योगः कर्मसु कौशलम्
IN PURSUIT OF PERFECTION

VIVEKANANDA INSTITUTE OF PROFESSIONAL STUDIES - TECHNICAL CAMPUS

Grade A++ Accredited Institution by NAAC

NBA Accredited for MCA Programme; Recognized under Section 2(f) by UGC;
Affiliated to GGSIP University, Delhi; Recognized by Bar Council of India and AICTE
An ISO 9001:2015 Certified Institution

SCHOOL OF ENGINEERING & TECHNOLOGY

B.Tech Programme: Computer Science & Engineering

Course Title: Distributed Systems and Cloud Computing Lab

Course Code: CIE-407P

Semester: 7th

Submitted By:

Name: Ishaan Jain

Enrollment No: 06117702722

Branch & Section: CSE-B

योगः कर्मसु कौशलम्
IN PURSUIT OF PERFECTION

An ISO 9001:2015 Certified Institution
SCHOOL OF ENGINEERING & TECHNOLOGY

VIVEKANANDA INSTITUTE OF PROFESSIONAL STUDIES - TECHNICAL CAMPUS
Grade A++ Accredited Institution by NAAC
NBA Accredited for MCA Programme; Recognized under Section 2(f) by UGC;
Affiliated to GGSIP University, Delhi; Recognized by Bar Council of India and AICTE

VISION OF INSTITUTE

To be an educational institute that empowers the field of engineering to build a sustainable future by providing quality education with innovative practices that supports people, planet and profit.

MISSION OF INSTITUTE

To groom the future engineers by providing value-based education and awakening students' curiosity, nurturing creativity and building capabilities to enable them to make significant contributions to the world.

VIPS
योगः कर्मसु कौशलम्
IN PURSUIT OF PERFECTION

VIVEKANANDA INSTITUTE OF PROFESSIONAL STUDIES - TECHNICAL CAMPUS

Grade A++ Accredited Institution by NAAC

NBA Accredited for MCA Programme; Recognized under Section 2(f) by UGC; Affiliated to GGSIP University, Delhi; Recognized by Bar Council of India and AICTE An ISO 9001:2015 Certified Institution

SCHOOL OF ENGINEERING & TECHNOLOGY

INDEX

S.no	EXPERIMENT	Date	Marks			Remark	Updated Marks	Faculty Signature
			Laboratory Assessment	Class Participation	Viva			

EXPERIMENT - 1

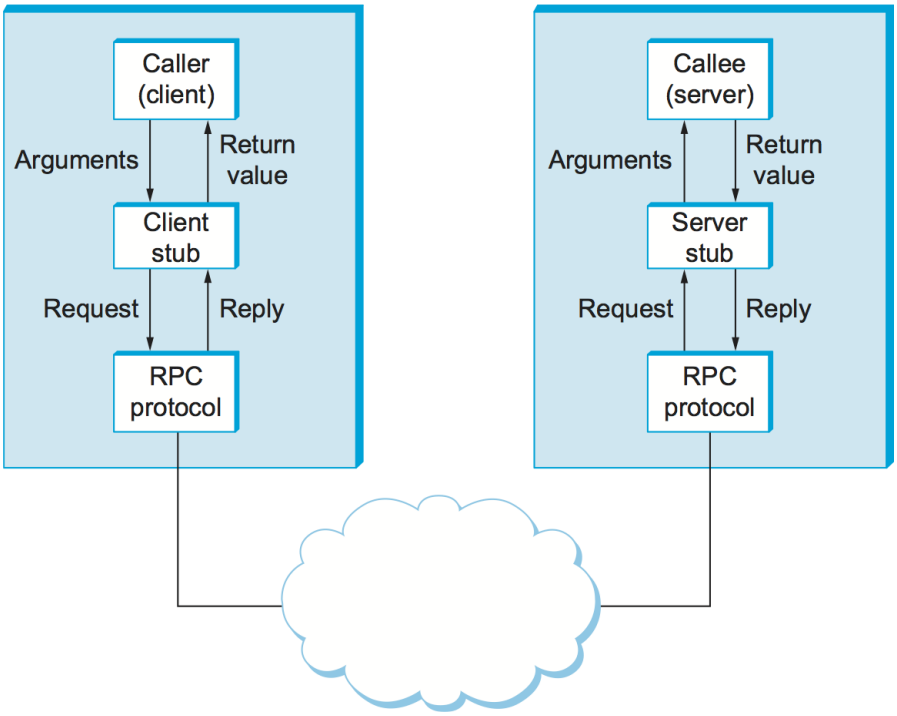
AIM: Write a Program in Java to implement RPC

THEORY:

A **Remote Procedure Call (RPC)** is a protocol that allows a program on one computer (the **client**) to execute procedures (methods/functions) on another computer (the **server**) as if they were local.
It hides the complexity of network communication → the programmer just calls a method, and the underlying system handles message passing, marshalling, and network transport.

Key Characteristics

- **Transparency**: Client calls remote methods just like local ones.
- **Language independent**: RPC can work across different programming languages (via IDL – Interface Definition Language).
- **Communication**: Uses request–response over a network (TCP/UDP).
- **Client-Server model**: The client initiates, the server executes.



Architecture of RPC
Advantages

- Simplifies distributed computing.
- Hides details of network programming (no sockets needed for the user).
- Increases modularity (clients and servers can evolve separately).

Disadvantages

- **Overhead**: More expensive than local calls due to network latency.
- **Partial failures**: If the server or network fails, the client might hang or crash.
- **Not transparent for non-functional aspects**: E.g., performance, failures, and security issues are different from local calls.

CODE:

Server.java

```
package rpcpassword;

import java.net.*;
import java.io.*;

public class server {
   public static void main(String[] args) {
      try (ServerSocket serverSocket = new ServerSocket(6000)) {
         System.out.println("\nServer listening on port 6000\n");
         while (true) {
            Socket clientSocket = serverSocket.accept();
            new Thread(() -> {
               try (BufferedReader in = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));
                   PrintWriter out = new PrintWriter(clientSocket.getOutputStream(), true)) {

                  String request = in.readLine();
                  System.out.println("Received password from client: " + request);

                  if ("password123".equals(request)) {
                     out.println("Access granted\n");
                     System.out.println("Access granted to client\n");
                  } else {
                     out.println("Access denied\n");
                     System.out.println("Access denied to client - incorrect password\n");
                  }

               } catch (IOException e) {
                  System.err.println("Client error: " + e.getMessage());
               }
            }).start();
         }
      } catch (IOException e) {
         System.err.println("Server error: " + e.getMessage());
      }
   }

}
```

Client.java

```
package rpcpassword;

import java.net.*;
import java.io.*;
import java.util.Scanner;

public class client {
   public static void main(String[] args) {
      try (Socket socket = new Socket("localhost", 6000)) {
         PrintWriter out = new PrintWriter(socket.getOutputStream(), true);
         BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
         Scanner scanner = new Scanner(System.in);

         System.out.print("\nEnter password: ");
         String password = scanner.nextLine();

         out.println(password);

         String response = in.readLine();
         System.out.println("Server response: " + response);

         scanner.close();
      } catch (IOException e) {
```

```
        System.err.println("Client error: " + e.getMessage());
    }
  }
}
```

OUTPUT:

Server

```
Server (1) [Java Application] C:\Users\Aditya Jindal\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_21.0.7.v20250502-0916\jre\bin\javaw.exe (16-Sept-2025, 10:09:17 pm elaps
Server listening on port 6000
Received password from client: password123
Access granted to client
```

Client

```
<terminated> Client [Java Application] C:\Users\Aditya Jindal\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_21.0.7.v20250502-0916\jre\bin\javaw.exe (16-Sept-2025, 10:09:3
Enter password: password123
Server response: Access granted
```

LEARNING OUTCOME:

EXPERIMENT - 2

AIM: Implement the concept of Remote Method Invocation in Java.

THEORY:
Remote Method Invocation (RMI) is Java's mechanism to implement Remote Procedure Calls (RPC) in an object-oriented way.
It allows a Java program running on one Java Virtual Machine (JVM) to invoke methods of an object running on another JVM, possibly on a different machine, as if the method were local.

Key Features
- Purely Java-based distributed computing.
- Provides location transparency: the client does not need to know where the object is running.
- Supports object serialization: allows passing complex objects (not just primitive data types) across the network.
- Built-in security manager and exception handling for remote operations.

RMI Components
1. Remote Interface
   - Declares methods that can be invoked remotely.
   - Must extend java.rmi.Remote.
   - Methods must throw RemoteException.
2. Remote Object Implementation
   - Class that provides the actual logic.
   - Must extend UnicastRemoteObject.
3. Stub (Client-side proxy)
   - Acts as a placeholder for the remote object.
   - Handles method call forwarding to the server.
4. Skeleton (Server-side proxy) *(in older Java versions)*
   - Received requests from stub and forwarded them to the actual remote object.
   - From Java 2 onwards, skeletons are generated automatically inside JVM.
5. RMI Registry
   - A naming service where remote objects are registered.
   - Clients use it to look up objects using a URL-like name

Advantages of RMI
- **Simplicity**: Hides low-level socket communication.
- **Object-oriented**: Works directly with Java objects.
- **Reusability**: Remote services can be reused across applications.
- **Built-in security**: Supports access restrictions via security manager.

Limitations
- Java-only: Cannot directly interact with non-Java systems (unlike gRPC or CORBA).
- Network overhead: Slower than local calls.
- Requires JVM on both client and server.

CODE:

Adder.java
```
package rmi;
import java.rmi.*;

public interface Adder extends Remote {
 int add(int a, int b) throws RemoteException;
}
```

AdderClient.java

```
package rmi;

import java.rmi.*;

public class AdderClient {
   public static void main(String[] args) {
      try {
         Adder stub = (Adder) Naming.lookup("rmi://localhost/AdderService");
         int result = stub.add(9, 7);
         System.out.println("Result: " + result);
      } catch (Exception e) {
         System.out.println("Client exception: " + e);
      }
   }
}
```

AdderImpl.java

```
package rmi;

import java.rmi.*;
import java.rmi.server.*;

public class AdderImpl extends UnicastRemoteObject implements Adder {
   AdderImpl() throws RemoteException {
      super();
   }

   public int add(int a, int b) throws RemoteException {
      return a + b;
   }

   public static void main(String[] args) {
      try {
         AdderImpl obj = new AdderImpl();
         Naming.rebind("rmi://localhost/AdderService", obj);
         System.out.println("Adder Service is running...");
      } catch (Exception e) {
         System.out.println("Server exception: " + e);
      }
   }
}
```

OUTPUT:

```
C:\Users\Aditya Jindal\eclipse-workspace\DSCC\bin>java rmi.AdderImpl
AdderService is bound and ready...
```

```
C:\Users\Aditya Jindal\eclipse-workspace\DSCC\bin>java rmi.AdderClient
Result: 30

C:\Users\Aditya Jindal\eclipse-workspace\DSCC\bin>
```

LEARNING OUTCOME: