Bringing It Together
Mav's Ice Cream Emporium

CSE 1325 – Fall 2017 – Homework #7-#12

Sprint material is due for grading every Thursday at 8:00 am

One-week projects are fairly rare in the "real world", though not extinct. "Short" projects take many weeks, while longer projects can run into months or years. We can't squeeze a multi-year project into our class, but multi-week – that we can do! In this homework, you'll apply the simplified Scrum process you practiced with our Library Management System project to manage a 6-week development of a simple management system for an ice cream emporium.

As before, we'll get started by coding the Model with regression tests and a simple command line interface to facilitate interactive testing, then add a graphical user interface via Controller and View. Feel free to baseline your earlier homework submissions, or the suggested solutions that were provided, in developing your project. Also note that, for the first time, you are permitted but not required to work on a self-selected team of up to 4 students for the duration of this project.

Introduction

Mav's Ice Cream Emporium 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.

For reasons not stated, they really want a mouse-driven application.

Your job is to win this project (with associated profit and glory) from Mav's Ice Cream Emporium by producing a proposal package, including a prototype that wows and other creative and persuasive artifacts that prove you know your stuff. You have exactly 6 weeks to deliver your emporium manager prototype v1.0 with your proposal package to the Owner, at first featuring a glorious text menu or command line interface (CLI), and then finally a thoroughly compelling Graphical User Interface (GUI).

The Owner's Monologue

Here's what the Owner of Mav's Ice Cream Emporium (MICE) says they need.

MICE offers a revolving selection of ice cream flavors (sold by the **Scoop**), ice cream **Containers** (each of which can hold a maximum number of Scoops), and **Toppings** (which can be added on top of the ice cream in several quantities).

Each of these **Items** has a name, description, wholesale cost, retail price, stock remaining, and picture with which to entice the customer.

- An ice cream Scoop includes no additional information. Examples include vanilla, chocolate, cookies and cream, and raspberry swirl.

- An ice cream Container also includes the number of scoops of ice cream it can hold. Examples include bowls, cups, and sugar and waffle cones.
- An ice cream Topping can be requested as light, normal, extra, and drenched.

The emporium employs **Servers**, who have a name, uniquely assigned employee number, number of orders filled during their career, and their hourly salary.

The emporium also keeps track of **Customers** by name, phone number, and uniquely assigned customer number. Any Server can create a new Customer.

A Server or a Customer may select a Container, one or more scoops of ice cream Flavors (not exceeding the maximum for that container), and zero or more Toppings to create a new **Serving**. The retail price of the Serving is the sum of the retail prices for the Container, scoops of ice cream Flavors, and Toppings.

The Server or Customer further assembles one or more Servings into an **Order**. The Order associates the Server, the Customer, and one or more Servings, along with a unique order number (for this store), the total price (which is the sum of the price of each Serving contained therein, ignoring sales tax), and the state of the order: Unfilled, Filled, Paid, or Canceled.

- Once created, an order is initially **Unfilled**.
- The Customer for that order may transition an order from Unfilled to **Canceled**, which is a final state. Nothing else changes for a canceled order.
- Any Server may change the state from Unfilled to **Filled**, at which time the stock remaining of each Item in each Serving in the Order is reduced by the amount consumed. In addition, for every tenth order filled, the Server's hourly salary is deducted from the store's **Cash Register**.
- Any Server may change the state from Filled to **Paid**, which is a final state. When an Order is paid, the retail price of the Order is added to the store's Cash Register.

Any Server may replace items in the store – Scoops, Containers, and Toppings. The wholesale price of the items replaced is deducted from the Cash Register. Restocking is equivalent to filling 2 orders, regardless of the number of Items restocked, for purposes of paying the Server their hourly wage.

Manager may add new Items to the store, hire new Servers, and view reports on the store's operation.

- A **Server Report**, showing all Server data including how much they have earned total.
- A **Customer Report**, showing all Customer data.
- An **Inventory Report**, showing every Item and how many of each are currently available.
- An **Order Report**, showing every active Order, its current state, its wholesale cost and retail price, and the profit for that Order. An option also includes Filled and Canceled orders.
- A **Profit and Loss Statement**, showing income from all Orders, the cost of Server salaries and of Items added to stock, and calculating the difference as net profit.

To support future franchise operations, all of the above should be able to be managed independently for more than one **Emporium**. That is, a running instance of the program should be able to load the data for an Emporium, modify and save it, then load the data for another Emporium.

These features have associated priorities, but the **Owner** hasn't decided on them yet. Lots of additional features are also needed, which the Owner will inevitably think of as we go along. Welcome to reality.

Software Development Process

The first week (or part of it) will include our requirements analysis and design, as well as some implementation. Subsequent weeks will update our design based on what we've learned and on changes to the Product Backlog, then plan the sprint, then execute it, then deliver to Blackboard.

Analysis

- Create a **Use Case Diagram** for MICE by reviewing the Owner's Monologue. First, identify the actors. Then look for the use cases they need from your manager software. For example, we know that a "Customer" uses the "MICE Software" to "Create an Order".
- Elaborate at least ONE Use Case(s) with a non-trivial **Sequence Diagram**. For example, the process of creating, filling, and accepting payment for an Order is a good candidate.
- Elaborate ALL OTHER Use Cases with a brief **Text Description** sufficient to support creation of our Scrum Feature backlog.
- We will add a an **Activity Diagram** and a **State Diagram** later. Omit them for now.

Design

- Create a class diagram, again referring to the Owner's Monologue and your Use Case diagram to identify classes and their relationships.
    - The capitalized words may offer hints on which classes will make sense.
    - Carefully consider inheritance, aggregation, and association relationships between the classes.
    - Look for opportunities to apply Software Design Patterns to your design.
- Design a Menu Bar and Task Bar for the GUI.
    - It's a good practice to also design a simple CLI menu based on your Menu Bar to use in getting the Model up and running prior to adding the GUI.

Implementation

- After reviewing the Feature Backlog in the Scrum Spreadsheet, **select the features you will complete in the remainder of Sprint 1**. Plan their implementation n the Sprint 01 Backlog tab.
- **Each sprint has a minimum number of features that must be completed for full credit. These will be graded at the end of that sprint.** Be careful to at least complete the required features each week to maintain your homework grade.

- If you implement additional features during a sprint beyond the required minimum, *they will not be graded during that sprint*. You can treat the additional features as early implementation to be graded after a subsequent sprint, or you can continue working and go beyond the features selected for the project to obtain extra credit – potentially *lots* of extra credit.

Every class documented in the standard C++ 11 or 14 library and gtkmm 3.0 are available for your use on this project. You may also consider third party libraries that have been approved **in advance** by the Product Owner (that's the Professor, not the TAs!) in writing, as long as you comply with all license agreements. You will also need to obtain your own graphics, again in compliance with all applicable license agreements.

You must also of course use git (either locally or with an online repository such as GitHub) to version manage your work, and write regression tests for the classes in your Model.

Do NOT write all of your code for the entire project and then ask us for help getting it to compile. That's not how we teach software development, nor is it how professional developers operate in the real world.

Code a little, test a lot, backup frequently, version control always.

Subsequent Sprints

At the start of each subsequent sprint:

- **Update your UML models** based on what you have learned in the previous sprint.
- **Select the features** you'll implement for the next sprint.
- **Plan the tasks** to implement those features on that sprint's Sprint Backlog tab.
- **Continue implementing** to your plan.

Neither the professor nor TA will discuss your project with you at all unless you bring your UML models and up-to-date Scrum spreadsheet with you.

Deliverables

Each week, the student will deliver the usual artifacts to Blackboard in a CSE1325_07.zip through CSE1325_12.zip ZIP archive, including the following:

- The **Scrum spreadsheet** representing management of the project;
- Supporting **UML diagrams** including at a minimum a use case diagram, class diagram, and sequence diagram, with state and activity diagrams added in a later sprint;
- Archive of the local **git repository** or link to the Github repository, showing all source code, a Makefile, regression tests, and resources in commits;
- (Sprint 6 Only) Any sales material deemed to increase the probability of winning the contract (e.g., PowerPoint slides, PDF brochures, YouTube videos), which will garner additional credit.

Each student or team should be prepared to provide a final Product Demo to the class, if selected and scheduled by the Professor after completion of the final sprint. The number of Product Demos, if any, is dependent on availability of class time. More info to follow.

Bonus and Extreme Bonus

Unlike previous homework, your opportunity for **bonus credit is primarily rooted in completing additional features within the 6 sprint project limit.** Thus, it is to your (or your team's) advantage to complete features ahead of the sprint schedule to allow time for additional features to be added by the end of the scheduled project.

Each additional feature implemented in priority order is worth bonus 75 points for an individual, 40 points each for a team of 2, 30 points each for a team of 3, and 25 points each for a team of 4 (as always, adjusted by each team member's contribution to that feature per the Sprint Backlog). If you wish to implement features out of priority order, be sure to receive written approval from the Product Owner (the Professor) in advance.

In addition, finalists chosen to present to the class will be awarded 25 bonus points. The student or team selected by the class to receive the contract will be awarded an additional 50 bonus points.

Teaming

It is permissible but not required (*and not recommended*) to work in teams of up to 4 students, and these students may be in different CSE 1325 sections 001 through 003, since all sections are implementing the same project. Remember, if you decide to form a team, your grade will depend in part on the work ethic of your teammates. Choose wisely.

To form a team:

1. Talk with each invited team member to ensure they are enthused about the team. Be sure to discuss and reach agreement on how much additional work you want the team to accomplish for extra credit.

2. Select a name for you team. It may be as creative as you like, within the obvious constraints of non-vulgarity, respectfulness, and professionalism.

3. When all are in agreement, **EACH team member should email the Professor and TA, with cc: to each teach member.** EACH email must include the name of the team in the subject, and the list of team members in the body.

4. **The professor will Reply All to EACH email to acknowledge the team.** This email authorizes the team. **If you do not receive a reply, your team is NOT AUTHORIZED** – contact the Professor for details ASAP.

5. **EACH TEAM MEMBER must submit the identical deliverable EVERY WEEK to receive a grade.** "My team member submitted it" will result in a grade of 0 for that week. This is a feature of Blackboard – we literally cannot enter a grade if you do not submit your own work.

Once formed, the team will continue for the duration of the project except for extreme circumstances. If you believe the team cannot continue as authorized, e.g., a team member withdraws from the class, contact the Professor ASAP.

Miscellany

As before, I will provide weekly suggested solutions for the required features at the end of each sprint, and you have the option of adopting some or all of the provided code for future work. Thus, a bad week shouldn't derail your subsequent sprints.

Your final proposal (aka, your completed project) is due in Blackboard by Thursday, November 30 at 8 a.m. This is the next to last day of class. No extensions will be given. If you develop incrementally as we have taught you, then you **will** have a working program on that date - guaranteed. Deliver it, enjoy the Guest / TA lectures and project demos (hopefully being selected as a finalist to perform a demo yourself), and then prepare for finals.