LAB 3

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.

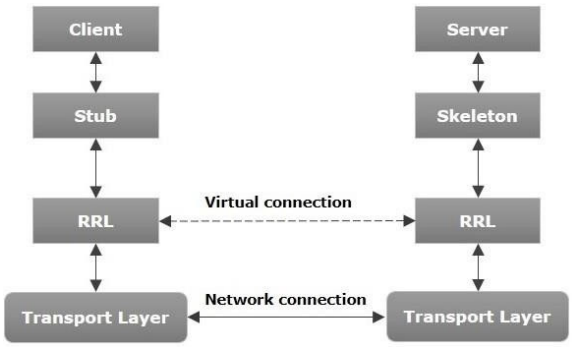
Architecture of an RMI Application

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.

The following diagram shows the architecture of an RMI application.

RMI Architecture



To write an RMI Java application, you would have to follow the steps given below –

- Define the remote interface
- Develop the implementation class (remote object)

- Develop the server program
- Develop the client program
- Compile the application
- Execute the application

Code:

Adder.java

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

AdderRemote.java

```
// Implementing the remote interface
public class AdderRemote implements Adder {
	// Implementing the interface method
	public int add(int x, int y) {
		return x+y;
	}
}
```

Client.java

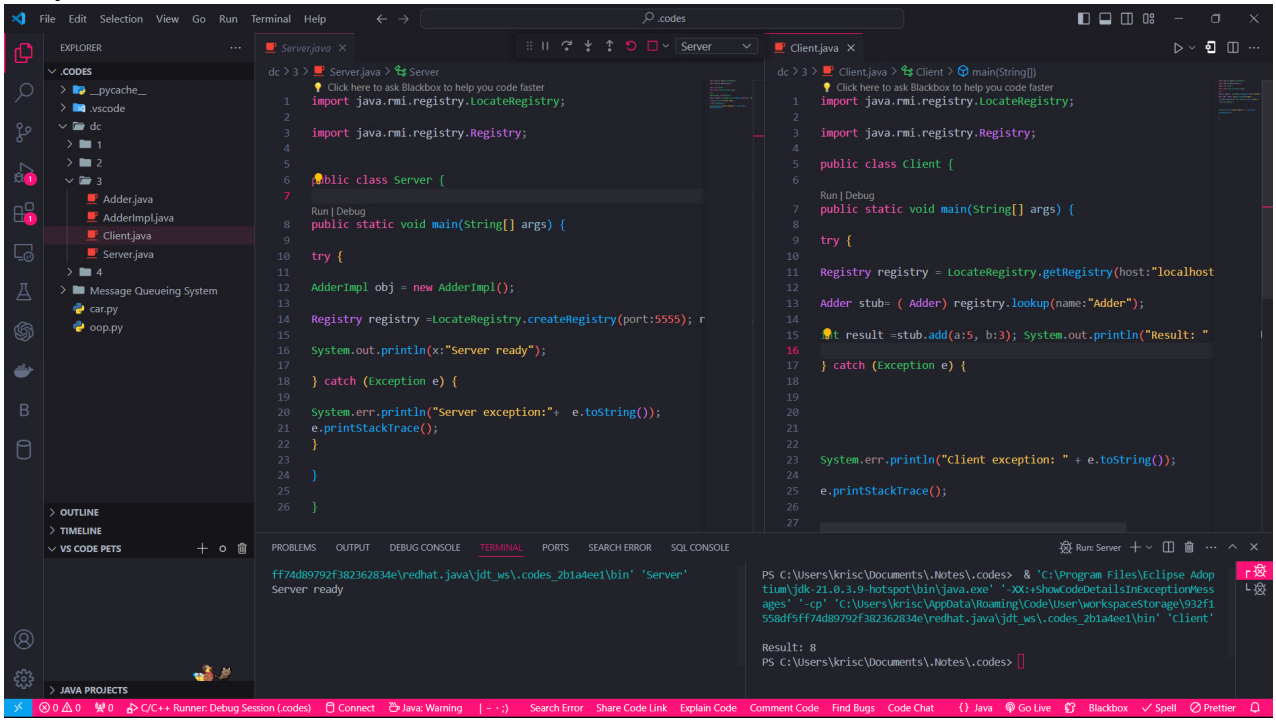
```
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;

public class Client {
   private Client() {}

public static void main(String[] args) {
      try {
         // Getting the registry
         Registry registry = LocateRegistry.getRegistry(null);

         // Looking up the registry for the remote object
         Adder stub = (Adder) registry.lookup("Hello");

         // Calling the remote method using the obtained object
```

```
         int result = stub.add(Integer.parseInt(args[0]),
Integer.parseInt(args[1]));
         System.out.println("Result From Server: " + result);

         // System.out.println("Remote method invoked");
      } catch (Exception e) {
         System.err.println("Client exception: " + e.toString());
         e.printStackTrace();
      }
   }
}
```

Server.java

```
import java.rmi.registry.Registry;
import java.rmi.registry.LocateRegistry;
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;

public class Server extends AdderRemote {
   public Server() {}
   public static void main(String args[]) {
      try {
         // Instantiating the implementation class
         AdderRemote obj = new AdderRemote();

         // Exporting the object of implementation class
         // (here we are exporting the remote object to the stub)
         Adder stub = (Adder) UnicastRemoteObject.exportObject(obj, 0);

         // Binding the remote object (stub) in the registry
         Registry registry = LocateRegistry.getRegistry();

         registry.bind("Hello", stub);
         System.err.println("Server ready");
      } catch (Exception e) {
         System.err.println("Server exception: " + e.toString());
         e.printStackTrace();
      }
   }
}
```

Output:



Conclusions:

In conclusion, the remote method invocation (RMI) in Java is a powerful technology that enables distributed computing across different machines connected via a network. The experiment performed on RMI has shown that it is possible to invoke methods on remote objects in Java using RMI, which can facilitate communication between different Java applications.

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.