VIT Vidyalankar Institute of Technology Accredited A+ by NAAC	DEPARTMENT OF COMPUTER ENGINEERING

Experiment No. 03

Semester	B.E. Semester VIII – Computer Engineering
Subject	Distributed Computing Lab
Subject Professor In-charge	Dr. Umesh Kulkarni
Assisting Professor	Prof. Prakash Parmar
Academic Year	2024-25

Student Name	Deep Salunkhe
Roll Number	21102A0014

Title: Design a Distributed Application for remote computation

---

Explanation:

1. Introduction

In a distributed computing environment, multiple computers work together to achieve a common goal by sharing resources and computational tasks over a network. **Remote Method Invocation (RMI)** in Java enables a Java program to invoke methods on a remote object located on another machine, facilitating distributed applications.

This lab focuses on implementing a **distributed remote computation system** using **Java RMI**, where a client requests computations from a remote server.

---

2. Objectives

- To understand the **concept of Remote Procedure Call (RPC) / Remote Method Invocation (RMI)**.

- To implement **a remote computation service** that allows clients to perform computations remotely.
- To explore **how Java RMI handles object serialization, remote interfaces, and distributed computing**.

---

3. Remote Method Invocation (RMI)

3.1 What is Java RMI?

Java RMI is a **Java API that enables communication between Java objects in different JVMs (Java Virtual Machines)**, allowing method calls across network boundaries as if they were local method calls.

3.2 Components of Java RMI

Java RMI consists of the following components:

1. **Remote Interface**
    - Defines the methods that the client can invoke remotely.
    - Implemented by the server.
2. **Remote Object (Implementation Class)**
    - Implements the remote interface and contains the business logic.
    - Must extend UnicastRemoteObject to be accessible remotely.
3. **RMI Registry**
    - A simple lookup service where remote objects are registered.
    - Clients query the registry to obtain a reference to a remote object.
4. **Client**
    - Connects to the RMI registry and invokes remote methods.
5. **Stub and Skeleton**
    - **Stub:** Client-side proxy for the remote object.
    - **Skeleton:** Server-side proxy that interacts with the stub (used in Java

versions prior to Java 5).

---

4. Working of Java RMI

The Java RMI architecture follows these steps:

1. **Server creates a remote object** and binds it to the RMI registry.
2. **Client looks up the remote object** from the registry.
3. **Client calls a method on the stub** (local proxy for the remote object).
4. **Stub forwards the request** to the remote object via RMI runtime.
5. **Remote object executes the method** and returns the result.

---

5. Implementation Steps

5.1 Define a Remote Interface

The remote interface extends java.rmi.Remote and declares methods that can be invoked remotely.

5.2 Implement the Remote Object

The implementation class extends UnicastRemoteObject and implements the remote interface.

5.3 Start the RMI Registry

The **RMI Registry (rmiregistry)** acts as a lookup service where the remote object is registered.

5.4 Bind the Remote Object

The remote object is **bound to a name** in the RMI registry, making it accessible to clients.

5.5 Implement the Client

The client looks up the remote object and invokes the desired remote methods

CODE:

```
PS E:\GIt\Sem-8\DC\Lab3> cat .\Compute.java
import java.rmi.Remote;
import java.rmi.RemoteException;

// Remote interface
public interface Compute extends Remote {
    int factorial(int number) throws RemoteException;
}
```

```
PS E:\GIt\Sem-8\DC\Lab3> cat .\ComputeImpl.java
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;

// Implementation of Compute interface
public class ComputeImpl extends UnicastRemoteObject implements Compute {

    // Constructor
    protected ComputeImpl() throws RemoteException {
        super();
    }

    // Factorial calculation
    @Override
    public int factorial(int number) throws RemoteException {
        if (number == 0 || number == 1) return 1;
        return number * factorial(number - 1);
    }
}
```

```
PS E:\GIt\Sem-8\DC\Lab3> cat .\Server.java
import java.rmi.Naming;
import java.rmi.registry.LocateRegistry;

public class Server {
    public static void main(String[] args) {
        try {
            // Create a Compute object
            Compute compute = new ComputeImpl();

            // Start RMI registry on port 5000
            LocateRegistry.createRegistry(5000);

            // Bind the Compute object to a name
            Naming.rebind("rmi://localhost:5000/ComputeService", compute);

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

```
PS E:\GIt\Sem-8\DC\Lab3> cat .\Client.java
import java.rmi.Naming;

public class Client {
    public static void main(String[] args) {
        try {
            // Look up the Compute service in the RMI registry
            Compute compute = (Compute) Naming.lookup("rmi://localhost:5000/ComputeService");

            // Input number for factorial
            int number = 5; // Example input
            int result = compute.factorial(number);

            // Print the result
            System.out.println("Factorial of " + number + " is: " + result);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Output:

```
PS E:\GIt\Sem-8\DC\Lab3> javac *.java
PS E:\GIt\Sem-8\DC\Lab3> rmiregistry
WARNING: A terminally deprecated method in java.lang.System has
been called
WARNING: System::setSecurityManager has been called by sun.rmi.r
egistry.RegistryImpl
WARNING: Please consider reporting this to the maintainers of su
n.rmi.registry.RegistryImpl
WARNING: System::setSecurityManager will be removed in a future
release
PS E:\GIt\Sem-8\DC\Lab3> start rmiregistry
PS E:\GIt\Sem-8\DC\Lab3> ^C
PS E:\GIt\Sem-8\DC\Lab3> javac *.java
PS E:\GIt\Sem-8\DC\Lab3> rmiregistry
WARNING: A terminally deprecated method in java.lang.System has
been called
WARNING: System::setSecurityManager has been called by sun.rmi.r
egistry.RegistryImpl
WARNING: Please consider reporting this to the maintainers of su
n.rmi.registry.RegistryImpl
WARNING: System::setSecurityManager will be removed in a future
release
```

```
            // Bind the Compute object to a name
            Naming.rebind("rmi://localhost/ComputeService", comp
ute);

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

PS E:\GIt\Sem-8\DC\Lab3> nvim .\Server.class
PS E:\GIt\Sem-8\DC\Lab3> nvim .\Server.java
PS E:\GIt\Sem-8\DC\Lab3>
PS E:\GIt\Sem-8\DC\Lab3> java Server
Server is ready.
```

```
        at java.base/java.util.concurrent.ThreadPoolExecutor.run
Worker(ThreadPoolExecutor.java:1144)
        at java.base/java.util.concurrent.ThreadPoolExecutor$Wor
ker.run(ThreadPoolExecutor.java:642)
        at java.base/java.lang.Thread.run(Thread.java:1583)
        at java.rmi/sun.rmi.transport.StreamRemoteCall.exception
ReceivedFromServer(StreamRemoteCall.java:304)
        at java.rmi/sun.rmi.transport.StreamRemoteCall.executeCa
ll(StreamRemoteCall.java:280)
        at java.rmi/sun.rmi.server.UnicastRef.invoke(UnicastRef.
java:382)
        at java.rmi/sun.rmi.registry.RegistryImpl_Stub.lookup(Re
gistryImpl_Stub.java:123)
        at java.rmi/java.rmi.Naming.lookup(Naming.java:101)
        at Client.main(Client.java:7)
PS E:\GIt\Sem-8\DC\Lab3> java Client
Factorial of 5 is: 120
PS E:\GIt\Sem-8\DC\Lab3>
```