PRACTICAL :- 2

AIM : Implement Remote Method Invocation in JAVA.

❖ RMI(REMOTE METHOD INVOCATION)

The RMI (Remote Method Invocation) is an API that provides a mechanism to create distributed application in java. The RMI allows an object to invoke methods on an object running in another JVM.

The RMI provides remote communication between the applications using two objects stub and skeleton.

Stub Object: The stub object on the client machine builds an information block and sends this information to the server.

The block consists of

1. An identifier of the remote object to be used
2. Method name which is to be invoked
3. Parameters to the remote JVM

Skeleton Object: The skeleton object passes the request from the stub object to the remote object. It performs the following tasks

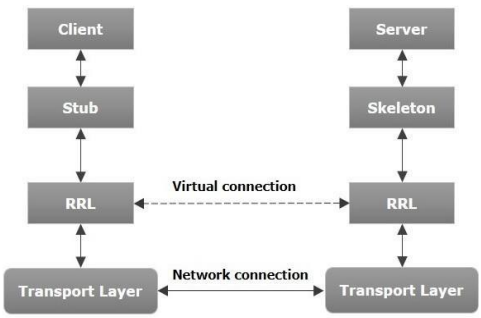
1. It calls the desired method on the real object present on the server.
2. It forwards the parameters received from the stub object to the method.

● 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.



Let us now discuss the components of this architecture.

- 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.

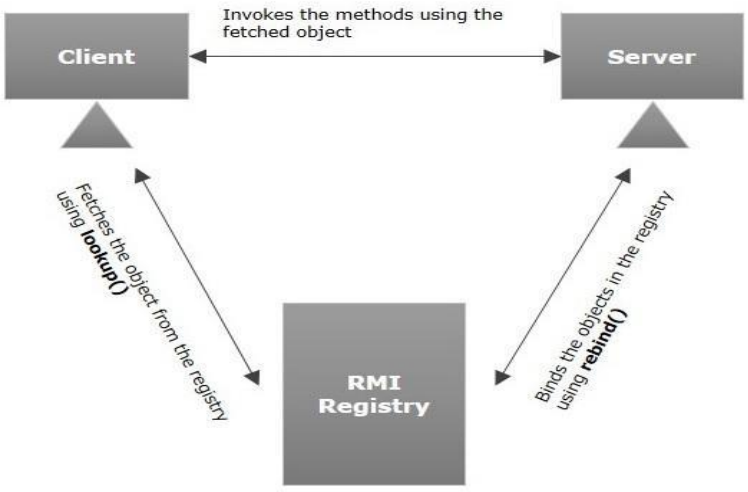
- **RMI Registry**

  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 −



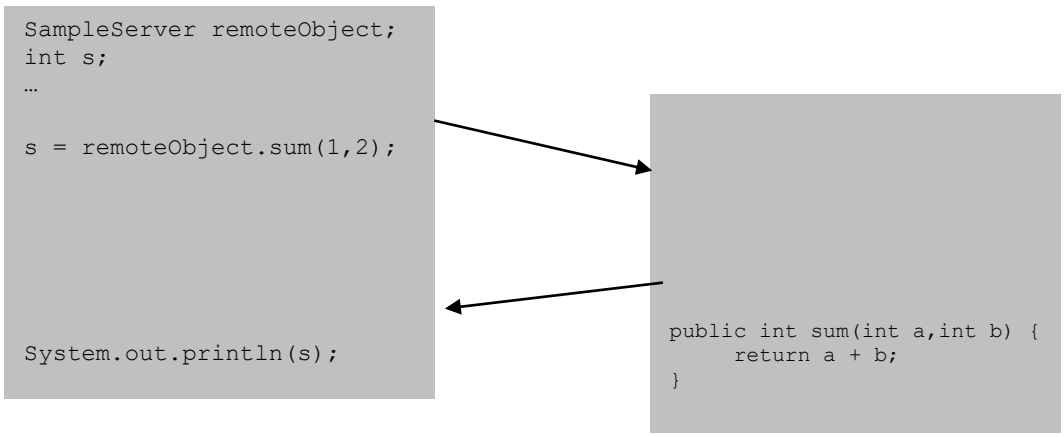
- **Goals of RMI**

  Following are the goals of RMI −

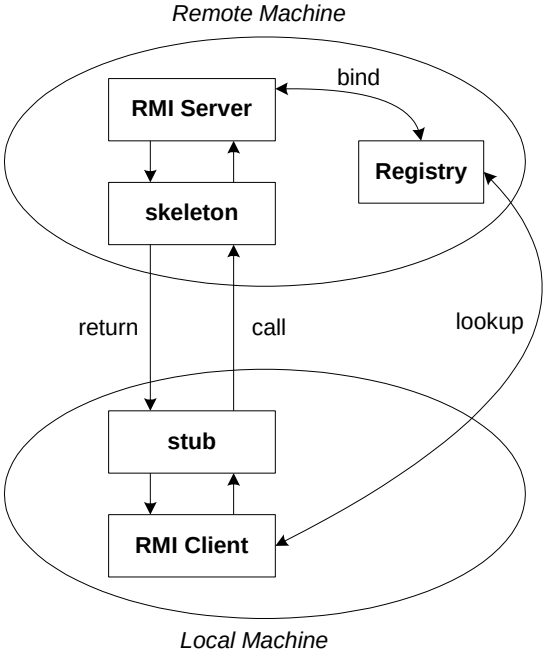
  - To minimize the complexity of the application.
  - To preserve type safety.
  - Distributed garbage collection.
  - Minimize the difference between working with local and remote objects.

Operation Java RMI

Java RMI allowed programmer to execute remote function class using the same semantics as local functions calls.

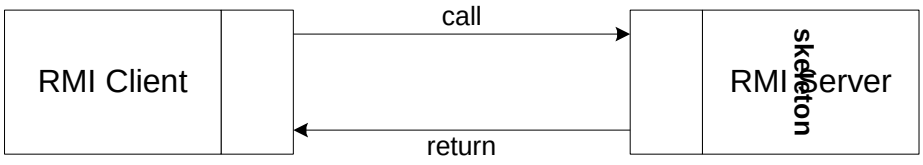


Here, general RMI architecture:



- The server must first bind its name to the registry
- The client lookup the server name in the registry to establish remote references.
- The Stub serializing the parameters to skeleton, the skeleton invoking the remote method and serializing the result back to the stub.

Stub and Skeleton



- A client invokes a remote method, the call is first forwarded to stub.
- The stub is responsible for sending the remote call over to the server-side skeleton
- The stub opening a socket to the remote server, marshaling the object parameters and forwarding the data stream to the skeleton.
- A skeleton contains a method that receives the remote calls, unmarshals the parameters, and invokes the actual remote object implementation.

Step of developing an RMI System:

1. Define the remote interface
2. Develop the remote object by implementing the remote interface.
3. Develop the client program.
4. Compile the Java source files.
5. Generate the client stubs and server skeletons.
6. Start the RMI registry.
7. Start the remote server objects.
8. Run the client

Step 1. Defining the Remote Interface

To create an RMI application, the first step is defining of a remote interface between the client and server objects.

```
/* SampleServer.java */
import java.rmi.*;

public interface SampleServer extends Remote
{
  public int sum(int a,int b) throws RemoteException;
}
```

Step 2. Develop the remote object by implement the remote interface

- The server is a simple unicast remote server.
- Create server by extending java.rmi.server.UnicastRemoteObject.
- The server uses the RMISecurityManager to protect its resources while engaging in remote communication.

```
/* SampleServerImpl.java */
import java.rmi.*;
import java.rmi.server.*;
import java.rmi.registry.*;

public class SampleServerImpl extends UnicastRemoteObject
                        implements SampleServer
{
  SampleServerImpl() throws RemoteException
  {
    super();
  }
```

- The server must bind its name to the registry, the client will look up the server name.
- Use java.rmi.Naming class to bind the server name to registry. In this example the name call “SAMPLE-SERVER”.
- In the main method of your server object, the RMI security manager is created and installed.

```
/* SampleServerImpl.java */
public static void main(String args[])
{
    try
    {
    //  System.setSecurityManager(new RMISecurityManager());
    //set the security manager

      //create a local instance of the object
      SampleServerImpl Server = new SampleServerImpl();

      //put the local instance in the registry
      Naming.rebind("SAMPLE-SERVER" , Server);

      System.out.println("Server waiting.....");
    }
    catch (java.net.MalformedURLException me)       {
      System.out.println("Malformed URL: " + me.toString());   }
    catch (RemoteException re)  {
       System.out.println("Remote exception: " + re.toString());  }
}
```

- Implement the remote mehtods

```
/* SampleServerImpl.java */
public int sum(int a,int b) throws RemoteException
```

```
 {
   return a + b;
 }
}
```

Step 3. Develop the client program

- In order for the client object to invoke methods on the server, it must first look up the name of server in the registry. You use the java.rmi.Naming class to lookup the server name.
- The server name is specified as URL in the from (rmi://host:port/name)
- Default RMI port is 1099.
- The name specified in the URL must exactly match the name that the server has bound to the registry. In this example, the name is “SAMPLE-SERVER”
- The remote method invocation is programmed using the remote interface name (remoteObject) as prefix and the remote method name (sum) as suffix.

```
import java.rmi.*;
import java.rmi.server.*;

public class SampleClient
{
  public static void main(String[]  args)
  {
    // set the security manager for the client
   // System.setSecurityManager(new RMISecurityManager());

    //get the remote object from the registry
    try
      {
        System.out.println("Security Manager loaded");
        String url = "//localhost/SAMPLE-SERVER";

        SampleServer remoteObject = (SampleServer)Naming.lookup(url);
        System.out.println("Got remote object");

        System.out.println(" 1 + 2 = " + remoteObject.sum(1,2) );
      }
      catch (RemoteException exc) {
        System.out.println("Error in lookup: " + exc.toString()); }
      catch (java.net.MalformedURLException exc) {
        System.out.println("Malformed URL: " + exc.toString());    }
      catch (java.rmi.NotBoundException exc)  {
        System.out.println("NotBound: " + exc.toString());
      }
```

```
  }
}
```

Step 4 & 5. Compile the Java source files & Generate the client stubs and server skeletons

- Assume the program compile and executing at elpis on ~/rmi
- Once the interface is completed, you need to generate stubs and skeleton code. The RMI system provides an RMI compiler (rmic) that takes your generated interface class and procedures stub code on its self.

elpis:~/rmi> set CLASSPATH=”~/rmi”

elpis:~/rmi> javac SampleServer.java

elpis:~/rmi> javac SampleServerImpl.java

elpis:~/rmi> rmic SampleServerImpl

elpis:~/rmi> javac SampleClient.java

6. Start the RMI registry

- The RMI applications need install to Registry. And the Registry must start manual by call rmiregisty.
- The rmiregistry us uses port 1099 by default. You can also bind rmiregistry to a different port by indicating the new port number as : rmiregistry <new port>

elpis:~/rmi> rmiregistry

remark: *On Windows, you have to type in from the command line:*

> start rmiregistry

Step 7 & 8. Start the remote server objects & Run the client

Once the Registry is started, the server can be started and will be able to store itself in the