Java RMI (Remote Method Invocation)

The Java Remote Method Invocation (RMI) application programming interface (API) enables client and server communications over the Internet. Typically, client programs send requests to a server program, and the server program responds to those requests. In reality the client program invoke (activate) a method (function) implemented at remote server that is why it is called Remote Method Invocation. A related term is RPC (Remote Procedure Call) which is same as RMI, but the difference lies in the fact that RPC supports structure oriented programming paradigm while RMI supports Object Oriented Programming paradigm.

A common example is sharing a word processing program over a network. The word processor is installed on a server, and anyone who wants to use it starts it from his or her machine by double clicking an icon on the desktop or typing at the command line. The invocation sends a request to a server program for access to the software, and the server program responds by making the software available to the requestor.

The RMI API lets you create a publicly accessible remote server object that enables client and server communications through simple method calls on the server object. Clients can easily communicate directly with the server object and indirectly with each other through the server object using Uniform Resource Locators (URLs) and HyperText Transfer Protocol (HTTP).

To write a java RMI application, one must follow the below steps.

- Define The remote Interface.
- Implement the server.
- Implement the client.
- Compile the source code.
- Start the Java RMI registry, server and client.

In this tutorial, the client is sending a text to the remote server, by activating a function defined on the server to convert the text into upper case.

Defining the communication interface:

```
package Tutorial3;

import java.rmi.Remote;
import java.rmi.RemoteException;

public interface CommunicationInterface extends Remote {
    // Remote method declaration
    public String toUpperCase(String s) throws RemoteException;
}
```

A **remote object** is an instance of a class that implements this remote interface.

```
package Tutorial3;

import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;
```

```
public class RemoteObject extends UnicastRemoteObject
        implements CommunicationInterface {
    public RemoteObject() throws RemoteException {
        super();
    }

    // Implementation of the remote method toUpperCase
    public String toUpperCase(String str) throws RemoteException {

        return str.toUpperCase();
    }
}
```

We use the UnicastRemoteObject to create our stub implementation. The stub is what does the magic of communicating with the server over the underlying RMI protocol. The RemoteObject class is implementing the CommunicationInterface.

Implementing Server:

A "server" class, in this context, is the class which has a main method that creates an instance of the remote object implementation, exports the remote object, and then binds that instance to a name in a Java RMI registry. The class that contains this main method could be the implementation class itself, or another class entirely.

```
package Tutorial3;

import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;

public class RemoteServer {
    public static void main(String[] args) throws Exception {
        // Create and export the remote object
        CommunicationInterface remoteObject = new RemoteObject();

        // Create a RMI registry
        Registry registry = LocateRegistry.createRegistry(1090);

        //Bind the remoteObject to the registry
        registry.bind("RemoteObject", remoteObject);

        System.out.println("Server is listening");
    }
}
```

We create a remoteObject using

```
CommunicationInterface remoteObject = new RemoteObject();
```

We bind the remote object's stub in the registry: The following two lines of code create a registry to which stubs can be bound by servers and discovered by clients.

```
Registry registry = LocateRegistry.createRegistry(1090);
registry.bind("RemoteObject", remoteObject);
```

Implementing Client:

The client program obtains a stub for the registry on the server's host, looks up the remote object's stub by name in the registry, and then invokes the *toUpperCase* method on the remote object using the stub.

```
package Tutorial3;

import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;

public class RemoteObject extends UnicastRemoteObject
        implements CommunicationInterface {
    public RemoteObject() throws RemoteException {
        super();
    }

    // Implementation of the remote method toUpperCase
    public String toUpperCase(String str) throws RemoteException {

        return str.toUpperCase();
    }
}
```

Starting the Java RMI registry, server and client:

The following steps should be performed in the given order.

- Start the java RMI registry: On windows use the following command to run RMI registry. The detail for other OS is given in the link below.
  start rmiregistry

In Windows:

1. Open the Command Prompt
2. Navigate to the directory where your Java classes are located using the cd command.
3. Type the following command to start the RMI registry : start rmiregistry

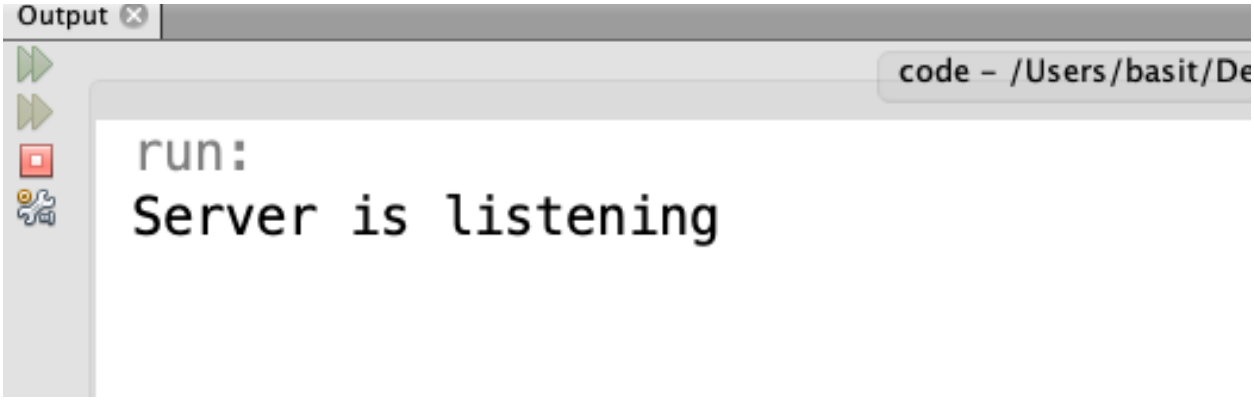
- Start the server: run the server program from your IDE.
- Start the client: run the client program from your IDE

For more detail follow the following link:

https://docs.oracle.com/javase/8/docs/technotes/guides/rmi/hello/hello-world.html

Output: After running the server and client, we have the following output.

Server side:



Client side: