Distributed Calculator System

Using Java RMI

Executive Summary

This document presents the complete technical documentation of a distributed calculator system implemented using Java Remote Method Invocation (RMI). The system allows clients to perform basic arithmetic operations through remote calls to a centralized server.

Version 1.0
January 2, 2026

Contents

1 Introduction

1.1 Overview

The system implements a distributed client-server architecture that enables remote execution of basic mathematical operations. Using Java RMI (Remote Method Invocation) technology, the system allows multiple clients to connect simultaneously to the server to perform calculations.

1.2 System Objectives

- Provide a remote interface for basic arithmetic operations
- Demonstrate the use of distributed communication with Java RMI
- Implement a robust and scalable client-server system
- Ensure proper error and exception handling

1.3 Technologies Used

- **Language:** Java
- **Communication Framework:** Java RMI (Remote Method Invocation)
- **Protocol:** RMI over TCP/IP
- **Default Port:** 1099 (RMI Registry)

2 System Architecture

2.1 Client-Server Model

The system adopts the client-server architectural pattern, where:

Server Component

Responsible for:

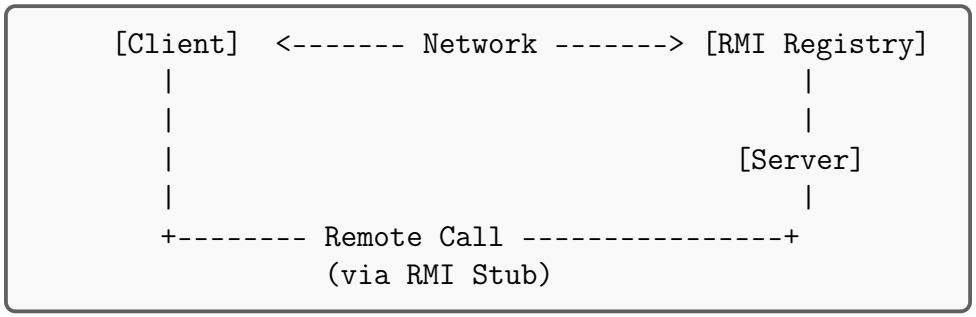
- Implementing the business logic for mathematical operations
- Registering the remote object in the RMI Registry
- Waiting for and processing client requests
- Returning calculated results

Client Component

Responsible for:

- Locating the remote object in the RMI Registry
- Interacting with the user through a console interface
- Sending calculation requests to the server
- Displaying received results

2.2 Component Diagram



3 System Components

3.1 Calculadora Interface

3.1.1 Description

The `Calculadora` interface defines the remote contract that must be implemented by the server. It extends the Java RMI `Remote` interface, marking it as an object that can be invoked remotely.

3.1.2 Available Methods

Method	Parameters	Description
`somar`	Double a, Double b	Returns the sum of two numbers
`subtrair`	Double a, Double b	Returns the subtraction of two numbers
`multiplicar`	Double a, Double b	Returns the multiplication of two numbers
`dividir`	Double a, Double b	Returns the division of two numbers

Table 1: Methods of the Calculadora interface

3.1.3 Source Code

```java
import java.rmi.Remote;
import java.rmi.RemoteException;

public interface Calculadora extends Remote {
    Double somar(Double a, Double b) throws RemoteException;
    Double subtrair(Double a, Double b) throws RemoteException;
    Double multiplicar(Double a, Double b) throws RemoteException;
    Double dividir(Double a, Double b) throws RemoteException;
}
```

Listing 1: Calculadora Interface

3.2 CalculadoraImpl Class

3.2.1 Description

Concrete implementation of the `Calculadora` interface. This class extends `UnicastRemoteObject`, becoming an exported remote object that can receive remote method calls.

3.2.2 Characteristics

- **Inheritance:** Extends `UnicastRemoteObject`
- **Implementation:** `Calculadora` interface
- **Constructor:** Throws `RemoteException` for RMI compliance
- **Operations:** Implements the four basic arithmetic operations

3.2.3 Technical Notes

Note on Division by Zero

The code contains a comment indicating that division by zero handling should be done at input. Currently, this validation is performed on the client side.

3.2.4 Source Code

```java
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;

public class CalculadoraImpl extends UnicastRemoteObject
        implements Calculadora {

    public CalculadoraImpl() throws RemoteException {}

    public Double somar(Double n1, Double n2)
            throws RemoteException {
        return n1 + n2;
    }

    public Double subtrair(Double n1, Double n2)
            throws RemoteException {
        return n1 - n2;
```

```
    }

    public Double multiplicar(Double n1, Double n2)
            throws RemoteException {
        return n1 * n2;
    }

    public Double dividir(Double n1, Double n2)
            throws RemoteException {
        return n1 / n2;
        // Handle division by 0 at input
    }
}
```

Listing 2: CalculadoraImpl Class

3.3 Server Class

3.3.1 Description

The `Server` class is responsible for initializing and starting the RMI server. It creates an instance of the calculator implementation and registers it in the RMI Registry.

3.3.2 Functionality

1. Creates an instance of `CalculadoraImpl`
2. Creates or locates the RMI Registry on port 1099
3. Binds the calculator object with the name "Calculadora"
4. Keeps the server running to handle client requests

3.3.3 Source Code

```
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;

public class Server {
    public static void main(String[] args) {
        try {
            CalculadoraImpl obj = new CalculadoraImpl();
            Registry registry = LocateRegistry.createRegistry(1099);
            registry.rebind("Calculadora", obj);
            System.out.println("Server ready");
        } catch (Exception e) {
            System.err.println("Server error: " + e);
        }
    }
}
```

Listing 3: Server Class

3.4 Client Class

3.4.1 Description

The `Client` class provides the user interface for interacting with the remote calculator. It connects to the RMI Registry, obtains a reference to the remote object, and processes user operations.

3.4.2 Key Features

- **Interactive Menu:** Console-based interface for operation selection
- **Input Validation:** Prevents division by zero
- **Result Formatting:** Rounds results to two decimal places
- **Continuous Operation:** Runs in a loop until user exits
- **Error Handling:** Catches and displays exceptions

3.4.3 Supported Operations

Operation (Portuguese)	Function
adição	Addition
subtração	Subtraction
multiplicação	Multiplication
divisão	Division (with zero validation)
sair	Exit application

Table 2: Available client operations

3.4.4 Source Code

```
1  import java.rmi.RemoteException;
2  import java.rmi.registry.LocateRegistry;
3  import java.rmi.registry.Registry;
4  import java.util.Scanner;
5
6  public class Client {
7      public static void main(String[] args) throws RemoteException {
8          Double response = 0.0;
9          double pvalor;
10         double svalor;
11         String entrada;
12         Scanner sc = new Scanner(System.in);
13
14         try {
15             Registry registry = LocateRegistry.getRegistry("localhost");
16             Calculadora stub = (Calculadora) registry.lookup("
    Calculadora");
17
18             do {
```

```
                System.out.println("Enter operation (adicao / multiplicacao / " +
                                   "divisao / subtracao) or sair: ");
                entrada = sc.nextLine();

                switch (entrada) {
                    case "adicao" -> {
                        System.out.println("Enter first value: ");
                        pvalor = sc.nextDouble();
                        System.out.println("Enter second value: ");
                        svalor = sc.nextDouble();
                        sc.nextLine();
                        response = stub.somar(pvalor, svalor);
                        response = Math.round(response * 100) / 100.0;
                    }
                    case "multiplicacao" -> {
                        System.out.println("Enter first value: ");
                        pvalor = sc.nextDouble();
                        System.out.println("Enter second value: ");
                        svalor = sc.nextDouble();
                        sc.nextLine();
                        response = stub.multiplicar(pvalor, svalor);
                        response = Math.round(response * 100) / 100.0;
                    }
                    case "divisao" -> {
                        System.out.println("Enter first value: ");
                        pvalor = sc.nextDouble();
                        do {
                            System.out.println("Enter second value: ");
                            svalor = sc.nextDouble();
                            sc.nextLine();
                            if (svalor == 0) {
                                System.out.println("Divisor must be non-zero");
                            }
                        } while (svalor == 0);
                        response = stub.dividir(pvalor, svalor);
                        response = Math.round(response * 100) / 100.0;
                    }
                    case "subtracao" -> {
                        System.out.println("Enter first value: ");
                        pvalor = sc.nextDouble();
                        System.out.println("Enter second value: ");
                        svalor = sc.nextDouble();
                        sc.nextLine();
                        response = stub.subtrair(pvalor, svalor);
                        response = Math.round(response * 100) / 100.0;
                    }
                    case "sair" -> System.exit(0);
                }
                System.out.println("Server response: " + response);
            } while (true);

        } catch (Exception e) {
            System.err.println("Client error: " + e);
        }
    }
}
```

Listing 4: Client Class

4 Communication Flow

4.1 Sequence Diagram

The typical communication flow between client and server follows these steps:

1. **Server Initialization**
    - Server creates CalculadoraImpl instance
    - Server creates RMI Registry on port 1099
    - Server registers calculator object as "Calculadora"
2. **Client Connection**
    - Client locates RMI Registry at localhost
    - Client looks up "Calculadora" object
    - Client receives stub (proxy) to remote object
3. **Operation Execution**
    - User selects operation and inputs values
    - Client invokes method on stub
    - Stub serializes parameters and sends to server
    - Server executes method and returns result
    - Client receives and displays result
4. **Continuous Operation**
    - Process repeats until user chooses to exit

5 Deployment and Execution

5.1 Prerequisites

- Java Development Kit (JDK) 8 or higher
- Network connectivity between client and server machines
- Port 1099 available for RMI Registry

5.2 Compilation

```
javac Calculadora.java
javac CalculadoraImpl.java
javac Server.java
javac Client.java
```

Listing 5: Compilation Commands

5.3 Execution Procedure

5.3.1 Starting the Server

```
java Server
```

Listing 6: Server Execution

Expected output:

```
Server ready
```

5.3.2 Starting the Client

```
java Client
```

Listing 7: Client Execution

5.4 Configuration for Remote Deployment

For deployment across different machines, modify the client connection:

```
// Replace "localhost" with server IP address
Registry registry = LocateRegistry.getRegistry("192.168.1.100");
```

Listing 8: Remote Server Connection

Security Note

When deploying across networks, ensure proper firewall configuration and consider implementing authentication and encryption mechanisms for production environments.

6 Error Handling and Validation

6.1 Exception Handling

The system implements comprehensive exception handling:

Exception Type	Handling Strategy
RemoteException	Caught and logged with descriptive error messages
NotBoundException	Occurs if "Calculadora" object not found in registry
Division by Zero	Prevented through client-side validation loop
Input Format Errors	Handled by Scanner exception mechanisms

Table 3: Exception handling strategies

6.2 Input Validation

6.2.1 Division by Zero Prevention

The client implements a validation loop that prevents division by zero:

```
do {
    System.out.println("Enter second value: ");
    svalor = sc.nextDouble();
    sc.nextLine();
    if (svalor == 0) {
        System.out.println("Divisor must be non-zero");
    }
} while (svalor == 0);
```

Listing 9: Division Validation Logic

6.2.2 Result Precision

All results are rounded to two decimal places:

```
response = Math.round(response * 100) / 100.0;
```

Listing 10: Result Rounding

7 Advantages

- **Distributed Architecture:** Enables separation of client and server on different machines
- **Scalability:** Multiple clients can connect simultaneously
- **Transparency:** RMI provides location transparency for remote objects
- **Type Safety:** Java's strong typing ensures type-safe remote calls
- **Simplicity:** Clean interface definition and straightforward implementation

8 Possible Enhancements

8.1 Security Improvements

- Implement SSL/TLS for encrypted communication

- Add user authentication and authorization
- Implement access control lists (ACLs)

8.2 Functionality Extensions

- Add scientific calculator operations (power, square root, trigonometry)
- Implement expression parsing for complex calculations
- Add calculation history and logging
- Support for multiple number bases (binary, hexadecimal)

8.3 User Interface Improvements

- Develop graphical user interface (GUI)
- Internationalization (i18n) support for multiple languages
- Improved error messages and user guidance
- Command-line argument support for configuration

8.4 Architectural Enhancements

- Implement connection pooling
- Add load balancing for multiple server instances
- Implement failover and redundancy mechanisms
- Add monitoring and metrics collection

9 Conclusion

This distributed calculator system demonstrates effective use of Java RMI technology to create a client-server application. The implementation provides a solid foundation for understanding distributed computing concepts including remote method invocation, serialization, and network communication.

The system successfully achieves its primary objectives of providing remote arithmetic operations through a clean, well-structured architecture. While the current implementation serves as an excellent educational example, the suggested enhancements would make it suitable for production use.

The modular design and clear separation of concerns make the codebase maintainable and extensible, providing numerous opportunities for further development and learning.