

Bringing It Together - Mav's Ice Cream Emporium (Sprint 5)

Due Tuesday, November 22 at 8 a.m.

CSE 1325 - Fall 2022 - Homework #11 / Sprint 5 - 1

Revision 0

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 5 of 6, in which we add Customers for our Orders and allow a Customer to reorder a favorite Serving with a single click (hello, HashMap!). We also finally use that price and cost data to show the price on our receipts. Then we'll do a little clean up and call it a project!

(Sprint 6 has no mandatory features, and you may skip it without penalty. It exists solely for those who want to implement some of the bonus features for additional credit.)

The Scrum Spreadsheet

Copy your P10 directory completely to P11. Some Scrum spreadsheet updates are available that should be integrated with your progress to date however you find to be most convenient.

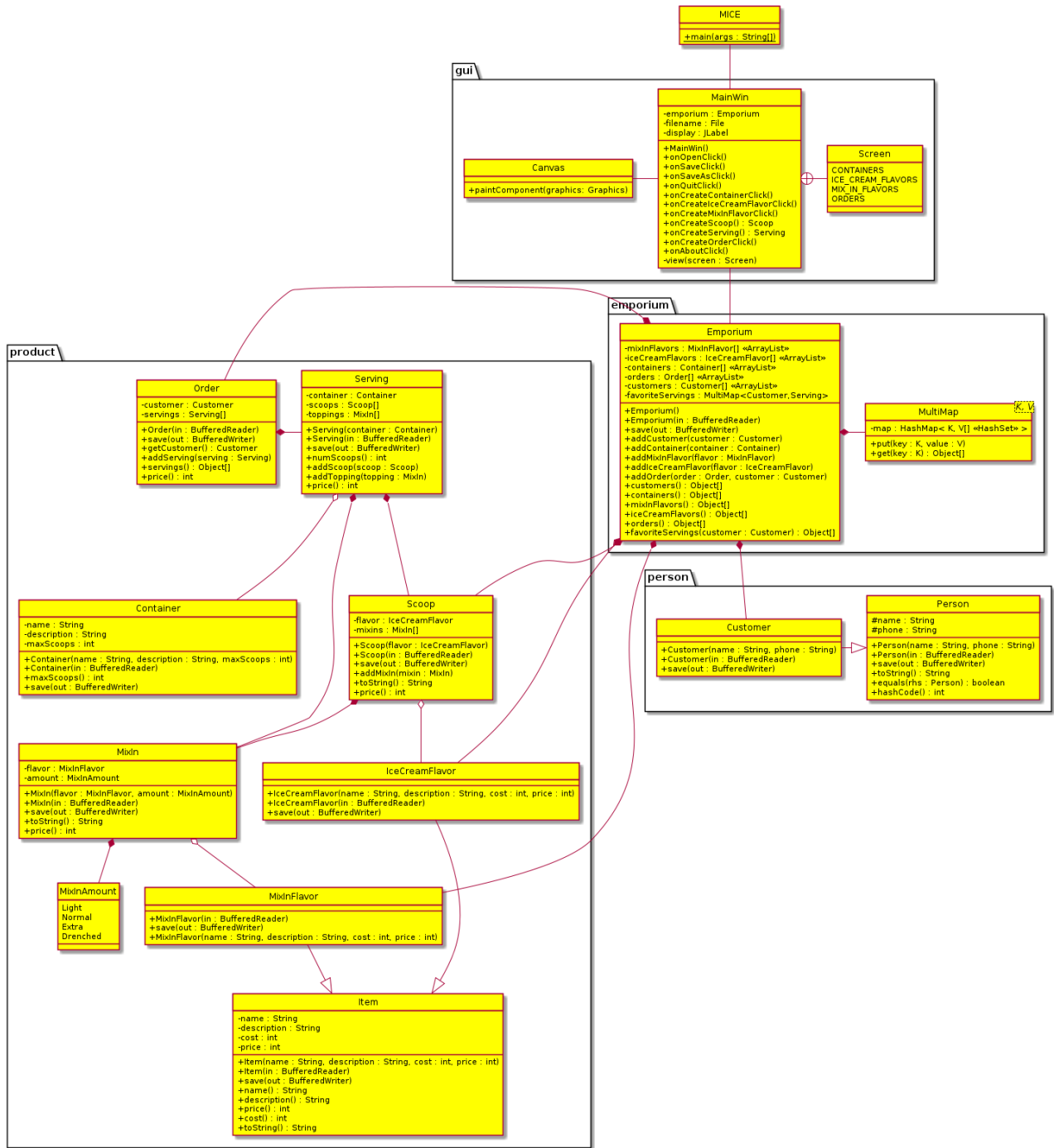
As always, update your Product Backlog tab to reflect your plan and your actual work for Sprint 5. Then, on the Sprint 5 Backlog tab, plan the tasks you'll need to implement the new features (the hints below should help) and update your status as you go.

Be sure to add, commit, and push the updated spreadsheet at `cse1325/P11/Scrum_MICE.xlsx` before the end of the sprint!

Class Diagram

The diagram on the next page has been updated to reflect the new features in this sprint. Note the addition of a Person superclass with Customer subclass. These will need menu items and toolbar buttons to request and to list in the main data area, along with a single dialog to create.

One or more updates with additional guidance and bug fixes should certainly be expected in any non-trivial project plan such as this.



Add the Price to the Receipt

Add a price method to Mixin, Scoop, Serving, and Order.

The price of a MixIn is the price of a MixInFlavor. If you like, multiply by .8 if Light, 1.2 if Extra, and 2 if Drenched.

The price of a Scoop is the sum of the prices for the IceCreamFlavors and MixIns in it.

The price of a Serving is the sum of the prices for the Scoops and toppings in it. (The container is included at no extra charge.)

The price for the Order is the sum of the prices for the Servings in it.

Then update the Order summary in MainWin to display the price of the Order.

Limit Scoops to Container's Max

This was an oversight in the previous sprint.

Add a maxScoops() getter to class Container, and a numScoops() method to Serving that returns the size of the scoops ArrayList.

Then modify MainWin.onCreateServing to stop requesting additional scoops once the Container is full.

Track Our Beloved Customers

Person and Its Subclass Customer

This should be simple for you by now. Person encapsulates 2 String fields, name and phone number. You need the usual constructor, getters, and toString, plus of course your save(BufferedWriter) and Person(BufferedReader) I/O classes. This is very similar to the Item class, so you might duplicate and edit it.

Customer inherits from Person, so you just need constructors and (if you like, although it would inherit) a save method.

We'll be using these as keys in a HashMap, though, and besides, now you know better. **So also override the equals(Object) and hashCode() methods** for these classes! (It isn't necessary to go back and "fix" all of the other classes, though you might add a feature to your backlog if you plan to include this project as part of your resume.)

Next, in Emporium, add an ArrayList of Customer objects, with the usual addCustomer(Customer) and customers() methods to store and retrieve them, per the class diagram.

Finally, in MainWin, add Create > Customer and View > Customers menu items and associated buttons, with the the usual onCreateCustomerClick() method to display a dialog, instance a Customer, and add it to Emporium.

Add Customer to Order

Now add a Customer field to the Order class, supplied in the constructor, along with a getCustomer getter method. Also update toString to show the Customer for this order.

In MainWin.onCreateOrderClick(), start by asking the user for the Customer via a combo box (for example, using a JOptionPane.showInputDialog or a JComboBox as the message in a JOptionPane.showConfirmDialog). Use the selected Customer when instantiating the Order.

Provide Customers Quick Access to Favorite Servings

Now we get to the important (and most valuable point-wise) portion of this sprint. Since Customers tend to order the same Servings rather often, we'd like to remember their favorites to save them time and encourage them to return often. But first, we need to remedy a minor omission from the Java Class Library.

Write a Generic MultiMap

As you should recall from Lecture 23 (and mentioned in a half dozen earlier lectures), Java offers a HashMap collection that is just like ArrayList except that the subscript can be of any non-primitive type. So we could write this to store the area in square miles for a couple of states in the US:

```
import java.util.HashMap;

public class DemoHashMap {
    public static void main(String[] args) {
        HashMap<String, Integer> stateAreas = new HashMap<>();
        stateAreas.put("Texas", 268596);
        stateAreas.put("Rhode Island", 1545);
        for(var state : stateAreas.keySet())
            System.out.println(state + " has an area of "
                               + stateAreas.get(state) + " square miles");
    }
}
```

When a Customer places an Order, we want to remember the Servings in that Order so they can quickly select them again and again. And a HashMap is just the thing!

We could create a HashMap<Customer, Serving>, and in Emporium.addOrder(Order order) not only add the Order to the orders ArrayList, but *also* iterate over the Servings in that Order and add them to the HashMap using Order.getCustomer() as the key!

Except...

A HashMap can store only **one** value for each key - that is, only **one** favorite Serving for each Customer. We want to save them all! What we need is a *MultiMap* - a HashMap that stores as many values for each key (as many Servings per Customer) as we care to put!

C++ has just such a class, but we're back to Java now and Java does not. However, we can write a simple one in about a dozen lines of code.

Consider the MultiMap class in the class diagram. Class MultiMap is *generic* (see Lecture 22), accepting two type parameters K (for the key) and V (for the value). This is exactly like Java's HashMap.

So we use K and V to declare a Java HashMap field, with K as the key and HashSet<V> (not V!) as the value. You should remember from Lecture 23 that a HashSet is like a HashMap, except that we don't use a key at all. We simply add values as we like, and it stores exactly one of each unique value, removing any duplicates. What's more, it has a toArray() method that returns all of the values we stored in the HashSet as an array of Objects - just what we need for our ice cream emporium!

So now you've declared your MultiMap generic class with K and V type parameters, and declared your private HashMap field with K for the key and HashSet<V> for the value. We just need 2 methods to use it for our favorite Servings.

For method put, first check if map.get(key) is null - if so, this is the very first time your MultiMap has seen this key, so put a new HashSet<V> at this key. Then either way, add the value to the HashSet at map.get(key).

For method `get`, first check if `map.get(key)` is null - if so, return an `Object` array with 0 elements. Otherwise, return `map.get(key).toArray()`, an `Object[]` containing all of the values stored for this key.

And just like that, you've created your own generic class!

For a full implementation, of course, you would need many additional methods similar to `HashMap`'s methods. But those two methods are all we need for our ice cream emporium.

Store Favorite Servings

Now that you have a working `MultiMap`, we can save and retrieve each Customer's favorite Servings while creating an `Order`.

Add a `MultiMap` field named `favoriteServings` to class `Emporium`.

In `Emporium.addOrder`, after adding the `Order` to the `orders` `ArrayList`, get the `Customer` from it. Then iterate over `Order.servings`, putting each into `favoriteServings` for that `Customer`. (The `HashSet` in our `MultiMap` will simply ignore any Servings that are already there.)

It's OK to defer saving and reconstructing the `MultiMap` until the end of this sprint, so that in effect favorite orders are not saved. It's a little tricky, and it's not the focus of this sprint. But it is included in the suggested solution, and if you have time, the experience will definitely be worth the effort.

Offer Favorite Servings

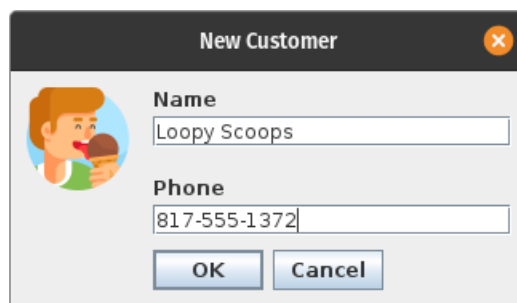
Now at last we can offer our Customers their favorite Servings! Modify `MainWin.onCreateServing` to accept a `Customer` parameter. Extract the Customer's `favoriteServings` from `Emporium` into an `Object[]` variable. If the array length is not 0, meaning this Customer has ordered before, add their favorites to a combo box so the Customer can select one if desired. If the Customer selects `Cancel`, return null. If the Customer selects `Yes`, return the selected `Serving`. Otherwise, proceed as before, allowing the Customer to create a new `Serving` that is destined to become a new favorite.

Finally, modify `MainWin.onCreateOrder` to supply the `Customer` that the user selected on each call to `onCreateServing`.

Screenshots




Your application is NOT required to look like the suggested solution! 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.











Screenshots will be added here in the first update.



Mavs Ice Cream Emporium

FileViewCreateHelp





Orders

Order 1 \$1.15 For Prof Rice:
Waffle Cone with a scoop of Vanilla with Snickers (Extra) and topped with Sprinkles (Light)

Select favorite serving?

Small Cup with scoops of Chocolate with Mini M and Ms (Light), Cookies and Cream, Vanilla with Snickers (Extra) and topped with Sprinkles

Small Cup with scoops of Chocolate with Mini M and Ms (Light), Cookies and Cream, Vanilla with Snickers (Extra) and topped with Sprinkles

Waffle Cone with a scoop of Vanilla with Snickers (Extra) and topped with Sprinkles (Light)