KALASALINGAM
ACADEMY OF RESEARCH AND EDUCATION
(DEEMED TO BE UNIVERSITY)
Under sec. 3 of UGC Act 1956. Accredited by NAAC with "A++" Grade
Anand Nagar, Krishnankoil, Srivilliputtur (Via), Virudhunagar (Dt) - 626126, Tamil Nadu | info@kalasalingam.ac.in | www.kalasalingam.ac.in

School of Computing

DEPARTMENT OF INFORMATION TECHNOLOGY

COMPONENT BASED TECHNOLOGY (213INT2311)

LAB RECORD

Name of the Student	: ______________________________
Register No.	: ______________________________
Department	: ______________________________
Year/Sem/Sec	: ______________________________
Academic Year	: 2024 – 2025 (Even Semester)

DEPARTMENT OF INFORMATION TECHNOLOGY

BONAFIDE CERTIFICATE

Bonafide record of the work done by Mr./Ms.________________________________

Register No. ____________________ of _____ year _____ sec in B.Tech. Information Technology for the ____Subject _____________code during EVEN semester in academic year 20____- 20___.

Staff in-charge **Head of the Department**

Submitted to the Practical Examination held at KARE on _____________________

Register Number										

INTERNAL EXAMINER **EXTERNAL EXAMINER**

List of Experiments

Exp. No	**Name of the Experiment**	**Date**	**Page No.**	**Sign**
1.a	Development of simple com components in VB and use them in applications.			
2.a	Deploying EJB for simple arithmetic operators.			
3.a	Deploying RMI for client server applications.			
3.b	Deploying RMI for client server applications			
4.a	Creation of DLL using VB and deploy it in Java			
4.b	Programs using Active X controls			
5.a	Naming Services in CORBA			
5.b	DSI, DII in CORBA.			
5.c	Simple application using CORBA.			
6.a	Inter ORB in communication [IIOP,IOR] Java ORB & Visi broker ORB			
7.a	Deploying components for handling Multi media files.			
7.b	Deploying components for e-Business application			
7.c	Applications using COM / DCOM.			
7.d	Components in web applications.			
8.a	Distributed objects deployment-EJB and CORBA.			
9.a	Programs in Reflection, Introspection, Cloning Concepts			
10.a	Studying J2EE Server.			

EXP:1	**COM COMPONENT**		
1.a	Development of simple com components in VB and use them in applications.	3	3

AIM

To calculate the total, average and grade with remarks for the given student marks using VISUAL BASIC.

PROCEDURE

1. Create a form using VB and create labels , text boxes and command buttons

2. Get the student marks from the user

3. Find out the total for the given input

4. Find out the average.

5. Based on the marks give a grade and remarks to the student

PROGRAM

```
Dim Op1, Op2 As Double
Dim Opr As String

Private Sub Command1_Click()
Text1.Text = Text1.Text +
Command1.Caption If cleardisplay Then
Text1.Text = ""
cleardisplay =
False End If End
Sub

Private Sub Command2_Click()
Text1.Text = Text1.Text +
Command2.Caption If cleardisplay Then
Text1.Text = ""
cleardisplay =
False End If End
Sub

Private Sub Command3_Click()
Text1.Text = Text1.Text +
Command3.Caption If cleardisplay Then
```

```
Text1.Text = ""
cleardisplay =
False End If End
Sub

Private Sub Command4_Click()
Text1.Text = Text1.Text +
Command4.Caption If cleardisplay Then
Text1.Text = ""
cleardisplay =
False End If End
Sub

Private Sub Command5_Click()
Text1.Text = Text1.Text +
Command5.Caption If cleardisplay Then
Text1.Text = ""
cleardisplay =
False End If End
Sub

Private Sub Command6_Click()
Text1.Text = Text1.Text +
Command6.Caption If cleardisplay Then
Text1.Text = ""
cleardisplay =
False End If End
Sub

Private Sub Command7_Click()
Text1.Text = Text1.Text +
Command7.Caption If cleardisplay Then
Text1.Text = ""
cleardisplay =
False End If End
Sub

Private Sub Command8_Click()
Text1.Text = Text1.Text +
Command8.Caption If cleardisplay Then
Text1.Text = ""
cleardisplay =
False End If End
Sub

Private Sub Command9_Click()
Text1.Text = Text1.Text +
Command9.Caption If cleardisplay Then
Text1.Text = ""
cleardisplay =
False End If End
Sub
Private Sub
Command10_Clic
k()
Text1.Text = Text1.Text +
Command10.Caption If cleardisplay Then
Text1.Text = ""
cleardisplay =
False End If
End Sub
Private Sub
CommandDot_Click() If
InStr(Text1.Text, ".") Then Exit
Sub
Else
Text1.Text = Text1.Text +
"." End If
End Sub

Private Sub CommandPlus_Click()
Op1 = Val(Text1.Text)
Opr = "+"
Text1.Text = ""
End Sub

Private Sub CommandMinus_Click()
Op1 = Val(Text1.Text)
Opr = "-"
Text1.Text = ""
End Sub

Private Sub CommandMul_Click()
Op1 = Val(Text1.Text)
Opr = "*"
Text1.Text = ""
End Sub

Private Sub CommandDiv_Click()
Op1 = Val(Text1.Text)
Opr = "/"
Text1.Text = ""
End Sub

Private Sub CommandEqu_Click()
Op2 = Val(Text1.Text)
Select Case Opr
Case "+": Text1.Text = Op1 + Op2
Case "*": Text1.Text = Op1 * Op2
Case "-": Text1.Text = Op1 - Op2
Case "/": Text1.Text = Op1 / Op2
End Select End Sub

Private Sub Commandexit_Click()
End End Sub

Private Sub Form_Load()
Text1.Text = ""
End Sub
```

OUT COME:

After completing these exercise, student can able to do the VISUAL BASIC program to calculate CGPA

EXP : 2	**ENTERPRISE JAVA BEANS**		
2.a	Deploying EJB for simple arithmetic operator.	3	6

AIM

To write a Java application for bank accounts using visibroker.

PROCEDURE

Step 1: Start

Step 2: Go to File => New Project and choose Enterprise from categories and select Enterprise Application from project

Step 3: Name the project name as ejb1 and click on next and finish

Step 4: In project name tab, right click on the ejb1-ejb & choose New => Session bean

Step 5: Name the EJB name as ejb1class & package name as ejbpack and click on stateless in Session type & check the remote in create interface

Step 6: Right click on the coding area and select EJB Methods => Add Business Method

Step 7: Name the business method as add and return type as java.lang.Integer and add parameters x, y & their type as int and click on ok button

Step 8: In ejb1classBean.java under add function change return NULL to return x+y

Step 9: In projects tab right click on ejb1-war & choose New => select [servlet]

Step 10: Name class name as ejb1servlet and package as ejb1pack and click on next and finish

Step 11: Add the necessary imports to the ejb1servlet java code under doget()

Step 12: Compile the ejb1servlet.java program

Step 13: In project tab, under ejb1-war click on webpages => index.jsp

Step 14: Replace the code on the “Hello World” statement

Step 15: Save and Run the project then the designed output will be displayed

PROGRAM: EJB – ARITHMETIC OPERATION

```
//ejb1servlet.java

package ejb1pack;

import java.io.*;

import java.net.*;

import javax.ejb.EJB;

import

javax.servlet.*;

import javax.servlet.http.*;

public class ejb1servlet extends HttpServlet {

@EJB

private ejb1classRemote ob;

   @Override

   protected void doGet(HttpServletRequest request, HttpServletResponse response)

   throws ServletException, IOException {

      response.setContentType("text/html;charset=UTF-

      8"); PrintWriter out = response.getWriter();

      int  a=Integer.parseInt(request.getParameter("one"));

     int  b=Integer.parseInt(request.getParameter("two"));

     out.println(ob.add(a,b));

   }
```

index.jsp

```
<form action="ejb1servlet" method="get">
                ENTER FIRST NUMBER &nbsp;
        <input type="text" name="one" value="" /><br>
ENTER SECOND NUMBER &nbsp;
       <input type="text" name="two" value="" /><br>
<input type="submit" value="submit" />
</form>
```

OUTCOME

Thus the EJB bean Java program for arithmetic operation was performed using servlets and the output was verified

EXP: 3	**REMOTE METHOD INVOCATION**		
3.a	Deploying RMI for client server applications RMI PALINDROME.	3	9

AIM

To write a Java program for the implementation of Remote Method Invocation for palindrome.

ALGO

RITHM

SERVER

SIDE

Step 1: Start
Step 2: Define the class rmiserver
Step 3: Create the object twox in try
Step 4: Register the object twox
Step 5: Display the exception in
catch Step 6: Stop

CLIENT SIDE

Step 1: Start
Step 2: Define the class rmiclient
Step 3: Initialize the string s1 in try
Step 4: Create and Initialize the object onex
Step 5: Assign the value to m by calling the method
palin Step 6: Check whether the string is palindrome or
not Step 7: Display whether the string is palindrome or
not Step 8: Display the exception in catch
Step 9: Stop

PROGRAM REMOTE METHOD INVOCATION – PALINDROME
one.java

```
import java.rmi.*;
interface one1 extends Remote
{ public int palin(String a)throws RemoteException;
}
```

two.java

```
import java.rmi.*;
import java.lang.*;
import java.rmi.server;
    public class two2 extends UnicastRemoteObject implements one1
    {
    public two2()throws RemoteException{}
    public int palin(String a)throws RemoteException
    {
    System.out.println("Hello");
    StringBuffer str=new StringBuffer(a);
    String str1=str.toString();
    System.out.println("print:"+str1.toString())
    ; StringBuffer str2=str.reverse();
    System.out.println("print:"+str2.toString())
    ; int b=str1.compareTo(str2.toString());
    System.out.println("print:"+b);
    if(b==0
    ) return
    1; else
    return
    0;
    }
    }
```

rmiclient.java

```
import java.io.*;
import java.rmi.*;
import java.net.*;
public class rmiclient1
{
public static void main(String args[])throws Exception
{
try
{
String s1="rmi://localhost/palin";
one1
onex=(one1)Naming.lookup(s1); int
m=onex.palin("madaom");
System.out.println("Print:"+m);
if(m==1)
System.out.println("The Given String is Palindrome ");
else
System.out.println("The Given String is Not a Palindrome ");
}
catch(Exception e)
{
System.out.println("Exception: "+e);
}
}
}
```

rmiserver.java

```
import java.io.*;
import java.rmi.*;
import java.net.*;
public class
rmiserver1
{
public static void main(String args[])throws Exception{
try{
two2 twox=new two2();
Naming.bind("palin",twox);
System.out.println("object is registered");
}
catch(Exception e)
{
System.out.println("Exception : "+e);
}
}
}
```

RESULT:

Thus the Java program for the implementation of Remote Method Invocation for palindrome was performed and the output was verified.

3.b	**Deploying RMI for client server applications** **RMI FOR ADDITION AND SUBTRACTION.**	3	9

AIM

To write a Java program for the implementation of Remote Method Invocation for addition and subtraction operations.

ALGORITHM

SERVER SIDE

Step 1: Start
Step 2: Define the class rmiserver
Step 3: Create the objects twox, twoy in try
Step 4: Register the objects twox, twoy
Step 5: Display the exception in catch
Step 6: Stop

CLIENT SIDE

Step 1: Start
Step 2: Define the class rmiclient
Step 3: Initialize the string s1 & s2 in try
Step 4: Create the objects onex, oney
Step 5: Assign the value to m by calling the method add(4,2)
Step 6: Assign the value to n by calling the method sub(40,30)
Step 7: Display m & n
Step 8: Display the exception in
catch Step 9: Stop

PROGRAM:

one.java

```
import java.rmi.*;
interface one extends Remote{
public int add(int a,int b)throws RemoteException;
public int sub(int a,int b) throws RemoteException;}
```

two.java

```
import java.rmi.*;
importjava.rmi.server;
```

```
     public class two extends UnicastRemoteObject implements
one{
public int add(int a,int b)throws RemoteException{
System.out.println("Hello");
return(a+b);}
public int sub(int a,int b)throws RemoteException{
System.out.println("Hello");
return(a-b);}}
```

rmiserver.java

```
import java.io.*;
import java.net.*;
import java.rmi.*;
public class
rmiserver{
public static void main(String args[]){
try{
two twox=new two();
Naming.bind("ad",twox);
System.out.println("Object is registered.");
two twoy=new two();
Naming.bind("sub",twoy);
System.out.println("Object is registered.");
}
catch(Exception e){
System.out.println("Exception:"+e);}}}
```

rmiclient.java

```
import java.io.*;
import java.rmi.*;
import java.net.*;
public class
rmiclient{
public static void main(String args[])throws Exception{
try{
String s1="rmi://localhost/ad";
one
onex=(one)Naming.lookup(s1); int
m=onex.add(40,30);
System.out.println("Addition:"+m)
; String s2="rmi://localhost/sub";
one
oney=(one)Naming.lookup(s2); int
n=oney.sub(40,30);
System.out.println("Subtraction:"+n);
}
catch(Exception e){
System.out.println("Exception:"+e);}}}
```

RESULT:

Thus the Java program for the implementation of Remote Method Invocation for addition and subtraction was performed and the output was verified.

EXP : 4	**VISUAL BASIC**		
4.a	**Creation of DLL using VB and deploy it in Java**	**3**	**12**

AIM:

To create a dll for adding two numbers in Visual Basic and deploy in Java.

PROCEDURE:

Creating a visual basic DLL:

1) Start->All Programs->Microsoft Visual Studio6.0->Microsoft Visual Basic6.0

2) In new project dialog choose ActiveX DLL.

3) Now the default project name will be Project1 and default class name will

be Class1.

4) Click on form ->Tools->Add procedure.

5) Choose function->Give function name as add.

6) In the editor window type the native methods.

```
Public Function add(ByVal a As Integer, ByVal b As Integer)
MsgBox "Result:" + CStr(a + b)
End Function
```

7) Save the files in proper location. For example C:\wings\jnidemo\

8) Now choose File menu and click Make Project1.dll

9) Close the visual basic editor.

Creating Java Class using native methods:

1) Create a java file NativeImpl.java in C:\wings\jnidemo\

2) Code the file as follows:

PROGRAM:

NativeImpl.java

```
import

java.io.DataInputStream;

import java.io.IOException;

class NativeImpl

{

public native void add(int a, int b);

public static void main(String args[])throws IOException

{

DataInputStream in=new DataInputStream(System.in);

NativeImpl impl=new NativeImpl();

System.out.println("Enter number 1 & 2:");

impl.add(Integer.parseInt(in.readLine()), Integer.parseInt(in.readLine()));
}

static
{

System.loadLibrary("dllproj");
}
}
```

3) Compile the file using javac NativeImpl.java

4) Now create the JNI header files for the class file using the following

command javah –jni NativeImpl

5) Now the Header file is created.

Creating VC++ wrapper DLL:

1) Open VC++ IDE.
2) Create a new MFC AppWizard(DLL) Project name it as dllproj and

   choose C:\wings\jnidemo\ as project location.

3) Copy the NativeImpl.h file to the directory C:\wings\jnidemo\dllproj\

4) Also copy JNI header files provided by java as follows:

   copy the jni.h file from C:\ jdk1.5\include\ to C:\wings\jnidemo\dllproj\

   copy the jni_md.h file from C:\ jdk1.5\include\win32 to C:\wings\jnidemo\dllproj\

5) Right click on dllproj files and select Add Files to Project ->Select the

   NativeImpl.h , jni.h ,jni_md.h files to add it;

6) Copy the DLL created by the VB application to C:\wings\jnidemo\dllproj\

7) Now open the stdafx.h file and add the following content at the top of the

   file: #include "c:\jdk1.5\include\jni.h"

```
#include "c:\wings\jnidemo\NativeImpl.h"

#import "c:\wings\jnidemo\project1.dll"

using namespace Project1;
```

8) Open the dllproj.cpp file and add the following content at end of the file

```
  #include <jni.h>

 JNIEXPORT void JNICALL Java_NativeImpl_add(JNIEnv *env, jobject obj,

jint a, jint b) {

   int num1 = (int)a;

   int num2 = (int)b;

   // Function logic goes here (e.g., print, return a value, etc.)

}
```

```
int num3; jint

ans;

      ::CoInitialize(NULL);

      hresult=CLSIDFromProgID(OLESTR("Project1.Class1"), &clsid);

      hresult=CoCreateInstance(clsid,NULL,CLSCTX_INPROC_SERVER,
      _uuidof(_Class1),(LPVOID*) &t);

      if(hresult == S_OK)

      {

      t->add(a,b);

      }

      else

      {

      AfxMessageBox("Error");

      } }
```

9) Now press F7 to build the dll.

10) Once the dll is built it will be present in the debug directory of the dllproj folder.

Copy the dllproj.dll to C:\wings\jnidemo\

11) Run the java file by using the

command java NativeImpl.

RESULT:

Thus, the Java program for the implementation of DLL creation using VB and deployment in Java was performed successfully, and the output was verified.

4.b	**Programs using Active X DLL**	**3**	**15**

ActiveX Controls

A Microsoft ActiveX control is essentially a simple OLE object that supports the IUnknown interface. This section offers solutions for making a control work well in the Internet environment, with the ultimate goal of delivering optimal quality of service to users. For example, because browser speed is one of the primary factors in users' perception of quality, this section aims to provide solutions that allow an HTML document or page to become visible as soon as possible and interactive very shortly thereafter, while allowing controls to retrieve large data blocks in the background.

AIM

To create a VB application for the implementation of simple calculator using ActiveX DLL.

ALGORITHM

MiniCalc.dll

Step 1: Define the function addition() with the parameters n1, n2 as Integer

Step 2: Calculate addition □ n1+n2

Step 3: Define the function subtraction() with the parameters n1, n2 as Integer

Step 4: Calculate subtraction □ n1 -n2

Step 5: Define the function multiplication() with the parameters n1, n2 as Integer

Step 6: Calculate multiplication □ n1*n2

Step 7: Define the function division() with the parameters n1, n2 as Double

Step 8: Calculate division □ n1/n2

Standard EXE

Step 1: Declare the object o for MINICALLib.MiniCalc

Step 2: Set the object o for MiniCalc

Step 3: Call the addition() function

Step 4: Set the object o for MiniCalc

Step 5: Call the subtraction() function

Step 6: Set the object o for MiniCalc

Step 7: Call the multiplication() function

Step 8: Set the object o for MiniCalc

Step 9: Call the division() function

PROGRAM:

MiniCalc.dll

```
Public Function addition(n1 As Integer, n2 As Integer) As Integer

addition = n1 + n2


End Function

Public Function subtraction(n1 As Integer, n2 As Integer) As Integer

subtraction = n1 - n2

End Function

Public Function multiplication(n1 As Integer, n2 As Integer) As Integer

multiplication = n1 * n2

End Function

Public Function division(n1 As Double, n2 As Double) As Double

division = n1 / n2

End Function
```

Standard EXE

```
Dim o As New MINICALLib.MiniCalc

Private Sub Command1_Click()

Set o = New MiniCalc
```

```
Text3.Text = o. addition(Val(Text1.Text), Val(Text2.Text))

End Sub

Private Sub

Command2_Click() Set o =

New MiniCalc

Text3.Text = o. subtraction(Val(Text1.Text),

Val(Text2.Text)) End Sub

Private Sub

Command3_Click() Set o =

New MiniCalc

Text3.Text = o. multiplication(Val(Text1.Text), Val(Text2.Text))

End Sub

Private Sub

Command4_Click() Set o =

New MiniCalc

Text3.Text = o. division(Val(Text1.Text), Val(Text2.Text))

End Sub
```

RESULT:

Thus the VB application for simple calculator using AciveX DLL was performed and the output was verified.

EXP: 5	**CORBA**		
5.a	**Naming Services in CORBA**	**3**	**18**

AIM:

To write a Java program for the implementation of celcius to farenheit using CORBA.

ALGORITHM:

SERVER SIDE:

```
Step 1: Start
Step 2: Define the class Tempcalcserver
Step 3: Create & Initialize the objects orb, tempImpl, objRef, ncRef, path[] in try
Step 4: Connect the object tempImpl
Step 5: Rebind the objects path, tempImpl
Step 6: Make ready the Tempconv server
Step 7: Call the method printStackTrace() in catch
Step 8: Stop
```

CLIENT SIDE:

```
Step 1: Start
Step 2: Define the class Tempcalcclient
Step 3: Declare the objects din, orb, ncRef path[], calc in try
Step 4: Read the celcius
Step 5: Display the temperature in farenheit by calling the method
celcius() Step 6: Call the method printStackTrace() in catch
Step 7: Stop
```

SOURCE CODE:

Tempcalc.idl

```
module Tempconvcalc
{

interface Tempcalc
{
double get_celcius(in string symbol);
};
};
```

TempcalcImp.java

```
import org.omg.CosNaming.*;
import org.omg.CORBA.*;
import Tempconvcalc.*;
public class TempcalcImp extends _TempcalcImplBase
{
public double get_celcius(String symbol)
{
double celcius=Double.parseDouble(symbol);
double farenheit=1.8*celcius+32;
return farenheit;
}
public TempcalcImp()
{
super();
}
}
```

Tempcalcclient.java

```
import org.omg.CosNaming.*;
import org.omg.CORBA.*;
import Tempconvcalc.*;
import java.io.DataInputStream;
public class Tempcalcclient
{
public static void main(String args[])
{
try
{
DataInputStream din=new DataInputStream(System.in);
ORB orb=ORB.init(args,null);
NamingContext
ncRef=NamingContextHelper.narrow(orb.resolve_initial_references("NameService"));
NameComponent path[]={new NameComponent("TEMPCALC","")};
Tempcalc calc=TempcalcHelper.narrow(ncRef.resolve(path));
System.out.println("\nEnter the celcius:");
String cel=din.readLine();
System.out.println("\nTemparature of the city in farenhit is "+calc.get_celcius(cel));
}
catch(Exception e)
{
e.printStackTrace();
}
}
}
```

Tempcalcserver.java

```
import
org.omg.CosNaming.*;
import org.omg.CORBA.*;
import Tempconvcalc.*;
public class Tempcalcserver
{
public static void main(String args[])
{
try
{
ORB orb=ORB.init(args,null);
TempcalcImp tempImpl=new TempcalcImp();
orb.connect(tempImpl);
org.omg.CORBA.Object
objref=orb.resolve_initial_references("NameService"); NamingContext
ncRef=NamingContextHelper.narrow(objref); NameComponent nc=new
NameComponent("TEMPCALC",""); NameComponent path[]={nc};
ncRef.rebind(path,tempImpl);
System.out.println("Tempconv server is
ready"); Thread.currentThread().join();
}
catch(Exception e)
{
e.printStackTrace();
}
}
}
```

RESULT:

Thus the Java program for the conversion of celcius to farenheit was performed and the output was verified.

5.b	**DSI, DII in CORBA using Visibroker and Java.**	**3**	**21**

AIM:

To write a Java application for bank accounts using visibroker.

ALGORITHM:

SERVER:

Step 1:Start
Step 2:Create the Bank folder using Bank.idl module program
Step 3:Initialize ORB
Step 4:Create the account manager to link
client Step 5:Compile and run the server
program Step 6:Display the accounts created
Step 7:Stop

CLIENT:

Step 1:Start
Step 2:Initialize ORB
Step 3:Create the account manager to link server
Step 4:Requesting server the account manager to open a named
account Step 5:Display the balance of the account’s holder
Step 6:Stop

SOURCE CODE:

MODULE:

```
module Bank {
interface Account
{

float balance();
};
Interface AccountManager {
Account open(in string name);
};
};
```

IMPLEMENTATION:

```
Public class AccountImpl extends Bank._AccountImplBase
{ Public AccountImpl(float balance) {
_balance =balance;
}
Public float balance() {
Return_balance;
}
Private float_balance;
}
```

SERVER:

```
Public class Server {
Public static void main(String[] args) {
Org.omg.CORBA.ORB orb=org.omg.CORBA.ORB.init(args,null);
Org.omg.CORBA.BOA boa=((com.visigenic.vbroker.orb.ORB)orb).BOA_init();
Bank.AccountManager=new AccountManagerImpl("BankManager");
Boa.obj_is_ready(manager);
System.out.println(manager + "is ready.");
Boa.impl_is_ready();
}
}
```

CLIENT:

```
   Public class Client {
   Public static void main(String[] args) {
Org.omg.CORBA.ORB orb=org.omg.CORBA.ORB.init(args,null);
Bank.AccountManager manager=Bank.AccountManagerHelper.bind(orb,"BankManager");
String name=args.length>0?args[0]:"Jack B.Quick";
Bank.Account account=manager.open(name);
float balance=account.balance();
System.out.println("The balance in"+name+""'s account is $"+balance);
}
}
```

RESULT:

Thus the program for bank accounts using visibroker was performed and the output was verified.

5.c	**SIMPLE APPLICATION IN CORBA**	**3**	**21**
	STOCK APPLICATION IN CORBA		

AIM

To write a Java program for the implementation of stock application using CORBA.

ALGOR ITHM

SERVER SIDE:

```
Step 1: Start
Step 2: Define the class StockMarketServer
Step 3: Initialize the objects orb, StockMarketImpl, objRef, ncRef, nc, path[] in try
Step 4: Connect the object StockMarketImpl
Step 5: Rebind the objects path, StockMarketImpl
Step 6: Make ready the StockMarket server
Step 7: Call the method printStackTrace() in catch
Step 8: Stop
```

CLIENT SIDE

```
Step 1: Start
Step 2: Define the class StockMarketClient
Step 3: Initialize the objects din, orb, ncRef , path[], market in
try Step 4: Read the company name
Step 5: Display the price of the company by calling the method get_price()
Step 6: Call the method printStackTrace() in catch
Step 7: Stop
```

SOURCE CODE

StockMarket.idl

```
module SimpleStocks
{

  interface StockMarket
  {
    float get_price(in string symbol);
  };
};
```

StockMarketImpl.java

```
import org.omg.CosNaming.*;
import org.omg.CORBA.*;
import SimpleStocks.*;
public class StockMarketImpl extends _StockMarketImplBase
{
    public float get_price(String symbol)
    {
        float price=0;
        if(symbol.equals("TCS"))
            price=1000;
        else if(symbol.equals("CTS"))
            price=800;
        else if(symbol.equals("INFOSYS"))
            price=1200;
        else if(symbol.equals("WIPRO"))
            price=900;
        else
            price=0;
        return price;
    }
    public StockMarketImpl()
    {
        super();
    }
}
```

StockMarketServer.java

```
import org.omg.CosNaming.*;
import org.omg.CORBA.*;
import SimpleStocks.*;
public class StockMarketServer
{
    public static void main(String args[])
    {
        try
        {
            ORB orb=ORB.init(args,null);
            StockMarketImpl stockMarketImpl=new StockMarketImpl();
            orb.connect(stockMarketImpl);
            org.omg.CORBA.Object
            objRef=orb.resolve_initial_references("NameService"); NamingContext
            ncRef=NamingContextHelper.narrow(objRef); NameComponent nc=new
            NameComponent("NASDAQ","");
            NameComponent path[]={nc};
            ncRef.rebind(path,stockMarketImpl);
            System.out.println("Stock market server is
            ready"); Thread.currentThread().join();
        }
        catch(Exception e)
        {
            e.printStackTrace();
        }
    }
}
```

StockMarketClient.java

```
import
org.omg.CosNaming.*;
import org.omg.CORBA.*;
import SimpleStocks.*;
import java.io.DataInputStream;
public class StockMarketClient
{
    public static void main(String args[])
    {
        try
        {
            DataInputStream din=new DataInputStream(System.in);
            ORB orb=ORB.init(args,null);
            NamingContext
ncRef=NamingContextHelper.narrow(orb.resolve_initial_references("NameService"));
NameComponent path[]={new NameComponent("NASDAQ","")};
StockMarket market=StockMarketHelper.narrow(ncRef.resolve(path));
System.out.println("\nEnter the company name: ");
String company=din.readLine();
System.out.println("\nPrice of my company is: " +market.get_price(company));}
        catch(Exception e)
        {
            e.printStackTrace();}}}
```

RESULT:

Thus the Java program for the implementation of stock application was performed and the output was verified.

6	**IIOP**		
6.a	Inter ORB in communication [IIOP,IOR] Java ORB & Visi broker ORB	3	27

AIM

To write a RMI IIOP program to inoperable with CORBA object.

PROCEDURE

SERVER SIDE

1. Import statement import javax.rmi.PortableRemoteObject; import javax.naming.InitialContext;
2. Implementation class of a remote object public class MyObjectImpl
extends PortableRemoteObject implements MyObject
3. Name registration of a remote object InitialContext.rebind("Robocop",ObjRef);
4. Generate a tie for IIOP with rmic -iiop
5. Run tnameserv.exe as a name server
6. Generate IDL with rmic -idl for CORBA clients

CLIENT SIDE

1. Import statement

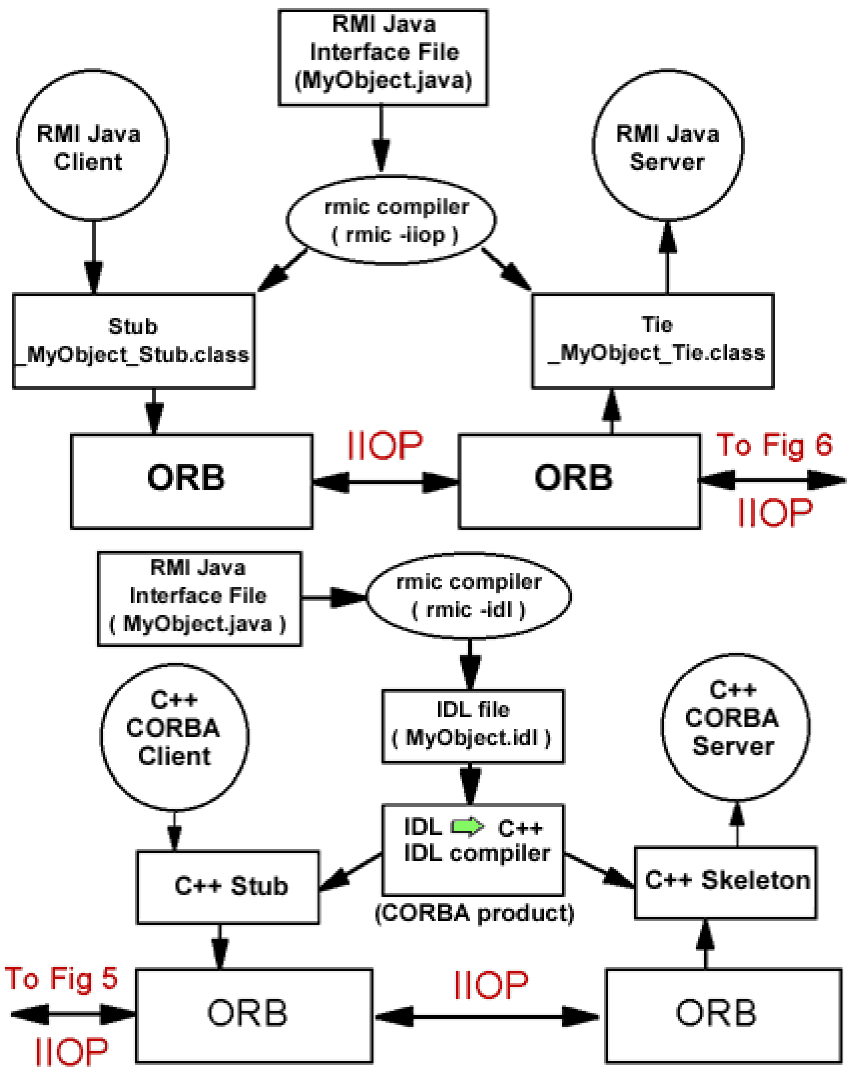
import javax.rmi.PortableRemoteObject;
import javax.naming.InitialContext;

2. Identify a remote object by name

InitialContext IC = new
InitialContext(env); Object obj =
IC.lookup("Robocop");
MyObject myobj = (MyObject)PortableRemoteObject.narrow(obj,MyObject.class);

3. Generate a stub for IIOP with rmic -iiop

DEVELOPMENT PROCEDURE:


RMI Java Interface File (MyObject.java)
RMI Java Client
rmic compiler (rmic -iiop)
RMI Java Server
Stub _MyObject_Stub.class
Tie _MyObject_Tie.class
ORB
IIOP
ORB
To Fig 6
IIOP
RMI Java Interface File (MyObject.java)
rmic compiler (rmic -idl)
IDL file (MyObject.idl)
C++ CORBA Client
C++ CORBA Server
IDL C++ IDL compiler
(CORBA product)
C++ Stub
C++ Skeleton
To Fig 5
IIOP
ORB
IIOP
ORB

PROGRAM:
RMI-IIOP:

```
//hellointerface.java
import java.rmi.*;
import
java.rmi.Remote;
public interface hellointerface extends Remote
{
  public void sayhello(String from)throws RemoteException;
```

```
}
//helloimpl.java
import
java.rmi.*;
import javax.rmi.PortableRemoteObject;
public class helloimpl extends PortableRemoteObject implements hellointerface
{
  public helloimpl() throws RemoteException
   {
     super();
   }
  public void sayhello(String from)throws RemoteException
   {
     System.out.println("hello from"+from);
   }
}

//helloclient.java
import java.rmi.RemoteException;
import java.net.MalformedURLException;
import java.rmi.NotBoundException;
import javax.rmi.*;
import java.util.Vector;
import
javax.naming.NamingException;
import javax.naming.InitialContext;
import javax.naming.Context;
public class helloclient
 {
   public static void main(String args[])throws Exception
    {
      Context ic;
      Object objref;
      hellointerface
      hi; try
       {
        ic=new InitialContext();
        objref=ic.lookup("helloservice");
        System.out.println("Client obtained a ref to server");
        hi=(hellointerface)PortableRemoteObject.narrow(objref,hellointerface.class);
        hi.sayhello("mars");
       }
      catch(Exception e)
       {
         System.out.println("exception:"+e);
         return;
       }
     }
}

//helloserver.java

import javax.naming.InitialContext;
import javax.naming.Context;
public class helloserver
{
public static void main(String args[])
{
  try
   {
     helloimpl helloref=new helloimpl();
     Context InitialNamingContext =new InitialContext();
     InitialNamingContext.rebind("helloservice",helloref)
     ; System.out.println("object is registered");
   }
  catch(Exception e)
  {
    System.out.println("Exception:"+e);
  }
}
}
```

RESULT:

Thus the JAVA RMI IIOP program to inoperable with CORBA object was studied and executed successful.

7	**WEB APPLICATIONS**		
7.a	Deploying components for handli Multimedia files.	3	30

AIM:

To write a Java program that works as a simple calculator. Use a grid layout to arrange buttons for the digits and for the + – * % operations. Add a text field to display the result.

APPLET:

```
//appletcalculator.java
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
public class appletcalculator extends Applet implements ActionListener
{
String m="";
Button Add,Sub,Mul,Div;
TextField opr1,opr2,ans;
public void init()
{
Add=new Button("+");
Sub=new Button("-");
Mul=new Button("*");
Div=new Button("/");
opr1 = new TextField("Enter 1st operand",20);
opr2 = new TextField("Enter 2ndoperand",20);
ans = new TextField(30);
setLayout(new FlowLayout(FlowLayout.LEFT));
add(opr1);
add(opr2);
add(ans);
add(Add);
add(Sub);
add(Mul);
add(Div);
Add.addActionListener(this);
Sub.addActionListener(this);
Mul.addActionListener(this);
Div.addActionListener(this);
opr1.addActionListener(this
);
opr2.addActionListener(this
);
ans.addActionListener(this);
}
public void actionPerformed(ActionEvent e)
{
String s=e.getActionCommand();
int op1;
int op2;
float result;
Stringtemp;
if(s.equals("+"))
m="You have pressed Add";
op1 = Integer.parseInt(opr1.getText());
op2 = Integer.parseInt(opr2.getText());
result = (float)op1+op2;
temp = "The ans is"+result;
ans.setText(temp);}
elseif(s.equals("- ")){
m="You  have  pressed
Sub";
op1 = Integer.parseInt(opr1.getText());
op2 = Integer.parseInt(opr2.getText());
result = (float)op1-op2;
temp = "The ans is"+result;
ans.setText(temp);}
elseif(s.equals("*"))
op1 = Integer.parseInt(opr1.getText());
op2 = Integer.parseInt(opr2.getText());
result = (float)op1*op2;
temp = "The ans is"+result;
ans.setText(temp);}
else
{
m="You have pressed Div";
op1 = Integer.parseInt(opr1.getText());
op2 = Integer.parseInt(opr2.getText());
result = (float)op1/op2;
temp = "The ans is"+result;
ans.setText(temp);}
repaint();}
```

RESULT

Thus a Java program that works as a simple calculator, Using a grid layout to arrange buttons for the digits and for the + – * % operations was executed successfully.

7.b	**Deploying components for e-Busin applications** **.**	**3**	**33**

AIM

To study JDBC connectivity with MS ACCESS to store application data from java component to a DATABASE

PROCEDURE:

JDBC Connectivity with MS-Access

database connectivity with MS-Access is done by creating DataSourceName(dsn) in this example*/
/* Steps to use this example:
* go to ms-access and make a table called "student_base" and give it a file name student_base.mdb
* 1. Go to Control Panel

1. Click on Administrative Tools(windows 2000/xp), Click on ODBC(win98)
2. click on ODBC
3. Then , you will see a ODBC dialog box. Click on UserDSn
4. Click on Add Button
5. Select Microsoft Access Driver(*.mdb) driver and click on finish
6. Give a Data Source Name : student_base
7. Then Click on Select
8. Browse on the database name you have created and

click it:student_base.mdb is a database file where all data
will be stored

9. Click on OK.

Once the DSN is created, you can do this example*/

PROGRAM:

```
Import java.sql.*;
Import java.io.*;
Class jdbcdb
{
Public static void main(string a[])
{
Connection
con; Statement
st;
Resultset
rs; Int x;
Class.forName("sun.jdbc.odbc.jdbcodbcDriver");
Con=DriverManager.getconnection("jdbc:odbc:student"
) St=con.create Statement( );
X=st.executequery("select * from stu");
While(rs.next( ))
{
system.out.println(rs.getsting("no")+"\t"+rs.getstring("name"));
}
}
}
```

RESULT

Thus the JDBC connectivity with MS ACCESS to store application data from java component to a database was studied and executed successfully.

8	**EJB AND CORBA**		
8.a	Distributed objects deployment-EJB and CORBA.	3	42

AIM

To implement distributed object using EJB and CORBA.

PROCEDURE:

STEP1: Develop the CORBA client

To develop the CORBA client, we'll perform the following steps:

Generate the IDL from session bean home and component remote interfaces

Simplify the generated IDL

Compile the IDL interfaces to the corresponding programming language -- C++ in our case -- to generate stubs and other necessary mappings

Find out how to use JNDI (Java Naming and Directory Interface) as a CORBA naming service

Develop the C++ client and the support for value types

STEP 2: Build the client

Generate the IDL

To generate the IDL from Java interfaces, you can use any tool that supports the Java Language Mapping to OMG IDL specification. Examples include:

rmic compiler, using the -idl option, bundled with Java SDK 1.3 and higher

java2idl, bundled with VisiBroker

rmic and ejbc (using -idl) compilers, bundled with BEA WebLogic

First, we have to set WebLogic's environment variables. Then, we can use the
following command:

```
java weblogic.ejbc eai.jar eaiC.jar -idl -idlDirectory ./idl
```

The ejbc compiler creates IDL files in the ./idl directory, generating 17 IDL files.

Simplify the generated IDL

Compiling and implementing all the generated interfaces and value types in C++ is time consuming. However, we won't need all the generated interfaces because we won't use certain methods. Therefore, we can save a lot of work by manually simplifying the generated IDLs.

For this first example, suppose we don't invoke methods on the EJBObject and EJBHome interfaces, and we don't need the EJBMetaData. We won't need a specialized RemoveException, so we can leave out the RemoteException and RemoveEx IDL interfaces. We can also eliminate all the IDL interfaces: java.io, java.lang, and java.rmi.

To simplify the IDL even further, we can merge the IDL interfaces into one file. Therefore, after simplifying the IDL, we get the following file, named CorbaEai.idl:

```
#include "orb.idl"

module javax {

module ejb {

    valuetype CreateException {

        #pragma ID CreateException
"RMI:javax.ejb.CreateException:FD98A9711F66DF7F:575FB6C03D49AD6
A"

    };

  };

};

module javax {

  module ejb {

    exception CreateEx {

        CreateException value;

    };

  };

};

module eai {

  interface CorbaEai

  {
    long sum( in long arg0, in long arg1);

    #pragma ID CorbaEai "RMI:eai.CorbaEai:0000000000000000"

   };

};

module eai {

  interface CorbaEaiHome {

    ::eai::CorbaEai create() raises (::javax::ejb::CreateEx);

    #pragma ID CorbaEaiHome "RMI:eai.CorbaEaiHome:0000000000000000"

   };

};
```

STEP 4: Compile the IDL interfaces

Next, we map the IDL interfaces to the client's programming language. We use C++ for the client, so let's use the idl2cpp compiler included with VisiBroker for C++ 4.5. (You can use any IDL-to-C++ compiler, as long as it complies with CORBA 2.3.1 or higher.)

If you look closer at the IDL definition, you'll see that it includes orb.idl. In our example, we will use the orb.idl file that comes with VisiBroker. Note that, in this case, VisiBroker is installed in the /prg/vbroker folder; therefore, we must specify the /prg/vbroker/idl directory in which the orb.idl file resides in our command. We use the -I switch for the idl2cpp compiler.

We force the IDL compiler to use strict code generation with the -strict switch. Such code corresponds to the CORBA specification; otherwise, VisiBroker generates additional methods. We don't need the server side, so we suppress its generation (-no_servant switch). Additionally, we direct that the idl2cpp compiler should implement IDL modules as C++ namespaces (-namespace switch). Finally, we define that the file suffix should be cpp instead of cc, which is VisiBroker's default (-src_suffix switch).

The command line looks like this:

```
idl2cpp -strict -no_servant -namespace -I/prg/vbroker/idl -src_suffix cpp CorbaEai.idl
```

STEP 5: Use JNDI as a CORBA naming service

Before we develop the C++ client, think about how it obtains the initial reference to the session bean home interface. CorbaEaiHome is registered with the application server's (WebLogic in our case) JNDI, so the C++ client must access the JNDI naming service.

Some application servers, like BEA WebLogic, provide an interface compliant with the CORBA naming service to access JNDI. To test this, we use the WebLogic host2ior utility, which outputs the IORs (Interoperable Object References) for the naming service used with IIOP and IIOP/SSL (Secure Socket Layer). For our example's purposes, we are only interested in the nonsecure IIOP.

The CORBA naming service is realized as a CORBA distributed object with a standardized interface, technically the same as other CORBA objects. Therefore, we could use the IOR directly to connect to the JNDI.

STEP 6: Develop the C++ client

We develop a simple C++ client that invokes the sum() method on the stateless session bean CorbaEai. I assume you are familiar with CORBA. You can find details of developing CORBA C++ clients in Professional J2EE EAI.

First, we declare the necessary includes, followed by the main() method:

```
#include "CosNaming_c.hh"

#include "CorbaEai_c.hh"

USE_STD_NS

int main(int argc, char* const* argv)

{
```

Then we initialize the ORB, connect to the naming service, and obtain the initial context:

```
  try {

    CORBA::ORB_var orb = CORBA::ORB_init(argc,

    argv); cout << "Accessing the naming service" << endl;

    CORBA::Object_ptr o = orb->resolve_initial_references("NameService");

    cout << "Getting the naming context" << endl;

    CosNaming::NamingContext_var context = CosNaming::NamingContext::_narrow(o);
```

Next, we create a name in an appropriate format for the CORBA naming service, under which our EJB home interface is stored in the JNDI. Note that we use the JNDI name corba-eai:

```
    CosNaming::Name name;

    name.length(1);

    name[0].id = CORBA::string_dup("corba-eai");

    name[0].kind = CORBA::string_dup("");
```

Next, we resolve the name and narrow the object reference to our session bean's CorbaEaiHome home interface:

```
    cout << "Resolving the home interface" << endl;

    CORBA::Object_var object = context->resolve(name);

    eai::CorbaEaiHome_var ceaihome = eai::CorbaEaiHome::_narrow(object);
```

We now have a reference to the home interface, so we can create a new instance of the session bean and invoke a method on it:

```
    cout << "Creating a new instance" << endl;

    eai::CorbaEai_ptr ceai = ceaihome->create();

    CORBA::Long r = ceai->sum((CORBA::Long)15,(CORBA::Long)20);



    cout << "Result: " << r << endl;

  }

  catch(const CORBA::Exception& e) {

    cout << "Exception: " << e << endl;

  }

  return 0;

}
```

STEP 7: Build and run the client

To build the C++ client, we use Microsoft Visual C++ 5.0 (or higher) command-line tools. Before we start, be sure you have set the environment variables for Visual C++. The easiest way is to use the vcvars32.bat batch file, which you can find in the Visual C++ installation bin subdirectory. Below is the relevant command line to compile and link the C++ client. Again, VisiBroker should be installed in the /prg/vbroker folder:

```
CL /MD /DTHREAD -DWIN32 /GX /DSTRICT /DALIGNED /DVISIBROKER

/DMSVCUSING_BUG /DMSVCNESTEDUSING_BUG -I/prg/vbroker/include -

I/prg/vbroker/include/stubs CorbaEai_c.cpp Client.cpp /link

/out:Client.exe /LIBPATH:/prg/vbroker/lib
```

To run the example, do the following:

Start the application server (WebLogic, in our case)

Deploy the session bean CorbaEai to the application

server

Start the C++ EJB client, using the command line shown below:

```
Client -ORBInitRef

NameService=iioploc://localhost:7001/NameService
```

Notice that in this example the EJB application server resides on the same computer as the client. To enable remote communication, specify an actual name or IP address instead of localhost.

RESULT:

Thus the distributed object using EJB and CORBA was created and executed successfully

9	**Java**		
9.a	Programs in Reflection, Introspection, Cloning Concepts	3	45

Basic java concepts

1. Interfaces 2. Refelction 3. Cloning 4. Serialization

1. Interfaces

In COM , applications can interact with each other and applications can interact with the system through the collection of functions called interfaces. It is a group of functions through which the clients and component objects can communicate.

AIM

To find out the total and average for the given numbers using interface concept in java

PROCEDURE

Step1 : create a interface using the syntax,

Interface interface name

{ Step 2 : give function declaration }

Step3 : Create a class that should implement the interface by using the syntax,

Class classname implements interface name

{ Step 4: Give function definition. }

Step5 : Create a class with main method.

Step6 : Get the number of integers to find out the sum from the user

Step 7 : Get the numbers to sum

Step 8 : Find out the total and

average Step9 : Display the result.

Interface Code:

```
//testinterface.java
import java.io.*;
import
java.lang.*;

interface inter1
{
 public int avg(int a[],int n);
}

class A implements inter1
{
 public int avg(int a[],int n)
 {
  int sum = 0;
  for(int i=0;i<n;i++)
 {
    sum = sum+a[i];
  }
  return sum;
  }
}
class testinterface
{
 public static void main(String args[])throws IOException
 {
 A a = new
 A(); int z;
 int arr[] = new int[100];
 DataInputStream in = new DataInputStream(System.in);
 System.out.println("Enter the number of integers to find sum");
 String str = in.readLine();
 int num = Integer.parseInt(str);
 for(int i=0;i<num;i++) {
     System.out.println("Enter " + (i+1) + " number to find the sum");
     str = in.readLine();
     z = Integer.parseInt(str);
     arr[i] = z;}
 int result = a.avg(arr,num);
System.out.println("the total is " +
result);
 System.out.println("the avg is " + (float)result/num); }}
```

2. REFLECTION

Reflection is the ability to obtain the information about the fields, constructors and methods of a class.

AIM

To retrieve the methods and fields for the java.lang class using the reflection concept in java.

PROCEDURE

Step1: Import the java.lang.reflect.* package.

Step2:Create a class object for java.lang.Integer using forName() method;

Step3: Using the created object.getConstructors , object.getFields(), object.getMethods() , can retrieve the methods and fields.

Step4: Display the available methods and fields for the given input class.

REFLECTION PROGRAM

```
//testreflect.java
import java.lang.reflect.*;

public class testreflect
{
 public static void main(String args[])throws Exception
     {
      try
      {
      Class c = Class.forName("java.lang.Integer");
      String name = c.getName();
      Constructor[] constructors = c.getConstructors();
      Field[] fields = c.getFields();
      Method[] methods = c.getMethods();
      System.out.println(name+" has "+constructors.length+" constructors
      "); for(int i=0;i<constructors.length;i++)

     System.out.println(" "+constructors[i]);

      System.out.println(name+" has "+fields.length+" fields

      "); for(int i=0;i<fields.length;i++)
      System.out.println(" "+fields[i]);

      System.out.println(name+" has "+methods.length+" methods ");

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

        System.out.println(" "+methods[i]);

      }

      catch(ClassNotFoundException e)

      {

      System.out.println("class not found");

      }}}
```

3. SERIALIZATION

AIM

To write a Java program illustrates how to use object serialization and deserialization.

PROCEDURE

1. It begins by instantiating an object of class **MyClass**.
2. This object has three instance variables that are of types**String**, **int**, and **double**.
3. This is the information we want to save and restore.
4. A **FileOutputStream** is created that refers to a file named "serial," and an **ObjectOutputStream** is created for that file stream.
5. The **writeObject()** method of **ObjectOutputStream** is then used to serialize our object. The object output stream is flushed and closed.
6. A **FileInputStream** is then created that refers to the file named "serial," and an **ObjectInputStream** is created for that file stream.
7. The **readObject()** method of **ObjectInputStream** is then used to deserialize our object. The object input stream is then closed.
8. **MyClass** is defined to implement the **Serializable** interface. If this is not done, a **NotSerializableException** is thrown.
9. Try experimenting with this program by declaring some of the**MyClass** instance variables to be **transient**.
10. That data is then not saved during serialization.

SERIALIZATION PROGRAM

```
import java.io.*;
public class SerializationDemo {
public static void main(String args[]) {
// Object serialization
try {
MyClass object1 = new MyClass("Hello", -7, 2.7e10);
System.out.println("object1: " + object1);
FileOutputStream fos = new FileOutputStream("serial");
ObjectOutputStream oos = new ObjectOutputStream(fos);
oos.writeObject(object1);
oos.flush();
oos.close();
}
catch(Exception e) {
System.out.println("Exception during serialization: " + e);
System.exit(0);
}
// Object deserialization
try {
MyClass object2;
FileInputStream fis = new FileInputStream("serial");
ObjectInputStream ois = new
ObjectInputStream(fis); object2 =
(MyClass)ois.readObject();
ois.close();
System.out.println("object2: " + object2);
}
catch(Exception e) {
System.out.println("Exception during deserialization: "
+ e);
System.exit(0);
}
}
}
class MyClass implements Serializable {
String s;
int i;
double
d;
public MyClass(String s, int i, double d) {
this.s = s;
this.i = i;
this.d =
d;
}
public String toString() {
return "s=" + s + "; i=" + i + "; d=" + d; } }
```

RESULT:

Thus Java program illustrates how to use object serialization and deserialization.

4. CLONING

OBJECT CLONING IN JAVA

The object cloning is a way to create exact copy of an object. For this purpose, clone() method of Object class is used to clone an object.

The java.lang.Cloneable interface must be implemented by the class whose object clone we want to create. If we don't implement Cloneable interface, clone() method generates CloneNotSupportedException.

The clone() method is defined in the Object class. Syntax of the clone() method is as follows:
protected Object clone() **throws** CloneNotSupportedException

PROGRAM

CLONING:

```
//clonetest.java

import

java.io.*;

class clonetest implements Cloneable
{
 int w,h;
 clonetest(int w,int h)
 {
  this.w =
  w; this.h =
  h;
 }

 public Object clone()
 {
  try
     {
     return super.clone();
     }
  catch(CloneNotSupportedException e)
     {
     System.out.println("Clone error");
     return this;
     }
  }
  public static void main(String args[])
  {
    clonetest s1 = new clonetest(4,2);
    clonetest s2;
    s2 = (clonetest)s1.clone();
    System.out.println("Before
    change");
    System.out.println("1st width = "+s1.w+" 1st height = "+s1.h);
    System.out.println("2nd width = "+s2.w+" 2nd height = "+s2.h);
    System.out.println("After change");
    System.out.println("2nd width = "+(s2.w*10)+" 2nd height = "+(s2.h*10));
  }
}
```

RESULT:

Thus the Java concepts like Interface, Reflection, Serialization and cloning were studied and implemented successfully.

10	**J2EE**		
10.a	Studying J2EE Server.	2	48

The J2EE platform uses a multitiered distributed application model. Application logic is divided into components according to function, and the various application components that make up a J2EE application are installed on different machines depending on the tier in the multitiered J2EE environment to which the application component belongs. Figure 1-1 shows two multitiered J2EE applications divided into the tiers described in the following list. The J2EE application parts shown in The J2EE platform uses a multitiered distributed application model. Application logic is divided into components according to function, and the various application components that make up a J2EE application are installed on different machines depending on the tier in the multitiered J2EE environment to which the application component belongs. Figure 1-1 shows two multitiered J2EE applications divided into the tiers described in the following list. The J2EE application parts shown in Figure 1-1 are presented in J2EE Components.

- Client-tier components run on the client machine.
- Web-tier components run on the J2EE server.
- Business-tier components run on the J2EE server.
- Enterprise information system (EIS)-tier software runs on the EIS server.

Although a J2EE application can consist of the three or four tiers shown in Figure 1-1, J2EE multitiered applications are generally considered to be three-tiered applications because they are distributed over three different locations: client machines, the J2EE server machine, and the database or legacy machines at the back end. Three-tiered applications that run in this way extend the standard two-tiered client and server model by placing a multithreaded application server between the client application and back-end.

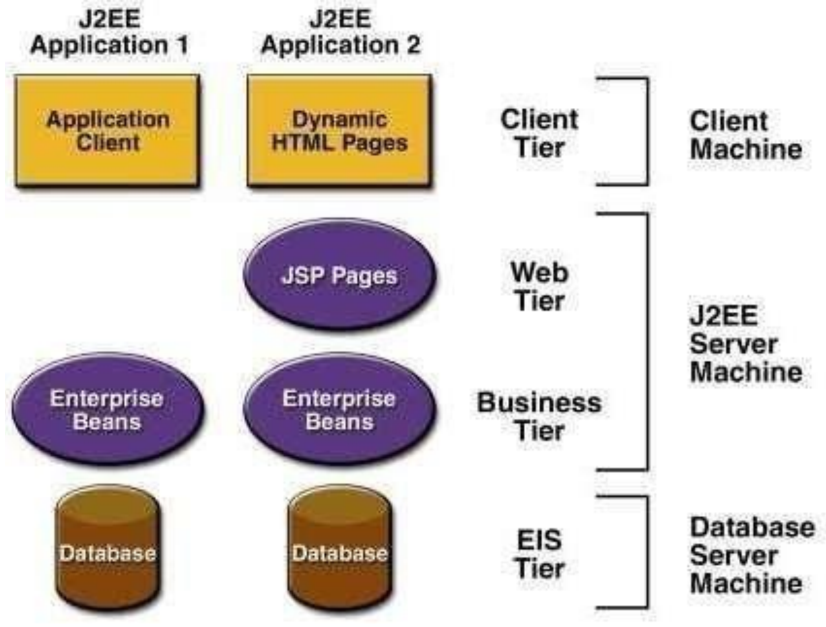


Figure 1-1 Multitiered Applications J2EE Components

J2EE applications are made up of components. A *J2EE component* is a self-contained functional software unit that is assembled into a J2EE application with its related classes and files and that communicates with other components. The J2EE specification defines the following J2EE components:

- Application clients and applets are components that run on the client.
- Java Servlet and JavaServer Pages (JSP) technology components are Web components that run on the server.
- Enterprise JavaBeans (EJB) components (enterprise beans) are business components that run on the server.

J2EE components are written in the Java programming language and are compiled in the same way as any program in the language. The difference between J2EE components and "standard" Java classes is that J2EE components are assembled into a J2EE application, verified to be well formed and in compliance with the J2EE specification, and deployed to production, where they are run and managed by the J2EE server.

J2EE Clients

A J2EE client can be a Web client or an application client.

Web Clients

A Web client consists of two parts: dynamic Web pages containing various types of markup language (HTML, XML, and so on), which are generated by Web components running in the Web tier, and a Web browser, which renders the pages received from the server.

A Web client is sometimes called a *thin client*. Thin clients usually do not do things like query databases, execute complex business rules, or connect to legacy applications. When you use a thin client, heavyweight operations like these are off-loaded to enterprise beans executing on the J2EE server where they can leverage the security, speed, services, and reliability of J2EE server- side technologies.

Applets

A Web page received from the Web tier can include an embedded applet. An applet is a small client application written in the Java programming language that executes in the Java virtual machine installed in the Web browser. However, client systems will likely need the Java Plug-in and possibly a security policy file in order for the applet to successfully execute in the Web browser.

Web components are the preferred API for creating a Web client program because no plug-ins or security policy files are needed on the client systems. Also, Web components enable cleaner and more modular application design because they provide a way to separate applications programming from Web page design. Personnel involved in Web page design thus do not need to understand Java programming language syntax to do their jobs.

Application Clients

A J2EE application client runs on a client machine and provides a way for users to handle tasks that require a richer user interface than can be provided by a markup language. It typically has a graphical user interface (GUI) created from Swing or Abstract Window Toolkit (AWT) APIs, but a command-line interface is certainly possible.

Application clients directly access enterprise beans running in the business tier. However, if application requirements warrant it, a J2EE application client can open an HTTP connection to establish communication with a servlet running in the Web tier.

JavaBeans Component Architecture

The server and client tiers might also include components based on the JavaBeans component architecture (JavaBeans component) to manage the data flow between an application client or applet and components running on the J2EE server or between server components and a database. JavaBeans components are not considered J2EE components by the J2EE specification. JavaBeans components have instance variables and get and set methods for accessing the data in the instance variables. JavaBeans components used in this way are typically simple in design and implementation, but should conform to the naming and design conventions outlined in the JavaBeans component architecture.

J2EE Server Communications

Figure 1-2 shows the various elements that can make up the client tier. The client communicates with the business tier running on the J2EE server either directly or, as in the case of a client running in a browser, by going through JSP pages or servlets running in the Web tier.
Your J2EE application uses a thin browser-based client or thick application client. In deciding which one to use, you should be aware of the trade-offs between keeping functionality on the client and close to the user (thick client) and off-loading as much functionality as possible to the server (thin client). The more functionality you off-load to the server, the easier it is to distribute, deploy, and manage the application; however, keeping more functionality on the client can make for a better perceived user experience.

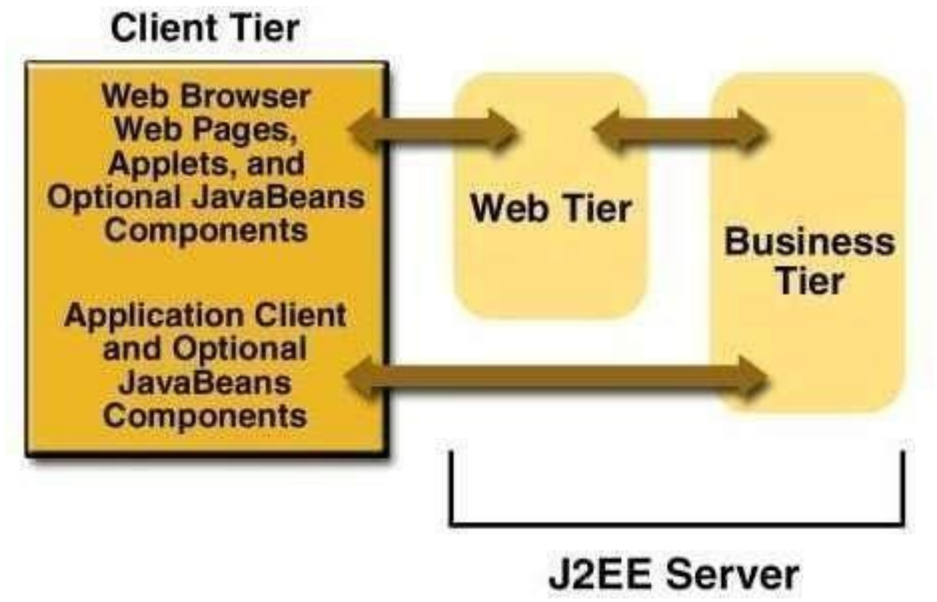

Client Tier
Web Browser Web Pages, Applets, and Optional JavaBeans Components
Application Client and Optional JavaBeans Components
Web Tier
Business Tier
J2EE Server

Figure 1-2 Server Communications

Web Components

J2EE Web components can be either servlets or JSP pages. *Servlets* are Java programming language classes that dynamically process requests and construct responses. *JSP pages* are text-based documents that execute as servlets but allow a more natural approach to creating static content.
Static HTML pages and applets are bundled with Web components during application assembly, but are not considered Web components by the J2EE specification. Server-side utility classes can

also be bundled with Web components and, like HTML pages, are not considered Web components.
Like the client tier and as shown in Figure 1-3, the Web tier might include a JavaBeans component to manage the user input and send that input to enterprise beans running in the business tier for processing.

Business Components

Business code, which is logic that solves or meets the needs of a particular business domain such as banking, retail, or finance, is handled by enterprise beans running in the business tier. Figure 1-4 shows how an enterprise bean receives data from client programs, processes it (if necessary), and sends it to the enterprise information system tier for storage. An enterprise bean also retrieves data from storage, processes it (if necessary), and sends it back to the client program.

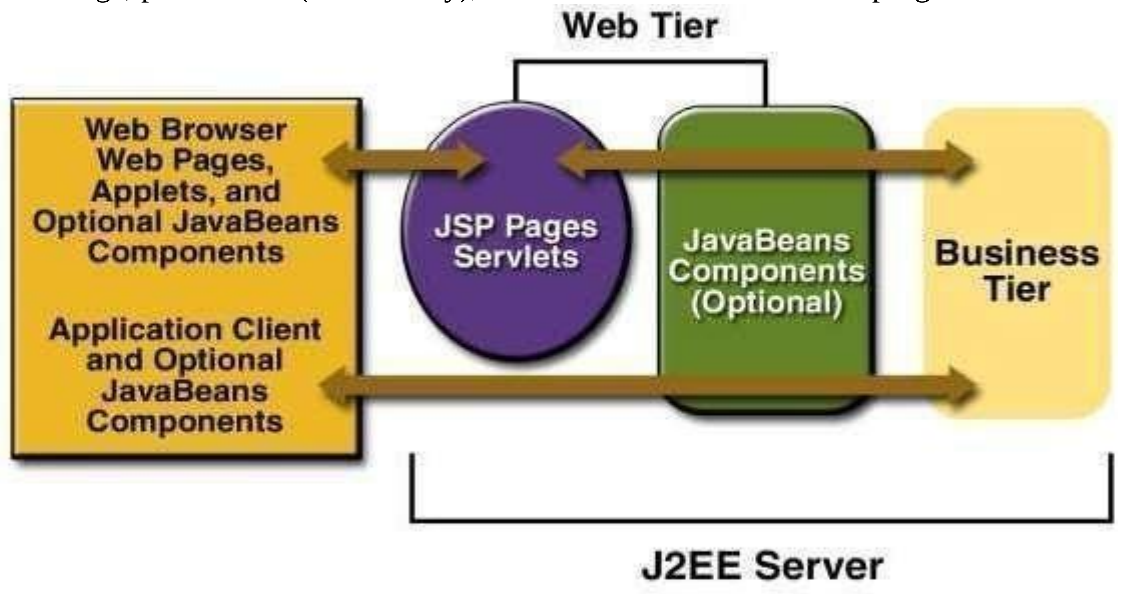

Web Tier
Web Browser Web Pages, Applets, and Optional JavaBeans Components
Application Client and Optional JavaBeans Components
JSP Pages Servlets
JavaBeans Components (Optional)
Business Tier
J2EE Server

Figure 1-3 Web Tier and J2EE Application

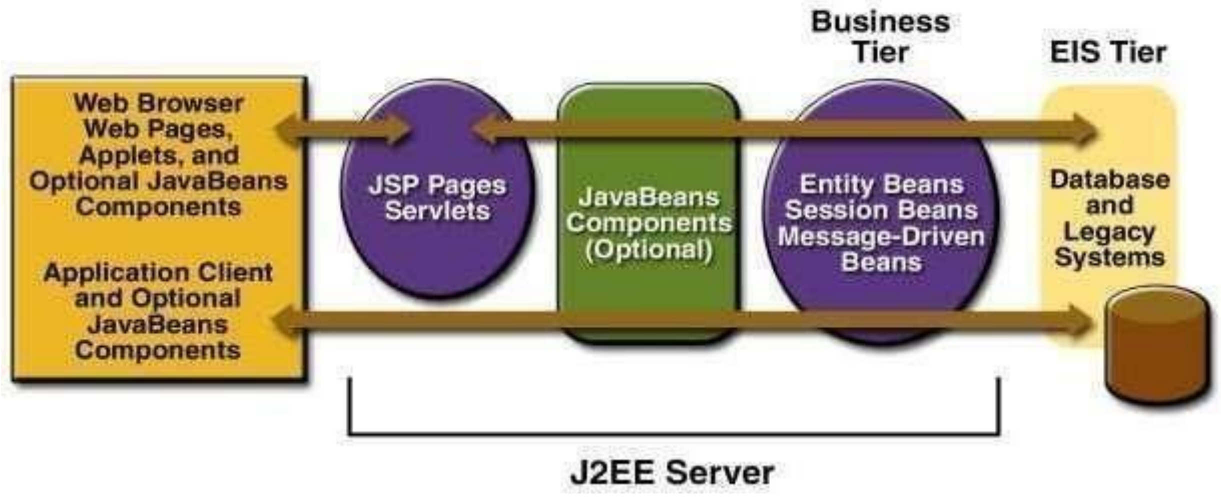

Business Tier
EIS Tier
Web Browser Web Pages, Applets, and Optional JavaBeans Components
Application Client and Optional JavaBeans Components
JSP Pages Servlets
JavaBeans Components (Optional)
Entity Beans Session Beans Message-Driven Beans
Database and Legacy Systems
J2EE Server

Figure 1-4 Business and EIS Tiers

There are three kinds of enterprise beans: session beans, entity beans, and message-driven beans. A *session bean* represents a transient conversation with a client. When the client finishes executing, the session bean and its data are gone. In contrast, an *entity bean* represents persistent

data stored in one row of a database table. If the client terminates or if the server shuts down, the underlying services ensure that the entity bean data is saved.
A *message-driven bean* combines features of a session bean and a Java Message Service ("JMS") message listener, allowing a business component to receive JMS messages asynchronously. This tutorial describes entity beans and session beans.

Result:

Thus the document provides an overview of the J2EE Server Architecture.