L19 Homework: 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.createx.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 have JavaFX working. This semester's HW0 went over the JavaFX verification, so if JavaFX isn't working, you should revisit HW0.

As part of your solution, you'll be provided two files, which **you MUST submit alongside any classes you write** (the autograder won't provide them as some can be changed for extra credit):

- 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, 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:

- It needs the JavaFX application `start` method. You don't need a main method; if it's present, it must do only launch(args): the autograder won't call your main method.
- The title of the window must be "Problem Ideation Form". **Any submissions without this title will not be considered**. This is automatically enforced by the autograder.
- It must have one label (javafx.scene.control.Label) with your initials (ex: O.O.) anywhere in the main window. **Any submissions without this label will not be considered**. This is **NOT** automatically enforced by the autograder, but it must be present for your submission to be considered (**it has to appear in the scene render provided in Gradescope**).
- 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 use
        - 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 empty, or out of appropriate bounds for integer values, then the submit button should create a popup alert (use the Alert class in JavaFX) with a helpful message instead of adding it to the list. The error message doesn’t have to be specific about what caused the error, but it needs to mention that at least one value is invalid.
            - For the String inputs, please do not prohibit anything other than an empty field. For quick testing, we may put integer values in String fields, and those should be deemed valid. For example, “1” is a perfectly valid input to the problem field
    - 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.
        - The provided compareTo already sorts them in the correct way (so that the first one in the list is the best), but if you change it for extra credit you need to make it is consistent with that behavior
    - 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 (use the Alert class in JavaFX) 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, and selecting no stops the reset operation
    - Button that just has your GTID on it. **You must have this button in the main window to have your submission considered**.
    - Button that saves all the added Ideas to ideas.txt
        - 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.
            - If you use LinkedList, use Java’s. Don’t use your LinkedList from the previous homework.

Tips and Tricks

- The Java API is your friend. Use it to your advantage.
- 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!
    - If some of your methods are too long, you might have to split them into smaller ones

Testing Your Solution

You can launch and use your JavaFX application through the command line. Make sure to use the correct JDK (Zulu JDK).

Rubric

[100] StarterUpper.java

- [15] Required components in any valid submission
    - [5] Application runs and has correct Stage title
    - [5] Label with initials
    - [5] Button with GTID
- [15] Entry Fields
- [50] Buttons
    - [10] Add idea button
        - [2.5] Handles Input Validation
        - [2.5] Has an Alert for invalid Input
        - [2.5] Clears fields after success
        - [2.5] Adds idea to list
    - [10] Sort Ideas Button
    - [15] Reset Button
        - [5] Has Alert
        - [2.5] Deletes existing ideas.txt when appropriate
        - [2.5] Clears list when appropriate
        - [5] Clears text in fields when appropriate
    - [15] Save Button
- [10] At least one anonymous inner class
- [10] At least one lambda expression

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 **60 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 60 points.

To earn extra credit, you should indicate which additional extra credit options you chose to do in a file named **ECDeclaration.txt**. If you don't submit this file, we may skip grading of your extra credit.

- 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: 15-30 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 6 points for each problem idea/analysis that applies to both students and non-students and up to 3 points for those that apply to just students.
    - If you have time, here's an essay on startup ideas that you may read: http://www.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**.
        - The problem field should have at least 3 sentences and no more than 5. It should be a problem that can reasonably be solved with current technology, and, while there might be some competition, it must be an idea that isn't about a problem "completely solved" (example: an alarm clock wouldn't be valid as it addresses a problem that is already

"completely solved").
        - Target customers can be a description of the population that could benefit, but the list of competitors must be a list of specific companies and/or products that compete.
        - Indicate which population your problem applies to at the end of the problem field
            - Separate the problem description and this information with a dash, such as "[your problem description] – BOTH" or "[your problem description] – STUDENTS"
        - For reference, you should spend 1-2 hours on each idea. We reserve the right to remove ideas that we deem too basic or not well-researched, at our discretion.
    - 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: 10 points**
    - Should populate that specific problem into the form as a new idea is added.
    - Hint: there are many ways to do this, but take a look at ObservableList and
        - 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 edit 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 to be awarded
- Implement a way to remove specific ideas
    - **Awards: 4 points**
    - Implementation up to you but should be apparent or explained in ECComments.txt
- Add a picture
    - **Awards: 3 points**
    - Implementation up to you, but the image should be visible on the main window, and should come from a web URL (starting with **https://**)
- Add some color to the form
    - **Awards: 2 points**

    - Implementation up to you, but the color should be visible on the main window or on one of its controls

Allowed Imports

You can import almost anything from the Java API! You can import from anywhere except from these packages and their subpackages:

- java.awt
- javax.swing
- sun

Feature Restriction

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

- **var** (the reserved keyword)

Checkstyle

The Checkstyle deduction limit for this assignment is 40 points, counting Javadoc errors. Refer to the course guide for the use of Checkstyle and review the autograder Checkstyle report on your submissions.

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].

If the autograder encounters issues processing your collaboration statement, begin it with "Collaboration Statement: ".

Allowed Collaboration

When completing homeworks for CS 1331 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

Upload the files listed below to the corresponding assignment on Gradescope:

- FileUtil.java (provided)
- StartUpIdea.java (provided, but may be modified for extra credit)
- StarterUpper.java
- ECDeclaration.txt (if you have any extra credit)
- ECComments.txt (if you need to add any comments about your extra credit for grading)
- ECIdeas.txt (if you are doing the first extra credit item. The format must be the same as the one for an ideas.txt file)

For each submission, you will be able to see the results of a few basic test cases on your code. Each test typically corresponds to a rubric item, and the score returned represents the performance of your code on those rubric items only. If you fail a test, you can look at the output to determine what went wrong and resubmit once you have fixed the issue.

Important Notes

- Non-compiling files will receive a 0 for all associated rubric items
- Do not submit .class files or .jar 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