**Aim: To implement Remote Method Invocation**

Lab Outcome:

Develop test and debug using Message-Oriented Communication or RPC/RMI based client-

server programs

Theory:

RMI stands for Remote Method Invocation. It is a mechanism that allows an object residing in one system (JVM) to access/invoke an object running on another JVM.

RMI is used to build distributed applications; it provides remote communication between Java programs. It is provided in the package java.rmi.

Architecture of an RMI Application

In an RMI application, we write two programs, a server program (resides on the server) and a client program (resides on the client).

• Inside the server program, a remote object is created and reference of that object is made available for the client (using the registry).

• The client program requests the remote objects on the server and tries to invoke its methods.

Codes :

Server.java

import java.rmi.\*;

import java.rmi.registry.\*;

public class Server {

public static void main(String[] args) {

try {

// Create a remote object

RemoteObject remoteObject = new RemoteObjectImpl();

// Bind the remote object to the RMI registry

Registry registry = LocateRegistry.createRegistry(37083);

registry.rebind("RemoteObject", remoteObject);

System.out.println("Server is ready...");

} catch (Exception e) {

System.err.println("Server exception: " + e.toString());

e.printStackTrace();

}

}

}

Client.java

import java.rmi.\*;

import java.rmi.registry.\*;

public class Client {

public static void main(String[] args) {

try {

// Get reference to the remote object from the RMI registry

Registry registry = LocateRegistry.getRegistry("localhost", 37083);

RemoteObject remoteObject = (RemoteObject) registry.lookup("RemoteObject");

// Invoke remote method

String message = remoteObject.sayHello();

System.out.println("Message from server: " + message);

// Invoke remote method to add two numbers

int result = remoteObject.add(5, 3);

System.out.println("Result of addition: " + result);

} catch (Exception e) {

System.err.println("Client exception: " + e.toString());

e.printStackTrace();

}

}

}

RemoteObject.java

import java.rmi.\*;

public interface RemoteObject extends Remote {

// Define remote methods to be invoked by the client

String sayHello() throws RemoteException;

int add(int a, int b) throws RemoteException;

}

RemoteObject.Impl.java

import java.rmi.\*;

import java.rmi.server.\*;

public class RemoteObjectImpl extends UnicastRemoteObject implements RemoteObject {

public RemoteObjectImpl() throws RemoteException {

super();

}

@Override

public String sayHello() throws RemoteException {

return "Hello from the server!";

}

@Override

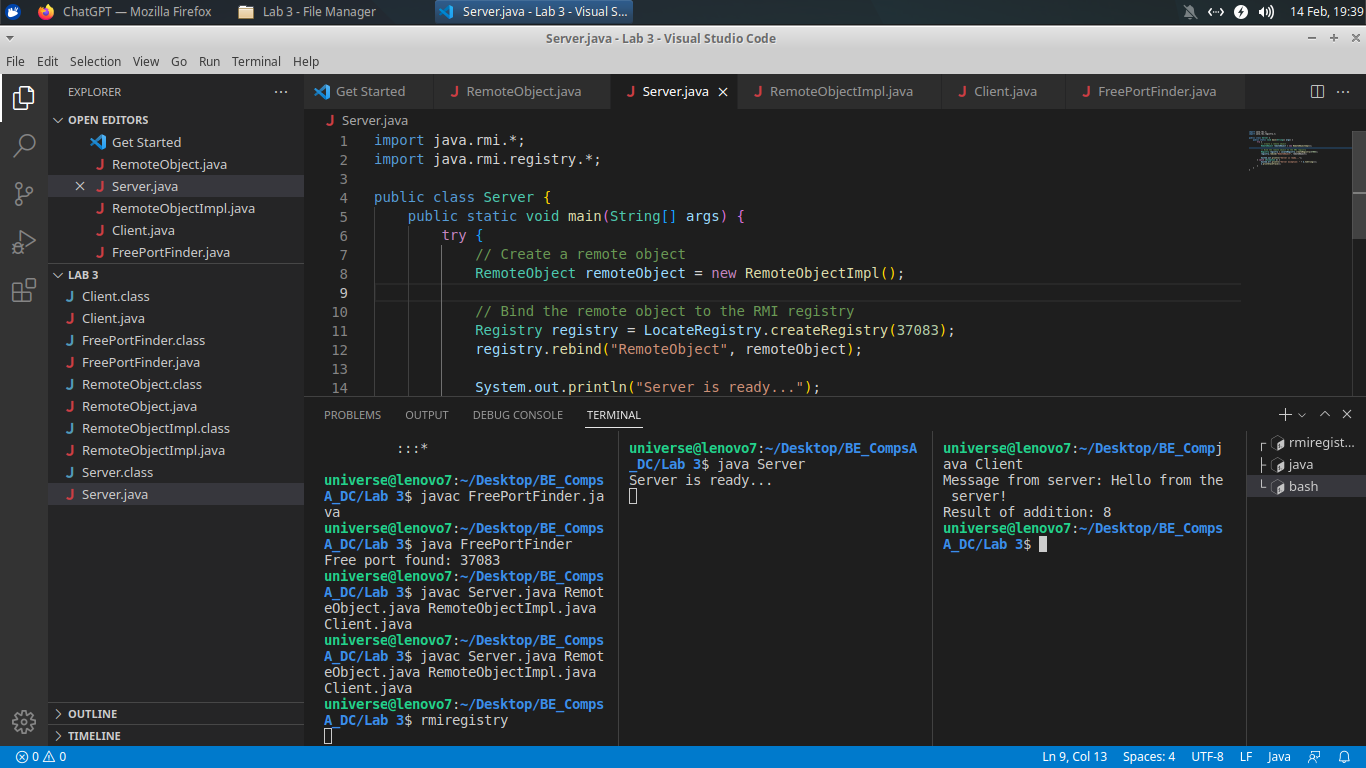
public int add(int a, int b) throws RemoteException {

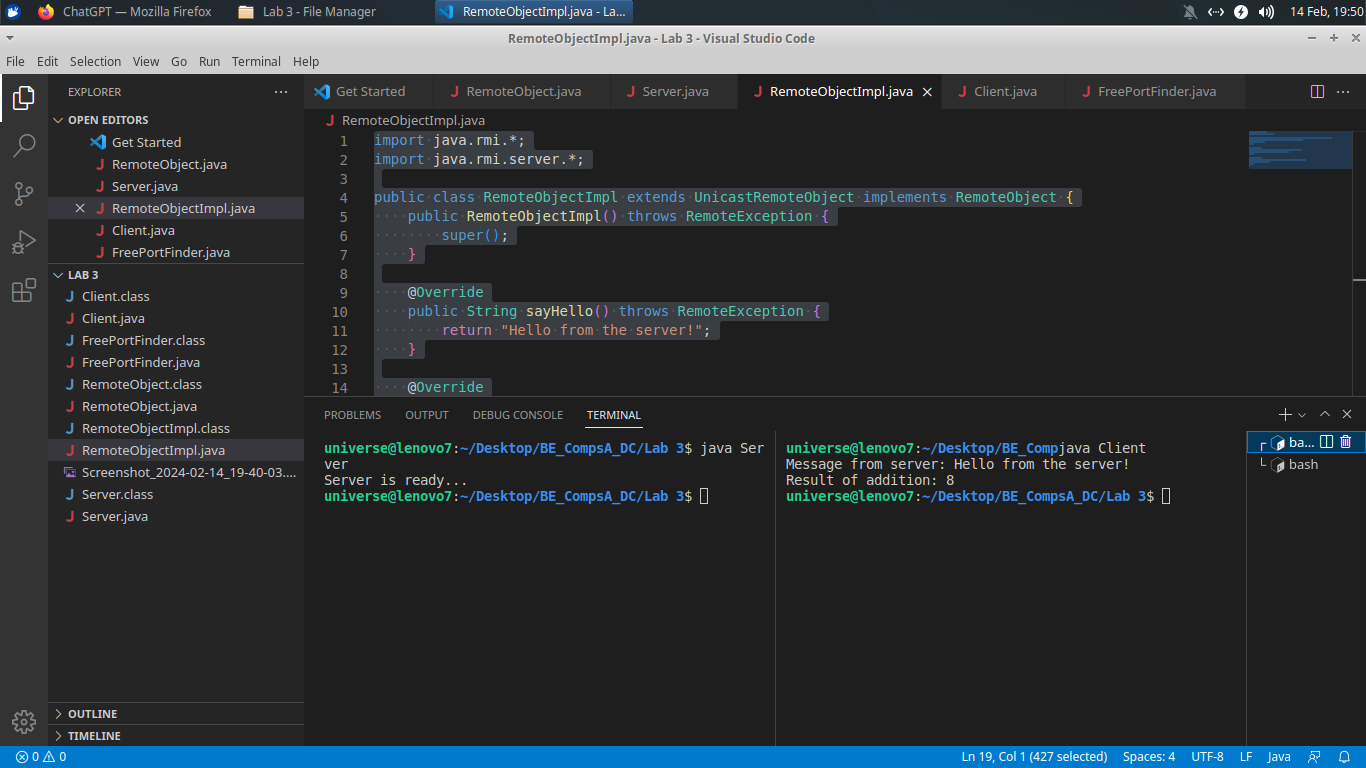
return a + b;

}

}

Output:





Working of the code :

1. **Server Program (Server.java)**:
   * This program serves as the server side of the RMI application.
   * It creates a remote object (RemoteObjectImpl) and makes it available for remote invocation by the client.
   * The server program binds the remote object to the RMI registry, making it accessible to clients.
2. **Remote Object Interface (RemoteObject.java)**:
   * This interface defines the methods that can be invoked remotely by the client.
   * In this example, it declares two methods:
     + sayHello(): A simple method that returns a greeting message.
     + add(int a, int b): A method that takes two integers as arguments and returns their sum.
3. **Remote Object Implementation (RemoteObjectImpl.java)**:
   * This class implements the RemoteObject interface.
   * It provides the actual implementation for the methods declared in the interface.
   * In this example, it implements the sayHello() method to return a greeting message and the add() method to perform addition.
4. **Client Program (Client.java)**:
   * This program serves as the client side of the RMI application.
   * It looks up the remote object (RemoteObject) from the RMI registry using the lookup() method.
   * It then invokes the remote methods (sayHello() and add()) on the remote object obtained from the registry.
   * The client program prints the results returned by the remote methods.

Overall Flow:

* The server program creates a remote object and binds it to the RMI registry.
* The client program looks up the remote object from the RMI registry and invokes remote methods on it.
* Remote method invocations are transparently handled by the RMI framework, allowing communication between the client and server over the network.

Postlab Questions:

1. What are the different times at which a client can be bound to a server?

2. How does a binding process locate a server?

3. Name some optimization methods adopted for better performance of distributed

applications using RPC and RMI.

