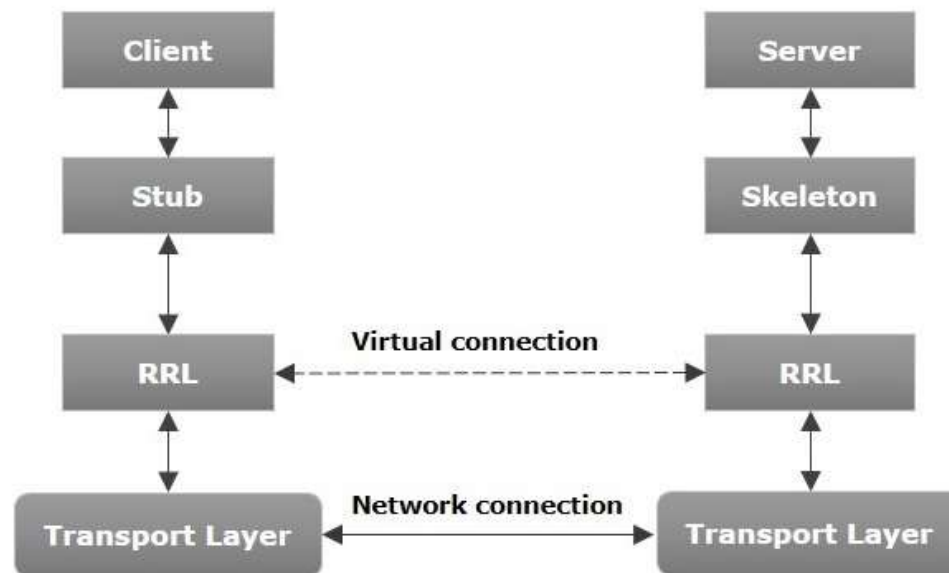


Remote Method Invocation (RMI)

RMI Introduction

- ❑ 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. Java RMI is one of several methods for creating distributed applications in Java. Others include Jini, JavaSpaces, Java Message Service (JMS), and Enterprise JavaBeans (EJBs).
- ❑ It is provided in the package **java.rmi**.
- ❑ 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 a 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.

Architecture of an RMI Application



Architecture of an RMI Application

Transport Layer – This layer connects the client and the server. It manages the existing connection and also sets up new connections.

Stub – A stub is a representation (proxy) of the remote object at 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 request to the remote object.

RRL(Remote Reference Layer) – It is the layer which manages the references made by the client to the remote object.



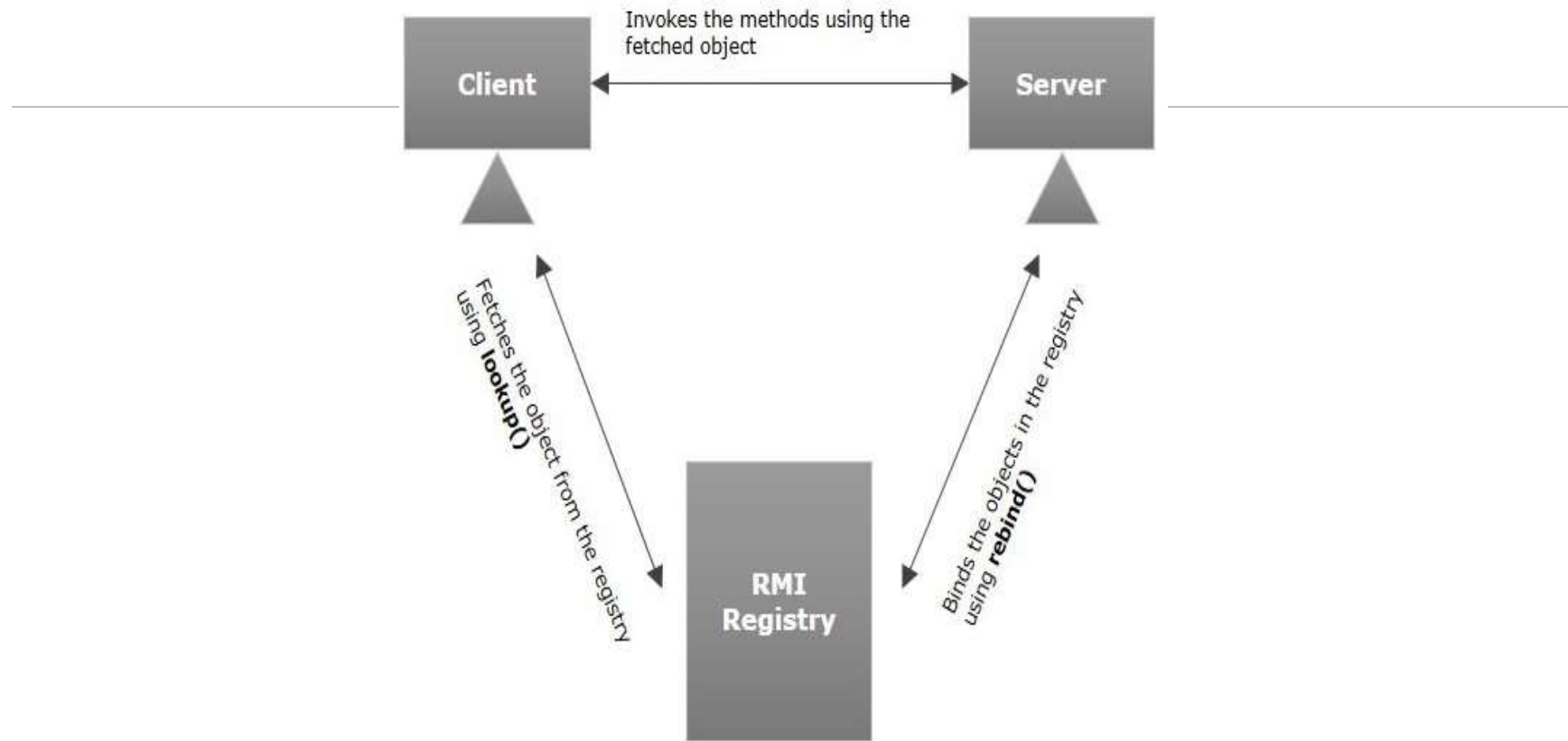
Working of an RMI Application

- ❑ When the client makes a call to the remote object, it is received by the **stub** which eventually passes this request to the RRL.
- ❑ 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.
- ❑ 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.
- ❑ The result is passed all the way back to the client.


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).

RMI Registry-Processes



RMI Components

- **RMI Interfaces:** Defines the methods that can be invoked remotely.
 - **RMI Implementation Classes:** Contains the actual code for the methods defined in the interface.
 - **RMI Client and Server:** The client is the JVM that invokes methods on a remote object. The server is the JVM that hosts the remote object.
- 

STEPS-RMI Programming

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


Defining the Remote Interface

- ❑ A remote interface provides the description of all the methods of a particular remote object. The client communicates with this remote interface.
- ❑ Create an interface that extends the predefined interface **Remote** which belongs to the package.
- ❑ Declare all the business methods that can be invoked by the client in this interface.
- ❑ Since there is a chance of network issues during remote calls, an exception named **RemoteException** may occur; throw it.

Remote Interface

```
import java.rmi.Remote;  
import java.rmi.RemoteException;  
  
public interface HelloService extends Remote {  
    String sayHello() throws RemoteException;  
}
```

Developing the Implementation Class (Remote Object)

- ❑ We can write an implementation class separately or we can directly make the server program implement this interface.
 - ❑ Implement the interface created in the previous step.
 - ❑ Provide implementation to all the abstract methods of the remote interface.
- 

Implementing the remote interface

// Implementing the remote interface


```
import java.rmi.RemoteException;

public class HelloServiceImpl extends
UnicastRemoteObject implements HelloService {

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

    public String sayHello() throws RemoteException {
        return "Hello, world!";
    }
}
```

Developing the Server Program

- ❑ An RMI server program should implement the remote interface or extend the implementation class.
 - ❑ Here, we should create a remote object and bind it to the RMIregistry.
- 

To develop a server program –

- **Create a Server class** from where you want invoke the remote object.
- **Create a remote object** by instantiating the implementation class as shown below.
- **Export the remote object** using the method **exportObject()** of the class named **UnicastRemoteObject** which belongs to the package **java.rmi.server**.
- **Get the RMI registry** using the **getRegistry()** method of the **LocateRegistry** class which belongs to the package **java.rmi.registry**.
- **Bind the remote object created to the registry using the bind() method** of the class named **Registry**. To this method, pass a string representing the bind name and the object exported, as parameters.

```
import java.rmi.registry LocateRegistry;
import java.rmi.registry Registry;
import java.rmi.Naming;
public class HelloServer {
    public static void main(String[] args) {
        try {
            HelloService helloService = new HelloServiceImpl();
            Registry registry = LocateRegistry.createRegistry(1099);
            registry.rebind("HelloService", helloService);
            System.out.println("HelloService started...");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Developing the Client Program

Write a client program, fetch the remote object and invoke the required method using this object.

To develop a client program –

- **Create a client class** from where your intended to invoke the remote object.
- **Get the RMI registry** using the **getRegistry()** method of the **LocateRegistry** class which belongs to the package **java.rmi.registry**.
- **Fetch the object from the registry** using the method **lookup()** of the class **Registry** which belongs to the package **java.rmi.registry**.
- To this method, you need to pass a string value representing the bind name as a parameter. This will return you the remote object.
- The **lookup()** returns an object of type remote, down cast it to the type Hello.
- Finally invoke the required method using the obtained remote object.

```
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
public class HelloClient {
    public static void main(String[] args) {
        try {
            Registry registry =
                LocateRegistry.getRegistry("localhost", 1099);
            HelloService helloService = (HelloService)
                registry.lookup("HelloService");

            String response = helloService.sayHello();
            System.out.println("Response from server: " +
                response);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Compiling the Application

- ☐ Compile the Remote interface.
- ☐ Compile the implementation class.
- ☐ Compile the server program.
- ☐ Compile the client program.

OR

Open the folder where you have stored all the programs and compile all the Java files as shown below.

```
>javac *.java
```

Executing the Application

Step 1 – Start the **rmi** registry using the following command(manual) or start in the code. In the above example, it is included in the code itself)

You can start the RMI registry within your Java application using the `LocateRegistry.createRegistry(int port)` method, which creates and exports a Registry instance on the local host that accepts requests on the specified port.

start rmiregistry 1099 & (manually)

(This will start an **rmi** registry on a separate window)

Step 2 – Run the server class file (**java** `HelloServer`)

Step 3 – Run the client class file (**java** `HelloClient`)

