ASSIGNMENT NO. 1

Problem Statement:

To implement multi-threaded client/server Process communication using RMI.

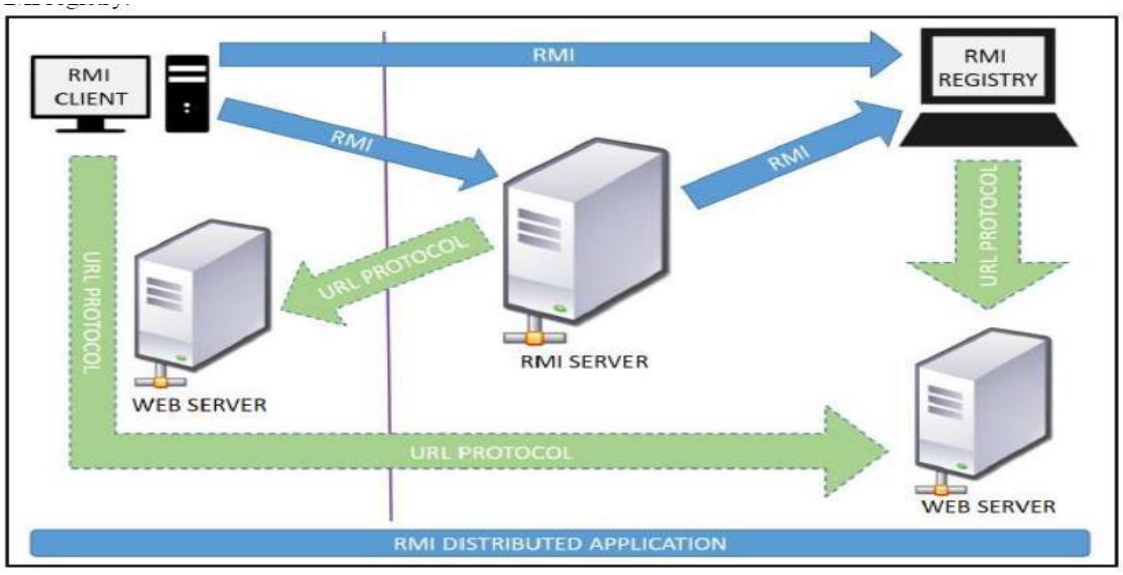
Tools / Environment:

Eclipse, Java 8, rmiregistry

Related Theory:

RMI provides communication between java applications that are deployed on different servers and connected remotely using objects called **stub** and **skeleton**. This communication architecture makes a distributed application seem like a group of objects communicating across a remote connection. These objects are encapsulated by exposing an interface, which helps access the private state and behavior of an object through its methods.

The following diagram shows how RMI happens between the RMI client and RMI server with the help of the RMI registry:



RMI REGISTRY is a remote object registry, a Bootstrap naming service, that is used by **RMI SERVER** on the same host to bind remote objects to names. Clients on local and remote hosts then look up the remote objects and make remote method invocations.

Key terminologies of RMI:

The following are some of the important terminologies used in a Remote Method Invocation.

Remote object: This is an object in a specific JVM whose methods are exposed so they could be invoked by another program deployed on a different JVM.

Remote interface: This is a Java interface that defines the methods that exist in a remote object. A remote object can implement more than one remote interface to adopt multiple remote interface behaviors.

RMI: This is a way of invoking a remote object's methods with the help of a remote interface. It can be carried with a syntax that is similar to the local method invocation.

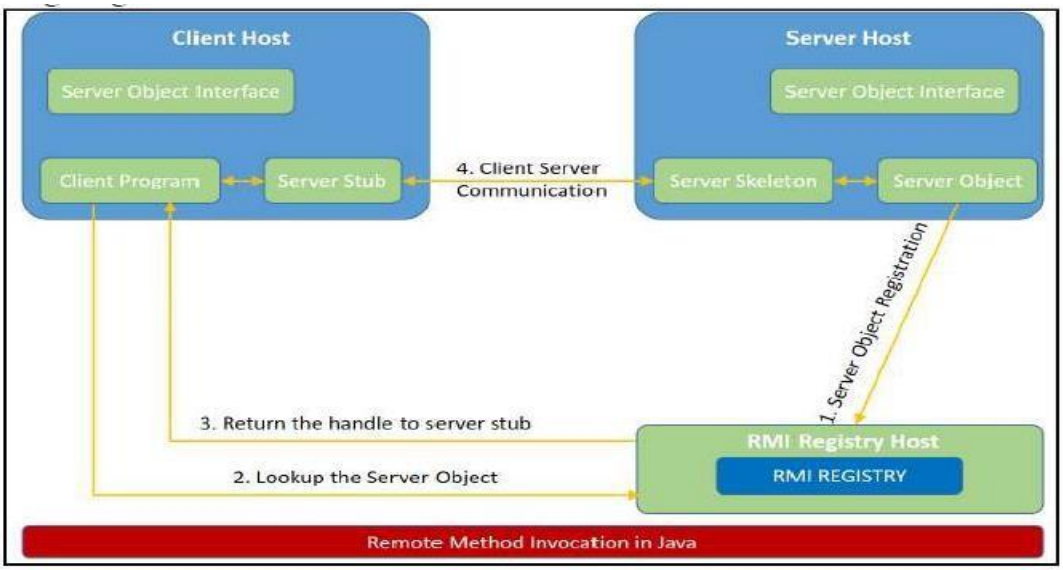
Stub: This is a Java object that acts as an entry point for the client object to route any outgoing requests. It exists on the client JVM and represents the handle to the remote object. If any object invokes a method on the stub object, the stub establishes RMI by following these steps:

1. It initiates a connection to the remote machine JVM.
2. It marshals (write and transmit) the parameters passed to it via the remote JVM.
3. It waits for a response from the remote object and unmarshals (read) the returned value or exception, then it responds to the caller with that value or exception.

Skeleton: This is an object that behaves like a gateway on the server side. It acts as a remote object with which the client objects interact through the stub. This means that any requests coming from the remote client are routed through it. If the skeleton receives a request, it establishes RMI through these steps:

1. It reads the parameter sent to the remote method.
2. It invokes the actual remote object method.
3. It marshals (writes and transmits) the result back to the caller (stub).

The following diagram demonstrates RMI communication with stub and skeleton involved:



Designing the solution:

The essential steps that need to be followed to develop a distributed application with RMI are as follows:

1. Design and implement a component that should not only be involved in the distributed application, but also the local components.
2. Ensure that the components that participate in the RMI calls are accessible across networks.
3. Establish a network connection between applications that need to interact using the RMI.

1. **Remote interface definition**: The purpose of defining a remote interface is to declare the methods that should be available for invocation by a remote client. Programming the interface instead of programming the component implementation is an essential design principle adopted by all modern Java frameworks, including Spring. In the same pattern, the definition of a remote interface takes importance in RMI design as well.

2. **Remote object implementation**: Java allows a class to implement more than one interface at a time. This helps remote objects implement one or more remote interfaces. The remote object class may have to implement other local interfaces and methods that it is responsible for. Avoid adding complexity to this scenario, in terms of how the arguments or return parameter values of such component methods should be written.

3. **Remote client implementation**: Client objects that interact with remote server objects can be written once the remote interfaces are carefully defined even after the remote objects are deployed.

Implementing the solution:

Steps to implement RMI:

1. Defining a remote interface
2. Implementing the remote interface
3. Creating Stub and Skeleton objects from the implementation class using `rmic (rmi compiler)`
4. Start the `rmiregistry`
5. Create and execute the server application program
6. Create and execute the client application program.

STEP 1: Defining the remote interface:

- To create an interface which will provide the description of the methods that can be invoked by remote clients.
- This interface should extend the Remote interface and the method prototype within the interface should throw the RemoteException.

```
// Creating a Search interface
(Search.java) import java.rmi.*; public
interface Search extends Remote
{

// Declaring the method prototype public String
query(String search) throws RemoteException;
}
```

STEP 2: Implementing the remote interface

- To implement the remote interface, the class should extend to `UnicastRemoteObject` class of `java.rmi` package.

```
// Java program to implement the Search interface
(SearchQuery.java) import java.rmi.*; import
java.rmi.server.*; public class SearchQuery
extends UnicastRemoteObject implements Search

{ // Implementation of the query interface

public String query(String search) throws RemoteException
{
     String result; if
     (search.equals("Reflection in Java"))
     result = "Found";
     else result = "Not
          Found";
     return result; } }
```

STEP 3: Creating Stub and Skeleton objects from the implementation class using `rmic`

The rmic tool is used to invoke the rmi compiler that creates the Stub and Skeleton objects. Its prototype is rmic classname. The command need to be executed at the command prompt

```
# rmic SearchQuery
```

STEP 4: Start the rmiregistry • Start the registry service by issuing the command at the command prompt :

```
# start rmiregistry
```

STEP 5: Create and execute the server application program

- To create the server application program and execute it on a separate command prompt.

• The server program uses `createRegistry` method of LocateRegistry class to create `rmiregistry` within the server JVM with the port number passed as argument.

• The rebind method of Naming class is used to bind the remote object to the new Name.

```
//program for server application
(SearchServer.java) import java.rmi.*; import
java.rmi.registry.*; public class SearchServer
{
public static void main(String args[])
{
tr
y
{// Create an object of the interface
// implementation class
Search obj = new SearchQuery();
// rmiregistry within the server JVM with port number 1900
LocateRegistry.createRegistry(1900);
// Binds the remote object by the name LP-V
Naming.rebind("rmi://localhost:1900"+"/LP-V",obj);
}
catch(Exception
ae)
{
System.out.println(ae);
} } }
```

Step 6: Create and execute the client application program

• The last step is to create the client application program and execute it on a separate command prompt .

• The lookup method of Naming class is used to get the reference of the Stub object.

```
//program for client application
(ClientRequest.java) import java.rmi.*; public
class ClientRequest
{  public static void main(String
args[])
{ String answer,value= "RMI in Java";
try
{ // lookup method to find reference of remote object
Search access =
```

```
(Search)Naming.lookup("rmi://localhost:1900/cl9");
answer = access.query(value);
System.out.println("Article on " + value +
" " + answer+" at cl9");
}
catch(Exception
ae)
{
System.out.println(ae);
} } }
```

Step 7: Compile and execute application programs:

```
#javac SearchQuery.java
#rmic SearchQuery
#rmiregistry on console
On console-1:
Compile Server Application:
#javac SearchServer.java
#java SearchServer
On console-2:
Compile ClientRequet Application:
#javac ClientRequest.java
#java ClientRequest
```

Conclusion:

In this assignment, we have studied how Remote Method Invocation (RMI) allows us to build Java applications that are distributed among several machines. Remote Method Invocation (RMI) allows a Java object that executes on one machine to invoke a method of a Java object that executes on another machine. This is an important feature, because it allows us to build distributed applications.

Submitted By: Shreya Mahajan

Roll No.: 4345

Class: B.E.I.T.

Staff Name: Ms. M. A. Rane/ Mr. K. V. Patil

Staff Signature:

Date: