ㅣ

KIM, CHAEMIN

# github URL (optional):

|  |  |
| --- | --- |
|  | [과제의 목적을 간단하게 설명] |

# #개요

|  |  |
| --- | --- |
|  | 주제: Calculator application using Java socket API 언어: Java  프로그램의 요구조건  1. Define a protocol for Command/Response (communication message formats)  2. The calculator should support the following four operations  3. The response from the server may be either answer for the expression or error message  4. The server can handle multiple clients at a time by Thread Pool & Runnable interface  5. The client attempts to connect by reading a text file containing the server's IP and port number information |

# REPORT

* 과제의 기본 요구 사항을 이곳에 원하는 만큼 작성 (e.g. 구조도, protocol 내용 상세 설명 등)
* Procotol에 대한 설명
* 소스코드는 마지막에 제공하는 양식을 이용하여 제출 (zip파일로도 제출)

# PRotocol

|  |  |
| --- | --- |
|  | 소켓 프로그래밍을 설명할 때, 사용하는 가장 일반적인 프로토콜은 TCP와 UDP이다.  TCP는 전송 제어 프로토콜로, 이 프로토콜의 특징으로는 신뢰할 수 있는 연결 지향적 통신 제공과 연속적인 스트림 지향성, 수신확인 및 재전송 메커니즘을 사용함으로써 오류를 줄이는 신뢰성 등이 있다.  UDP는 사용자 데이터그램 프로토콜로, 이 프로토콜의 특징은 각 패킷을 독립적인 데이터 단위로 취급하고, 데이터그램을 지향한다. 또한 TCP에 비해 오버헤드가 낮지만 지연 시간을 줄이기 위해 신뢰성을 희생한다  TCP를 사용하든 UDP를 사용하든 간에 소켓을 통해 전송되는 실제 메시지는 특정 형식을 따라야 하는데 그 형색은 아래와 같다:  - 헤더: 메시지의 길이, 유형 같은 메시지에 정보 포함  - Data Payload: 메시지의 실제 내용  - 바닥글/CRC: 데이터의 무결성을 보장하기 위한 오류 검사 정보 |

# ARCHITECTURE DIAGRAM

|  |
| --- |
|  |

## Server SOURCE CODES

|  |
| --- |
| **import** java.io.BufferedReader;  **import** java.io.BufferedWriter;  **import** java.io.IOException;  **import** java.io.InputStreamReader;  **import** java.io.OutputStreamWriter;  **import** java.net.ServerSocket;  **import** java.net.Socket;  **import** java.util.StringTokenizer;  **import** java.util.concurrent.ExecutorService;  **import** java.util.concurrent.Executors;  /\*\*  \* A simple server application that performs basic arithmetic calculations based on client requests.  \*/  **public** **class** HW1\_Server {  /\*\*  \* Calculates the result of a given arithmetic expression.  \*  \* **@param** exp The arithmetic expression to be evaluated.  \* **@return** The result of the calculation or an error message if the expression is invalid.  \*/  **public** **static** String calc(String exp) {  StringTokenizer st = **new** StringTokenizer(exp, " ");  **if** (st.countTokens() < 3)  **return** "error: too few arguments";  **if** (st.countTokens() > 3)  **return** "error: too many arguments";  String res = "";  **int** op1 = Integer.*parseInt*(st.nextToken());  String opcode = st.nextToken();  **int** op2 = Integer.*parseInt*(st.nextToken());  **switch** (opcode) {  **case** "+":  res = Integer.*toString*(op1 + op2);  System.***out***.println(res);  **break**;  **case** "-":  res = Integer.*toString*(op1 - op2);  System.***out***.println(res);  **break**;  **case** "\*":  res = Integer.*toString*(op1 \* op2);  System.***out***.println(res);  **break**;  **case** "/":  **if** (op2 == 0) **return** "error: divided by zero";  res = Integer.*toString*(op1 / op2);  System.***out***.println(res);  **break**;  **default**:  res = "error";  }  **return** res;  }  /\*\*  \* The main entry point for the server application.  \*  \* **@param** args Command line arguments (not used in this application).  \*/  **public** **static** **void** main(String[] args) {  ExecutorService executorService = Executors.*newFixedThreadPool*(5);  **try** (ServerSocket listener = **new** ServerSocket(9999)) {  System.***out***.println("Waiting for connections...");  **while** (**true**) {  Socket socket = listener.accept();  System.***out***.println("Connected");  Runnable worker = **new** ServerWorker(socket);  executorService.execute(worker);  }  } **catch** (IOException e) {  System.***out***.println(e.getMessage());  } **finally** {  executorService.shutdown();  }  }  /\*\*  \* A worker thread that handles communication with a specific client.  \*/  **static** **class** ServerWorker **implements** Runnable {  **private** Socket socket;  /\*\*  \* Constructs a new ServerWorker instance.  \*  \* **@param** socket The socket associated with the client connection.  \*/  **public** ServerWorker(Socket socket) {  **this**.socket = socket;  }  @Override  **public** **void** run() {  **try** (BufferedReader in = **new** BufferedReader(**new** InputStreamReader(socket.getInputStream()));  BufferedWriter out = **new** BufferedWriter(**new** OutputStreamWriter(socket.getOutputStream()))) {  **while** (**true**) {  String inputMessage = in.readLine();  **if** (inputMessage.equalsIgnoreCase("bye")) {  System.***out***.println("Connection closed");  **break**;  }  System.***out***.println("Received: " + inputMessage);  String res = *calc*(inputMessage);  out.write(res + "\n");  out.flush();  }  } **catch** (IOException e) {  e.printStackTrace();  } **finally** {  **try** {  **if** (socket != **null**)  socket.close();  } **catch** (IOException e) {  e.printStackTrace();  }  }  }  }  } |

## Client SOURCE CODES

|  |
| --- |
| **import** java.io.BufferedReader;  **import** java.io.BufferedWriter;  **import** java.io.IOException;  **import** java.io.InputStreamReader;  **import** java.io.OutputStreamWriter;  **import** java.net.Socket;  **import** java.util.Scanner;  /\*\*  \* A simple client application for interacting with the arithmetic calculator server.  \*/  **public** **class** HW1\_Client {  **public** **static** **void** main(String[] args) {  BufferedReader in = **null**;  BufferedWriter out = **null**;  Socket socket = **null**;  Scanner scanner = **new** Scanner(System.***in***);  **try** {  // Connect to the server on localhost and port 9999  socket = **new** Socket("localhost", 9999);  in = **new** BufferedReader(**new** InputStreamReader(socket.getInputStream()));  out = **new** BufferedWriter(**new** OutputStreamWriter(socket.getOutputStream()));  **while** (**true**) {  System.***out***.println("Enter an arithmetic expression (separated by spaces): ");  String outputMessage = scanner.nextLine(); // Read the arithmetic expression from the keyboard  **if** (outputMessage.equalsIgnoreCase("bye")) {  // If the user enters "bye," send it to the server and break out of the loop  out.write(outputMessage + "\n");  out.flush();  **break**;  }  // Send the arithmetic expression to the server  out.write(outputMessage + "\n");  out.flush();  // Receive and print the result from the server  String inputMessage = in.readLine();  System.***out***.println("Calculation result: " + inputMessage);  }  } **catch** (IOException e) {  System.***out***.println(e.getMessage());  } **finally** {  **try** {  // Close resources and handle potential exceptions  scanner.close();  **if** (socket != **null**)  socket.close();  } **catch** (IOException e) {  System.***out***.println("An error occurred while chatting with the server.");  }  }  }  } |

## OUTPUT (Screen Shots)

|  |  |
| --- | --- |
| #1 | 위에는 client, 아래는 server |
|  | |

|  |  |
| --- | --- |
| #2 | 위에는 client, 아래는 server |
|  | |

|  |  |
| --- | --- |
| #3 | 위에는 client, 아래는 server |
|  | |