

**TRIBHUVAN UNIVERSITY**

**INSTITUTE OF ENGINEERING**

**THAPATHALI CAMPUS**

**A Lab Report**

**Of**

**Distributed System**

**On**

**Implementation of JAVA RMI Mechanism**

**Submitted By:**

**Santosh Pandey (THA076BCT041)**

**Submitted To:**

Department of Electronics and Computer Engineering

Thapathali Campus

Kathmandu, Nepal

27th June, 2023

# Theory:

RMI (Remote Method Invocation) is a Java API that allows communication between distributed Java programs. It enables Java objects in one JVM (Java Virtual Machine) to invoke methods on Java objects residing in a different JVM, potentially on a different physical machine. RMI provides a simple and intuitive way to implement distributed computing in Java.

The key components of JAVA RMI are:

**Remote Interface:** In RMI, the remote interface plays a crucial role in defining the methods that can be accessed remotely. It extends the *java.rmi.Remote* interface and declares all the methods that the client is allowed to invoke on the server-side.

**Remote Object:** The remote object represents the implementation of the remote interface. It extends the *java.rmi.server.UnicastRemoteObject* class to enable remote method invocation. Remote objects are responsible for registering with the RMI registry and can be accessed by clients for invoking remote methods.

**Stub and Skeleton:** The stub and skeleton are automatically generated by the RMI compiler (**rmic**) and handle the communication between the client and server. The stub resides on the client side and acts as a proxy for the remote object, while the skeleton resides on the server side and receives remote method invocations, dispatching them to the actual remote object.

**RMI Client:** The RMI client is a program that communicates with a remote server, invoking remote methods as if they were local. It utilizes a client stub as a local representation of the remote object. The client's responsibilities include locating the remote object, establishing a connection, and sending method invocation requests.

**RMI Server:** The RMI server implements the remote interface, operates remotely, and handles client requests. It enables remote invocation by clients and executes requested methods. The server object is registered with the RMI registry for client interaction.

# Implementation:

## Factorial.java

package Lab2.Factorial;

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;

}

## FactorialImpl.java

package Lab2.Factorial;

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();

    }

    // Implementation of the method fact()

    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;

    }

}

## FactorialServer.java

package Lab2.Factorial;

import java.rmi.Naming;

public class FactorialServer {

     // Constructor

    public FactorialServer()

    {

        try {

            // Create a object reference for the interface

            Factorial c = new FactorialImpl();

            // Bind the localhost with the service

            Naming.rebind("rmi://localhost:1099/FactorialService", c);

        }

        catch (Exception e) {

            // print the error

            System.out.println("ERR: " + e);

        }

    }

    public static void main(String[] args)

    {

        new FactorialServer();

    }

}

## FactorialClient.java

package Lab2.Factorial;

import java.net.MalformedURLException;

import java.rmi.Naming;

import java.rmi.NotBoundException;

import java.rmi.RemoteException;

import java.util.Scanner;

public class FactorialClient {

    public static void main(String[] args)

    {

        try {

            // Create an remote object with the same name

            //and cast the lookup result to the interface

            Factorial c = (Factorial)

            Naming.lookup("rmi://localhost:1099/FactorialService");

            // take input number from user to calculate factorial

            Scanner scanner = new Scanner(System.in);

            System.out.print("Enter number to calculate Factorial: ");

            int num = scanner.nextInt();

            scanner.close();

            // Call the method for the results

            System.out.println("Factorial of "+ num +"= "+c.fact(num));

        }

        // 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) {

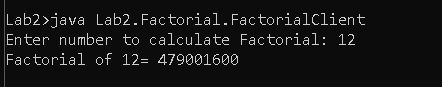
            System.out.println("\nArithmeticException: " + ae);

        }

    }

}

# Output:



# Conclusion:

In this Lab we learnt about RMI and implemented it on JAVA for calculation of factorial of a number.