Swami Keshvanand Institute of Technology, Jaipur

Remote Method Invocation

Presented By:
Sohan Gupta
Assistant Professor

Remote Method Invocation(RMI)

- 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.

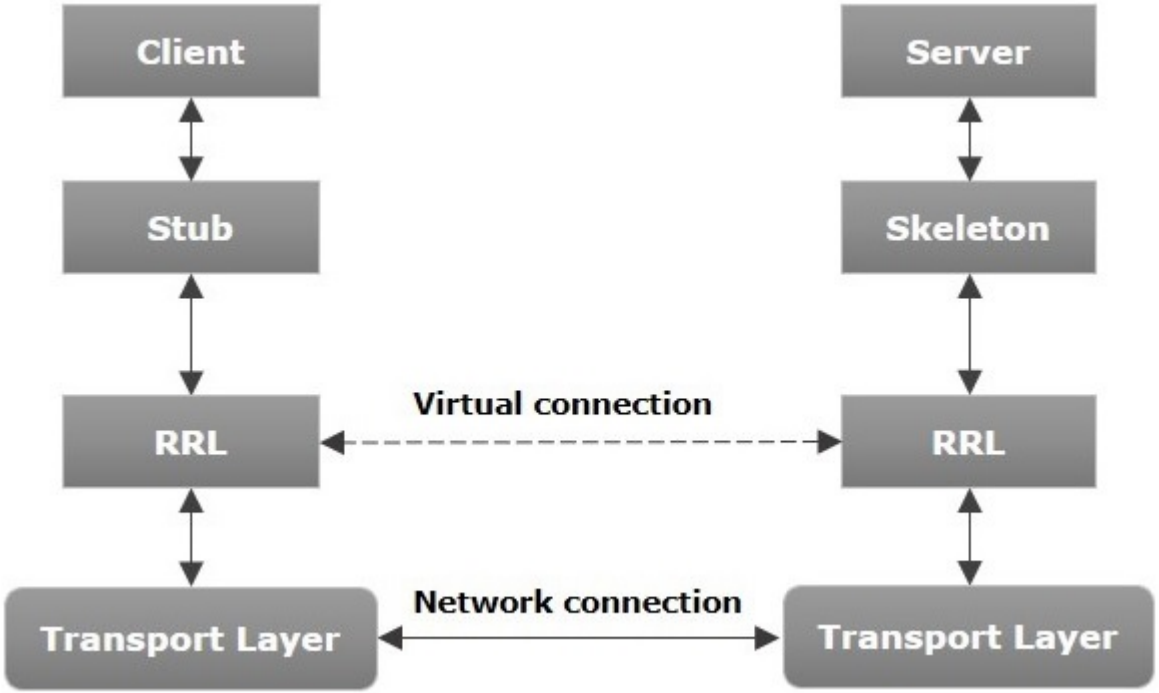


Fig: Architecture of an RMI application

- **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.

Stub: the stub object perform the following tasks:

- It initiates a connection with remote Virtual Machine (JVM),
- It writes and transmits the parameters to the remote Virtual Machine (JVM),
- It waits for the result
- It reads the return value or exception, and
- It finally, returns the value to the caller.

Skeleton : the skeleton object perform the following tasks:

- It reads the parameter for the remote method
- It invokes the method on the actual remote object, and
- It writes and transmits the result to the caller.

Working of an RMI Application

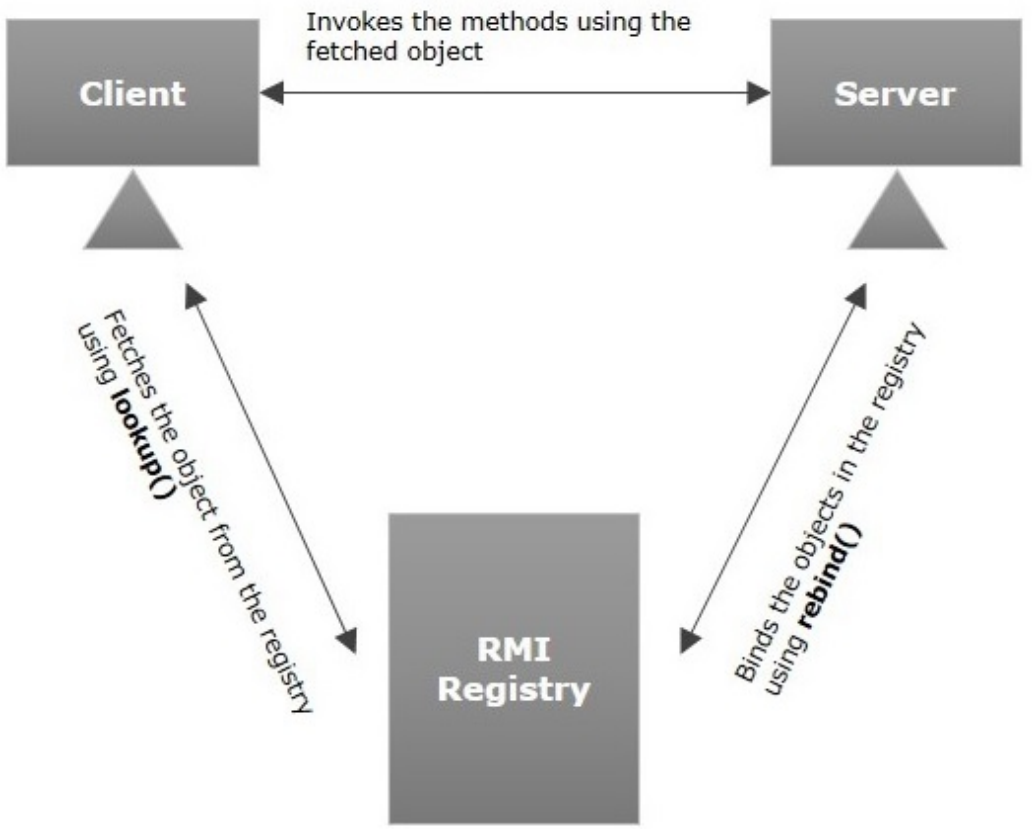
The following points summarize how an RMI application works –

- When the client makes a call to the remote object, it is received by the stub which 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).

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 −



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

1. **Create** the remote interface
2. Provide the **implementation** of the remote interface
3. **Create the stub and skeleton** objects using the rmic tool.
4. **Start the registry** service by the rmiregistry tool
5. **Create and run the server** application
6. **Create and run the client** application

1. Create the remote interface

For creating the remote interface, extend the Remote interface and declare the RemoteException with all the methods of the remote interface. Here, we are creating a remote interface that extends the Remote interface. There is only one method named add() and it declares RemoteException.

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

2. implementation the remote interface

- For providing the implementation of the Remote interface, we need to
  - Either extend the UnicastRemoteObject class,
  - or use the exportObject() method of the UnicastRemoteObject class
- In case, you extend the UnicastRemoteObject class, you must define a constructor that declares RemoteException.

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

3) create the stub and skeleton objects using the rmic tool

Next step is to create stub and skeleton objects using the rmi compiler. The rmic tool invokes the RMI compiler and creates stub and skeleton objects.

```
rmic AdderRemote
```

4) Start the registry service by the rmiregistry tool

- Now start the registry service by using the rmiregistry tool.
- If you don't specify the port number, it uses a default port number. In this example, we are using the port number 5000.

```
rmiregistry 5000
```

5) Create and run the server application

- Now rmi services need to be hosted in a server process.
- The Naming class provides methods to get and store the remote object. The Naming class provides 5 methods.

public static java.rmi.Remote **lookup**(java.lang.String) throws java.rmi.NotBoundException, java.net.MalformedURLException, java.rmi.RemoteException;	It returns the reference of the remote object.
public static void **bind**(java.lang.String, java.rmi.Remote) throws java.rmi.AlreadyBoundException, java.net.MalformedURLException, java.rmi.RemoteException;	It binds the remote object with the given name.
public static void **unbind**(java.lang.String) throws java.rmi.RemoteException, java.rmi.NotBoundException, java.net.MalformedURLException;	It destroys the remote object which is bound with the given name.
public static void **rebind**(java.lang.String, java.rmi.Remote) throws java.rmi.RemoteException, java.net.MalformedURLException;	It binds the remote object to the new name.
public static java.lang.String[] **list**(java.lang.String) throws java.rmi.RemoteException, java.net.MalformedURLException;	It returns an array of the names of the remote objects bound in the registry.

- In this example, we are binding the remote object by the name sonoo.

```
import java.rmi.*;
import java.rmi.registry.*;
public class MyServer{
public static void main(String args[]){
try{
Adder stub=new AdderRemote();
Naming.rebind("rmi://localhost:5000/sonoo",stub);
}catch(Exception e){System.out.println(e);}
}
}
```

6) Create and run the client application

- At the client we are getting the stub object by the lookup() method of the Naming class and invoking the method on this object.
- In this example, we are running the server and client applications, in the same machine so we are using localhost.
- If you want to access the remote object from another machine, change the localhost to the host name (or IP address) where the remote object is located.

```
import java.rmi.*;
public class MyClient{
public static void main(String args[]){
try{
Adder stub=(Adder)Naming.lookup("rmi://localhost:5000/sonoo");
System.out.println(stub.add(34,4));
}catch(Exception e){}
}
}
```

For running **this** rmi example,

1) compile all the java files

    javac *.java

2)create stub and skeleton object by rmic tool

    rmic AdderRemote

3)start rmi registry in one command prompt

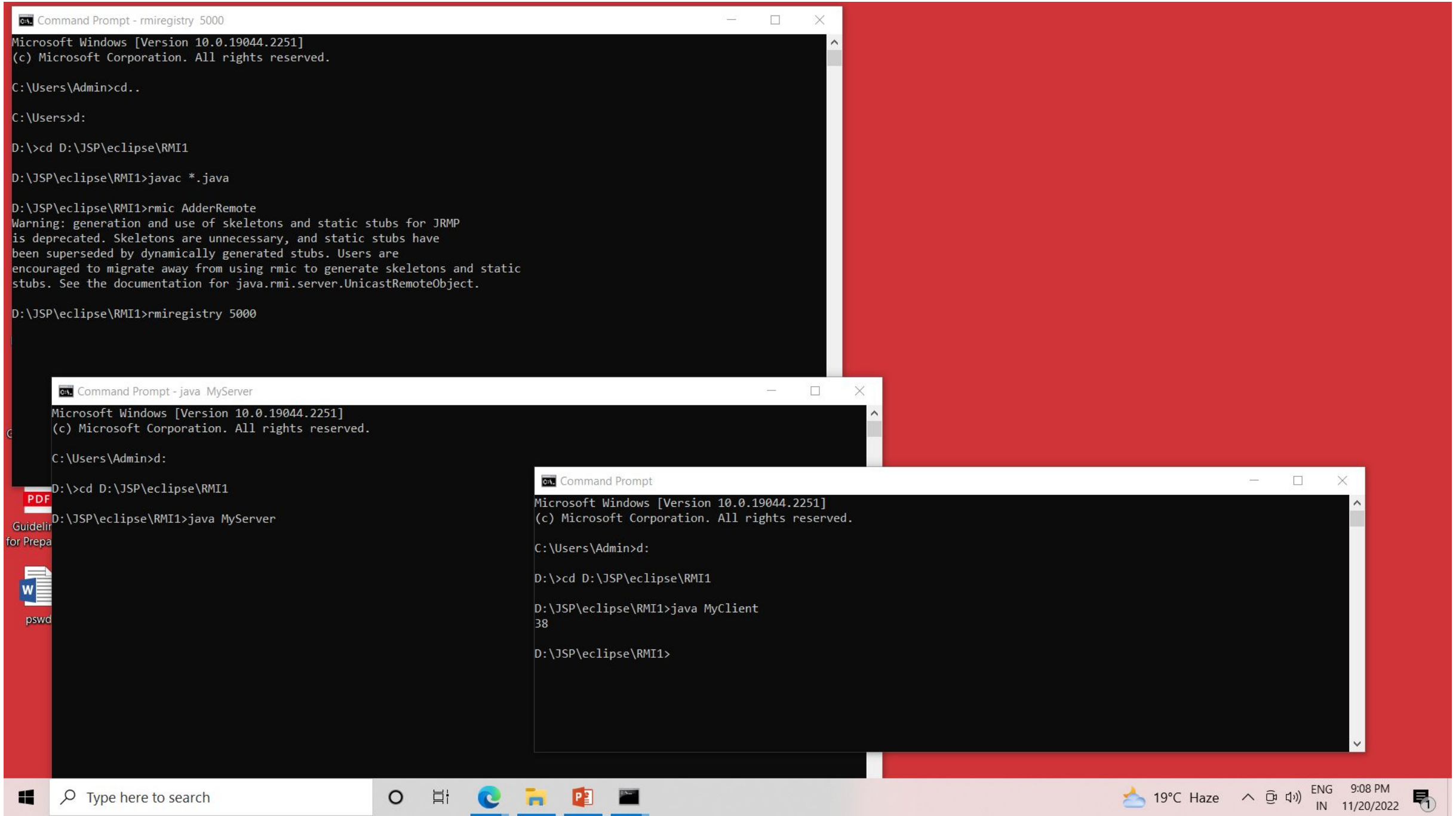
    rmiregistry 5000

4)start the server in another command prompt

    java MyServer

5)start the client application in another command prompt

    java MyClient


Command Prompt - rmiregistry 5000
Microsoft Windows [Version 10.0.19044.2251]
(c) Microsoft Corporation. All rights reserved.
C:\Users\Admin>cd..
C:\Users>d:
D:\>cd D:\JSP\eclipse\RMI1
D:\JSP\eclipse\RMI1>javac *.java
D:\JSP\eclipse\RMI1>rmic AdderRemote
Warning: generation and use of skeletons and static stubs for JRMP
is deprecated. Skeletons are unnecessary, and static stubs have
been superseded by dynamically generated stubs. Users are
encouraged to migrate away from using rmic to generate skeletons and static
stubs. See the documentation for java.rmi.server.UnicastRemoteObject.
D:\JSP\eclipse\RMI1>rmiregistry 5000
Command Prompt - java MyServer
Microsoft Windows [Version 10.0.19044.2251]
(c) Microsoft Corporation. All rights reserved.
C:\Users\Admin>d:
D:\>cd D:\JSP\eclipse\RMI1
D:\JSP\eclipse\RMI1>java MyServer
Command Prompt
Microsoft Windows [Version 10.0.19044.2251]
(c) Microsoft Corporation. All rights reserved.
C:\Users\Admin>d:
D:\>cd D:\JSP\eclipse\RMI1
D:\JSP\eclipse\RMI1>java MyClient
38
D:\JSP\eclipse\RMI1>
Guideli
for Prepa
pswd
Type here to search
19°C Haze
ENG
IN
9:08 PM
11/20/2022