Homework 09 – To-Do List

Authors: Vinayak, Alexander, Joshua, Daniel, Helen
Topics: Java FX, Event Driven Programming

Problem Description

Please make sure to read the document fully before starting!

You have been aiming to manage your time in a more efficient manner as the semester ends, and thus you aim to use the Object-Oriented Programming skills you have learned in this course to make a To-Do List using JavaFX.

JavaFX Installation Directions

JavaFX installation instructions can be found in the Modules section on Canvas.

To compile a JavaFX program, run this command:

```
javac --module-path javafx-sdk-11.0.2/lib/ --add-
modules=javafx.controls ClassName.java
```

To run a JavaFX program, run this command:

```
java --module-path javafx-sdk-11.0.2/lib/ --add-
modules=javafx.controls ClassName
```

Modify the module path if your javafx download is not located in the same folder.

Solution Description

For this assignment, you will create a JavaFX Application named `ToDoList.java`.

IMPORTANT: You may NOT use FXML or any of the FXML libraries. The only modules you are allowed to include are javafx.controls and javafx.media

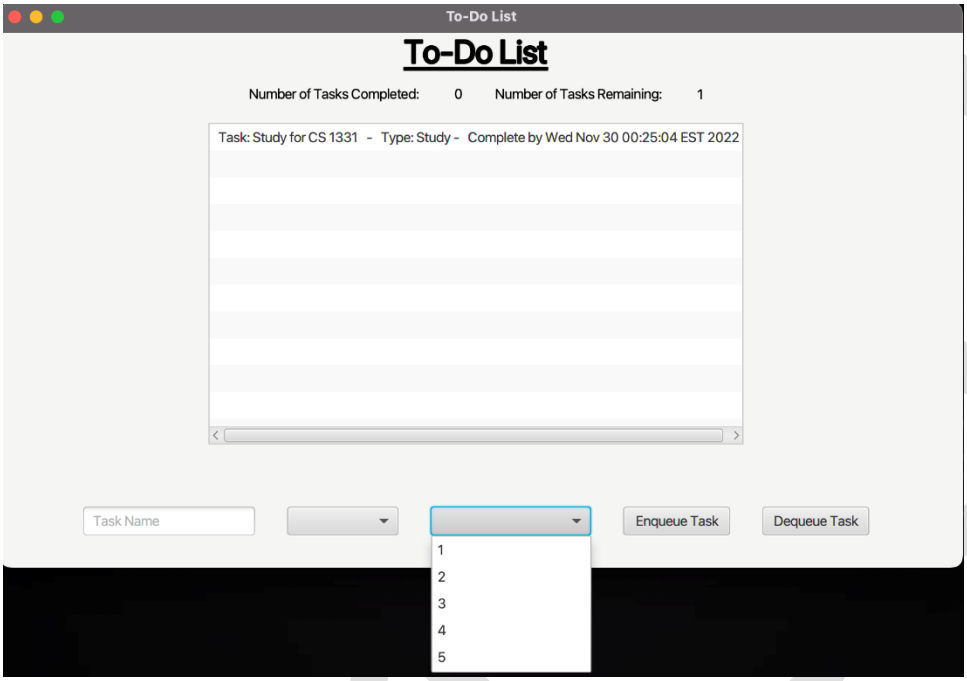
The class must meet the following requirements:

- Create a window with the title "To Do List." The window should be sized appropriately, such that contents within the window are not cut off.
- Use at least **one anonymous inner class** and **one lambda expression** in your implementation.
- Create a **title** for the page. The title should be visible at the top of the screen.
- Indicate the **number** of tasks remaining and **number** of tasks completed on the screen.
    - The number of tasks remaining is the number of tasks on the list.
    - The number of tasks completed is the number of tasks dequeued from the list.
    - These numbers should be updated when needed.
- Include an **input** section, which will be an area for the user to input the name of the task as well as the number of hours it will take to complete the task.

THIS GRADED ASSESSMENT IS NOT FOR DISTRIBUTION.
ANY DUPLICATION OUTSIDE OF GEORGIA TECH'S LMS IS UNAUTHORIZED.

  - There should be a **text box** for the user to input the name of the Task.
  - There should be another control for the user to choose the number of hours it will take to complete the task, where one can choose an integer in the range [1,5].
  - There should be another control for the user to choose the type of Task that one will be doing. The user should be able to choose one of four options: [Study, Shop, Cook, Sleep].
- Include a **list** section to display the current list of tasks and number of hours it takes to complete them.
  - This should display a list of tasks, with each task including:
    - The name of the task
    - The type of the task
    - The date and time the task is due (calculated by summing the current time and the inputted time it will take the user to do the task)
  - New tasks should be added to the bottom of the list.
  - The user should be able to continuously add tasks to the list (the list should be expanded using a scroll bar if the number of tasks exceeds the visible list length)
    - **HINT**: A ListView object may be very useful here!
- Include some **buttons** that will be used to enqueue and dequeue the task.
  - One button should be used to **enqueue** a task.
    - If the task text box is blank *and/or* the user hasn't selected a number of hours, **alert** the user to fill in the appropriate text fields.
    - If the task **already exists** (identical task names and number of hours), **alert** the user that the task is already added.
    - If the task name **already exists** (but the number of hours is different), update the number of hours of the respective task.
    - Once a task has been added to the list, the task name text box and number of hours should be cleared.
  - One button to **dequeue** a task.
    - The oldest task (at the top of the list) should be removed from the list.

Below is an Example of how the To-Do list **may** look:



Tips and Tricks

- The Java API is your friend. Use it to your advantage.
- Make sure you can properly run your JavaFX programs before starting this assignment. Do **NOT** wait until the last minute to configure your JavaFX!
- Work incrementally. Get a basic UI up and running. Add buttons, cells for display, etc., piece by piece. Think of what layout manager(s) you want to use.
- The pillars of Object-Oriented Programming should be particularly useful for this Homework. Try and use Inheritance, Polymorphism, Encapsulation, and Abstraction to your advantage to simplify and organize your code. This will make it much easier to understand and pinpoint any issues and will make your code much more readable.

Extra Credit Opportunities

As this homework is relatively open-ended, there will be up to **30 bonus points available** for expressing creativity through your implementation.

NOTE: If you decide to do extra credit, also submit an **`extraCredit.txt`** file detailing what extra credit you decided to implement (a few sentences or so). We **WILL NOT grade extra credit** if a .txt file is not provided!!!

If your code requires a different command to compile and run (i.e., using different javafx modules), then put the command in the extraCredit.txt.

All these items are subjectively graded. Additions will be considered under the following categories. You may earn points under a certain category multiple times:

THIS GRADED ASSESSMENT IS NOT FOR DISTRIBUTION.
ANY DUPLICATION OUTSIDE OF GEORGIA TECH'S LMS IS UNAUTHORIZED.

- 5 : Non-trivial enhancement to graphics of the program
- 5-15 : Non-trivial enhancement to the controls of the program
- 5-15 : Non-trivial new feature added

Some examples include, but are certainly not limited to:

- Adding animations, sound effects, or backgrounds to the program
- Adding specific images/icons regarding the corresponding difficulty of a task depending on how many hours the task takes to complete
- Adding another section to differentiate between the tasks that are completed and the tasks that are in-progress and adding buttons to move specific tasks between sections.
- An impressive, simply amazing submission that blows the TAs away (We've seen some students do really cool things before, just run away with the assignment and make it awesome)
- **Extra credit features should not affect any of the base functionality!**

Feel free to be creative; these are just a few ideas!

Checkstyle

You must run Checkstyle on your submission (To learn more about Checkstyle, check out cs1331-style-guide.pdf under CheckStyle Resources in the Modules section of Canvas.) **The Checkstyle cap for this assignment is 40 points.** This means there is a maximum point deduction of 40. If you don't have Checkstyle yet, download it from Canvas -> Modules -> CheckStyle Resources -> checkstyle-8.28.jar. Place it in the same folder as the files you want to run Checkstyle on. Run Checkstyle on your code like so:

```
$ java -jar checkstyle-8.28.jar yourFileName.java
Starting audit...
Audit done.
```

The message above means there were no Checkstyle errors. If you had any errors, they would show up above this message, and the number at the end would be the points we would take off (limited by the checkstyle cap mentioned in the Rubric section). Note that you can ignore any Checkstyle errors in code we provide you, where applicable.

Additionally, you must Javadoc your code.

Run the following to only check your Javadocs:

```
$ java -jar checkstyle-8.28.jar -j yourFileName.java
```

Run the following to check both Javadocs and Checkstyle:

```
$ java -jar checkstyle-8.28.jar -a yourFileName.java
```

For additional help with Checkstyle see the CS 1331 Style Guide.

Turn-In Procedure

Submission

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

- `ToDoList.java`
- Any additional files needed for your code to compile
- `extraCredit.txt` if you did extra credit

Make sure you see the message stating the assignment was submitted successfully. From this point, Gradescope will run a basic autograder on your submission as discussed in the next section. **Any autograder test are provided as a courtesy to help "sanity check" your work and you may not see all the test cases used to grade your work.** Autograder tests are **NOT** guaranteed to be released when the assignment is released, so ***YOU*** are responsible for thoroughly testing your submission on your own to ensure you have fulfilled the requirements of this assignment. If you have questions about the requirements given, reach out to a TA or Professor via our class discussion forum for clarification.

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 latest submission. **Be sure to submit every file each time you resubmit**.

Gradescope Autograder

If an autograder is enabled for this assignment, you may be able to see the results of a few basic test cases on your code. Typically, tests will correspond 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.

The Gradescope tests serve two main purposes:

- Prevent upload mistakes (e.g. non-compiling code)
- Provide basic formatting and usage validation

In other words, the test cases on Gradescope are by no means comprehensive. Be sure to thoroughly test your code by considering edge cases and writing your own test files. You also should avoid using Gradescope to compile, run, or Checkstyle your code; you can do that locally on your machine. Other portions of your assignment can also be graded by a TA once the submission deadline has passed, so the output on Gradescope may not necessarily reflect your grade for the assignment.

Allowed Imports

- Any imports from any package that starts with `java` or `javafx` that do not belong to the AWT or Swing APIs.
- Remember that JavaFX is different that AWT or Swing. If you are trying to import something from javax.swing or java.awt, you are probably importing the wrong class.
- **NO FXML IS ALLOWED TO BE USED TO COMPLETE THIS ASSIGNMENT.**

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`
- `System.arraycopy`

Collaboration

Only discussion of the Homework (HW) at a conceptual high level is allowed. You can discuss course concepts and HW assignments broadly, that is, at a conceptual level to increase your understanding. If you find yourself dropping to a level where specific Java code is being discussed, that is going too far. Those discussions should be reserved for the instructor and TAs. To be clear, you should never exchange code related to an assignment with anyone other than the instructor and TAs.

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 Ed Discussion for a note containing all official clarifications and sample outputs**

It is expected that everyone will follow the Student-Faculty Expectations document, and the Student Code of Conduct. The professor expects a **positive, respectful, and engaged academic environment** inside the classroom, outside the classroom, in all electronic communications, on all file submissions, and on any document submitted throughout the duration of the course. **No inappropriate language is to be used, and any assignment, deemed by the professor, to contain inappropriate, offensive language or threats will get a zero**. You are to use professionalism in your work. Violations of this conduct policy will be turned over to the Office of Student Integrity for misconduct.