EXPERIMENT NO. 3

Title:
To develop any distributed application through implementing client-server communication programs based on Java Sockets and RMI techniques
Objective:
To study how sockets and Remote Method Invocation (RMI) are used for communication between client and server, and to implement both techniques using Java.

⚙️ Part A – Java Socket Programming
Aim:
To implement a client-server communication system using Java Sockets to perform simple operations such as addition of digits, factorial, and string reversal.

🖥️ 1. Server Program (SocketServer.java)

```
// Filename: SocketServer.java
import java.io.*;
import java.net.*;
import java.util.*;

public class SocketServer {
   public static void main(String[] args) throws IOException {
      ServerSocket serverSocket = new ServerSocket(5000);
      System.out.println("Server started and waiting for client connection...");

      Socket socket = serverSocket.accept();
      System.out.println("Client connected!");

      BufferedReader in = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
      PrintWriter out = new PrintWriter(socket.getOutputStream(), true);

      String choice;
      while ((choice = in.readLine()) != null) {
         System.out.println("Client choice: " + choice);

         if (choice.equals("1")) {
            int num = Integer.parseInt(in.readLine());
            int sum = 0;
            while (num != 0) {
               sum += num % 10;
               num /= 10;
            }
            out.println("Sum of digits = " + sum);
         } else if (choice.equals("2")) {
            int n = Integer.parseInt(in.readLine());
            int fact = 1;
            for (int i = 1; i <= n; i++) fact *= i;
            out.println("Factorial = " + fact);
         } else if (choice.equals("3")) {
            String str = in.readLine();
```

```
            String rev = new StringBuilder(str).reverse().toString();
            out.println("Reversed String = " + rev);
        } else if (choice.equals("exit")) {
            out.println("Connection closed by client.");
            break;
        } else {
            out.println("Invalid choice!");
        }
    }

    socket.close();
    serverSocket.close();
    System.out.println("Server closed.");
  }
}
```

👨‍💻 2. Client Program (SocketClient.java)

```
// Filename: SocketClient.java
import java.io.*;
import java.net.*;
import java.util.*;

public class SocketClient {
  public static void main(String[] args) throws IOException {
    Socket socket = new Socket("localhost", 5000);
    System.out.println("Connected to Server.");

    BufferedReader in = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
    PrintWriter out = new PrintWriter(socket.getOutputStream(), true);
    Scanner sc = new Scanner(System.in);

    String choice;
    do {
      System.out.println("\n--- MENU ---");
      System.out.println("1. Sum of digits");
      System.out.println("2. Factorial");
      System.out.println("3. Reverse String");
      System.out.println("Type 'exit' to quit");
      System.out.print("Enter your choice: ");
      choice = sc.nextLine();

      out.println(choice);
      if (choice.equals("1")) {
        System.out.print("Enter number: ");
        out.println(sc.nextLine());
      } else if (choice.equals("2")) {
        System.out.print("Enter number: ");
        out.println(sc.nextLine());
      } else if (choice.equals("3")) {
```

```
            System.out.print("Enter string: ");
            out.println(sc.nextLine());
        }

        String response = in.readLine();
        System.out.println("Server Response: " + response);
    } while (!choice.equals("exit"));

    socket.close();
    System.out.println("Connection closed.");
  }
}
```

✅ How to Run (Sockets):

1. Open two terminals.
2. Compile both files:
3. javac SocketServer.java SocketClient.java
4. Run Server:
5. java SocketServer
6. Run Client (in another terminal):
7. java SocketClient

Sample Output:

```
--- MENU ---
1. Sum of digits
2. Factorial
3. Reverse String
Type 'exit' to quit
Enter your choice: 3
Enter string: OpenMP
Server Response: Reversed String = PMnepO
```

⚙️ Part B – Java RMI (Remote Method Invocation)

Aim:

To implement a distributed application using Java RMI where the client invokes a remote method to perform addition.

🧩 1. Interface (AddI.java)

```
// Filename: AddI.java
import java.rmi.*;
public interface AddI extends Remote {
   public int add(int a, int b) throws RemoteException;
}
```

🧩 2. Server Implementation (AddServer.java)

```
// Filename: AddServer.java
import java.rmi.*;
import java.rmi.server.*;

public class AddServer extends UnicastRemoteObject implements AddI {
```

```
    public AddServer() throws RemoteException { super(); }

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

🧩 3. Register the Server (RegisterMe.java)

```
// Filename: RegisterMe.java
import java.rmi.*;

public class RegisterMe {
    public static void main(String args[]) {
        try {
            AddServer obj = new AddServer();
            Naming.rebind("add", obj);
            System.out.println("Server registered successfully!");
        } catch (Exception e) {
            System.out.println(e);
        }
    }
}
```

🧩 4. Client Program (AddClient.java)

```
// Filename: AddClient.java
import java.rmi.*;

public class AddClient {
    public static void main(String args[]) {
        try {
            AddI obj = (AddI) Naming.lookup("rmi://localhost/add");
            int result = obj.add(15, 25);
            System.out.println("Addition Result = " + result);
        } catch (Exception e) {
            System.out.println(e);
        }
    }
}
```

✅ How to Run (RMI):

1. Compile all Java files:
2. javac *.java
3. Start RMI registry (in background):
4. start rmiregistry

(Keep it running in the same directory)

5. Run the Server:
6. java RegisterMe
7. Run the Client:
8. java AddClient

Sample Output:

Server registered successfully!
Addition Result = 40

🧠 Conclusion
In this experiment:

- We implemented Socket-based client-server communication for arithmetic and string operations.
- We also implemented Java RMI-based distributed computation, showing how remote methods are invoked transparently.
- These demonstrate both low-level communication (Sockets) and high-level object-based communication (RMI) in Java.