Assignment 1

TriviaTime GUI Application (Part 1)

Due: not later than Monday, April 2nd, 2018, 11:59 p.m.

Description:

The purpose of this assignment is to build an application that allows the user to play a trivia game by displaying a series of questions and possible answers loaded from a file. You will thus demonstrate your understanding of the following course learning requirements, as seen in the *CST8284—Object Oriented Programming (Java)* course outline:

1. Install and use Eclipse IDE to debug your code (CLR I, VIII)
2. Write java programming code to solve a problem, based on the problem context, including UML diagrams, using object-oriented techniques (CLR II)
3. Use basic data structures (CLR III)
4. Implement program Input/Output operations (CLR V)
5. Identify appropriate strategies for solving a problem (CLR IX)

Worth
13%
of your total mark

Assignment 1

TriviaTime GUI Application (Part 1)

Program Description

This program is designed to present a series of questions to the user, which are answered by selecting the radio button corresponding to the correct answer. The user confirms their choice by clicking on a submission button ("That's my answer" in figure II). The words 'Right!' or 'Wrong' are then displayed below the questions, along with a brief explanation of the correct answer (not shown in figure I). As well, the "Next Question" button is activated, so the user can click on it to move to the next question.

When the quiz is finished, a GUI screen appears that summarizes the results, listing the question by number, whether it was right or wrong, and the final score.

Each bank of questions is stored in a .trivia file loaded into a folder called C:\TriviaTime on the user's hard drive. ((Note: normally, it is inadvisable to hard code a path into your code. But in this case we'll assume that a startup directory was established when the application was loaded, and that the C:\TriviaTime folder was created as part of this initialization. In actual fact, you'll need to manually create the folder in Section I).

The contents of the file are loaded into an array of QA objects. Each object extends from the abstract QARequirements class, hence each QA object is guaranteed to have set of appropriate methods, such as getQuestion(), getAnswers(), getCorrectAnswerNumber(), etc. which return a String, and array of Strings, and an integer respectively.

Initially, your code will only load one .trivia file, Computer_Trivia_Java100.trivia, which has only seven questions. In Assignment II, we'll modify this to load other .trivia files of varying sizes.

I. Load the code for this lab into Package Explorer, and study the classes provided.

a. In Eclipse, import the CST8284_Assignment1 project into package explorer, using Assignment1.zip, which may be downloaded off of Blackboard.

b. Additionally, you'll need to create a folder on your hard drive called C:\TriviaTime. Load the file Computer_Trivia_Java100.trivial into it; this is also available on Blackboard.

   If for any reason you cannot create this directory on the C:\ drive, then you must make note of this in your submission, otherwise you may lose marks if it appears that your code doesn't not work as required.

II. Review the files in Assignment1.zip and make sure you understand their purpose

There are five Java classes included with this assignment: TriviaTimeLaunch, Controls, QAPane, FileUtils, and QARequirements, which you must use in your assignment, *and which you must submit with your finished submission*. Additionally, you'll need to provide the QA class, which extends QARequirements. QARequirements should remain untouched, along with FileUtils; in the remaining classes, you'll be required to supply the code described in the TODOs, as outlined in this document.

An overview of the five classes provided follows.

a) **TriviaTimeLaunch** is the main point of entry for your program, hence it contains both the `main()` method and the `start()` method, as described in the

lecture and the course notes (Module 05). The code is provided for you. Additionally, as indicated in the UML diagram (below), this class should contain setters and getters for the rootPane object, which will be loaded into the scene, which you instantiate just prior to showing the primaryStage. You'll need to supply this code, as indicated in the two TODOs.

Your setter thus needs to instantiate a new BorderPane object, which should be passed with the appropriate "Welcome" message, as shown in figure I below. (We'll change this to something 'splashier' in Part II of this assignment)

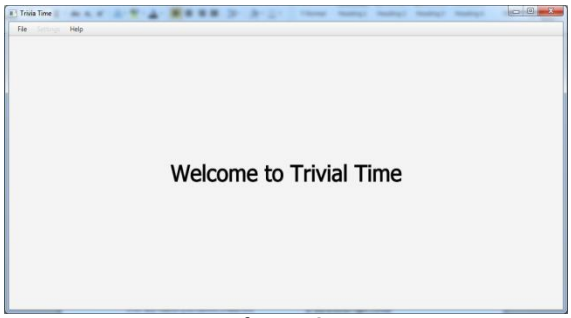


figure I

Note that this initial startup pane doesn't actually do anything (yet). Real activity only begins when the user loads a new .trivia file from the File menu.

b) Controls contains static methods related mostly to the menus, as well as the 'Next Question' button that appears in the bottom right corner of the display when the questions are loaded (see figure II below). Since the menu and 'Next' button remain constant features throughout program execution, they can be static, as indicated in the UML diagram.

The three components of a menu are a MenuBar, a Menu, and a MenuItem. For an overview on these controls, consult the following:

- https://docs.oracle.com/javafx/2/ui_controls/menu_controls.htm.
- http://www.java2s.com/Tutorials/Java/JavaFX/0560__JavaFX_Menu.htm
- https://o7planning.org/en/11125/javafx-menu-tutorial

Note that the UML diagram indicates that the public method getMenuBar() returns a MenuBar object. Thus to load a menu into the top pane of a BorderPane object, bp, you'd simply use:

```
bp.setTop(Controls.getMenuBar());
```

as indicated in the TriviaTimeLaunch class. Of course, your MenuBar must be preloaded with Menus and MenuItems first, as indicated in the UML diagram, and in the aforementioned websites, which you'll need to consult.

For this assignment, your menus/menuitems need only consist of the following:

- File
  - New Game
  - Exit
- Settings
- Help
  - About

Note that the Settings menu item can be disabled for now. We'll be using it in Assignment II.

(At no point should you need to implement an ActionListener() to any Menu/MenuItem, or indeed, to any control used in this assignment. You should be able to do everything needed using the setOnAction() method associated with each Menu or Node object.)

In addition to static menu controls, the Controls class also handles clicks on the 'Next Question' button, seen in the bottom right hand corner of figure II below.

Remember that your program is *event driven*. Thus *any* click on *any* control will usually result in some action being performed. The amount of work you'll need to do varies considerably. For example, the easiest event handler to implement—aside from the one for getMnuItemAbout(), since the code is provided for you—is the Exit MenuItem. Use `Platform.exit()` to initiate application shutdown. This takes only a single line of code (if implemented in a lambda expression), so if you're unsure how to begin, this is a good place to start.

At the other end of the spectrum is getMnuItmNewGame(), whose selection causes a .trivia File to be loaded, using a static method in the FileUtils class. As already mentioned, each .trivia file contains an array of QA objects, and so selecting this menu item must first cause the file to be transferred to an array using FileUtils.setQAArray(), and then load the first element of the QA array to into the center panel of the border pane using the QAPane constructor(see below). Done properly, this amounts to less than a dozen lines of code. However, debugging this can easily take a couple of hours if problems are encountered.

Because the various menus and nodes loaded in the Controls class rely so heavily on the primary stage—how else would they know what they're loaded onto?—the stage is passed as a parameter to getMenuBar(). It should be saved with an appropriate setter, as indicated in the UML diagram, so that it can be retrieved later by any control that needs to access the top level control into which everything is loaded. (For information on how to use the stage object to access any control on the scene, see the Tips section at the end of this document.)

c) Not all of the controls seen in figure II are handled in the Controls class. Each displayed question, set of answers, 'That's my choice' button, along with the explanation for why the answer chosen was right of wrong, is handled by the **QAPane** class. Each QAPane object is instantiated with a QA object, which you read out of the .trivia file via the appropriate FileUtils method, described next. The complete layout—question on top, possible answers below, followed by room for the explanation (don't forget the words "Right!" or "Wrong") and the 'That's my answer' button—must be laid out in the constructor, so that the pane this is displayed in can then

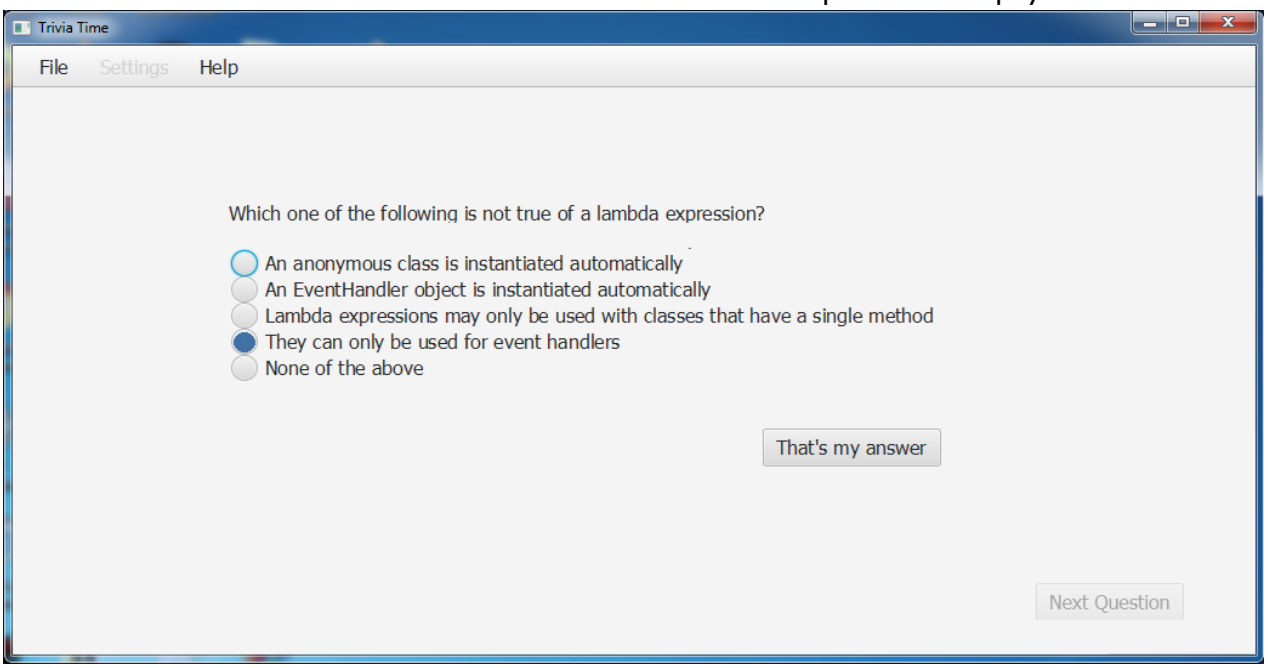


figure II

be made available for loading into the center panel via a public getter—as indicated in the UML. (Note that some of this business can be offloaded to other methods, like getRadioButtonSelected() and getAnswerPane(). You are free to add additional methods as you see fit.)

While most of this constructor deals with the laborious business of laying out controls in an attractive and consistent fashion, there is one tricky piece of business which you must address in this constructor. When the 'thatsMyAnswer' button is clicked, the setOnAction() method must perform the following:

(1) Read the array of radiobuttons to see which answer the user selected (again, the getRadioButtonSelected() method is a good place for this code).
(2) Compare this result to the correct answer using the current QAs getCorrectAnswerNumber() method, and save the result in the current QA object using its setResult() method, true is the answer chosen was correct, false otherwise. (See the note at the end of this document about the offset between the answer array's numbering and the number stored as the correct answer in the QA array.)
(3) Indicate to the user that they were right or wrong based on the above result, and display the explanation below the answers
(4) Disable the thatsMyAnswer button, as well as the radio buttons, since the user should not be able to re-enter another answer
(5) Enable the 'Next Question' button, so the user can advance to the next question.

As with getMnuItmNewGame(), the core of your code can be implemented in under a dozen lines—with responsibility for the radio buttons handled with calls to specialized methods, also indicated in the UML—but debugging this can be time consuming; don't wait to the last moment to get started, or you'll find the most crucial features of your program—which pack the most marks—are inoperable.

An excellent source of information on the use of `ToggleGroups` and `RadioButtons` can be found at

http://docs.oracle.com/javafx/2/ui_controls/radio-button.htm

Additionally, the following resources may prove valuable in helping you to align your controls:

- http://docs.oracle.com/javafx/2/layout/size_align.htm
- http://stackoverflow.com/questions/26169445/how-do-i-center-javafx-controls
- http://stackoverflow.com/questions/16083683/button-alignment-in-javafx
- http://tutorials.jenkov.com/javafx/gridpane.html

d) **FileUtils** contains static methods for loading an array of QA objects from the designated file. Once you've called setQAArray(), you should be able to call getQAArray() and return this to an array of QA objects. Of course, you must first supply a QA class, extended from QA requirements, before FileUtils works properly. Otherwise, you should not need to make any changes to this class.

e) **QARequirments** is an abstract class that contains numerous abstract setters and getters related to the QA objects that store the features . As already mentioned, you must provide a sixth class, **QA**, which extends QARequirements. You are thus responsible for overriding all of QARequirements abstract methods.

(g) Additionally, you must summarize the results of the quiz by listing each question number and whether it was correct or not, along with the

final score. The actual implementation is left to you, however you'll probably want to create an additional **Results** class to store the methods you'll need to output the trivia quiz details.

III. Notes, Suggestions, Tips, and Warnings

a. While you have been given a UML diagram (at the end of this document) and told how this program *must* behave, you have considerable liberty in determining *how* to implement your code. So while the UML diagram indicates the absolute minimum code that will be needed to write this application, you may elect to add new methods if you feel it is necessary. Similarly, you are not required to use a BorderPane if you prefer to use some other Pane to structure your layout. Thus while the UML diagram, along with the information supplied in Section II, are *indicators* of how your code should be written, you are at liberty to implement the code in any reasonable fashion, provided the program execution is bug-free. (However, you would be well-advised to follow the suggestions provided in this document.)

b. Certain features of this program are specified in anticipation of code you will be required to supply in Assignments 2, (such as the disabled Settings menu). So while these features may seem trivial and unnecessary now, they will be put to use shortly, and must be implemented as described.

c. Since each of the controls will have event handlers associated with them, it is highly advisable to structure your program in a way that keeps each code module as simple as possible, with all of the code related to the operation of that control inside a single method (similar to the way the About alert is handled in getMnuItmAbout()).

d. Before contacting the instructor about problems you are having with your code, be sure to first use debug to isolate the location of the problem.

   After just a few hours of working on your code, *you* are the world expert on its operation, and it's unreasonable to expect that anyone else would be able to jump in on short notice and spot your errors (particularly if your code is unstructured and undocumented).

   Therefore, before requesting assistance, you should set breakpoints in your code at the 'last known good' location (as indicated in the red error message that appear in the console), and step forward from there in debug until the error is encountered. And if the 'last know good' location is the first line of the program, then that's where you'll set your breakpoint. Read the error dump in the console, and locate where the problem occurred. Debug at that location. Reset the breakpoint to the next 'good' location. Repeat as required.

   Contact the instructor if you are stuck, but only after you've made a valid attempt to fix your program using debug.

e. You are ***not*** required to supply documentation with Assignment 1.

f. Students are reminded that:

   - You should not need to use code/concepts that lie outside of the ideas presented in the course notes (aside from, e.g., those topics that you are expected to research on your own, like menubars and radio buttons.)
   - You *must* cite all sources used in the production of you code according to the information provided in Module00. Failure to do so *will* result in a charge of plagiarism. The one exception to this rule is the course notes; you are not required to cite them.
   - Students must be able to explain the execution of their code. If you can't

explain its execution, then it is reasonable to question whether the you actually wrote the code or not. Partial marks, including a mark of zero, may be awarded if a student is unable to explain the execution of code that he/she presumably authored.

- Getters and setters should be employed throughout your program; *never* reference a private variable directly when a getter/setter already exists.
- Your name MUST appear in the About dialog, or *marks will be deducted*.

g. You are not required to use lambda expressions; you can use inner or anonymous classes if you prefer. However, your code will be considerably simplified if you employ lambdas.

h. The correct answers obtained from the QA objects are 'one-based', not 'zero-based'. That is, answer are loaded as an array of Strings, and this array starts at 0. But the value indicated as the correct answer in the QA file starts at 1, not zero. So there is an offset of 1 between the answer stored—which is based on what a human being would recognize as the correct answer—rather than the numbering of the answer array.

i. As previously stated, you have considerable latitude in the breakdown of your code and in the layout of the display, including which type of pane(s) you use, which fonts, etc. Note however that regardless of how your code is written, the execution of the program must result in standard 'Windows-type' behaviour. Examples include:

- All controls are reasonably organized, not scattered in unlikely places around the screen;
- Garish colour schemes are to be avoided
- Controls which cannot be used are greyed out (e.g. menus that are not implemented, or navigation buttons, when the user has reached end of the underlying array);
- Menus, once shown, should not mysteriously disappear during program execution;
- Text messages (which may have been used for debugging purposes) should not appear in the Eclipse console of a GUI application during execution: remove them;
- Dialog boxes should close once a selection is made;
- The user always has options to select; the screen should never be entirely blank, or appear 'frozen', with nothing 'clickable' in view;
- ALL TODOs are removed from the comments;
- Scroll bars should be hidden when they are not needed;
- etc.

We'll refer to these, and all other unstated, obvious Windows App operations as 'reasonable behaviours'. Failure of your program to behave in a typical fashion will result in lost marks, even when such behaviours are not included in the above list.

j. Given the time constraints of the course, there are no extensions for this assignment, baring a family or personal emergency (which must be documented by a suitably-accredited professional, e.g. a doctor.)

k. The instructor's version of the code will be released on midnight, April 5th. For those students who do not complete this lab on time, or who wish to build their Assignment 2 off of more robust code than they produced, you are free to use this 'official' version, without citation. The code may also prove valuable to student's who simply wish to study the code to help with their own understanding of Java.

Note however, that once the 'official' version of the code is released, it essentially nullifies any outstanding assignments: all unsubmitted Assignment 1's are immediately worth 0. This even applies to students who have been given special dispensation due to

personal circumstances; I can only put off releasing my version of the code for a few days after the deadline, otherwise a large number of students will not have access to the code they need to complete Assignment 2.

l. The left, right, top, and bottom regions of the BorderPane disappear if not filled with something. You can load 'buffer' panes into these regions to help keep the objects in the center region properly centered. For example, if bp is a BorderPane object, then:

```
VBox lBuf = new VBox();
lBuf.setPrefWidth(100);
bp.setLeft(Buf);
```

has the effect of inserting an empty border region of 100 pixels into the Left region of the borderpane.

m. Given the primary stage, you should be able to locate any node object loaded into any parent object such as a Pane, provided you know in which order it was loaded. For example, given `primaryStage pS`, the scene is returned using:

```
Scene scene = pS.getScene();
```

The parent loaded into the scene can be retrieved by using the Scene's `getRoot()` method. Since the Parent class is an abstract superclass of some concrete subclass object (usually a Pane), we can get that object by casting the Parent object to the appropriate Pane type, e.g.

```
BorderPane bp =
(BorderPane)pS.getScene().getRoot();
```

From there, we can use the BorderPane's methods to load it's regions and subPanes, obtain access to, and fire, buttons, etc. Thus having the primary stage gives us access to anything loaded into it.

Similarly, if you have a method that returns an HBox or VBox, you can get access to any of its children using the `getChildren()` method. If you know the order in which the child nodes were loaded you can access any one of them by number, using the `get()` method. Hence

```
(Button)getNextBtnPane().
              getChildren().get(0)
```

will return the 'Next Question' button

n. Note that you should not insert SerialVersionUID into your code. *While it may be useful to help you debug your code initially, it will cause your program to fail when run on other devices, i.e. the instructor will not be able to mark your program on his PC.*

Common causes for this are:

- Your `public String[] getAnswer()` method does not load an array of Strings correctly. So if your code says:

  ```
  this.answer = answer;
  ```

  then you've only loaded a reference value to an array, but you haven't actually download the array of Strings itself

- The signature of your QA object does not match the signature of the QA object inside the .trivial file. Perhaps your QA constructor (required as of Vers. 1.1 of this document) has the wrong signature. The QA object's signature is fairly precise and any departures from what is written in the UML diagram will cause problems.

- Not using the latest version of the .trivia file, loaded up on Blackboard. Best to make sure you are using fresh copies of the source files before you begin.

Again, using your own SerialVersionUID may help you overcome this problems, but if you code won't work on the lab professor's laptop, then your code cannot be marked. So adding SerialVersionUID is not the solution to your problems.

IV. Submission Guidelines

Your code should be uploaded to Blackboard (via the link posted) in a single zip file obtained by:

1) selecting the ***package*** name (`cst8284.triviatime`)

DO NOT SELECT
THE PROJECT NAME

2) right clicking on 'Export'
3) make sure 'Archive File' is selected, and click Next
4) in the Archive File window make sure *all* of the program files are selected, including those provided to you, in the pane at right, and the 'Save in zip format' radio button is selected below
5) after creating the zip file, give it the following name EXACTLY AS WRITTEN

LastName_FirstName_Assignment1.zip

including the underscores, but with *your* last and first name inserted as indicated.

Note:

- you do not need to include the .trivia files with your submission.
- Since there are always some students who upload empty zip files with their assignment, a good safety precaution is to always take the .zip file you've just uploaded to Blackboard and open it onyour laptop. Better still, start a new project in Eclipse, and load your code into it. Make sure it works exactly as you expect after it gets transferred (you'd be surprised how often this simple test fails, and the uploaded zip is empty)
- You can upload as many attempts at Assignment 1 as you'd like, but only the final attempt is marked: all other attempts are ignored.

Addenda:

Corrections to this lab may be posted as required, so check Blackboard periodically for notice of any corrections to this document.

Version 1.1: March 20

- QA constructor added to UML diagram

Version 1.2 March 22

- Additional information was tacked on to section II.(e) which has nothing to do with this lab. This has been removed from the document.

Version 1.3 March 27

- Corrections to UML diagram, where there were differences from the code supplied, e.g. static missing, and should have difficulty property rather than difference. Note that most of these were fairly obvious, and none of these errors should have any real effect on your program.
- Note that the signature of the QA constructor must be an exact match of the code in the UML to work. This includes:
  i. Loading the QA constructor in the order indicated in the UML diagram (Note that 'difference' should have been 'difficulty', which has now been corrected in the UML)

ii. The array of answers must be correctly loaded. This means instantiating a new array of Strings, and actually loading that array by looping through the array *pointed to* by `answers`. Thus writing only

```
this.answers = answers
```

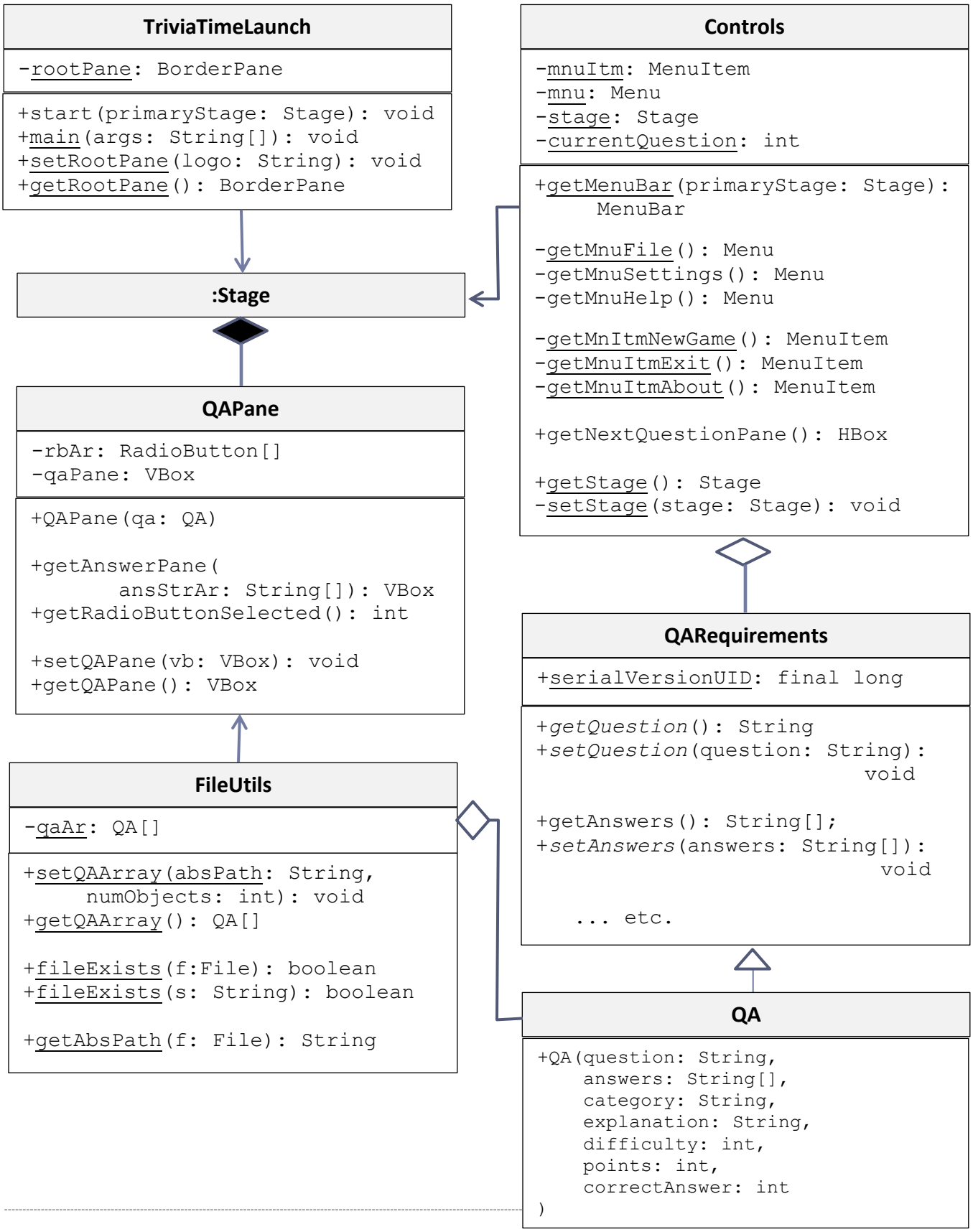
in your setter is not enough; the setter must load the strings into the answers array.

iii. The private variables in QA constructor must match the variable names indicated in the UML. This is somewhat bizarre, since Java shouldn't care about the actual names of the identifiers: but for mysterious reasons, it does. Thus your private declarations in the QA array will need to be:

```
private String question,
   category, explanation;
private int
   correctAnswer, points,
   difficulty;
private String[] answers;
private boolean result;
```

and your getters and setters will need to use these identifiers as well, of course. Again, this is someone unexpected and inexplicable, but it is required to make the code work.

- Added note about *not* overriding SerialVersionUID. Your code will not work on the instructor's laptop if you use this to *fix* your problems, and you cannot be marked if your code won't run on the professor's laptop because you added your own SerialVersionUID. So under no circumstances can you have SerialVersionUID added to your QA object.


TriviaTimeLaunch
-rootPane: BorderPane
+start(primaryStage: Stage): void
+main(args: String[]): void
+setRootPane(logo: String): void
+getRootPane(): BorderPane
Controls
-mnuItm: MenuItem
-mnu: Menu
-stage: Stage
-currentQuestion: int
+getMenuBar(primaryStage: Stage): MenuBar
-getMnuFile(): Menu
-getMnuSettings(): Menu
-getMnuHelp(): Menu
-getMnItmNewGame(): MenuItem
-getMnuItmExit(): MenuItem
-getMnuItmAbout(): MenuItem
+getNextQuestionPane(): HBox
+getStage(): Stage
-setStage(stage: Stage): void
:Stage
QAPane
-rbAr: RadioButton[]
-qaPane: VBox
+QAPane(qa: QA)
+getAnswerPane(ansStrAr: String[]): VBox
+getRadioButtonSelected(): int
+setQAPane(vb: VBox): void
+getQAPane(): VBox
QARequirements
+serialVersionUID: final long
+getQuestion(): String
+setQuestion(question: String): void
+getAnswers(): String[];
+setAnswers(answers: String[]): void
... etc.
FileUtils
-qaAr: QA[]
+setQAArray(absPath: String, numObjects: int): void
+getQAArray(): QA[]
+fileExists(f:File): boolean
+fileExists(s: String): boolean
+getAbsPath(f: File): String
QA
+QA(question: String, answers: String[], category: String, explanation: String, difficulty: int, points: int, correctAnswer: int)