Bringing It Together - Mav’s Ice Cream Emporium (Sprint 1)

Due Tuesday, October 4 at 8 a.m.

CSE 1325 - Fall 2022 - Homework #5 / Sprint 1 - 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 1 of 6.

The Scrum Spreadsheet

The intent of using a *very* simplified version of Scrum on this project is to introduce you to the concept of planning your work rather than just hacking code at the last minute and hoping for the best. **Professionals make a plan and then execute it.** Amateurs sling code and hope for the best.

Submitting an *accurate* spreadsheet is part of your grade for these assignments. This is not what you wished had happened - it's what actually happened. **Be certain you update the spreadsheet and add, commit, and push it at cse1325/P05/Scrum_MICE.xlsx before the end of the sprint!**

How to Read the Product Backlog

For the Product Backlog tab's list of features, read the header for columns H, I, and J right before the feature's cell. So for example, read

As a...	I want to...	So that I can...
Manager	Create a new ice cream flavor	Stock and sell dairy products

like this: "As a Manager, I want to Create a new ice cream flavor So that I can Stock and sell dairy products." That's a concise description of who wants the feature, what the feature is, and why they want it.

How to Update the Spreadsheet

You need to add, commit, and push the spreadsheet as cse1325/P05/Scrum_MICE.xlsx in addition to your usual code and build.xml.

Product Backlog

- **On the Product Backlog tab, fill in the green cells at the top and (at least) on rows 24-27 - the 4 features that will be graded this sprint. (You know because of the "1" in the Required column for each of those features.)**
    - Planned (column F) is the sprint in which you plan to implement that feature - in this case, "1".

    - Status (column G) is the sprint in which you actually implement that feature - in this case, "Finished in Sprint 1" if you complete it, "In Work" or "In Test" if you started but it's not complete yet.

Sprint 01 Backlog

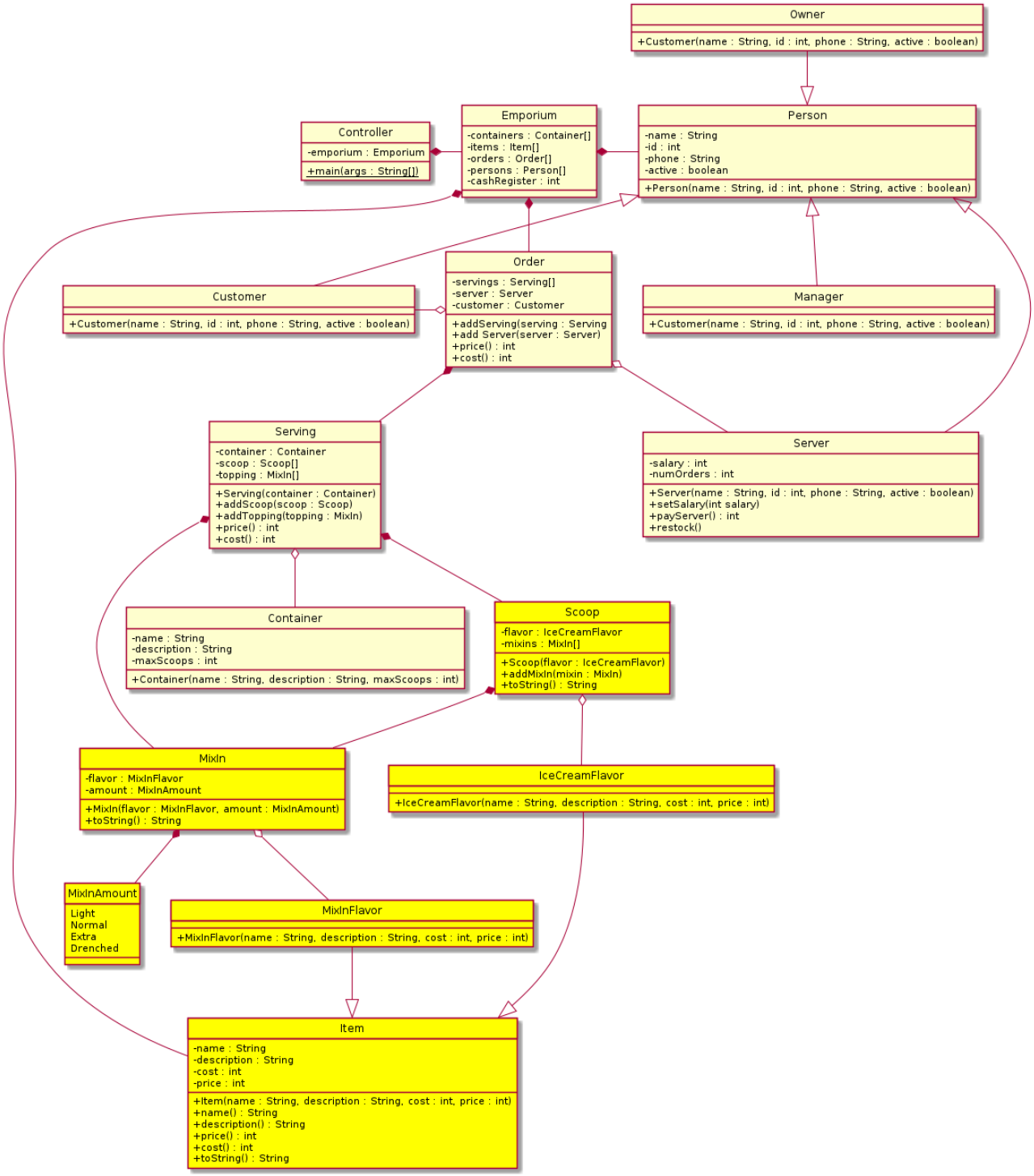
- **On the Sprint 01 Backlog tab, fill in the Feature ID (column B), Description (column D), and Status (column E), for 3 to 5 tasks associated with each feature. For example, for Feature ID "CF" (from the Product Backlog tab, cell A24), your tasks might be "Write Item superclass", "Copy build.xml into repo and test compile Item", "Write and compile IceCreamFlavor subclass", "Test IceCreamFlavor subclass via constructor and toString", and "Add IceCreamFlavor to GitHub".**
    - Select the Feature ID (column B) from the drop-down list. This matches up with column A on the Product Backlog tab. We call this "traceability", because it answers the question, "Why are you doing this task?" The answer is, "It's need to implement feature XXX".
    - Write a brief to do item under Description (column D) similar to the example above.
    - Set the Status (column E) to the day of the sprint on which you finished that task. For example, if you finished the first task on Wednesday, set Status to "Completed Day 2". If you don't complete the task, set Status to "In Work" if started or leave blank if not started. (If you don't complete all of the tasks for a specific Feature ID, then you can't set the associated Feature on the Product Backlog tab to "Finished in Sprint 1", right?)

Now add, commit, and push. That's all you need to do.

Class Diagram

The diagram on the next page covers much more than Sprint 1 (although not everything in the MICE system). It may make more sense if you read The Owner's Monologue from MICE_Project_Overview.pdf alongside it. You will only be implementing the dark yellow classes this Sprint, saving the other classes for later Sprints.

A more complete diagram will be provided once you have a basic understanding of Swing and Graphical User Interfaces. And updates to fix bugs and design issues are all but inevitable.


Owner
+Customer(name : String, id : int, phone : String, active : boolean)
Controller
-emporium : Emporium
+main(args : String[])
Emporium
-containers : Container[]
-items : Item[]
-orders : Order[]
-persons : Person[]
-cashRegister : int
Person
-name : String
-id : int
-phone : String
-active : boolean
+Person(name : String, id : int, phone : String, active : boolean)
Order
-servings : Serving[]
-server : Server
-customer : Customer
+addServing(serving : Serving
+add Server(server : Server)
+price() : int
+cost() : int
Customer
+Customer(name : String, id : int, phone : String, active : boolean)
Manager
+Customer(name : String, id : int, phone : String, active : boolean)
Serving
-container : Container
-scoop : Scoop[]
-topping : MixIn[]
+Serving(container : Container)
+addScoop(scoop : Scoop)
+addTopping(topping : MixIn)
+price() : int
+cost() : int
Server
-salary : int
-numOrders : int
+Server(name : String, id : int, phone : String, active : boolean)
+setSalary(int salary)
+payServer() : int
+restock()
Container
-name : String
-description : String
-maxScoops : int
+Container(name : String, description : String, maxScoops : int)
Scoop
-flavor : IceCreamFlavor
-mixins : MixIn[]
+Scoop(flavor : IceCreamFlavor)
+addMixIn(mixin : MixIn)
+toString() : String
MixIn
-flavor : MixInFlavor
-amount : MixInAmount
+MixIn(flavor : MixInFlavor, amount : MixInAmount)
+toString() : String
IceCreamFlavor
+IceCreamFlavor(name : String, description : String, cost : int, price : int)
MixInAmount
Light
Normal
Extra
Drenched
MixInFlavor
+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)
+name() : String
+description() : String
+price() : int
+cost() : int
+toString() : String

Product Package

Item

Yes, this is the least dependent class. Did you get that right?

This is a basic data class. Configure the toString() override to return what works best for you through the project. The suggested solution just returns the name field.

IceCreamFlavor and MixInFlavor

Yes, no members but the constructor (which delegates to the superclass).

MixInAmount

Just your standard enum. Gee, this project is going quickly, isn't it?

MixIn

This class has two private fields with a constructor to initialize them. The toString() is non-trivial, though - return the flavor followed by the amount in parentheses, but only if the amount is NOT MixInAmount.Normal.

Scoop

Scoop's fields consist of an IceCreamFlavor and an ArrayList of MixIn choices, from zero to as many as the Customer can afford. The IceCreamFlavor for a Scoop is selected by the constructor, then MixIn choices are created and added to the mixins ArrayList via the addMixIn method.

The toString() should return the flavor and, if any mixins have been added, append " with " and a comma-separated list of mixins. For example,

- "Vanilla" (no mixins added)
- "Vanilla with Crushed Snickers" (a normal amount of Crushed Snickers mixed in)
- "Vanilla with Chocolate Chips (Extra)" (extra Chocolate Chips mixed in)
- "Vanilla with Crushed Snickers, Chocolate Chips (Extra)" both of the above mixed in!

TestScoop

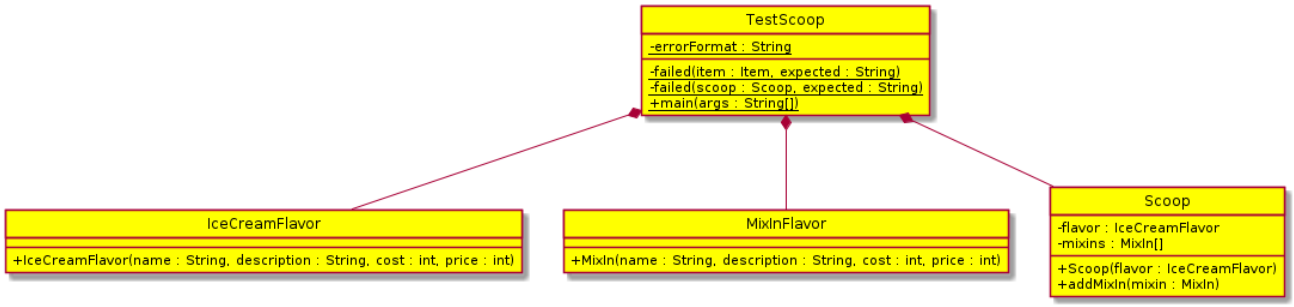
Write a TestScoop class that either

- Performs regression tests on your Scoop, IceCreamFlavor, and MixInFlavor classes (and indirectly the rest of the classes) OR
- An interactive Scoop builder with which to test your code.

The regression tests are probably easier, but you may choose your own path.

Regression Test

The class diagram below represents the suggested solution's regression tests, though you are NOT required to implement our TestScoop that way. No output is expected if all of the tests pass, right? But don't forget to "break" your code just long enough to "test your test"!



Static method failed

Regression tests are inherently redundant - that is, you tend to repeat a LOT of code. So it's very normal to practice DRY (Don't Repeat Yourself) by factoring out repeated code into static methods. So I wrote two overloaded methods that accept either an Item (MixInFlavor or IceCreamFlavor) or Scoop and an expected String and compare them, printing an error if they don't match and returning true if the test failed and false otherwise. Static field errorFormat is the printf format string for the error message (it's good practice to make that a class constant).

Since Item.toString() only returns the name, we need to use Item's getters for name, description, cost, and price to produce a String that fully represents the Item. We can use Scoop.toString(), though, since it adequately represents the Scoop's contents (at least in my opinion).

Static method main

I used the `failed` method to test 4 parameter variations of MixInFlavor:

- Normal case, where the name and description are typical English strings and the cost and price are positive integers, with price > cost.
- Empty description, 0 price and cost == price.
- Empty name and description, with negative cost and price and with price < cost.
- Foriegn language case for the name and description (I used Japanese for no discernable reason).

I used the `` failed method to test only the Normal case for IceCreamFlavor, since it is virtually identical to MixInFlavor.

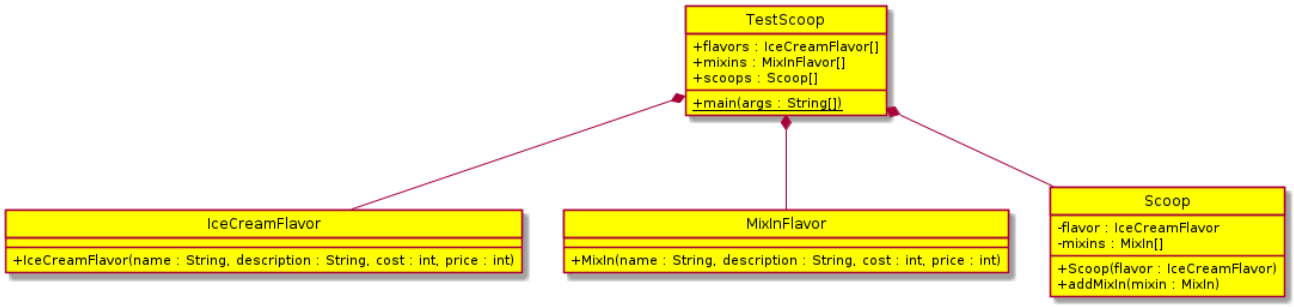
I used the other `failed` method to test 3 Scoop instances:

- No MixIns.
- One MixIn.
- Two MixIns.

This was actually the same number of lines as the Interactive Test, but a lot of copy pasta was involved so overall it was much faster to write. Recommended.

Interactive Test

The class diagram below represents the suggested solution's interactive tester, though you are NOT required to implement our TestScoop that way. An example run of the suggested solution's interactive tester follows the class diagram. (Notice that it doesn't offer to create a Scoop until an IceCreamFlavor is available, since one is required for Scoop's constructor. But you needn't be that clever.)




```
ricegf@antares:~/dev/202208/P05$ java TestScoop

================
MICE Tester v0.1
================

Create new (m)ixin, (i)ce cream flavor, or (q)uit? i

Creating new Ice Cream Flavor!

Name? Vanilla
Description? Luscious creamy vanilla bean ice cream
Price? 195
Cost? 35

================
MICE Tester v0.1
================

Create new (m)ixin, (i)ce cream flavor, (s)coop, or (q)uit? m

Creating new MixIn Flavor!

Name? Snickers
Description? Moderately chopped Snickers bars
Price? 99
Cost? 35
```

```
================
MICE Tester v0.1
================

Create new (m)ixin, (i)ce cream flavor, (s)coop, or (q)uit? m

Creating new MixIn Flavor!

Name? Chocolate Chips
Description? Mini chips of semi-sweet chocolate
Price? 79
Cost? 24

================
MICE Tester v0.1
================

Create new (m)ixin, (i)ce cream flavor, (s)coop, or (q)uit? s

Creating a scoop of ice cream!

0) Vanilla

Flavor? 0
0) Snickers
1) Chocolate Chips

Mixin?  0
0) Light
1) Normal
2) Extra
3) Drenched

Amount? 2
0) Snickers
1) Chocolate Chips

Another mixin? 1
0) Light
1) Normal
2) Extra
3) Drenched

Amount? 1
0) Snickers
1) Chocolate Chips

Another mixin? -1
Adding Vanilla with Snickers (Extra), Chocolate Chips
```

```
================
MICE Tester v0.1
================

List of Ice Cream Scoops:
  Vanilla with Snickers (Extra), Chocolate Chips

Create new (m)ixin, (i)ce cream flavor, (s)coop, or (q)uit?
```