Assignment 2

TriviaTime GUI Application (Part 2)

Due: not later than Wednesday, April 18th, 2018, 11:59 p.m.

Description:

In this assignment you'll add new features to your existing TriviaTime application , as described in the document below. In the process you will demonstrate your understanding of the following course learning requirements, as listed 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, such as ArrayList (CLR III)
4. Use simple generic classes to manage objects, e.g. using Collection class objects (CLR IV)
5. Implement program Input/Output Operations (CLR V)
6. Prepare program documentation using prescribed program specifiers (CLR VII)
7. Debug program problems using manual methods and computerized tools in an appropriate manner (CLR VIII)
8. Identify appropriate strategies for solving a problem (CLR IX)

Worth
10%
of your total mark

Assignment 2

TriviaTime GUI Application (Part 2)

I. Copy the existing Assignment 1 into a new project folder

a. In Eclipse, open your existing CST8284_Assignment1 folder, right click on the `src` folder, and select 'Copy' from the drop down menu.

b. Now create a new project called CST8284_Assignment2. Right click on the project folder, and select 'Paste' from the drop down menu. This effectively backs-up all the files in the Assignment1 folder in the Assignment2 folder.

c. Set up the Build Path for JavaFX according to the instructions outlined in Lab01. This should remove all of the red error labels from your code. Also, this is a good time to refactor your code, if necessary.

II. Add the following features to the TriviaTime application

a) **Replace the QA array with an ArrayList of QA objects**
The original trivia test file contained seven QA objects, and thus the original program was hardcoded to deal with trivia files displaying exactly seven questions.

We now wish to allow for trivia files having *any* number of QA objects. To do this, first replace the QA array with an ArrayList of QA objects. Therefore, you'll need to replace

```
private static QA[] qaAr;
```

with:

```
private static ArrayList<QA> qaAL;
```

You'll need to alter any code that previously used `qaAr` to use `qaAL` instead.

Next rename `setQAArray()` to `setQAArrayList()`, and change all references in code from the former to the latter method. Because we'll be using ArrayLists of variable size, there's no need to pass the number objects into our new method, hence your declaration should be:

```
public static void
   setQAArrayList(String absPath)
```

Finally, to ensure we can read in QA files of any size, alter the try…catch block in FileUtils as follows:

```
try (
  FileInputStream fis
   = new FileInputStream(absPath);
  ObjectInputStream ois
   = new ObjectInputStream(fis);){
  while (true)
   // load the QA ArrayList here
}
catch (EOFException e) {}
catch (IOException |
      ClassNotFoundException e) {}
```

You'll need to load the QA objects from the trivia file into the array list (inside the while loop) just as you loaded them into the `qaAr` array in Assignment 1.

b) **Add a SplashScreen Animation**
Currently, the root pane displays only a single line of text: 'Trivia Time', but it does little else. We need to liven this up with some animation.

Oracle has provided a nice summary of Java's animation features; it's located at https://docs.oracle.com/javase/8/javafx/api/javafx/animation/package-summary.html. JavaFX animations fall into two categories: Timeline and Transition. (See http://docs.oracle.com/javafx/2/animations/ba

sics.htm for a good overview.) In this course, we are only interested in Transitions, of which there are many to choose from, including the

```
ScaleTransition,
RotateTransition,
PathTransition,
FadeTransition,
SequentialTransition, and
ParallelTransition.
```

For an introduction to Transitions, see any of the following:

https://docs.oracle.com/javase/8/javafx/visual-effects-tutorial/basics.htm#BEIIJJJB

http://www.java2s.com/Tutorials/Java/JavaFX/1000__JavaFX_Transitions.htm

https://rterp.wordpress.com/2015/09/01/creating-custom-animated-transitions-with-javafx/

Note that transitions may be run either sequentially or in parallel (or in combinations of serial and parallel). See

http://www.dummies.com/programming/java/how-to-combine-transitions-in-javafx/

for details. Also: while most of these examples use rectangles for their animations and effects, any node will do, including the text object containing the words 'Trivia Time'.

Furthermore, JavaFX provides an Effects class, which allows you to produce results such as the following `DropShadow` Effect:

The list of Effects includes bends, blurs, glows, shadows, and reflections. For a complete list, see https://docs.oracle.com/javafx/2/api/javafx/scene/effect/package-summary.html

A good introduction to the `DropShadow` Effect, seen above, can be found at http://docs.oracle.com/javafx/2/visual_effects/drop_shadow.htm#CEGFADCA, from which the above graphic was taken. For information on the other Effects available and their use, see the other web pages in this series. (Note that Transitions and Effects may be combined with impressive results.)

To complete the requirements for this section, you must, as a minimum, use any two of the following:

- `ScaleTransition`
- `RotateTransition`
- `PathTransition`
- `FadeTransition`

combined with either of the following

- `SequentialTransition`
- `ParallelTransition`

including any combination of the above. Additionally, you should provide at least one Effect with your Transition.

Your mission for this part of the assignment then is: apply animation and effects to the 'Trivia Time' text box and make a memorable splash screen.

c) Allow the user to Select trivia files in other folders

In Assignment1 the location of the trivia file was fixed in the C:\TriviaTime directory. Add a FileChooser to your application that uses C:\TriviaTime as the default location, but allows the user to select trivia files from another folder, if they so choose.

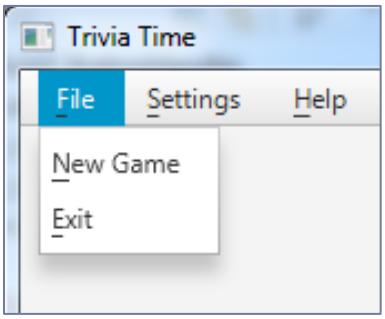
d) **Add Accelerator keys to the menu**

Add accelerator/shortcut keys to each menu/menu item, as follows.
First, to indicate which key is the shortcut/accelerator key, place the underscore character directly before that character. So,

for example, to load a menu using Ctl+'F' used as the accelerator key, you would load the Menu with the relevant character preceded by '_', e.g.

```
Menu mnuFile = new Menu("_File");
```

The underscore causes the 'F' to be 'decorated' with an underline, as indicated in the figure below:



Note that in JavaFX you will need to press the 'Alt' key first for this mnemonic character to appear in the menu. (This may vary with OS)

Second, in Windows, the default key to press is the 'Alt' key, along with the 'decorated' key—'Alt-F' in the above example. We wish to override this default with the Control key. Hence to select the File Menu, you would press Ctl-F, not Atl-F. You'll need to do some research on this, and a good place to start is:

https://blog.idrsolutions.com/2014/04/tutorial-how-to-setup-key-combinations-in-javafx/

As before, this feature should ideally be handled inside the method responsible for that particular menu item. If you've modularized your code correctly according to the guidelines provided in Assignment 1, adding this feature to each method should be straightforward, once you've figured out how to use the `KeyCombination` object.

e) Ability to sort according to difficulty, points, etc using Settings Menu.

In Assignment 1 we included the Settings menu, which is initially disabled. In this assignment, we wish to add three features to the Settings menu:

- Randomize Questions
- Increasing Difficulty
- By Topic

This is where your QA ArrayList is put to use. These topics will require some research, but each should require only a few lines of code to implement (at most). Here are some tips to get you started:

- To randomize the questions, use the `Collections.shuffle()` method.
- To sort by difficulty (from the easiest to the hardest), use `Collections.sort()` with the `compare()` method using `getDifficulty()`
- To sort alphabetically by topic, use `Collections.sort()` with the `compare()` method using `getCategory()`

We will allow these selections to be made even *after* the game has started. Therefore, if the user elects to change the Setting after answering the first few questions, the game should start over again (i.e. reload the first QA object, but this time with the sort applied) without penalty.

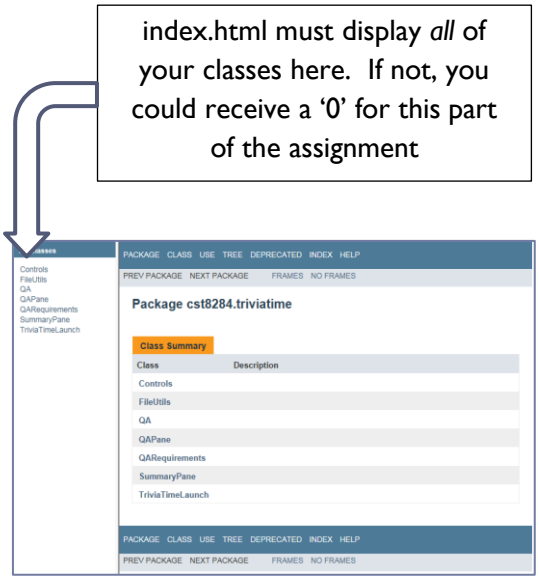
f) Documentation/JavaDoc Document all classes and methods using the JavaFX documentation generator

Follow the instructions in Lab 1 regarding the use of the JavaDoc generator. Additionally, review the document "Generating JavaDocs" which is available on Blackboard along with this Assignment.

A copy of the ACDS can be downloaded from Blackboard as well, along with the pdf *JavaDocRecommendations*.

Note that the ACDS mostly follows the Oracle documentation available at http://www.oracle.com/technetwork/articles/java/index-137868.html, "How to Write Doc Comments for the JavaDoc Tool." This web page provides a useful example of *what is expected of you* in documenting your code, which is reproduced in the text box above.

As noted in Lab 1, a common error is to forget to select the project *before* generating Javadocs. If you run the Java generator when only a single class is selected in the Eclipse Package Explorer, then only that class will have its comments appear in the JavaDoc doc folder—and you will lose marks despite the fact that you've documented your other class's methods correctly, but they just don't show up in the doc folder.



Therefore, *after* you have generated your JavaDoc folder in Eclipse, but *before* you have zipped and shipped the *entire project* (as described below), open the doc folder and double click on index.html. This will generate a web page in Eclipse. Assuming you have selected the project folder *before* running JavaDocs, *all of your project's classes should appear in this web page in the pane on the left hand panel of the webpage*. If classes are missing, then go back and regenerate the JavaDoc doc folder again, this time being certain to click on the Project folder *first* since otherwise you will lose marks (since I can't mark what isn't submitted).

III. Notes, Suggestions, Tips, and Warnings

a. As with Assignment 1, while you are told how this program is expected to behave, you are at liberty to implement the code in any reasonable fashion, provided the program execution is bug-free. (As before, you would be well-advised to follow any suggestions provided in this document.)

b. As with Assignment1, 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 a control located inside a single method

c. 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.

   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.

d. Students are reminded that:

- You may use the Assignment 1 code as your basis for building Assignment 2. You do not need to cite the professor by name; however, in all other cases, you must provide the source of your code through appropriate citations. Failure to do so *will* result in a charge of plagiarism.

- 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.)

- 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 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*.

e. 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.

f. As stated in Assignment 1, 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;
- If the user cancels a choice in FileChooser, the program does not generate a NullPointerException
- 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.

g. Given the time constraints of the course, there are no possible extensions for this assignment.

IV. Bonus Marks (3 marks = 1% extra)

If you followed the Assignment 1 instructions correctly, you may have noticed that most of your Menu Item methods do pretty much the same thing. They (1) instantiate a new menuItem with a given name (2) load an event handler, and (3) return the new menuItem, which can be used to load the Menu. In this assignment, we add a new feature to this list: (4) add an accelerator key to the menu item.

This strongly suggests that we could tidy up a considerable amount of code by creating a single static method that took as parameters (1) a string that passes the name of the new menu item (like "New Game" or "Exit") (2) a character to indicate the accelerator key, and (3) an EventHandler object, but using a lambda expression (to avoid having to instantiate space-consuming inner classes).

Similarly, each Menu method does three things, with little variation: it (1) instantiates a new Menu with a given name (like "File") (2) uses getItems().addAll() to load one or more menuItems, and (3) returns the new Menu from the method. Again, this suggests that a single static method, similar to the one for menu items, could be used to do the job (including adding the accelerator key).

Your task then is to build these two static methods, called `getMnuItm()` and `getMnu()`, and employ them in the Menus class to tidy up your repetitious code. You may find you need to:

(1) Make a few local properties global within the class, i.e. move them from inside the methods they currently reside in, and 'elevate' them up to global visibility in the class itself

(2) Create separate methods for the overridden `handle()` code. Where previously you referred to this code in lambda expressions, probably using something like `()->{…}`, now, to keep things tidy, you'll need to refer to this code via separate auxiliary methods inside the Controls class.

Nonetheless, you'll find that if you implement these two methods and employ them correctly, that you'll probably save upwards of 50 lines of code in the Menus class, with the added benefit that the code will be both more compact and much clearer.

(And if you don't find your code works better as a result, then you're missing something. Contact me for further directions if you're stuck.)

Note that if you don't see an obvious improvement in your code, then you haven't implemented your static methods correctly, and you will only receive a fraction of the bonus marks available.

Hint: you'll want to look into Java *varargs*, which will be useful in passing the new menuItems to `getMnu()`.

NOTE: If you attempt the bonus, clearly indicate that you have done so by adding a note with your submission to that effect. (Otherwise your hard work may get overlooked, and remain unmarked.)

V. Submission Guidelines

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

1) selecting the ***project*** name (`CST8284_Assignment2`)
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 on your laptop. Better still, start a new project in Eclipse, and load your zipped you expect after it gets transferred (you'd be surprised how often this into simple test fails.)
- 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.