Assignment No : 1

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

Objective:

Multi-threading.
How Remote Method Invocation Work.
How RMI allows objects to invoke methods on remote objects.
How to write Distributed Object Application.

Outcome:

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

Explanation:

Tools used :

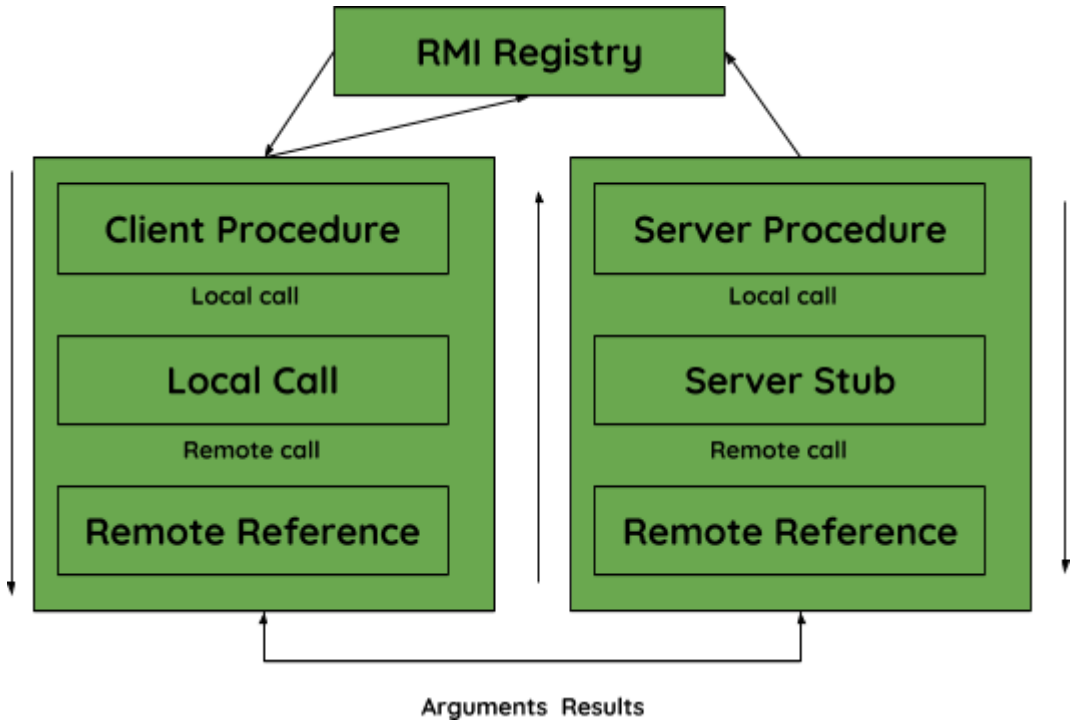
1. Windows
2. Linux
3. Java language

RMI (Remote Method Invocation) is used for distributed object references system. A distributed object is an object which publishes its interface on other machines. A Remote Object is a distributed object whose state is encapsulated. Stub and Skeleton are two objects used to communicate with the remote object.
Stub: Stub is a gateway for client program which is used to communicate with skeleton object, by establishing a connection between them.
Skeleton: Resides on Server program which is used for passing the request from stub to the remote interface.

Steps to Run Java RMI Application in Console

Creation of classes and interfaces for the problem statement: The steps involved in this are as follows:
Create a Remote Interface which extends java.rmi.Remote:
A remote interface determines the object that can be invoked remotely by the client. This interface can be communicated with the client's program. This Interface must extend **java.rmi.Remote** Interface.
Problem Statement: Create an RMI Application for finding the factorial of a numbe

How communication and process takes place in RMI:



Interface program

```
import java.math.BigInteger;

// Creating an Interface
public interface Factorial
        extends java.rmi.Remote {

        // Declaring the method
        public BigInteger fact(int num)
                throws java.rmi.RemoteException;
}
```

Create a class which extends java.rmi.server.UnicastRemoteObject and implements the previous interface.

This class will implement the remote interface. Do the required calculation for the problem statement.

Implementation of Interface

```
import java.math.BigInteger;

// Extends and Implement the class
// and interface respectively
```

```
public class FactorialImpl
        extends java.rmi.server.UnicastRemoteObject
        implements Factorial {

        // Constructor Declaration
        public FactorialImpl()
                throws java.rmi.RemoteException
        {
                super();
        }

        // Calculation for the problem statement
        // Implementing the method fact()
        // to find factorial of a number
        public BigInteger fact(int num)
                throws java.rmi.RemoteException
        {
                BigInteger factorial = BigInteger.ONE;

                for (int i = 1; i <= num; ++i) {
                        factorial = factorial
                                                        .multiply(
                                                                BigInteger
                                                                        .valueOf(i));
                }
                return factorial;
        }
}
```

Server Program

```
import java.rmi.Naming;

public class FactorialServer {

        // Implement the constructor of the class
        public FactorialServer()
        {
                try {
                        // Create a object reference for the interface
                        Factorial c = new FactorialImpl();

                        // Bind the localhost with the service
```

```
                Naming.rebind("rmi:// localhost/FactorialService", c);
            }
            catch (Exception e) {
                // If any error occur
                System.out.println("ERR: " + e);
            }
        }

        public static void main(String[] args)
        {
            // Create an object
            new FactorialServer();
        }
}
```

Client Program

```
import java.net.MalformedURLException;
import java.rmi.Naming;
import java.rmi.NotBoundException;
import java.rmi.RemoteException;

public class FactorialClient {
        public static void main(String[] args)
        {

            try {
                // Create an remote object with the same name
                // Cast the lookup result to the interface
                Factorial c = (Factorial);
                Naming.lookup("rmi:// localhost/FactorialService");

                // Call the method for the results
                System.out.println(c.fact(30));
            }

            // If any error occur
            catch (MalformedURLException murle) {
                System.out.println("\nMalformedURLException: "
                                                + murle);
            }

            catch (RemoteException re) {
                System.out.println("\nRemoteException: "
```

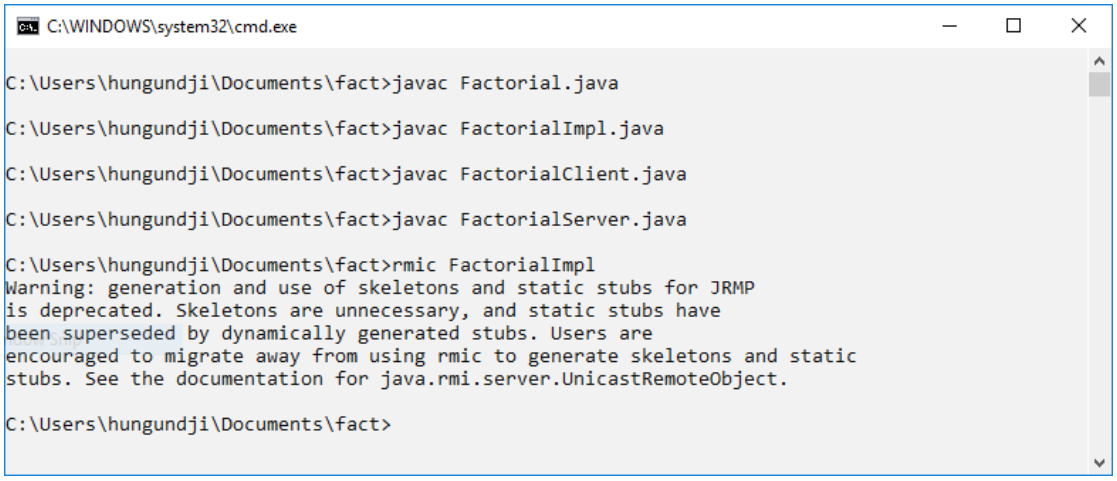
```
                                                    + re);
                }

                catch (NotBoundException nbe) {
                        System.out.println("\nNotBoundException: "
                                                    + nbe);
                }

                catch (java.lang.ArithmeticException ae) {
                        System.out.println("\nArithmeticException: " + ae);
                }
        }
}
```

Compilation of all program

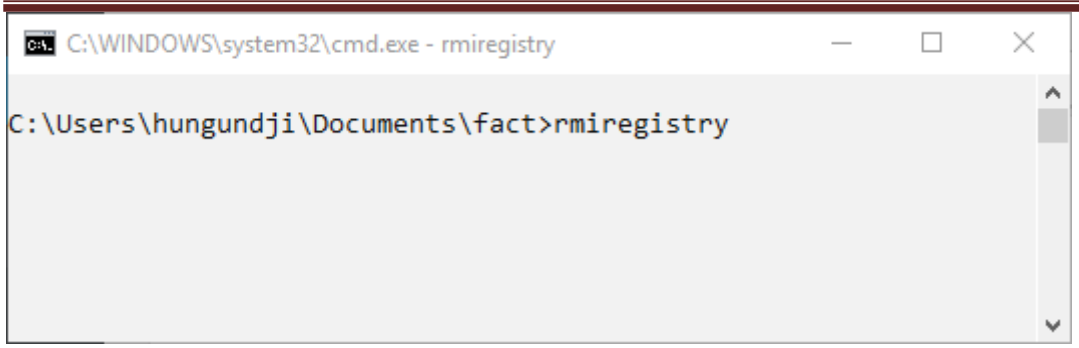
Use javac to compile all four programs and rmic (RMI Compiler) to create a stub and skeleton class files.



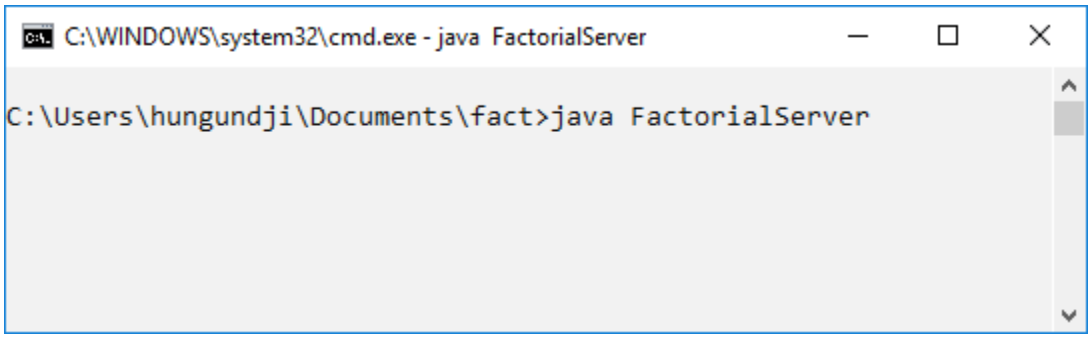
Running the system:

After the compilation phase, the system is now ready to run. To run the system, open three console screen (move to that path where the program resides). One for the client, one for server and one for the RMI Registry.

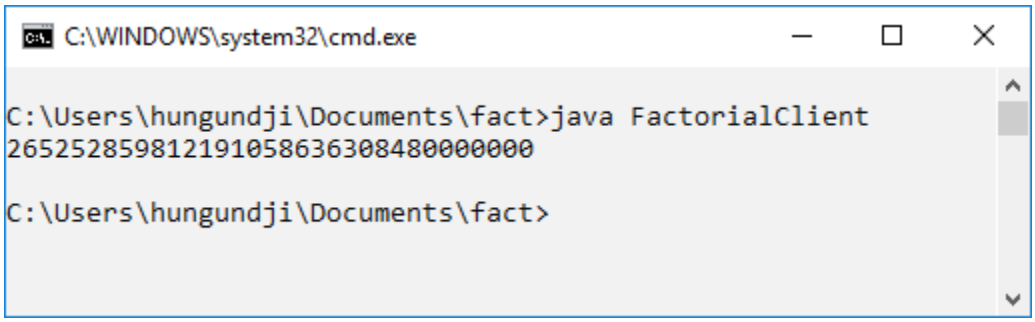
- Start with a registry, use **rmiregistry**, if there is no error registry will start running and now move to second screen.



In the second console run the server program and host the FactorialService. It will start and wait for the client connection and it will load the implementation into memory.



In the third console, run the client program.



Result:
Successfully the Client-Server communication is done

Date:	
Marks obtained:	
Sign of course coordinator:	
Name of course Coordinator :	