Team Number: 09

Team Name: Breaking Bread

Team Members:
Elena Ingraham
Friedrich Amouzou
Hussain Quadri
Phi Trang

Vision

Our vision is to practice object oriented programming by designing a shop that utilizes a lot of the principles and patterns that we have learned in class. The shop initially was intended to be backed up by a SQL database, but we ended up using the singleton design pattern to store our items in an inventory.

Project description

Breaking Bread is an online bakery store with a catalogue of bakery utensils, merchandise, and customizable baked goods that are shipped directly to your home. Users can sign up, login, and search products to add to their cart. Once they search an item, they can add a certain quantity into their cart and customize the goods using decorators.

Features Implemented

User Requirements

- UR-01: As a customer or admin, I can use my email to sign up
- UR-02: As a customer or admin, I can search for available products
- UR-03: As a customer, I can add items to my cart
- UR-04: As a customer, I can delete items from my cart
- UR-05: As a customer, I can submit orders
- UR-09: As an admin, I can remove products
- UR-10: As an admin, I can add new products
- UR-11: As an admin, I can modify products
- UR-13: As an admin, I can view customer accounts
- UR-14: As an admin, I can remove customer accounts

Business Requirements

- BR-01: Login names must be the email used when signing up
- BR-02: Admins must use a company email to sign up

Functional Requirements

- FR-01: When a customer submits an order, the quantity of products should decrease accordingly
- FR-02: When a customer adds an item to cart they should be able to customize it
- FR-03: When a customer searches for products, they should be provided with matching products
- FR-04: When an admin signs on, they should be able to view a dashboard

Non-Functional Requirements

- NR-02: Support up to 10,000 products

Features not Implemented from P2

User Requirements

- UR-06: As a customer, I can view my orders
- UR-07: As a customer, I can customize items
- UR-08: As a customer, I can cancel orders
- UR-12: As an admin, I can add temporary deals to products

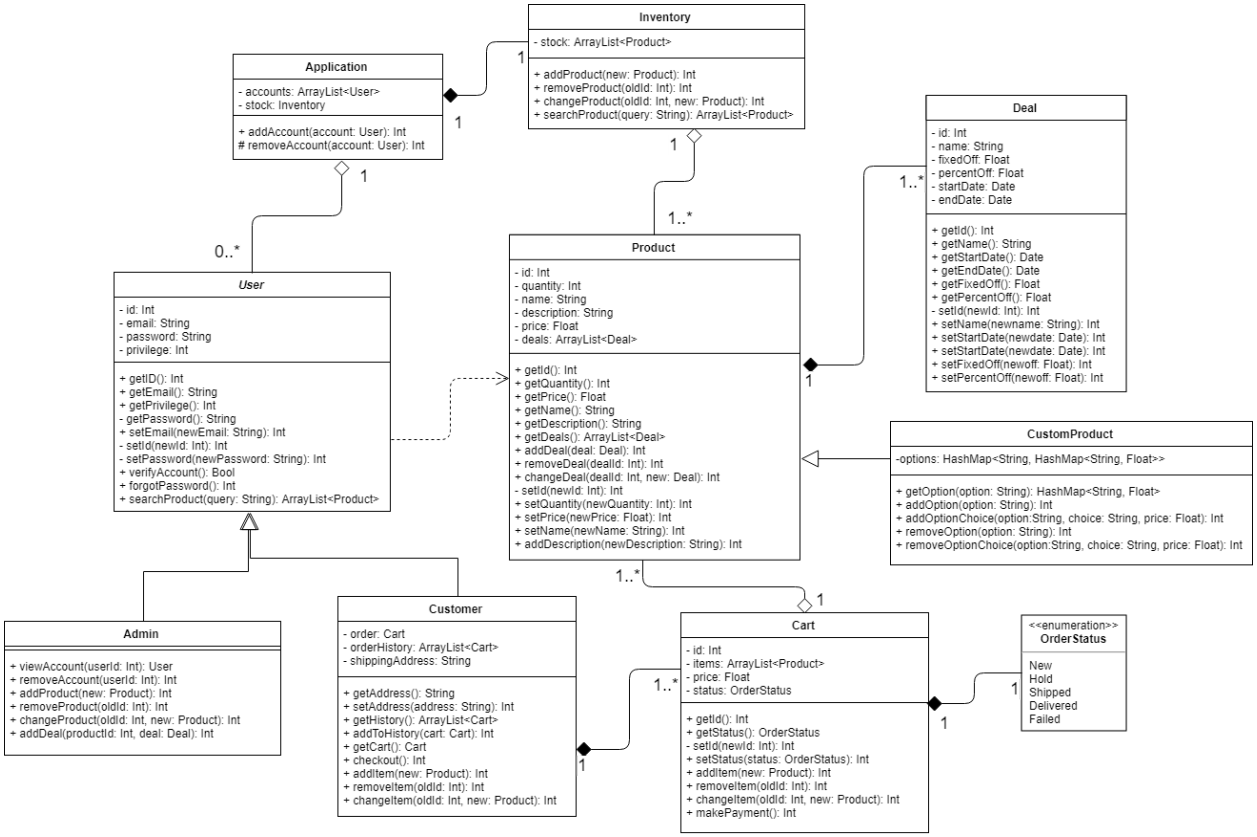
Functional Requirements

- FR-02: When a customer adds an item to cart they should be able to customize it

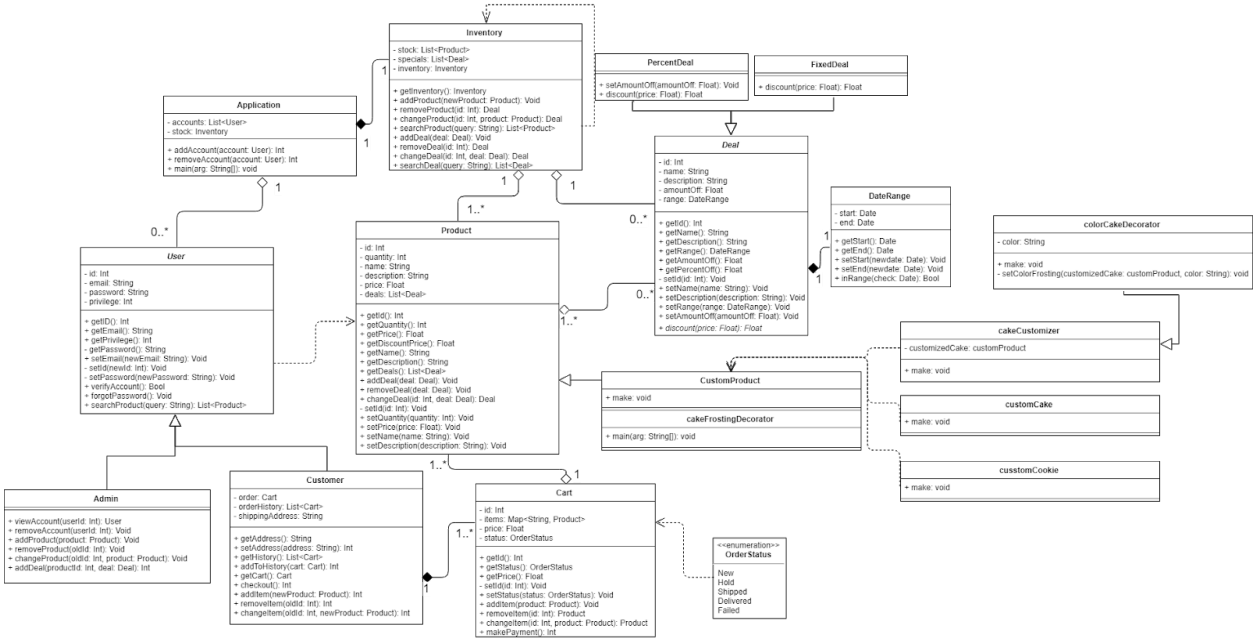
Non-Functional Requirements

- NR-01: Support up to 200 concurrent users
- NR-03: The system should be up 24/7
- NR-04: All user information should be encrypted

Part 2 Class Diagram



Final Class Diagram



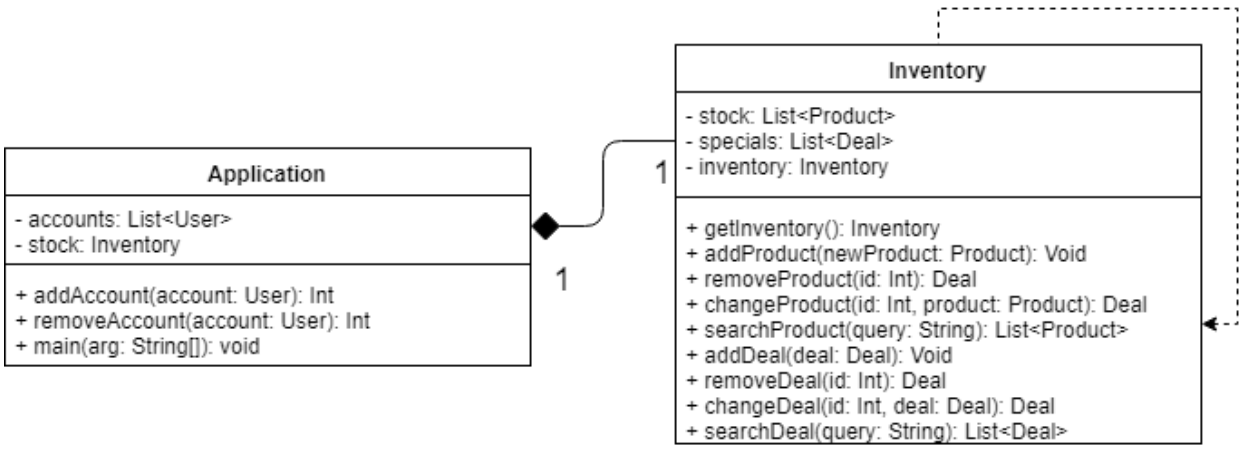
Things That Changed:

Our initial class diagram had some methods that didn't really make sense and were repetitive. Here are just some of the things that we changed:

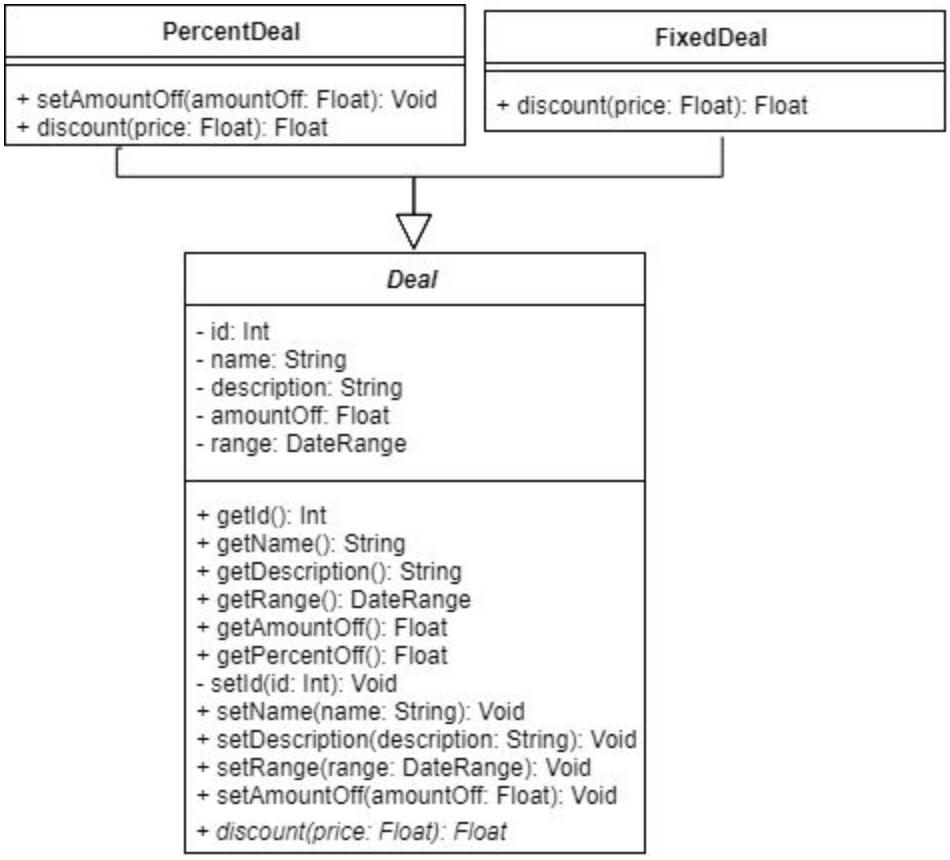
- Our Cart class no longer inherited from Inventory. We initially thought that the Cart would have the same functions as Inventory, but we were wrong. A lot of the methods within Inventory were never used within Cart, we scrapped the idea.
- Our Inventory class, we added a new attribute Inventory inventory, which was the single instance of the class. In order to create a Singleton, we created a new method getInstance.
- For adding items into the inventory and cart, we changed the parameters from an integer ID to a Product object.
- We added an entirely new class Application to be our driver
-
- We changed our initial custom products class to incorporate a decorator pattern.

Design Patterns

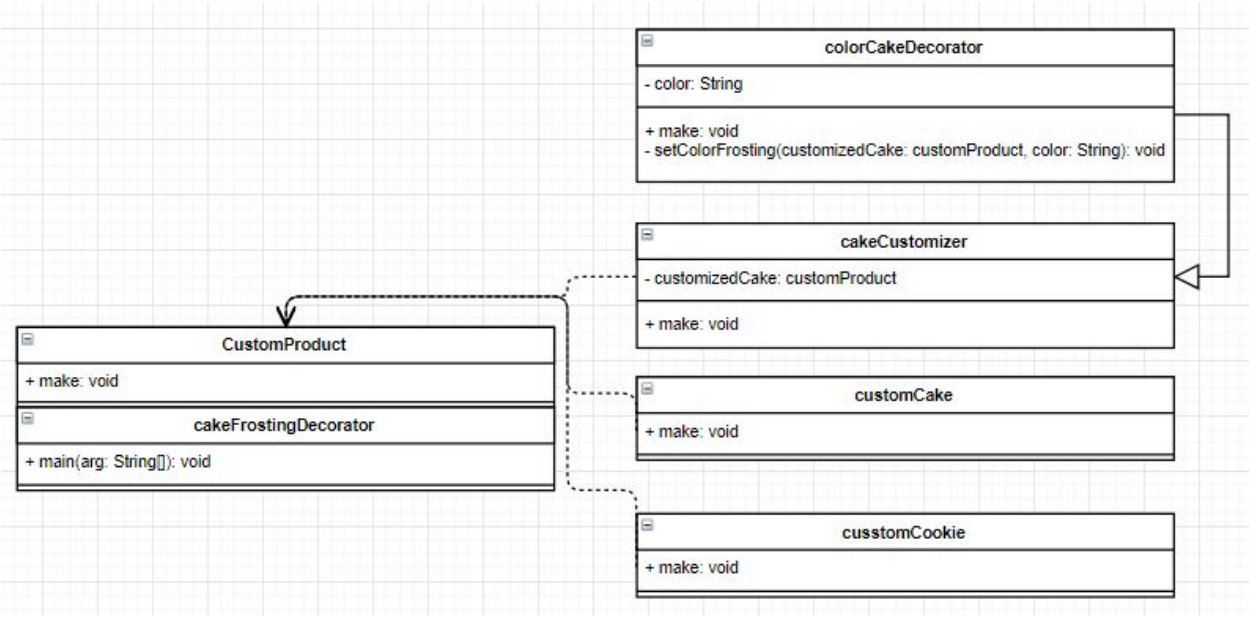
Singleton : Although it is the most frequently abused design pattern, we implemented our Inventory Class following the conventions of Singleton. Since only one Inventory() object will be instantiated within the Main class, this is an appropriate use of Singleton and does not violate any OOD principles. Through of use of Singleton, our inventory class could be globally used by other methods and classes.



Strategy: Since there are different discounts being applied the Strategy design pattern is helpful for easily making use of multiple algorithms. In our case the algorithms are various discount amounts which are applied to products in the Cart. By using the Strategy design pattern we minimize coupling and program to an interface as is prefered. The pattern also allow for the system to better follow the Open Closed principle of OOD since the client will not have to re-tinker with the Cart class if the discounts change or expire.



Decorator: The decorator design pattern was used for our product customisation classes. This allowed us to have multiple customisations that allow us to wrap together a users choices and create a final product to check out.. The decorator design pattern allows us to easily make items customizable and works by changing the color of the frosting on all of the objects that implement the custom product interface. These products include cakes, cupcakes, donuts and cookies and the user can change their frosting color thanks to the decorator design pattern.



What Have We Learned

We have learned that the most difficult and also the most important part of programming is probably not even programming itself, but the prior design. There are so many ways to go about solving an issue or creating a system, but if a group designs a well thought out and constructed outline beforehand, the process will be much easier. Additionally, we learned a lot about how design patterns can make coding so much easier and solve very specific problems.