**Sprawozdanie**

**z przedmiotu**

**Programowanie w Języku Java**

**Laboratorium komputerowe nr 4**

**Autorzy:**

**Sylwia Jaworska**

**Grzegorz Listwan**

**Krzysztof Pacura**

Temat zadania

Wykorzystaj aplikację napisaną na laboratorium nr 2 (TCP/UDP), zmieniając odczyt/zapis plików tekstowych na odczyt/zapis danych z bazy MySQL. Bazę danych i tabele stwórz z poziomu języka Java. Oprócz kodu, umieść w rozwiązaniu również screeny potwierdzające działanie aplikacji.

**Program składa się z pięciu klas ClientHandler, CreateDatabaseTable, TCPClient, TCPServer, Qusetion.**

**Klasa CreateDatabaseTable**

public class CreateDatabaseTable {  
 static final String *JDBC\_DRIVER* = "com.mysql.cj.jdbc.Driver";  
 static final String *DB\_URL* = "jdbc:mysql://localhost/";  
  
 static final String *USER* = "root";  
 static final String *PASS* = "";  
  
 public static void main(String[] args) {  
  
  
 Connection connection = null;  
 Statement statement = null;  
  
 try {  
 Class.*forName*(*JDBC\_DRIVER*);  
  
 System.*out*.println("Łączenie z bazą danych ");  
 connection = DriverManager.*getConnection*(*DB\_URL*, *USER*, *PASS*);  
 statement = connection.createStatement();  
  
 String createDatabaseSQL = "CREATE DATABASE IF NOT EXISTS kolokwium";  
 statement.executeUpdate(createDatabaseSQL);  
 System.*out*.println("Baza danych utworzona");  
  
 String useDatabaseSQL = "USE kolokwium";  
 statement.executeUpdate(useDatabaseSQL);  
 System.*out*.println("Użyto bazy danych kolokwium");  
  
 String createTableSQL = "CREATE TABLE IF NOT EXISTS question " +  
 "(id int not null auto\_increment primary key, " +  
 "question varchar(255) not null," +  
 "answerA varchar(255) not null," +  
 "answerB varchar(255) not null," +  
 "answerC varchar(255) not null," +  
 "answerD varchar(255) not null," +  
 "correctAnswer varchar(1) not null)";  
 statement.executeUpdate(createTableSQL);  
 System.*out*.println("Stworzono tabelę question");  
  
  
 createTableSQL = "CREATE TABLE IF NOT EXISTS result " +  
 "(id int not null auto\_increment primary key, " +  
 "user varchar(255) not null," +  
 "result int not null)";  
 statement.executeUpdate(createTableSQL);  
 System.*out*.println("Stworzono tabelę result");  
  
 createTableSQL = "CREATE TABLE IF NOT EXISTS answer " +  
 "(id int not null auto\_increment primary key, " +  
 "user varchar(255) not null," +  
 "questionID int not null," +  
 "answer varchar(1) not null,"+  
 "foreign key (questionID) references question(id))";  
 statement.executeUpdate(createTableSQL);  
 System.*out*.println("Stworzono tabelę answer");  
 String insert = "INSERT INTO question (question, answerA, answerB, answerC, answerD, correctAnswer) VALUES ('1. Jakie jest stolica Francji?', 'a. Berlin', 'b. Paryż', 'c. Londyn', 'd. Madryt', 'b');";  
 statement.executeUpdate(insert);  
 System.*out*.println("dodano: "+insert);  
 insert = "INSERT INTO question (question, answerA, answerB, answerC, answerD, correctAnswer) VALUES ('2. Który pierwiastek chemiczny reprezentowany jest symbolem \"O\"?', 'a. Tlen', 'b. Wodór', 'c. Azot', 'd. Sód', 'a');";  
 statement.executeUpdate(insert);  
 System.*out*.println("dodano: "+insert);  
 insert = "INSERT INTO question (question, answerA, answerB, answerC, answerD, correctAnswer) VALUES ('3. Kto napisał \"Romeo i Julia\"?', 'a. Charles Dickens', 'b. William Shakespeare', 'c. Jane Austen', 'd. Fiodor Dostojewski', 'b');";  
 statement.executeUpdate(insert);  
 System.*out*.println("dodano: "+insert);  
 insert = "INSERT INTO question (question, answerA, answerB, answerC, answerD, correctAnswer) VALUES ('4. Jaki kolor ma tradycyjnie skrzynka na listy?', 'a. Czerwony', 'b. Zielony', 'c. Niebieski', 'd. Żółty', 'a');";  
 statement.executeUpdate(insert);  
 System.*out*.println("dodano: "+insert);  
 insert = "INSERT INTO question (question, answerA, answerB, answerC, answerD, correctAnswer) VALUES ('5. Ile kontynentów jest na świecie?', 'a. 5', 'b. 6', 'c. 7', 'd. 8', 'b');";  
 statement.executeUpdate(insert);  
 System.*out*.println("dodano: "+insert);  
 insert = "INSERT INTO question (question, answerA, answerB, answerC, answerD, correctAnswer) VALUES ('6. Która planeta jest znana jako \"Czerwona Planeta\"?', 'a. Wenus', 'b. Mars', 'c. Jowisz', 'd. Saturn', 'b');";  
 statement.executeUpdate(insert);  
 System.*out*.println("dodano: "+insert);  
 insert = "INSERT INTO question (question, answerA, answerB, answerC, answerD, correctAnswer) VALUES ('7. Które zwierzę jest symbolem mądrości w wielu kulturach?', 'a. Sowa', 'b. Tygrys', 'c. Żyrafa', 'd. Wąż', 'a');";  
 statement.executeUpdate(insert);  
 System.*out*.println("dodano: "+insert);  
 insert = "INSERT INTO question (question, answerA, answerB, answerC, answerD, correctAnswer) VALUES ('8. Jak nazywa się proces, podczas którego roślina przetwarza światło na energię?', 'a. Fotosynteza', 'b. Respiracja', 'c. Fermentacja', 'd. Dekompozycja', 'a');";  
 statement.executeUpdate(insert);  
 System.*out*.println("dodano: "+insert);  
 insert = "INSERT INTO question (question, answerA, answerB, answerC, answerD, correctAnswer) VALUES ('9. Które państwo jest największe pod względem powierzchni?', 'a. Rosja', 'b. Stany Zjednoczone', 'c. Kanada', 'd. Australia', 'a');";  
 statement.executeUpdate(insert);  
 System.*out*.println("dodano: "+insert);  
 insert = "INSERT INTO question (question, answerA, answerB, answerC, answerD, correctAnswer) VALUES ('10. Co oznacza skrót HTML w kontekście informatyki?', 'a. HyperText Markup Language', 'b. HighTech Modern Language', 'c. Hotmail', 'd. HyperTransfer Machine Language', 'a');";  
 statement.executeUpdate(insert);  
 System.*out*.println("dodano: "+insert);  
  
 } catch (SQLException | ClassNotFoundException e){  
 e.printStackTrace();  
 }finally {  
 try{  
 if (statement!=null)statement.close();  
 if(connection!=null)connection.close();  
 }catch (SQLException e){  
 e.printStackTrace();  
 }  
 }  
 }  
}

**W powyższej klasie wyróżniamy następujące części:**

1. Ustawianie Parametrów Połączenia:

Określenie sterownika JDBC (Java Database Connectivity) dla MySQL.

Zdefiniowanie adresu bazy danych, użytkownika i hasła.

1. Inicjalizacja Zmiennych Połączenia:

Utworzenie zmiennych reprezentujących połączenie z bazą danych (Connection) oraz obiekt do wykonywania instrukcji SQL (Statement).

1. Łączenie z Bazą Danych:

W bloku try, za pomocą Class.forName, wczytanie sterownika JDBC do obsługi połączenia z bazą danych MySQL.

Nawiązanie połączenia z bazą danych za pomocą DriverManager.getConnection przy użyciu wcześniej zdefiniowanych parametrów (adres, użytkownik, hasło).

Utworzenie obiektu Statement do wykonywania instrukcji SQL na bazie danych.

1. Tworzenie Bazy Danych i Ustawienie Aktywnej Bazy:

Wykonanie instrukcji SQL, która tworzy bazę danych o nazwie "kolokwium", jeśli taka nie istnieje.

Ustawienie aktywnej bazy danych na "kolokwium".

1. Tworzenie tabel i wstawianie danych do tych tabel

Wykonani instrukcji „Create table „ do tworzenia tabel w bazie danych.

Wykonani instrukcji „insert” do wstawiania danych do tabel.

1. Zakończenie połączenia z bazą danych

**Klasa TCPClient**

public class TCPClient {  
  
 public static void main(String[] args) {  
 Scanner sc = new Scanner(System.*in*);  
 try {  
 // Nawiązywanie połączenia z serwerem na porcie 12345  
 Socket socket = new Socket("localhost", 12345);  
 System.*out*.println("Nawiązano połączenie z serwerem: " + socket);  
  
  
 // Strumień do odczytu danych  
 BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()));  
 PrintWriter out = new PrintWriter(socket.getOutputStream(), true);  
  
 // Odczytywanie danych od serwera  
 String receivedData;  
 while ((receivedData = in.readLine()) != null) {  
 System.*out*.println( receivedData);  
 if(receivedData.contains("Twój wynik"))  
 break;  
 System.*out*.println("Twoja odpoiwedź: ");  
 String answer = sc.nextLine();  
 out.println(answer);  
 }  
  
 // Zamknięcie strumienia i gniazda  
 in.close();  
 out.close();  
 socket.close();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
}

**W powyższej klasie można wyróżnić następujące części:**

1. Inicjalizacja:

- Tworzenie obiektu `Scanner` do pobierania danych od użytkownika ze standardowego wejścia.

2. Nawiązywanie Połączenia:

- Tworzenie gniazda (`Socket`) i próba połączenia się z serwerem o adresie "localhost" i porcie 12345.

- Wyświetlanie komunikatu o nawiązaniu połączenia.

3. Inicjalizacja Strumieni:

- Tworzenie strumienia do odczytu danych (`BufferedReader`) i strumienia do wysyłania danych (`PrintWriter`) na podstawie strumienia wyjściowego gniazda.

4. Odczytywanie i Wysyłanie Danych:

- Ustawianie pętli, która odczytuje dane od serwera za pomocą `in.readLine()`.

- Wyświetlanie odczytanych danych.

- Jeśli odczytane dane zawierają frazę "Twój wynik", pętla jest przerywana.

- W przeciwnym razie klient prosi użytkownika o odpowiedź, pobiera ją za pomocą `Scanner` i wysyła do serwera za pomocą `out.println(answer)`.

5. Zamykanie Połączenia:

- Po zakończeniu pętli (na podstawie warunku "Twój wynik"), zamykane są strumienie i gniazdo.

6. Obsługa Wyjątków:

- Obsługa wyjątku `IOException`, który może wystąpić podczas operacji wejścia/wyjścia (np. problem z połączeniem z serwerem).

W skrócie, klient łączy się z serwerem na określonym porcie, odbiera dane od serwera, wysyła odpowiedzi użytkownika i kończy połączenie po otrzymaniu pewnego komunikatu od serwera.

**Klasa TCPServer**

public class TCPServer {  
 static final String *JDBC\_DRIVER* = "com.mysql.cj.jdbc.Driver";  
 static final String *DB\_URL* = "jdbc:mysql://localhost/";  
  
 static final String *USER* = "root";  
 static final String *PASS* = "";  
  
  
static final int *MAX\_CLIENTS* = 250;  
 public static void main(String[] args) {  
 ExecutorService executor = Executors.*newFixedThreadPool*(*MAX\_CLIENTS*);  
 ArrayList<Question> questions = new ArrayList<>();  
 Statement statement = null;  
 try{  
 Class.*forName*(*JDBC\_DRIVER*);  
 Connection connection = DriverManager.*getConnection*(*DB\_URL*, *USER*, *PASS*);  
 String useDatabaseSQL = "USE kolokwium";  
 statement = connection.createStatement();  
 statement.executeUpdate(useDatabaseSQL);  
 String query = "Select \* from question";  
 PreparedStatement preparedStatement = connection.prepareStatement(query);  
 ResultSet resultSet = preparedStatement.executeQuery();  
 while (resultSet.next()){  
 int id = resultSet.getInt("id");  
 String question = resultSet.getString("question");  
 String answerA = resultSet.getString("answerA");  
 String answerB = resultSet.getString("answerB");  
 String answerC = resultSet.getString("answerC");  
 String answerD = resultSet.getString("answerD");  
 String correctAnswer = resultSet.getString("correctAnswer");  
 Question q = new Question(id,question,answerA,answerB,answerC,answerD,correctAnswer);  
 questions.add(q);  
 }  
  
  
 ServerSocket serverSocket = new ServerSocket(12345);  
 System.*out*.println("Serwer działa na porcie 12345");  
  
 while (true){  
 Socket clientSocket = serverSocket.accept();  
 System.*out*.println("Nawiązano połączenie z klientem "+ clientSocket);  
 executor.execute(new ClientHandler(clientSocket,questions));  
 }  
 }catch (IOException | ClassNotFoundException | SQLException e){  
 e.printStackTrace();  
 }finally {  
 executor.shutdown();  
 }  
  
  
 }  
}

**W powyższym kodzie wyróżniamy następujące części:**

1. Ustawienia Bazy Danych:

- Definicja stałych dla sterownika JDBC, adresu bazy danych, nazwy użytkownika i hasła.

2. Maksymalna Liczba Klientów:

- Ustalenie maksymalnej liczby obsługiwanych klientów na 250.

3. Inicjalizacja Serwera:

- Utworzenie puli wątków za pomocą `ExecutorService` o stałej liczbie wątków równą `MAX\_CLIENTS`.

4. Pobieranie Pytań z Bazy Danych:

- Łączenie z bazą danych MySQL, wczytywanie pytań z tabeli "question" z bazy "kolokwium" i tworzenie obiektów klasy `Question` na podstawie wyników zapytania.

5. Utworzenie Gniazda Serwera:

- Utworzenie gniazda serwera na porcie 12345 za pomocą `ServerSocket`.

6. Nieskończona Pętla Obsługi Klientów:

- W nieskończonej pętli serwer oczekuje na połączenie z nowym klientem za pomocą `serverSocket.accept()`.

- Po nawiązaniu połączenia, tworzony jest nowy wątek `ClientHandler` do obsługi komunikacji z klientem, a następnie jest on przekazywany do puli wątków do obsługi.

7. Obsługa Wyjątków:

- Obsługa wyjątków związanych z operacjami wejścia/wyjścia, brakiem sterownika JDBC, błędem połączenia z bazą danych czy problemem z zapytaniem SQL.

- Wydrukowanie informacji o błędzie w przypadku wystąpienia wyjątku.

8. Zamykanie ExecutorService:

- W bloku `finally`, zamykanie puli wątków `ExecutorService` po zakończeniu pracy.

W skrócie, ten serwer obsługuje komunikację z wieloma klientami, pobiera pytania z bazy danych, tworzy wątek obsługujący każde połączenie klienta i umieszcza go w puli wątków do równoczesnej obsługi wielu klientów.

**Klasa ClientHandler**

public class ClientHandler implements Runnable{  
private final Socket clientSocket;  
private Statement statement;  
 static final String *DB\_URL* = "jdbc:mysql://localhost/";  
  
 static final String *USER* = "root";  
 static final String *PASS* = "";  
 ArrayList<Question> questions;  
 ArrayList<String> correctAnswers;  
 String name;  
public ClientHandler(Socket clientSocket, ArrayList<Question> questions){  
 this.clientSocket = clientSocket;  
 statement=null;  
 correctAnswers = new ArrayList<>();  
 this.questions = questions;  
}  
 @Override  
 public void run() {  
 try {  
 BufferedReader in = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));  
 PrintWriter out = new PrintWriter(clientSocket.getOutputStream(),true);  
  
 Connection connection = DriverManager.*getConnection*(*DB\_URL*, *USER*, *PASS*);  
 String useDatabaseSQL = "USE kolokwium";  
 statement = connection.createStatement();  
 statement.executeUpdate(useDatabaseSQL);  
  
 String name = "Podaj imię i nazwisko";  
 out.println(name);  
 name = in.readLine();  
 AtomicInteger result= new AtomicInteger();  
 String finalName = name;  
 questions.forEach(question -> {  
 out.println(question.toString());  
 String clientAnswer = null;  
 try {  
 clientAnswer = in.readLine();  
 if(question.isCorrect(clientAnswer))  
 result.getAndIncrement();  
 String insertAnswer = "Insert into answer (user, questionID, answer) values (?,?,?)";  
 PreparedStatement preparedStatement = connection.prepareStatement(insertAnswer);  
 preparedStatement.setString(1, finalName);  
 preparedStatement.setInt(2, question.getId());  
 preparedStatement.setString(3, clientAnswer);  
 preparedStatement.executeUpdate();  
 } catch (IOException | SQLException e) {  
 e.printStackTrace();  
 }  
  
 });  
  
 out.println("Twój wynik: "+result+"/10");  
 String insert = "INSERT INTO result (user, result) values (?,?)";  
 PreparedStatement preparedStatement = connection.prepareStatement(insert);  
 preparedStatement.setString(1, name);  
 preparedStatement.setInt(2, result.get());  
 preparedStatement.executeUpdate();  
 connection.close();  
 in.close();  
 out.close();  
 clientSocket.close();  
 }catch (IOException | SQLException e){  
 e.printStackTrace();  
 }  
 }  
}

**W powyższej klasie wyróżniamy następujące części:**

1. Zmienne Instancyjne:

- `clientSocket`: Gniazdo klienta, przez które serwer komunikuje się z danym klientem.

- `statement`: Obiekt `Statement` używany do wykonywania operacji na bazie danych.

- `DB\_URL`, `USER`, `PASS`: Parametry do połączenia z bazą danych MySQL.

- `questions`: Lista pytań, które zostaną przekazane klientowi do odpowiedzi.

- `correctAnswers`: Lista przechowująca poprawne odpowiedzi klienta.

- `name`: Zmienna przechowująca imię i nazwisko klienta.

2. Konstruktor:

- Przyjmuje gniazdo klienta i listę pytań jako parametry.

- Inicjalizuje zmienne instancyjne, ustawiając `statement` na null, tworząc pustą listę poprawnych odpowiedzi (`correctAnswers`) i przypisując listę pytań (`questions`).

3. Metoda `run`:

- Nadpisanie metody `run` z interfejsu `Runnable`.

- Ustawienie strumieni wejścia i wyjścia (`BufferedReader` i `PrintWriter`) dla komunikacji z klientem przez gniazdo (`clientSocket`).

4. Nawiązywanie Połączenia z Bazą Danych:

- Tworzenie połączenia z bazą danych MySQL na podstawie parametrów `DB\_URL`, `USER` i `PASS`.

- Ustawienie aktywnej bazy danych na "kolokwium".

5. Przesyłanie Imienia Klienta:

- Wysyłanie komunikatu do klienta z prośbą o podanie imienia i nazwiska.

- Odbieranie imienia od klienta.

6. Przetwarzanie Pytań i Odpowiedzi:

- Iteracja po liście pytań.

- Wysyłanie każdego pytania do klienta.

- Odbieranie odpowiedzi od klienta.

- Sprawdzanie, czy odpowiedź jest poprawna, zwiększanie licznika poprawnych odpowiedzi oraz zapisywanie odpowiedzi do bazy danych.

7. Wysyłanie Wyniku Klientowi:

- Wysyłanie klientowi końcowego wyniku, czyli liczby poprawnych odpowiedzi na 10 pytań.

8. Zapis Wyniku do Bazy Danych:

- Zapisywanie wyniku klienta do tabeli "result" w bazie danych.

9. Zamykanie Połączenia i Obsługa Wyjątków:

- Zamykanie połączenia z bazą danych, strumieni i gniazda klienta.

- Obsługa wyjątków związanych z operacjami wejścia/wyjścia (IOException) oraz operacjami na bazie danych (SQLException). W przypadku wystąpienia błędu, stos jest wydrukowywany (`e.printStackTrace()`).

**Klasa Question**

public class Question {  
 private final int id;  
 private final String question;  
 private final String answerA;  
 private final String answerB;  
 private final String answerC;  
 private final String answerD;  
 private final String correctAnswer;  
  
 public Question(int id,String question, String answerA, String answerB, String answerC, String answerD, String correctAnswer) {  
 this.id = id;  
 this.question = question;  
 this.answerA = answerA;  
 this.answerB = answerB;  
 this.answerC = answerC;  
 this.answerD = answerD;  
 this.correctAnswer = correctAnswer;  
 }  
  
 @Override  
 public String toString() {  
 return question + "\t"+answerA+"\t"+answerB+"\t"+answerC+"\t"+answerD;  
 }  
 public boolean isCorrect(String answer){  
 return answer.equals(correctAnswer);  
 }  
  
  
 public int getId() {  
 return id;  
 }  
}

Kod przedstawia prostą klasę Question, która reprezentuje pytanie z odpowiedziami i poprawną odpowiedzią. Dzięki tym metodą, obiekt Question może być łatwo używany do reprezentacji, wyświetlania i sprawdzania poprawności odpowiedzi na pytanie. W kontekście systemu do przeprowadzania quizu, obiekty tej klasy są używane do przechowywania informacji o pytaniach w bazie danych oraz do przetwarzania odpowiedzi od użytkowników.

**Opis działania aplikacji:**

Tworzenie bazy danych z poziomu aplikacji

Obraz zawierający tekst, zrzut ekranu, Czcionka, design

Opis wygenerowany automatycznie

Brak bazy danych „kolokwium”

Obraz zawierający tekst, zrzut ekranu, czarne, Czcionka

Opis wygenerowany automatycznie

Po uruchomieniu programu CreateDatabaseTable widzimy komunikaty o utworzeniu bazy danych, użyciu jej, utworzeniu tabel oraz o operacji wstawienia danych.

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Czcionka

Opis wygenerowany automatycznie

Tutaj baza danych kolokwium została już utworzona wraz z tabelami.

Obraz zawierający tekst, zrzut ekranu, numer, oprogramowanie

Opis wygenerowany automatycznie

Po utworzeniu bazy danych kolokwium uzupełniona została tylko tabela z pytaniami. Pozostałe tabele są puste.

Obraz zawierający tekst, zrzut ekranu, Czcionka, oprogramowanie

Opis wygenerowany automatycznie

Obraz zawierający tekst, zrzut ekranu, Czcionka, oprogramowanie

Opis wygenerowany automatycznie

Uruchamiamy SerwerTCP i ClientTCP

Obraz zawierający tekst, Czcionka, zrzut ekranu, oprogramowanie

Opis wygenerowany automatycznie

Obraz zawierający tekst, oprogramowanie, Oprogramowanie multimedialne, Oprogramowanie graficzne

Opis wygenerowany automatycznie

Klient pyta o podanie imienia i nazwiska

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Oprogramowanie multimedialne

Opis wygenerowany automatycznie

Następnie wyświetla pytanie i warianty odpowiedzi użytkownik odpowiada na pytanie przesyłając jedną z odpowiedzi.

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

Po podaniu odpowiedzi na wszystkie pytania zwracany jest wynik i program kończy działanie.

Serwer dalej pracuje oczekując na kolejnych klientów

Obraz zawierający tekst, Czcionka, zrzut ekranu, oprogramowanie

Opis wygenerowany automatycznie

Oczywiście można uruchomić kilka klientów i serwer wszystkich przyjmuje do ilości 250.

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Oprogramowanie multimedialne

Opis wygenerowany automatycznie  
W bazie danych w tabeli answer widzimy odpowiedzi użytkowników mamy numer pytania, imię i nazwisko klienta oraz odpowiedź.

Obraz zawierający tekst, Czcionka, numer, zrzut ekranu

Opis wygenerowany automatycznie

Obraz zawierający tekst, Czcionka, numer, zrzut ekranu

Opis wygenerowany automatycznie

W tabeli result mamy imię i nazwisko klienta oraz jego wynik

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

Podsumowując aplikacja działa zgonie z wytycznymi w zadaniu poza jedną kwestią. Nie udało się zaimplementować ograniczenia czasu na wprowadzenie odpowiedzi przez klienta.