DS & CC Lab Unit 3: Remote Method Invocation

Aim: 1: Program to retrieve time and date function from server to client. This program should display server date and time. Implement using RMI

Theory:
Remote Method Invocation (RMI) is a Java API that allows an object on one Java virtual machine (JVM) to invoke methods on an object located in another JVM, possibly on a different machine. It enables distributed applications by allowing clients to access services on remote servers as if they were local. In this program, we will use RMI to retrieve the server's current date and time. The client will make a remote method call to the server, which will respond by sending its current system date and time.

Code:
Server package
timeServerInterface.java

```
package server;
import java.rmi.Remote;
import java.rmi.RemoteException;
public interface timeServerInterface extends Remote {
public String getDateAndTime() throws RemoteException;
}
```

timeServerImplementation.java

```
package server;
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;
import java.text.SimpleDateFormat;
import java.util.Date;
public class timeServerImplementation extends UnicastRemoteObject implements
timeServerInterface {
protected timeServerImplementation() throws RemoteException{
        super();
}
@Override
public String getDateAndTime() throws RemoteException{
        SimpleDateFormat formatter = new SimpleDateFormat("dd/MM/yyyy HH:mm:ss");
        Date date = new Date();
        return formatter.format(date);
}
}
```

timeServer.java

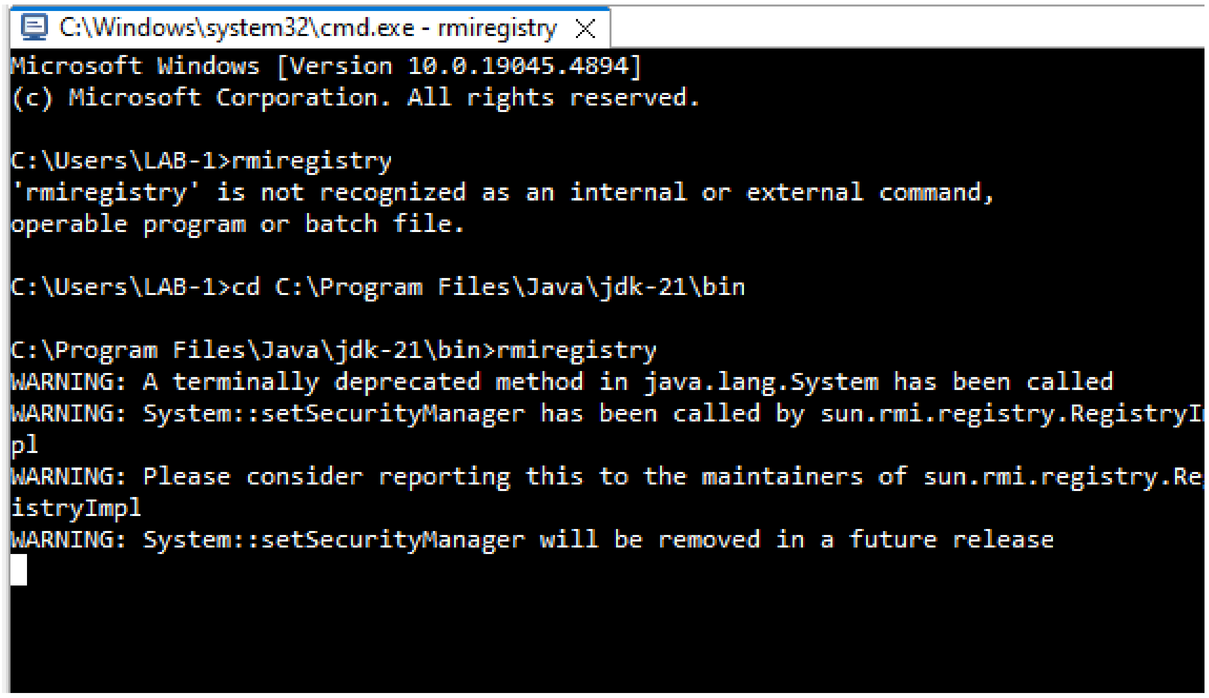
```
package server;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
public class timeServer {
	public static void main(String[] args) {
		try {
			timeServerImplementation obj = new timeServerImplementation();
			Registry registry = LocateRegistry.createRegistry(1234); //To open port
using Registry
			registry.bind("TimeServer", obj);
			System.out.println("Time Server is running.....");
		}catch(Exception e) {
			System.out.println("Server Exception: "+ e.getMessage());
			e.printStackTrace();
		}
	}
}
```

Client Package

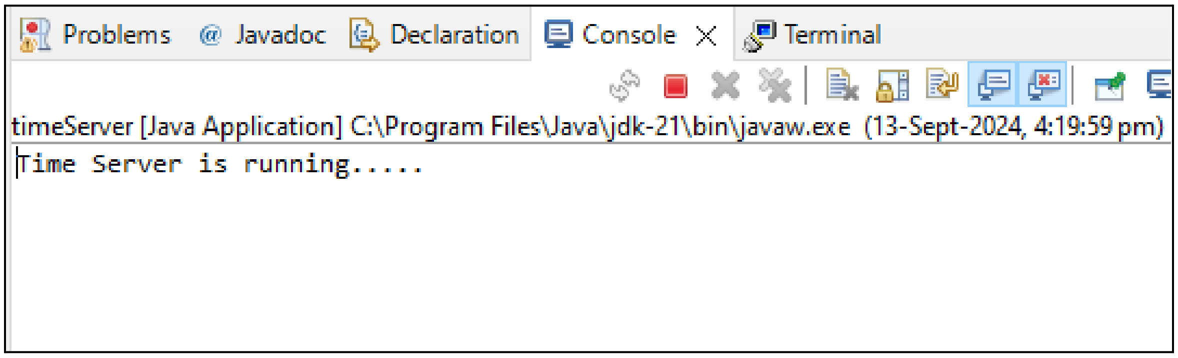
timeClient.java

```
package client;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import server.timeServerInterface;
public class timeClient {
	public static void main(String[] args) {
try {
	Registry registry = LocateRegistry.getRegistry("localhost",1234);
	timeServerInterface stub = (timeServerInterface)registry.lookup("TimeServer");
	String response = stub.getDateAndTime();
	System.out.println("Server's Date and Time"+response);
}catch(Exception e) {
	System.out.println("Client exception: "+e.getMessage());
	e.printStackTrace();
}
	}
}
```

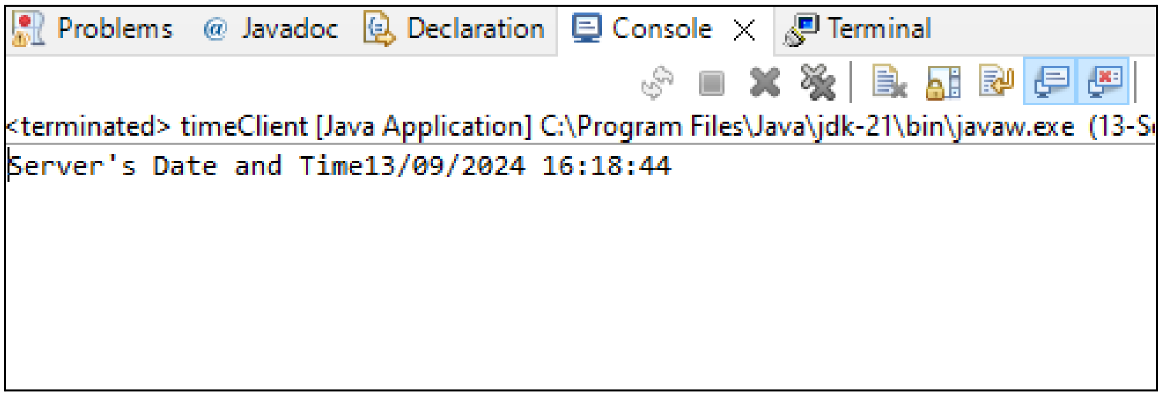
Output:



```
Microsoft Windows [Version 10.0.19045.4894]
(c) Microsoft Corporation. All rights reserved.

C:\Users\LAB-1>rmiregistry
'rmiregistry' is not recognized as an internal or external command,
operable program or batch file.

C:\Users\LAB-1>cd C:\Program Files\Java\jdk-21\bin

C:\Program Files\Java\jdk-21\bin>rmiregistry
WARNING: A terminally deprecated method in java.lang.System has been called
WARNING: System::setSecurityManager has been called by sun.rmi.registry.RegistryI
pl
WARNING: Please consider reporting this to the maintainers of sun.rmi.registry.Re
istryImpl
WARNING: System::setSecurityManager will be removed in a future release
```



```
timeServer [Java Application] C:\Program Files\Java\jdk-21\bin\javaw.exe  (13-Sept-2024, 4:19:59 pm)
Time Server is running.....
```



```
<terminated> timeClient [Java Application] C:\Program Files\Java\jdk-21\bin\javaw.exe  (13-S
Server's Date and Time13/09/2024 16:18:44
```

Aim: 2: Design a Graphical User Interface to find the greatest of two numbers. Implement using RMI.

Theory :
Remote Method Invocation (RMI) in Java allows an object on one machine to invoke methods on an object located on a different machine, facilitating distributed computing. In this program, the server hosts the logic to compare two numbers and return the greater number. The client inputs two numbers using a GUI and makes a remote call to the server to retrieve the result. The RMI registry acts as a mediator, binding the client and server together, enabling seamless communication. The GUI provides an interactive way for the client to submit numbers and display the results.

Program:

Greatest.java
```
package practical;
import java.rmi.Remote;
import java.rmi.RemoteException;
public interface Greatest extends Remote {
  public int findGreatest(int num1, int num2) throws RemoteException;
}
```

GreatestImpl.java
```
package practical;
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;
public class GreatestImpl extends UnicastRemoteObject implements Greatest {

  public GreatestImpl() throws RemoteException {
     super();
  }
  @Override
  public int findGreatest(int num1, int num2) throws RemoteException {
     return Math.max(num1, num2);
  }
}
```

Server.java

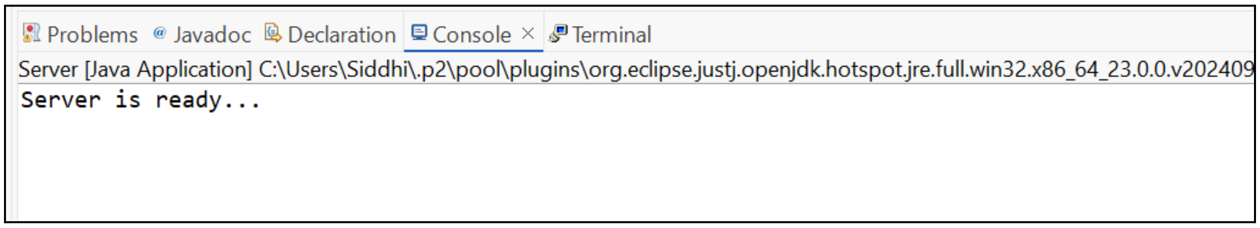
```
package practical;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
public class Server {
  public static void main(String[] args) {
      try {

          GreatestImpl obj = new GreatestImpl();

          Registry registry = LocateRegistry.createRegistry(1100);
          registry.bind("GreatestService", obj);
          System.out.println("Server is ready...");
      } catch (Exception e) {
          System.out.println("Server Exception: " + e.getMessage());
          e.printStackTrace();
      }
  }
}
```

Client.java

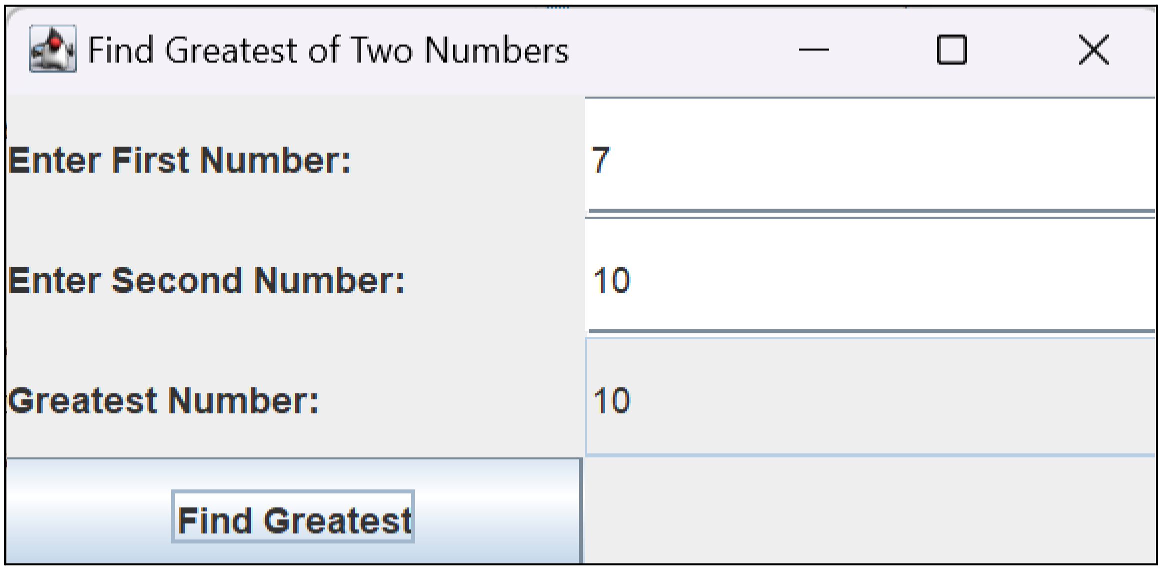
```
package practical;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import javax.swing.*;
public class Client extends JFrame {
  private JTextField num1Field, num2Field, resultField;
  private JButton findButton;
  public Client() {
      // Setting up the GUI
      setTitle("Find Greatest of Two Numbers");
      setSize(400, 200);
      setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      setLayout(new GridLayout(4, 2));
      // Number 1 input
      add(new JLabel("Enter First Number:"));
      num1Field = new JTextField();
      add(num1Field);
      // Number 2 input
```

```
        add(new JLabel("Enter Second Number:"));
        num2Field = new JTextField();
        add(num2Field);
        // Result display
        add(new JLabel("Greatest Number:"));
        resultField = new JTextField();
        resultField.setEditable(false);
        add(resultField);
        // Find Button
        findButton = new JButton("Find Greatest");
        add(findButton);
        findButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                try {

                    Registry registry = LocateRegistry.getRegistry("localhost", 1100);
                    Greatest stub = (Greatest) registry.lookup("GreatestService");
                    int num1 = Integer.parseInt(num1Field.getText());
                    int num2 = Integer.parseInt(num2Field.getText());

                    int greatest = stub.findGreatest(num1, num2);

                    resultField.setText(String.valueOf(greatest));
                } catch (Exception ex) {
                    ex.printStackTrace();
                    resultField.setText("Error occurred");
                }
            }
        });
    }
    public static void main(String[] args) {
        Client client = new Client();
        client.setVisible(true);
    }
}
```

Output:



```
Server [Java Application] C:\Users\Siddhi\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_23.0.0.v202409
Server is ready...
```



Enter First Number: 7
Enter Second Number: 10
Greatest Number: 10

Conclusion:
The client GUI will allow the user to input two numbers and display the greater number after pressing the "Find Greatest" button. The result will be fetched from the server via RMI.

Aim: 3: Program to implement the server which will solve equation
c=(a+b)^2 And c=(a+b)^3
Implement the same using RMI

Theory :
In RMI, a remote object runs on the server, and clients can interact with this object as if it were local. The RMI architecture includes the client, server, and RMI registry, which serves as a directory for remote objects.
In this program, the server will provide the functionality to solve two equations:

- c=(a+b)2c = (a + b)^2c=(a+b)2
- c=(a+b)3c = (a + b)^3c=(a+b)3

The client will send two numbers (a and b) to the server, and the server will compute the results for both equations and return them. The client will make remote calls to the server to retrieve these computed values.

Program:
EquationSolver.java

```
package practical;
import java.rmi.Remote;
import java.rmi.RemoteException;
public interface EquationSolver extends Remote {
  public int squareEquation(int a, int b) throws RemoteException;
  public int cubeEquation(int a, int b) throws RemoteException;
}
```

EquationSolverImpl.java

```
package practical;
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;
public class EquationSolverImpl extends UnicastRemoteObject implements EquationSolver {
  public EquationSolverImpl() throws RemoteException {
     super();
  }
  @Override
  public int squareEquation(int a, int b) throws RemoteException {
     return (int) Math.pow((a + b), 2);
  }
  @Override
  public int cubeEquation(int a, int b) throws RemoteException {
     return (int) Math.pow((a + b), 3);
  }
}
```

Server2.java

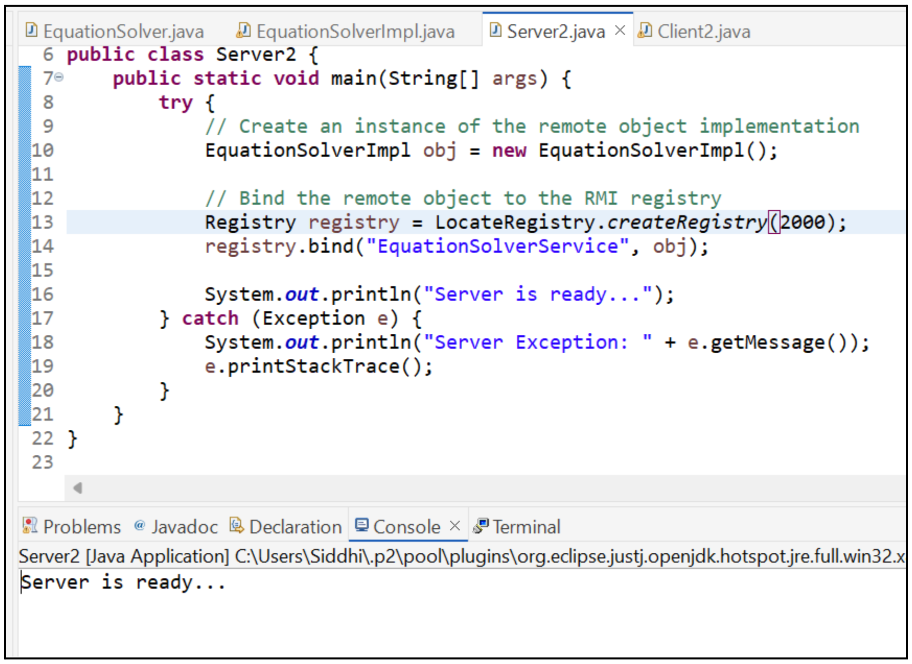
```
package practical;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
public class Server2 {
  public static void main(String[] args) {
     try {
        // Create an instance of the remote object implementation
        EquationSolverImpl obj = new EquationSolverImpl();

        // Bind the remote object to the RMI registry
        Registry registry = LocateRegistry.createRegistry(2000);
        registry.bind("EquationSolverService", obj);
        System.out.println("Server is ready...");
     } catch (Exception e) {
        System.out.println("Server Exception: " + e.getMessage());
        e.printStackTrace();
     }
  }
}
```

Client2.java

```
package practical;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.util.Scanner;
public class Client2 {
  public static void main(String[] args) {
     try {
        // Connect to the RMI registry running on localhost at port 1099
        Registry registry = LocateRegistry.getRegistry("localhost", 2000);

        // Lookup the remote object in the registry
        EquationSolver stub = (EquationSolver) registry.lookup("EquationSolverService");
        // Take input from user
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter value for a: ");
        int a = scanner.nextInt();
        System.out.print("Enter value for b: ");
        int b = scanner.nextInt();
        // Call the remote methods
        int resultSquare = stub.squareEquation(a, b);
        int resultCube = stub.cubeEquation(a, b);
        // Display results
```

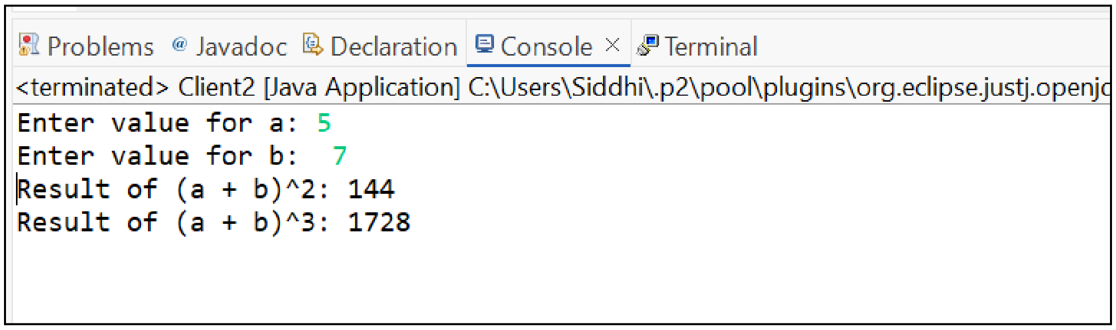
```
            System.out.println("Result of (a + b)^2: " + resultSquare);
            System.out.println("Result of (a + b)^3: " + resultCube);
        } catch (Exception e) {
            System.out.println("Client Exception: " + e.getMessage());
            e.printStackTrace();
        }
    }
}
```

Output:



```
public class Server2 {
    public static void main(String[] args) {
        try {
            // Create an instance of the remote object implementation
            EquationSolverImpl obj = new EquationSolverImpl();

            // Bind the remote object to the RMI registry
            Registry registry = LocateRegistry.createRegistry(2000);
            registry.bind("EquationSolverService", obj);

            System.out.println("Server is ready...");
        } catch (Exception e) {
            System.out.println("Server Exception: " + e.getMessage());
            e.printStackTrace();
        }
    }
}
```

```
Server2 [Java Application] C:\Users\Siddhi\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x
Server is ready...
```



```
<terminated> Client2 [Java Application] C:\Users\Siddhi\.p2\pool\plugins\org.eclipse.justj.openjc
Enter value for a: 5
Enter value for b:  7
Result of (a + b)^2: 144
Result of (a + b)^3: 1728
```

Conclusion: In this program, we successfully implemented a distributed application using Java RMI to compute two mathematical equations c=(a+b)2c = (a + b)^2c=(a+b)2 and c=(a+b)3c = (a + b)^3c=(a+b)3. By utilizing RMI, the server hosted the logic to perform the calculations, while the client accessed this functionality remotely.