FLOW SOLID - PART 2

Copenhagen Business Academy

Computer Science AP

1st semester

PART 2: GENERAL QUIZZ MACHINE

Write an extended version of your program from part 1.

A) GREEN FUNCIONALITY

The user must be able to practice the ability to remember another set of related words (or short pieces of text) .

Ex: Code templates in NetBeans: ”sout” means ”System.out.println()”.
Ex: Geography: ”Capital of France” is ”Paris”.

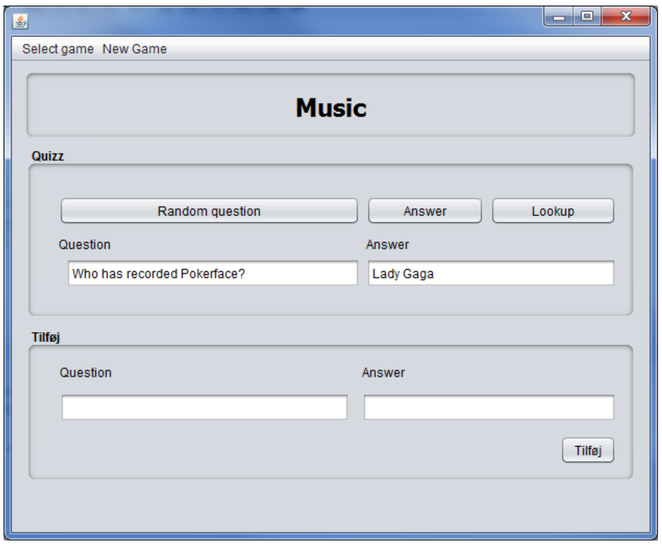
The user must be able to select one from a number of possible ”games” (types of tasks) from a list. The program then ”adapts” to the selected game. This adaptation includes:

- Lead text on the user interface
- The actual word pairs (question/answer)
- Which actual file is used to store the word pairs on

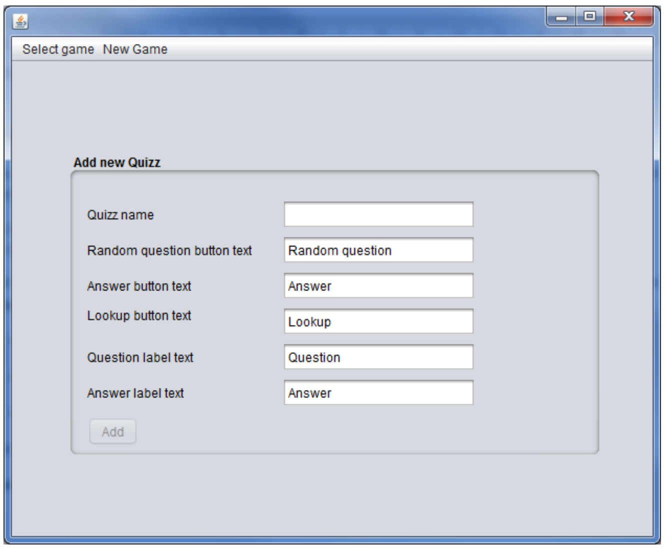
The possibility of entering new word pairs (relevant for the selected type of task) must be preserved.

Load and save from file should be done only on explicit request from the user.

Example GUI



Question: Who has recorded Pokerface?
Answer: Lady Gaga



Quizz name	
Random question button text	Random question
Answer button text	Answer
Lookup button text	Lookup
Question label text	Question
Answer label text	Answer

B) Yellow functionality

The user must be able to create a new quiz (or game).

The user must be able to add new word pairs to an existing quiz (or game).

C) Red functionality

Create a small game that people can play in your program.

From 2 to unlimited players should be created.

The game must keep track of points/score and name for each player.

Make your own interface that extends the *QuizzControlInterface* so that

Extend the *QuizzControlInterface* with your own interface and implement it in your own controller.

Constraints:

1. The Control class must implement a new predefined Java Interface (*QuizzControlInterface.java* - see separate file).
2. Comments in the code. You must have appropriate comments to explain your code and make it quicker to understand. Each class must also have a Javadoc comment explaining the responsibility of the class.
3. You must implement a <u>set of relevant test cases</u> and a table for all the test cases, for the new methods in the Control class in JUnit.
4. Each game file must have a fixed format – as follows

```
GameName
Question1,Answer1
Question2,Answer2
….
```

5. By adhering to a fixed file format you should be able to swap games across groups. Ask another group for their game and show that you can actually do the swap. **Exception**: RED programmers need not do this (because they extended the interface with their own).
6. You must maintain a rough design class diagram (DCD) showing the overall structure of the program.

HAND IN (PART 2):

Format: One (1) zip'ed file containing:

1) The program - as an exported (zip'ed) NetBeans project – complete with comments
2) Design Class Diagram
3) Short note that describe:
   I. Observed benefits or drawbacks of specific details of your design
   II. Short instruction: How to get started to use the program
   III. your proposals - if any – for an improved interface (*QuizzControlInterface.java*)
4) Test:
   I. Table with overview over test case.
   II. The actual (Java coded) JUnit test classes.
   III. The result of the 1st JUnit test run and the last.