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

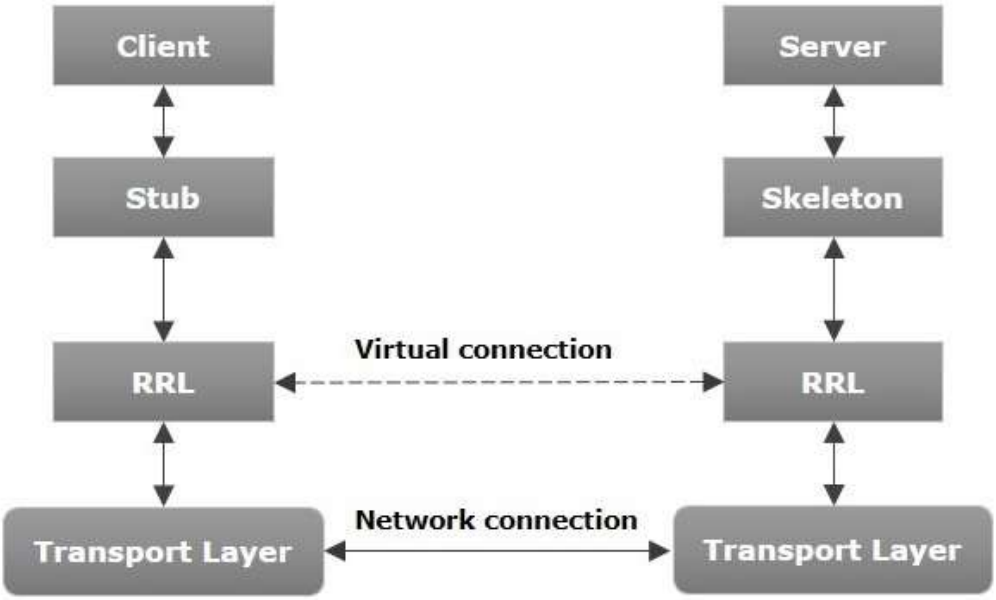
Theory:
Remote method invocation (RMI) allows a java object to invoke method on an object running on another machine. RMI provide remote communication between java program. RMI is used for building distributed application.

RMI applications often comprise two separate programs, a server and a client.
1. A typical server program creates some remote objects, makes references to these objects accessible, and 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 and then invokes methods on them.

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

Distributed object applications need to do the following:
- Locate remote objects.
- Communicate with remote objects.
- Load class definitions for objects that are passed around.

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.

How RMI provide remote Communication?

The RMI provides remote communication between the applications using two objects stub and skeleton. A remote object is an object whose method can be invoked from another JVM.

1. stub

The stub is an object, acts as a gateway for the client side. All the outgoing requests are routed through it. It resides at the client side and represents the remote object. When the caller invokes method on the stub object, it does the following tasks:

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

2. Skeleton

The skeleton is an object, acts as a gateway for the server side object. All the incoming requests are routed through it. When the skeleton receives the incoming request, it does the following tasks:

1. It reads the parameter for the remote method
2. It invokes the method on the actual remote object, and
3. It writes and transmits (marshals) the result to the caller.

Remote Interfaces, Objects, and Methods

The distributed application built using Java RMI is made up of interfaces and classes. The interfaces declare methods. The classes implement the methods declared in the interfaces and, 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 following 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 the object is passed from one Java virtual machine to another Java virtual machine. Rather than making a copy of the implementation object in the receiving Java virtual machine, RMI passes a remote stub for a remote object. The stub acts as the local representative, or proxy, for the remote object and 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 the 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 the receiving Java virtual machine.

Marshalling and Unmarshalling

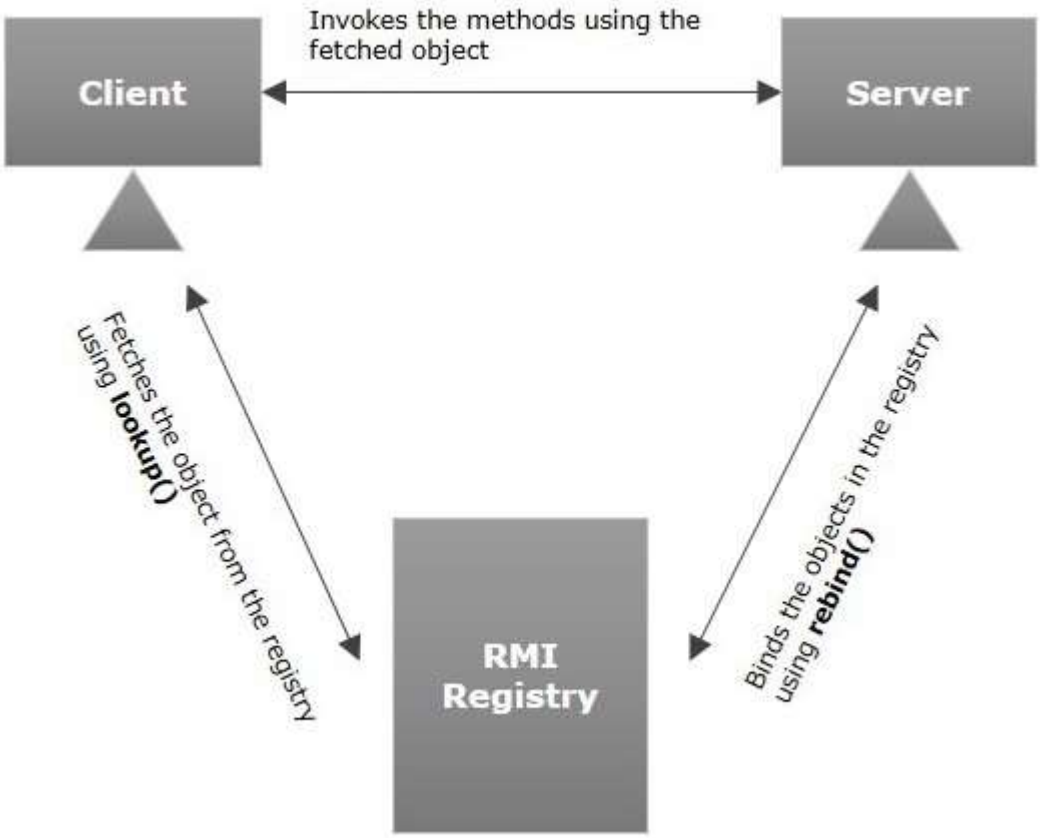
Whenever a client invokes a method that accepts parameters on a remote object, the parameters are bundled into a message before being sent over the network. These parameters may be of primitive type or objects. In case of primitive type, the parameters are put together and a header is attached to it. In case the parameters are objects, then they are serialized. This process is known as **marshalling**.

At the server side, the packed parameters are unbundled and then the required method is invoked. This process is known as **unmarshalling**.

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).



Creating Distributed Applications by Using RMI
Using RMI to develop a distributed application involves these general steps:
1. Designing and implementing the components of your distributed application.
2. Compiling sources.
3. Making classes network accessible.
4. Starting the application.

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.

Conclusion:

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