SCIT

School of Computing and Information Technology

Spring 2015

CSCI213/MCS9213/CSCI813 — Java Programming and Applications

Assignment 4 (10 marks)

Due Time and Date:

23:59:59 on Sunday, 18 October

Objectives

In this assignment, you will design and implement client/server applications for a Java quiz system using Java sockets and threads. The system consists of a GUI client program and a console server program. The GUI client program has a user interface that allows users to connect to the server and answer Java quiz. The console server program loads Java quiz questions on the server and send questions to multiple clients using Java threads.

Upon completion of this assignment, you should be able to design and implement networked applications; design and implement a simple communication protocol; and design and implement multi-threaded applications.

Tasks and Requirements

In this assignment, you will design and implement client/server applications for a Java quiz system using Java sockets and threads according to the user requirements.

Your server console application program should be called `JavaQuizServer.java` and your client GUI application program `JavaQuizClient.java`. You should create classes to represent questions and students that support serialised objects. (You may adapt your code from your Assignment 1. This is the last chance to make your question classes and the code to load questions from a file to work if they did work properly in your Assignment 1.) Your classes should be arranged as required.

The question library file format is given in the Appendix A.

General Requirements

- You should observe the common principles of OO programming when you design your Java classes, which include abstraction, encapsulation, inheritance and polymorphism.
- You should create your classes by making use of features of the Java classes
- You should make proper documentation and implementation comments in your codes where they are necessary.
- Logical structures and statements are properly used for specific purposes.
- Your code should comply with the major requirements of Java Code Conventions.

Structures of the Client/server Packages

The Java classes created in this assignment will have the following package structure. You may add more supporting classes in appropriate packages if necessary.

- **Server packages:**

```
/WorkingDirectory/Server
      JavaQuizServer.java
      questions.xml
      (Other configuration files if any)
      /au/edu/uow
            /Networking
                  JavaClientHandler.java
                  (Classes related to networking)
            /UserInterface
                  UserInterface.java
                  (Other interface related classes)
            /QuestionLibrary
                  Question.java
                  MultipleChoiceQuestion.java
                  TrueAndFalseQuestion.java
                  (Other related classes to load questions)
```

Your main server application will be created in `JavaQuizServer.java` file, which will load the Java questions from the questions file and serve concurrent clients. The server should send different quizzes to different clients. The client handling task class is implemented in `JavaClientHandler.java` file.

- **Client packages:**

```
/WorkingDirectory/Client
      JavaQuizClient.java
      (Other configuration files if any)
      /au/edu/uow
            /Networking
                  (Classes related to networking)
            /ClientGUI
                  QuizClientGUIFrame.java
                  Student.java
                  (Other GUI related classes)
            /QuestionLibrary
                  Question.class
                  MultipleChoiceQuestion.class
                  TrueAndFalseQuestion.class
```

Your main client application will be created in `JavaQuizClient.java` file, which will load your interface from `QuizClientFrame.java`. (Hint: develop your `QuestionLibrary` package in the server directory and copy only the compiled class files to the client directory.)

Client/server Communication Protocol:

The following methods are defined:

- **REGISTER method:**

  Request: REGISTER *username*
  The client sends the REGISTER message with a one word string *username* of the client user name to the server to register the user.
  Response: OK
  Upon receiving the REGISTER message, the server sends back an acknowledgement message, OK, to the requesting client.

- **GET QUESTION method:**

  Request: GET QUESTION
  The client sends the GET QUESTION message to the server to request a Java quiz question.
  Response: (*a question object*)
  Upon receiving the GET QUESTION message, the server sends back a serialised question object to the requesting client.
  Do not send all details of a question in strings but send only an object of a multiple choice question or true-and-false question class to the client using object serialisation.

- **BYE method:**

  Request: BYE
  The client sends the BYE message to the server to end a session.
  Response: (*null*)
  Upon receiving the BYE message, the server closes the requesting client connection. The server does not need to send anything back to the requesting client.

Server Requirements

The server listens to a socket port for client connection requests. The server program can receive a command line argument for the listening port number. For example,

```
java JavaQuizServer 40213
```

If the server program starts without any argument, it listens to a default port.

The server should display information about the listening port and status of connected users in the console. For example:

```
JavaQuizServer listening at: 40213
Peter registered
Donna registered
Donna disconnected
Peter disconnected
```

Client GUI Requirements

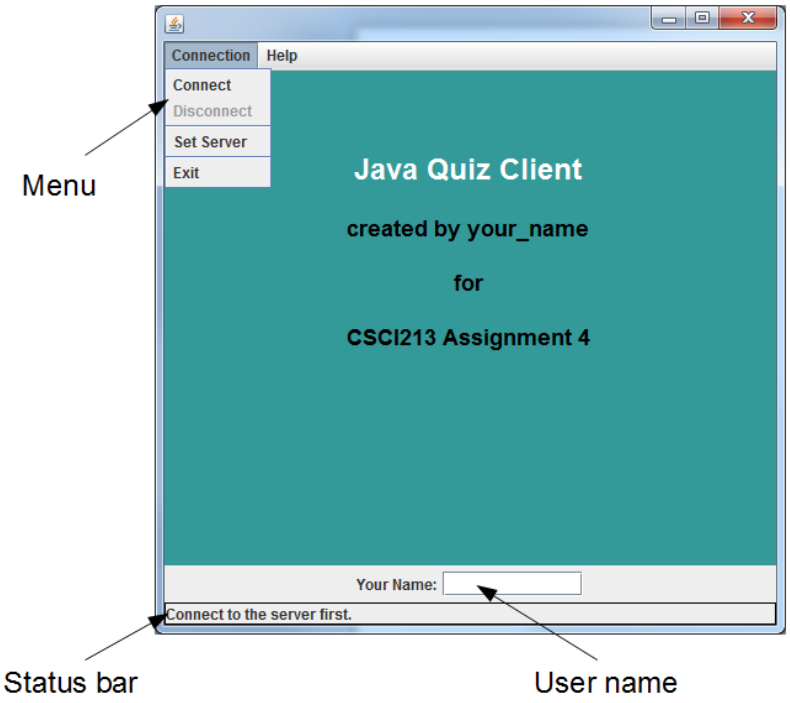
The client program can receive a command line argument for the server name and/or port number. For example,

`java JavaQuizClient localhost` (or `localhost:40213`)

If the client program starts without any argument, it connects to local host and a default port.

- Before connection

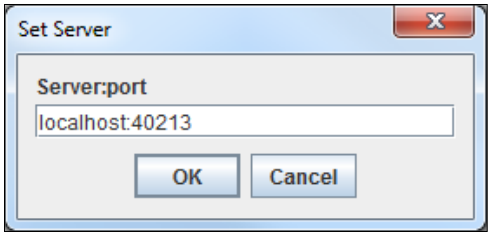
After starting the client program, you should have the following interface. The status bar should show the current status of the program. Inapplicable items in a state should be disabled.



The Connection menu has menu items shown in the figure.

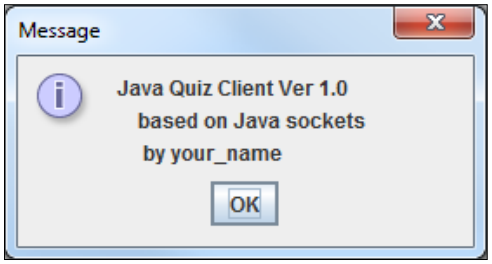
– The "Connect" item connects the client to the server. If the user attempts to connect, an information dialog should pop up as shown.

- The "Disconnect" item disconnects the client from the server;
- The "Set Server" item pops up an input dialog for the user to specify the server name and port number as shown. The default server name should be "localhost:40213".



- The "Exit" item terminates the program. You need make sure to terminate the program before properly disconnecting it from the server if it is connected.

The Help menu has one menu item called "About" that pops up an information dialog as shown:



If the user closes the window at any time, you need make sure the client is disconnected from the server before the window closes. This can be done by setting the proper `DefaultCloseOperation` for the `JFrame` and processing the window event.

- After connection

After the client has connected to the server and registered, you should have the following interface displaying question details and allow the user to select an answer as shown in the figure Question Screen.

The status bar should always show the status of the program.

When the user selects an answer, the next question will be presented. The client should request 5 questions from the server.

When the last question is presented, the `Next` button should be changed to "`Get Score`" button.

Once the user pressed the "`Get Score`" button, The connection should be terminated and the final score should be presented as shown in the figure Final Score Screen.

The user name and scores should be maintained by the `Student` object in the client program.

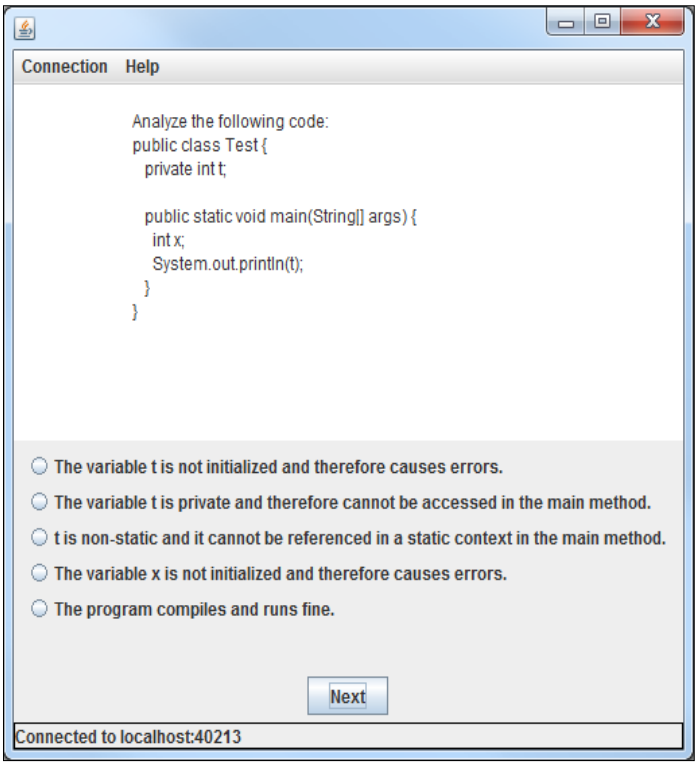


Figure: Question Screen

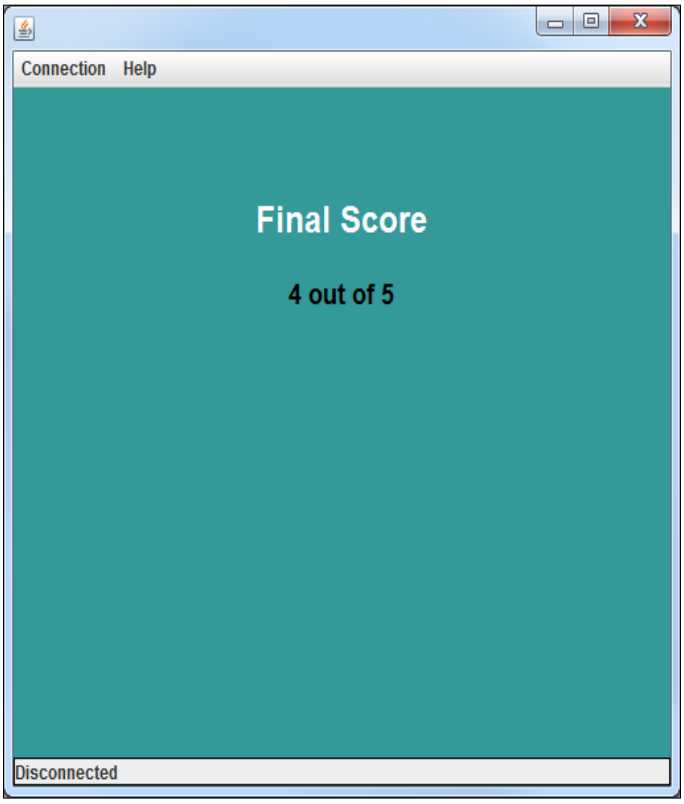


Figure: Final Score Screen

Extension Task: Display all concurrent users at the client
(***Optional for CSCI213/MCS9213 students; compulsory for CSCI813 students***)

In this task, you add a menu called `Users` with a menu item called "`Display All Users`". When the menu item is selected, a window is popped up to display all concurrent users. Implement this function using the MVC design pattern.

The displayed user list must show all registered concurrent users at the server. The newly connected client users must be added to the display and the disconnected client users removed from the display, automatically.

Submission

- The **`@author`** line in all your source code files should include **your Subject Code**, **your name**, **student ID**, and **Unix login name**.
- IMPORTANT: Any submission that cannot be decompressed may receive zero mark. Any submission that resulted in compiling errors may have 50% marks deducted.
- You should first zip your files including subdirectory paths into a file called **`a4.zip`**. For this assignment, you should submit all your programs that are required to run your program **excluding** files that may be generated by the IDE you might have used. Failing to do so may result in deduction of marks.
- Make sure that it contains both client and server subdirectories. (It is a good idea to do your assignment in a working directory that contains only client and server subdirectories so that the zip program would not include other irrelevant directories or files.)
- Submit the file from your University Unix account on Banshee (`banshee.uow.edu.au`) as follows:

  ```
  turnin -c csci213 -a a4 a4.zip
  ```

  or, if you submit it after the deadline:

  ```
  turnin -c csci213 -a a4-late a4.zip
  ```

- Use the following command to check the submitted file:

  ```
  turnout -c csci213 -a a4
  ```

  or, if you submit it after the deadline:

  ```
  turnout -c csci213 -a a4-late
  ```

Marking Scheme

Mark to 0.5 divisions.

For CSCI213/MCS9213 students

General requirements	-0.5 ~ -1.0 mark for each error
Client GUI implementation	2
Question related functionality	2
Networking functionality	3
Server program	1
Threads	2
Extension Task	3

Satisfactory completion of the Extension Task will be rewarded extra marks that will be added to the final mark for assignments and labs. The maximal final marks for all assignments and labs remain 50.

For CSCI813 students

General requirements	-0.5 ~ -1.0 mark for each error
Client GUI implementation	1.5
Question related functionality	1.5
Networking functionality	2
Server program	1
Threads	2
Extension Task	2

Extension Task is compulsory for CSCI813 students.

Appendix A: Question Library File Format

All questions and their sections in the question library document are marked with start-tags and matching end-tags, a format conforming to XML 1.0 Specification.

A XML document is essentially a marked up string of characters, which can be read as a text file.

The root tag for the document is `JavaQuestions`. The multiple choice questions, tagged as `MQuestion`, have three sections that are `question`, `choices` and `answer` while the true and false questions, tagged as `TFQuestion`, have two sections that are `question` and `answer`. Sections may appear in any order and each section may have different numbers of lines or choices.

You should not change the format. A sample question file, called `questions.xml`, is provided for your conformance test.

A segment of an example question file is shown below.

```
<?xml version="1.0"?>
<JavaQuestions>
<MQuestion>
<question>
Which of the following statements is correct?
</question>
<choices>
A class in Java can extend one or more classes
A class in Java can extend one class and implement one or more interfaces
A class in Java can extend one interface and implement one or two classes
A class in Java can extend one class and implement one interface
</choices>
<answer>
2
</answer>
</MQuestion>
<TFQuestion>
<question>
Superclasses can contain abstract methods.
</question>
<answer>
true
</answer>
</TFQuestion>

. . .

</JavaQuestions>
```