Koo, JUNSEO (구준서)

# github URL (optional): https://github.com/jsk4581/QuizGame/

|  |  |
| --- | --- |
|  | **VERSION control & WIKI** |

# #개요

|  |  |
| --- | --- |
|  | **자바 소켓 프로그래밍을 활용하여, 서버와 클라이언트가 통신하며 진행하는 퀴즈게임을 구현한다** |

**Requirements**

* < 요구조건 #1: 서버와 클라이언트가 connection-oriented 방식으로 통신 >
* < 요구조건 #2: 서버에서 퀴즈 문제를 제공하면, 클라이언트는 실시간으로 답을 입력하고 서버의 피드백을 받아야 함 >
* < 요구조건 #3: 통신은 미리 정의된 아스키 기반 통신규약에 따라 이루어져야 함 >
* < 요구조건 #4: 게임이 끝나면 점수를 알려주고, 연결을 종료해야 함 >

# REPORT

* 구조도

텍스트, 스크린샷, 폰트, 도표이(가) 표시된 사진

자동 생성된 설명

* 추가기능
* 클라이언트는 server\_info.dat 파일에서 서버 설정을 읽어옴
* 퀴즈를 랜덤하게 섞는 기능
* 대소문자나 ‘layer’ 키워드 유무를 무시하고 정답 판별
* 서버 ThreadPool 활용 다중 클라이언트 연결

# PRotocol

|  |  |
| --- | --- |
|  | **통신 메시지 형식을 정의** |

* **서버 -> 클라이언트 메시지**

**[QUESTION, 질문전송]**

QUESTIONX:<question\_text>

* X는 1부터 시작하는 질문번호이며, :<question\_text>에 퀴즈구문이 포함된다.

**[RESULT, 답변처리 형식]**

RESULT:CORRECT

* 응답이 정답인 경우

RESULT:WRONG:<correct\_answer>

* 응답이 오답인경우, <correct\_answer>에 올바른 답안을 제시

RESULT:INVALID:<error\_message>

* 예외처리: 클라이언트가 ANSWER: 만 보내고 실제 응답이 비어있는 경우, 에러메시지 전송 (Answer cannot be empty)
* 예외처리: 클라이언트가 ANSWER:로 시작하지 않는 메시지를 전송한 경우, 에러메시지 전송

(Invalid answer format. Please use 'ANSWER:<your answer>')

RESULT:ERROR:<error\_message>

* 예외처리: 서버의 정답데이터가 NULL인 경우, 에러메시지를 전송 (ERROR:Server-side issue. Skipping this question.)

**[QUIZ\_END, 퀴즈종료를 나타냄]**

QUIZ\_END:Your total score: X/Y

* 최종점수를 전송, X는 맞힌 수, Y는 문제의 수
* **클라이언트 -> 서버 메시지**

**[ANSWER, 답변전송]**

ANSWER:<user\_answer>

* <user\_answer>: 클라이언트가 제출한 답변의 내용.

## SOURCE CODES - SERVER

|  |
| --- |
| **import java.io.\*;**  **import java.net.\*;**  **import java.util.Collections;**  **import java.util.List;**  **import java.util.ArrayList;**  **import java.util.concurrent.ExecutorService;**  **import java.util.concurrent.Executors;**  **import java.util.concurrent.atomic.AtomicInteger;**  **public class QuizServer {**  **// Updated set of questions and their answers**  **private static final String[][] QUESTIONS = {**  **{"Enter the name of matching layer - Protocols that are part of a distributed network application.", "application layer"},**  **{"Enter the name of matching layer - Transfer of data between one process and another process (typically on different hosts)", "transport layer"},**  **{"Enter the name of matching layer - Delivery of datagrams from a source host to a destination host (typically).", "network layer"},**  **{"Enter the name of matching layer - Transfer of data between neighboring network devices.", "link layer"},**  **{"Enter the name of matching layer - Transfer of a bit into and out of a transmission media.", "physical layer"}**  **};**  **// Atomic counter for client numbers**  **private static final AtomicInteger clientCounter = new AtomicInteger(0);**  **public static void main(String[] args) {**  **ExecutorService threadPool = Executors.newFixedThreadPool(5); // ThreadPool with 5 threads**  **try (ServerSocket serverSocket = new ServerSocket(8888)) { // Create server socket on port 8888**  **System.out.println("Quiz Server is running...");**  **while (true) { // Keep server running for multiple clients**  **System.out.println("Waiting for a client...");**  **Socket clientSocket = serverSocket.accept(); // Wait for a client connection**  **int clientNumber = clientCounter.incrementAndGet(); // Increment client counter**  **System.out.println("Client " + clientNumber + " connected!");**  **// Submit client handling task to the thread pool**  **threadPool.submit(new ClientHandler(clientSocket, QUESTIONS, clientNumber));**  **}**  **} catch (IOException e) {**  **System.out.println("Server error: " + e.getMessage());**  **} finally {**  **// Shutdown thread pool gracefully**  **threadPool.shutdown();**  **System.out.println("Server shut down.");**  **}**  **}**  **}**  **// ClientHandler implements Runnable to handle each client in a separate thread**  **class ClientHandler implements Runnable {**  **private final Socket clientSocket;**  **private final String[][] questions;**  **private final int clientNumber;**  **public ClientHandler(Socket clientSocket, String[][] questions, int clientNumber) {**  **this.clientSocket = clientSocket;**  **this.questions = questions;**  **this.clientNumber = clientNumber;**  **}**  **@Override**  **public void run() {**  **try (**  **BufferedReader in = new BufferedReader(new InputStreamReader(clientSocket.getInputStream())); // To read data from the client**  **BufferedWriter out = new BufferedWriter(new OutputStreamWriter(clientSocket.getOutputStream())) // To send data to the client**  **) {**  **// Notify client of their number**  **out.write("Welcome, Client " + clientNumber + "!\n");**  **out.flush();**  **// Create a list of question indices and shuffle for random order**  **List<Integer> questionOrder = new ArrayList<>();**  **for (int i = 0; i < questions.length; i++) {**  **questionOrder.add(i);**  **}**  **Collections.shuffle(questionOrder); // Shuffle the question order**  **int score = 0; // Initialize score**  **// Iterate through the shuffled question indices**  **for (int i = 0; i < questionOrder.size(); i++) {**  **int questionIndex = questionOrder.get(i);**  **String[] question = questions[questionIndex];**  **// Send question to client with its unique number in ASCII-based format**  **out.write("QUESTION" + (i + 1) + ":" + question[0] + "\n");**  **out.flush();**  **// Read client's answer**  **String clientAnswer = in.readLine();**  **if (clientAnswer == null) {**  **System.out.println("Client " + clientNumber + " disconnected unexpectedly.");**  **break; // End the quiz for this client**  **}**  **// Check if the client's message starts with "ANSWER:"**  **if (!clientAnswer.startsWith("ANSWER:")) {**  **out.write("RESULT:INVALID:Invalid answer format. Please use 'ANSWER:<your answer>'\n");**  **out.flush();**  **continue; // Skip to the next question**  **}**  **String answer = clientAnswer.substring(7).trim().toLowerCase();**  **if (answer.isEmpty()) {**  **out.write("RESULT:INVALID:Answer cannot be empty\n");**  **out.flush();**  **continue; // Skip to the next question**  **}**  **String correctAnswer = question[1];**  **if (correctAnswer == null) {**  **out.write("RESULT:ERROR:Server-side issue. Skipping this question.\n");**  **out.flush();**  **continue; // Skip to the next question**  **}**  **// Normalize and compare the answers**  **answer = answer.replace("layer", "").trim();**  **correctAnswer = correctAnswer.toLowerCase().replace("layer", "").trim();**  **if (answer.equals(correctAnswer)) {**  **// If the answer is correct**  **score++;**  **out.write("RESULT:CORRECT\n");**  **} else {**  **// If the answer is incorrect, send the correct answer**  **out.write("RESULT:WRONG:" + question[1] + "\n");**  **}**  **out.flush();**  **}**  **// Send the final score to the client**  **out.write("QUIZ\_END:Your total score: " + score + "/" + questions.length + "\n");**  **out.flush();**  **} catch (IOException e) {**  **System.out.println("Error handling Client " + clientNumber + ": " + e.getMessage());**  **} finally {**  **// Ensure the client socket is closed**  **try {**  **clientSocket.close();**  **System.out.println("Client " + clientNumber + " connection closed.");**  **} catch (IOException e) {**  **System.out.println("Error closing Client " + clientNumber + "'s socket: " + e.getMessage());**  **}**  **}**  **}**  **}** |

## SOURCE CODES - CLIENT

|  |
| --- |
| **import java.io.\*;**  **import java.net.\*;**  **public class QuizClient {**  **public static void main(String[] args) {**  **String serverIP = "localhost"; // Default IP**  **int serverPort = 8888; // Default port**  **try (Socket socket = new Socket(serverIP, serverPort)) { // Connect to server with IP and port**  **System.out.println("Connected to Quiz Server");**  **try (**  **BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream())); // To receive data from the server**  **BufferedReader userInput = new BufferedReader(new InputStreamReader(System.in)); // To read user input from the console**  **BufferedWriter out = new BufferedWriter(new OutputStreamWriter(socket.getOutputStream())) // To send data to the server**  **) {**  **String serverMessage; // Variable to store messages from the server**  **// Loop to handle the quiz interaction**  **while ((serverMessage = in.readLine()) != null) {**  **if (serverMessage.startsWith("QUESTION")) {**  **// Display the question to the user**  **System.out.println(serverMessage);**  **System.out.print("Your answer: ");**  **String answer = userInput.readLine(); // Read the user's answer**  **out.write("ANSWER:" + answer + "\n"); // Send the answer to the server**  **out.flush();**  **} else if (serverMessage.startsWith("RESULT:")) {**  **if (serverMessage.startsWith("RESULT:CORRECT")) {**  **System.out.println("Correct!");**  **} else if (serverMessage.startsWith("RESULT:WRONG:")) {**  **System.out.println("Wrong! The correct answer is: " + serverMessage.substring(13));**  **} else if (serverMessage.startsWith("RESULT:INVALID:")) {**  **System.out.println("Invalid answer: " + serverMessage.substring(15));**  **} else if (serverMessage.startsWith("RESULT:ERROR:")) {**  **System.out.println("Error: " + serverMessage.substring(13));**  **}**  **} else if (serverMessage.startsWith("QUIZ\_END:")) {**  **// If the quiz has ended, display the final score**  **System.out.println(serverMessage);**  **break; // Exit the loop**  **}**  **}**  **}**  **} catch (IOException e) {**  **// Handle client-side errors**  **System.out.println("Client error: Unable to connect to Quiz Server");**  **System.out.println("Error: " + e.getMessage());**  **}**  **}**  **}** |

## OUTPUT (Screen Shots)

|  |  |
| --- | --- |
| #1 | Server wait & connection |
|  | |

|  |  |
| --- | --- |
| #2 | Client playing quiz |
|  | |

|  |  |
| --- | --- |
| #3 | Multi client support (server thread) |
|  | |

|  |  |
| --- | --- |
| #4 | Example of RESULT:INVALID |
|  | |