Page :
Date : / /

Assignment No. - 1

Aim :- Implement multi-threaded client / server Process communication using RMI.

- **Objectives :-** To develop a multi-threaded client / server process communication using Java RMI.

- **Infrastructure :-**

Software Used :- Java, Eclipse, IDE, JDK

Theory :- Remote method invocation (RMI) allows a java object to invoke method on an object running on another machine. RMI provide remote communication betⁿ java program. RMI is used for building distributed application.

RMI applications often comprise two separate programs, a server & a client.

1) A typical server program creates some remote objects, makes references to these objects accessible, & waits for clients to invoke methods on these objects.
2) A typical client program obtains a remote reference to one or more remote objects on a server & then invokes methods on them.

RMI provides the mechanism by which the server & the client communicate & pass info back & forth. Such an application is sometimes referred to as a distributed object application.

Distributed object applications need to do the follo :-

- Locate remote objects.
- Communicate with remote objects.
- Load class definitions for objects that are passed around.

The RMI provides remote communication betn applications using two objects stub & skeleton. A object is an object whose method can be invoked another JVM.

- **Stub :-**

The stub is an object, acts as a gateway fo side. All the outgoing requests are routed thro' i resides at the client side & represents the remo object. When the caller invokes method on the object, it does the follo. tasks :-

1) It initiates a connection with remote Virtual Ma
2) It writes & transmits (marshals) the parameter remote Virtual Machine (JVM)
3) It waits for the result.
4) It reads (unmarshals) the return value or exceptio
5) It finally, returns the value to the caller.

- **Skeleton :-**

The Skeleton is an object, acts as a gateway the server side object. All the incoming requests routed thro' it, When the skeleton receives the in request, it does the follo. tasks :-

1) It reads the parameter for the remote method.
2) It invokes the method on the actual remote objec
3) It writes & transmits (marshals) the result to the

- **Remote Interfaces, Objects, and Methods :-**

Like any other Java application, a distributed built by using Java RMI is made up of interfac classes. The interfaces declare methods. The class

implement the methods declared in interfaces & perhaps, declare additional methods as well. In a distributed application, some implementations might reside in some Java Virtual Machines but not others. Objects with methods that can be invoked across Java Virtual machines are called remote objects.

An object becomes remote by implementing a remote interface, which has the follo. characteristics :-

- A remote interface extends the interface java.rmi.Remote
- Each method of the interface declares java.rmi.RemoteException in its throws clause, in addition to any application-specific exceptions.

RMI treats a remote object differently from a non-remote object when object is passed from one JVM to another JVM. Rather than making a copy of implementation object in receiving JVM, RMI passes a remote stub for a remote object. The stub acts as the local representative, or proxy, for the remote object & basically is, to the client, the remote reference. The client invokes a method on the local stub, which is responsible for carrying out the method invocation on the remote object.

A stub for a remote object implements the same set of remote interfaces that remote object implements. This property enables a stub to be cast to any of the interfaces that the remote object implements. However, only those methods defined in a remote interface are available to be called from receiving JVM.

Creating Distributed Applications by Using RMI:-

Using RMI to develop a distributed application involves these general steps:-

1) Designing and implementing the components distributed application.
2) Compiling sources.
3) Making classes network accessible.
4) Starting the application.

Conclusion :-

We implemented a multi-thread client / server process communication using RMI.

Practical No.1

Name: Pallavi Kamlakar Chopade

Roll No.:14

PRN NO :72036169K

Sub: Distributed System

Class: BE(IT)

Hello.java

```
import java.rmi.Remote;
import java.rmi.RemoteException;
public interface Hello extends Remote
{
default void message() throws RemoteException{}
}
```

InterfaceImplementation.java

```
public class ImplExample implements Hello{
    public void message() {
    System.out.println("This is at server");
    }
    }
```

Server.java

```
import java.rmi.registry.Registry;
import java.rmi.registry.LocateRegistry;
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;
public class Server extends ImplExample {
public Server() {}
public static void main(String args[]) {
try {
// Instantiating the implementation class
ImplExample obj = new ImplExample();
// Exporting the object of implementation class
// (here we are exporting the remote object to the stub)
```

```
Hello stub = (Hello) UnicastRemoteObject.exportObject(obj, 0);
// Binding the remote object (stub) in the registry
Registry registry = LocateRegistry.getRegistry();
registry.bind("Hello", stub);
System.err.println("Server ready");
} catch (Exception e) {
System.err.println("Server exception: " + e.toString());
e.printStackTrace();
}
}
}
```

Client.java

```
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
public class Client {
private Client() {}
public static void main(String[] args) {
try {
// Getting the registry
Registry registry = LocateRegistry.getRegistry(null);
// Looking up the registry for the remote object
Hello stub = (Hello) registry.lookup("Hello");
// Calling the remote method using the obtained object
stub.message();
// System.out.println("Remote method invoked");
} catch (Exception e) {
    System.err.println("Client exception: " + e.toString());
e.printStackTrace();
}
}
}
```

```
Command Prompt
Microsoft Windows [Version 10.0.22621.1194]
(c) Microsoft Corporation. All rights reserved.

C:\Users\pralh>cd C:\Users\pralh\eclipse-workspace\AssignmentNo1\src

C:\Users\pralh\eclipse-workspace\AssignmentNo1\src>start rmiregistry

C:\Users\pralh\eclipse-workspace\AssignmentNo1\src>
```

Here we have started the RMI

```
C:\Program Files\Java\jdk-19\
WARNING: A terminally deprecated method in java.lang.System has been called
WARNING: System::setSecurityManager has been called by sun.rmi.registry.RegistryImpl
WARNING: Please consider reporting this to the maintainers of sun.rmi.registry.RegistryImpl
WARNING: System::setSecurityManager will be removed in a future release
```

```
Command Prompt - java Ser

C:\DS Prac>cd C:\Users\pralh\eclipse-workspace\AssignmentNo1\src

C:\Users\pralh\eclipse-workspace\AssignmentNo1\src>javac *.java
module-info.java:8: warning: [module] module name component AssignmentNo1 should avoid terminal digits
module AssignmentNo1 {

1 warning

C:\Users\pralh\eclipse-workspace\AssignmentNo1\src>start rmiregistry

C:\Users\pralh\eclipse-workspace\AssignmentNo1\src>java Server
Server ready
```

Then we have started the server

```
Command Prompt
Microsoft Windows [Version 10.0.22621.1194]
(c) Microsoft Corporation. All rights reserved.

C:\Users\pralh>cd C:\Users\pralh\eclipse-workspace\AssignmentNo1\src

C:\Users\pralh\eclipse-workspace\AssignmentNo1\src>java Client

C:\Users\pralh\eclipse-workspace\AssignmentNo1\src>
```

```
Command Prompt - java Ser

C:\DS Prac>cd C:\Users\pralh\eclipse-workspace\AssignmentNo1\src

C:\Users\pralh\eclipse-workspace\AssignmentNo1\src>javac *.java
module-info.java:8: warning: [module] module name component AssignmentNo1 should avoid terminal digits
module AssignmentNo1 {

1 warning

C:\Users\pralh\eclipse-workspace\AssignmentNo1\src>start rmiregistry

C:\Users\pralh\eclipse-workspace\AssignmentNo1\src>java Server
Server ready
This is at server
```

Finally, the output of the implemented code.