UNIVERSITY OF
BIRMINGHAM

Software Workshop 2

—

Real Time Multi-User Quiz System

Benjamin Crispin,
Samuel Farmer,
Deedar Fatima,
Rowan Stringer,
Josh Wainwright

Group Osaka

Supervisor: Joe Gardiner

Date: March 2014

Contents

1 Protocol

The protocol for communicating between the different parts of the system is based around objects. There exists an object class that can be created for any of the several message types that could be needed to transfer information from the server to the client, or from the client to the server. Each of these objects implements the `Serializable` interface, allowing them to be converted to bytes and transfered across the socket connection using object streams.

AnswerResponse object
To respond to a question, the Student selects the desired option. This information is passed to the server using an AnswerResponse object which simply holds the response and the timestamp to indicate when they made the selection.

DisplayQuestion object
In order to signify that the allotted time for the current question has ended, and the next question should be displayed, the server sends a DisplayQuestion object to all of the clients and they should move on to the next question in the Quiz object and change the GUI accordingly.

LoginReply object
Once a loginRequest has been received by the server, a LoginReply will be sent back. This gives the client the information about the requested login, most importantly if the login was successful, as well as the type of user that made the login, Student or Admin. This information is used to display the correct user interface.

LoginRequest object
This is the first object that could be created. It is sent, by the client, to the server and contains the username of the Student that is attempting to login and the `java.lang.String` hash code of the inserted password. Though the security concerns of such a trivial system are non-existent, the password is never stored in plaintext.

Question object
There exist several question objects in each Quiz object. They store the information required to present a Student with a question and the possible answers. Again, there is very little functionality as the question only serves as a wrapper for the question text and the possible answers that the user could respond with.

Quiz object
This is the most important object. It has very little functionality, simply acting as a wrapper to hold and easily transfer several Question objects.

QuizRequest object

Score object
When the quiz has been completed, each of the clients will display the position of it's user relative to the other Students. This object contains the score of all of the users so that each of the clients can work out where they are in the ranking.

StartQuiz object
Once the user has successfully logged into the system, the next major event is the start of the quiz. This is signaled by an Admin user who is connected to the server, this information must then be relayed to each of the connected clients. A

StartQuiz object is sent to each of the clients, who, on receiving it, will display the first question to the user.

2 Client

The client exists to accept messages sent by the server, and present them in an order and a format that the user interface can present to the user, as well as accept the information entered by the user into the user interface and pass it to the server.

2.1 System Design

1. **Login**

   When a client is started, a `QuizClient` object is created and starts the main loop. The initial stages set up the connection with the server and waits for the user to login. When the user enters their information, a `LoginRequest` object is created and pass to the server containing the username and password, to be checked against the contents of the database. The client then waits for a reponse from the server to indicate if the login was successfull or not. This comes in the form of a `LoginReply` object. If this says that the login was unsuccessful, the user is asked to re-input their details, otherwise, the display is changed and the options screen is shown.

2. **Student/Admin**

   There exists separate functionality withing the client depending on if the user is a Student user, i.e. going to be answering questions, or an Admin, i.e. the teacher who starts and moderates the quiz. Distinguishing between these two is done by the server by checking the details associated with that user in the database. The `LoginReply` object contains this information and the client can then display the correct interface depending on the type of user that logged in.

3. **Client Listens from Server**

   From here, the user can select the "Start Quiz" option to start the quiz. This causes the display to change to display the waiting screen and the client waits for information from the server.

   From this point on, the client waits for any object to be sent from the server and acts according to what that object was. The possible objects that the client now expects to be able to distinguish between are:

   - `Quiz`
   - `StartQuiz`
   - `DisplayQuestion`
   - `Score`

   `Quiz`
   : This object contains the information about the quiz itself, the number of questions, their contents and the duration that each question should be displayed for. It should only ever be recieved once by the client, at the start of the session, reducing the transfer of information over the connection.

`StartQuiz`
: Once an Admin has logged in, they have control over the start of the quiz. When they decide to start the quiz, this object is sent to each of the listening clients and so the client will procede to show the first question from the `Quiz` object.

`DisplayQuestion`
: The first question is displayed as soon as the `StartQuiz` object is recieved. After this point in the quiz, the questions are changed when this object is received. The value contained verifies which question is to be displayed.

`Score`
: After each question has been answered, the client can display a leader board showing the score of all the clients that have so far answered the current question along with the current client's position in this list. This object tells the client the relevant information for displaying the scores of the other clients.

4. **Sent by Client**

   There are also a number of objects that the client can create and send to the server at different stages of the quiz:

   - `LoginRequest`
   - `QuizRequest`
   - `AnswerResponse`

`QuizRequest`
: This is sent by the client when an Admin is logged in in order to request a particular quiz from the server. Since the server can hold many quizzes, each with their own set of questions and answers, the Admin has the option to choose which of these to play when they log in.

`AnswerResponse`
: This is the object that tells the server what answer the Student gave. It contains their response, so that it can be logged in the database, and the time it took for the Student to make their selection.

3 Server

The server exists to create a link between the database and the quiz program, as well creating connections with, and processing requests from, clients.

3.1 System Design

1. **Initialising the server**

   When the server starts, a QuizServer object is created. The object creates a connection with the database, which allows the server object to retrieve, and update, information contained in the database. In addition to this, a ServerSocket object is created, which waits to receive connections from QuizClient objects. When a connection from a client is received, a new ClientThread object is created.

2. **Database Connectivity**

   The server interacts with the database through static methods contained in the QuizJDBC class. The class allows the server to establish a connection with the database through a getConnection method, which returns a Connection object.

   A second method, isUser, is called by the server when it recieves a LoginRequest object from a client thread. The method returns a LoginReply object containing the results of the query, which is sent to the client.

   The final method, getQuiz, is called when the server recieves a QuizRequest object from a client, and returns the a Quiz object.

3. **ClientThread Objects**

   When the server establishes a connection with a client, a ClientThread object is created, spawing a thread which allows interactions to occur between the server and client. The server then waits to receive a LoginRequest from the client, and returns a LoginReply object upon recieval. Once the client has logged in, the server distinguishes between student and admin users.

4. **Admin Clients**

   If a connection to an admin has been made, the server waits for a QuizRequest object. Once received, the server updates a static boolean variable, which informs all connected clients that a Quiz object has been created, and the quiz is started.

5. **Student Client**

   Once a quiz has been started, the server sends a Quiz object to all connected clients, and then waits to receive an AnswerReponse from each of the student client threads. The AnswerReposnse objects contain the clients response to a question, and the time that it took for them to anwer it. The server than calculates the clients score, and the results are stored in ArrayList, which contains the results of all connected student clients. This ArrayList of scores is then sent to the connected clients at the end of each question.

4 Database

5 Team Organisation

The group consists of 5 students. Since there were a numbver of distint sections to the project, the different components were distributed among the members of the team with one person in a position of responsibility for that component and another to help. The allocations were as follows.

Component	Responsible	Assiting
GUI	Benjamin Crispin	Deedar Fatima
Client	Josh Wainwright	Rowan Stringer
Server	Rowan Stringer	Benjamin Crispin
Database	Deedar Fatima	Sam Farmer
Protocol	Sam Farmer	Josh Wainwright

These roles were followed closly during the inital stages of development and through the first round of testing. As the program became more complete and bug fixes were required, the roles were shared more evenly through the team. This has the advantage that all members have a full understanding of all aspects of the project having worked on all parts at some time.

References

[1] Elliotte Harold. *Java Network Programming.* O'Reilly Media Inc., 2013.

[2] Harvey Deitel and Paul Deitel. *Java: How to Program.* Pearson, 2002.

[3] Scott Oaks and Henry Wong. *Java Threads.* O'Reilly Media Inc., 1999.

[4] Xingchen Chu Rajkumar Buyya, S Thamarai Selvi. *Object-Oriented Programming with Java.* Tata McGraw Hill, 2009.

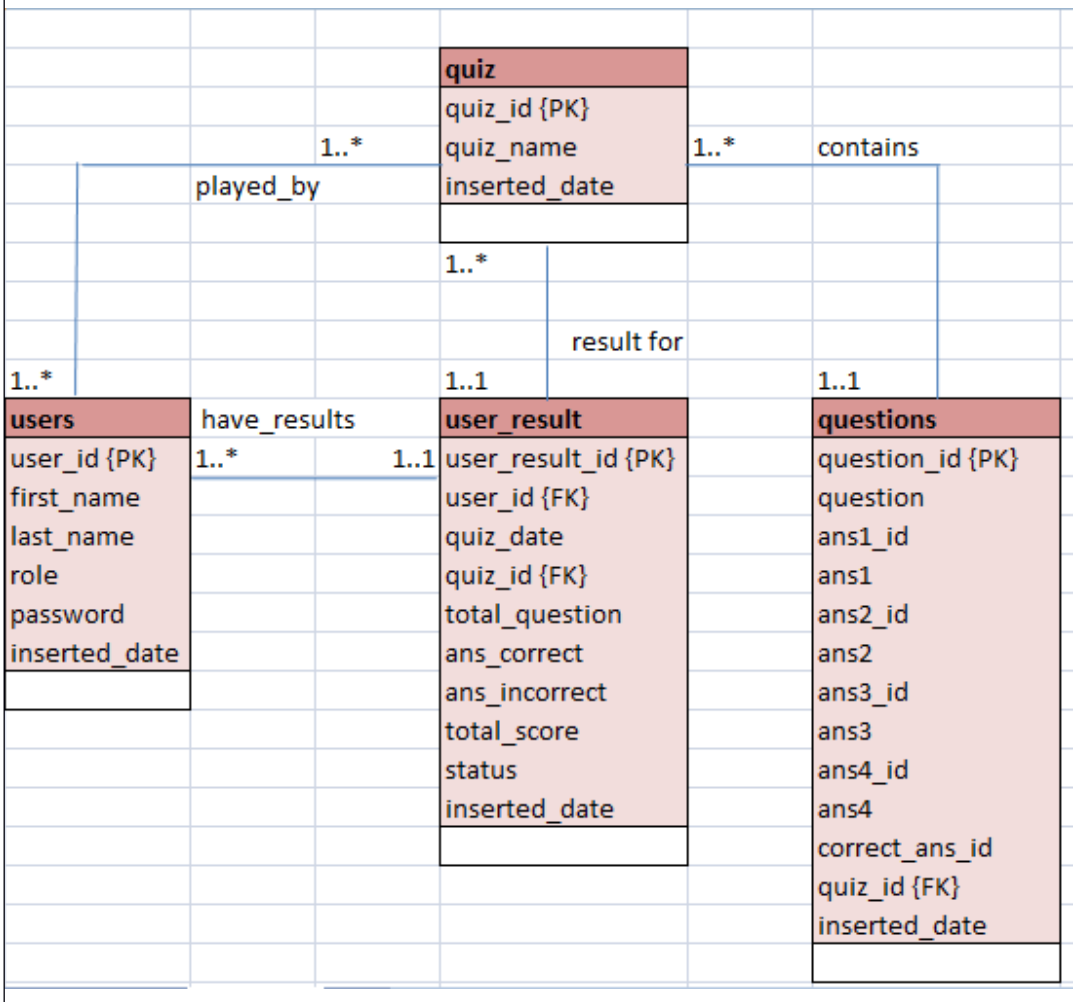
[5] Kenneth L. Calvert and Michael J. Donahoo. *TCP/IP Sockets in Java.* Morgan Kauffman, 2001.

Osaka
Database Design Specification

Server: The server used for the project Osaka was dbteach2.

Database: The database created for this project is called **osakagp.** The owner of the database is dxf321 (Deedar). The other team members will be given access to the database shortly.

Entity-Relationship Diagram:



Tables:

The tables created in osakagp DB are as follows.

USERS

Table name	users			
Description	The user login details are stored in the users table. The user details are inserted into this table when a new user or administrator registers. The login credentials entered by the users are validated and the users are allowed to login only if the entered credentials exist in the users table.			
Attribute	**Description**	**Type**	**Nullability**	**Example of values**
user_id	Unique ID of an admin/student	BIG INT	NOT NULL	Between 1 and 9223372036854775807
first_name	First name of admin/student	VARCHAR (20)	NULL	Mary
last_name	Last name of admin/student	VARCHAR (20)	NULL	Ande
role	Role of user	VARCHAR (20)	NULL	admin or student
password	Password entered by admin/student to access the tool	VARCHAR (10)	NULL	
inserted_date	Timestamp of the transaction	TIMESTAMP	NOT NULL	DEFAULT is the current timestamp.
Primary Key	user_id			
Foreign Key				
SQL code	SELECT * FROM users;			

QUIZ

Table name	quiz

Description	The quiz topics are stored in this table. The quiz could be on the following topics - Politics, Sports, History, Geography, Java, Database, etc. The admin chooses the quiz topic from quiz table and fetches the topic-related questions from the questions table.			
Attribute	**Description**	**Type**	**Nullability**	**Example of values**
quiz_id	Unique ID of quiz	BIG INT	NOT NULL	Between 1 and 9223372036854775807
quiz_name	Topic of quiz	VARCHAR (40)	NULL	Politics, Sports
inserted_date	Timestamp of the transaction	TIMESTAMP	NOT NULL	DEFAULT is the current timestamp.
Primary Key	quiz_id			
Foreign Key				
SQL code	SELECT * FROM quiz;			

QUESTIONS

Table name	questions			
Description	The questions table contains the questions which are answered in quiz. Question and possible answers are stored as rows in the questions table. The table also contains a separate column for quiz ID. The admin uses the quiz ID to get the questions for the chosen quiz topic.			
Attribute	**Description**	**Type**	**Nullability**	**Example of values**
question_id	Unique ID of question	BIG INT	NOT NULL	Between 1 and 9223372036854775807
question	The question to be answered by students	VARCHAR (100)	NULL	In which country is the Albert canal?
ans1_id	ID of first possible answer	INT	NOT NULL	DEFAULT is 1
ans1	First possible answer	VARCHAR (40)	NULL	Spain
ans2_id	ID of second possible answer	INT	NOT NULL	DEFAULT is 2
ans2	Second possible answer	VARCHAR (40)	NULL	Belgium
ans3_id	ID of third	INT	NOT NULL	DEFAULT is 3

	possible answer			
ans3	Third possible answer	VARCHAR (40)	NULL	Canada
ans4_id	ID of fourth possible answer	INT	NOT NULL	DEFAULT is 4
ans4	Fourth possible answer	VARCHAR (40)	NULL	Portugal
correct_ans_id	The ID of correct answer	INT	NOT NULL	2
quiz_id	The ID of quiz	BIG INT	NOT NULL	4
inserted_date	Timestamp of the transaction	TIMESTAMP	NOT NULL	DEFAULT is the current timestamp.
Primary Key	question_id			
Foreign Key	quiz_id			
SQL code	SELECT * FROM questions;			

USER_RESULT

Table name	user_result			
Description	This table contains the quiz results for all the students. It is loaded with quiz result once the quiz is completed. The user can see the result by quiz date, quiz topic, score and quiz status.			
Attribute	**Description**	**Type**	**Nullability**	**Example of values**
user_result_id	Unique ID for the row	BIG INT	NOT NULL	Between 1 and 9223372036854775807
user_id	ID of user	BIG INT	NOT NULL	Between 1 and 9223372036854775807
quiz_date	Date on which quiz is played	TIMESTAMP	NOT NULL	DEFAULT is Current Timestamp
quiz_id	ID of quiz played by the student	BIG INT	NOT NULL	Between 1 and 9223372036854775807
total_question	Number of questions displayed in a quiz	INT	NULL	10

ans_correct	Number of questions answered correctly before any other student	INT	NULL	Between 0 and 10
ans_incorrect	Number of questions answered incorrectly	INT	NULL	Between 0 and 10
total_score	Number of questions answered correctly before any other student	INT	NULL	Between 0 and 10
status	If a student won or lost the quiz	VARCHAR(10)	NULL	WON or LOST
inserted_date	Timestamp of the transaction	TIMESTAMP	NOT NULL	DEFAULT is the current timestamp.
Primary Key	user_result_id			
Foreign Key	user_id, quiz_id			
SQL code	SELECT * FROM user_result;			

SQL statements for database and tables creation:

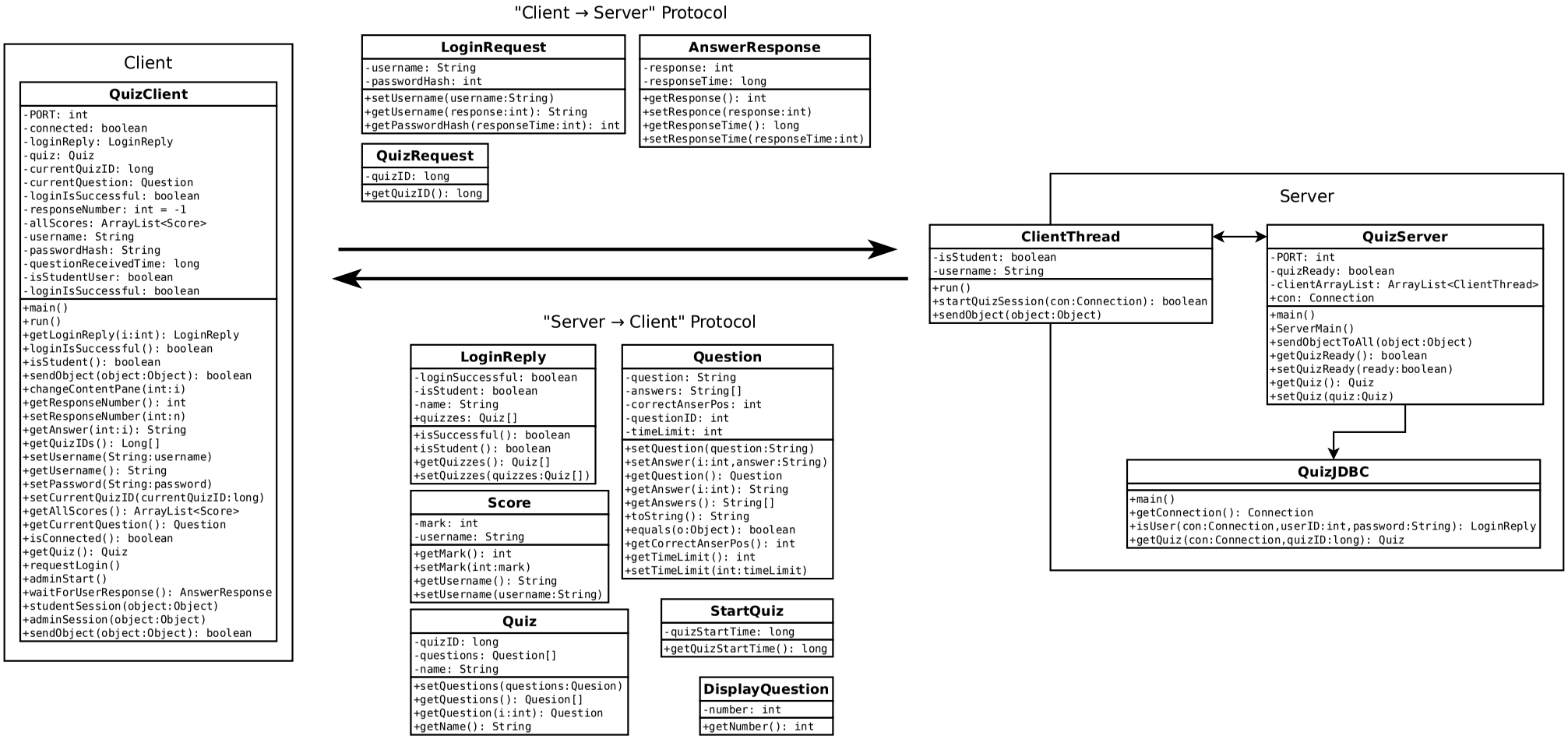
```
CREATE DATABASE osakadb OWNER dxf321;

CREATE TABLE users
(
user_id  BIGSERIAL PRIMARY KEY,
first_name VARCHAR (20),
last_name VARCHAR (20),
role VARCHAR (20),
password VARCHAR (10),
inserted_date timestamp default current_timestamp
);


CREATE TABLE quiz
 (
quiz_id  BIGSERIAL PRIMARY KEY,
quiz_name VARCHAR (40),
 inserted_date timestamp default current_timestamp
);
```

```
CREATE TABLE questions
 (
question_id  BIGSERIAL PRIMARY KEY,
question VARCHAR (100),
ans1_id INT default 1,
ans1 VARCHAR (40),
ans2_id INT default 2,
ans2 VARCHAR (40),
ans3_id INT default 3,
ans3 VARCHAR (40),
ans4_id INT default 4,
 ans4 VARCHAR (40),
correct_ans_id INT NOT NULL,
quiz_id BIGINT REFERENCES quiz (quiz_id),
inserted_date timestamp default current_timestamp
);


CREATE TABLE user_result (
user_result_id  BIGSERIAL PRIMARY KEY,
 user_id  BIGINT  REFERENCES users (user_id),
quiz_date timestamp default current_timestamp,
quiz_id BIGINT  REFERENCES quiz(quiz_id),
total_question INT,
ans_correct INT,
ans_incorrect INT,
total_score INT,
status VARCHAR (10),
 inserted_date timestamp default current_timestamp
);
```

Tables Load:

```
INSERT INTO users (first_name, last_name, role) VALUES ('Mary', 'Ande', 'student');
 INSERT INTO users (first_name, last_name, role) VALUES ('Andrew', 'Baker',
'student');
 INSERT INTO users (first_name, last_name, role) VALUES ('Katie', 'Bowyer',
'student');
 INSERT INTO users (first_name, last_name, role) VALUES ('Katherine', 'Brittain',
'student');
 INSERT INTO users (first_name, last_name, role) VALUES ('Thomas', 'Chapman',
'student');
 INSERT INTO users (first_name, last_name, role) VALUES ('Andrew', 'Green',
'student');
INSERT INTO users (first_name, last_name, role) VALUES ('Matthew', 'Harris',
'student');
INSERT INTO users (first_name, last_name, role) VALUES ('Ella', 'Hibbert', 'student');
```

```
INSERT INTO users (first_name, last_name, role) VALUES ('Daniel', 'Hirst', 'student');
INSERT INTO users (first_name, last_name, role) VALUES ('Antony', 'Judd', 'student');
INSERT INTO users (first_name, last_name, role) VALUES ('George', 'Kiff', 'admin');
INSERT INTO users (first_name, last_name, role) VALUES ('Joseph', 'May', 'admin');

INSERT INTO quiz (quiz_name) VALUES ('History');
INSERT INTO quiz (quiz_name) VALUES ('Politics');
INSERT INTO quiz (quiz_name) VALUES ('Sports');
INSERT INTO quiz (quiz_name) VALUES ('Java');
INSERT INTO quiz (quiz_name) VALUES ('Database');
INSERT INTO quiz (quiz_name) VALUES ('Geography');


INSERT INTO questions (question, ans1, ans2, ans3, ans4, correct_ans_id, quiz_id)
VALUES ('In which country is the Albert canal?', 'Spain', ‘Belgium’, ‘Canada’,
‘Portugal’, 2, 4);

INSERT INTO questions (question, ans1, ans2, ans3, ans4, correct_ans_id, quiz_id)
VALUES ('Which is the only US state named after an English county?',  'Kentucky',
'North Dakota',  'Vermont',  'New Hampshire', 4, 6);

INSERT INTO questions (question, ans1, ans2, ans3, ans4, correct_ans_id, quiz_id)
VALUES ('Which British cyclist won the 100th edition of the Tour de France?',  'Chris
Froome',  'Lizzie Armitstead',  'Matt Crampton',  'Kyle Evans', 1, 3);

INSERT INTO questions (question, ans1, ans2, ans3, ans4, correct_ans_id, quiz_id)
VALUES ('How many players are there in a basketball team?', '11', '14', '5', 6, 3, 3);



INSERT INTO user_result (user_id,quiz_id,total_question, ans_correct, ans_incorrect,
total_score, status) VALUES ( 1, 2, 10, 8,  2, 8, 'WON');
INSERT INTO user_result (user_id,quiz_id,total_question, ans_correct, ans_incorrect,
total_score, status) VALUES ( 1, 4, 10, 10,  0, 10, 'WON');
INSERT INTO user_result (user_id,quiz_id,total_question, ans_correct, ans_incorrect,
total_score, status) VALUES ( 6, 2, 10, 10,  0, 10, 'WON');
INSERT INTO user_result (user_id,quiz_id,total_question, ans_correct, ans_incorrect,
total_score, status) VALUES ( 6, 6, 10, 9,  1, 9, 'WON');
INSERT INTO user_result (user_id,quiz_id,total_question, ans_correct, ans_incorrect,
total_score, status) VALUES ( 7, 3, 10, 5,  5, 5, 'LOST');
```


Client
QuizClient
-PORT: int
-connected: boolean
-loginReply: LoginReply
-quiz: Quiz
-currentQuizID: long
-currentQuestion: Question
-loginIsSuccessful: boolean
-responseNumber: int = -1
-allScores: ArrayList<Score>
-username: String
-passwordHash: String
-questionReceivedTime: long
-isStudentUser: boolean
-loginIsSuccessful: boolean
+main()
+run()
+getLoginReply(i:int): LoginReply
+loginIsSuccessful(): boolean
+isStudent(): boolean
+sendObject(object:Object): boolean
+changeContentPane(int:i)
+getResponseNumber(): int
+setResponseNumber(int:n)
+getAnswer(int:i): String
+getQuizIDs(): Long[]
+setUsername(String:username)
+getUsername(): String
+setPassword(String:password)
+setCurrentQuizID(currentQuizID:long)
+getAllScores(): ArrayList<Score>
+getCurrentQuestion(): Question
+isConnected(): boolean
+getQuiz(): Quiz
+requestLogin()
+adminStart()
+waitForUserResponse(): AnswerResponse
+studentSession(object:Object)
+adminSession(object:Object)
+sendObject(object:Object): boolean
"Client → Server" Protocol
LoginRequest
-username: String
-passwordHash: int
+setUsername(username:String)
+getUsername(response:int): String
+getPasswordHash(responseTime:int): int
AnswerResponse
-response: int
-responseTime: long
+getResponse(): int
+setResponce(response:int)
+getResponseTime(): long
+setResponseTime(responseTime:int)
QuizRequest
-quizID: long
+getQuizID(): long
Server
ClientThread
-isStudent: boolean
-username: String
+run()
+startQuizSession(con:Connection): boolean
+sendObject(object:Object)
QuizServer
-PORT: int
-quizReady: boolean
-clientArrayList: ArrayList<ClientThread>
+con: Connection
+main()
+ServerMain()
+sendObjectToAll(object:Object)
+getQuizReady(): boolean
+setQuizReady(ready:boolean)
+getQuiz(): Quiz
+setQuiz(quiz:Quiz)
QuizJDBC
+main()
+getConnection(): Connection
+isUser(con:Connection,userID:int,password:String): LoginReply
+getQuiz(con:Connection,quizID:long): Quiz
"Server → Client" Protocol
LoginReply
-loginSuccessful: boolean
-isStudent: boolean
-name: String
+quizzes: Quiz[]
+isSuccessful(): boolean
+isStudent(): boolean
+getQuizzes(): Quiz[]
+setQuizzes(quizzes:Quiz[])
Question
-question: String
-answers: String[]
-correctAnserPos: int
-questionID: int
-timeLimit: int
+setQuestion(question:String)
+setAnswer(i:int,answer:String)
+getQuestion(): Question
+getAnswer(i:int): String
+getAnswers(): String[]
+toString(): String
+equals(o:Object): boolean
+getCorrectAnserPos(): int
+getTimeLimit(): int
+setTimeLimit(int:timeLimit)
Score
-mark: int
-username: String
+getMark(): int
+setMark(int:mark)
+getUsername(): String
+setUsername(username:String)
StartQuiz
-quizStartTime: long
+getQuizStartTime(): long
Quiz
-quizID: long
-questions: Question[]
-name: String
+setQuestions(questions:Quesion)
+getQuestions(): Quesion[]
+getQuestion(i:int): Question
+getName(): String
DisplayQuestion
-number: int
+getNumber(): int