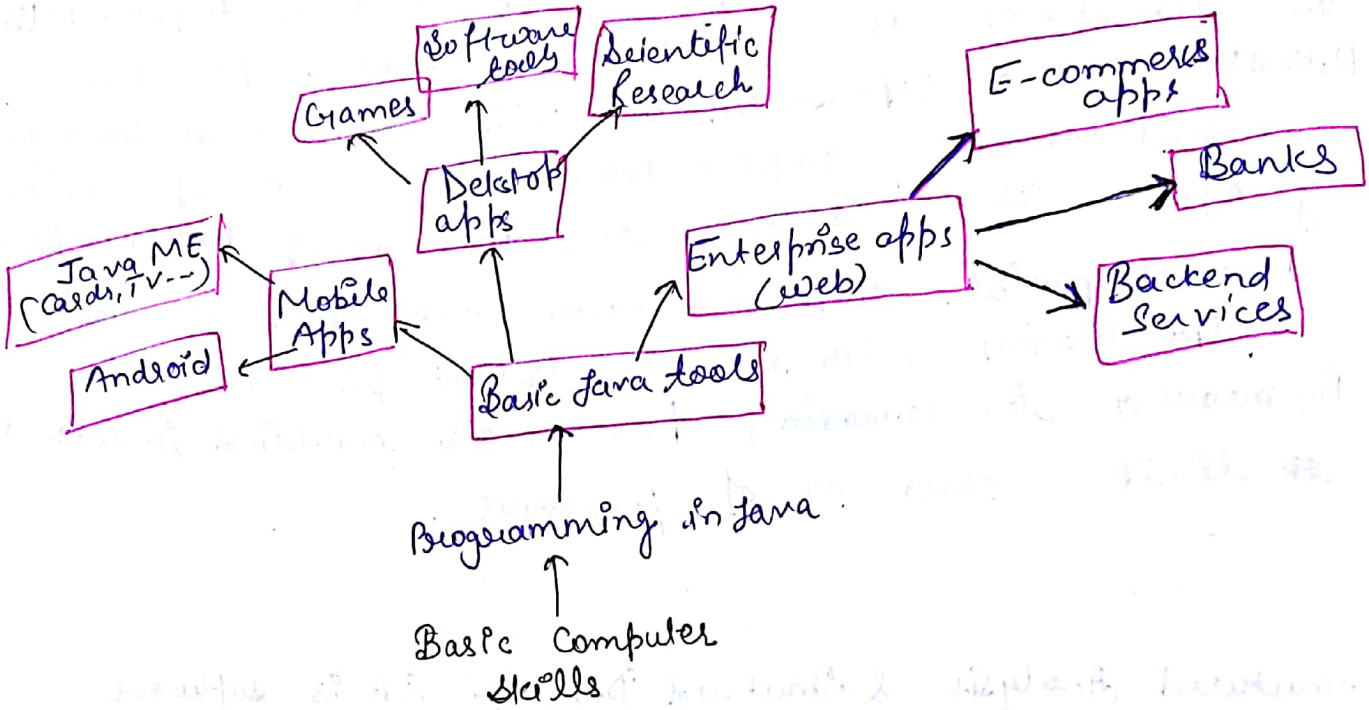
JAVA



⇒ Random no. generator.

Ex-

```
Random generator = new Random();
Boolean attacker = generator.nextBoolean();
```

⇒ Variables
- → local variables (local to a fun^{n}).
- → Instance variables (out-side every fun^{n} & constructor).
- → Class variables. (static variables).

Java is a general purpose computer based programming language. It is also a platform.

Java applications are typically compiled to bytecode that can run on any JVM, regardless of computer architecture.
All three JVM, JRE & JDK are platform dependent because configuration of each OS is different. But, Java is platform independent.

1. JDK: JDK is intended for software developers & includes tools such as Java compiler, Javadoc, Jar & a debugger.

2. JRE: JRE is a subset of JDK. It contains JVM in addition to Java package classes & Java libraries required to run java program.

3. JVM: (Java Virtual Machine) is an abstract machine. It provides runtime environment in which Java Bytecode can be executed.

Data types

			Range
①	byte	8	-2^7 to 2^7-1 i.e. -128 to 127
②	short	16	-2^{15} to $2^{15}-1$
③	int	32	-2^{31} to $2^{31}-1$
④	long	64	-2^{63} to $2^{63}-1$.

Floating Point type

⑤ double → 64 bits

⑥ float → 32 bits

⑦ Character: 16 bits 0 - 65,536.
Java uses unicode to represent characters.

⑧ Boolean :- true or false.

⇒ In case of Java JDK 8, switch case can use String for condition checking.

→ In Java, comma is used as seperator not operator.

⇒ New version of for loop is for-each loop.

```
int nums[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
for (int x : nums) {
    System.out.println(x);
}
```

By default the for-each loop iterates until all elements in an array have been examined.

⇒ Type conversion :- And Casting.

i) Implicit Type conversion → when types are compatible

ii) Explicit type conversion → when type are not compatible.

a) Compatibility : to Char and boolean are not compatible with each other.

b) Source type should be smaller than destination type.

When the size of the whole number component is too large to fit into Integer type then the modulo of the value will be stored in the target type.

e.g.

```
int i=257;
byte b = (byte)i;
System.out.println(b);
```

Output : 1.

⇒ Automatic Type Promotion in Expressions:-

Java automatically promotes each byte, short or char operand to int- when evaluating an expression.

After automatic promotion the result of exp is of type int- which cannot be assigned to smaller types so we need casting for that-.

Type promotion rule:-

The whole expression is promoted to the greatest- type; for instance, if any operand is long, the whole exp is promoted to long.

⇒ ARRAYS :-

```
int- a1 = new int[10];
int- a2 = {1, 2, 3, 4, 5};
```

Note:- Arrays in Java are implemented as ~~as~~ objects.
So, each array have a attribute named as length an instance variable

```
SOP("length of a1 is "+ a1.length);
```

INTRODUCING CLASSES -

Number of classes in a file will be equal to number of class file generated, so, each class can be put in its own source file.

Class is virtual, object is real entity.

It defines a new data type.

The variables in a class are called instance variables.

ex →

```
class Box{ int ln;
           int br;
           int ht;
         }
```

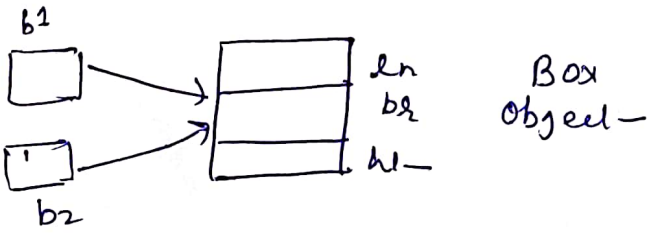
```
Box obj = new Box();        // Actually creating object.
```

ex →
```
Box mybox;                  //declare reference to an object of type box.
mybox = new Box();          // allocate a box object.
```

⇒ Object Reference variable :-

```
Box b1 = new Box();
Box b2 = b1;   // does not create new object.
```

here, no object is created, no memory is allocated, rather, b2 (pointed by b2) will point the same object as does b1;



Assigning one object variable to other is only making a copy of reference, not copy of the object.

Parameter v/s Argument-: A parameter is a variable that receives a value when the method is called. An argument- is the value that is passed to a method.

* Garbage Collection:- Deallocation of dynamically created memory. Java does it automatically unlike C++.

⇒ The "this" keyword:- 'this' keyword can be used inside any method to refer current- object-.
This keyword can be used in case when formal parameters names hides instance variable.

ex.
```
Box (double width, double height-) {
    width = width;
    height- = height-;
```

⇒ In java, if objects are passed as parameter, then they are always passed using call by reference method. So, any change, you will make in method will be reflected back in the object-.

ex:-
```
class Test { int- a, b;
    Test (int- i, int-1) { a=i;
                           b=j;
                         }
    void math (Test o) {
        o.a *= 2;
        o.b /= 2;
    }
}
class P { public static void main (String arg[]) {
    Test ob= new Test (15, 20);
    S.O.P ("ob.a & ob.b before call " + ob.a + " " + ob.b);
    ob.math(ob);
    S.O.P ("ob.a & ob.b after call " + ob.a + " " + ob.b);
}
```

O/p: 15 20
30 10

The math() have effected the object- used as an argument.

Packages :- Categorisation of class on the basis of functionality & usability

→ We can use same class name in different- Packages.
→ There can be different- Packages in same project-.

Javac -d . Hello.java.
java packagename.Hello

→ Only one class can be declared public in single java file.
→ Name of file should be same as name of public class.
→ A package is both naming & visibility control mechanism.

Ex-

```
package hello;

public class hello {
    int a;
public  Hello(int x) {
        a=x;
    }
    public void display() {
        System.out.println("In diff. package &non subclass");
    }
}
```

public → class & methods being imported needs to be declared public.

```
package anu;
import hello.*;
public class Anushka {
    public static void main (String arg[]) {
        Hello ob = new Hello(6);
    } ob.display();
```

Access Specification;-

	Def	Public	Private	Protected
Same class	Yes	Yes	Yes	Yes
Same Package Subclass	Yes	Yes	No	Yes
Same Package Non-Subclass	Yes	Yes	No	Yes
Different- Package Subclass	No	Yes	No	Yes
Different- Package Non-Subclass	No	Yes	No	No.

* If class is declared Public it- can be accessed by any code, if it- is declared ~~it~~ default-, it- can be accessed by any code in same package.

Interface

An Interface is a collection of abstract- methods.
A class implements an Interface.
Contains Behaviour that- a class implements.
Unless the class that- implements interface is abstract-, all the methods of interface must- be defined in a class.

ex- Interface i1 { void show(); }

abstract- class c1 implements i1 {
    public void show() {
        Sy.o.P("I am show method of Interface");
    }
}

↓ when no access modifier is used then default- access results, and Interface is available to the very package it belongs to

ex:-

```
Interface flying {
        void test();
    }
interface nonflying {
        void test-();
    }
class birds implement flying, nonflying {
        public void test-(){ SOP(" ");
        }
    }
class Test{
    public static void main(String[]){
        bird.flying ob = new birds();
        ob.test-;
    }
}
```

⇒ Partial Implementations:-

If a class implements interface partially, then it must be declare as abstract.

ex- abstract class A implements flying{

```
    ins-a;
    void show(){
        SOP("show");
    }
}
```

// A does not-
implement test();

⇒ Nested Interface :-

?? = 2 classes needed for interface implementation.

No

```
class A {
    public interface i1 (int x) {
        void fun (int x) {
            SOP("x is " + x);
        }
    }
}
class B implements A.i1 {
    void fun (int x) {
    }
}
class C { public static void main (String a[]) {
        A.i1 ob = new B();
        ob.fun(5);
    }
}
```

→ accessed using classname & '.' operator.

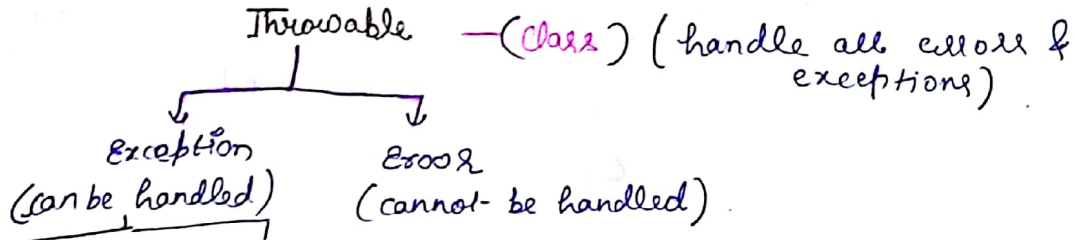
Ex:-

```
interface stack { void push (int x);
                  void pop();
}
class staticstack implements stack {
    private int top;
    private int stck[];
    public void push (int x) {
        if stack[
```

```
staticstack ( int- size) {
        stack = new int-[size]
          top = -1;
        }
public void push (int- x) {
        if ( top = = stck . length-1) {
          SOP ("Overflow");
            }
        else
          stck[++tos] = x;
    }
public void pop ( ) {
            if ( top = = -1) {
              SOP( "Empty Stack"); return;
            }
            else{ int-a;
              a = stck[top];
              top--;
              SOP(a);
            }
        }
}
```

Exception Handling

An exception is an abnormal condition that arises in a code sequence at run-time.



⇒ Some examples of exceptions :-

i) when we try to open a file that doesn't exist.
ii) devide by zero.

eg.

```
try { FileOutputStream fos = new FileOutputStream("abc.txt");
    }
catch ( ArithmeticException ae){

    }
catch (IOException ie) {

}
catch (Exception e) {
// handles all type of exception.
}
```

ex1:

```
class Exceptn { public static void main (String arg[]) {
                    int i, j, k=0; //must be initialised before try block.
                    i=8;
                    j=0;
                    try {
                        k = i/j;

                    }
                    catch (Exception e) {
                        SOP ("Cannot devide by zero");

                    }SOP (k);
```

Ex 2: Multiple catches with one try :-

```
class Demo {
    P.S.V.M (String Arg[]) {
        int arr[] = new int[4]; int i = 4 j = 5;
        try {
            k = j/J; // If this raises exception then rest try block wont run & will go only AE.
            for (int c=0 ; c<=4 ; c++)
            { a[c] = c+1;
            }
        }
        catch (ArithmeticException e){
            SOP ("Cannot divide by zero");
        }
        catch(ArrayIndexOutOfBoundException e)
        { SOP ("Maximum limit exceeds");
        }
        catch(Exception e) {    // this one should be last-catch.
            SOP ("Unknown Exception");
        }
    }
}
```

⇒ Finally { } : Does'nt matter whether exception occurs or not, its going to executed.

Uses : Can be used to close the resources (like files) currently in use.

⇒ STRINGS In Java :-

1) Lexicographic comparison :- Lexicographic means string which will come first in dictionary.
Java provide fun^n for comparison (lexicographically).

String.CompareTo() method.

ex- String2.compareTo(String2)

- <0 if S1 comes first-
- ==0 if S1 == S2
- >0 if S2 comes first.

Note :- There is also a variant- .compareToIgnoreCase().

⇒ Try with resources:-

⇒ User defined exception:-

```
public class DemoEx {
    public static void main(String[] args) {
        int i=5;
        try {  if(i<5) {
                throw new MyException ("Errorr"); // calling constructor.
            }
        } catch (Exception e) {
            System.out.println(e);
        }
    }
}
class MyException extends Exception {
    public MyException (String msg) {
        super(msg);      // to print the message or
                         // calling constructor of super class.
    }
}
```

⇒ THROWS :- If a method is capable of causing an exception that it does not handle, it must specify this behaviour so that callers of the method can guard themselves against that exception.

ex.

```
class ThrowDemo {
    static void throwOne() throws IllegalAccessException {
        SOP("Inside throwOne");
        throw new IllegalAccessException("demo");
    }
    public static void main(String args[]) {
        try {
            throwOne();
        } catch (IllegalAccessException e) {
            SOP("caught-" + e);
        }
    }
}
```

MULTITHREADED PROGRAMMING

A multithreaded program contains two or more parts that can run concurrently. Each part is called thread.

Threads :- Unit of Process. If we break a process in smaller parts we get thread.

A thread is an independent path of execution within a program.

Every thread in Java is created and controlled by the java.lang.Thread class.

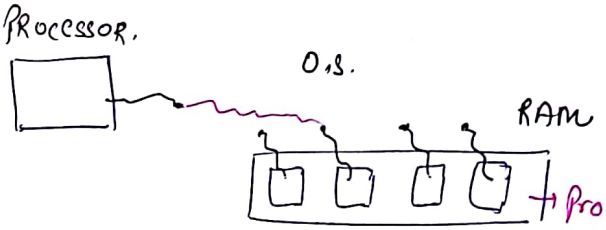
By default we have main thread, all the execution is carried out by using main thread.

⇒Threads can be implement using two ways :- (To create a new thread)

1. Using Runnable Interface
2. Using Thread Class.

max^m use of processing power available in the system.

Instance :-



Processes are put in circular queue.

Steps - ① Create thread
② attache code with thread
③ Run thread.

1. Using Runnable Interface: (java.lang.Runnable).

```
* Interface Runnable {
      run() {
      }
}.
```

Thread t1 = new Thread(); — object of class

ex- class A implements Runnable {
public void run() {
int i;
for(i=0; i<=10; i++)
{ SOP("Thread A" + i);
}
}

```
class B implements Runnable {
    public void run() {
        int i;
        for(i=0; i<=10; i++)
            SOP("Thread B" + i);
    }
}

public class Example {
    public static void main (String[] args) {
        Thread t1 = new Thread(new A());   // Attaching
        Thread t2 = new Thread(new B());
        t1.start();
        t2.start();
    }
}
```

// creating thread

// running thread

2. Using Thread Class :-

ex-

```
class A extends Thread {
    public void run() {
        int i;
        for(i=1; i<=10; i++)
            SOP("i = " + i + "Thread A");
    }
}
class B extends Thread {
    ---
    ---
    ---
}

public class Example {
    public static void main (String[] args) {
        A ob1 = new A();
        B ob2 = new B();
        ob1.start();
        ob2.start();
```

Using Runnable interface is preferred over extending class as, if we implement thread using Thread class then we cannot have any parent class of that class, as multiple inheritance is not allowed in Java.

→ Using delay in output :-

```
class Hi extends Thread {
        public void run display() {
            for(int i=0; i<5; i++) {
                SOP("Hi");
                try { Thread.sleep(500); } catch(Exception e){}
            }
        }
}
class Hello extends Thread {
        public void run show() {
            for(int i=0; i<5; i++) {
                SOP("Hello");
                try { Thread.sleep(500); } catch(Exception e){}
            }
        }
}
class Demo { public static void main(String[] arg) {
        Hi ob1 = new Hi();
        Hello ob2 = new Hello();
        ob1.display();
        ob2.display();
    }
}
```

ob1.start(); ob2.start(); → copy

Both ob1 & ob2 may reach the scheduler at same time at any point of execution, to avoid that → delay is used.

→ Code & O/P -

Hello
Hi
Hello
Hi
Hello
Hi
Hello
Hi

→ using runnable Interface :- this interface has only one function to be implemented by derived class & i.e. run().

```
class Hi implements Runnable {
        public void run(){
            ==========
            ==========
        }
}
class Hello implements Runnable {
        public void run() {
        }
}
class Demo {
    Runnable Hi ob1 = new Hi();
    Hello ob2 = new Hello();
    Thread t1 = new Thread(ob1);   // passing runnable object to thread class constructor.
    Thread t2 = new Thread(ob2);
    ob1.start();   // no start() in runnable, its of thread class.
    ob2.start();
}
```

⇒ join() and isAlive() :- join() :- used to wait for a thread to finish. isAlive method returns true if thread upon which it is called is still running.

after that - other messages will get display.

[Refer book]

⇒ Synchronized Keyword :- to be used when we want only one thread to execute the method at a time.

```
Ex- class Counter {  int count;
        public synchronized void increment() {
            count++; }
}
public class Demo { public static void main(String arg[]) throws Exception
    { Counter C = new counter();
      Thread t1 = new Thread(new Runnable()
        { public void run() {
            for(int i=1; i<=1000; i++) {
                C.increment();
            }
        });
```

```
Thread t2 = new Thread(new Runnable()
        { public void run(){
                for(int i=1;i<=1000;i++)
                { c.increment();
                }
        }
    );
    t1.start();
    t2.start();
    t1.join();
    t2.join();
    SOP("Count-"+c.count);
}
```

Output:- 2000 → count - with synchronized keyword.
check without - ib.

⇒ Static Synchronization : at class level :-

Synchronized method :-

```
class Table {
        void printTable(int n){
            for(int i=1;i<10;i++)
                SOP(n*i);
            }
}

class MT1 extends Thread {
        Table T1;
        MT1(Table t){
            T1=t;
        }
        public void run(){
            T1.printTable(5);
        }
```

```
psvm (str, obj[]) {
    MT1 obj1 = new MT1(obj);
    MT2 obj2 = new MT2(obj);
    MT1.start();
    MT2.start();
}
```

Table obj = new Table();

);

ble,

)

Scanned by CamScanner

3/10/2013

SWING → Basic app. Pr to provide GUI.
→ Supports plagable look & feel.
→ Supports large no. of components
→ light-weight.
→ Jts component- have enhanced functionality
→ Support- MVC.

AWT does not support- MVC (Model View Controller)

import- javax.swing.*;

Any swing version of Applets must- implement- JApplet-
↓
A large no. of features.

⇒ Contentpane

container get-Content-pane()
void add (comp)

→ Icons :- Labels can be associated with text- as well as icons. ImageIcon class contains foll. constructors :-
1) ImageIcon (String filename)
2) ImageIcon (URL url) → when image is remotely available.

Methods to manipulate icon :-
1) int get-IconLength() → in terms of no. of pixels
2) int get-IconWidth()
3) int paint-Icon (Component- comp, Graphics g, int-x, int-y).

→ Label :- constructors :-
1) JLabel (Icon i)
2) JLabel (String s)
3) JLabel (Icon i, String s)
4) JLabel (Icon i, String s, int-align)

} → both text- as well as icon.

Methods :-
1.) Icon get-Icon()
2.) String get-Text()
3.) void set-Icon(Icon i)
4.) void set-Text(String s).

⇒ Simple Applet- Program :-

```
import- javax.swing.*;
/* applet- code = " JlabelDemo" width = 250 height = 150>
                </applet-> */

public class JLabelDemo extends JApplet- {
        public void init() {
            Container content-pane = get-Content-Pane ();
            ImageIcon i1 = new ImageIcon ("frame.gif");
            JLabel J1 = new JLabel ("Frame", L1, J lable,
            content-pane . add (J1) ;
        }
```

COMPONENTS :- A component is an independent visual control. In order for a comp. to be displayed, it must be held in a container (JFC) (Java Foundation Classes)

Thus all swing GUIs must have atleast one container.

A container can hold another container.

ex- A frame can hold another frame, panels, buttons, text-fields etc as well.

Top level containers is not contained within anyother container.

⇒ Text- field :-

```
JText-field ( )
JText-field (int- cols)
JText-Field ( String s, int- cols)
JText-Field ( strig s).
```

⇒ Button :-

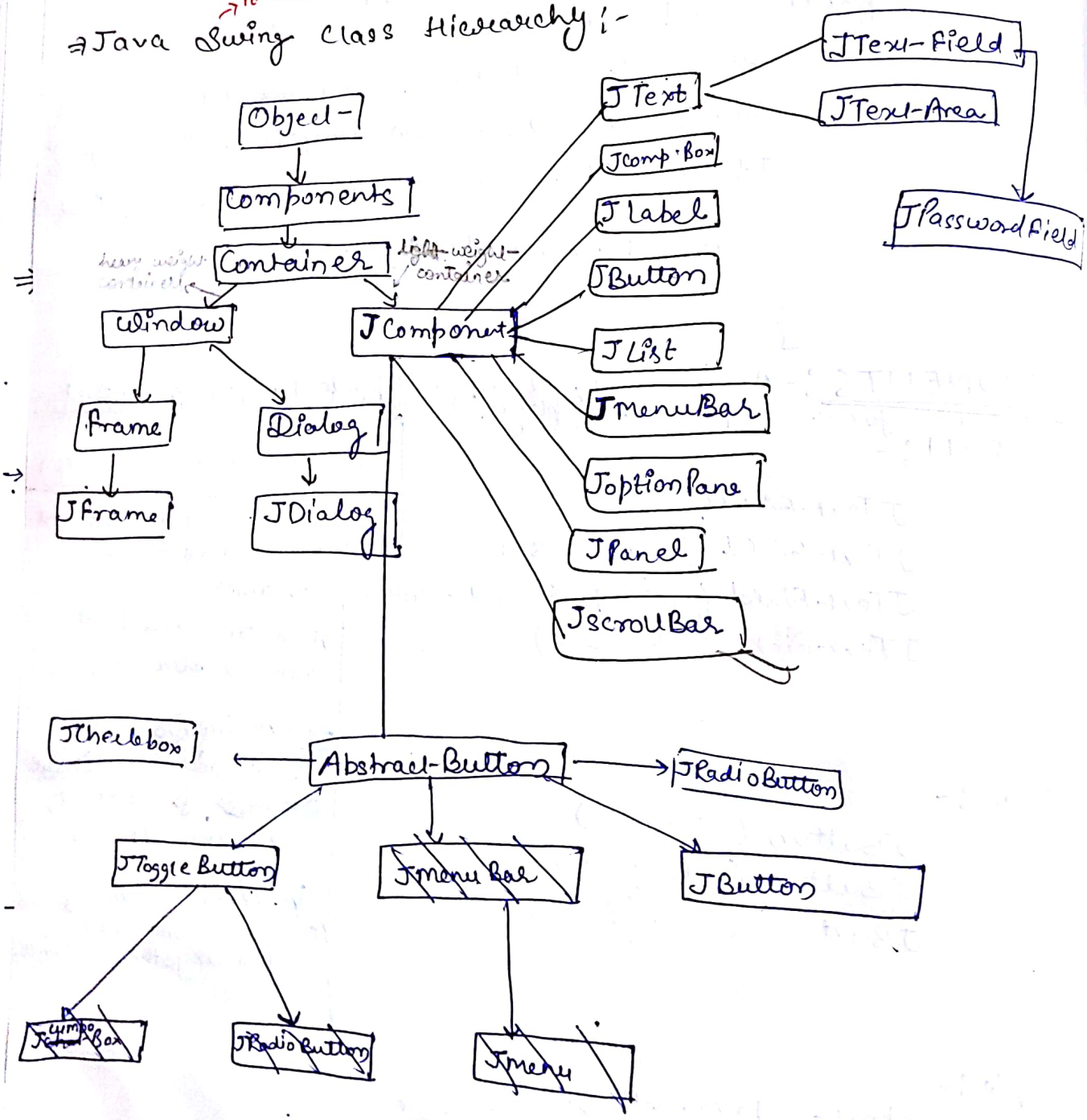
```
JButton (Icon i)
JButton (String s)
JButton (Icon i, String S).
```

⇒ Example :-

```
import- Java.swing.*;
Public class first-SwingEx {
        public static void main (string arg[]) {
            JFrame f = new JFrame ();
            JButton b = new JButton ("click");
            b.set-Bound (130, 100, 100, 40);
              f.add (b);
              f.set-size (400, 500);
```

```
f.set-layout- (null);
f.set-visible (true);
    }
}
```

⇒ Java Swing Class Hierarchy :- (it also contains some heavy wt components.)



⇒ Example:

```
import java.swing.*;
import java.event.*;
public class TextFieldEx implements ActionListener {
    JTextField *tf1, *tf2, *tf3;
    JButton b1, b2;
    TextFieldExample() {
        JFrame f = new JFrame();
        tf1 = new JTextfield();
        tF1 = setBounds(50, 52, 150, 120);
        tF2 = new JText-Field();
        tf2 = setBounds(50, 100, 180, 20);
        tF3 = new JText-field();
        tf3 = SetBounds(50, 150, 180, 20);

        b1 = new JButton("*");
        b1.setBounds(120, 120, 20, 20,);
        b2 = new JButton("-");
        b2.setBounds(120, 140, 20, 20);
        b1.AddButton
        b2.AddActionListener(this);
        b2.AddActionListener(this);
        f.add(tf1);     f.add(b2);
        f.add(tbf2);    f.add(b3);
        f.add(tf3);
        f.add(b1);
        f.set-size(100, 150);
        f.setLayout-(null);
        f.setvisible(true);
}
```

Scanned by CamScanner

```
public void actionPerformed (ActionEvent e) {

    String s1 = tf1.getText();
    String s2 = tf2.getText();

    int a = Integer.parseInt(s1);
    int b = Integer.parseInt(s2);
    int c = 0

    if (e.getSource() == b1) {
        c = a+b;
    } else c = a-b;
    String result = String.valueOf(c)

    tf3.setText(result);

    public static void main(--) {

        new TextFieldEx();

    }
```

Q. If u enter URL of any website, it should return the IP Address.

→ Components :-

1. ⇒ JTextArea :- → Class
   1) Constructors :-
      JTextArea (String s)
      JTextArea (int row, int col).
      JTextArea (String s, int r, int c)
      JTextArea().
   2) Methods
      void setRows (int r)
      void setColumns (int c)
      void setFont (Font f).

2. JPasswordField → Class extends JTextField
   Constructors :-
      JPasswordField()
      JPasswordField (String s)

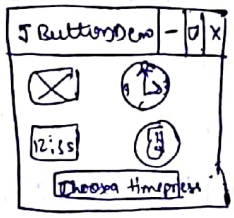
JPasswordField (String s, int c).
↓ no. of characters.

JCheckBox
↓ extends
JToggleButton implements Accessible

JCheckBox()
JCheckBox (String s)
JCheckBox(

9/10/18

Ex:- import java.awt.*;
import java.awt.event.*;
import java.awt.swing.*;



public class JButtonDemo implements
ActionListener {

JLabel jlab;

public JButtonDemo(){

JFrame jfrm = new JFrame("JButtonDemo");

jfrm.setLayout(new FlowLayout());

jfrm.setSize(500,400); jfrm.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

ImageIcon hourglass = new ImageIcon("hourglass.png");

JButton jb = new JButton("hour glass", hourglass);

jb.setActionCommand("HourGlass");

jb.addActionListener(this);

jfrm.add(jb);

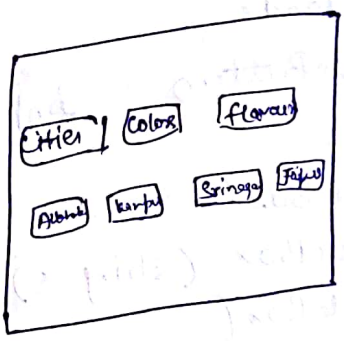
;

label = new JLabel("Choose a time");

JLabel.add(label)

jlabel.setVisible(true);

→ JTabbedPane: To attach more than one components to a tab, means when we click on tab, the components will be displayed.



Steps:
1. Create an instance of JTabbedPane.
2. Add each tab by calling addTab().
3. Add the tabbed pane to the content pane.

```
private void makeclick (){
    JTabbedPane Jtp = new JTabbedPane();
    Jtp.addTab("Cities", new CitiesPanel());
    Jtp.addTab("Color", new ColorPanel());
    Jtp.addTab("Flavours", new FlavourPanll());
     Jtp.add();
}

class CitiesPanel extends JPanel {

        public CitiesPanel(){

            JButton b1 = new JButton("New York");
            add(b1);
            JButton b2 = new JButton("Allahabad");
            add(b2);
            JButton b3 = new JButton("Tokyo");
            add(b3);

            }

}
class ColorPanel extends JPanel {
        Public colorPanel(){
            JCheckBox cb1 = new JCheckBox("Red");
            JCheckBox cb2 = new JCheckBox("Blue");
        }}
```

⇒ JScrollPane :

steps ; –

1. Create the component to be scrolled.
2. Create an instance of JscrollPane, passing it- to the object- to scroll Table.
3. Add the scroll pane to content- pane.

ex-

```
private void makeGui() {

    JPanel Jp = new JPanel();
    Jp.setLayout- (new GridLayout- (200, 200));
    int- b = 0;
    for(int- i=0; i<20; i++) {
        for(int- j=0; j<20; j++){
            Jp.add(new JButton("Button"+b);
            b++;
        }
    }
    JscrollPane Jsp = new Jscrollpane(Jp);
    add(Jsp, BoderLayout-.CENTER);
                    }
}
```

11/10/19

JTree :-

Constructors:

JTree (Object- obj[]) → Homogeneous.
JTree (Vector <>) → Heterogeneous.
JTree (TreeNode tn)

⇒ Model
① Tree Model ② Tree Selection Model.

→ Tree ExpansionEvent –
→ Tree SelectionEvent – (which node has been selection)
→ Tree Model Event –

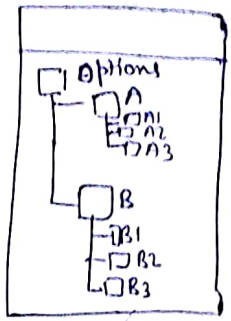
⇒ Program to create JTree :-

⇒ Steps :-
1. Create an instance of JTree
2. Create a Scroll Pane.
3. Add the tree in the Scroll-Pane.
4. Add the scroll pane to the content-Pane.

Program :-

```
import- java.awt.*;
import- java.swing.event-*;
import- javax.swing.*;
import- java.swing.tree.*;
/*<applet- create = "JFrameDemo" width=400 height- = 400>
</applet->*/
public class JTreeDemo extends JApplet- {
    JTree tree;
    JLabel Jlab;
    public void init-(){
        try {
            SwingUtilities.invokedwait(
            new Runnable() {
```

```
Public void run(){
                makeGUI(); } } );
} catch (Exception ex ) {
        SOPL---);
        }
```



Tree

```
public void makeGUI(){
  // creating tree
  Default- Mutable Tree Node top = new Default- Mutable Tree Node("Options");
  "  "  "  "  a = "  "  "  "  "  ("A");
     top.add(a);
  "  "  "  "  a1 = "  "  "  "  "  ("A1");
     a.add(a1);
  "  "  "  "  a2 = "  "  "  "  "  ("A2");
     a.add(a2);
  "  "  "  "  b = "  "  "  "  ("B");
     top.add(b);
  "  "  "  "  b1 = "  "  "  "  ("B1");
     b.add(b1);
  // Same for b2 also.
     tree = new ~~JTree~~ JTree(top);
     add(tree);
// creating Scroll Pane
     JScrollPane Jsp = new JScrollPane(Tree);
     add(Jsp);
// Creating label, path from given node to root node will be
     returned as string.
     Jlab = new Jlabel();
     add(Jlab, BorderLayout.SOUTH);
```

```
3) tree.addTreeSelectionListener(new TreeSelectionListener(){
    public void valueChanged(TreeSelectionEvent te){
        jlab.setText("Selection is " + tree.getSelectionPath());
    }});
```

⇒ JTable Class

⇒ Constructor :-

```
JTable(Object data[][], Object colHeading[]);
```

Example :-

```
public void makeGUI(){
    String[] colHeadings = {"Name", , "ID #"};
    Object[][] data = {
        {"Gail", "6125", "855"},
        {"Ken", "7666", "555"}
    };
    JTable table = new JTable(data, colHeadings);
    JScrollPane jsp = new JScrollPane(table);
    add(jsp);
}}
```

Ques1. write a program to find the frequency of characters in string.

Ques2: write a java program to remove all whitespace from string

Ques3: write a java prog.

Servlet- J2EE

Small java prog to enable functionality of Client-& Server.

- Application Server
  - .jar
  - ↓
  - Java archive file
- Web Server.
  - .war
  - ~~.jar~~
  - web archive file.

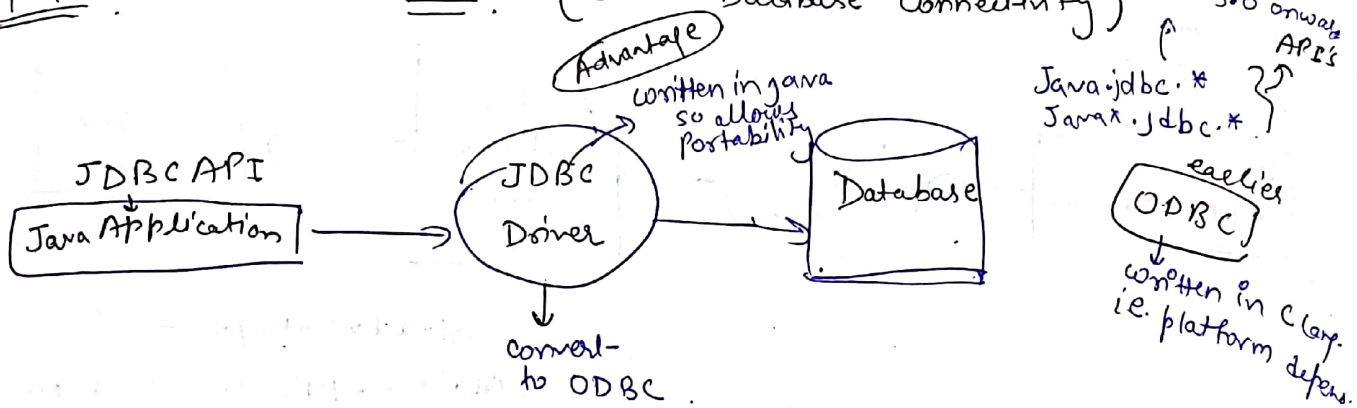
Command;-

java cvf filename *.* ↵

29/10/18-

JDBC (Java Database Connectivity) JDK 5.0 onwards APIs

Java.jdbc.*
Javax.jdbc.*

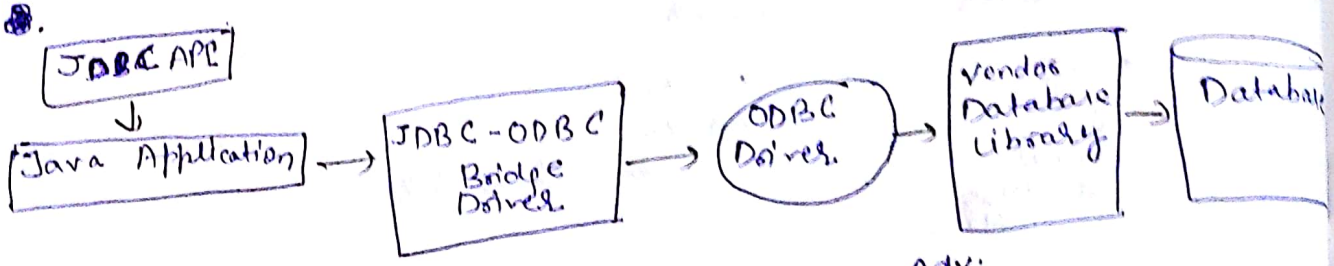


→ Environment- variables

JAVA HOME
CLASSPATH → Java compiler to find compiled code.
PATH → OS

⇒ JDBC Drivers :-

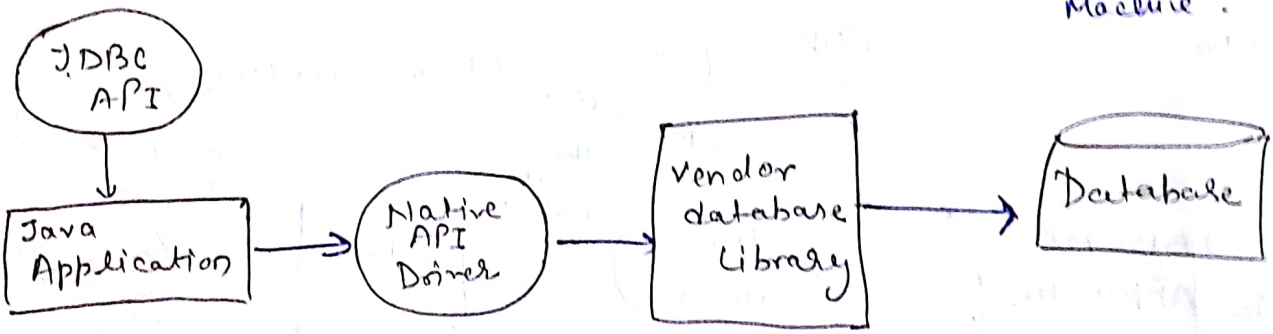
1. JDBC - ODBC Bridge Driver.



Adv:
1) easy to use

Disadv:
1) Performance Degradation due to additional JDBC driver.
2) ODBC driver needs to be installed in each Client Machine.

2. Native - API Driver (Type - II)



Disadvantage —
1) Native - API Driver needs to be installed in each Client Machine.

3. Network Protocol Driver

4. Thin Driver. (Widely used).

⇒ Steps to Connect – Java with DataBase;-

1. Register the driver class
   forName()

```
Class.forName("oracle.jdbc.driver.OracleDriver");
```

2. Creating Connections:-

```
Connection con = DriverManger.getConnection("jdbc:oracle:thin@
                 localhost:1521:xe", "System", "password");
```

3. Creating the Statement Object

```
Statement stat = con.createStatement();
```

4. Execute Query

```
ResultSet rs = Statement.execute("select * from emp");
while(rs.next()){
    SOP(rs.getInt(1) + rs.getString(2));}
```

(1 ↓ column no.)

5. Close the Connection Object -

```
con.close();
```

Servlet

website
↓
webpages
Static Dynamic

TOMCAT → opensource
↓ contains packages
javax.servlet.*;
javax.servlet.http;

Performance of servlet is much better than CGI because CGI is written in C, C++, Perl i.e. non-portable languages whereas servlet uses java app. hence are portable so consume less memory & processes faster using multithreading.

⇒ Steps:
1. Create & compile the servlet source
2. Start Tomcat
3. Start web browser & request for the servlet.

example:-

```
import java.io.*;
import javax.servlet.*;
public class HelloServlet extends GenericServlet {
    public void service(ServletRequest request,
                        ServletResponse response)
    {
        throws servlet.exception IOException {
        response.ContentType("Text/html");
        PrintWriter pw = response.getWriter();
        pw.println("<B>Hello");
        pw.close();
    }
}
```

init()
service()
destroy()
close();

import- java.servlet-.*

Servlet- API

Interface	Class
Servlet-	Generic Servlet
Servlet-config	Servlet Inputstream
Servlet-Context	Servlet Outputstream
Servlet-Request	Servlet-Exception
Servlet-Response	Unavailableexception

Methods

void destroy().

Servlet-Config get-Servlet-Config()

Stop get-Servlet-Info()

void init (Servlet-.Config ge)
throw servlet-Exception.

void service (Servlet-Request- eq)

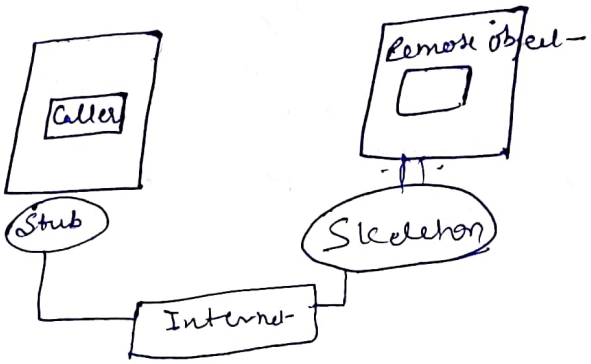
2

⇒ RMI (Remote Method Invocation)

Distributed application Method of one object may invoke method of remote object of remote machine

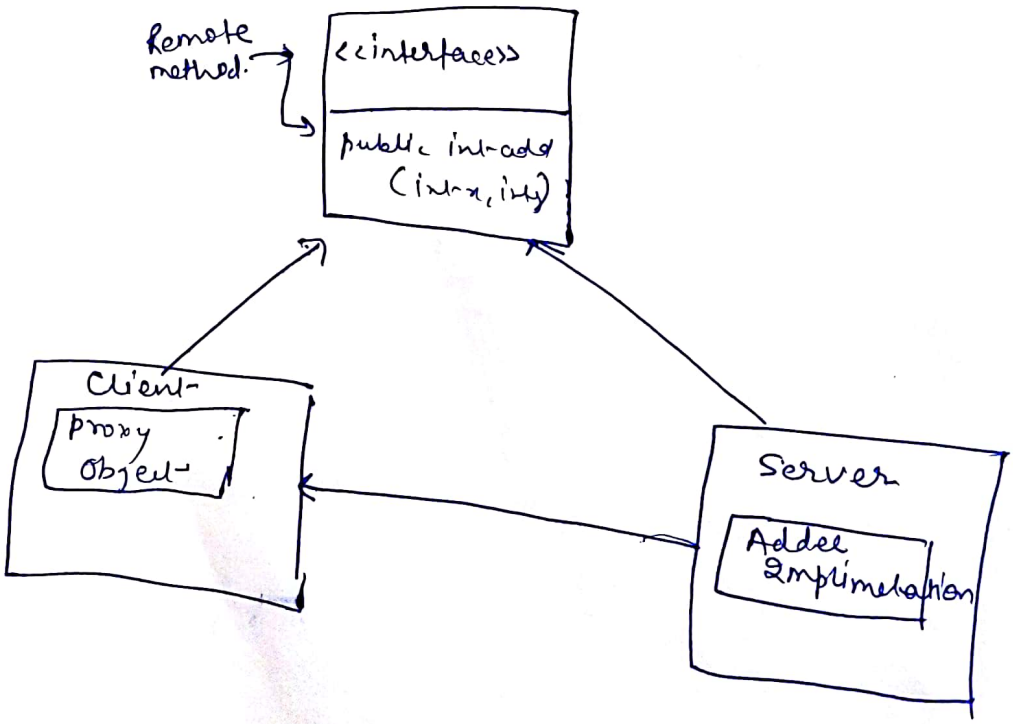
Machine A

Machine B



→ To build Distributed App.

1. To create remote Interface.
2. Provide the Implementation of the remote Inteface.
3. Compile the " class & create stub & skeleton using rmic tools.
4. Start the registering service using rmi registry tool.
5. Create & start remote app.
6. " " " client- " .



Step 1 :

```
import java.rmi.*;
public interface Adder extends Remote{
public int add(int x, int y)
        throw RemoteException;
}
```

Step 2 :-

```
import java.rmi.*;
import java.rmi.server.*;
public class AdderRemote extends UnicastRemoteObject
                         implements Adder {
  AdderRemote() throws RemoteException {
    super();
  }
  public int add(int x, int y)
    { return (x+y);
    }
}
```

Step 3 :- On command Prompt :-
rmic AdderRemote ↵

CMP :-

4 :- rmi registry (Portno) 5000

⇒ Naming class & methods defined in it :-
lookup() → returns reference to remote object.
bind() → binds the remote object with given name.
unbind() → destroys
rebind() → Object is bound with new name.
list() → returns a list of names bound to a given object.

5. Implementation of Remote Server :-

```
import- java.rmi.*;
import- java.rmi.registry.*;
public class MyServer {
    public static void main(--) {
        try {
            Adder stub = new AdderRemote();
            Naming.rebind("rmi://localhost:...");
        }
        catch(Exception e) { }
    }
```

6. Create client- application :-

```
import- java.rmi.*;
public class MyClient-{
    public static void main (--)
    {
    try {
        Adder stub = (Adder) Naming.lookup(
            ("rmi://localhost:5000/sos");
        Sop(stub.add(34,4));
    } catch (Exception e){{
        }
    }
```

Javac *.java

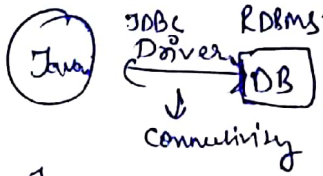
rmic AdderRemote

java myclient

java MyServer

⇒ Javabeans :- Comp. based.
BDK (Bean Dev. Kit)

JDBC : Java Database Connectivity.
To connect- GUI to DataBase.
We need to use a driver to connect- Java application with database.



4 type of Drivers :-

⇒ Steps for Connectivity :-

1. import- the package
2. Load & Register the Driver
   Register ⇒ Class. forName ("com.mysql.jdbc.Driver");
3. Establish the connection
4. Create the Statement-
5. Execute the Query
6. Procen the result-.
7. Close.

⇒ using code :-

Rno	Name
1.	Raj
2	Roci

① import java.sql.*;
main()
{
2(b) class.forName("com.mysql.jdbc.Driver");
3) Connection con = DriverManager.getConnection("URL", UserName, Password);
4) Statement- St- = con.CreateStatement-();
5) ResultSet- rs = St.executeQuery("Select * from t1");
6) while(rs.next-())
Sout-(rs.get-Int-(1) + " " + rs.get-Str'g(2));
↓ col. no.
7) St.close();
con.close();
}