Bringing It Together - Mav’s Ice Cream Emporium (Sprint 3)

Due Tuesday, October 18 at 8 a.m.

CSE 1325 - Fall 2022 - Homework #7 / Sprint 3 - 1

Assignment Background

Mav’s Ice Cream Emporium (MICE) is a fledgling start up in the dairy treat market. They are seeking bright young programmers to build a custom solution for defining new confections, tracking customers and the treats they buy, ensuring timely delivery of a quality product, and otherwise taking care of business. Your job is to win this project (with associated profit and glory) from Mav’s Ice Cream Emporium by producing a proposal package over 6 Sprints, including a prototype that wows and other creative and persuasive artifacts that prove you know your stuff.

This is Sprint 3 of 6, in which we add saving and loading our data, a snazzier interactively drawn logo, and (if you didn't complete it in Sprint #2) a toolbar!

I insist that you refer to E2_Handout.pdf while writing this assignment, as that's all you'll have on Exam #2!

The Scrum Spreadsheet

Copy your P06 directory completely to P07, including last week's Scrum spreadsheet. The Product Backlog has changed only *slightly* for this sprint - replace "as splash screen" in row 30 (Feature ID "LOGO") cell I (under "I want to...") with "in the About dialog", and delete "; a JWindow is probably ideal here" in column K (under "Notes").

As always, update your Product Backlog tab to reflect your plan and your actual work for Sprint 3. Then, on the Sprint 3 Backlog tab, plan the tasks you'll need to implement both features (the hints below should help) and update your status as you go.

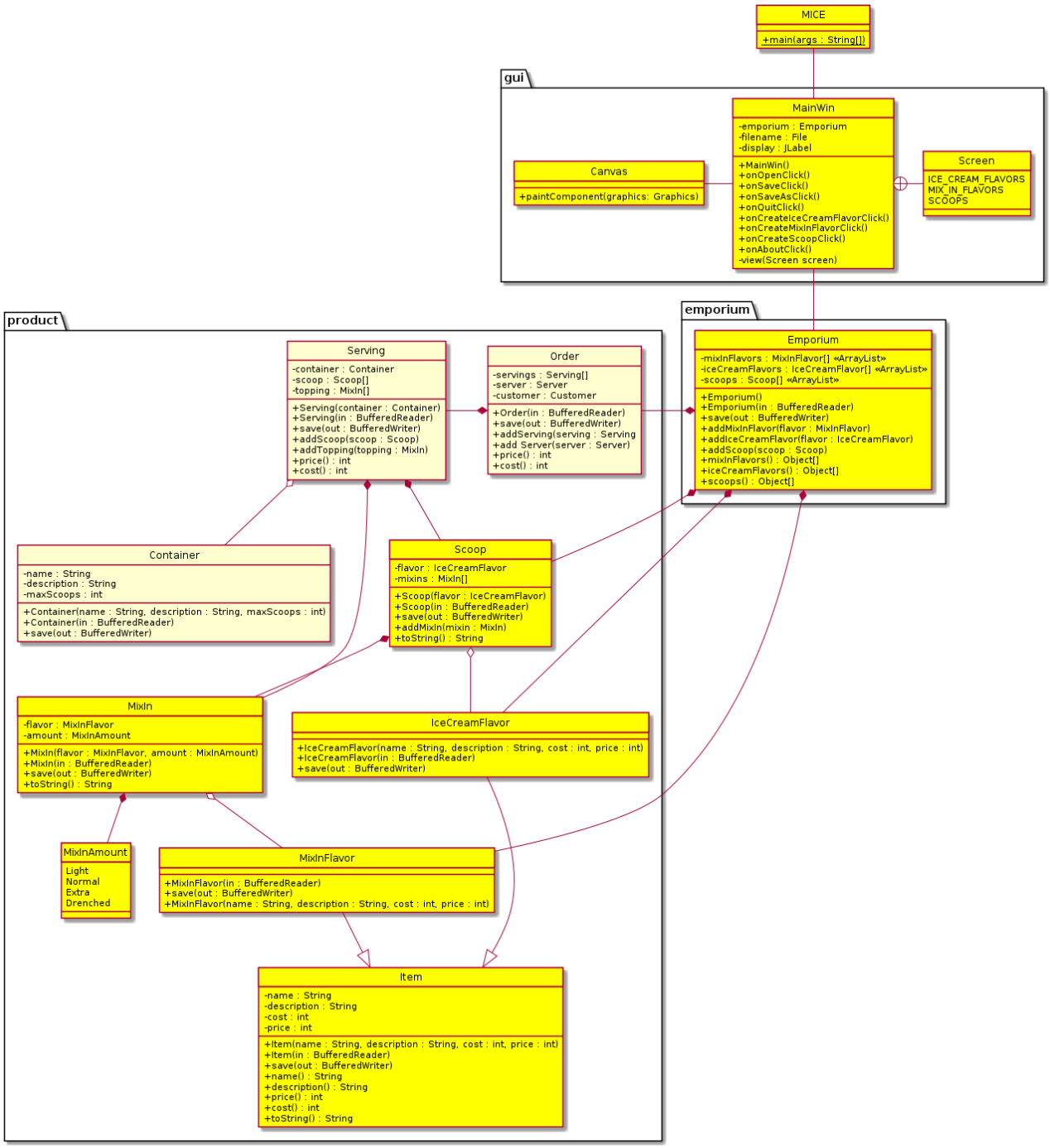
Be sure to add, commit, and push the updated spreadsheet at cse1325/P07/Scrum_MICE.xlsx before the end of the sprint!

Class Diagram

The diagram on the next page has been updated to refect the new features in this sprint. Note the addition of a save method (with a BufferedWriter parameter) and a constructor (with a BufferedReader parameter) for each class containing data - this is the encapsulation-friendly approach to File > Save and File > Open as we discussed in Lecture 14. Also note the new Canvas class for the fancy About dialog logo. The toolbar feature doesn't affect the class diagram.

As with Sprint 2 you will only be implementing the dark yellow classes this Sprint.

As before, updates to fix bugs and design issues are all but inevitable.


MICE
+main(args : String[])
gui
MainWin
-emporium : Emporium
-filename : File
-display : JLabel
+MainWin()
+onOpenClick()
+onSaveClick()
+onSaveAsClick()
+onQuitClick()
+onCreateIceCreamFlavorClick()
+onCreateMixInFlavorClick()
+onCreateScoopClick()
+onAboutClick()
-view(Screen screen)
Canvas
+paintComponent(graphics: Graphics)
Screen
ICE_CREAM_FLAVORS
MIX_IN_FLAVORS
SCOOPS
emporium
Emporium
-mixInFlavors : MixInFlavor[] «ArrayList»
-iceCreamFlavors : IceCreamFlavor[] «ArrayList»
-scoops : Scoop[] «ArrayList»
+Emporium()
+Emporium(in : BufferedReader)
+save(out : BufferedWriter)
+addMixInFlavor(flavor : MixInFlavor)
+addIceCreamFlavor(flavor : IceCreamFlavor)
+addScoop(scoop : Scoop)
+mixInFlavors() : Object[]
+iceCreamFlavors() : Object[]
+scoops() : Object[]
product
Serving
-container : Container
-scoop : Scoop[]
-topping : MixIn[]
+Serving(container : Container)
+Serving(in : BufferedReader)
+save(out : BufferedWriter)
+addScoop(scoop : Scoop)
+addTopping(topping : MixIn)
+price() : int
+cost() : int
Order
-servings : Serving[]
-server : Server
-customer : Customer
+Order(in : BufferedReader)
+save(out : BufferedWriter)
+addServing(serving : Serving
+add Server(server : Server)
+price() : int
+cost() : int
Container
-name : String
-description : String
-maxScoops : int
+Container(name : String, description : String, maxScoops : int)
+Container(in : BufferedReader)
+save(out : BufferedWriter)
Scoop
-flavor : IceCreamFlavor
-mixins : MixIn[]
+Scoop(flavor : IceCreamFlavor)
+Scoop(in : BufferedReader)
+save(out : BufferedWriter)
+addMixIn(mixin : MixIn)
+toString() : String
MixIn
-flavor : MixInFlavor
-amount : MixInAmount
+MixIn(flavor : MixInFlavor, amount : MixInAmount)
+MixIn(in : BufferedReader)
+save(out : BufferedWriter)
+toString() : String
IceCreamFlavor
+IceCreamFlavor(name : String, description : String, cost : int, price : int)
+IceCreamFlavor(in : BufferedReader)
+save(out : BufferedWriter)
MixInAmount
Light
Normal
Extra
Drenched
MixInFlavor
+MixInFlavor(in : BufferedReader)
+save(out : BufferedWriter)
+MixInFlavor(name : String, description : String, cost : int, price : int)
Item
-name : String
-description : String
-cost : int
-price : int
+Item(name : String, description : String, cost : int, price : int)
+Item(in : BufferedReader)
+save(out : BufferedWriter)
+name() : String
+description() : String
+price() : int
+cost() : int
+toString() : String

Add a Toolbar

(Feature TOOLB) Add a toolbar. You may baseline the code from Lecture 14's Nim example without attribution.

Include the following button groups. Note that you won't implement the listeners for the first three buttons until the next feature - just leave them empty. The last six buttons use existing listeners from your menu.

- (onSaveClick()) Save to write all data to the current filename.
- (onSaveAsClick()) Save As to change the current filename via a FileChooser dialog and then chain to Save.
- (onOpenClick()) Open to select a filename via a FileChooser dialog and then create a new Emporium from it, changing the current filename if successful.

- (onCreateIceCreamFlavorClick()) Create a new ice cream flavor.
- (onCreateMixInFlavorClick()) Create a new mix in flavor.
- (onCreateScoopClick()) Create a new scoop.

- (onViewIceCreamFlavorClick()) View all ice cream flavors in the main window.
- (onViewMixInFlavorClick()) View a new mix in flavors in the main window.
- (onViewScoopClick()) View a new scoops in the main window.

Be sure to document your button icons in your About dialog, even if you draw the icons yourself (your lawyer will thank me later). If you use icons from a source such as flaticon.com, freeicons.io, icons8.com, or thenounproject.com, be sure to follow the license. As long as you make a good faith effort to do so, you won't lose any license compliance points!

Save and Open an Emporium

(Features SAVD, LALL, SALL) Add the ability to save the emporium to the current or a newly selected filename and to open an emporium from a selected filename.

MainWin Members

Remember from Lecture 14 that MainWin creates the file stream itself based on the user-selected filename, and passes the stream to the model (in our case, the Emporium object and the objects it aggregates). The model then either writes each object's data to the stream created by MainWin, or recreates its objects from that stream.

To simplify parsing, write each value to a separate line (that is, add a newline where needed), and read entire lines using BufferedReader's readLine method.

I recommend that you baseline the onOpenGameClick (renamed as onOpenClick), onSaveGameClick (renamed as onSaveClick), and onSaveGameAsClick (renamed onSaveAsClick) from Nim's MainWin in Lecture 14.

- Use whatever file extension you like.

- Add a field name `filename` of type `File`, and initialize it to a new File object. The File constructor parameter is your default filename, traditionally named `untitled` with your chosen extension.

- You may keep or omit final fields such as NAME, VERSION, FILE_VERSION, and MAGIC_COOKIE (these are defined in the Nim baseline but are NOT required).
- You don't need to handle the computerPlayer toggle button (if you're baselining Nim), so delete that code.
- Be certain to update your data area after opening the file. (If you're baselining Nim, replace setSticks() with a call to your MainWin.view method.)
- **From each active method, call the appropriate Emporium method. onSaveAsClick delegates to onSaveClick, so it doesn't call Emporium directly.**
    - onOpenClick will invoke `new Emporium`, passing a BufferedReader object that streams from the selected file.
    - onSaveClick will call `Emporium.save`, passing a BufferedWriter object that streams to the selected file.

Data Classes

For all classes containing data fields (even if only inherited), you need to implement a save method (to write out your data) and a constructor (to read in your data). We covered this in Lecture 14.

Let's take MixIn as an example. It has two fields, `flavor` and `amount`.

In the save method, stream out each field.

- flavor is of type MixInFlavor, which is a class. So just delegate to that class, e.g., `flavor.save(bw);`
- amount is an simple enum. If it had additional members, we would just treat it as a class. But a simple enum with no additional members can be streamed out by simply writing its toString value to the BufferedWriter stream followed by a newline.

In the constructor, stream in each field.

- flavor is created in the usual way using the new keyword and passing the BufferedReader as the constructor parameter.
- amount, our simple enum, can be recreated using MixInAmount.valueOf with the String read from the BufferedReader as parameter.

Don't forget that each data class' constructor and your save method must both be declared as `throws IOException`, because BufferedReader.readLine and BufferedWriter.write may throw IOException and IOException is "checked" (that is, Java requires that you warn others that you might throw it). Of course, any other constructors that do no I/O do NOT need to use a throws clause.

Also don't forget that if your *default* constructor initializes a field (such as an array list), then your *BufferedReader* constructor must do the same. Otherwise, you will likely get a NullPointerException.

Data Subclasses

Our superclass Item is handled exactly like any other data class.

Our subclasses, IceCreamFlavor and MixInFlavor, can then simply chain to the superclass constructor and the superclass save method.

ArrayList Types

Saving an array list is accomplished by first saving the size (so the constructor knows how many objects to reconstruct), and then saving each object in the array list.

To reconstruct the array list in the constructor, first read the size, then loop and add that many objects to the array list using each class' constructor with the BufferedReader object as its parameter.

Class Emporium

Class Emporium has a few special considerations.

Because Emporium didn't previously have an explicit default constructor, you need to add an empty one. This is because when you define a constructor to handle a BufferedReader stream, Java will no longer provide you a default constructor "for free". (If you already declared a default constructor to initialize the ArrayList fields, then you're good to go already.)

Otherwise, you should just write out and reconstruct your mixInFlavors, iceCreamFlavors, and scoops array list fields as discussed under ArrayList Types above in your save method and constructor, respectively.

Draw a Logo

(Feature LOGO - The coding of this feature is covered in Lecture 15 on Thursday.)

Your P06 solution should have included an About dialog box (probably created with the gui.MainWin.onAboutClick() method) including a BufferedImage with your logo inside a JLabel. If not, see the suggested solution to add one.

Replace the JLabel containing your BufferedImage logo in the About dialog with a Canvas instance. Write your Canvas as a subclass of JPanel, just as we discussed in Lecture 15 (from which you may baseline the Canvas implementation without attribution if you like). The JPanel must be at least 30% larger than your logo, so that the grader can clearly see it.

Override method Canvas.paintComponent.

- Invoke the superclass constructor.
- Cast the Graphics parameter as a Graphics2D object. (You may assume it will cast successfully.)
- **Draw (using Graphics2D.drawLine) a multi-colored pattern of your choice across the JPanel.**
    - You MUST use at least 2 colors via Graphics2D.setColor method.
    - You MUST write your name somewhere in the drawing using the Graphics2D.drawString method.
- Copy your logo to roughly the center of your JPanel or another artistically-selected location (using Graphics2D.drawImage) such that your drawing and name are not obscured.

Be as artistic or as pedestrian as you like. The objective is simply to give you a little practice in drawing on a JPanel before Exam #2, where you will definitely see it again.

Screenshots

Your application is NOT required to look like this! You have significant freedom now to make this project your own, subject only to meeting the stated requirements. If you have questions as to whether your plans conform to the requirements, I'm happy to discuss with you via email or in my office.

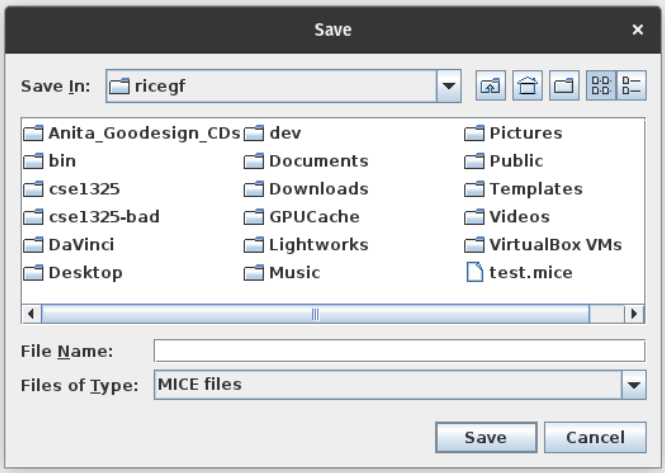

File View Create Help
Ice Cream Flavors
Open
Look In: ricegf
nita_Goodesign_CDs
in
se1325
se1325-bad
aVinci
esktop
dev
Documents
Downloads
GPUCache
Lightworks
Music
Pictures
Public
Templates
Videos
VirtualBox VMs
test.mice
File Name: test.mice
Files of Type: MICE files
Open
Cancel

File View Create Help

Scoops

Vanilla with Snickers (Extra), Peanuts

Vanilla



```
ricegf@antares:~/dev/202208/P07/tags$ cat ~/test.mice
Mïċẹ🍦🍨
1.0
2
Snickers
Delectable chunks of chewy, nutty Snickers bars
99
16
Peanuts
Chopped, salted, and delightful peanuts
69
11
1
Vanilla
Rich, creamy vanilla bean goodness!
199
32
1
Vanilla
Rich, creamy vanilla bean goodness!
199
32
2
Extra
Snickers
Delectable chunks of chewy, nutty Snickers bars
99
16
Normal
Peanuts
Chopped, salted, and delightful peanuts
69
11
```

MICE

Mavs Ice Cream Emporium

Version 0.3

Logo by Schmidsi, per the Pixabay License
https://pixabay.com/en/ice-ice-cream-cone-ice-ball-pink-1429596/
Ice cream icons created by Freepik - Flaticon
https://www.flaticon.com/free-icons/
File icons created by Pixel perfect - Flaticon
https://www.flaticon.com/free-icons/direct-download
https://www.flaticon.com/free-icons/file-upload
https://www.flaticon.com/free-icons/ui
https://www.flaticon.com/free-icons/

OK