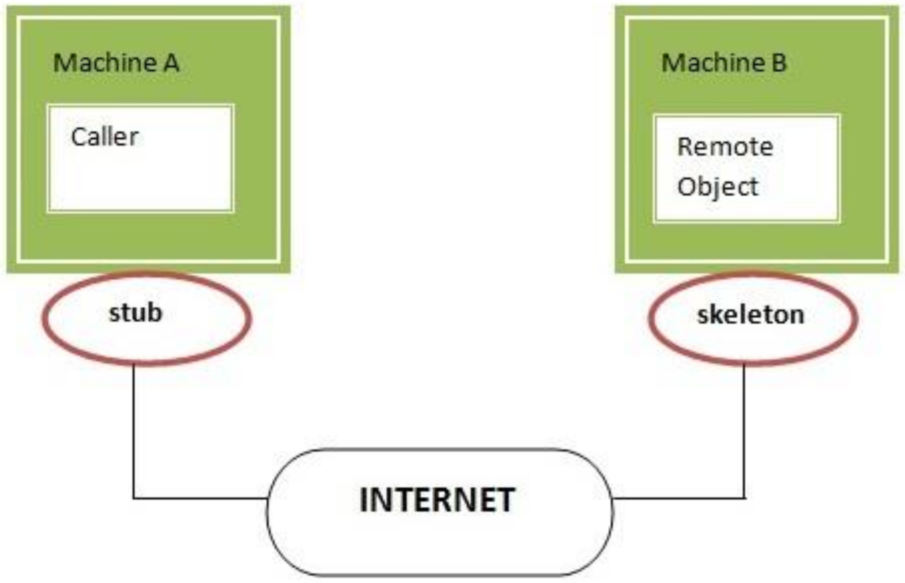
Program: 10

AIM: Write a java program to implement a Client/Server application using RMI.

THEORY:

Remote Method Invocation (RMI)

The RMI is an API that provides a mechanism to create distributed application in java. The RMI allows an object to invoke methods on an object running in another JVM. The RMI provides remote communication between the applications using two objects stub and skeleton.



Stub

The stub is an object, acts as a gateway for the client side. All the outgoing requests are routed through it. It resides at the client side and represents the remote object. When the caller invokes method on the stub object, it does the following tasks:

1. It initiates a connection with remote Virtual Machine (JVM),
2. It writes and transmits (marshals) the parameters to the remote Virtual Machine (JVM),
3. It waits for the result
4. It reads (unmarshals) the return value or exception, and
5. It finally, returns the value to the caller.

Skeleton

The skeleton is an object, acts as a gateway for the server-side object. All the incoming requests are routed through it. When the skeleton receives the incoming request, it does the following tasks:

1. It reads the parameter for the remote method
2. It invokes the method on the actual remote object, and
3. It writes and transmits (marshals) the result to the caller.

Creating and Running an RMI Application

The following steps are required to create and run an RMI application:

1. Create the remote interface
2. Provide the implementation class for the remote interface
3. Compile the implementation class and create the stub and skeleton objects using the rmic tool
4. Start the registry service by rmiregistry tool
5. Start the server
6. Create and start the client application

PROGRAM:

Create a Java project in Eclipse named *RMIChat*.

Step 1:

Create the remote interface named *RMI_Chat_Interface*

```
import java.rmi.Remote;

import java.rmi.RemoteException;


public interface RMI_Chat_Interface extends Remote {

	public void sendToServer(String message) throws RemoteException;

}
```

Step 2:

Provide the implementation class named *RMI_*Server for the remote interface.

```
import java.rmi.RemoteException;

import java.rmi.registry.LocateRegistry;

import java.rmi.registry.Registry;

import java.rmi.server.UnicastRemoteObject;

```

```
public class RMI_Server extends UnicastRemoteObject implements RMI_Chat_Interface {


        public RMI_Server() throws RemoteException {
                super();
        }


        @Override
        public void sendToServer(String message) throws RemoteException {
                System.out.println("Client says: " + message);
        }


        public static void main(String[] args) throws Exception {
                Registry rmiregistry = LocateRegistry.createRegistry(6000);
                rmiregistry.bind("chat", new RMI_Server());
                System.out.println("Chat server is running...");
        }
}
```

Step 3: Create a client class named *RMI_Client* and write the code.

```
import java.rmi.Naming;
import java.util.Scanner;


public class RMI_Client {


        static Scanner input = null;


        public static void main(String[] args) throws Exception {
                RMI_Chat_Interface chatapi = (RMI_Chat_Interface)
Naming.lookup("rmi://localhost:6000/chat");
                input = new Scanner(System.in);
                System.out.println("Connected to server...");
```

```
                    System.out.println("Type a message for sending to server...");
                    String message = input.nextLine();
                    while(!message.equals("Bye")) {
                              chatapi.sendToServer(message);
                              message = input.nextLine();
                    }
          }
}
```

INPUT AND OUTPUT:

Server

```
Chat server is running...
Client says: Hi
Client says: How
Client says: are you?
```

Client

```
Connected to server...
Type a message for sending to server...
Hi
How
are you?
```