Mehran University of Engineering and Technology, Khairpur
Department of Software Engineering

Course: SWE-Cloud Computing Practical			
Instructor	Dr. Asad Raza Malik	Practical/Lab No.	01
Date	22-07-2024	CLOs	03
Student's Roll no:		Point Scored:	
Date of Conduct:		Teacher's Signature:	

LAB PERFORMANCE INDICATOR	Subject Knowledge	Data Analysis and Interpretation	Ability to Conduct Experiment	Presentation	Calculation and Coding	Observation/ Result	Score

Topic	To work with Remote Method Invocation (RMI) API
Objective	• Understand the fundamentals of RMI. • Learn how to create and deploy RMI applications. • Gain hands-on experience with remote interfaces, implementations, and client-server communication. • Develop skills in object serialization and deserialization in the content of RMI.

Lab Discussion: Theoretical concepts and Procedural steps

Theory:

What is Remote Method Invocation (RMI)?

Remote Method Invocation (RMI) is a Java API that allows objects residing in different JVMs (Java Virtual Machines) to communicate with each other. It provides a way for a Java program running on one machine to invoke methods on an object running on another machine, making distributed computing easier and more intuitive.

Key Concepts of RMI

1. **Remote Interface**: Defines the methods that can be invoked remotely. It extends the java.rmi.Remote interface and declares that its methods can throw RemoteException.
2. **Remote Object**: Implements the remote interface and provides the actual implementation of the remote methods. It extends java.rmi.server.UnicastRemoteObject to make it a remote object that can be accessed from other JVMs.
3. **RMI Registry**: A service that allows remote objects to be registered and looked up. The registry listens on a specific port (default is 1099) and provides a way for clients to find and connect to remote objects.

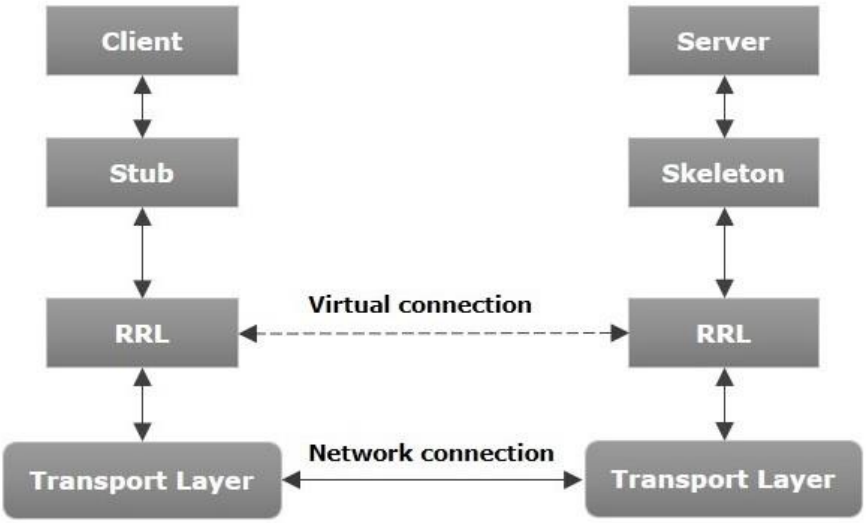
Remote Method Invocation (RMI) in Java Language.

RMI provides for remote communication between programs written in the Java programming language. A general technique for executing code on objects distributed across a network. Remote language. A general technique for executing code on objects distributed across a network. Remote language. A general technique for executing code on objects distributed across a network. Remote Method Invocation (RMI) is an API in Java that allows an object to

invoke a method on an object that exists in another address space, which could be on the same machine or a remote machine. Through RMI, an object running in a JVM present on a computer (Client side) can invoke methods on an object present in another JVM (Server side). RMI creates a public remote server object that enables client and server-side communications through simple method calls on the server object.

Concept of Remote Method Invocation (RMI) Application

An RMI application can be divided into two parts, Client program and Server program. A Server program creates some remote objects; and makes their references available for the client to invoke a method on it. A client program requests remote objects on the server and invokes methods on them. Stub and Skeleton are two important objects used for communication with remote objects.



- **Transport Layer** − This layer connects the client and the server. It manages the existing connection and sets up new connections.
- **Stub** − A stub is a representation (proxy) of the remote object at the client. It resides in the client system; it acts as a gateway for the client program.
- **Skeleton** − This is the object which resides on the server side. **stub** communicates with this skeleton to pass requests to the remote object.
- **RRL (Remote Reference Layer)** − It is the layer that manages the references made by the client to the remote object.

Working on an RMI Application

The following point summarizes how an RMI application works.

1. When the client makes a call to the remote object, it is received by the stub which eventually passes this request to the RRL.
2. When the client-side RRL receives the request, it invokes a method called **invoke()** of the object **remoteRef**. It passes the request to the RRL on the server side.
3. The RRL on the server side passes the request to the Skeleton (proxy on the server) which finally invokes the required object on the server.

4. The result is passed back to the client.

Marshalling and Unmarshalling

Whenever a client invokes a method that accepts parameters on a remote object, the parameters are bundled into a message before being sent over the network. These parameters may be of primitive type or objects. In case of primitive type, the parameters are put together and a header is attached to it. In case the parameters are objects, then they are serialized. This process is known as **marshalling**.

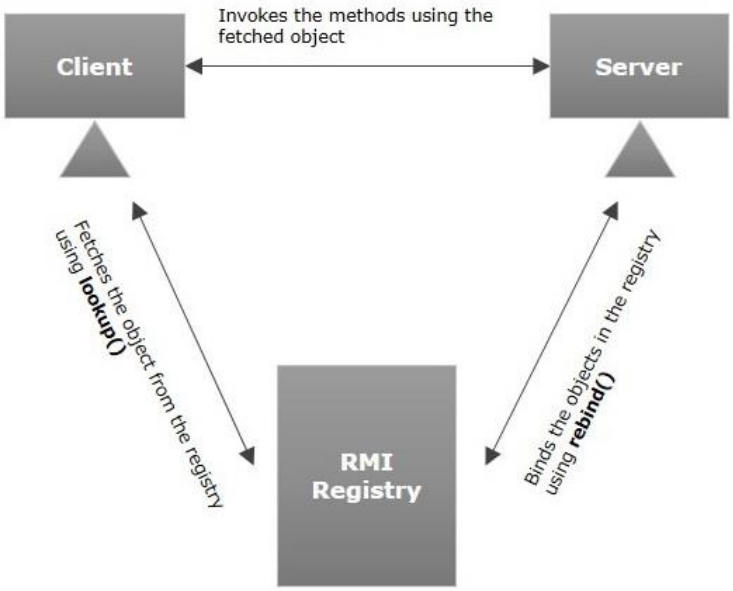
At the server side, the packed parameters are unbundled and then the required method is invoked. This process is known as **unmarshalling**.

RMI Registry

RMI registry is a namespace on which all server objects are placed. Each time the server creates an object, it registers this object with the RMIregistry
(using **bind()** or **reBind()** methods). These are registered using a unique name known as **bind name**.

To invoke a remote object, the client needs a reference of that object. At that time, the client fetches the object from the registry using its bind name (using **lookup()** method).

The following illustration explains the entire process –



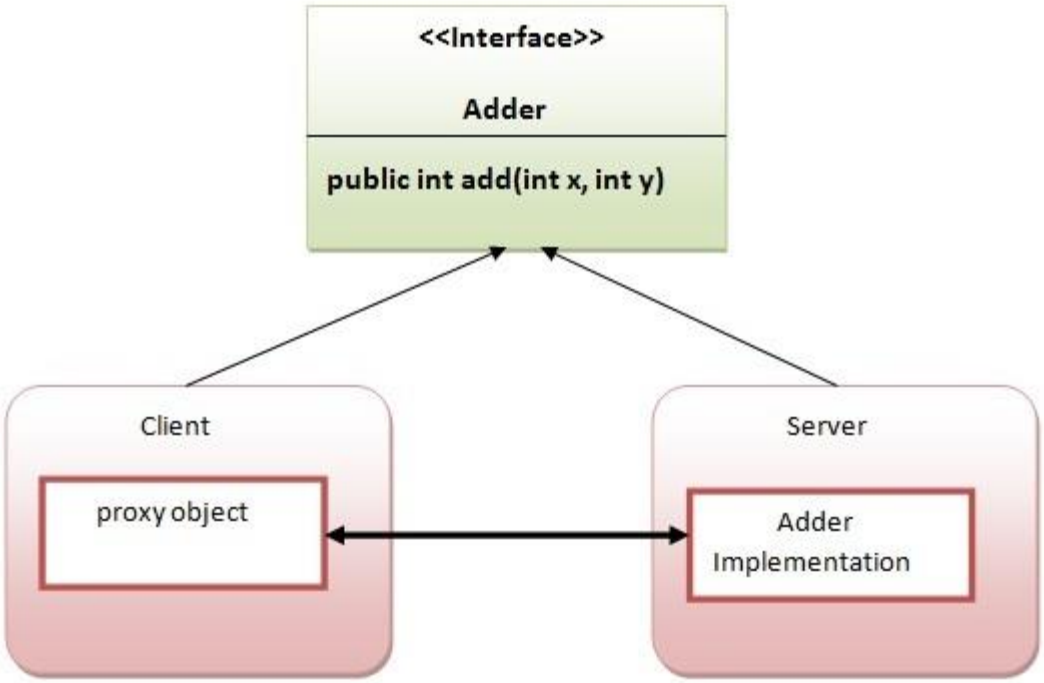
5. Goals of RMI
6. Following are the goals of RMI −
7. To minimize the complexity of the application.
8. To preserve type safety.
9. Distributed garbage collection.
10. Minimize the difference between working with local and remote objects.

RMI Application steps.

1. Define the remote interface
2. Develop the implementation class (remote object)
3. Develop the server program
4. Develop the client program
5. Compile the application
6. Execute the application.

RMI Application Example Program

In this example, we have followed all 6 steps to create and run the RMI application. The client application needs only two files, remote interface and client application. In the RMI application, both the client and server interact with the remote interface. The client application invokes methods on the proxy object, and RMI sends the request to the remote JVM. The return value is sent back to the proxy object and then to the client application.



Lab Activate:

1. Define the remote interface.

Adder:

```
package Lab1;
import java.rmi.*;
public interface Adder extends Remote {
   public int add(int x, int y) throws RemoteException;
}
```

2. Develop the implementation class (remote object)

Adder Remote:

```
package Lab1;
import java.rmi.server.*;
import java.rmi.*;
public class AdderRemote extends UnicastRemoteObject implements Adder{
   AdderRemote() throws RemoteException{
       super();
   }
   @Override
   public int add(int x, int y) throws RemoteException {
       return x+y;
   }
}
```

3. Develop the sever program

Myserver:

```
package Lab1;
import java.rmi.registry.*;
import java.rmi.*;
public class MyServer {
   public static void main(String[] args) {
       try {
           Adder stub = new AdderRemote();
           LocateRegistry.createRegistry(1900);
           Naming.rebind("rmi://localhost:1900/MyServer",stub);
       }catch (Exception e){
           System.out.println(e);
       }
   }
}
```

4. Develop the Client program

MyClient:

```
package Lab1;
import java.rmi.*;
public class MyClient {
   public static void main(String[] args) {
       try {
           Adder stub = (Adder)
           Naming.lookup("rmi://localhost:1900/MyServer");
           System.out.print("Addition = ");
           System.out.print(stub.add(23,25));
       }catch (Exception e){
   }
}}
```

5. Compile the application

Compile all 4 classes with Javac

Compile the implementation class AddrerRemote with rmic

Rmic AdderRemote

(this will generate a file stub and skeleton)

- ***Compile the java files.***

```
javac *.java
```

- ***Start the RMI registry.***

```
rmiregistry &
```

- ***Run the server***

```
java Server
```

- ***In another terminal, run the client***

```
java Client
```

The out of a Remote Method Invocation (RMI) application depends on the interactions between the client and the server, specifically on the methods defined in the remote interface and their implementations.

Lab Tasks

1. **Perform the classwork task, add code, and include output screenshots.**
2. **Create a simple RMI program that prints a custom message on the client screen. Add code and output screenshots.**
3. **Create an RMI program in which the server performs basic arithmetic operations. Add code and output screenshots.**

Conclusion:

Have you become familiar with the basic concepts of RMI programming? What concepts did you gain from this practical session about RMI programming? Comment.