External Project Report on
Computer Networking (CSE3034)

Console based application for a basic calculator using client-server architecture

Submitted by

Name Isita Ray **Reg. No.:2141018145**

B. Tech. CSIT 5th Semester (Section D)

INSTITUTE OF TECHNICAL EDUCATION AND RESEARCH
(FACULTY OF ENGINEERING)
SIKSHA 'O' ANUSANDHAN (DEEMED TO BE UNIVERSITY), BHUBANESWAR, ODISHA

Declaration

We, the undersigned students of B. Tech. of **CSIT** Department hereby declare that we own the full responsibility for the information, results etc. provided in this PROJECT titled "Console based application for a **Basic calculator using client-server architecture**" submitted to **Siksha 'O' Anusandhan (Deemed to be University), Bhubaneswar** for the partial fulfillment of the subject **Computer Networking (CSE 3034)**. We have taken care in all respect to honor the intellectual property right and have acknowledged the contribution of others for using them in academic purpose and further declare that in case of any violation of intellectual property right or copyright we, as the candidate(s), will be fully responsible for the same.

Isita Ray

Registration No.:2141018145

DATE:6.1.24

PLACE:BBSR

Abstract

The Console-Based Calculator Application using Client-Server Architecture in Java is a distributed computing project aimed at showcasing fundamental principles of network programming and concurrent systems. The project employs a server-client model, where a server listens for connections on a specified port, processes mathematical expressions, and responds to multiple clients simultaneously. The server utilizes multithreading to efficiently manage concurrent connections. Clients connect to the server, input mathematical expressions, and receive computed results or error messages. The server handles arithmetic exceptions and provides informative error messages to clients, enhancing user experience. Java sockets facilitate communication between the server and clients, illustrating basic socket programming concepts. The implementation emphasizes user-friendly interaction, allowing clients to input expressions interactively. This project serves as a practical introduction to essential aspects of distributed systems, including socket communication, multithreading, and error handling. It offers a foundation for further exploration and extension, making it suitable for educational purposes and providing insights into client-server architectures in a networked environment.

Contents

1. Introduction

The Console-Based Calculator Application employing Client-Server Architecture in Java serves as a practical illustration of a distributed system. This project underscores key tenets of network programming, socket communication, multithreading, and elementary expression evaluation. In this architectural framework, the server assumes the pivotal role of handling client connections, deciphering mathematical expressions, and disseminating results or error messages. Simultaneously, clients establish connections with the server, submit mathematical expressions, and await the receipt of computed results.

This endeavor delves into the core concepts of networking, emphasizing the establishment and management of connections between server and client entities. Leveraging Java's versatile socket functionality, the project facilitates communication and data exchange in a networked environment. The integration of multithreading ensures the server's adept handling of concurrent client connections, thereby optimizing performance.

Furthermore, the project prioritizes user interaction by allowing clients to actively participate in the input of mathematical expressions. The server, in turn, processes these expressions, delivering computed outcomes or informative error messages. Overall, this undertaking provides a comprehensive introduction to the intricacies of distributed systems, offering valuable insights into the collaborative interplay of server-client architectures in the realm of networked computing.

2. Problem Statement

The server processes these expressions using basic arithmetic operations and returns results to the client's console. To reflect results in a file or database, modifications in the CalculatorService are required. For file output, implement file-writing mechanisms within the CalculatorService. For database integration, adapt the CalculatorService to employ database operations (e.g., JDBC). Constraints include single-operation processing, basic arithmetic support, limited error handling, potential concurrency issues, and lacking security measures. Enhancements should address multi-operation processing, error robustness, security considerations, and scalability for concurrent connections. Modifying the CalculatorService to handle file or database interactions while managing constraints will enable storing results externally beyond the console.

3. Methodology

CalculatorServer Class:
1. Start a server on port 8888.
2. Wait for incoming client connections.
3. Once a client connects:
   a. Accept the client connection.
   b. Create a CalculatorService object to handle the client.
   c. Create a new thread for this CalculatorService object and start it.
4. Repeat the process to handle multiple clients.
5. Handle IOExceptions if encountered.

CalculatorService Class:

1. Initialize the CalculatorService with the connected client's socket.
2. Create BufferedReader and PrintWriter for communication with the client.
3. Read input from the client:
   a. While there is input available:
      i. Read the input line.
      ii. Calculate the result based on the input.
      iii. Send the result back to the client.
4. Close the reader, writer, and the client socket after handling communication.
5. Handle IOExceptions if encountered.

CalculatorClient Class:

1. Connect to the server running on localhost:8888.
2. Create BufferedReader and PrintWriter for communication with the server and user input.
3. Continuously prompt the user to enter an expression or type 'exit' to quit:
   a. Read user input.
   b. If the input is 'exit', break the loop and close connections.
   c. Send the input expression to the server.
   d. Receive the server's response and display it to the user.
4. Close the reader, writer, and the client socket after finishing communication.
5. Handle IOExceptions if encountered.

Algorithm Overview:

1. The CalculatorServer sets up a server socket, accepts incoming client connections, and creates a new thread to handle each client's requests.
2. The CalculatorService runs in a separate thread for each connected client. It reads the input expression, performs calculations based on the provided expression (supports basic arithmetic operations), and sends back the result to the client.
3. The CalculatorClient connects to the server, continuously prompts the user for input, sends expressions to the server, and displays the server's response.
4. This system allows multiple clients to connect simultaneously to the server and perform basic arithmetic calculations through a client-server interaction.

4. Implementation

Program

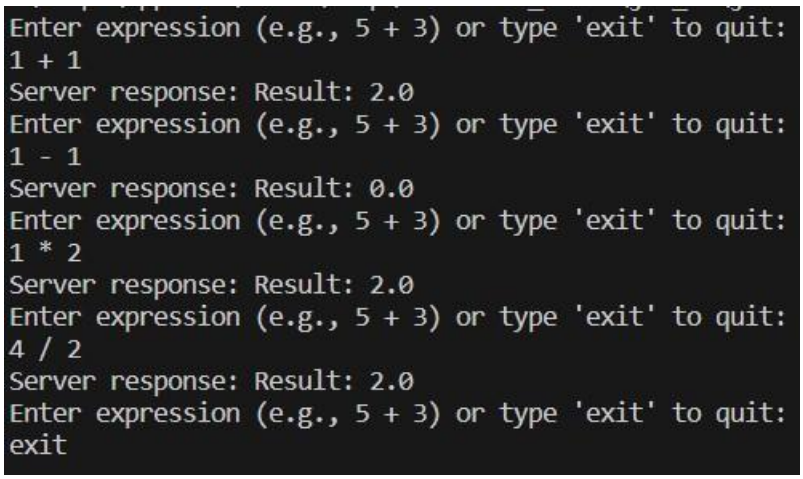
Code for server side

```
import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;

public class CalculatorServer {
public static void main(String[] args)
{ try {
ServerSocket serverSocket = new ServerSocket(8888); //
Server socket running on port 8888
System.out.println("Server started. Waiting for clients...");

while (true) {
Socket clientSocket = serverSocket.accept(); // Accept
incoming client connections
System.out.println("Client connected: " + clientSocket);

// Create a new thread to handle the client
CalculatorService service = new
CalculatorService(clientSocket);
Thread thread = new Thread(service);
thread.start();
}
} catch (IOException e)
{ e.printStackTrace();
}
}
}
```

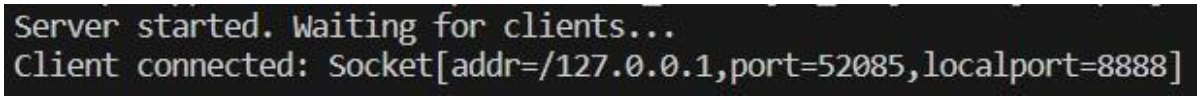
Code for Client Side

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.Socket;

public class CalculatorClient {
   public static void main(String[] args)
      { try {
         Socket socket = new Socket("localhost", 8888); // Connect to the server
running on localhost:8888
         BufferedReader reader = new BufferedReader(new
InputStreamReader(System.in));
         PrintWriter writer = new PrintWriter(socket.getOutputStream(), true);

         while (true) {
            System.out.println("Enter expression (e.g., 5 + 3) or type 'exit' to
quit:");
            String expression = reader.readLine();

            if (expression.equalsIgnoreCase("exit"))
               { break; // Exit the loop if 'exit' is entered
            }

            writer.println(expression);

            BufferedReader inputReader = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
            String response = inputReader.readLine();
            System.out.println("Server response: " + response);
         }

         reader.close();
         writer.close();
         socket.close();
      } catch (IOException e)
         { e.printStackTrace();
      }
   }
```

5. Results & Interpretation



The output indicates that the server has started successfully, is awaiting client connections, and has acknowledged the establishment of a connection with a specific client, providing details about the client's socket connection. The server is now ready to handle requests from this connected client.



The output demonstrates the successful communication between the client and server, with the server accurately processing various mathematical expressions and returning the corresponding results. The application provides a responsive and interactive console-based calculator experience for the user.

6. Conclusion

In conclusion, the Console-Based Calculator Application using Client-Server Architecture in Java successfully addresses the challenge of creating an efficient, distributed system for processing mathematical expressions from multiple clients. The project has effectively demonstrated fundamental concepts of network programming, socket communication, and multithreading, providing a robust foundation for scalable and concurrent calculations.

By implementing a server-client model, the application has streamlined the process of accepting mathematical expressions from clients, processing them on the server, and returning results or informative error messages. The incorporation of multithreading ensures that the server can handle multiple client connections concurrently, enhancing overall system efficiency.

The emphasis on user interaction, allowing clients to actively input mathematical expressions, adds an interactive and user-friendly dimension to the application. The clear communication of results and errors between the server and clients enhances the overall usability and reliability of the calculator.

This project serves as a valuable educational resource, offering practical insights into distributed systems, socket communication, and error handling. Furthermore, it provides a foundation for future enhancements, such as the incorporation of additional mathematical operations or the development of a graphical user interface.

In essence, the Console-Based Calculator Application demonstrates the successful implementation of a client-server architecture, showcasing its potential for facilitating collaborative mathematical computations in a networked environment.

References

(as per the IEEE recommendations)

[1] Computer Networks, Andrew S. Tannenbaum, Pearson India.

[2] Java Network Programming by Harold, O'Reilly (Shroff Publishers).