Homework 7 – Graphics and a Food Truck through JavaFX

Authors: Lucas, Andrew

Problem Description

As the semester starts coming to a close, you look for different outlets to de-stress and express yourself. You take up painting but quickly realize your artistic skills are… not as good as your programming skills. As a result, you decide to use JavaFX as your creative outlet. Hopefully by putting together different shapes and colors you will make something that is more than the sum of its parts.

Additionally, with all this JavaFX practice you get, you get offered a job programming the GUI for the CS1331 Food Truck! You are tasked with programming some base functionality that will be expanded upon in the coming weeks by your co-workers. You'll be providing future customers of the CS 1331 Food Truck with the ability to select how many of each item they want to purchase and you'll give them a receipt of their purchase.

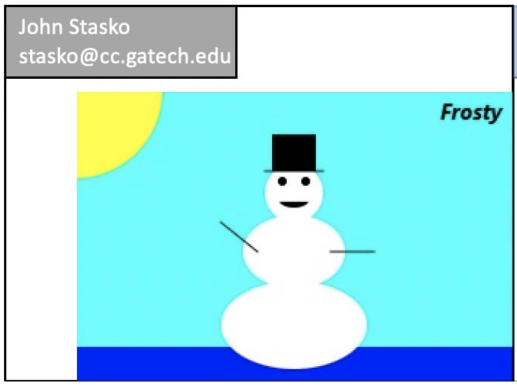
This assignment will help you learn about JavaFX, exceptions, and file I/O. You will have to use JavaFX to implement basic GUI elements, such as buttons and textboxes, and keep track of the current state of the food truck. You will have to create one checked exception and one unchecked exception that can be thrown by your program. You also will need to write some results to a file. The visual and functional requirements are described in more detail below.

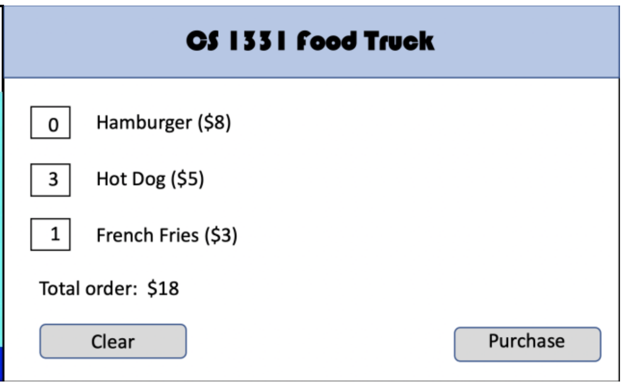
The JavaFX application you write will have two windows. In the first window, you will just be doing graphics calls. In one corner of the window, you should write your name and email address. Elsewhere in the window, draw some simple scene using line, rectangle, ellipse, text, etc., calls, kind of like we did in class for the Snowman example. Make sure to do your picture with individual graphical object calls, though, do not simply import an image for this part. And of course, just don't draw a snowman again. Think of something else creative.

In the second window of the application, you'll be creating a simple CS 1331 Food Truck interactive application where users can select multiple items and then make a purchase. You should have at least three food items, all with different costs. The user can provide a count of how many of each item that they want to order. The current cost for the active selections should always be shown in the UI. The interface also should contain a "Clear" button that, when pressed, sets the quantity of all items back to zero. Finally, when a purchase is made, the total order and its cost should be printed out to a file named "order.txt".

You should call your entire program FoodTruck.java. It should have two top-level windows.

Example Submission





Graphics window FoodTruck window

Graphics window visual requirements

These are the visual components your drawing window must have:

- **Shapes:** Create some graphical picture using at least 4 distinct shapes and 4 distinct colors.
- **Name and email:** Include your name and email address as text.
- **Do NOT** use images in this Java file.

Food Truck window visual requirements

These are the visual components your food truck window must have:

- **Header:** There must be a header on top of your program with text that says "CS 1331 Food Truck"
- **Three Items:** There must be at least three items available for purchase. An item consists of a text entry field where a quantity can be entered and some text next to it with the name and price of the item.
- **Current Cost Text:** There should be text output constantly visible and updated for the current total purchase cost given the items counts and costs.
- **Purchase Button:** There must be a purchase button that can be pressed to purchase the items and print an order summary to the order.txt file. Simply list the items and quantities being purchased along with the total cost.
- **Clear Button:** When this is pressed, the quantity should be set back to zero for all items.

Food Truck Functional Requirements

These requirements detail how the food truck must work along with implementation requirements.

- All quantity entry fields should be set to 0 when the program starts.
- When an item quantity is changed, the current cost label should be updated.
- Pressing the Clear button sets all item quantities back to 0 and the total cost back to $0.
- Upon pressing the purchase button:
    - Throw an `IllegalQuantityException` if any of the user entered quantities are not numeric. Handle the exception, do not purchase the items, and alert the user with a pop-up.
    - Throw a `NegativeQuantityException` if any of the user entered quantities are numeric, but not greater than or equal to 0. Handle the exception, do not purchase the items, and alert the user with a pop-up.
    - If all quantities are valid, print a receipt to the file "order.txt" with a summary of the order. The name of each item and its quantity should be printed with the total amount spent at the end of the receipt. If the quantity for an item is 0, it should not be included on the receipt.
- In implementing these functionalities, use at least one anonymous inner class and at least one lambda expression.

NegativeQuantityException.java

This should be an **unchecked** exception (be sure to extend the correct class). It should have a one-arg constructor that takes in the String attempted to be used as a quantity that sets the error message that would be displayed on the console if the exception was not caught to "Quantity <argument> is not positive."

IllegalQuantityException.java

This should be a **checked** exception (be sure to extend the correct class). It should have a one-arg constructor that takes in the String attempted to be used as a quantity that sets the error message that would be displayed on the console if the exception was not caught to "Quantity <argument> is not a number."

Optional Extra Credit Opportunity

Up to 25 points of extra credit can be earned on this assignment through non-trivial improvements to the GUI or functionality of either the drawing (5 points) or food truck (20 points). If you choose to participate in this, you must submit a file in addition called **extra_credit.txt** that briefly explains the improvements you made (just a few sentences per feature should suffice).

Examples of extra credit on the graphics window include:

- Especially nice pictures and scenes (subjective judgment on our part)
- Adding special effects like animation

Examples of extra credit on the food truck window include:

- Additional functions like adding condiments (checkboxes)
- Additional further GUI widgets and UI functionality
- Making the look-and-feel of the UI enhanced through color and layout
- Adding sounds when buttons are pressed
- Adding images to represent the items

Feel free to be creative on this part, but make sure that your extra credit additions do not wipe out the core functionality of the application. You must provide those base functions.

Tips

You can implement the store GUI however you want as long as it meets the requirements listed above. However, here are some tips and suggestions that might help.

Start this HW in slow steps. First get windows popping up. Then add a little functionality, and then a little more, and so on. Work very incrementally. That will make the task much more manageable. Also, leverage the example programs shown in class that were put into Canvas. They contain lots of useful little snippets of code. Furthermore, leverage online resources like the API and things like Stack Overflow. JavaFX has lots of classes and functions, so feel free to look up how they work or how to do some *specific* thing (ex. "How do you center text in a Label in JavaFX?"). However, it is not allowed to take code directly from online sources; the same policies apply to this homework as have applied to every other assignment for this course. Online sources can help with the syntax of specific classes and method calls, but plagiarism is not allowed.

Before thinking about implementation, think about how you want to lay out your components. Draw it out by hand and decide what kinds of layout panes you want the visual components in and how to nest those panes inside each other. Using a BorderPane as the outermost, or "root," pane is great way to organize the outer structure of a GUI window. VBox and HBox panes are also very helpful within that.

In order to update the information on the screen, having all of the labels, buttons, etc. declared as instance variables in the FoodTruck class could also be helpful. Then, if you want to change the text of a component, you can use the following: variableName.setText("New Text Example"). You can change pretty much anything about a component like this. To check the text of a component, you can use the getText() method. This returns a String containing the component's text.

To space things out, you can use Insets. After importing javafx.geometry.Insets, you can set a pane's padding by calling paneName.setPadding(new Insets(10, 10, 10, 10)). You can change the values to adjust the spacing.

Finally, methods are your friends! You can create methods to purchase items, etc. and then easily call them.

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

- `FoodTruck.java`
- `IllegalQuantityException.java`
- `InsufficientBalanceException.java`
- `extra_credit.txt (if you are attempting to fulfill the extra credit requirements)`

Checkstyle and Javadoc
You must run Checkstyle on your submission. The Checkstyle cap for this assignment is **20 points**. If you don't have Checkstyle yet, download it from Canvas -> Files ->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 above). The Java source files we provide contain no Checkstyle errors. In future homeworks we will be increasing this cap, so get into the habit of fixing these style errors early!

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
```

Import Restrictions
To prevent trivialization of the assignment, you may only import:

- `java.util.ArrayList`
- `java.io.File`
- `java.io.PrintWriter`
- `java.io.FileNotFoundException`
- `java.io.IOException`
- `java.util.Scanner`
- Anything from `javafx`

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`

Collaboration

No collaboration is allowed on this assignment. See syllabus for more details.

In addition, 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. Only post code on Ed Discussion in a **private** post.

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 "Import Restrictions" and “Feature Restrictions" to avoid losing points
- Check on Ed Discussion for all official clarifications
- Make sure to test your program manually based on the assignment instructions and expected outputs.
- The Gradescope autograder visible to you is **NOT** comprehensive. A few test cases will be hidden when you submit. This is done intentionally so that you learn to write your own test cases for your assignments.