Java RMI
Remote Method Invocation

CHITRAKANT BANCHHOR
IT, SCOE, Pune

References

1. Cay S. Horstmann, Core Java, Vol. 2: Advanced Features, 8th Edition,

2. Troy Bryan Downing, Java RMI: Remote Method Invocation,

3. Compiled from various web sites

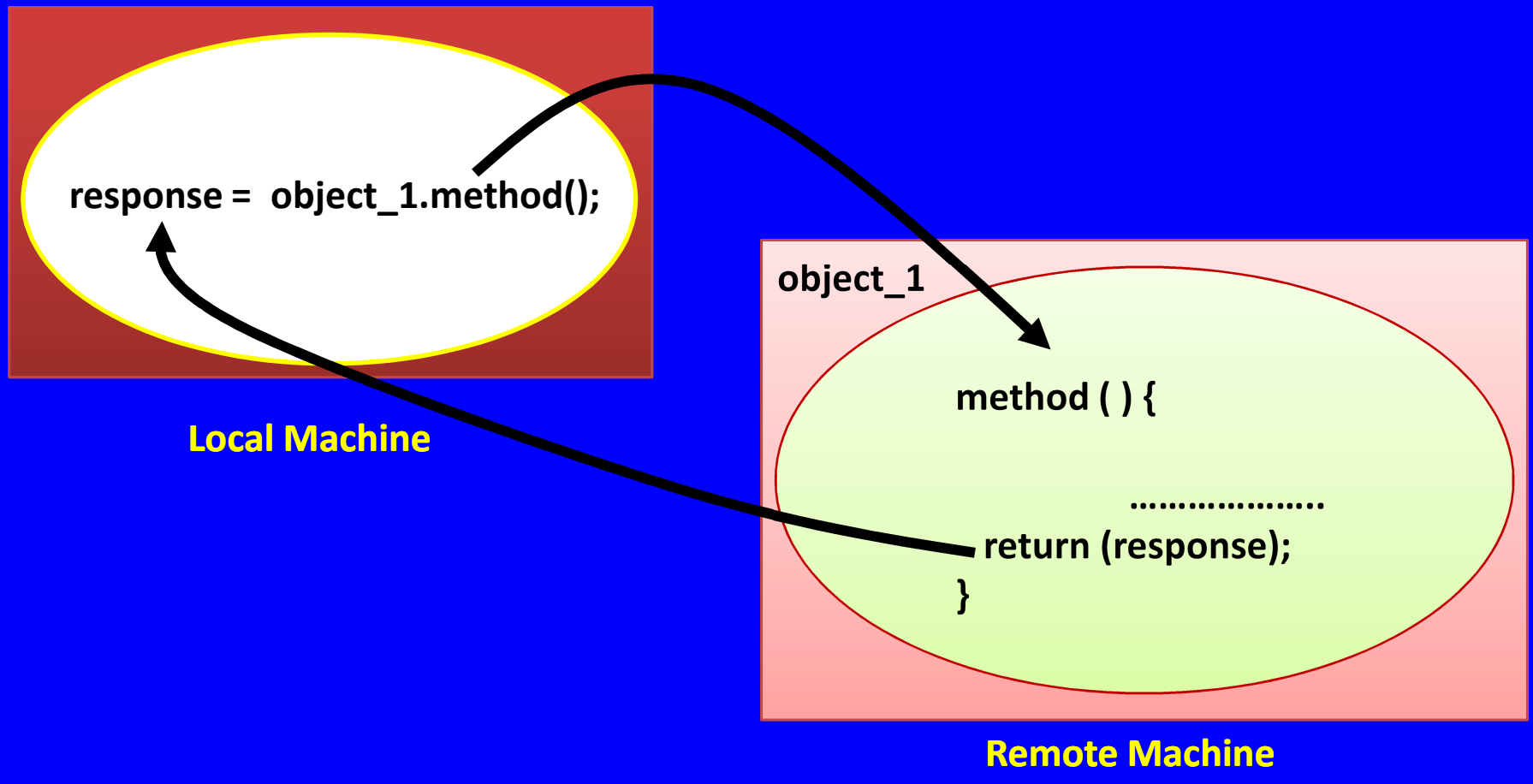
Remote Method Invocation

- RMI system allows an object running in one JVM to invoke methods on an object in another JVM.

- ❖ RMI can communicate only between programs written in Java.

- Example

❖ Assume code running in the local machine holds a remote reference to an object object_1 on a remote machine



Java RMI: main Components

1. Remote objects

- These are normal Java objects
- Extends some RMI inbuilt class that provides support for remote invocation.

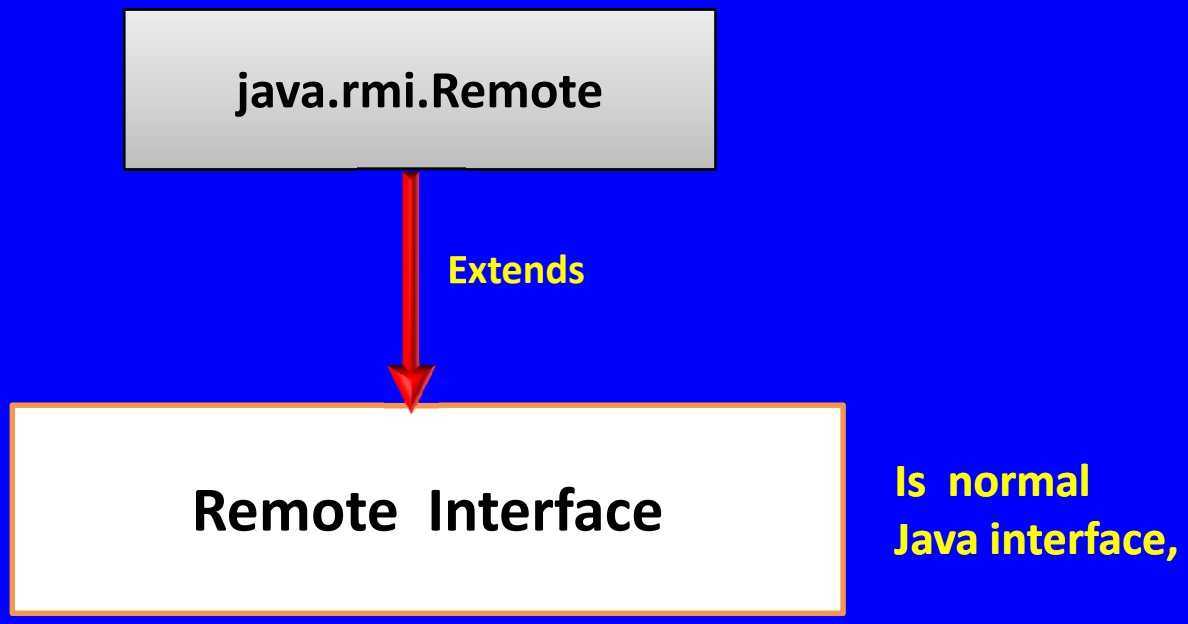
2. Remote reference

- These are object references
- Refer to remote objects, typically on a different computer

3. Remote interfaces

- Normal Java Interfaces that specify “API” of a remote interface
- They should extend java.rmi.Remote interface
- The remote interface must known to local and remote code

The Remote Interface



Is normal Java interface,

- All methods in a remote interface must be declared to throw the java.rmi.RemoteException exception.

- A remote interface contains declarations of remote methods.

```
public interface SumServerIntf extends Remote {

        retType  method_declaration_1()  throws  Exception;

        retType  method_declaration_1()  throws  Exception;

        --  --  --  --  --  --  --  --  --

}
```

- A remote interface must extend the java.rmi.Remote interface.

```
public interface SumServerIntf extends     java.rmi.Remote {

            retType  method_declaration_1()  throws  Exception;

            retType  method_declaration_1()  throws  Exception;

            --  --  --  --  --  --  --  --  --

}
```

- Each method declaration must include java.rmi.RemoteException .
  Or one of its superclasses in its throws clause.

```
public interface SumServerIntf extends java.rmi.Remote {

        retType  method_declaration_1()  throws  RemoteException;

        retType  method_declaration_1()  throws  Exception;

        --  --  --  --  --  --  --  --  --

}
```

```
import java.rmi.*;

public interface SumServerIntf extends Remote {

        int sum(int m, int n) throws RemoteException;

}
```

Figure : Remote interface

java.rmi.Remote

- marker interface
- It declares no methods or fields

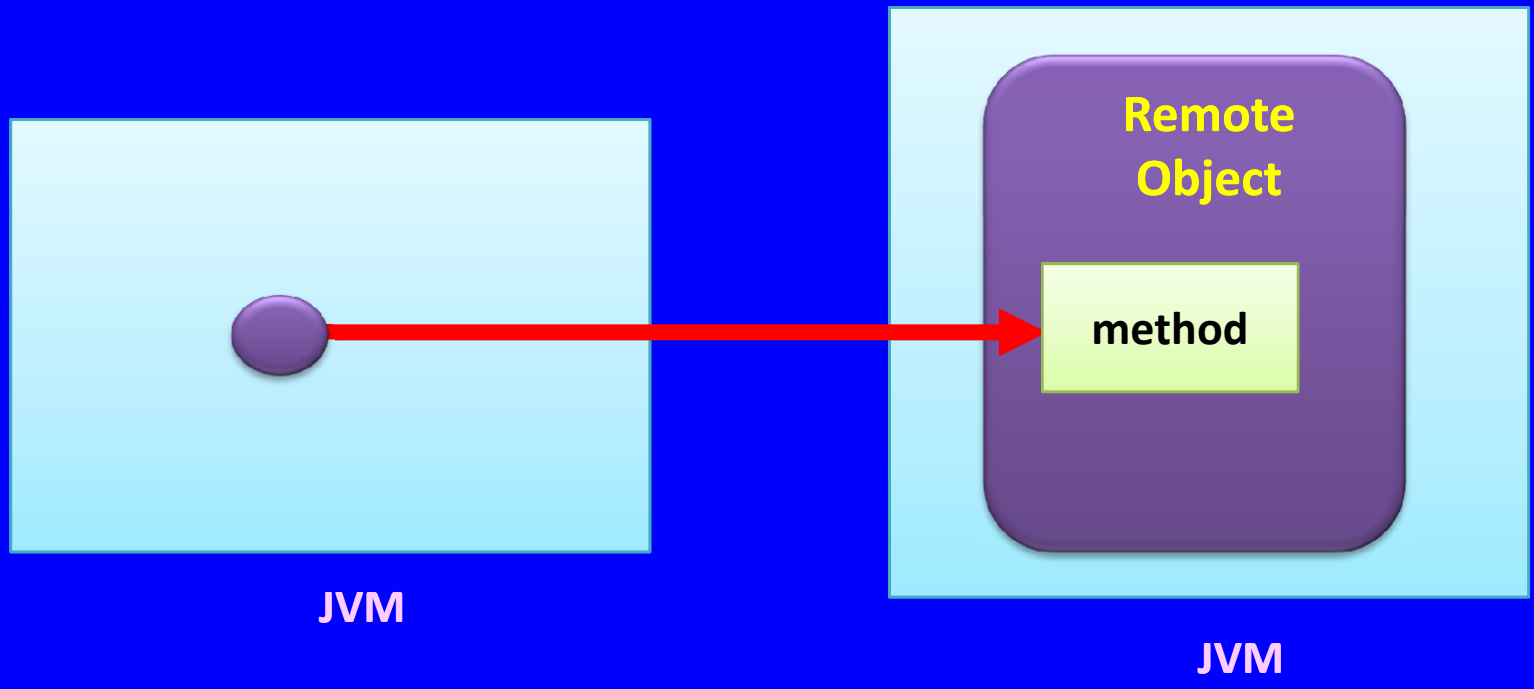
But extending it by another interface tells the RMI system to treat the interface concerned as a remote interface.

java.rmi.RemoteException

- RMI makes remote invocations look syntactically like local invocation.
- In practice, though, it cannot defend from problems unique to distributed computing — unexpected failure of the network or remote machine.

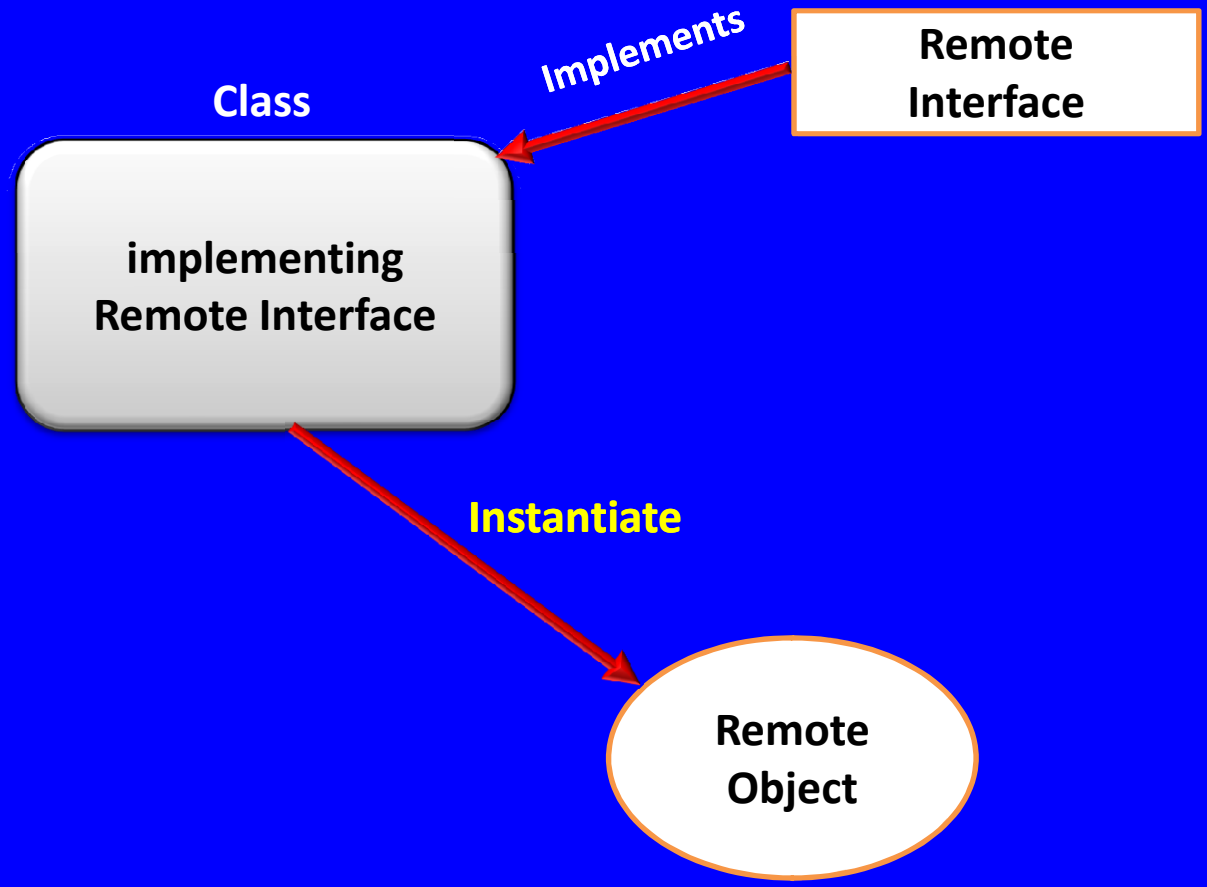
Remote Objects

- A remote object is one whose methods can be invoked from another JVM



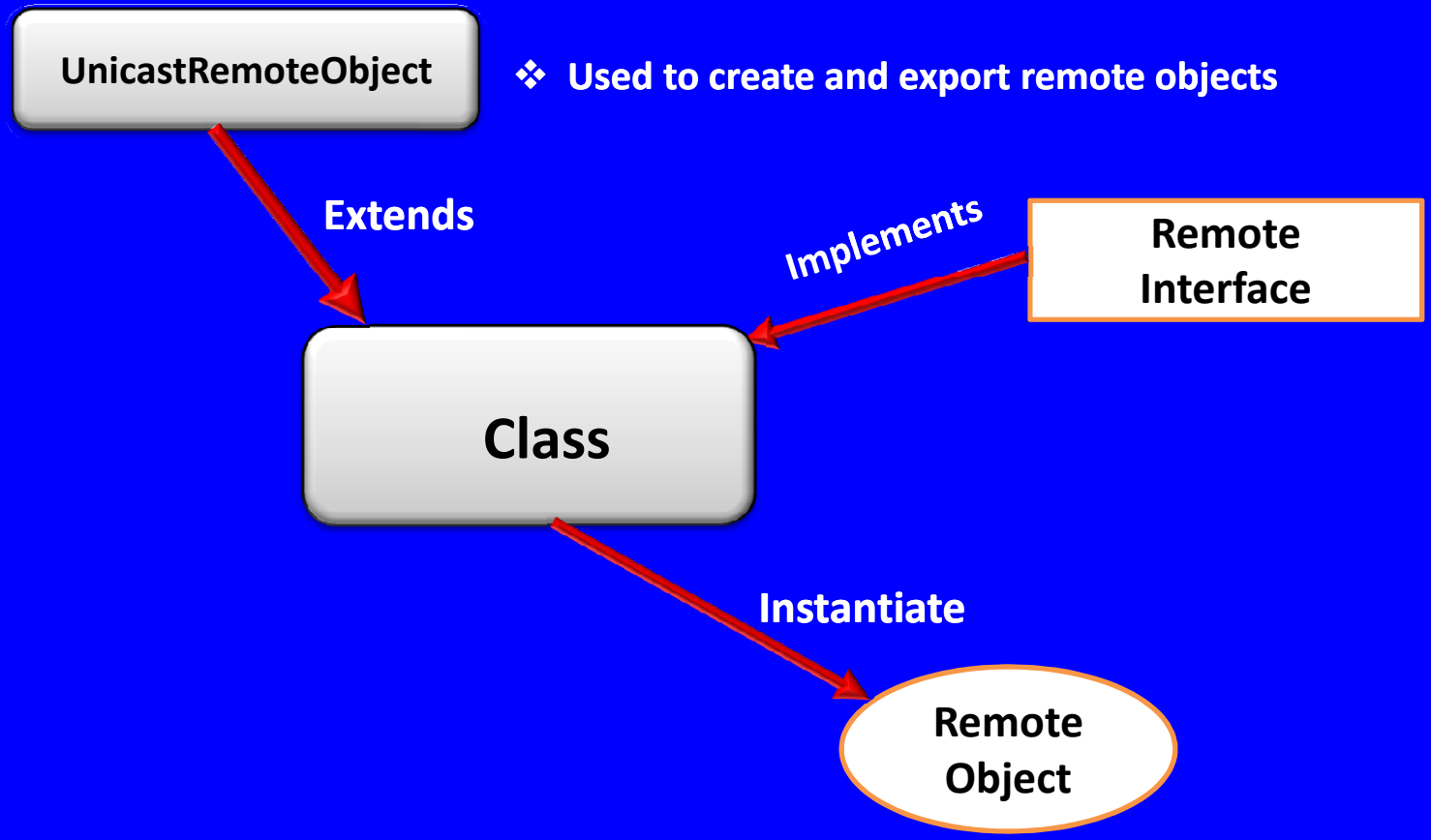
Remote Objects Cont.

- A remote object is an instance of a class that implements a remote interface.



- This class also extends the library class java.rmi.server.UnicastRemoteObject

  - Provides support for point-to-point active object references using TCP streams.



A Remote Object Implementation Class

SumServerImpl.java

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

public class SumServerImpl extends UnicastRemoteObject  implements SumServerIntf {

        public SumServerImpl() throws RemoteException {
        }
        public int sum(int m, int n) throws RemoteException {
                return m + n;
        }
}
```

Remarks

- If there is no explicit constructor invocation in the body of a subclass constructor, it implicitly invokes super()
- Hence the vital constructor of UnicastRemoteObject is called.

```
public SumServerImpl() throws RemoteException {

}
```

No explicit constructor invocation

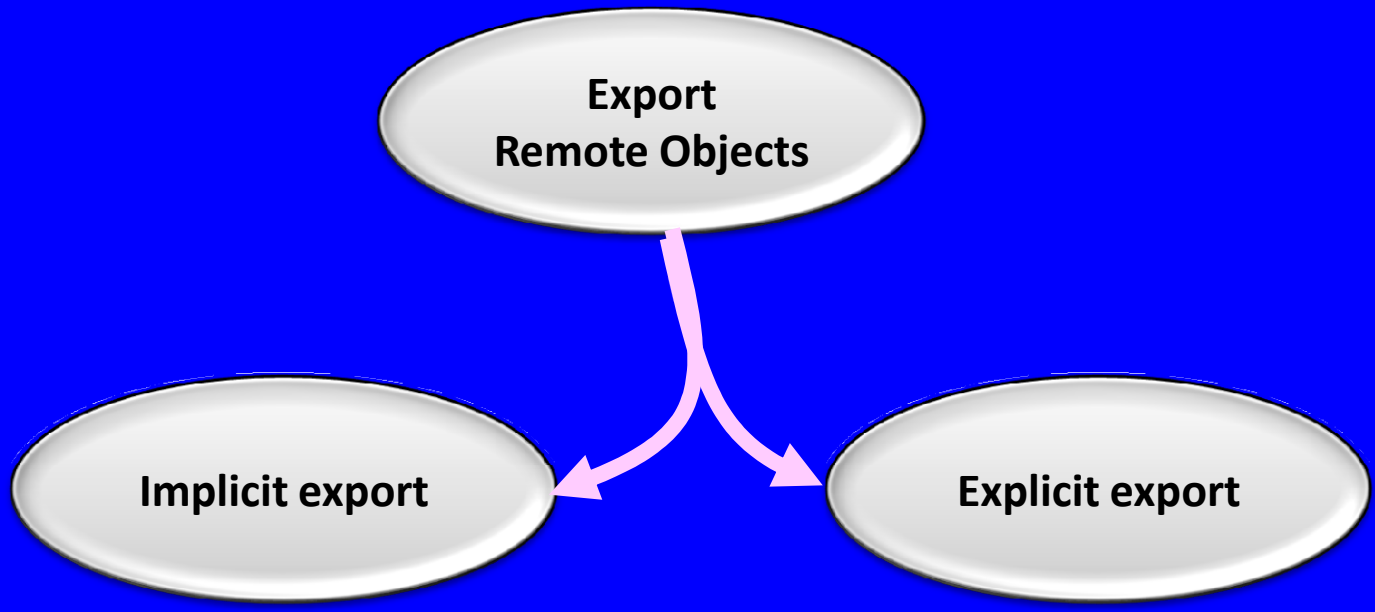
Exporting an Object

- **There are two choices when creating a remote object:**

1. **Extend UnicastRemoteObject class**

Export the object by using functions of UnicastRemoteObject.

UnicastRemoteObject Class



- **Extend UnicastRemoteObject**
  Remote object can be exported

- **Call exportObject():**
  explicitly export

UnicastRemoteObject: methods

1. exportObject
2. unexportObject

1. exportObject

```
public static  RemoteStub  exportObject ( Remote  obj )
```

- Exports the specified remote object to make it available to receive incoming calls, using an anonymous port.

2. exportObject

```
public static  Remote  exportObject ( Remote  obj , int  port )
```

- Exports the specified remote object to make it available to receive incoming calls, using the particular supplied port.

3. exportObject

```
public static  Remote  exportObject ( Remote  obj , int  port,
                                      RMIClientSocketFactory csf,
                                      RMIServerSocketFactory ssf
                                    )
```

- Export the remote object to make it available to receive incoming calls, using a transport specified by the given socket factory.

4. unexportObject

```
public static boolean unexportObject(
                                        java.rmi.Remote  obj,
                                        boolean force
                                    )
```

- This method makes the remote object unavailable for incoming calls.
- If the force parameter is true, the object is forcibly unexported even if there are pending calls to the remote object.

Naming class

- **Interact with the remote object registry: provides several methods .**

1. **bind**
2. **rebind**
3. **unbind**
4. **lookup**
5. **list**

1. bind

- public static void bind (String url, Remote obj)

- This method binds the specified name to a remote object.
- It throws an AlreadyBoundException if the name is already bound to an object.
  - Example: bind ("SUM-SERVER", RemoteObject);

2. rebind

- public static void rebind (String url, Remote obj)

- Rebinds the specified name to a new remote object.
- Any old binding for the name is replaced.

  - Example: rebind ("SUM-SERVER", RemoteObject);

3. unbind

- public static void unbind (String url)

- Removes an object reference from the registry.
- It throws a NotBoundException if there was no binding.

4. list

- public static String[] list (String url)

- Returns an array of the Strings bound in the registry.
- Each String object contains a name that is bound to a remote object.
- The names are URL-formatted strings.

5. lookup

- public static Remote lookup (String url)

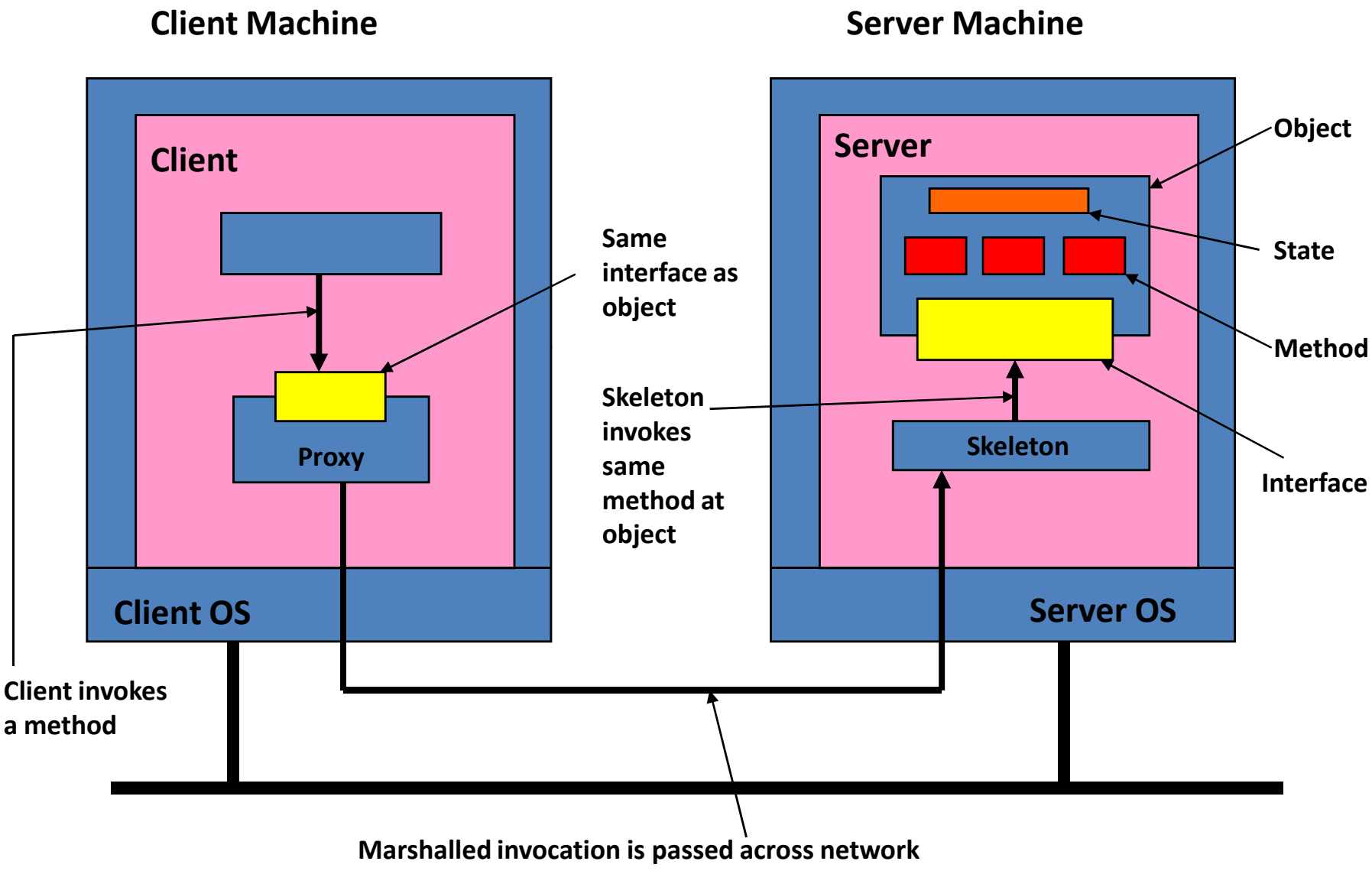
- Returns a stub for the remote object associated with the specified name.
- This method takes a string containing the URL to a remote object reference on the server.

- Example: lookup ("rmi://localhost:1099/SUM-SERVER");

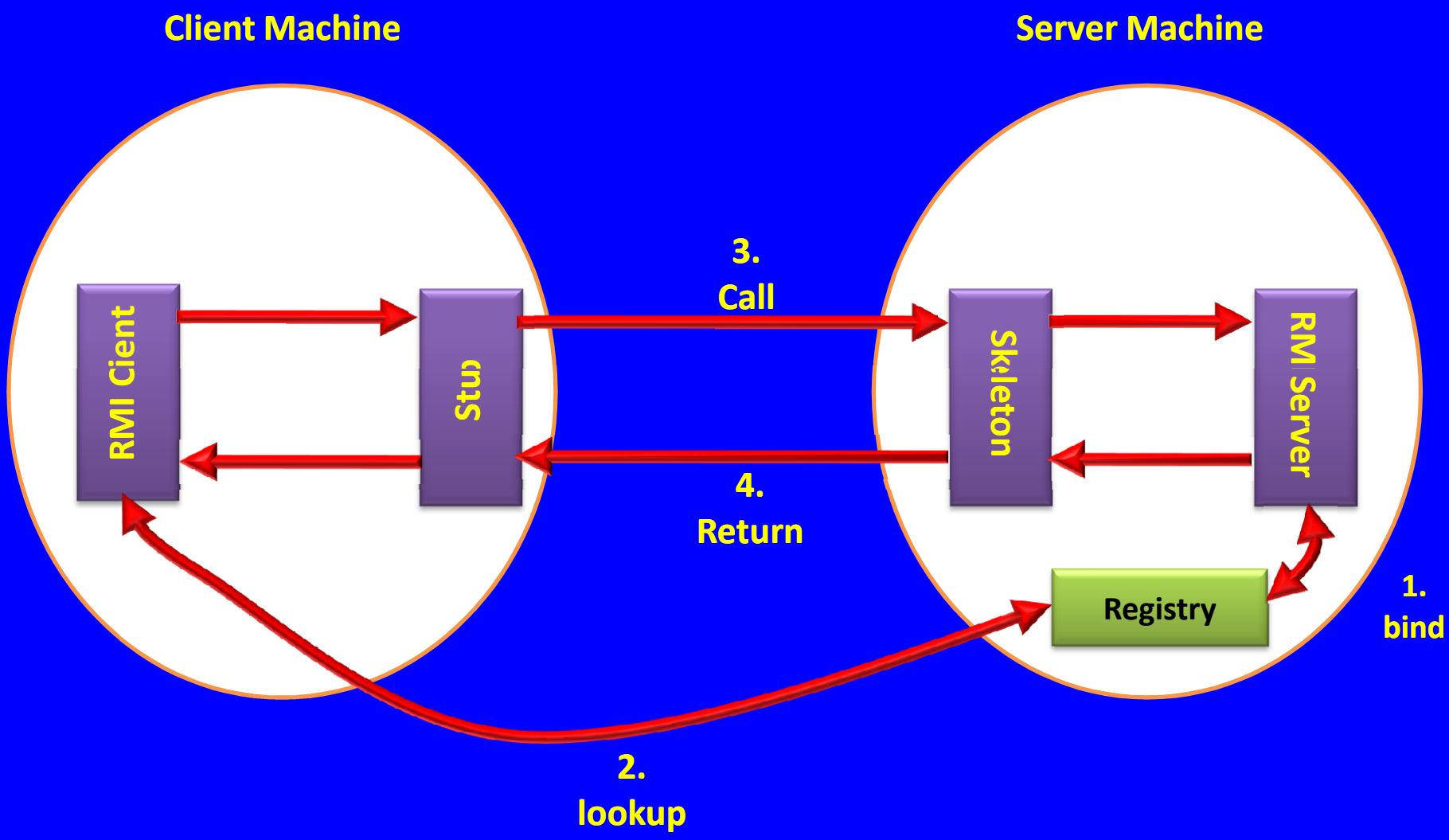
Distributed Objects

- Simple idea – objects existing on one machine (server) may be accessed from another machine through regular method call.

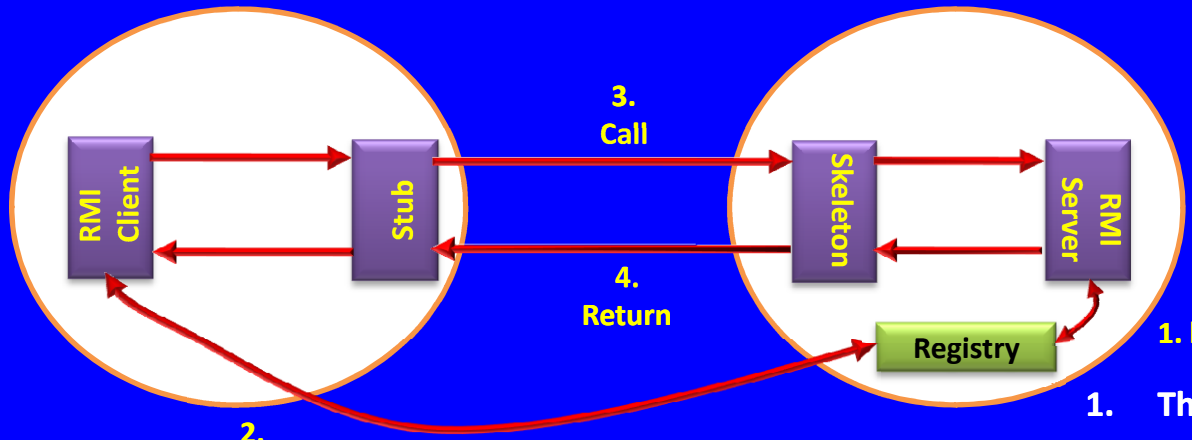
- **The following figure commonly referred as a distributed object.**



The General RMI Architecture



3. Serializing the parameters to skeleton



1. The server must first bind its name to the registry

2. The client lookup the server name in the registry to establish remote references.

Steps for Developing an RMI System

1. Define the remote interface
2. Develop the remote object by implementing the remote interface.
3. Develop the client program.
4. Compile the Java source files.
5. Generate the client stubs and server skeletons.
6. Start the RMI registry.
7. Start the remote server objects.
8. Run the client

1. Define the remote interface

SumServerIntf.java

```
import java.rmi.*;

public interface SumServerIntf extends Remote {
        int sum(int m, int n) throws RemoteException;
}
```

Step 2: Develop the remote object and its interface

- The server is a simple unicast remote server.
- Create server by extending java.rmi.server.UnicastRemoteObject

SumServerImpl.java

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

public class SumServerImpl extends UnicastRemoteObject  implements SumServerIntf {

        public SumServerImpl() throws RemoteException {
        }
        public int sum(int m, int n) throws RemoteException {
                return m + n;
        }
}
```

A Server Program

- The server must bind its name to the registry.
- Use java.rmi.Naming class to bind the server name to registry. In this example the name call “SUM-SERVER”.

SumServer.java

```
import java.net.*;
import java.rmi.*;

public class SumServer {
        public static void main(String args[]) {
                try {
                        SumServerImpl sumServerImpl = new SumServerImpl();
                        Naming.rebind(“SUM-SERVER", sumServerImpl);
                }
                catch(Exception e) {
                        System.out.println("Exception: " + e);
                }
        }
}
```

Step 3: Develop the client program

- Client first look up the name of server in the registry. Use the java.rmi.Naming class to lookup the server name.
- The server name is specified as URL in the form: (rmi://host:port/name)
- Default RMI port is 1099.
- The name specified in the URL must exactly match the name that the server has bound to the registry.
- In this example, the name is “SUM-SERVER”

SumClient.java

```
import java.rmi.*;

public class SumClient {
        public static void main(String args[]) {
                try {
                        String sumServerURL = "rmi://" + args[0] + "/SUM-SERVER";

                        SumServerIntf sumServerIntf =
                                        (SumServerIntf)Naming.lookup(sumServerURL);

                        int m = 5;
                        int n = 6;

                        System.out.println("The sum is: " + sumServerIntf.sum(m, n));
                }
                catch(Exception e) {
                        System.out.println("Exception: " + e);
                }
        }
}
```

SumClient.java

```
import java.rmi.*;

public class SumClient {
        public static void main(String args[]) {
                try {
                        String sumServerURL = "rmi://" + args[0] + "/SUM-SERVER";
                        SumServerIntf sumServerIntf =
                                        (SumServerIntf)Naming.lookup(sumServerURL);
                        System.out.println("The first number is: " + args[1]);
                        int m = Integer.valueOf(args[1]);
                        System.out.println("The second number is: " + args[2]);
                        int n = Integer.valueOf(args[2]);
                        System.out.println("The sum is: " + sumServerIntf.sum(m, n));
                }
                catch(Exception e) {
                        System.out.println("Exception: " + e);
                }
        }
}
```

Step 4 : Compile the Java source files

- javac SumServerIntf.java
- javac SumServerImpl.java
- javac SumServer.java
- javac SumClient.java

Step 5 : Generate stub and skeleton

- rmic SumServerImpl

Step 6: Start the RMI registry

- **RMI Registry on the Server Machine**

  - start rmiregistry

- Start a registry a non-default port number:

```
start  rmiregistry  4956 &
```

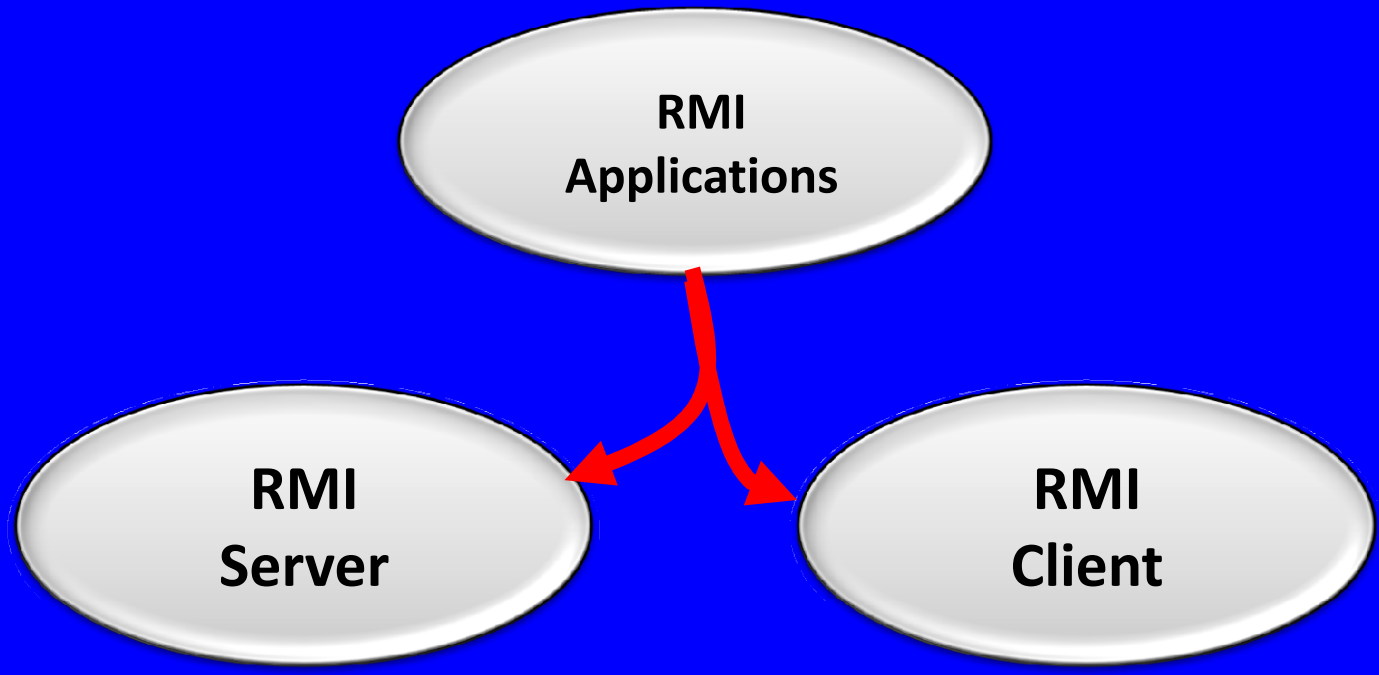
- The Naming calls become, e.g.:

```
Naming.rebind("rmi://localhost:4956/SUM-SERVER",  sumServer) ;

Naming.lookup("rmi:// localhost:4956/SUM-SERVER") ;
```

Steps 7 & 8: Start the remote server objects & Run the client

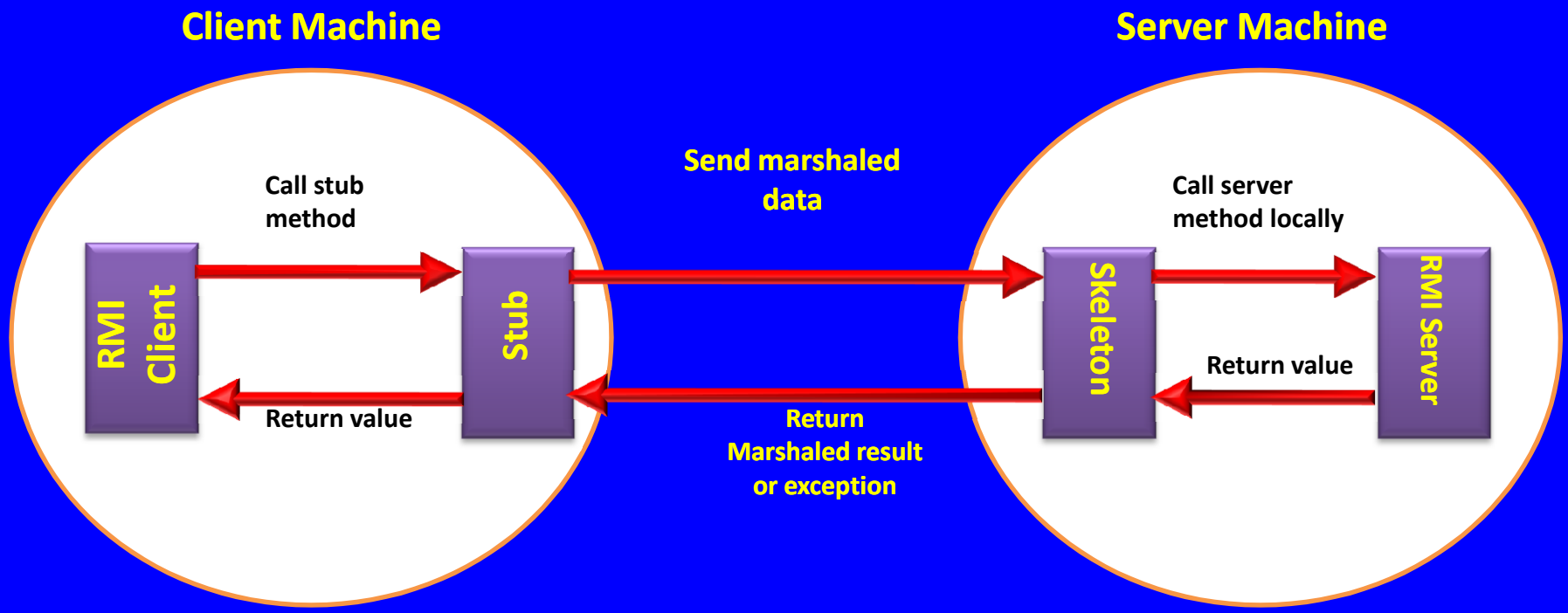
- **Start the Server**
  - `java SumServer`
- **Start the Client**
  - `java SumClient 127.0.0.1 5 6`



- A server creates remote objects.
- Registers these objects with rmiregistry

- The client gets a handle by looking up the remote object by its name in the server(s) registry and invokes methods on it

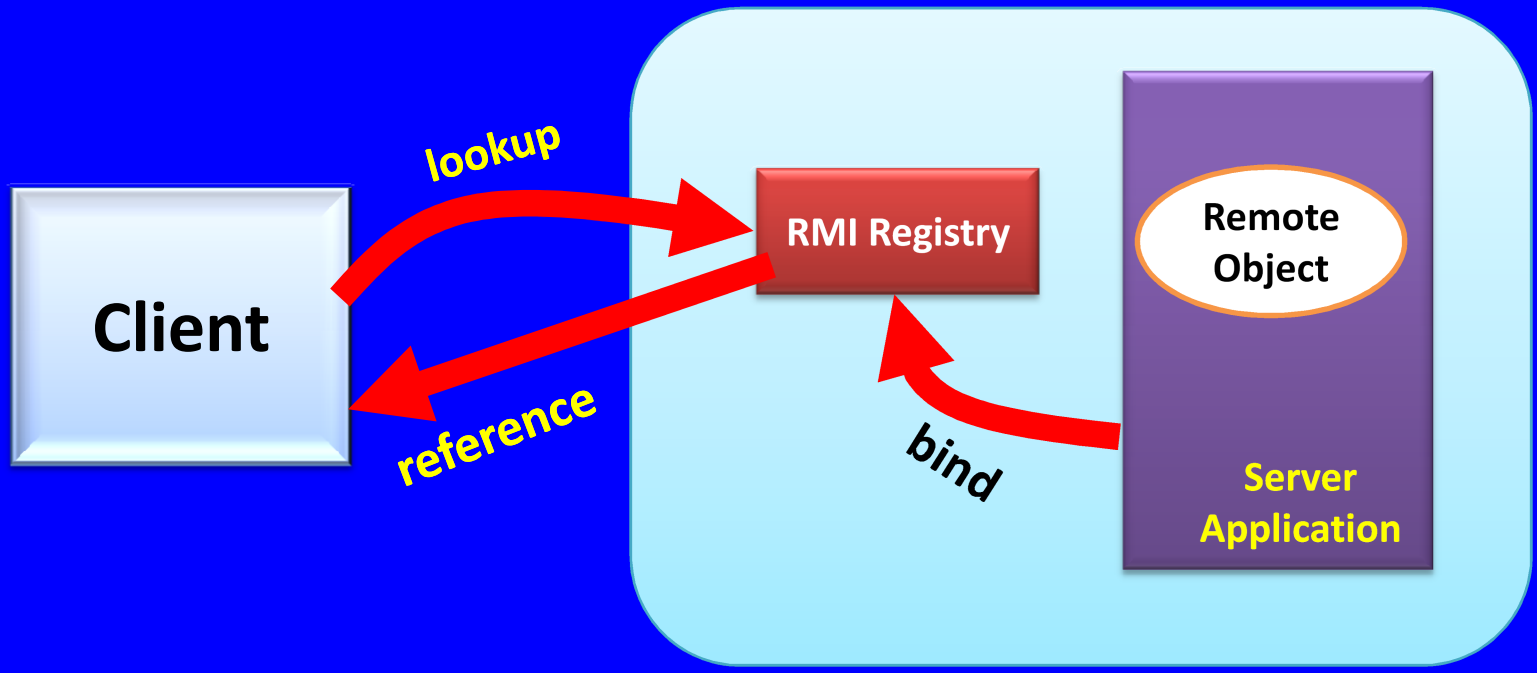
Data transfer between client and server



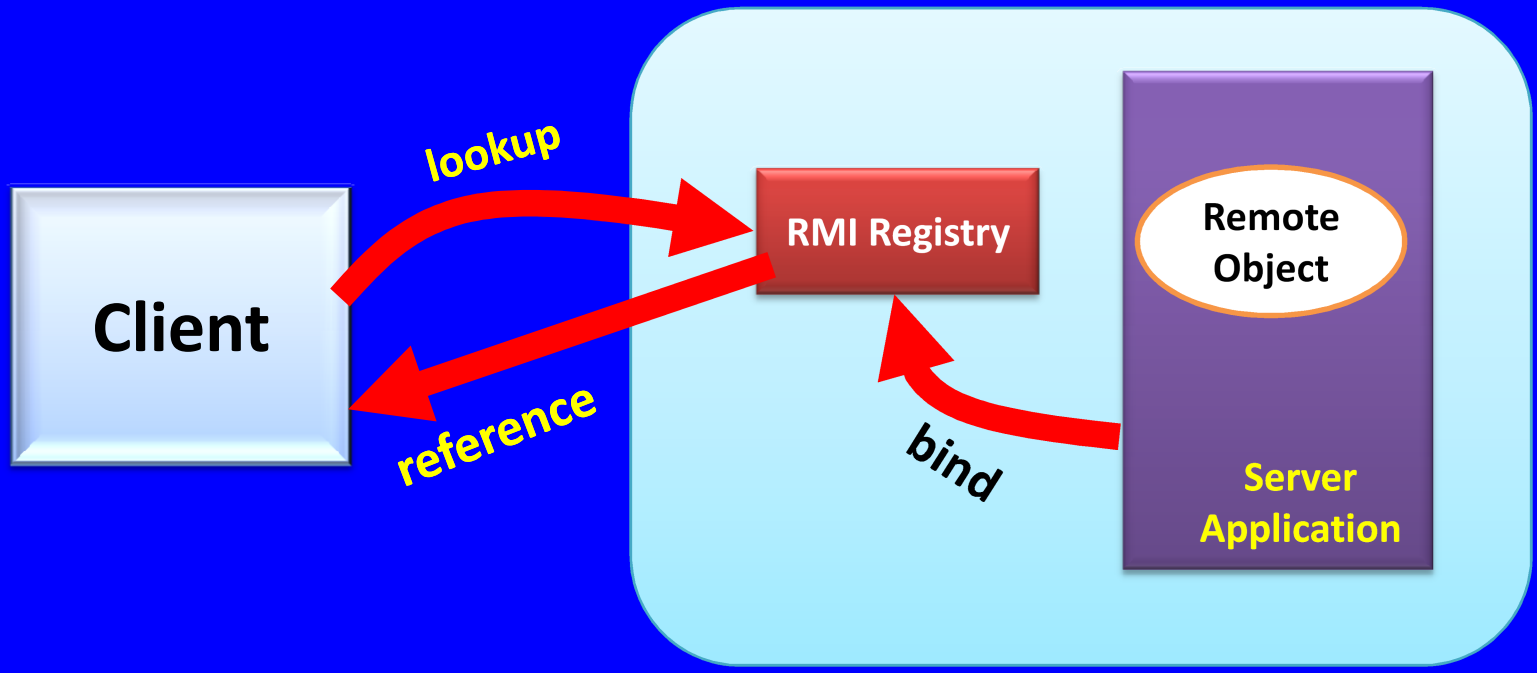
Bootstrap Name Server

Bootstrap Name Server

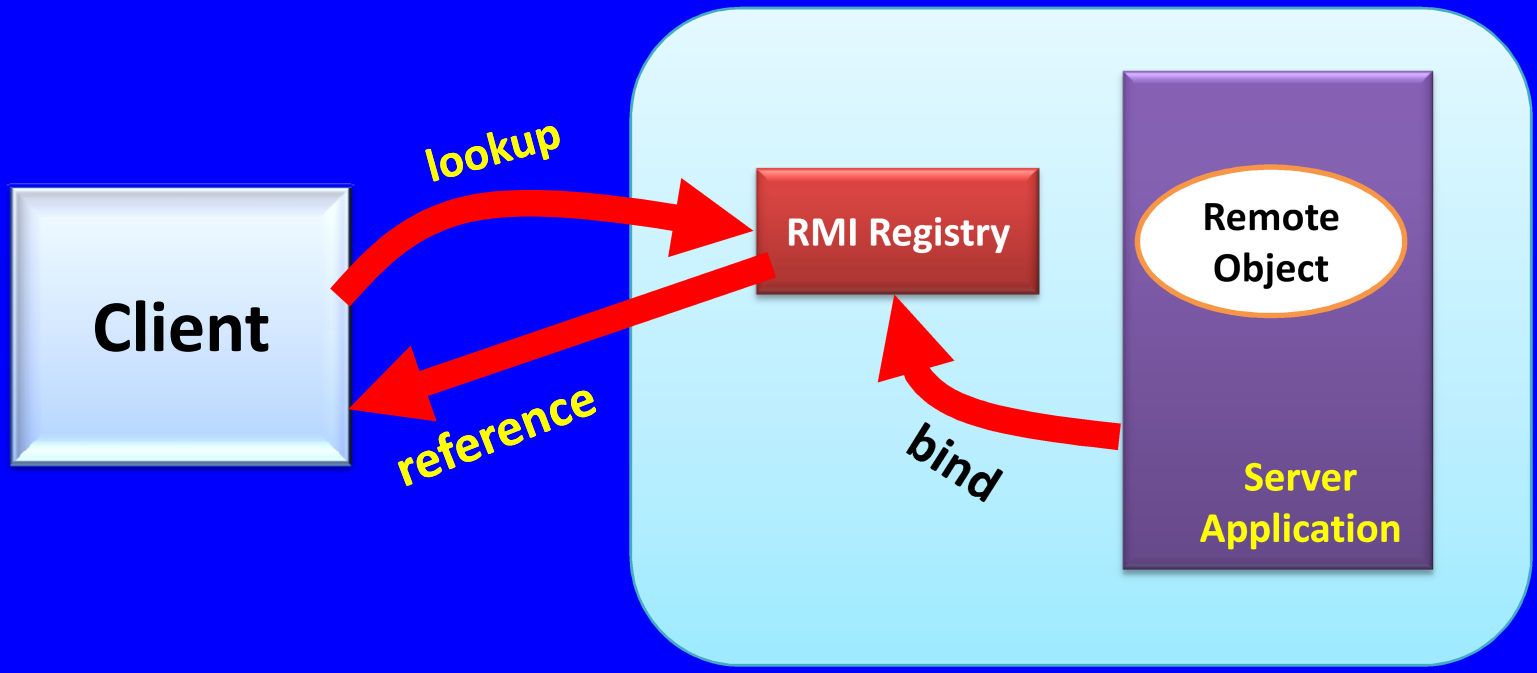
- The RMI registry is a simple name server provided for storing references to remote objects.



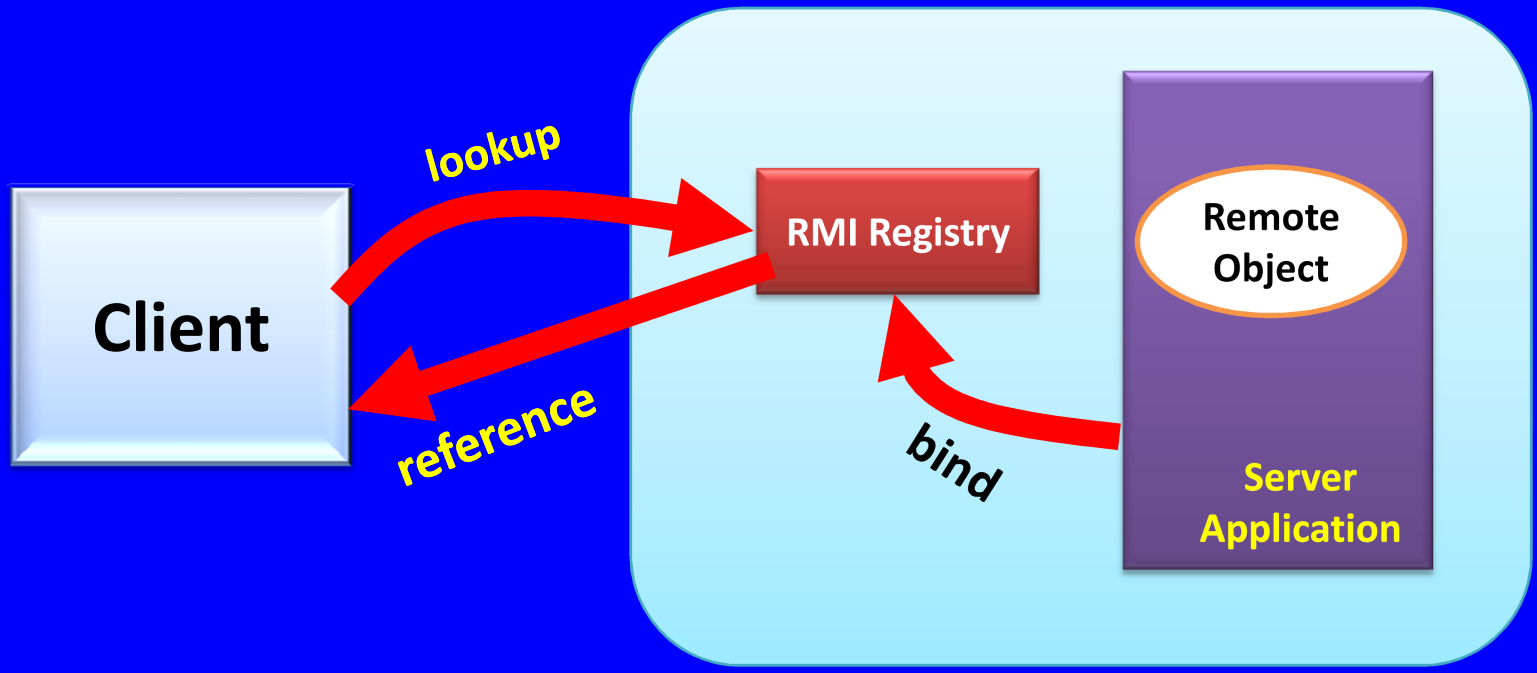
- The server is responsible for registering remote objects with the registry service.



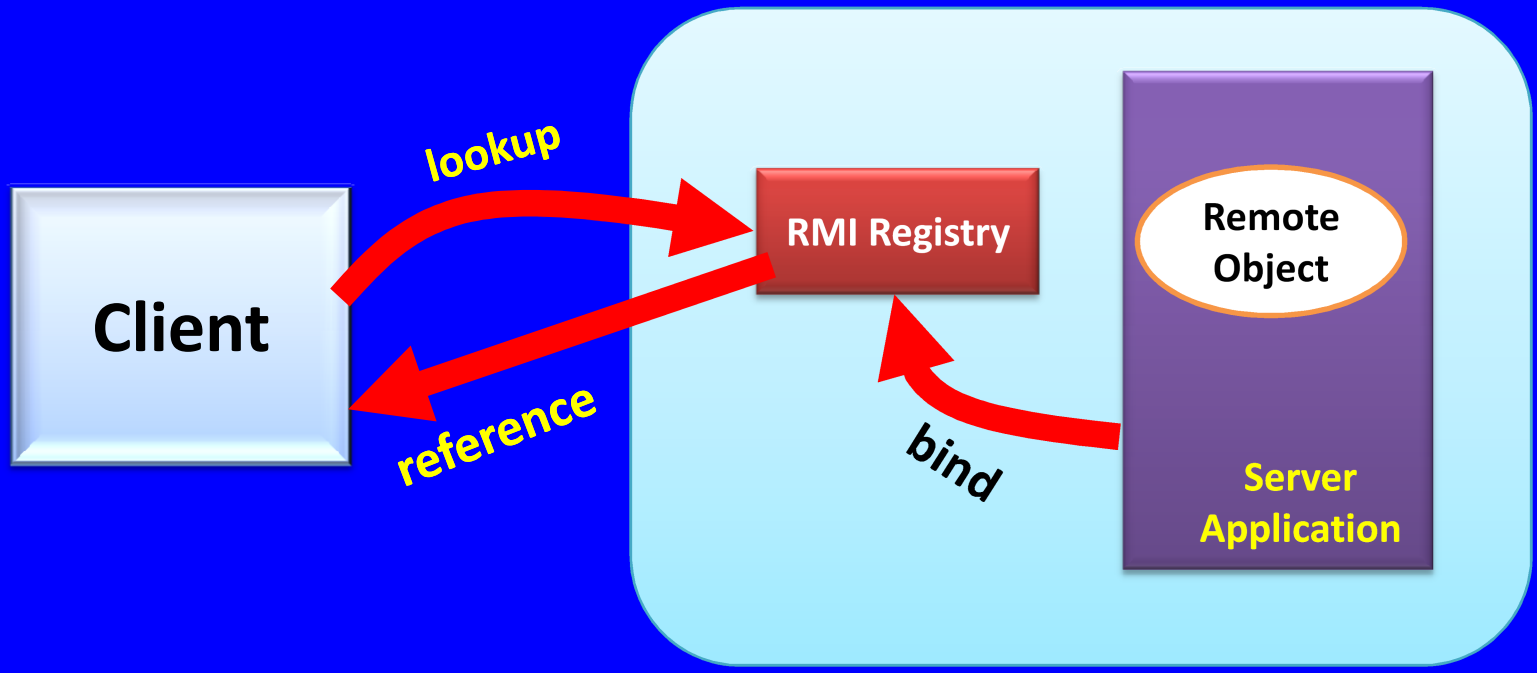
- The registry keeps track of the addresses of remote objects.
- A remote objects handle can be stored using the methods of java.rmi.Naming class.



- A client can obtain a reference to a remote object by looking up in the server(s) registry or as a result of a remote method call.



- An application can also run its own registry service.



Java RMI
Creating and running registry service

By

CHITRAKANT BANCHHOR

IT, SCOE, Pune

Registry Service

- There are two ways to start a registry:

1. Use the RMI registry application (rmiregistry)
2. Use java.rmi.registry package to write own registry service

write and start registry service

- The server stores the names of the users in a file.
- A user can register and view the names of already registered users.

Step – 1: Create Remote Interface

Methods

1. registerUser
2. getRegisteredUsersList

Methods

1. registerUser

- registerUser() stores the names of users in a file named "regFile.txt"

Methods

2. getRegisteredUsersList

- The getRegisteredUsersList() method returns a list of already registered users.

(RegistryServerInt.java)

- create an interface that extends the Remote interface.

```
1
2  import java.rmi.*;
3  import java.io.*;
4
5
6  public interface RegistryServerInt extends Remote {
7
8      //Used to register users in the registry ( write user names into registry file)
9      public String registerUser(String uName) throws RemoteException, IOException;
10
11     //returns list of users from the registy
12     public String getRegisteredUsersList() throws RemoteException, IOException;
13 }
14
15
16
```

Step – II: Implementing the Server

- This class implements the remote interface (RegistryServerInt.java) and extend UnicastRemoteObject.

Steps performed by the implementation: registerUser()

```
1. public  String registerUser(String uName);
```

- This method takes user name as its arguments and returns a string, if the name is successfully stored in the file.

1. **Creates a FileWriter object:**

```
FileWriter  fw  =  new  FileWriter();
```

2. **Creates a buffered character-output stream**

```
BufferedWriter bw =  new BufferedWriter(fw);
```

Steps performed by the implementation: getRegisteredUsersList()

2. public String getRegisteredUsersList();

- This method returns registered users list

1. **Creates a FileReader object:**

```
FileReader  fr  =  new  FileReader();
```

2. **Creates a buffered character-input stream to read text from the file:**

```
BufferedReader br =  new BufferedReader(fr);
```

RegistryServerImpl.java

```
2  import java.io.*;
3  import java.rmi.*;
4  import java.rmi.server.*;
5  import java.rmi.registry.*;
6
7  public class RegistryServerImpl extends UnicastRemoteObject
8          implements RegistryServerInt {
9
10     public RegistryServerImpl() throws RemoteException, IOException{
11
12     }
```

```
16      public String registerUser(String uName)
17              throws RemoteException, IOException {
18
19          //Constructs a FileWriter object given,
20          // true boolean value indicates to append the data written.
21
22          FileWriter fw = new FileWriter("regFile.txt", true);
23
24          //Create a buffered character output stream
25          //It uses default sized output buffer.
26          BufferedWriter bw = new BufferedWriter(fw);
27
28          //Write user name
29          bw.write(uName);
30
31          //Write a line separator
32          bw.newLine();
33
34          bw.flush();
35          bw.close();
36          fw.close();
37
38          return ( uName + "registered");
39
40      }//registerUser
```

public void **newLine**() throws IOException

```
42      public String getRegisteredUsersList()
43              throws RemoteException, IOException{
44
45          FileReader fr = new FileReader("regFile.txt");
46
47          //Create a buffered character input stream
48          //It uses default sized input buffer.
49          BufferedReader br = new BufferedReader(fr);
50
51          String uList = "";
52          String temp = "";
53
54          do{
55
56              //Read a line of text.
57              //A line terminated by ('\n'), ('\r'),
58              temp = br.readLine();
59              if( temp != null) {
60                  uList = uList + temp + "\n" ;
61              }
62
63          }while(temp != null);
64
65          return (uList);
66      }//getRegisteredUsersList
67  }
```

public String **readLine**() throws IOException

Step – III: Create the Server

-

The java.rmi.registry Package

- The package contains classes.

Final class

Interface

Registry Interface (public interface Registry extends Remote)

- The java.rmi.registry.Registry interface provides methods:

  - lookup,
  - binding,
  - rebinding,
  - listing the contents of a registry.

- **Actually, the methods of java.rmi.Naming class call the methods of a remote object that implements the Registry interface.**

1. **bind**
2. **rebind**
3. **unbind**
4. **lookup**
5. **list**

1. bind

```
public void bind ( String name, Remote obj )
```

- This method binds the specified "name" to a remote object.
- It throws an AlreadyBoundException, if the name is already bound to an object.

2. rebind

```
public void rebind ( String name, Remote obj )
```

- It rebinds the specified name to a remote object.
- It replaces an object that is already bound with the registry with the new object specified with obj.

3. unbind

```
public void unbind ( String name )
```

- It removes an object reference from the registry.
- It throws a NotBoundException if there was no binding.

4. lookup

```
public Remote lookup ( String name )
```

- The lookup method is used to get a remote object from a registry.
- It takes a string containing the URL to a remote object reference on the server.

5. list

```
public String list ( )
```

- It returns an array of the Strings bound in the registry.
- Each String object contains a name that is bound to a remote object.
- The names are URL-formatted strings.

LocateRegistry Class

- **The LocateRegistry class can be used to find or create a new *registry*.**

1. **createRegistry**
2. **getRegistry**

1. createRegistry

```
public static Registry createRegistry ( int port )
```

- This method is used to create and export a *registry* on the local host that accepts requests on the specified *port*.
- The port must be available otherwise an exception will be thrown.

2. getRegistry

```
public static Registry getRegistry ( )
```

- This method returns a reference to the *registry* on port no. 1099.

3. getRegistry

```
public static Registry getRegistry ( int port )
```

- This method attempts to return a reference to the *registry* on the specified port.

4. getRegistry

```
public static Registry getRegistry ( String host )
```

- This method returns a reference to the *registry* on the specified host on port no. 1099.
- If host is null, the local host is assumed.
- If the given host cannot be resolved, an UnknownHostException is thrown.

5. getRegistry

```
public static Registry getRegistry ( String host, int port )
```

- This method returns a reference to the *registry* on the specified host and port.
- If the given host cannot be resolved, an UnknownHostException is thrown.

6. getRegistry

```
public static Registry getRegistry ( String host, int port, RMIClientSocketFactory csf )
```

- **This method returns a locally created remote reference to the *registry* on the specified host and port. Communication with the *registry* will use the supplied RMIClientSocketFactory csf.**

Steps performed by the server

1. Server creates an instance of the remote object, which provides the service.

```
RegistryServerImpl regServerObject = new RegistryServerImpl();
```

2. Create a registry that accepts calls on a specific port.

```
Registry userRegistry = LocateRegistry.createRegistry(2000);
```

3. **Bind a specific name to remote object in the registry.**

```
userRegistry.rebind("REGISTRY-SERVER",  regServerObject);
```

RegistryServer.java

```
2  import java.rmi.*;
3  import java.rmi.server.*;
4  import java.rmi.registry.*;
5
6  public class RegistryServer {
7
8    public static void main(String[] args){
9
10     try {
11         //create a server object
12         RegistryServerImpl regServerObject = new RegistryServerImpl();
13
14         //Creates and exports a Registry instance on the local host
15         //that accepts requests on the specified port(here 2000).
16         Registry userRegistry = LocateRegistry.createRegistry(2000);
17
18         //Rebind the specified name to a remote object.
19         //Replaces the binding for the specified name in this registry
20         //with the supplied remote reference. If there is an existing binding
21         //for the specified name, it is discarded.
22         userRegistry.rebind("REGISTRY-SERVER", regServerObject);
23         System.out.println("server is ready for use");
24     }//try
25     catch(Exception re){
26         re.printStackTrace();
27     }//catch
28   }//main
29 }//class
```

Step – IV: Write the Client

1. Locate registry
2. Get reference of the interface when lookup into server registry
3. Request services of the server

RegistryClient.java

```
1  import java.io.*;
2  import java.rmi.*;
3  import java.rmi.registry.*;
4  import javax.swing.*;
5
6  public class RegistryClient {
7
8    public static void main(String[] args){
9
10     try{
11         //Returns a reference to the the remote object Registry
12         //for the local host on the specified port (here 2000).
13         Registry userRegistry = LocateRegistry.getRegistry(2000);
14
15        //Get a reference for the remote object bound to the
16        //specified name in this registry.
17         RegistryServerInt regserverIntf =
18               (RegistryServerInt)userRegistry.lookup("REGISTRY-SERVER");
19         String uCh1 = "1-->Register user name into registry\n";
20         String uCh2 = "2-->Get list of users";
21         String uChoices= uCh1 + uCh2;
22         String sCh= JOptionPane.showInputDialog(uChoices);
```

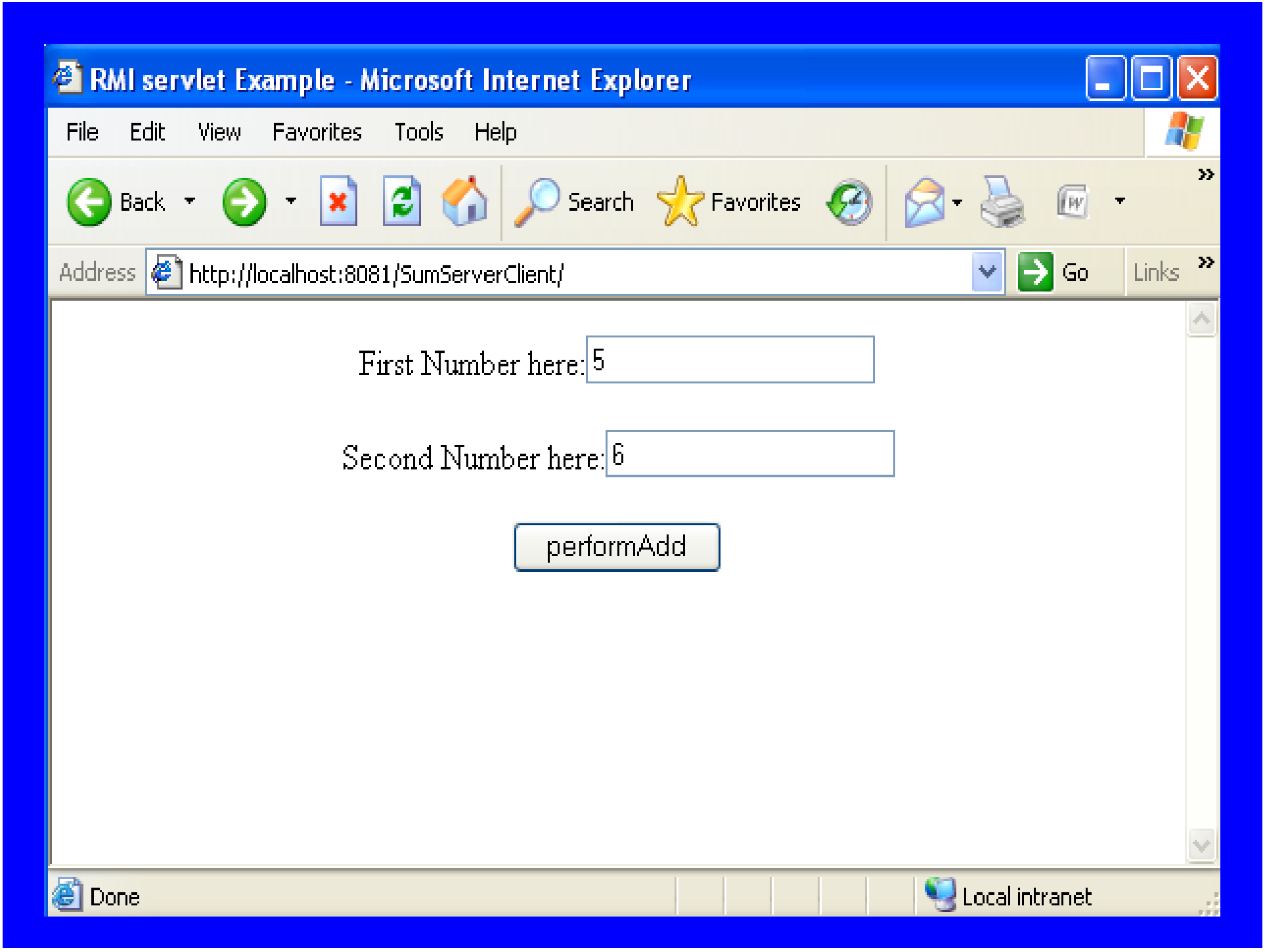
```
24          int ch = Integer.parseInt(sCh);
25          if( ch == 1){
26              ////call the remote method to register the user
27              String uName = JOptionPane.showInputDialog("Enter user name to register");
28              String message = regserverIntf.registerUser(uName);
29              JOptionPane.showMessageDialog(null,message);
30          }
31          else if(ch == 2){
32
33              //invoke the remote method to get the list
34              String list = regserverIntf.getRegisteredUsersList();
35              System.out.println(list);
36          }
37          else{
38              System.out.println("Unknown option");
39          }
40      }//try
41      catch(Exception e){
42              e.printStackTrace();
43      }
44    }//main
45  }
```

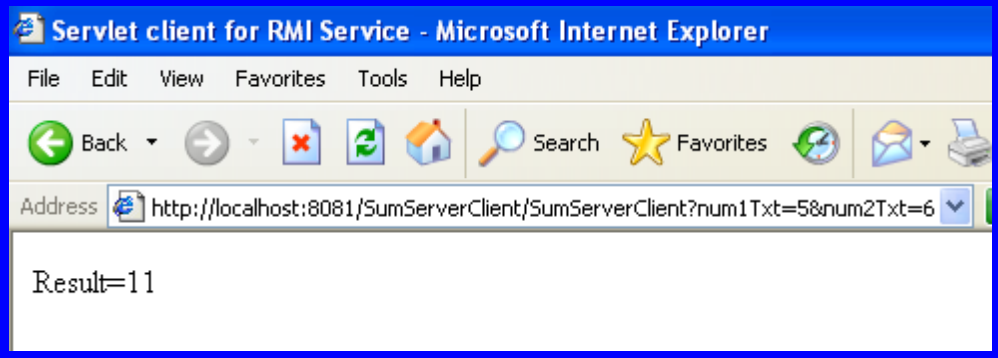
Java RMI
Servlets and RMI

CHITRAKANT BANCHHOR
IT, SCOE, Pune

Servlets

- *Servlets* are Java classes that are loaded and executed by a web server.
- How a servlet invokes remote methods.


RMI servlet Example - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back
Search
Favorites
Address
http://localhost:8081/SumServerClient/
Go
Links
First Number here: 5
Second Number here: 6
performAdd
Done
Local intranet


Servlet client for RMI Service - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back
Search
Favorites
Address
http://localhost:8081/SumServerClient/SumServerClient?num1Txt=5&num2Txt=6
Result=11

The Server Interface

- The SumServerIntf declares a method that takes two numbers as arguments and returns their addition.

SumServerIntf.java

```
1 import java.rmi.*;
2
3 public interface SumServerIntf extends Remote{
4
5     public int performAdd(int num1, int num2)
6         throws RemoteException;
7 }
8
9
```

SumServer Implementation

- The SumServerImpl class extends UnicastRemoteObject in order to be exported and implements the SumServerIntf.

SumServerImpl.java

```
 1 import java.rmi.*;
 2 import java.rmi.server.*;
 3
 4 public class SumServerImpl extends UnicastRemoteObject
 5         implements SumServerIntf{
 6
 7     public SumServerImpl() throws RemoteException {
 8         System.out.println("Creating server object");
 9     }
10
11     public int performAdd(int n1, int n2) throws RemoteException {
12         return n1 + n2;
13     }
14 }
15
```

SumServer class

- The main() method in this class creates an instance of the SumServerImpl (remote object).
- Then binds SumServerImpl with the registry service.

SumServer.java

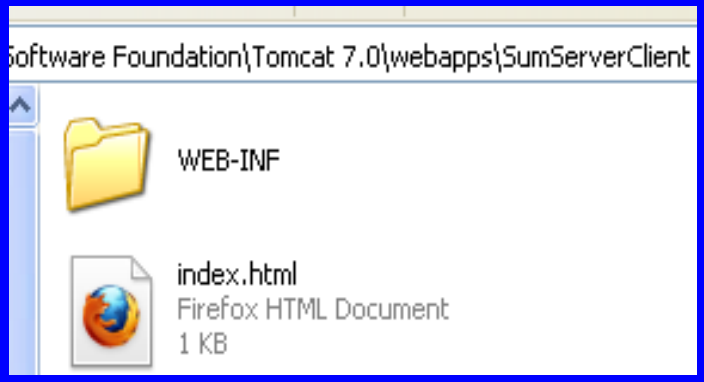
```
1  import java.rmi.*;
2  import java.rmi.server.*;
3  import java.rmi.registry.*;
4
5
6  public class SumServer {
7      public static void main(String[] args){
8          try{
9              SumServerImpl sumserverimplObj = new SumServerImpl();
10             Registry createdRegistry = LocateRegistry.createRegistry(1099);
11             createdRegistry.rebind("SUM-SERVER", sumserverimplObj);
12         }
13         catch(Exception e){
14             System.out.println("Exception:" + e.getMessage());
15
16         }
17     }
18 }
```

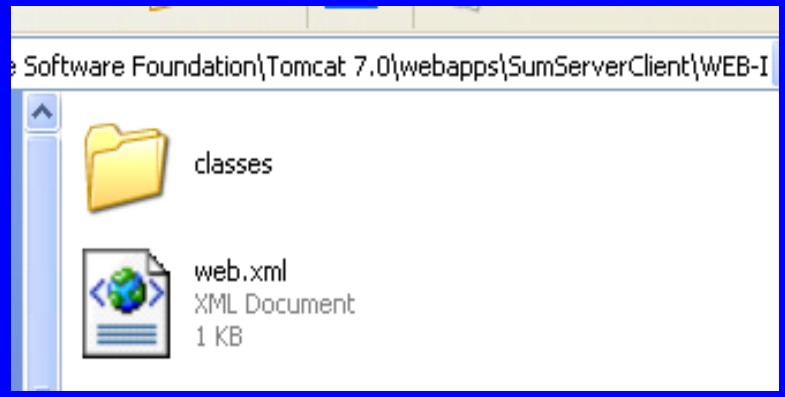
Servlet Client

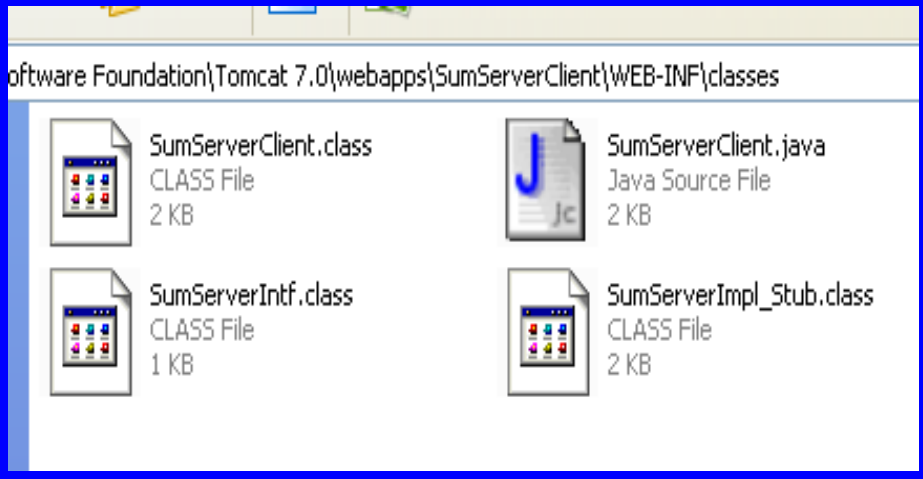
- First, servlet's init() method is called and then it gets a reference to the remote object.

```
sumServerIntf = (SumServerIntf)Naming.lookup( "SUM-SERVER“ );
```

Servlet Client







index.html

```
<html>
  <head>
   <title>RMI servlet Example</title>
  </head>
  <body>
    <center>
    <form name="form1" method="GET"
            action="/SumServerClient/SumServerClient">
      First Number here:<input type="text" name="num1Txt"/><br><br
      Second Number here:<input type="text" name="num2Txt"/><br><b
      <input type="submit" name="submit" value="performAdd">
    </form>
    </center>
  </body>
</html>
```

web.xml

```
- <web-app>
  - <servlet>
      <servlet-name>rmiclientservlet</servlet-name>
      <servlet-class>SumServerClient</servlet-class>
    </servlet>
  - <servlet-mapping>
      <servlet-name>rmiclientservlet</servlet-name>
      <url-pattern>/SumServerClient</url-pattern>
    </servlet-mapping>
  </web-app>
```

SumServerClient.java

```
1  import javax.servlet.*;
2  import javax.servlet.http.*;
3  import java.io.*;
4  import java.rmi.*;
5
6  public class SumServerClient extends HttpServlet {
7
8      private SumServerIntf sumServerIntf; //reference of remote object
9
10     public void init(ServletConfig config) throws ServletException {
11
12         try{
13             sumServerIntf = (SumServerIntf)Naming.lookup("SUM-SERVER");
14         }
15         catch(Exception e){
16             e.printStackTrace();
17         }
18     }//init
```

```
20      public void doGet(HttpServletRequest req, HttpServletResponse res)
21              throws ServletException, IOException{
22
23          res.setContentType("text/html");
24          PrintWriter out = res.getWriter();
25
26          out.println("<HTML>");
27          out.println("<HEAD>");
28          out.println("<TITLE>Servlet client for RMI Service</TITLE>");
29          out.println("</HEAD>");
30          out.println("<BODY>");
31
32          String sNum1 = req.getParameter("num1Txt");
33          String sNum2 = req.getParameter("num2Txt");
34
35          int iNum1 = Integer.parseInt(sNum1);
36          int iNum2 = Integer.parseInt(sNum2);
37
38          int resultSum = sumServerIntf.performAdd(iNum1, iNum2);
39          out.println("Result=" + resultSum);
40
41          out.println("</BODY>");
42          out.println("</HTML>");
43      }//doGet
44  }
```

Execution

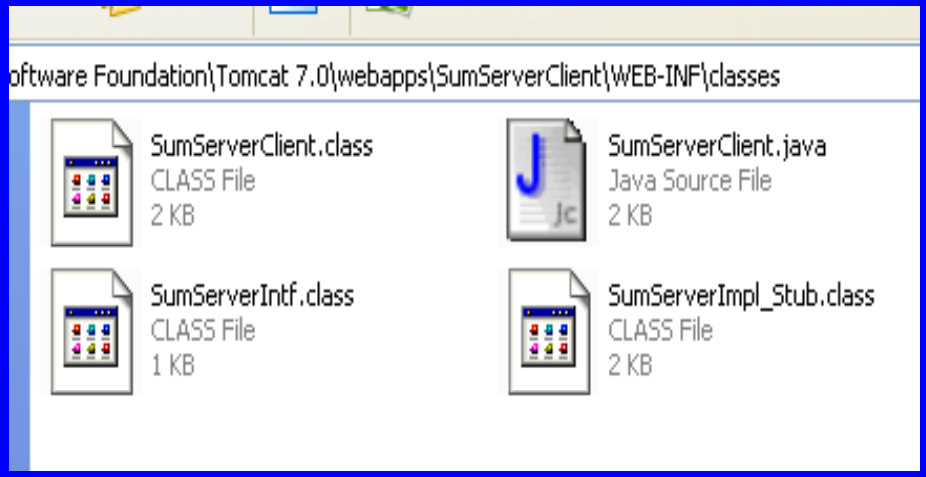
- Create stub using rmic

```
C:\>  rmic  SumServerImpl
```

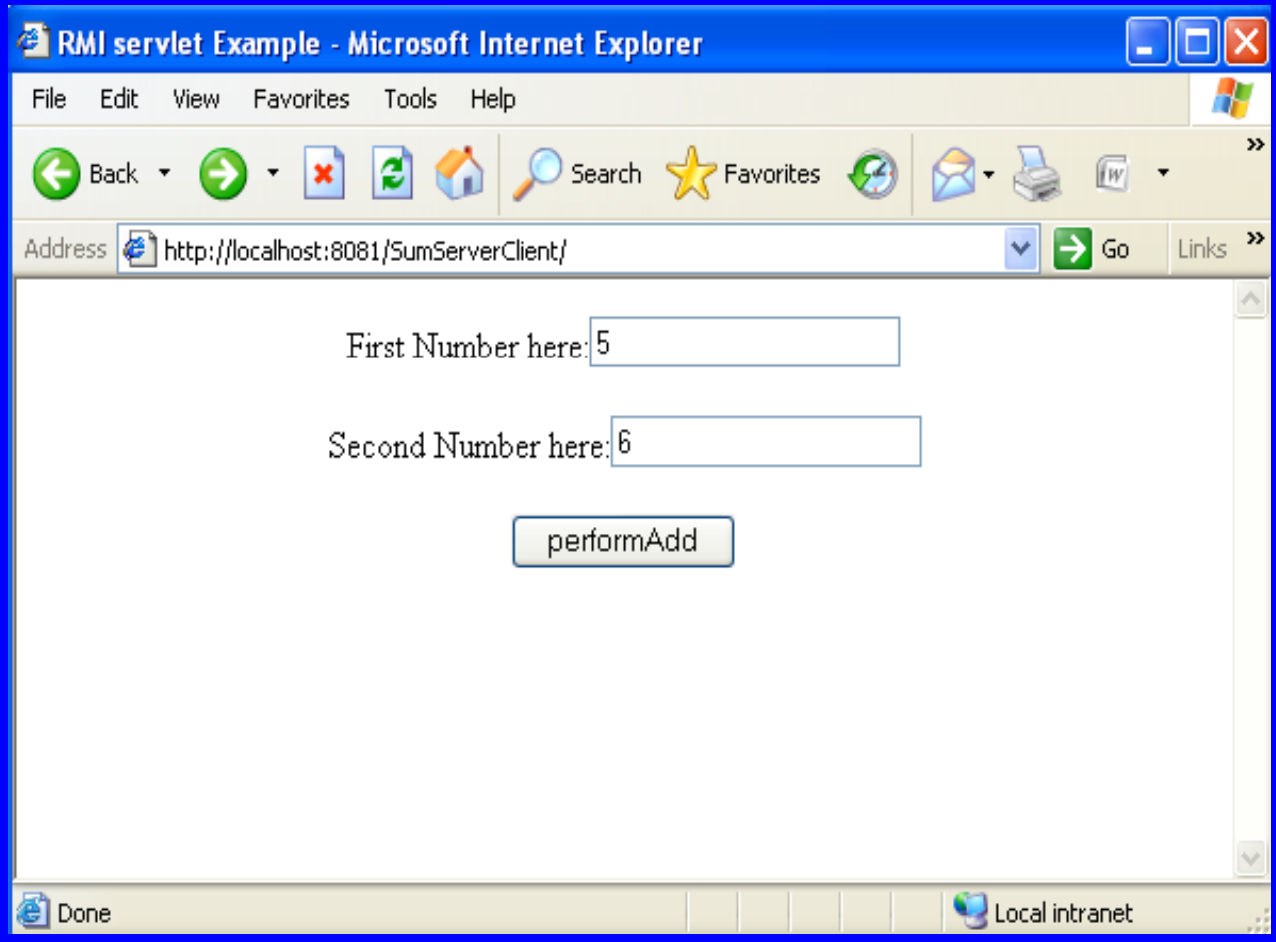
- Start the server (SumServer)

```
C:\>  start   java  SumServer
```

❖Deploy Servlet client to Tomcat Server



- Deploy Servlet client to Tomcat Server
- Run the servlet using browser:



Java Data Base Connectivity (JDBC)

CHITRAKANT BANCHHOR
IT, SCOE, Pune

References

- George Reese, “Database Programming with JDBC and JAVA, O'Reilly.
- W. Clay Richardson et al, “Professional Java”, Wrox

What is JDBC?

Actually an Application Programming Interface (API)

❖ **JDBC Consists :**

- **Java Classes,**
- **Interfaces**
- **Exceptions**

JDBC components

1. The application
2. The driver manager
3. The driver
4. The data source

The application

- **Performs:**

1. Requests a connection with a data source
2. Sends SQL statements to the data source
3. Requests results
4. Controls transactions by requesting commit or rollback operations
5. Closes the connection

The driver manager

- loads specific drivers for the user application.

The driver

- processes JDBC methods invocations,
- sends SQL statements to a specific data source,
- returns results back to the application.

2. Seven Basic steps of using JDBC

1. Load the driver
2. Define the connection URL
3. Establish the connection
4. Create a Statement object
5. Execute a query
6. Process the results
7. Close the connection

1

Load the driver

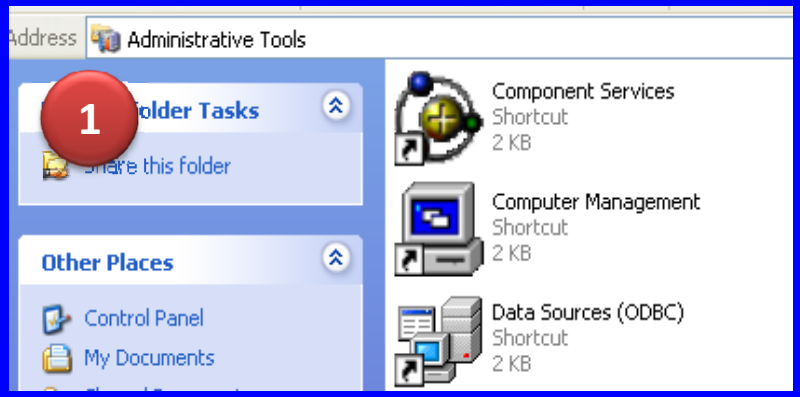
- It is necessary to load the JDBC drivers before attempting to connect to the database.
- The JDBC drivers automatically register themselves with the JDBC system when loaded.

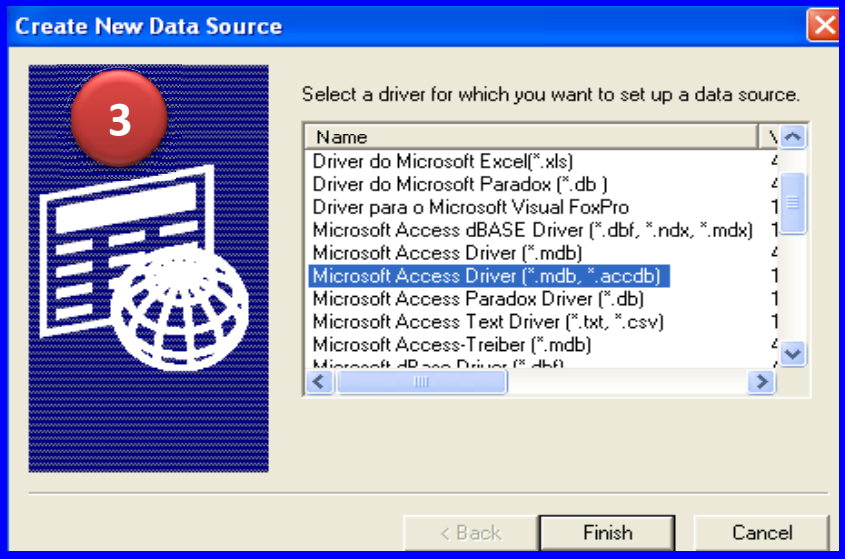
```
try {
        Class. forName ("sun.jdbc.odbc.JdbcOdbcDriver");
}
catch (ClassNotFoundException cnfe){
        System.out.println("Error loading driver" + cnfe);
}
```

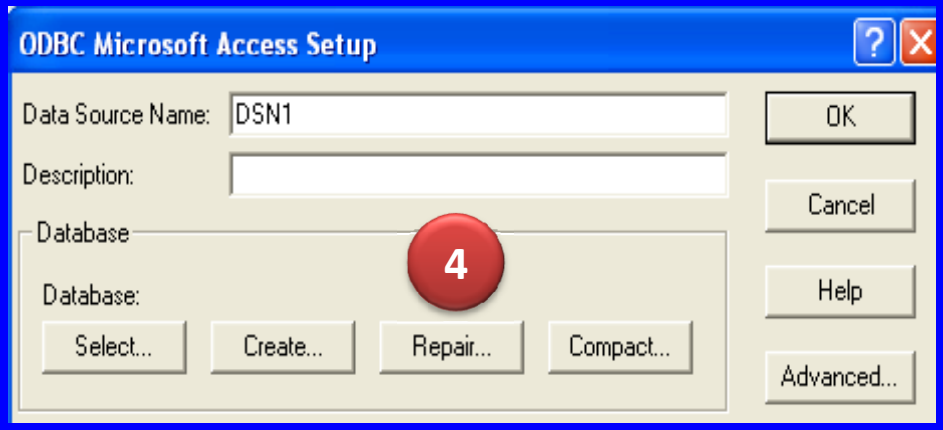
2

Define the connection URL

```
String msAccessURL = "jdbc:odbc:datasource";
```







String msAccessURL = "**jdbc:odbc:DSN1**";

3

Establish the connection

- Make the connection with the database server.
- In the Establishing Connection step we will logon to the database with user name and password.

```
Connection con =  DriverManager. getConnection
                        (
                                msAccessURL,
                                userName,
                                passWORD
                        );
```

4 **Create a Statement object**

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

5 **Execute a query**

```
String Query = "SELECT * FROM Student";
ResultSet rs = stmt.executeQuery(Query);
```

6

Process the results

- **In this step we receive the result of execute statement.**

Example:

```
while( rs.next() )  {

        System.out.print(rs.getInt(1));
        System.out.print(rs.getString("SName"));
}
```

7

Close the connection

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

Example - 1

```
1       qwe
2       wer
3       asd
```

```
/*packages required for jdbc*/
import java.sql.*;
import java.util.*;

public class Jdbc1{
        public static void main(String[] args){
                try{
                        Class.forName("sun.jdbc.JdbcOdbcDriver");
                }
                catch(ClassNotFoundException ce){
                        System.out.println("ClassNotFoundException");
                }
                try{
                        String URL = "jdbc:odbc:data1";
                        Connection con = DriverManager.getConnection(URL);
                        Statement stmt = con.createStatement();
                        ResultSet rs = stmt.executeQuery("Select * from student");
                        while(rs.next()){
                                System.out.print(rs.getInt(1) + "\t");
                                System.out.print(rs.getString(2) + "\t");
                                System.out.println();
                        }
                }
                catch(SQLException sql){
                        System.out.println("SQLException");
                }
        }
}
```

Example -2

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

public class Jdbc2
{
        public static void main(String[] args)
        {
                try
                {
                        Class.forName("sun.jdbc.JdbcOdbcDriver");
                }
                catch(ClassNotFoundException ce)
                {
                        System.out.println("ClassNotFoundException");
                }
                try
                {
                        String URL = "jdbc:odbc:data1";
                        Connection con = DriverManager.getConnection(URL);
                        Statement stmt = con.createStatement();
                        ResultSet rs = stmt.executeQuery("Select * from student");
                        while(rs.next())
                        {
                                System.out.print(rs.getInt(1) + "\t");
                                System.out.print(rs.getString(2) + "\t");
                                System.out.println();
                        }
```

```
                        con.close();
                        con = DriverManager.getConnection(URL);
                        stmt = con.createStatement();
                        stmt.executeUpdate("Insert into Student " +
                                "values (5, 'sder')");
                        con.close();
                }
                catch(SQLException sql)
                {
                        System.out.println("SQLException");
                }
        }
}
```

Example - 3

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

public class Jdbc4{
        public static void main(String[] args){
                String DRIVER = "sun.jdbc.odbc.JdbcOdbcDriver";
                String URL = "jdbc:odbc:data2";
                String QUERY = "Delete from Student where ID = 5";
                try{
                        Class.forName(DRIVER);
                }
                catch(ClassNotFoundException ce){
                        System.out.print("CE");
                }
                try{
                        Connection con = DriverManager.getConnection(URL);
                        Statement stmt = con.createStatement();
                        int nup = stmt.executeUpdate(QUERY);
                        System.out.println(nup);
                        con.close();
                }
                catch(SQLException sql){
                        System.out.print("SQLException");
                }
        }
}
```

Assignment
RMI and JDBC Example

SelectServerIntf.java

```
1
2 import java.rmi.*;
3 import java.util.*;
4
5 public interface SelectServerIntf extends Remote {
6
7     HashMap executeSelect() throws RemoteException;
8 }
9
```

SelectServerImpl.java

```
1  import java.rmi.*;
2  import java.sql.*;
3  import java.rmi.server.*;
4  import java.util.*;
5
6
7  public class SelectServerImpl extends UnicastRemoteObject implements SelectServerIntf {
8
9      public SelectServerImpl() throws RemoteException {
10     }
11
12     public HashMap executeSelect() throws RemoteException {
13
14         String DRIVER = "sun.jdbc.odbc.JdbcOdbcDriver";
15         String dbURL = "jdbc:odbc:DSN1";
16         String QUERY = "SELECT * FROM Student";
17         Connection con = null;
18         ResultSet rs = null;
19         Statement stmt = null;
20         HashMap hm = null;
21
22         try {
23                 Class.forName(DRIVER);
24
25         }
26         catch(ClassNotFoundException cnfe){
27             System.out.println("ClassNotFoundException" + cnfe);
28         }
```

```
29          try{
30
31              con = DriverManager.getConnection(dbURL);
32              stmt = con.createStatement();
33              rs = stmt.executeQuery(QUERY);
34              hm = new HashMap();
35              while(rs.next()){
36
37                  int rno = rs.getInt(1);
38                  String sname = rs.getString(2);
39                  hm.put(new Integer(rno), sname);
40              }
41              con.close();
42          }
43          catch(SQLException sqle){
44              System.out.println("SQLException" +  sqle);
45          }
46        return (hm);
47      }//executeSelect
48  }
```

SelectServer.java

```
1  import java.rmi.*;
2  import java.net.*;
3
4  public class SelectServer {
5
6      public static void main(String[] args){
7          try{
8              SelectServerImpl selectServerImpl = new SelectServerImpl();
9              Naming.rebind("SELECT-SERVER",selectServerImpl);
10         }
11         catch(Exception e){
12             System.out.println("Exception in server" + e);
13         }
14     }
15 }
```

SelectClient.java

```
1  import java.rmi.*;
2  import java.net.*;
3  import java.util.*;
4
5  public class SelectClient {
6
7      public static void main(String[] args){
8
9          String rmiURL = "rmi://" + args[0] + "/SELECT-SERVER";
10
11         try {
12
13                 SelectServerIntf selectServerIntf = (SelectServerIntf)Naming.lookup(rmiURL);
14
15                 HashMap hm2 = selectServerIntf.executeSelect();
16
17                 int sz = hm2.size();
18
19                 for(int i = 1; i <= sz; i++){
20
21                     if(hm2.containsKey(new Integer(i))){
22
23                         System.out.println(i + ":" + hm2.get(new Integer(i)));
24                     }//if
25                 }//for
26             }//try
27             catch(Exception e){
28                 System.out.println("Exception in server" + e);
29             }//catch
30     }//main
31 }//SelectClient
```

Hashmap: example for the assignment

Mapping with Hashtable and HashMap

- You need a one-way mapping from one data item to another.
- HashMap and Hashtable provide a one-way mapping from one set of object references to another.
- They are completely general purpose.

Example - 1

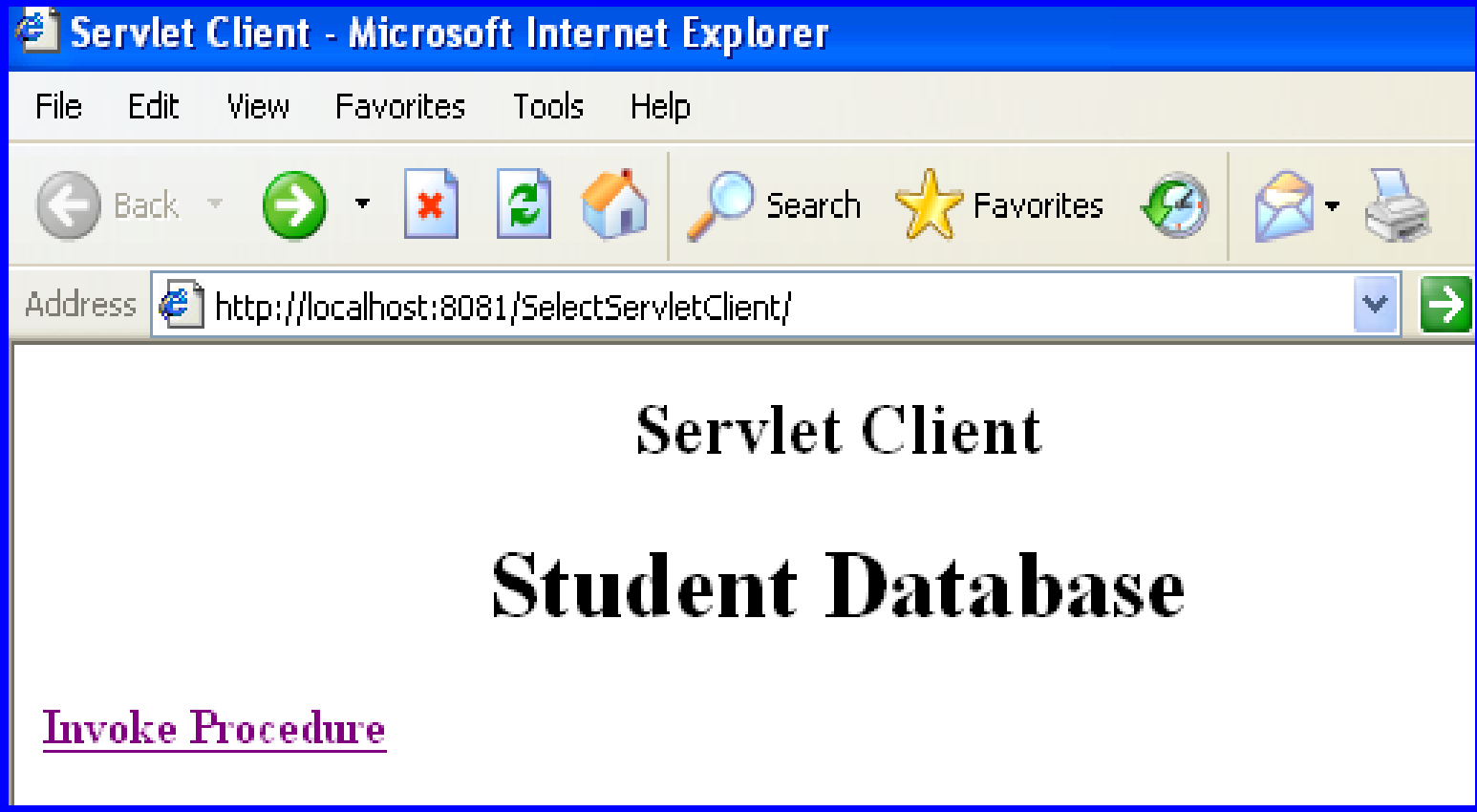
```
 1
 2 import java.lang.*;
 3 import java.util.HashMap;
 4
 5 public class HashDemo{
 6     public static void main(String[] args){
 7         HashMap h = new HashMap();
 8         h.put("Adobe","Kothrud,Pune");
 9         h.put("QuickHeal","PuneStation,Pune");
10         h.put("Microsoft","Shivaji Nagar, Pune");
11         String Query_String = "QuickHeal";
12         String Result = (String)h.get(Query_String);
13         System.out.println("Location = " + Result);
14     }
15 }
```

Example - 2

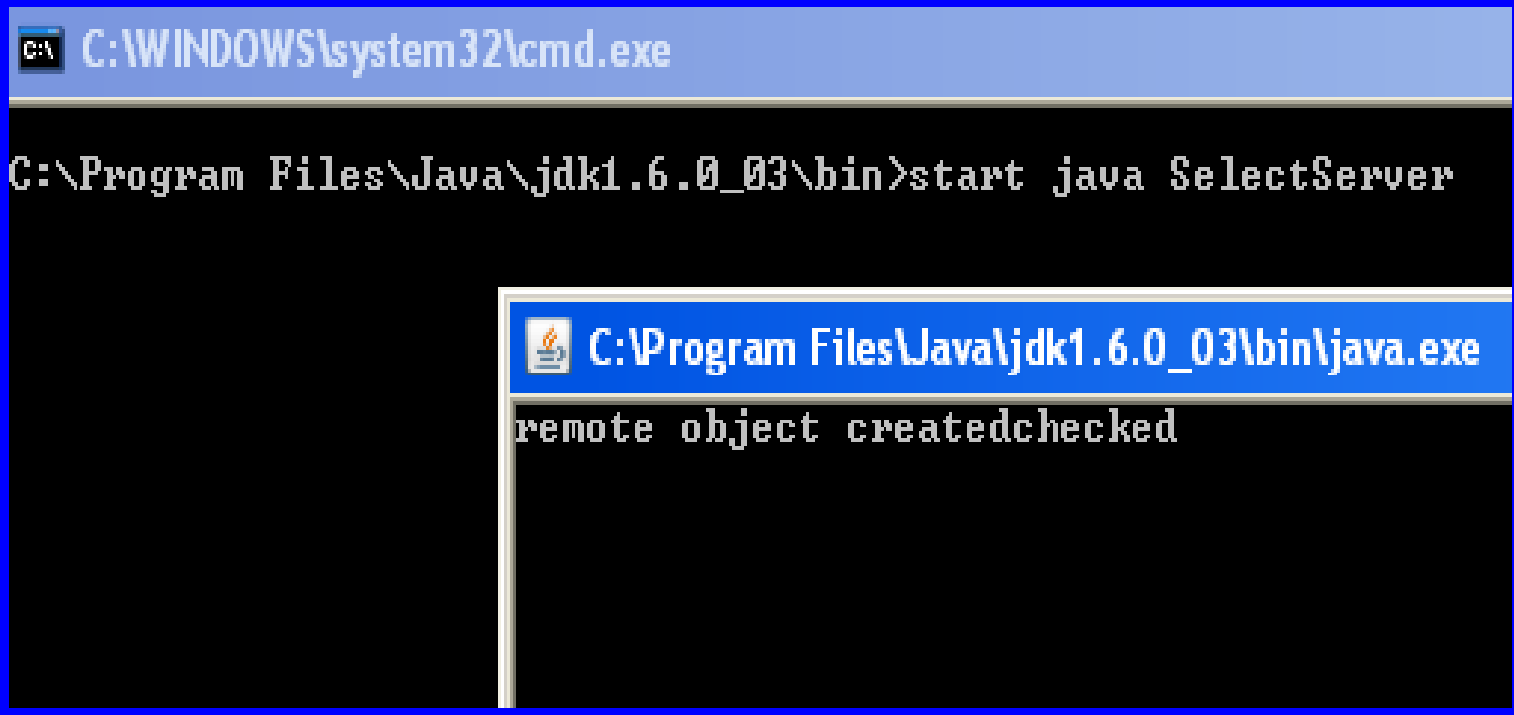
```
1  import java.util.HashMap;
2
3  public class HashMapDemo2 {
4
5      public static void main(String[] args){
6
7          HashMap hm = new HashMap();
8
9          hm.put(new Integer(1), "Ramesh");
10         hm.put(new Integer(2), "Rakesh");
11         hm.put(new Integer(3), "Rajesh");
12
13     /*  System.out.println(hm.get(new Integer(1)));
14         System.out.println(hm.get(new Integer(2)));
15         System.out.println(hm.get(new Integer(3)));
16         */
17
18         int sz = hm.size();
19         for(int i = 1; i <= sz; i++) {
20             if(hm.containsKey(new Integer(i))){
21                 System.out.println(i + ":" + hm.get(new Integer(i)));
22             }
23
24         }
25     }
26 }
```

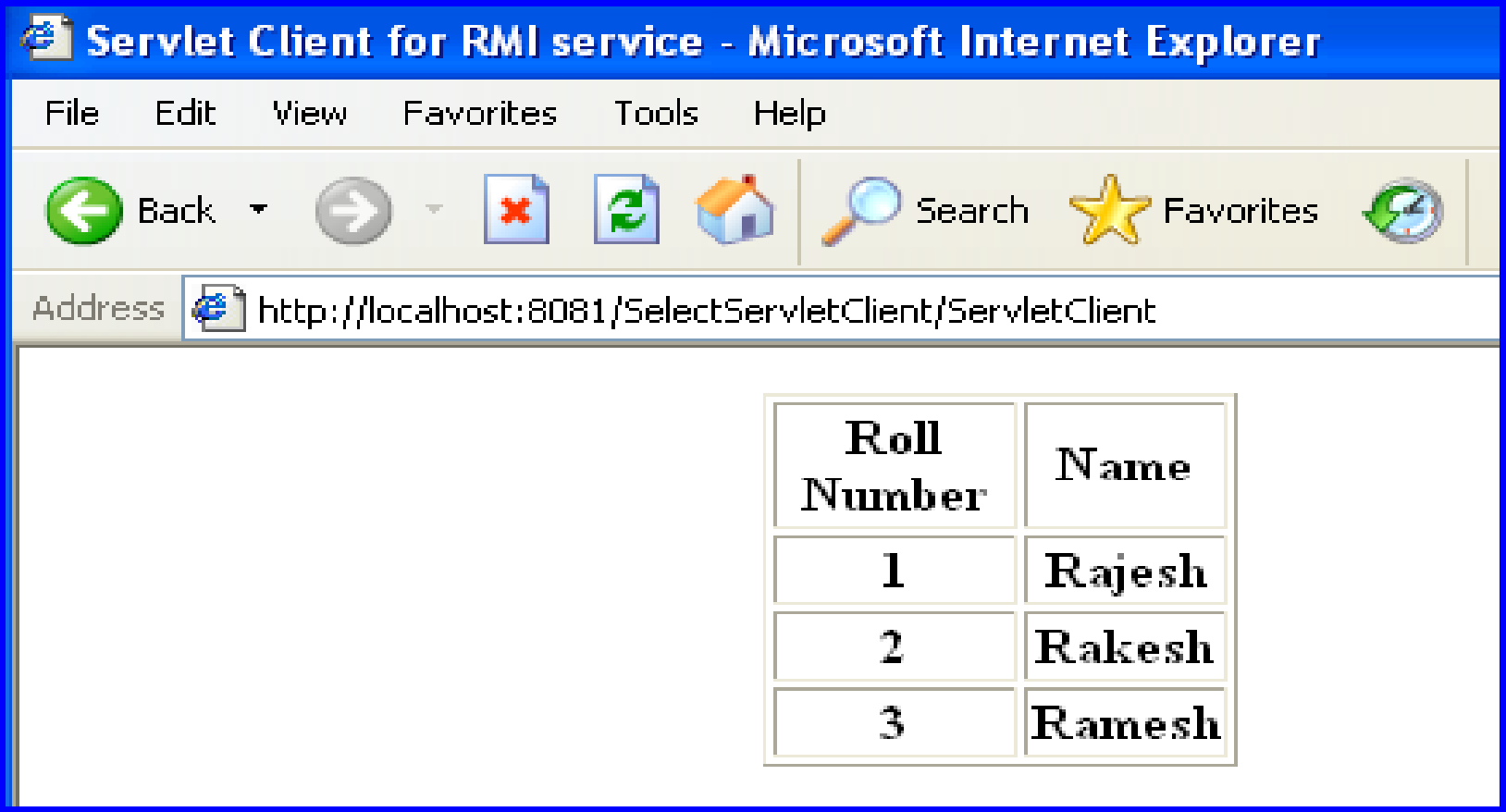
Thank You!

Student Database using RMI and Servlet


Servlet Client - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back
Search
Favorites
Address
http://localhost:8081/SelectServletClient/
Servlet Client
Student Database
Invoke Procedure

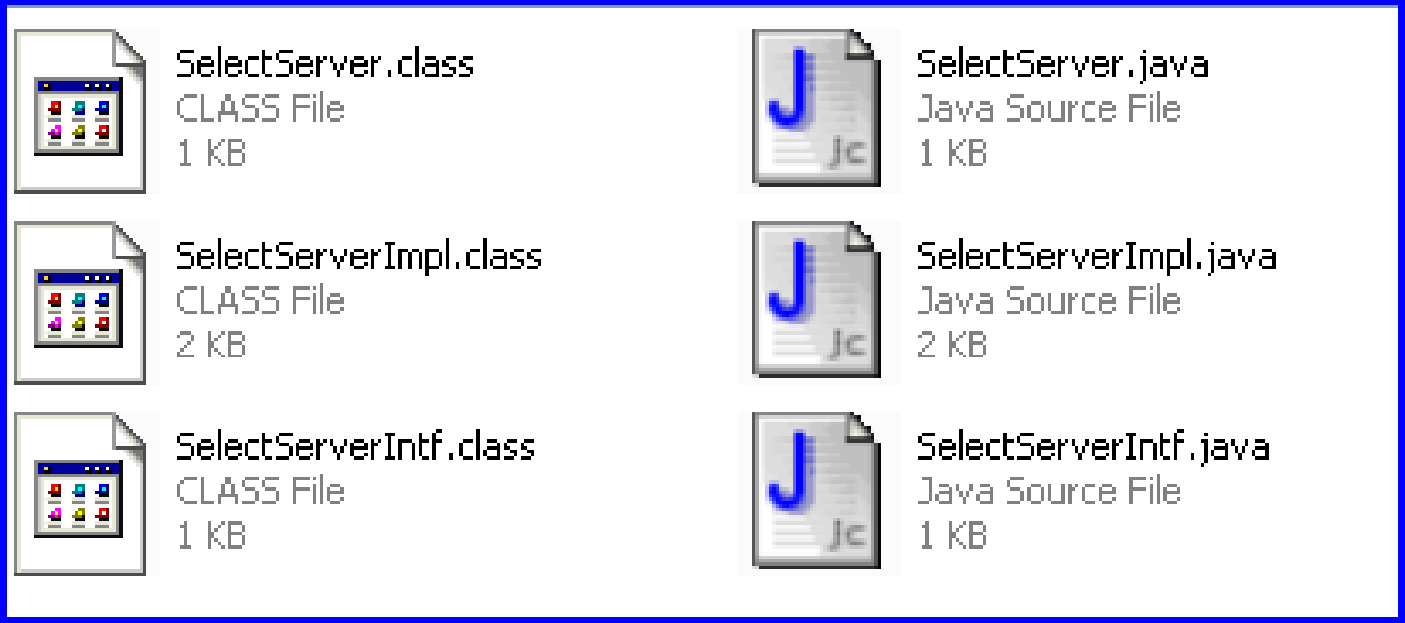
❖ **Start server**




Servlet Client for RMI service - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back
Search
Favorites
Address
http://localhost:8081/SelectServletClient/ServletClient
Roll Number
Name
1
Rajesh
2
Rakesh
3
Ramesh

Program Modules

SelectRMIServer



SelectServerIntf.java

```
1 import java.rmi.*;
2 import java.util.*;
3
4 public interface SelectServerIntf extends Remote {
5
6     public HashMap executeSelect() throws RemoteException;
7 }
8
```

SelectServerImpl.java

```
1  import java.sql.*;
2  import java.rmi.*;
3  import java.rmi.server.*;
4  import java.io.*;
5  import java.util.*;
6
7  public class SelectServerImpl extends
8            UnicastRemoteObject implements SelectServerIntf {
9
10     public SelectServerImpl() throws RemoteException {
11         System.out.print("remote object created");
12
13     }
```

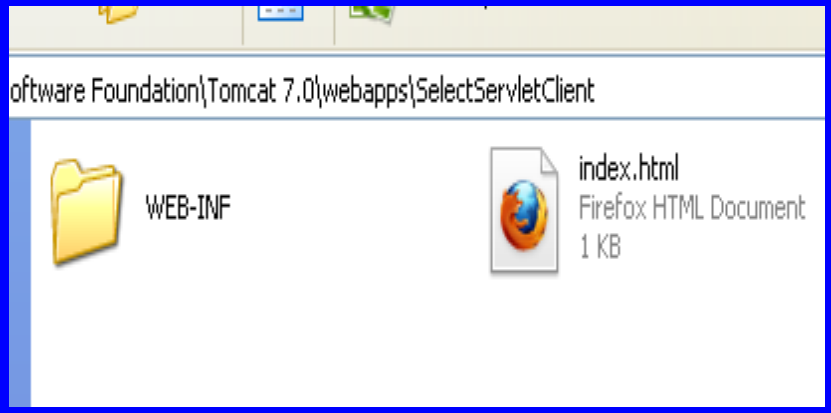
```
15      public HashMap executeSelect() throws RemoteException {
16
17          String driverName = "sun.jdbc.odbc.JdbcOdbcDriver";
18          String dbUrl = "jdbc:odbc:DSN3";
19          String sqlStmt = "select * from Student";
20          Connection con = null;
21          Statement stmt = null;
22          ResultSet rs = null;
23          HashMap hm = null;
24
25          try{
26              Class.forName(driverName);
27          }
28          catch(ClassNotFoundException cnfe){
29              cnfe.printStackTrace();
30          }
```

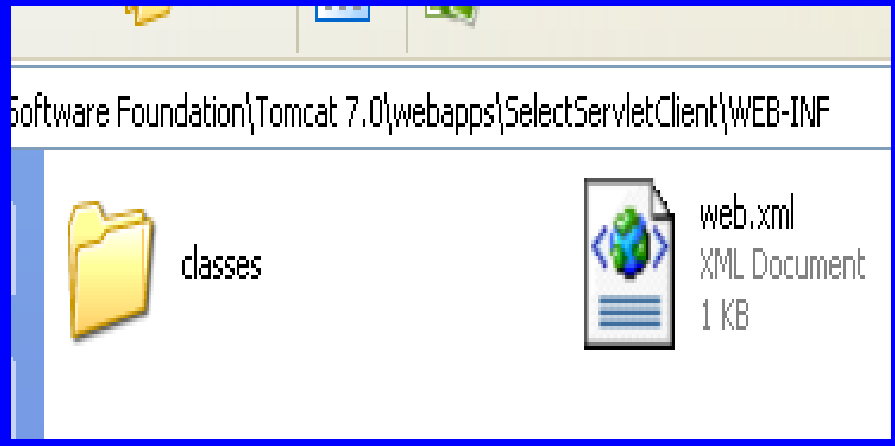
```
32          try{
33              con = DriverManager.getConnection(dbUrl);
34              stmt = con.createStatement();
35              rs = stmt.executeQuery(sqlStmt);
36              hm = new HashMap();
37
38              while(rs.next()){
39                  hm.put(new Integer(rs.getInt(1)), rs.getString(2));
40              }
41              con.close();
42
43          }
44          catch(SQLException sqle){
45              sqle.printStackTrace();
46          }
47          return hm;
48      }//executeSelect
49  }
```

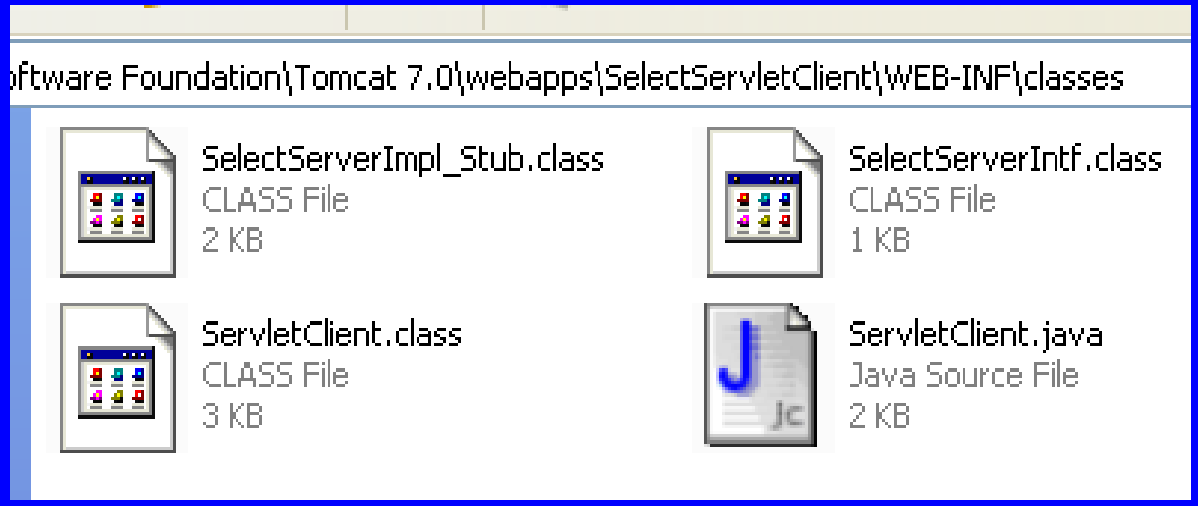
SelectServer.java

```
1  import java.rmi.*;
2  import java.rmi.server.*;
3  import java.rmi.registry.*;
4
5  public class SelectServer {
6      public static void main(String[] args){
7
8          try{
9              SelectServerImpl selectServerObj = new SelectServerImpl();
10
11             Registry createdRegistry = LocateRegistry.createRegistry(1099);
12             createdRegistry.rebind("SELECT-SERVER", selectServerObj);
13
14         }
15         catch(Exception e){
16             System.out.println("Error:" + e.getMessage());
17         }
18     }
19 }//SelectServer
```

SelectServletClient







Index.html

index.html - Notepad

File Edit Format View Help

```
<html>
 <head>
   <title>Servlet Client</title>
 </head>
 <body>
   <h2  align="center">Servlet Client</h2>

   <h1 align = "CENTER">Student Database</h1>
        <a href = "/SelectServletClient/ServletClient"><b>Invoke
Procedure</b></a>
 </body>
</html>
```

web.xml

```
- <web-app>
  - <servlet>
      <servlet-name>Client</servlet-name>
      <servlet-class>ServletClient</servlet-class>
    </servlet>
  - <servlet-mapping>
      <servlet-name>Client</servlet-name>
      <url-pattern>/ServletClient</url-pattern>
    </servlet-mapping>
  </web-app>
```

ServletClient.java

```
 8 public class ServletClient extends HttpServlet{
 9
10     SelectServerIntf selectServerIntf;
11
12     public void init(ServletConfig config){
13         try{
14             selectServerIntf = (SelectServerIntf)Naming.lookup("SELECT-SERVER");
15         }
16         catch(Exception e){
17             e.printStackTrace();
18         }
19     }//init()
```

```
21      public void doGet(HttpServletRequest req, HttpServletResponse res)
22          throws ServletException, IOException {
23
24              res.setContentType("text/html");
25              PrintWriter out = res.getWriter();
```

```
out.println("<HTML>");
out.println("<HEAD>");
out.println("<TITLE>Servlet Client for RMI service </TITLE></HEAD>");
out.println("<BODY>");
out.println("<TABLE BORDER=1 WIDTH=25% ALIGN=CENTER>");
out.println("<TR ALIGN=CENTER>");
out.println("<TH>Roll Number</TH>");
out.println("<TH>Name</TH>");
out.println("</TR>");
```

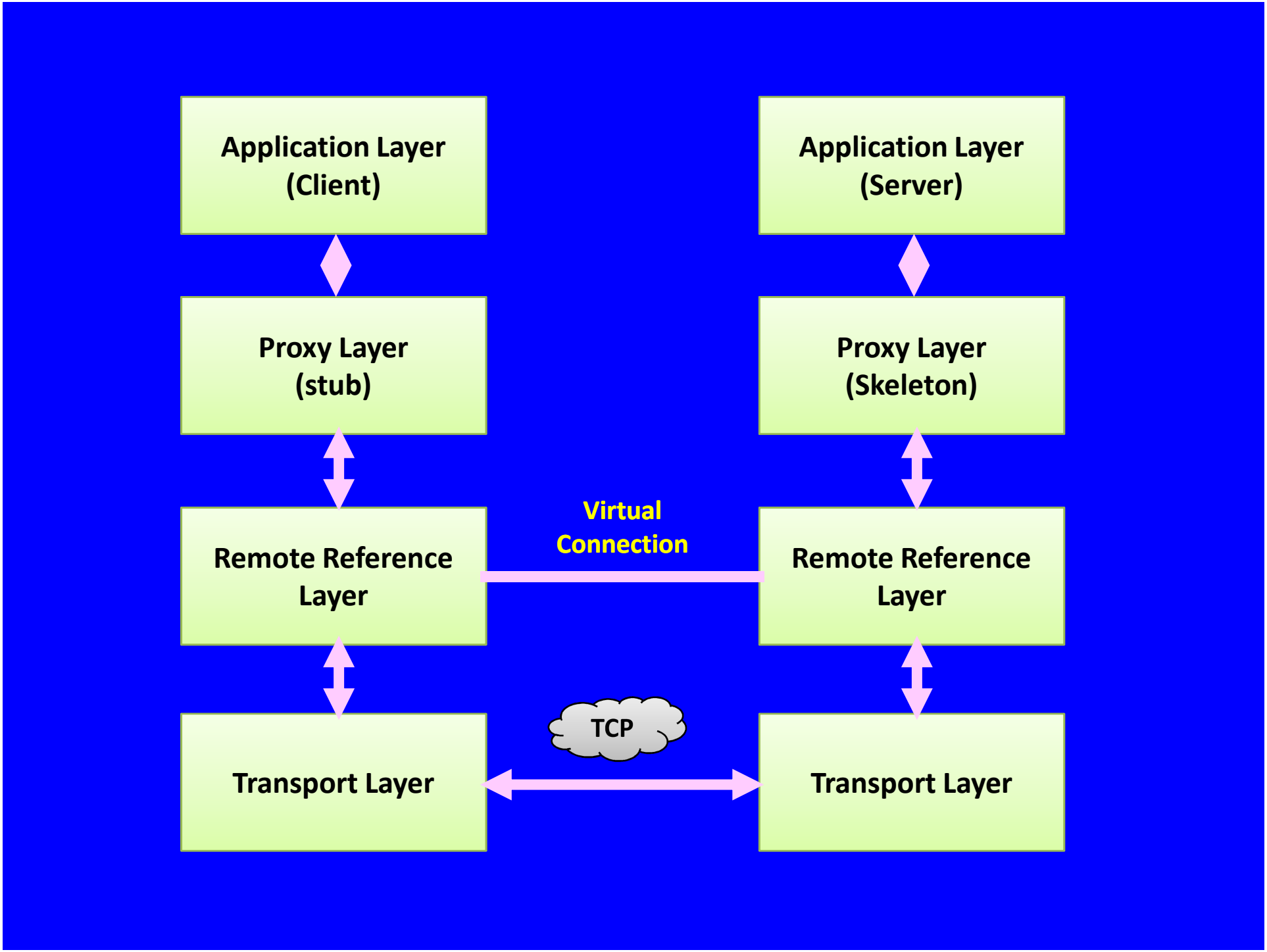
```
35              HashMap hm2 = selectServerIntf.executeSelect();
36              for(int i = 1; i <= hm2.size(); i++){
37                  if(hm2.containsKey(new Integer(i))){
38                      out.println("<TR>");
39                      out.println("<TH>" + i + "</TH>" );
40                      out.println("<TH>" + hm2.get(new Integer(i)) + "</TH>" );
41                      out.println("</TR>");
42                  }//if
43              }//for
44              out.println("</BODY>");
45              out.println("</HTML>");
46      }//doGet()
47  }//class:ServletClient
```

RMI Layers

RMI Layers

- Layers perform specific functions like establishing the connection, marshaling of parameters, etc.

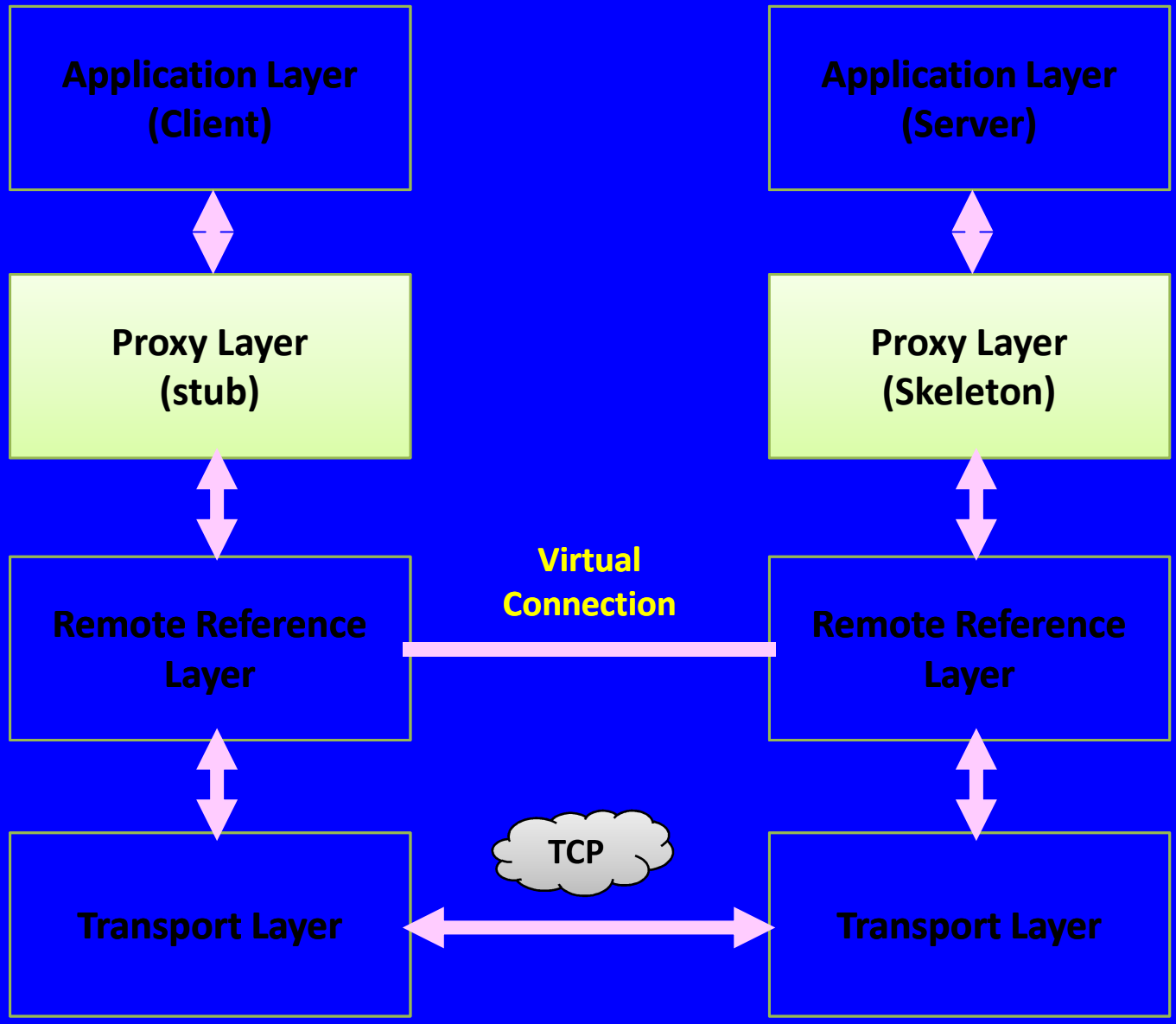
1. Application Layer
2. Proxy Layer
3. Remote Reference
4. Transport Layer


Application Layer
(Client)
Application Layer
(Server)
Proxy Layer
(stub)
Proxy Layer
(Skeleton)
Virtual
Connection
Remote Reference
Layer
Remote Reference
Layer
TCP
Transport Layer
Transport Layer

1. Application layer

- The application layer is the actual implementation of the client and server application.

2. Proxy layer



Stubs

- A stub acts as a client(s) local representative for remote object.
- The client invokes a method on the local stub. The stub is responsible for calling the method on the remote object.

  - Responsibilities of stub are:

    1. It connects to the remote JVM.
    2. It serializes any arguments to a remote method.
    3. It reads the values returned by a remote method.
    4. It returns the value to the client.

Skeleton

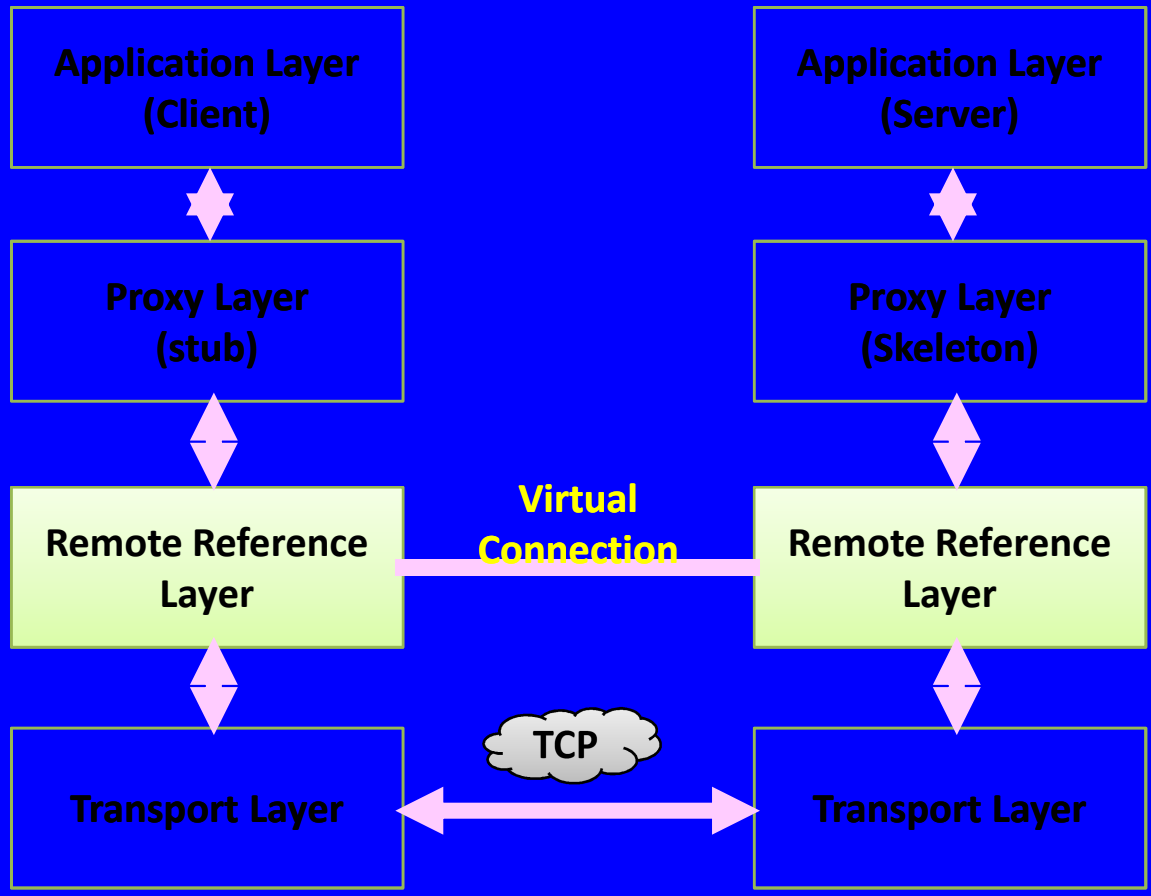
- The skeleton acts as a server side proxy of the remote objects.

❖ Responsibilities of skeleton are:

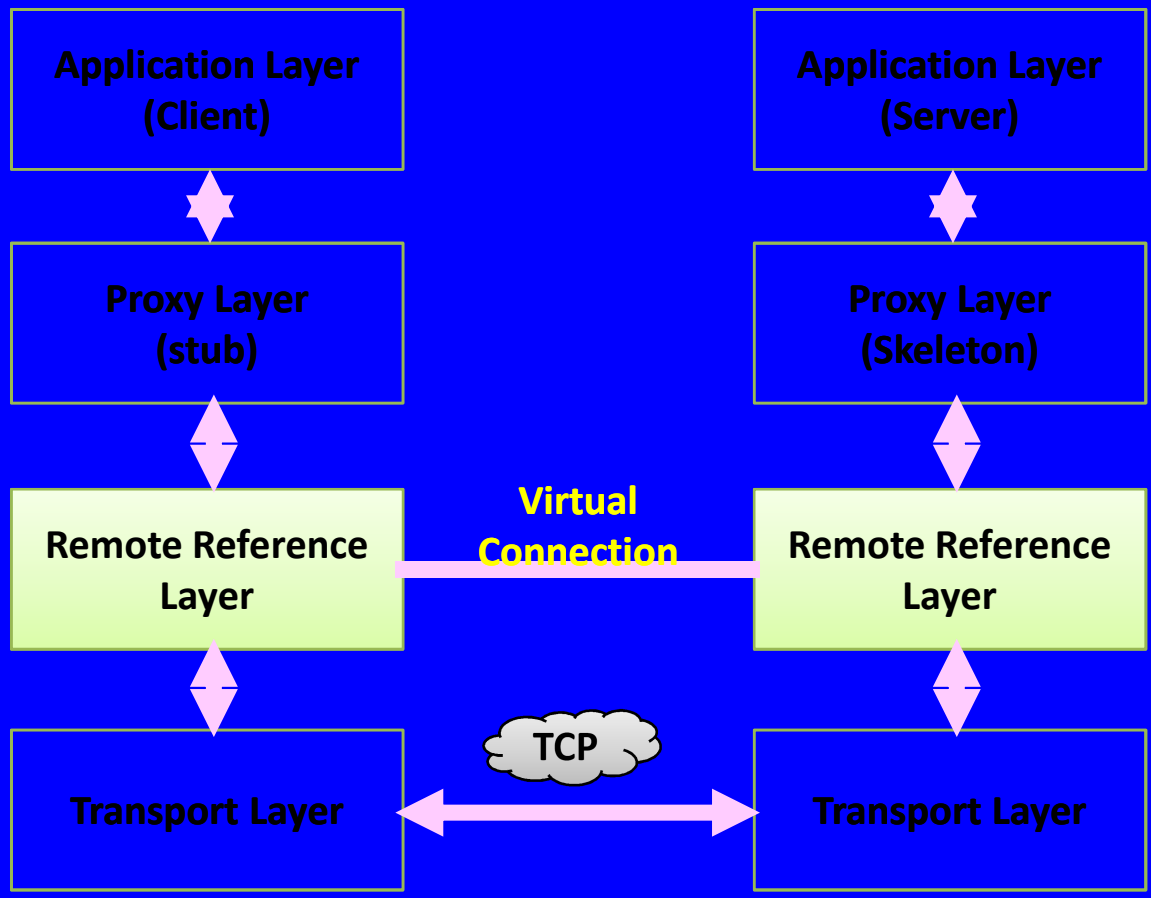
1. It reads the parameters for the remote object.
2. It locates the object to be called.
3. It invokes the desired method on remote object.
4. It transmits the result to the stub.

Remote Reference Layer

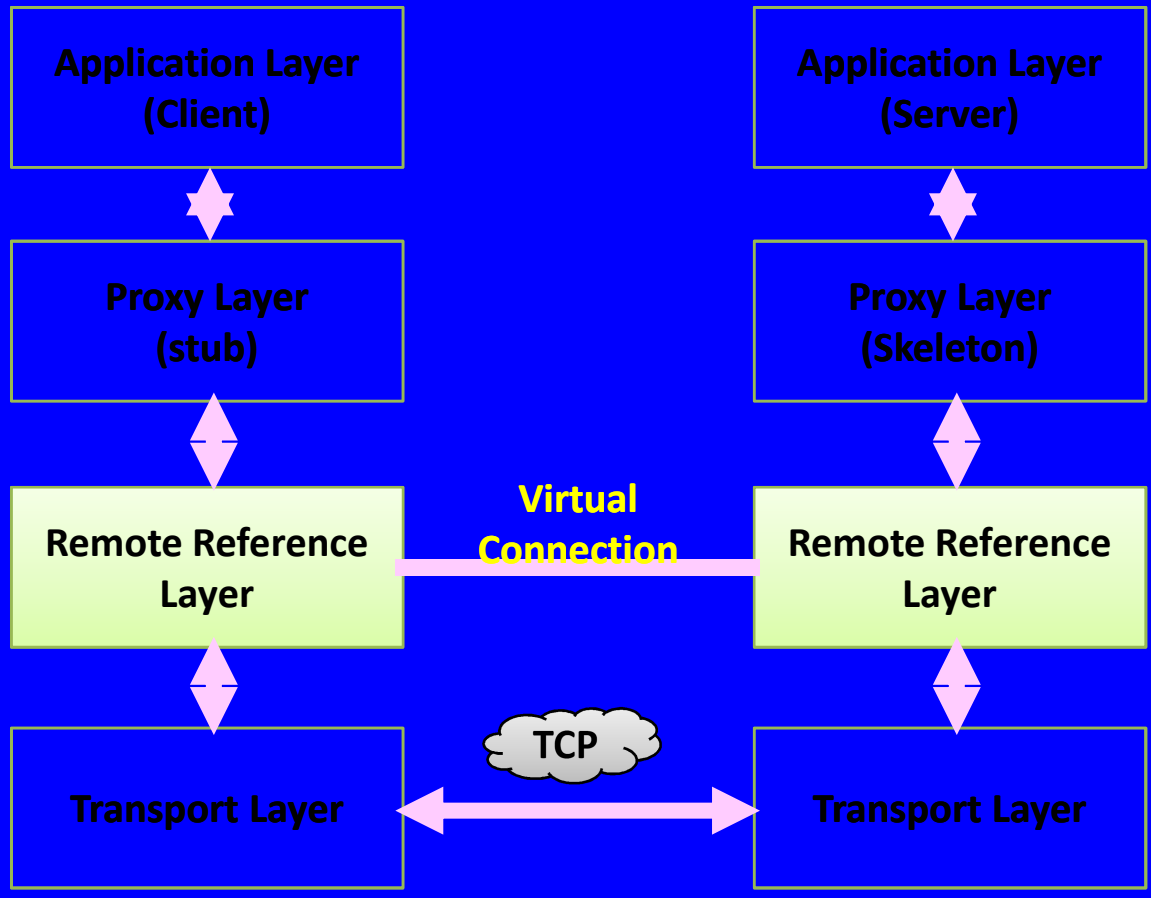
- The Remote Reference Layer is between the Transport layer and the Proxy layer.



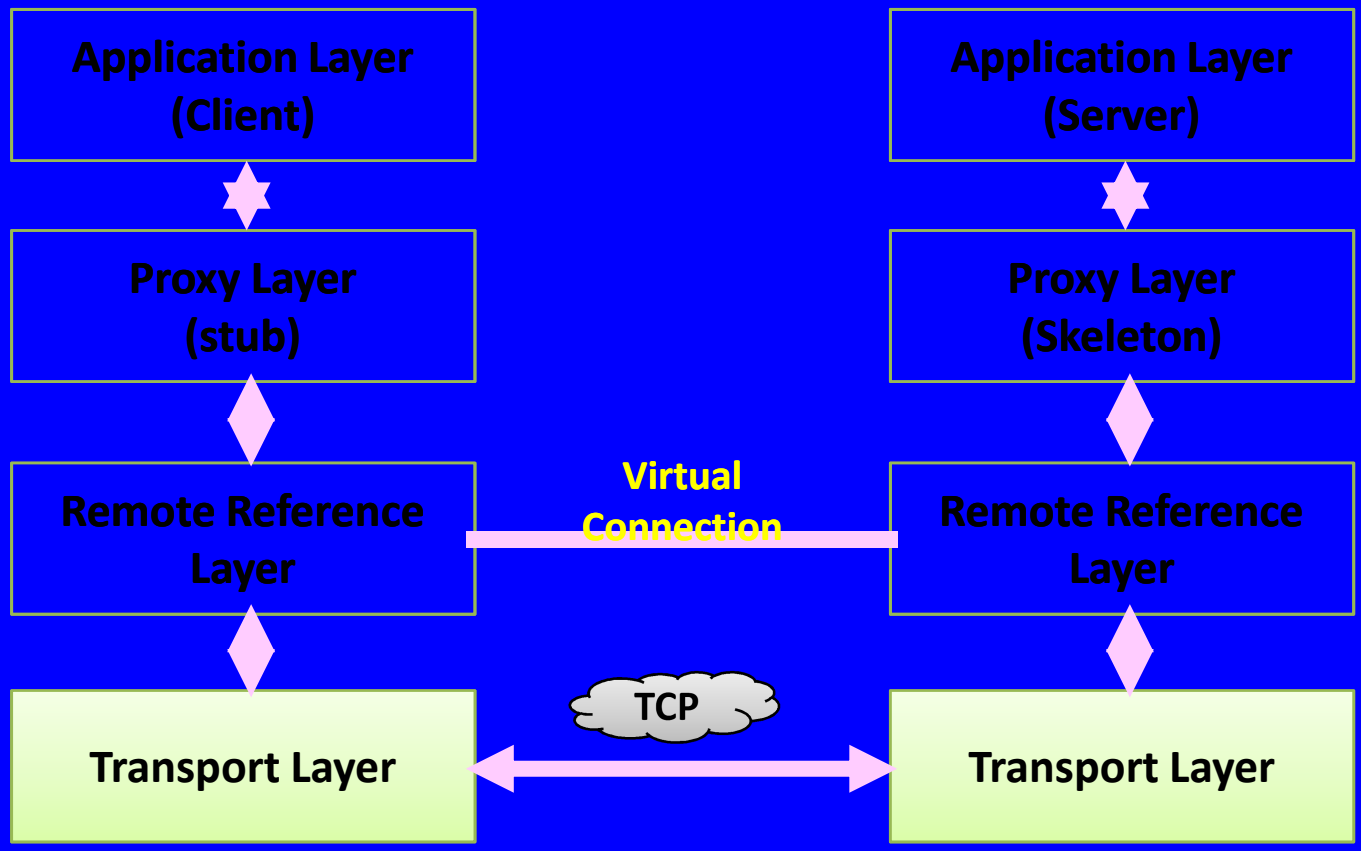
- It supports communication between the Stub and Skeleton.



- It is responsible for passing the stream-oriented data received from the Transport layer to the Proxy layer and vice versa.



Transport Layer



- Responsible for setting and managing connections to remote machines.

❖ Transport layer's task

- ❖ It sends to, and receives data from, other machines.
- ❖ When the transport layer receives a request from the client-side Remote Reference Layer, it establishes a socket connection to the server.
- ❖ If a significant amount of time passes with no activity on the connection, the transport layer is responsible for shutting down the connection.

Thank You!