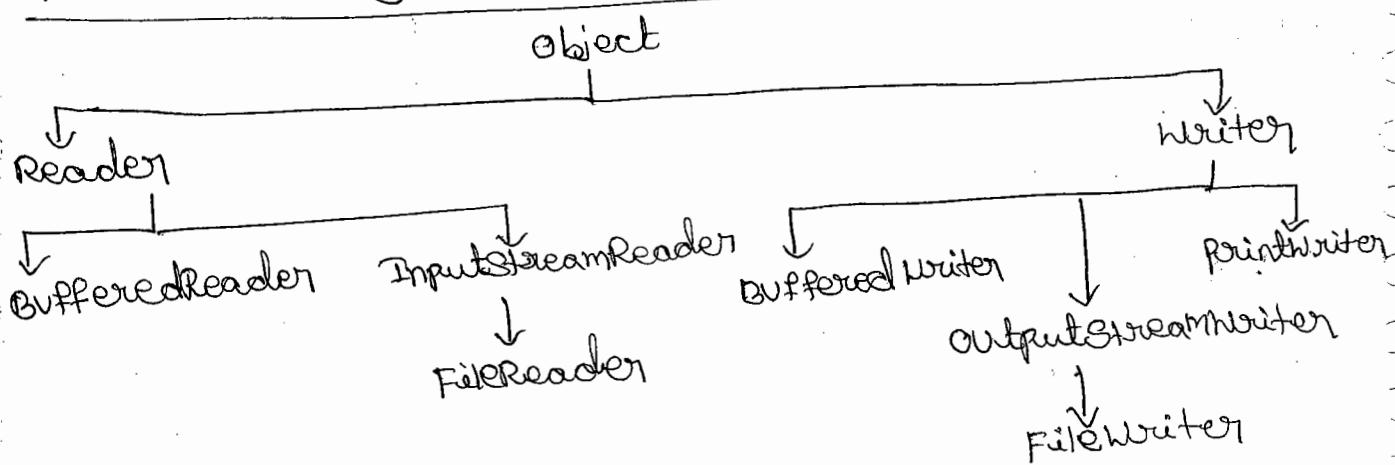
The Hierarchy of byte stream classes



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

→ ObjectOutputStream and ObjectOutputStream classes support objects. e.g (emp obj, student obj)

The Hierarchy of character stream classes



Object Streams:-

- 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

5. Class must implements `java.io.Serializable` Interface
to make Serializable object

`java.io.Serializable` Interface 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

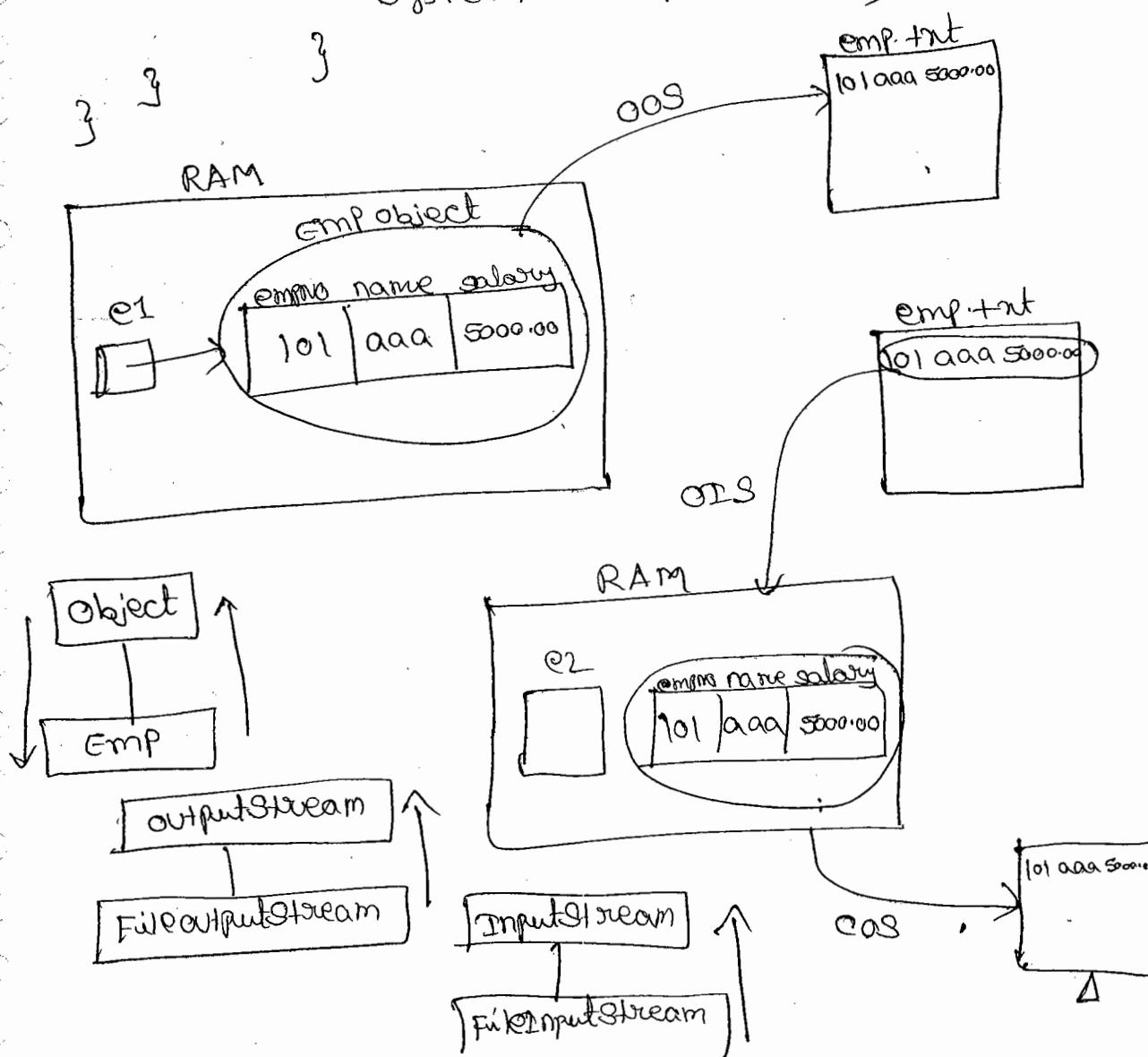
```
import java.io.*;  
class Emp implements Serializable {  
    int empno = 101;  
    String name = "aaa";  
    float salary = 5000.00f;  
  
}  
class Demo  
{  
    public static void main(String args[]){  
        try{  
            Emp e1 = new Emp();  
            FileOutputStream fos = new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

```

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

```



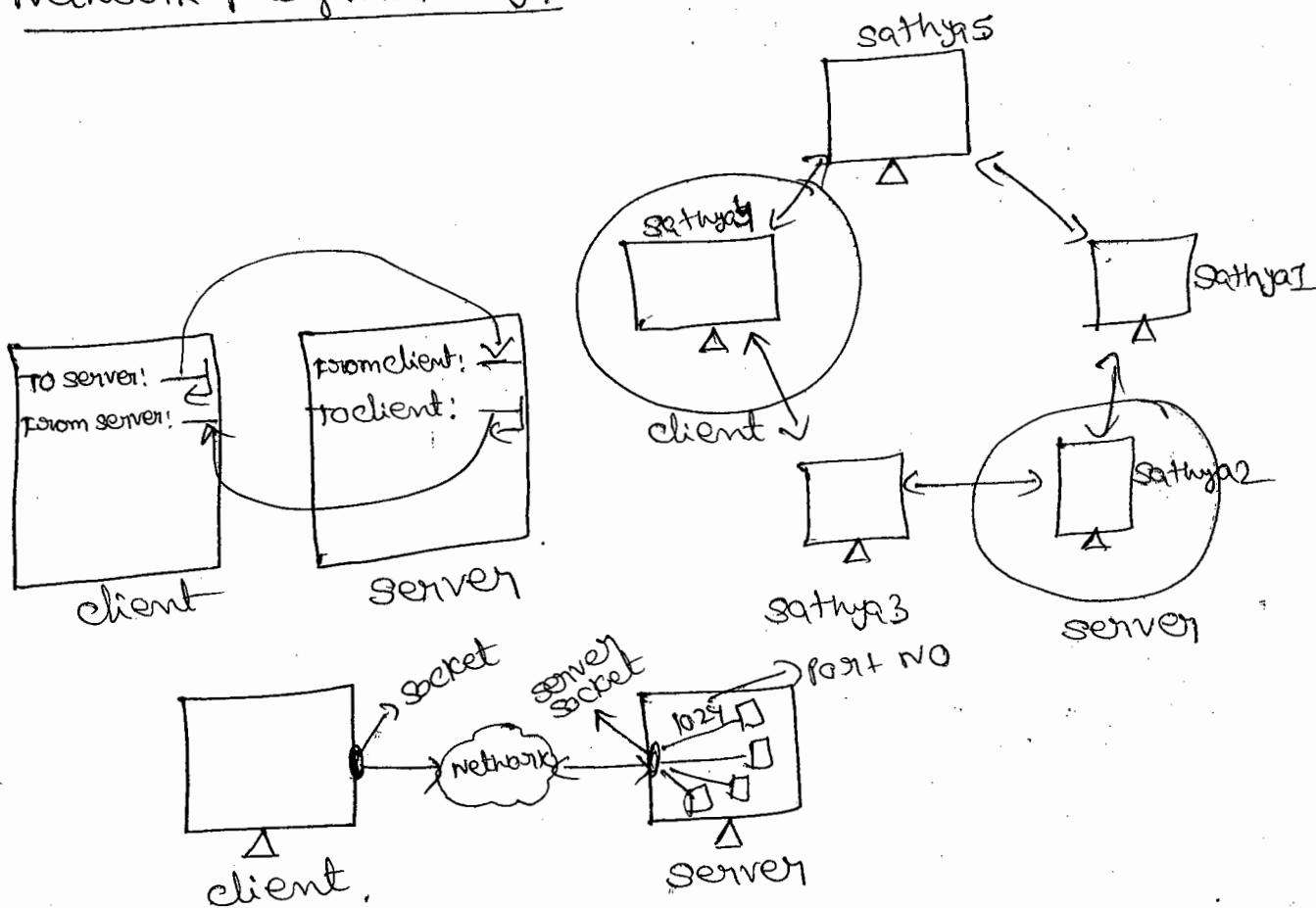
transient keyword! - (It is used to prevent serialization)

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.
 ↑ → port no.

public Socket(String, int) throws

UnknownHostException, IOException.

→ It is used to create a socket at client's 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:-

```
C import java.io.*;
C import java.net.*;
C class Client
C {
C     public static void main(String args[])
C     {
C         String add = args[0];
C         int port = Integer.parseInt(args[1]);
C         Socket s = new Socket(add, port); → creates socket object
C         OutputStream os = s.getOutputStream(); → creates output stream for sending information
C         InputStream is = s.getInputStream(); receiving
C         byte b1[] = new byte[100];
C         byte b2[] = new byte[100];
C         while(true)
C         {
C             System.out.print("To server : "); → displaying
C             System.in.read(b1); → reads the data from keyboard
C             os.write(b1); → send the data to server
C             is.read(b2); → receiving data from server
C             String s1 = new String(b2); → converting byteArray to string
C             String s2 = s1.trim(); removing extra spaces
C             System.out.println("From server : " + s2);
C         }
C     catch(Exception e)
C     {
C         System.out.println(e);
C     }
C }
```

↑ command line arguments received by main method

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

class Server
{
    public static void main(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 = s1.trim();
                    System.out.println("From client :" + s2);
                    System.out.print("To client:");
                    System.in.read(b2);
                    os.write(b2);
                }
            }
        } catch(Exception e)
        {
            System.out.println(e);
        }
    }
}
```

O: client side

C C:\>java client localhost 2000

C To server: Hello

C From server: Hi

C To server: How r u?

C From server: Fine

first server executed

C To server:

server side

C C:\>java server 2000

C From client: Hello

C To client: Hi

C From client: How r u?

C To client: Fine

Collections framework

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

Collection:-

C a collection is an object that represents group of
C objects

Advantages of collections framework:-

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

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

Examples of data structures:-

- C 1) Stack
- C 2) Queue
- C 3) Tree
- C 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:-

These are divided into 3 sub categories

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

Q 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

1) It maintains sequences

2) It allows duplicate elements

| | | | | |
|----|----|----|---|---------|
| 0 | 1 | 2 | 3 | → index |
| 45 | 72 | 45 | | |

size: 3
capacity: 3

No. of elements: 3

capacity: 4

Set

1) It maintains sets

2) It does not allow duplicate elements

| | | | | |
|----|----|----|----|-----------|
| 0 | 1 | 2 | 3 | 4 → index |
| 70 | 55 | 65 | 90 | |

size: 4
capacity: 4

No. of elements: 4

capacity: 5

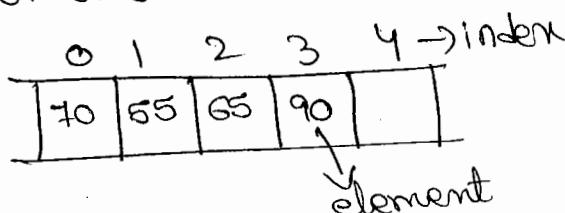
Map:-

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

Differences between Set and Map

Set

- 1) It is a one dimensional collection
- 2) It is a index based collection
- 3) It does not allow duplicate elements



size : 4
no. of elements : 4
capacity : 5.

Map

- 1) It is a two dimensional collection
- 2) It is a key based collection
- 3) It does not allow duplicate keys. Values may be duplicated.

| key | value |
|----------|----------|
| applets | corejava |
| servlets | Adv java |
| ANT | corejava |
| JSP | Adv java |

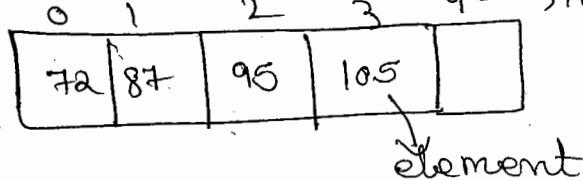
size:

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

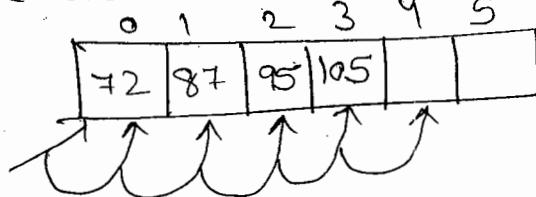
| key | value |
|----------|----------|
| applets | corejava |
| JSP | Adv.java |
| Servlets | Adv.java |
| String | corejava |

NavigationSet:-

It extends SortedSet interface and it is used to

navigate elements of a set

Ex:-



NavigationMap:-

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 rear end and deletion at front end

allows insertion at Rear end and deletion at front end

10) Deque:

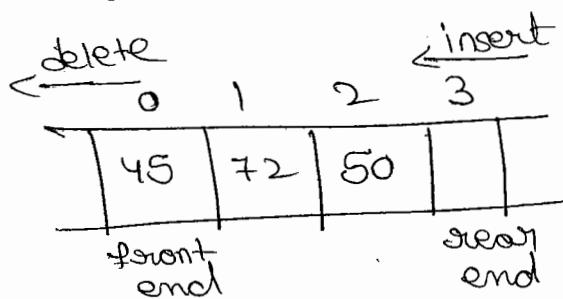
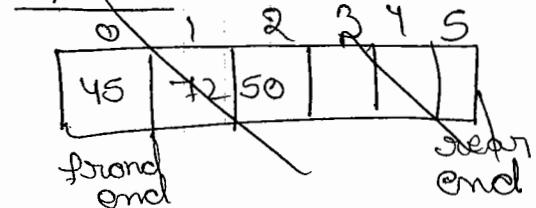
It is called as double ended queue because it allows both insertion and deletion at both the ends (front end and rear end)

Differences between queue and deque

queue

- 1) It is a first in first out list
- 2) It allows insertion at ~~rear~~ end only
- 3) It allows deletion at front end only

queue



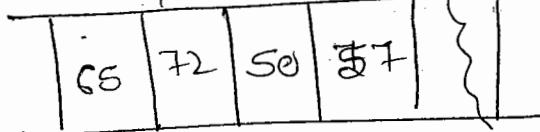
deque

- 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.

delete

insert

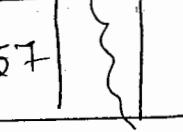
0 1 2 3



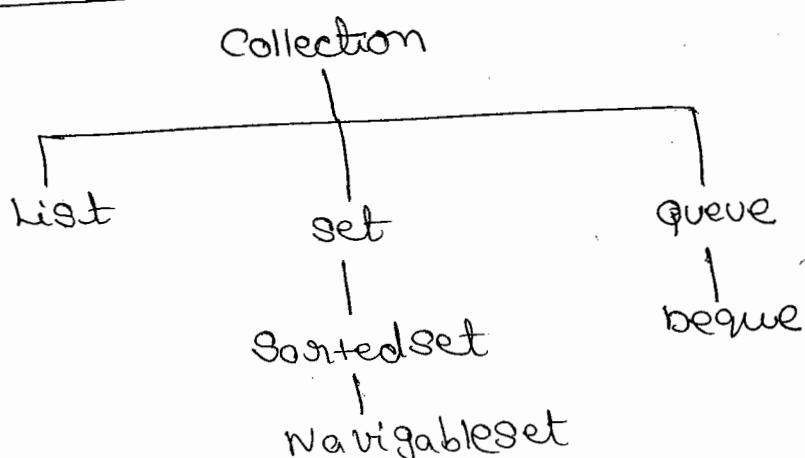
delete

insert

0 1 2 3



The Hierarchy of one dimensional collections



The Hierarchy of two dimensional collection



General purpose implementations :-

core collection Interfaces implementation classes are called general purpose 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(); index
 public Object get(int); element
 public Object set(int, Object); It is used to modify an element at specified index.
 public boolean add(Object);
 public void add(int, Object); It is used to insert an element at specified index.
 public Object remove(int);
 public boolean remove(Object); index.

program to demonstrate arraylist:-

```
import java.util.*;
```

```
class Demo
```

```
{
```

```
    public static void main(String args[])
```

```
{
```

```
    ArrayList al = new ArrayList();
```

```
    al.add(23);
```

```
    al.add(99);
```

```
    al.add(82);
```

```
    al.add(1, 72);
```

Auto boxing and upcasting

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

↳ al.toString

Implicitly

```
}
```

Output:-

[23, 72, 99, 82]

object
int → Integer
Auto boxing | upcasting

ArrayList object

al

| 0 | 1 | 2 | 3 | - | - |
|----|----|----|----|---|---|
| 23 | 99 | 72 | 82 | | |
| | 72 | 99 | 82 | | |

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

C Differences between Arrays and collection (Q1)

C Arrays and ArrayList

Arrays

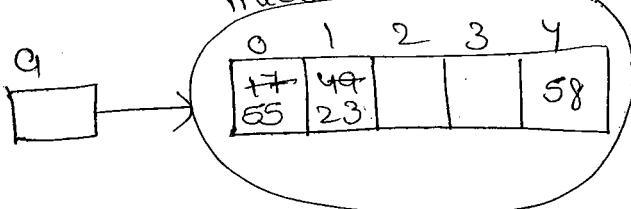
collections (ArrayList)

C 1) It is a collection of similar 1) ArrayList is a collection
C data elements class

C 2) The size of Array can 2) The size of collection can
C not grow and can not grow and can shrink
C ~~shrink~~ shrink dynamically as needed.

ArrayList:-

ArrayList `al = new ArrayList();`



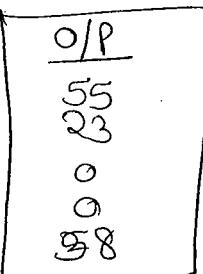
`a[0] = 17;`

`a[1] = 49;`

`a[0] = 55;`

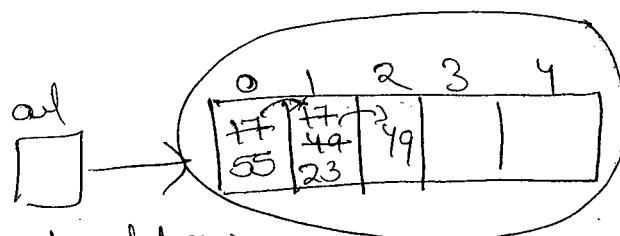
`a[4] = 58;`

`a[1] = 23;`



`for (int i=0; i<a.length; i++)`

{
 `s.o.p(a[i]);`
}



`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 :-

C It is an implementation of double linked list data
C structure.

C → 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

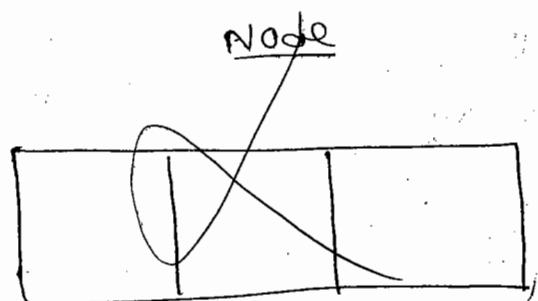
1) ~~previous node address~~

2)

1) left link field

2) Data field

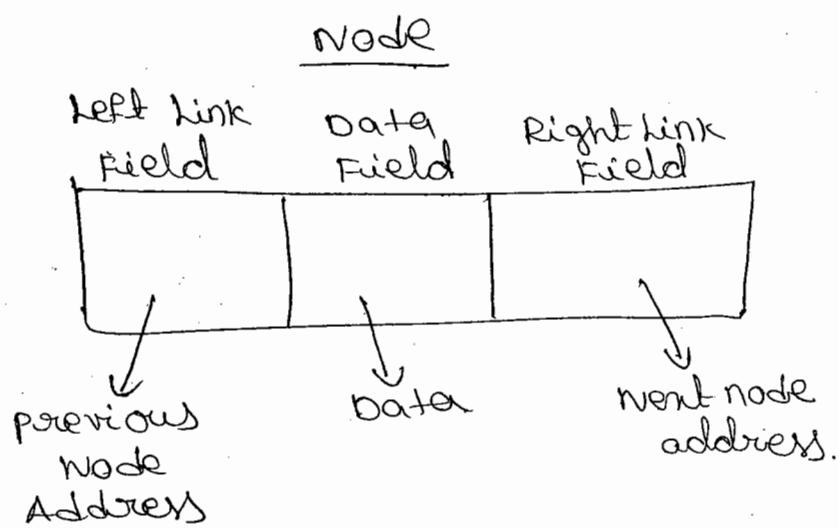
3) Right link field



→ Left link field contains previous node address

→ Data field contains data

→ Right link field contains next node address



Java.util.LinkedList:-

constructor:-

```
public LinkedList();
```

methods:-

```
public Object getFirst();
```

```
public Object getLast();
```

```
public Object removeFirst();
```

```
public Object removeLast();
```

```
public void addFirst(Object);
```

```
public void addLast(Object);
```

→ `2+` 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
```

```
{
```

```
    public static void main(String args[])
```

```
{
```

```
    LinkedList ll = new LinkedList();
```

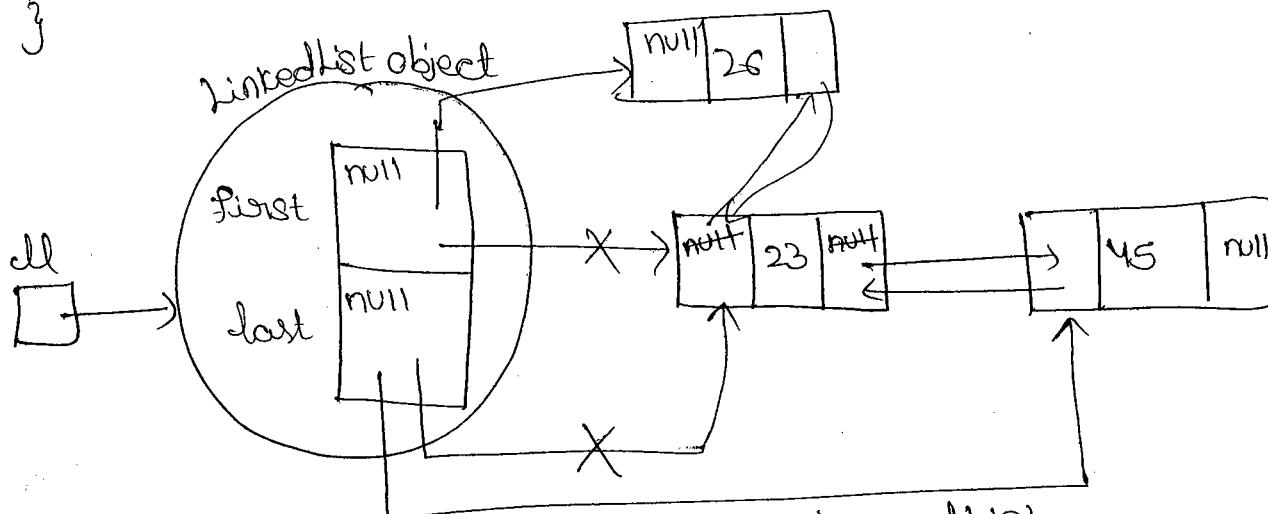
```
    ll.add(23);
```

```
    ll.add(45);
```

```
    ll.addFirst(26);
```

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

3
3



Differences b/w ArrayList and linkedlist

ArrayList

1) It is a linear list data structure

2) It occupies less memory

3) In ArrayList insertion and deletion ^{operation} require shuffling of data

linkedlist

1) It is a double linked list data structure

2) It occupies more memory because data stored in nodes

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
```

```
{ public String args[])
```

```
{ HashSet hs = new HashSet();
```

```
hs.add(23);
```

```
hs.add(45);
```

```
hs.add(99);
```

```
hs.add(54);
```

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

```
}
```

```
}
```

O/p:- can not be predicted.

LinkedHashSet:-

It is an implementation of Hashing technique with Linked representation

```
import java.util.*;
```

```
class Demo
```

```
{ public String args[])
```

```
{ LinkedHashSet lhs = new LinkedHashSet();
```

```
lhs.add(23);
```

```
lhs.add(45);
```

```
lhs.add(99);
```

```
lhs.add(54);
```

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

```
}
```

```
}
```

O/p : [23 45 99 54]

TreeSet:-

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

Binary search tree:-

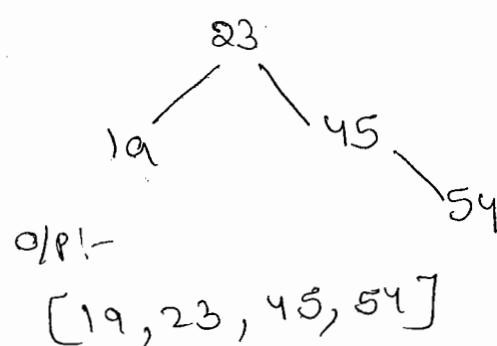
a binary tree is called as Binary search tree if it follows the following rules.

→ 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
{
    public static void main(String args[])
    {
        TreeSet ts = new TreeSet();
        ts.add(23);
        ts.add(45);
        ts.add(19);
        ts.add(54);
        System.out.println(ts);
    }
}
```



Q15

Differences b/w HashSet, LinkedHashSet and TreeSet

HashSet

i) It is an implementation of Hashing technique with array representation.

ii) It occupies less memory

iii) HashSet is unordered set and unsorted set

LinkedHashSet

i) It is an implementation of Hashing technique with linked representation.

ii) It occupies more memory because data stored in nodes

iii) LinkedHashSet is ordered set

TreeSet

i) It is an implementation of binary search technique with linked representation.

ii) It occupies more memory because data stored in nodes.

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");
        System.out.println(hm);
    }
}
  
```

O/p:- can not be predicted.

Differences b/w HashSet and HashMap

HashSet

- 1) It is a one dimensional collection
- 2) It is a index based collection
- 3) HashSet does not allow duplicate elements

HashMap

- 1) It is a two dimensional collection
- 2) It is a key based collection
- 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
```

```
{ public static void main(String args[])
```

```
{  
    LinkedHashMap lhm = new LinkedHashMap();  
    lhm.put("servlets", "Adv.Java");  
    lhm.put("applets", "Core.Java");  
    lhm.put("swing", "CoreJava");  
    lhm.put("JSP", "Adv.Java");  
    System.out.println(lhm);  
}
```

```
}
```

Output:- { servlets = Adv.Java, applets = Core Java,
swing = CoreJava, 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 TreeMap:-

```
import java.util.*;
class Demo
{
    public static void main(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");
        System.out.println(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

{ public static void main(String args[])

{ PriorityQueue pq = new PriorityQueue();

pq.offer("javase"); Object but we are
packing string at

pq.offer("javase"); valid upcasting

pq.offer("javame");

pq.offer("javafx");

System.out.println(pq); int we pass i.e
Autoboxing.

}

ArrayDeque-

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

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
```

```
{ public static void main(String args[])
```

```
{
```

```
    ArrayDeque ad = new ArrayDeque();
```

```
    ad.add(23);
```

```
    ad.addFirst(43);
```

O/P:- [88, 43, 23, 21, 92]

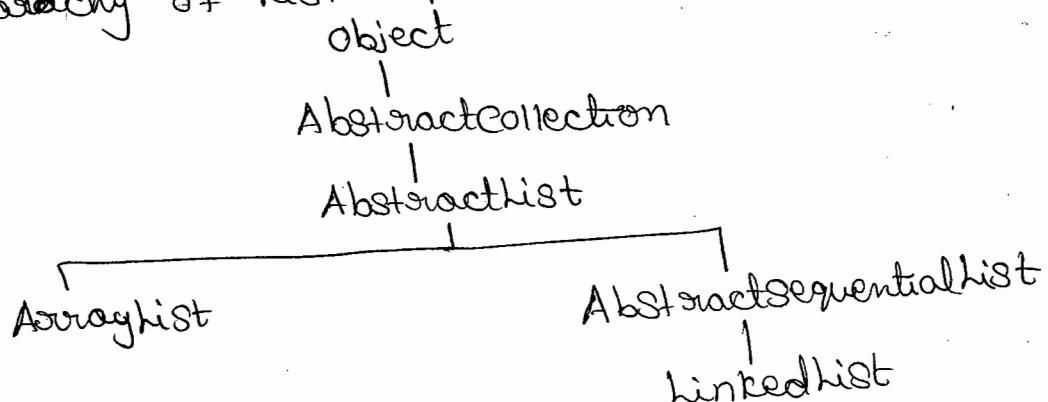
```
    ad.addLast(21);
```

```
    ad.addFirst(88);
```

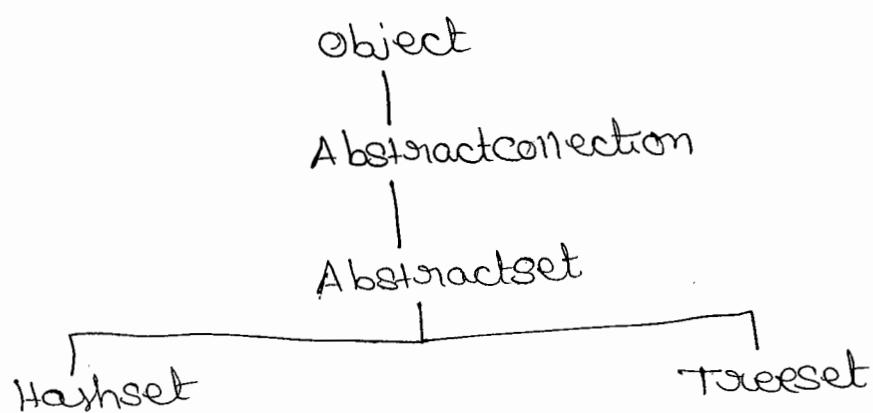
```
    ad.addLast(92);
```

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

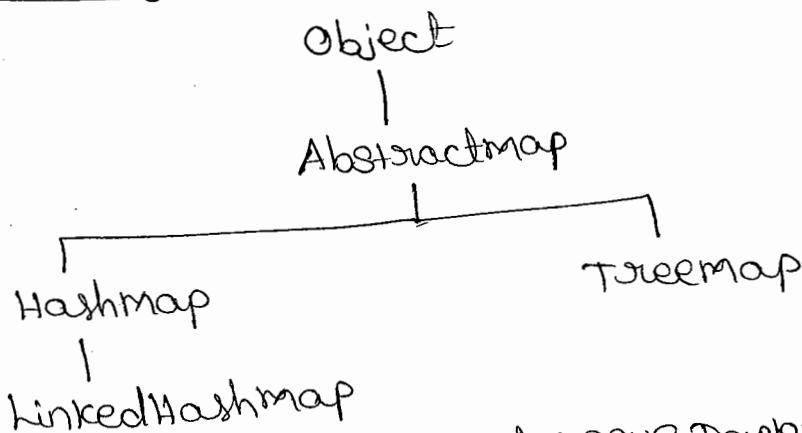
}
The Hierarchy of List implementation classes:-



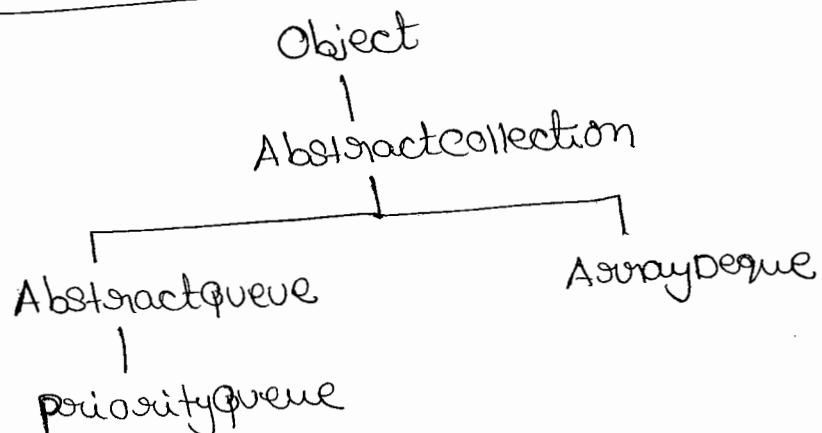
The Hierarchy of set implementation classes:-



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
```

```
{ public static void main(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();  $\Rightarrow$  downcasting &
    System.out.println(x+5);  $\Rightarrow$  Auto unboxing.
}

```



List Iterators:-

List Iterator :-
It is used to iterate elements of list implementation classes only (ArrayList & LinkedList)

`java.util.ListIterator`

methods:-

methods:-

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

↳ current element

→ It is used to remove current element

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

→ It is used to modify current element

→ It is used to do
public abstract void add(Object)

public abstract void add(Obj)

→ 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
{
    public static void main(String args[])
    {
        ArrayList al = new ArrayList();
        al.add(45);
        al.add(72);
        al.add(50);
        al.add(23);
        System.out.println(al);
        ListIterator li = al.listIterator(al.size());
        while(li.hasPrevious())
        {
            System.out.println(li.previous());
        }
    }
}
```

↳ we are passing
one more size
i.e 4 so
cursor moves to
last

3 3
o/p: [45, 72, 50, 23]
 ↓↓↓↓
 23
 50
 72
 45

al.listIterator(al.size());

li.previous()

Differences b/w Iterator and ListIterator

Iterator

ListIterator

- | | |
|---|--|
| <ul style="list-style-type: none">1) It is used to iterate elements of any collection2) Iterator allows to iterate elements only forward direction3) Iterator supports only remove operation while iterating elements | <ul style="list-style-type: none">1) It is used to iterate elements of ArrayList and LinkedList only2) ListIterator allows to iterate elements in both forward and backward directions.3) ListIterator allows supports add, remove and set operations while iterating elements |
|---|--|

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[]); element to
public static void sort(Object[]); ↑ search
public static int binarySearch(int[], int);
public static boolean equals(int[], int[]);
public static String toString(int[]);
→ It returns string representation of array elements

Program to demonstrate Arrays class

```
import java.util.*;  
class Demo  
{  
    public static void main(String args[])  
    {  
        int a[] = {23, 45, 81, 40, 34};  
        String s1 = Arrays.toString(a);  
        System.out.println("Before sorting -----");  
        System.out.println(s1);  
        Arrays.sort(a);  
        String s2 = Arrays.toString(a);  
        System.out.println("After sorting -----");  
        System.out.println(s2);  
    }  
}
```

4) collections:- class

It contains several methods that are used to perform on collections

java.util.Collections

methods:-

```
public static void sort(List<? extends Comparable> list);  
public static int binarySearch(List<? extends Comparable> list, Object value);  
public static void reverse(List<? extends Comparable> list);
```

element to search ↑

Q) program to demonstrate Collection class :-

```
C import java.util.*;  
C  
C class demo  
C {  
C     public static void main(String args[]){  
C         ArrayList al = new ArrayList();  
C         al.add(23);  
C         al.add(99);  
C         al.add(25);  
C         al.add(65);  
C         System.out.println(al);  
C         Collections.reverse(al);  
C         System.out.println(al);  
C     }  
C }  
C  
C O/P: [23 99 25 65]  
C Collections.sort(al);  
C System.out.println(al);  
C Collections.reverse(al);  
C System.out.println(al);  
C }
```

Q) 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 n = s.nextInt(); → instance method
        System.out.println(n); 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
- 2) It does not allow any other operation while Iterating elements
- 3) vector :- (class)

- 1) It is a collections framework collection
- 2) It allows remove operation while Iterating elements

It is similar to Arraylist class

Differences between vector and Arraylist

vector

ArrayList

- 1) It is a legacy collection
- 2) methods of vector class are synchronized
- 3) Stringtokenizer:-

- 1) It is a collection framework collection
- 2) methods of ArrayList class are not synchronized

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 collection 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

- 1) It is a legacy collection
- 2) Hashtable doesn't allow null keys & null values
- 3) methods of Hashtable are synchronized

HashMap

- 1) It is a collections framework collection.
- 2) HashMap allows one null key and many null values
- 3) methods of HashMap are not synchronized.

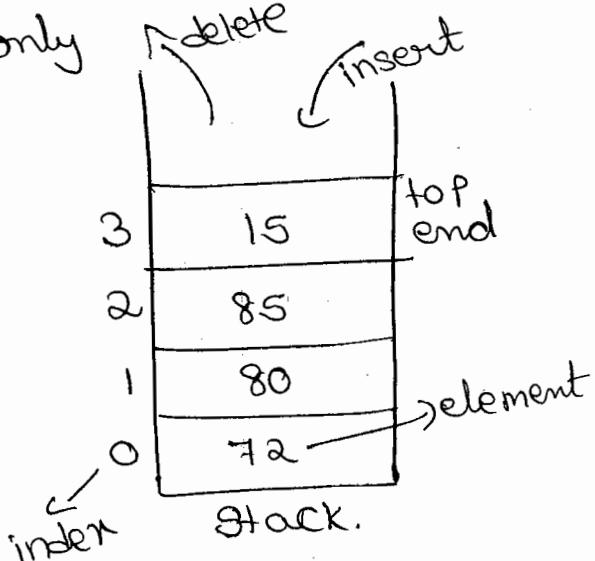
6) Stack (Class):-

It is called as Last in First out list

Differences between Stack and Queue

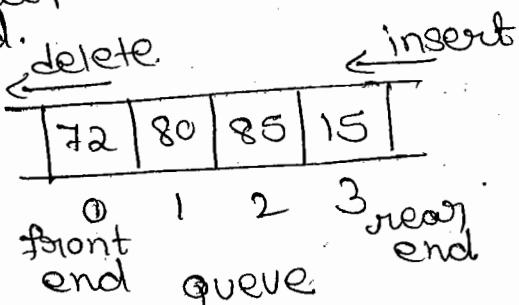
Stack

- 1) It is a last in first out list
- 2) Stack is a legacy collection
- 3) Stack allows both insertion and deletion at top only



Queue

- 1) It is a first in first out list
- 2) Queue is a collections framework collection.
- 3) Queue allows insertion at rear and deletion at front end.



C java.util.Stack

C constructor:-

C public Stack();

C methods:-

C public Object push(Object);

C → It is used to insert an element into a Stack

C public synchronized Object pop();

C → It is used to remove an element from stack

C Both are instance methods we have to create object

C for Stack class

C import java.util.*;

C class Demo

C {

C public static void main(String args[])

C {

C Stack s = new Stack();

C s.push(72);

C s.push(80);

C s.push(85);

C s.push(15);

C System.out.println(s);

C System.out.println(s.pop());

C System.out.println(s);

C }

C }

C Output: [72, 80, 85, 15]

C 15

C [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

O/P: can not predict

{

 Random r = new Random();

 System.out.println(r.nextBoolean());

 System.out.println(r.nextInt(100));

}

}

8) Date :-

It is used to get system date and time

Java.util.Date :-

constructor:-

public Date();

C Methods:-

```
C     public int getYear();
C     public int getMonth();
C     public int getDate();
C     public int getHours();
C     public int getMinutes();
C     public int getSeconds();

C import java.util.*;
C class Demo
C {
C     psvm(String args[])
C     {
C         Date d = new Date();
C         soplm(d.getHours() + ":" + d.getMinutes() + ":" +
C               d.getSeconds());
C     }
C }
```

Generics

```
C import java.util.*;
C class Demo
C {
C     psvm(String args[])
C     {
C         ArrayList al = new ArrayList();
C         al.add(92);
C         al.add(23);
C         al.add(42);
C         al.add(82);
C         al.add("abc");
C         soplm(al);
C     }
C }
```

Op:- [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

in 2004

Generics:-

This feature introduced in JDK 1.5 version 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  
{  
    public static void main(String args[]){  
        ArrayList<Integer> al = new ArrayList<Integer>();  
        al.add(92);  
        al.add(23);  
        al.add(42);  
        al.add(82);  
        //al.add("abc"); => Error  
        System.out.println(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 type-safe 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

→ type-safe

Advantage of above program:-

Type-safe 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>(); ✓  
ll1.set(23); ✓  
ll1.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> ll = new Linearlist<Integer>();
        ll.set(23);
        int y = ll.get();
        System.out.println(y);
        Linearlist<String> ll2 = new Linearlist<String>();
        ll2.set("abc");
        String s = ll2.get();
        System.out.println(s);
    }
}
```

Multithreading

Q&P

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.

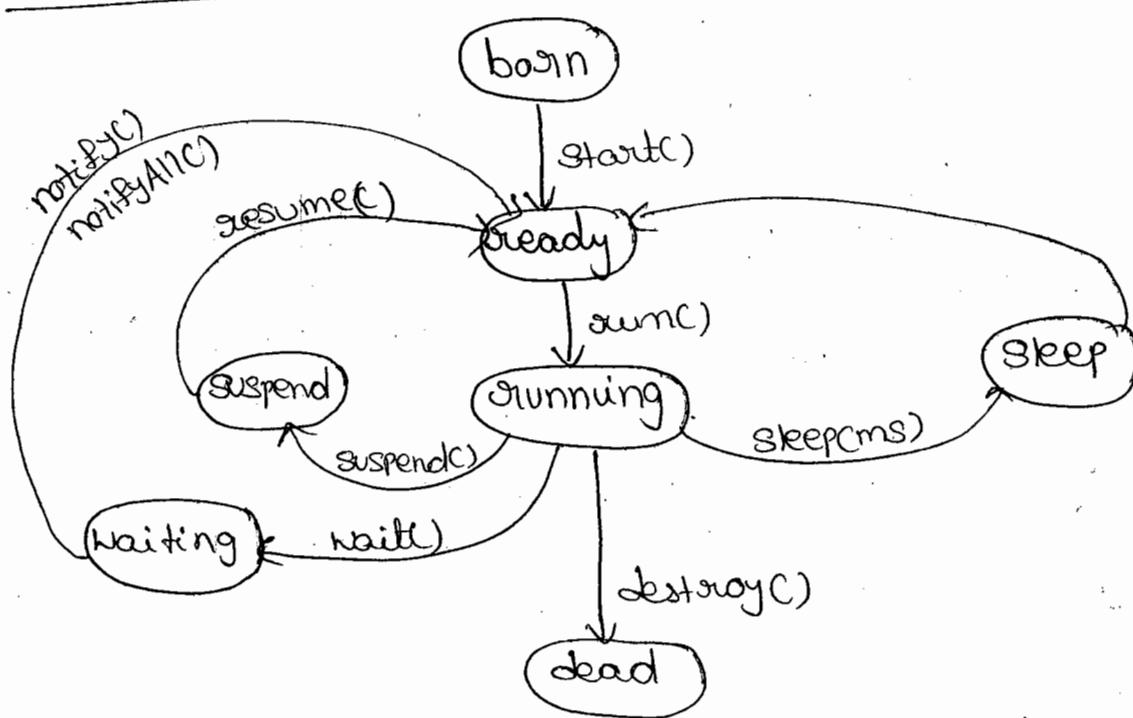
→ every program contains atleast one thread that is main thread.

→ 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 be born 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 as 2 constructors
1) extends
2) Implementing

Methods:-

public static native Thread ^{RT} currentThread();

public static native void sleep(long) ^{milliseconds}
throws InterruptedException; if (int) 4secs

public synchronized void start();

```
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
```

```
{  
    public static void main(String args[]){  
        Thread t = Thread.currentThread();  
        System.out.println(t.getName());  
        System.out.println(t.getPriority());  
        t.setName("myThread");  
        t.setPriority(Thread.MAX_PRIORITY);  
        System.out.println(t.getName());  
        System.out.println(t.getPriority());  
    }  
}
```

O/P:- main

5

myThread

10

In this program by default main thread is the thread

Program to demonstrate sleep method:-

Class Demo

```
public class Demo {
```

```
    public static void main(String args[]) {
```

```
        try {
```

```
            for (int i=1; i<=10; i++) {
```

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

sleep method throws exception that why

```
Thread.sleep(1000);
```

try & catch.

```
            catch (Exception e) {
```

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

```
            }
```

```
        }
```

~~Program 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();

child Thread.

open window

main
thread

}

wait()
notify()
notifyAll()

java.lang.Object

class

java.lang.Runnable

Abstract
method
run()

Interface

java.lang.Thread

class

sleep()
suspend()
resume()
start()

run() → null body
destroy()

```
C class Demo extends Thread
C {
C     public void run()
C     {
C         try {
C             for (int i=1; i<=10; i++)
C             {
C                 System.out.println("child thread: " + i);
C                 Thread.sleep(1000);
C             }
C         } catch (Exception e)
C         {
C             System.out.println(e);
C         }
C     }
C     public static void main(String args[])
C     {
C         try {
C             Demo ob = new Demo();
C             ob.start();
C             for (int i=1; i<=10; i++)
C             {
C                 System.out.println("main thread: " + i);
C                 Thread.sleep(1000);
C             }
C         } catch (Exception e)
C         {
C             System.out.println(e);
C         }
C     }
C }
```

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.out.println(e);

}

}

psvm (String args[])

{ try {

Demo ob = new Demo();

ob.start();

for (int i=1; i<=10; i++)

{

System.out.println("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.out.println(e);

} } }

```

C class Demo extends Thread
C {
C     public void run()
C     {
C         for(int i=1; i<=10; i++)
C         {
C             System.out.println("child thread");
C         }
C     }
C     public static void main(String args[])
C     {
C         Demo ob = new Demo();
C         ob.start();
C         System.out.println("main thread");
C     }
C }
C o/p:- can not be predicted

```

Program to demonstrate join() method:-

```

C class Demo extends Thread
C {
C     public void run()
C     {
C         for(int i=1; i<=10; i++)
C         {
C             System.out.println("child thread");
C         }
C     }
C     public static void main(String args[])
C     {
C         try{
C             Demo ob = new Demo();
C             ob.start();
C             ob.join(); // This statement puts the current
C             // thread into waiting state until
C             // child thread joins.
C             System.out.println("main thread");
C         } catch(InterruptedException e)
C         {
C             System.out.println(e);
C         }
C     }
C }

```

join() method
is in Thread
class we are
overriding
that method
in ~~Object~~ by
using Demo
class

(3) 4/15

13/41 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

2

→ public void runnC)

३

→ `PSVM(string args[])`

```
Runnable sr = new Demo();
```

```
Thread t = new Thread(y);
```

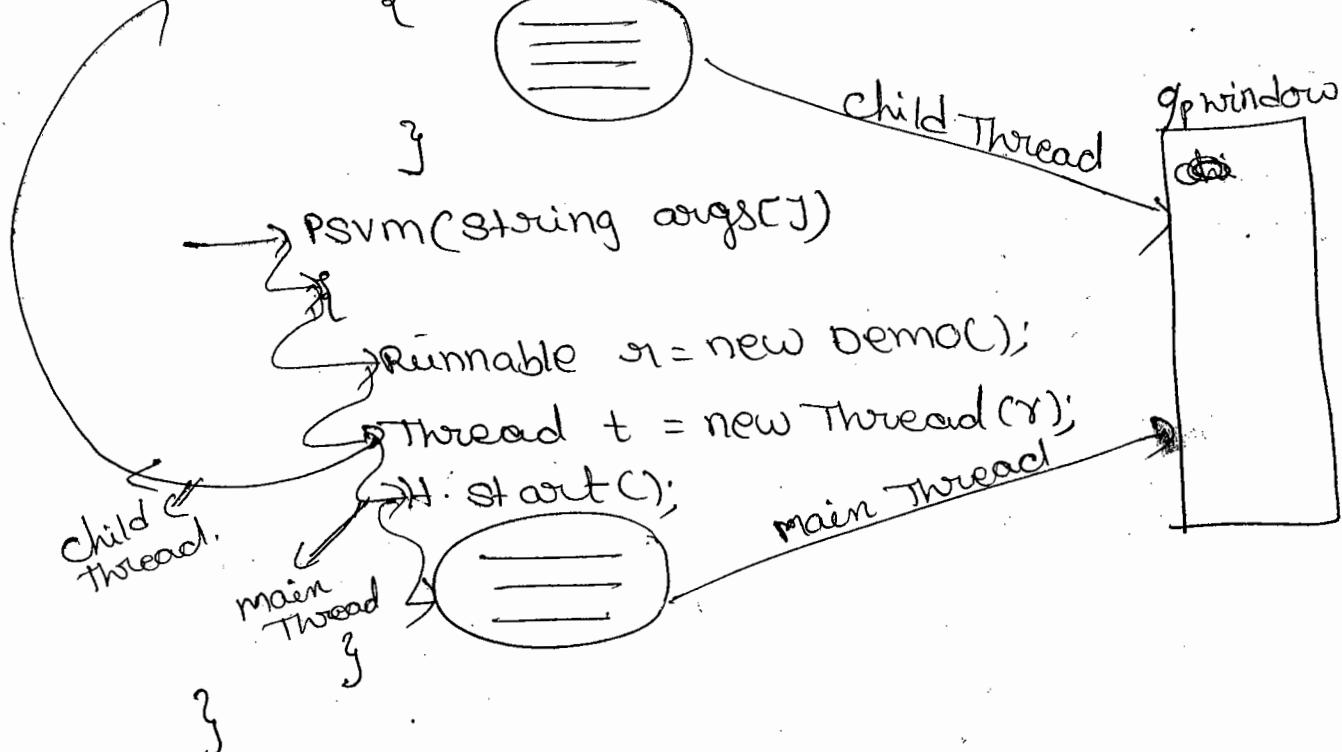
~~H.start();~~

Main

Child
thread

main
Thread

3



```

C class Demo implements Runnable
C {
C     public void run()
C     {
C         for(int i=1; i<=10000; i++)
C         {
C             System.out.println("child Thread");
C         }
C     }
C     public static void main(String args[])
C     {
C         Runnable r = new Demo();
C         Thread t = new Thread(r);
C         for(int i=1; i<=10000; i++)
C         {
C             System.out.println("main Thread");
C         }
C     }
C }

```

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(args1, args2, --)
{
    =====
}

```

2) Synchronizing a block of code:-

Syntax:-

```
ReturnType MethodName(Arg1, Arg2, ...)
```

```
    {  
        ======  
        synchronized(Object)  
        {  
            ======  
            {  
                ======  
    }
```

Program to demonstrate synchronization:-

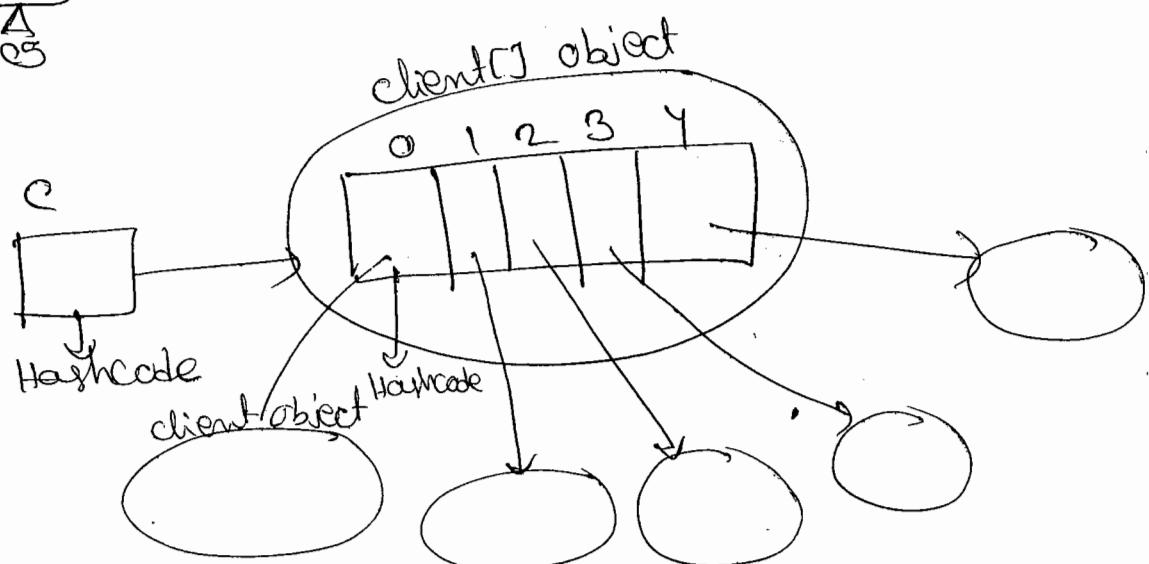
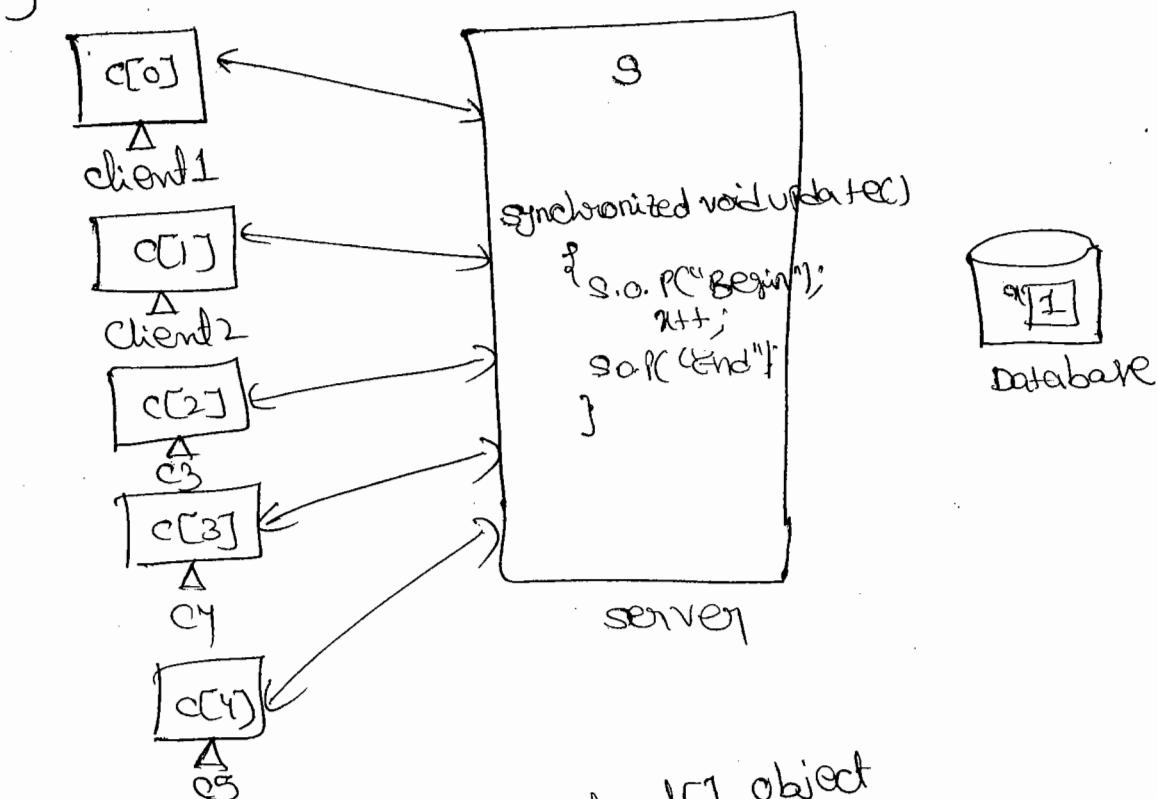
```
class Server  
{  
    int x=1;  
    synchronized void update()  
    {  
        System.out.println("Begin");  
        x++;  
        System.out.println("End");  
    }  
}
```

```
class Client extends Thread
```

```
{  
    Server s;  
    Client(Server s)  
    {  
        this.s=s;  
    }  
    public void run()  
    {  
        for(int i=1; i<=10; i++)  
        {  
            s.update();  
        }  
    }  
}
```

C class demo

```
C {
C     public static void main(String args[])
C     {
C         server s = new server();
C         client c[] = new client[5];
C         for(int i=0; i<c.length; i++)
C         {
C             c[i] = new client(s);
C             c[i].start();
C         }
C     }
C }
```



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();
    }
  }
}
```

```
  System.out.println(ob.total);
  catch(InterruptedException e)
  {
    System.out.println(e);
  }
}
```

NOTE:-

Both wait and notify methods must be used in synchronized area otherwise IllegalMonitorStateException will be thrown.

14/4/15

Inner classes

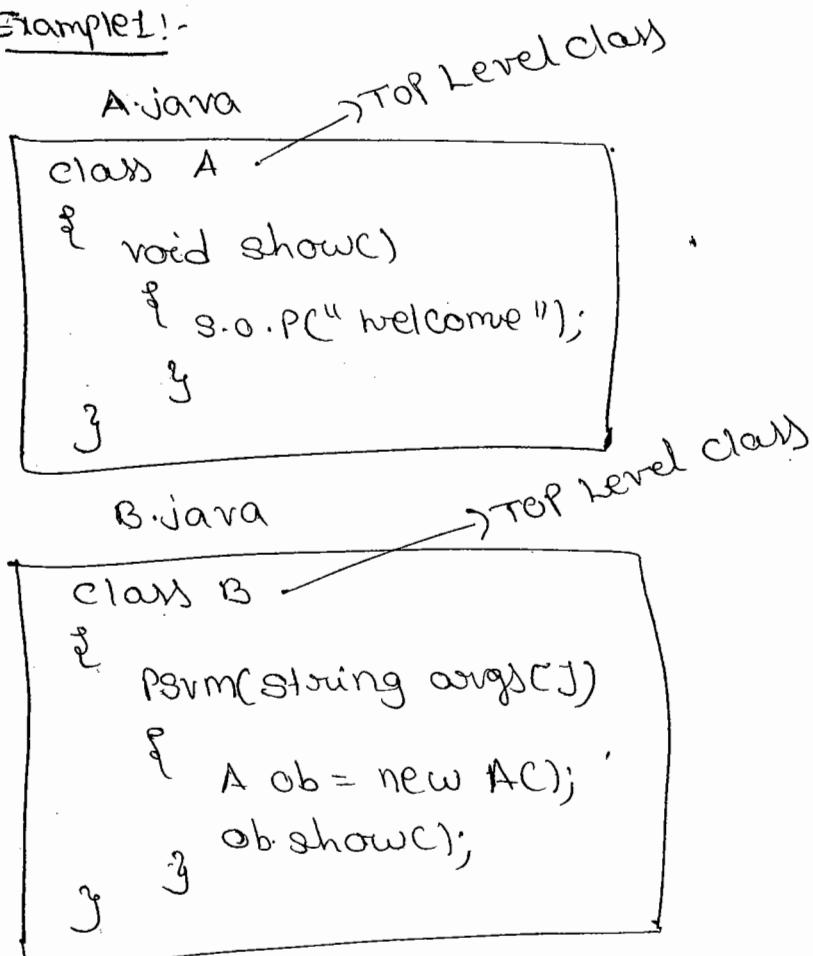
→ In java classes are divided into two categories

- 1) TOP Level classes
- 2) Inner classes

1) TOP Level class:-

a class that is defined under package is known as TOP Level class.

Example!:-



Example:-

A.java

```
package abc;  
public class A  
{  
    public void show()  
    {  
        System.out.println("welcome");  
    }  
}
```

TOP LEVEL
CLASS

B.java

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

TOP LEVEL
CLASS

C.java

```
class C  
{  
    public static void main(String args[])  
    {  
        abc.A ob = new abc.A();  
        ob.show();  
    }  
}
```

TOP LEVEL
CLASS

new A().show.

or

```
A a = new A();  
a.new B().show();
```

(or)

```
new A().new B().show();
```

Q 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()
 {
 System.out.println("welcome");
 }

class demo → top level class

{
 public static void main(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

Example:-

class A

{ static class B → nested top level class (Inner class)

{ void showC()

{ System.out.println("welcome");

}

}

}

class demo

{ public static void main(String args[])

{ A.B ob = new A.B();

ob.showC();

}

}

Example2:-

class A

{ static class B

{ static void showC()

{ System.out.println("welcome");

}

}

}

class demo

{ public static void main(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 point()

{ class B → Local class (inner class)

{ void show()

{ System.out.println("welcome");

}; };

B ob = new B();

ob.show();

} ;

Class demo

{ public static void main(String args[])

{ A ob = new A();

ob.point();

} ;

} ;

4) Anonymous Class:-

It is a one type of inner class which has no name.

It is always subclass of a class (or interface)

Example:-

interface A

```
{     void show();  
}
```

Class Demo

```
{     public static void main(String args[]){
```

```
        A ob = new A(){  
            public void show()  
            { s.o.p("welcome");  
            }  
        ob.show();  
    }  
}
```

```
c:\$javac demo.java
```

A.class

demo.class

demo\$1.class

A → Interface
show() Abstract method

- → Anonymous class
(It is an inner
class of demo class)

show() → concrete method

This program contains 3
classes A class, demo class,
Anonymous class (no name)
↳ demo\$1.class.

class — implements A
{
 public void show()
 { s.o.p("welcome");
 }
}
A ob = new — ();
ob.show();

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:- Red Green Blue

```
public Color(int, int, int);
```

→ ↓ ↓
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 embedded 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)  
    {  
        Random r = new Random();  
        for(int i=1; i<=100; i++)  
        {  
            int x = r.nextInt(500);  
            int y = r.nextInt(500);  
            int size = r.nextInt(50);  
            g.fillOval(x, y, size, size);  
        }  
    }  
}
```

Applets

Java supports two types of programs

- 1) application
- 2) Applets

1) Application:-

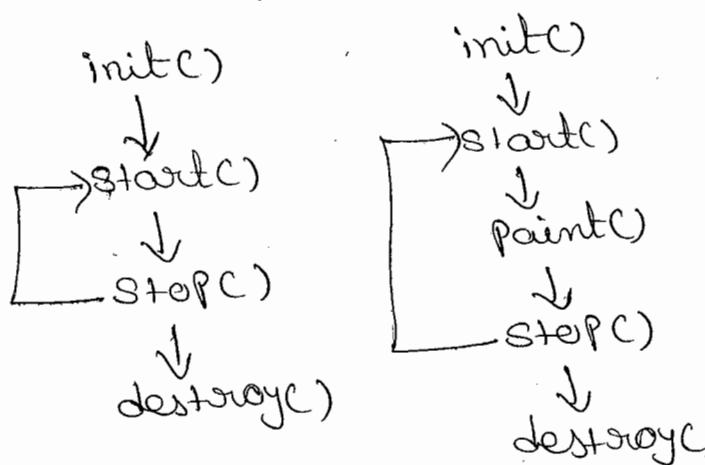
An application is a Java program which runs under operating system.

2) Applet:-

An Applet is a Java program which runs under web browser.

Applets do not have main method because applets run under ~~new~~ browser.

Applet has a life cycle to run in web browser.



open:-

Deactivation:

Activation:

close

init()

stop()

start()

stop()

↓
start()

↓
paint()

↓
paint()

↓
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:-

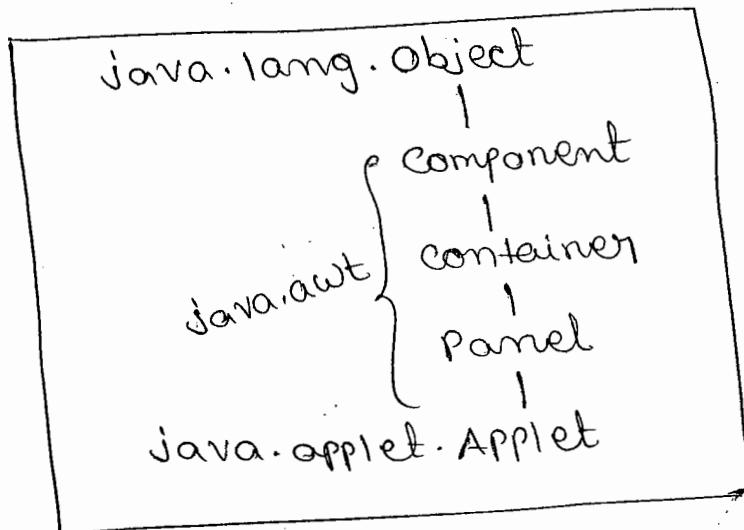
methods:-

```
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:-

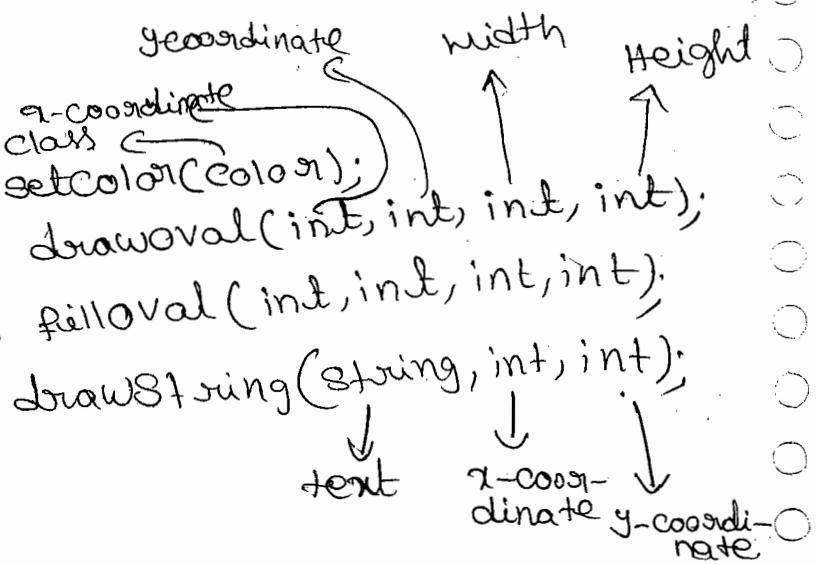
```
public abstract void setcolor(Color);
public abstract void drawoval(int, int, int, int);
public abstract void filloval(int, int, int, int);
public abstract void drawstring(String, int, int);
```

"

"

"

"



```
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.out.println(e);
}
}
}

→ Applets can also be executed by using applet viewer
appletviewer at the command prompt
c:\>appletviewer demo.html
```

swing

- Java Swing is a set of classes and interfaces that are used to develop graphical user interfaces.

AWT vs swing

AWT

(Abstract window Toolkit)

- C 1) AWT components are heavy weight components
 - C 2) AWT look and feel is different in different platforms.
 - C 3) AWT look and feel can not be changed by programmer
 - C 4) AWT components can not display ~~inter~~ images
 - C 5) AWT components developed in C, C++ languages also

gwing

- 1) swing components are light weight components
 - 2) swing look and feel is same in all platforms
 - 3) swing look and feel can be changed by programmer.
 - 4) swing components can display images.
 - 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.

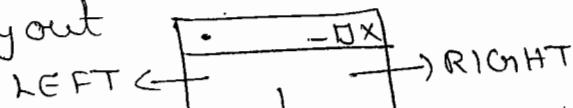
- | | |
|-------------------|--|
| 1) JButton | 6) JCheckBox <input checked="" type="checkbox"/> — |
| 2) JLabel | 7) JRadioButton <input type="radio"/> — |
| 3) JTextField | 8) JToggleButton |
| 4) JPasswordField | 9) JComboBox |
| 5) JTextArea | 10) JList |

3) Layout Managers:-

There are AWT classes.

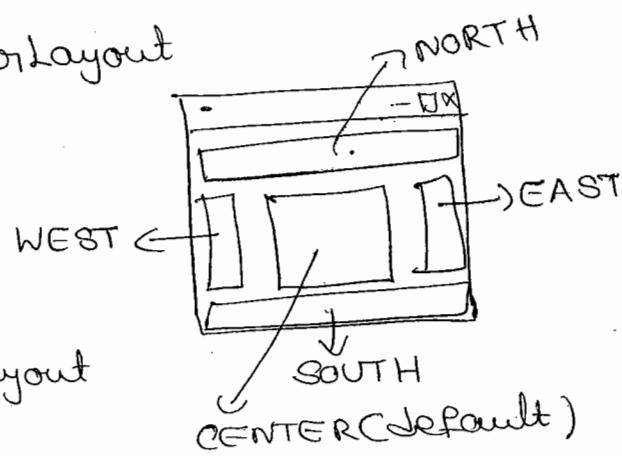
They are used to arrange components on container

1) java.awt.FlowLayout

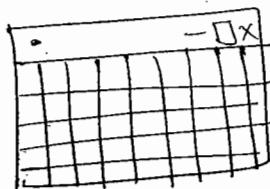


CENTER (Default)

2) java.awt.BorderLayout



3) java.awt.GridLayout



4) menu components:-

They are used to create menus

1) JMenuBar

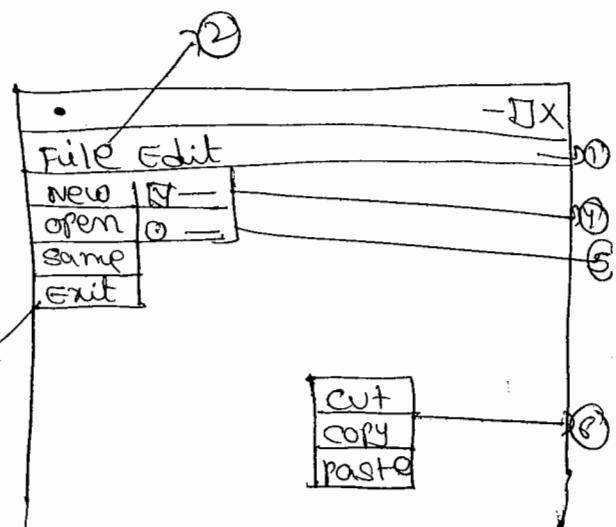
2) JMenu

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:-

① 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

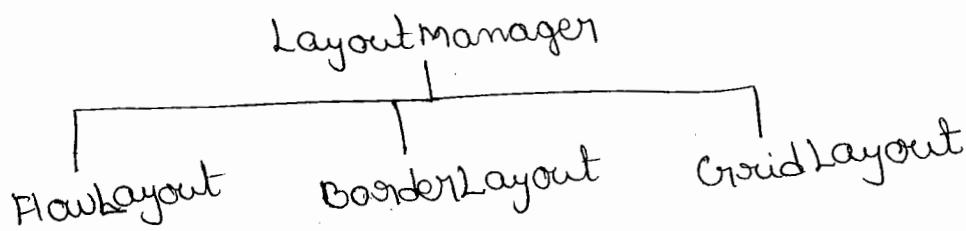
Step8:-

add components to a container

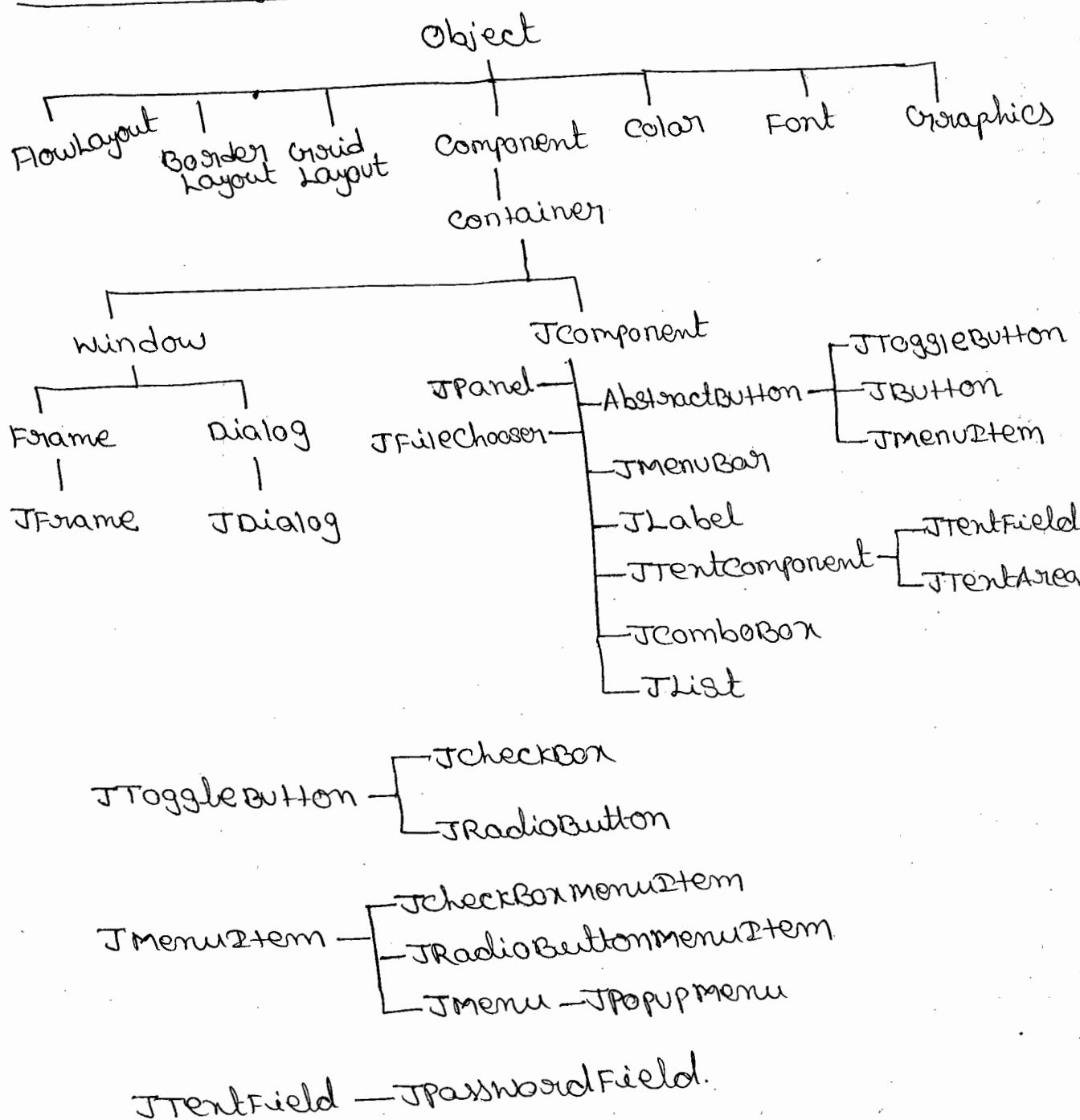
Step9:-

set the container size

- C Step10:-
C set the container visible mode.
 - C Step11:-
C write main method
 - C Step12:-
C create an object of current class
 - C Step13:-
C override event handlers
 - C Step14:-
C write task code
 - C The following method used to set the layout
 - C Java.awt.Container
 - C method:-
C public void setLayout(LayoutManager);
 - C Container Default Layout
 - C 1) JFrame → BorderLayout
 - C 2) JPanel → FlowLayout
 - C 3) JDialog → BorderLayout



The Hierarchy of swing classes:-



Java.awt.Window:-

Methods:-

width height
↑ ↑

public void setSize(int, int);
→ It is used to set the container size

public void setVisible(boolean);

→ It is used to set the visible mode.

Instantiating 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 ita = new JTextArea(10, 20); □ []

java.awt.Container

method! -

Method:-

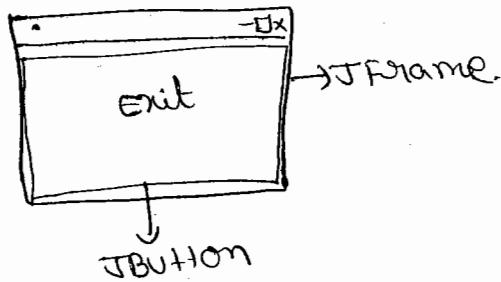
public Component add(Component);
→ It is used to add a component to a container.

```

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();
    }
}

```

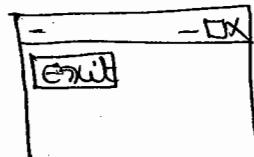
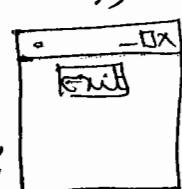
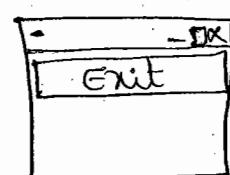
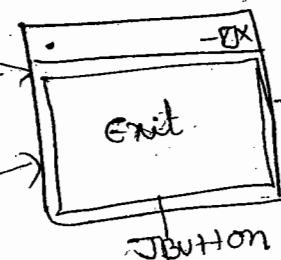
O/P:-



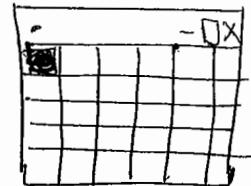
Setting Layouts to a Container!-

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);
 - 8) jb.setBounds(50,100,80,30);



- [JX]

— etc.

Event Handling

C Event Handling:-

- Event Handling:-
 - It is a mechanism and it is implemented by using event delegation model.

C Event Delegation model:-

It has 3 components

Example

1) Event source → java.awt.swing.JButton

2) Event object → `java.awt.event.ActionEvent`

- 2) Event object → `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
- action event object created, passed to actionPerformed
- method and actionPerformed method is executed.

java.java.lang.System

method 1-

Method:
public static void exit(int);

→ It is used to exit the application

- 0 => Normal Termination
- 1 => Abnormal Termination

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
{
    JButton jb;
    Demo()
    {
        setLayout(new FlowLayout());
        jb = new JButton("Exit");
        jb.addActionListener(this);
        add(jb);
        setSize(400, 400);
        setVisible(true);
    }
    public void actionPerformed(ActionEvent ae)
    {
        System.exit(0);
    }
}
```

↓
event listener

17/4/15

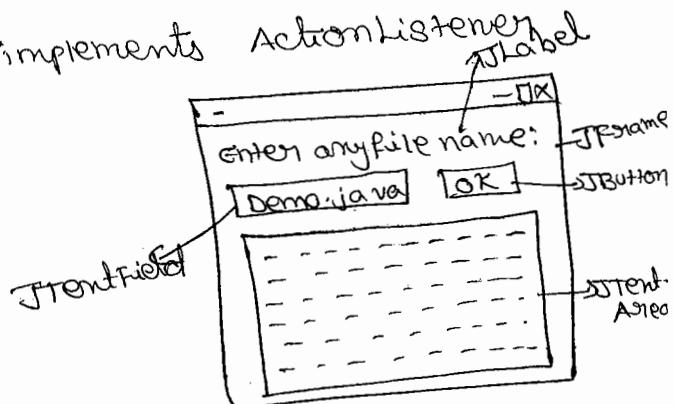
```
import java.io.*;
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;

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.
png"));

```

    f.i.s.read(b);
    String s = new String(b);      more than one button
    if(cal.getSource() == jb1) {
        i = 3;
    } else if(cal.getSource() == jb2) {
        i = 4;
    }
    System.out.println(e);
}
}
}

```

New Features :- (Last Topic for Core Java)

| Year | Version | New Features |
|------|---------|---|
| 1995 | JDK 1.0 | Initial 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) Covariant Return Type. 4) Annotations 5) Enumerations (enum) 6) Generics 7) static imports 8) varargs. |
| 2006 | JDK 1.6 | No changes at Language Level |
| 2007 | 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) toy with `try-with-resource` statement

- 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 JDK 1.4 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
```

```
{    public static void main(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));
```

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

```
}
```

```
}
```

Execution:-

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

```
c:\>java Demo
```

```
Enter any number b/w 1 and 10: 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 show()  
    {  
        system.out.println("B class");  
    }  
    psvm(string args[])  
    {  
        A ob = new BC();  
        ob.show();  
    }  
}
```

→ In the above example @override annotation informs the compiler "it is an overriding method"
→ All annotations start with '@' symbol

3) Lambda Expressions:-

This feature allows to write an anonymous class in a new way with anonymous method.

```

interface A {
    void show();
}

class Demo {
    public static void main(String args[]) {
        A ob = new A() {
            public void show() {
                System.out.println("welcome");
            }
        };
        ob.show();
    }
}

```

JDK 1.8

```

new A() {
    () -> { System.out.println("welcome"); }
}.show();

```

↓
Anonymous class

}

4) new Date & Time API :-

- 1) java.time.LocalDate
- 2) java.time.LocalDateTime
- 3) java.time.LocalDateTime

} classes.

```

import java.time.*;
class Demo {
    public static void main(String args[]) {
        LocalDate ld = LocalDate.now();
        System.out.println(ld);
        LocalDate ld2 = ld.plusWeeks(10);
        System.out.println(ld2);
        LocalDate ld3 = ld.plusDays(100);
        System.out.println(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