Queen Mary University of London

School of Electronic Engineering and Computer Science

Mini-Project Coursework:
A GUI for a Multiple Choice Question (MCQ) program

Module code: **EBU4201**
Module title: **Introductory Java Programming**
Hand-out date: **30th April 2019**
Hand-in date: **22nd May 2019**
Marks available: **50**
Feedback: **Individual marking sheet including feedback comments and a mark out of 50.**

Introduction: You are asked to write a graphical user interface (GUI) for a multiple choice question (MCQ) program. A series of questions will be presented to the user; with each question, four possible answers are also presented. One of the answers will be correct, and the other three answers will be incorrect.

TASK 1 – Basic Interface [28 marks]

The basic interface must be divided into 3 parts: a top, middle, and bottom. **Figure 1** shows the expected look of the interface:

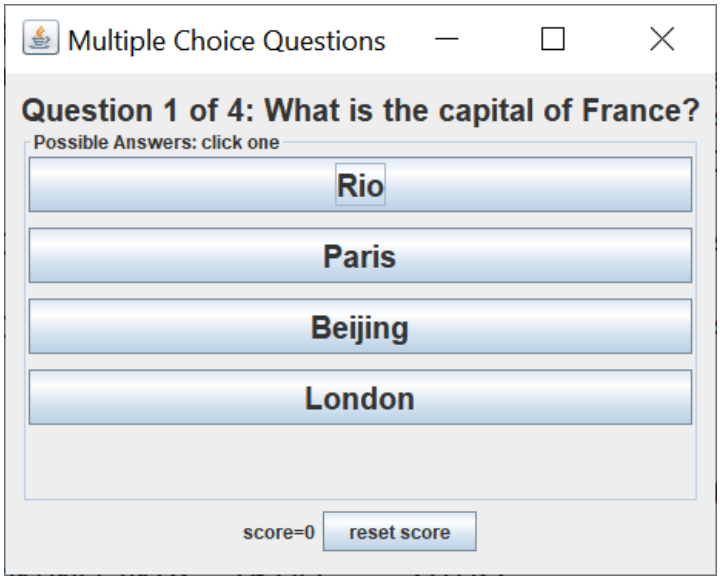

Figure 1

A question will be displayed at the top of the interface.

In the middle of the interface, the four choices for the answer are displayed. For each question, there will be one and only one correct answer. Clicking on the correct answer will increase the user's score by 1 point.

At the bottom of the interface, the number of correct answers given so far will be displayed. Next to the score will be a "reset" button which restarts the multiple choice questions, setting the score to zero.

On the interface you might like to also display the total number of questions in the quiz.

Note: You are provided with a text file (called **sampleQandA.txt**) containing some sample questions and answers (including an indication of which answer is correct for each question). You must decide how best to use this data in your application[1].

TASK 2 – Interface with Multiple Attempts [5 marks]

Typically with multiple choice questions, you only have one attempt at each question.

In this extension of the original application, the user taking the multiple choice test is allowed multiple attempts.

On the first attempt, if they are correct they score 2 points. If they are incorrect, they have another attempt, which if correct will score 1 point.

A way to allow users to have up to 2 attempts to answer question needs to be implemented. This could be from the command line, or via the GUI.

The extended application must still provide the functionality developed for **TASK 1**.

TASK 3 – Choice of EASY, MEDIUM and DIFFICULT questions [5 marks]

In this extension of the application, a set of radio buttons allows the user to choose the level of the questions.

There should be three radio buttons allowing the options ***easy***, ***medium***, and ***difficult***. ***Easy*** questions score 1 point, ***medium*** questions score 2 points, and ***difficult*** questions score 3 points.

The extended application must still provide the functionality developed for **TASK 1** and **TASK 2**.

Documentation [12 marks]

Your submitted work must include:

a. Automatically generated Javadoc files.

b. Comments (both internal and Javadocs) in your code.

c. User Manual; this should be a document[2] with no more than two A4 pages which must include instructions on how to run the program (i.e. both how to start it, and how to use it).

Note: All documentation files must be placed in a directory called **Documentation**.

---

[1] For example, a possible (though not the only one) solution could be for the list of questions and answers to be a variable in your application.

[2] Accepted document formats include: **.txt**, **.docx** and **.pdf**.

Marking Scheme

Marks will be awarded for the following:

1. a clearly laid out interface,
2. correctly functioning code and,
3. clearly structured code including comments and sensible variable, class, and method names.

Note: The main program file must be called **MultipleChoiceGui.java**, and it must compile and run from the command line; otherwise, it will not be possible to give marks for any implemented functionality.

Submission Instructions

You must zip all the following files:

1. The **Code** directory, including all **.java** files produced[3].
2. The **Files** directory, including any text files used to communicate with the application.
3. The **Documentation** directory, including all *Javadoc* comments and the **User Manual**.

Notes:

- You must name your .zip file **201721xxxx.zip**, where **201721xxxx** is your BUPT student number.
- You must submit your .zip file to the EBU4201 course area in QMplus[4], under the assignment activity **Mini-Project (submission)**.

IMPORTANT: This is an individual piece of assessed coursework; therefore, students must not work in groups and must not share code solutions.

[3] It will not be possible to mark your work, if you only provide **.class** files.

[4] Any work sent via email will be ignored and not marked.