Homework 10 – StarterUpper

Problem Description

You're a student who has always had an interest in startups and wish to get use this summer to get *started* on one – pun intended. After viewing countless videos about successful entrepreneurs, you found that a common recommendation is to start by identifying **a problem that people actually want solved**.

Given your newfound JavaFX skills, you thought it would be useful to build a user-friendly tool for organizing your problem ideas. In particular, you want to build a form-based GUI program that will: (a) collect your ideas along with bits of evidence you can use to assess their worth; and (b) roughly rank the ideas based on their respective value.

NOTE: There are going to be some basic requirements to the GUI, but the actual scene design is entirely up to you. The JavaFX module on Canvas provides a foundation on the basics of GUI building. Feel free to explore some other fun aspects of JavaFX! It is perfectly fine and essentially expected that you look up documentation on specific JavaFX classes. This homework is intentionally more open-ended, and we grade most of it manually.

Inspiration

This particular assignment was inspired by Dr. Omojokun's affiliation with GT's CREATE-X initiative. CREATE-X's fundamental goal is to "to instill entrepreneurial confidence in students and empower them to launch successful startups". It offers several programs and is open to anyone regardless of major and year. In fact, consider this assignment and perhaps the extra credit list of problem ideas you may submit as a springboard into one of the programs. *Yay, extra credit!*

You can learn more about CREATE-X by checking out https://create-x.gatech.edu or reaching out to Dr. Omojokun. Watch out though! He routinely mentors CREATE-X Capstone teams, and if you've met or are close to meeting all of your program's Capstone Design pre-requisites, he may work hard to convince you to take that particular version of Capstone. Even if you're not a CS major, you'll find that CREATE-X Capstone is also offered to BME, ECE, IE, and ME majors; see: http://www.create-x.gatech.edu/create-x-capstone-design

Finally, if you're still semesters away from taking Capstone Design, he recommends that you check out Startup Lab, which has no pre-requisites: https://create-x.gatech.edu/startup-lab

Solution Description

Before starting, make sure to install JavaFX. The instructions for that can be found on Canvas - Module 10.

As part of your solution, you'll be provided two files, which **you MUST submit alongside any classes you write**:

- `FileUtil.java` - A utility class to help with saving startup ideas into a file so that you can later view and submit them!

- `StartUpIdea.java` - A class that represents a problem space to base your startup around. This class implements `Comparable<StartUpIdea>`, so different startup ideas can be compared and sorted based on the attributes that will be described later. The `toString` method is used in `FileUtil` to save each idea. For some extra credit opportunities, modifications for StartUpIdea might be necessary, and in that case, you will submit your updated version of the file instead of the original version.

You are to write a JavaFX class called `StarterUpper`. The class must meet the following requirements:

- The title of the window must be "Problem Ideation Form"
- Make sure to use at least one anonymous inner class and one lambda function in your implementation. (This is required for full points, check rubric)
- The set-up:
  - An area to display the form
  - Fields of the form that display questions and where users can input answers (**hint: Label, TextField, etc will be helpful**):
    - Field that asks, "What is the problem?" which takes in a String from the user
    - Field that asks, "Who is the target customer?" which takes in a String from the user
    - Field that asks, "How badly does the customer NEED this problem fixed (1-10)?" which takes in an int from the user
    - Field that asks, "How many people do you know who might experience this problem?" which takes in an int from the user
    - Field that asks, "How big is the target market?" which takes in an int from the user
    - Field that asks, "Who are the competitors/existing solutions?" which take in a String from the user
    - Note: As extra credit, you'll be given the opportunity to research and include another question that one might ask about a given problem.
  - Button that adds the current idea written in the fields to a List of StartUpIdea objects
    - Note, the attributes of a StartUpIdea object represent the answers to the questions above.
    - If the input fields are not the correct type, empty, or negative (when relevant), then the submit button should create a popup alert with a helpful message instead of adding to the list. For example, if the user omits information about the target customer, a pop will show notifying that a field is empty.
    - The input fields must be cleared after successfully adding the idea to the List
  - Button that sorts the List of ideas based on their rough potential. The StartUpIdea at index 0 should be the one with the most potential!
    - Hint: Take a look at the class Collections' sort method. Similar to the Arrays' sort method, it is a static method that sorts a data structure in place using the object's natural ordering, which is conveyed through the compareTo method. The only difference is that Arrays.sort sorts arrays, while Collections.sort sorts Lists and its subclasses.

- Button that resets the form
    - Delete the File (if it exists) (hint: take a look at the File Java API)
    - Clear the current List of StartUpIdea's
    - Clear all fields on the form (Should look like an empty/blank form)
    - Must have a pop up that asks the user if they are sure
        - Implementation up to you, as long there's a clear yes/no option to perform the action
- Button that saves all the added Ideas to File
    - Take a look at the method in the provided FileUtil.java file
    - Save ideas in the file **ideas.txt**.
    - Note: ArrayList and LinkedList implement List, so they can be passed in as parameters. You can use any of these.

Tips and Tricks

- The Java API is your friend. Use it to your advantage.
- Make sure you are properly able to run JavaFX programs before starting this assignment. DO NOT wait until the last moment to configure your JavaFX!
- Work incrementally. Get a basic UI up and running. Add buttons, cells for display, etc., piece by piece. Think of which type of layout manager(s) you want to use
- **Remember to test for Checkstyle Errors and include Javadoc Comments!**

Rubric

NOTE: This assignment will mostly be manually graded. As long as you cover the main points provided, you can be as creative with your implementation!

[100] StarterUpper.java

- [5] Window title
- [15] Entry Fields
- [60] Buttons
    - [20] Add Idea Button
        - [5] Handles Input Validation
        - [5] Has Pop-up
        - [5] Clears fields after success
        - [5] Adds Idea to List
    - [10] Sort Ideas button
    - [15] Reset Button
        - [2.5] Deletes Existing File
        - [2.5] Clears List
        - [5] Clears Fields
        - [5] Has Pop-up
    - [15] Save Button
- [10] At least one Anonymous inner class

- [10] At least one Lambda function

We reserve the right to adjust the rubric, but this is typically only done for correcting mistakes.

Extra Credit

There are many ways of earning extra credit on this assignment. Note that there will be a max of **75 points** awarded as extra credit to the homework grade. Each of them will require doing your own research beyond what is provided in this assignment and in the modules – hence the "extra" in "extra credit". You can pick different options (listed below) for earning the 75 points.

To earn extra credit, you should indicate which additional extra credit options you chose to do in a file named **ECDeclaration.txt**

- Copy and paste the titles of the tasks from this document
- If some but not all subsections of a task were done, copy and paste all the subsections you did immediately below the idea, indenting them with a "- " (dash space) at the beginning of the line
- If you wish to submit any specific comments/notes regarding your extra credit work, submit a separate text file named **ECComments.txt**

Extra credit opportunities

- Spend some time researching, observing, discussing pain points that bother you, people you know, or even strangers. Translate them into potential startup problems and responses to each of the questions presented by the GUI. Submit the resulting list as a text file generated by the program. This file should be submitted as **ECIdeas.txt**
    - **Awards: 20-40 points**, depending on number of ideas and target population of them
    - Unsurprisingly, as a student, you might find it easier to come up with problems that specifically apply to students (*e.g.* It's hard to find a quiet place to study on campus).
    - We encourage you to think beyond just that population. You can receive up to 8 points for each problem idea/analysis that applies to **both students and non-students** and up to 4 points for those that apply to just students.
    - If you have time, here's an essay on startup ideas that you may read: http://paulgraham.com/startupideas.html
    - You should create this file with your app. Once you are done, save it, and rename the saved file to **ECIdeas.txt**.
        - Furthermore, indicate which population your problem applies for at the end of the problem field
        - Separate the problem description and this information with a dash, such as "PROBLEM_DESCRIPTION – BOTH" or "PROBLEM_DESCRIPTION – STUDENTS"
    - We'll accept a maximum of 5 entries, so if you have many, share your best.
- Add a GUI Control that displays problems on the scene as you are submitting them
    - **Awards: 5 points**
    - Should populate that specific problem into the form as a new idea is added. Whenever the elements are sorted, the GUI should be updated.

    - Hint: there are many ways to do this, but take a look at ObservableList and ListView
        - All of the GUI updates can be easily implemented by using an ObservableList as the source of elements of a ListView
    - Note: If you also implement "Implement a way to remove specific ideas" (see below), the idea should also be removed from the GUI.
    - Note: If you implement "Implement a way to remove specific ideas" (also see below), the idea should be updated in the GUI.
- Implement a way to edit specific ideas
    - **Awards: 10-12.5 points**, with partial credit possible
    - Implementation up to you
        - To maintain the proper functioning of the controls responsible to add new elements, it is recommended that you create a specific set of controls whose sole functionality is editing a selected element
    - Choose one implementation:
        - [2.5 points] Idea is selected by a text field and a button, and it doesn't utilize the index of the element.
            - For example, you can make a text field for "search problem" and find the idea that matches. You can assume that all problem names are unique
        - [5 points] Idea is selected by clicking on the idea in the GUI. This option requires proper implementation of "Add an GUI Control that displays problems"
    - [2.5 points] Values are properly updated and correspond to the correct idea
        - Update with a button for "done" or similar
        - If no options from "Choose one implementation" are used, you have the option of using text entry for the index of the element
    - [2.5 points] Selecting an element brings its information to controls used to edit the element (user only should edit/retype specific fields). Basically:
        - You select an element, and all fields are loaded in the controls
        - You update the fields you wish to change
        - When you are done, the values are updated in the app's list of ideas, and the content of the controls are cleared
        - Be sure to keep track of the index of the selected idea, so when you are done you can put the values back even if they are on longer selected in the GUI
            - Note: This is especially important for when you pick the second option under "Choose one implementation"
    - [2.5 points] New values should still pass input validation.
        - Do not change any values if any new value is invalid (show the same pop-up as specified in the main solution description)
- Add a new field for evaluating a startup idea
    - **Awards: 5 points**, with partial credit possible
    - [2.5 points] Add field to GUI, add attribute to StartUpIdea and edit StartUpIdea's toString() method
        - The field should fit in the context of evaluating ideas, but other than that the specifics (question, valid inputs, etc.) are up to you.

    - [2.5 points] Modify StartUpIdea's compareTo() to consider the values from this field in a meaningful way
    - Note: This task can only be done once. No more than one field can be added.
- When the app starts, it loads ideas from the file ideas.txt (previously generated) and loads them into the list
    - **Awards 5 points**, with partial credit possible
    - [4 points] Loads ideas if file exists
        - The file has a very specific format. Think of how you can use the Scanner to parse an idea and how you can use lists (either ArrayList or LinkedList) to create a collection of initially unknown number of elements.
    - [1 point] Does nothing if file does not exist. Requires submission and proper functioning of first subsection
- Implement a way to remove specific ideas
    - **Awards: 5 points**
    - Implementation up to you
- Audio/SFX when an element is added
    - **Awards: 5 points**
    - Implementation up to you, but the audio should be apparent
- Add a picture
    - **Awards: 5 points**
    - Implementation up to you, but the image should be visible on the main window
- Add some color to the form
    - **Awards: 2.5 points**
    - Implementation up to you, but the color should be visible on the main window

Allowed Imports

- You can import **anything from the Java Library**!
- You will have most JavaFX functionality as well (no Swing Interop or WebView components). You can import any JavaFX package, but your program is expected to compile and run while explicitly adding only the following modules:
    - javafx.controls
    - javafx.media
    - javafx.fxml
- Note that adding those modules implicitly adds the modules javafx.base and javafx.graphics.

Feature Restrictions

There are a few features and methods in Java that overly simplify the concepts we are trying to teach or break our auto grader. For that reason, do not use any of the following in your final submission:

- `var` (the reserved keyword)
- System.exit

Javadoc

For this assignment, you will be commenting your code with Javadoc. Javadoc comments are a clean and useful way to document your code's functionality. For more information on what Javadoc comments are and why they are awesome, the online overview for them is extremely detailed and helpful.

You can generate the Javadoc overview for your code using the command below, which will put all the files into a folder named "javadoc". Note you should execute this after adding Javadoc comments to your code:

```
$ javadoc *.java -d javadoc
```

The relevant tags that you need to include are `@author`, `@version`, `@param`, and `@return`. Here is an example of a properly Javadoc'd class:

```
import java.util.Scanner;


/**
 *This class represents a Dog object.
 *@author George P.Burdell
 *@version 1.0
 */
 public class Dog {


    /**
     *Creates an awesome dog(NOT a dawg!)
     */
     public Dog() {
     ...
     }


     /**
      *This method takes in two ints and returns their sum
      *@param a first number
      *@param b second number
      *@return sum of a and b
      */
      public int add(int a,int b) {
```

```
        ...
        }
}
```

A more thorough tutorial for Javadoc Comments can be found here.

Take note of a few things:

1. Javadoc comments begin with `/**` and end with `*/`.
2. Every class you write must be Javadoc'd and the @author and @verion tag included. The comments for a class should start with a brief description of the role of the class in your program.
3. Every non-private method you write must be Javadoc'd and the `@param` tag included for every method parameter. The format for an @param tag is `@param <name of parameter as written in method header> <description of parameter>`. If the method has a non-void return type, include the `@return` tag which should have a simple description of what the method returns, semantically.

Checkstyle can check for Javadoc comments using the -a flag, as described in the next section.

Checkstyle

For this assignment, we will be enforcing style guidelines with Checkstyle, a program we use to check Java style. Checkstyle can be downloaded here.

To run Checkstyle, put the jar file in the same folder as your homework files and run

```
java -jar checkstyle-6.2.2.jar -a *.java
```

The Checkstyle cap for this assignment is **40 points**. This means that up to 30 points can be lost from Checkstyle errors.

Collaboration

Collaboration Statement

To ensure that you acknowledge a collaboration and give credit where credit is due, **we require that you place a collaboration statement as a comment at the top of at least one .java file that you submit**. That collaboration statement should say either:

I worked on the homework assignment alone, using only course materials.

or

In order to help learn course concepts, I worked on the homework with [give the names of the people you worked with], discussed homework topics and issues with [provide names of people], and/or consulted related material that can be found at [cite any other materials not provided as course materials for CS 1331 that assisted your learning].

Allowed Collaboration

When completing homeworks for CS1331 you may talk with other students about:

- What general strategies or algorithms you used to solve problems in the homeworks
- Parts of the homework you are unsure of and need more explanation
- Online resources that helped you find a solution
- Key course concepts and Java language features used in your solution

You may **not** discuss, show, or share by other means the specifics of your code, including screenshots, file sharing, or showing someone else the code on your computer, or use code shared by others.

Examples of approved/disapproved collaboration:

- **approved**: "Hey, I'm really confused on how we are supposed to implement this part of the homework. What strategies/resources did you use to solve it?"
- **disapproved**: "Hey, it's 10:40 on Thursday... Can I see your code? I won't copy it directly I promise"

In addition to the above rules, note that it is not allowed to upload your code to any sort of public repository. This could be considered an Honor Code violation, even if it is after the homework is due.

Turn-In Procedure

Submission

To submit, upload the files listed below to the corresponding assignment on Gradescope:

- StarterUpper.java
- StartUpIdea.java
- FileUtil.java
- Any other files that are necessary for your program to compile and run
- Any extra credit related txt files

Make sure you see the message stating "HW08 submitted successfully". From this point, Gradescope will run a basic autograder on your submission as discussed in the next section.

You can submit as many times as you want before the deadline, so feel free to resubmit as you make substantial progress on the homework. We will only grade your last submission: be sure to **submit every file each time you resubmit**.

Gradescope Autograder

For this assignment, the autograder will merely check that your code compiles and is named correctly (along with the usual checkstyle and collaboration statement checks). You will receive a 0 if your code does not compile, as always, but otherwise the homework will be manually graded.

Important Notes (Don't Skip)

- Non-compiling files will receive a 0 for all associated rubric items
- Do not submit `.class` files
- Test your code in addition to the basic checks on Gradescope
- Submit every file each time you resubmit
- Read the "Allowed Imports" and "Restricted Features" to avoid losing points
- Check on Piazza for a note containing all official clarifications