Project Final Report
Hungry Foodie

Team members: Tianli Wu, Mengchen Gong
Language: Java (back-end), xml (front-end)
Toolsets: Android Studio, Firebase (database)
Project type: Android Mobile App

Final State of System

- **Features Implemented**
  - OO Patterns:
    - Strategy

      The usage of Strategy pattern is in user types, specifically different customer types, which are Customers and VIPCustomers. For VIP customers, all orders are Free to deliver. For normal Customers, if the order if under 25$, they need to pay 5$ for delivery fee on top of the food cost
    - Decorator

      Order total cost calculation with order adds-on calculation used Decorator Pattern. For each meal, the cost has been calculated and then if there is more things added (meat or cheese or both) by certain customer, the total price will be increment by using decorator pattern
    - Observer

      No matter customers choose takeout or delivery the order, they will be notified for cooking status or delivery status respectively. We have 2 observers here in the program takeOutObserver and DeliveryObserver. For difference choices customers chose, they will be notified differently
    - Adapter

      Adapter pattern used here is between the view and the list of items we have in the shopping cart. We use this adapter to provide an AdapterView, returning a view for each object in a collection of data objects that we provide.
    - Command

      We used a Command pattern between Customer and Merchant. DeliveryMan in our app works as an Invoker that takes the prepped order from Merchant and delivers the food to customers. To be specific, between customer and merchant, there are delivery man and app, DeliveryMan would be notified by Merchant, and work as our invoker (w/ the usage of the app) to execute the order.
    - Abstract Factory

      Abstract Factory pattern is used to produce meals. A normal meal contains 1 entree and 1 side for all restaurants' (merchants') menu. Burger meal for example, the combination is: Burger (entree), Pepsi (side). When a client clicks an image button in the Menu page, the meal name will be sent to the check out page. Pass the meal name to the Abstract factory pattern to create a meal object. So if a customer wants to put a meal in the shopping cart, the process is actually to use the Abstract Factory pattern to produce the meal and add it to the corresponding customer's cart list.

  - 4 users cases (user end)
    - Customers + VIPCustomers

      When registering, Customers choose to be VIPCustomer or Customer (casual). Those 2 user cases are really similar except for checkout. As mentioned before, VIP customers get free deliveries with no minimum spending; Normal customers would need to pay 5$ if the total cost is below $25

      New users are able register and then login. Once registered, their information will be stored in our database for further usage. After login, a message Good Morning / GoodAfternoon / Good evening + username for current user will show on the top corner depending on the actual time. Customers are able to choose

different restaurants and add as many meals as they want in the shopping cart. For each meal, they can choose to add in more meat and/or cheese.

Before checkout, it's the order summary review, if everythings is right, customers will check out and choose different methods (take-out vs. delivery) to acquire the food order.

- Merchants (Restaurants)

  Choose the user type when registering, have a separate window to login. After login, a message Good Morning / GoodAfternoon / Good evening + username for current user will show on the top corner depending on the actual time. If merchants try to login using the Customer's end, it will have a pop window showing that it is not at the right login page. After closing the opo-up window, merchants are able to click and go to the correct log-in page.

  2 main functionalities chains merchants can do are view-orders+assign drivers for delivery, and edit menu (didn't implement). For the current orders in the restaurant's list, it is able to assign to existing deliveryMans (the ones registered). After assigning the work, it is able to check the delivery status on driver's end or go back to main page

- DeliveryMan

  Choose the user type when registering, have a separate window to login. After login, a message Good Morning / GoodAfternoon / Good evening + username for current user will show on the top corner depending on the actual time. If deliveryMan tries to login using the Customer's end, it will have a pop window showing that it is not at the right login page. If no merchants has notified deliveryMans, the main page of them will show "no update". Once merchants notify them, they are able to pick up food orders, deliver them and update the status using the app.

- App interface (showed in demo)

  There are a total of 27 pages for the frontend for our app. We can mainly separate them into 4 sections

  - General
    - Register
  - Customers + VIPCustomers
    - Login
    - Login pop window wrong type message
    - Restaurant list
    - Restaurant's menu (x2)
    - Shopping cart
    - Add-on
    - Meal info (x2)
    - Checkout
    - Take-out & delivery option
    - Order summary (review)
    - Takeout status
    - Delivery status
  - Merchants (Restaurants)
    - Merchant log-in
    - Merchant Main
    - Merchant menu
    - Merchant edit (pop window)
    - Merchant order list (notify deliveryMan)
    - Merchant summary
    - Merchant delivery update (pop window)
  - DeliveryMan
    - DeliveryMan login
    - DeliveryMan order notification (pick up order)
    - DeliveryMan finish delivering
    - Merchant summary

        - Merchant delivery update

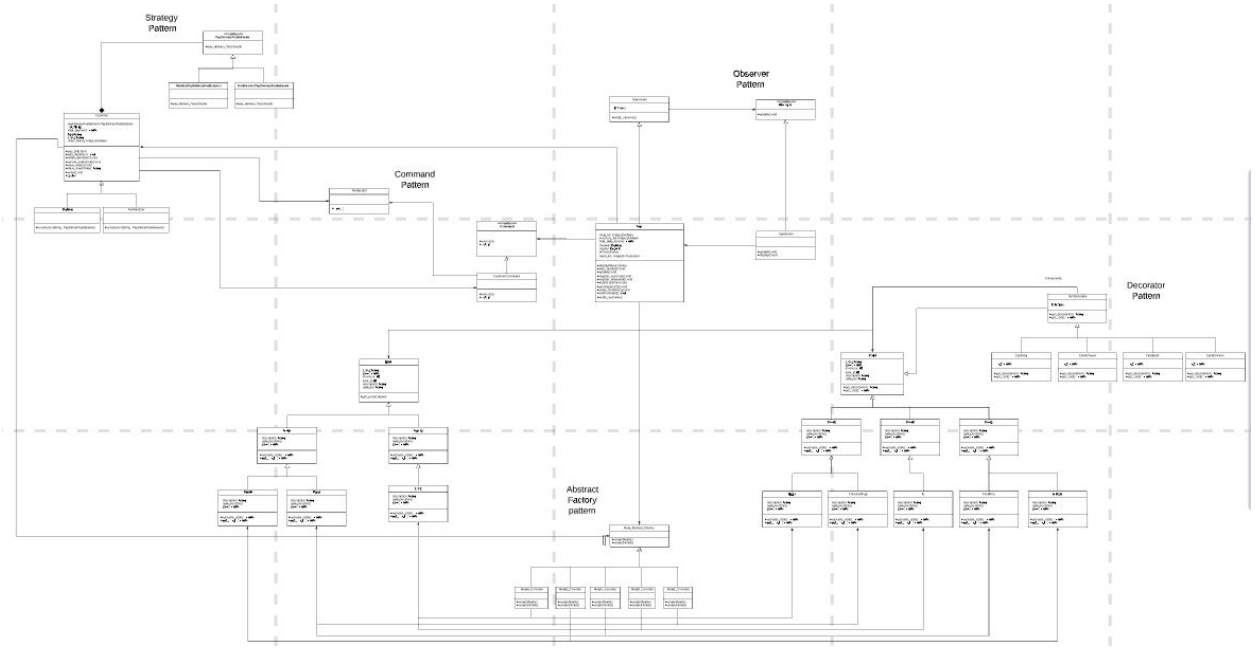
- Database structure
    - Users
        - Email
        - Name
        - Email
        - Password
        - Type
        - Cart
            - Description
            - Item name
            - Price
            - Quantity
    - Restaurant
        - Time
        - Customer email
        - Time
        - Option (take out/ delivery)
        - Orders
            - Description
            - Meal name
            - Quantity
    - DeliveryMan (Driver
        - Notifications
        - Restaurant name

- **Features Not implemented**
    - OO Patterns

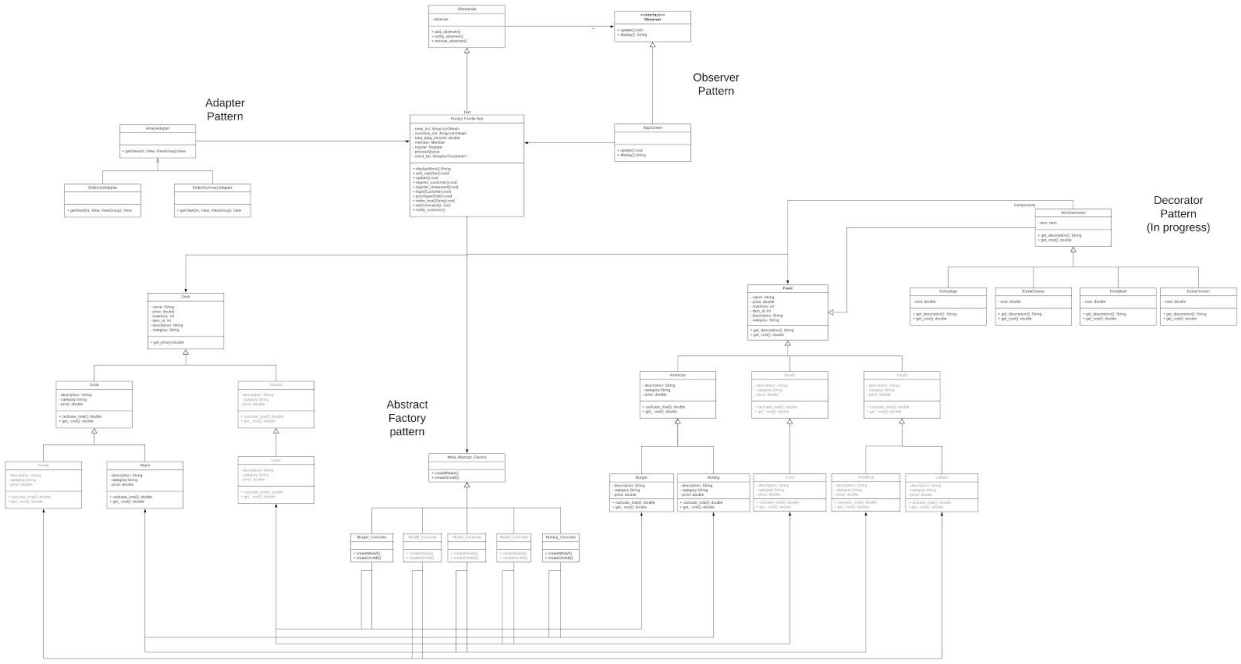
        We successfully implemented all the patterns we were planning to use, but some of the patterns like (decorator, abstract factory) didn't end up having as many classes as we expected. We also added an Adapter pattern for the usage of converting lists to views between activity classes.
    - User cases
        - Customers + VIPCustomers
            - Users profiles (reason: we spent most of the time on make the main function working and the connections to the database)
            - Review for restaurant (reason: at the end we don't think so far this function is important for the current stage)
        - Merchant
            - Edit menu function is not implemented (reason: when initially started, the thought is to have a edit button for each item on the menu and merchants are able to add in more meals with pictures, but it's way complicated compared to what we thought). INstead, a mock editing pop window is showing.
        - DeliveryMan
            - There are other functionalities get tip, order-deliver history are not implemented (reason: running out of time, and if we want to get tip, we need to get bac
    - App interface (in general)
        - User home (for information) (same reason as above)
        - When register, VIPCustomers should pay some fee (reason: so far focused on basic logic)
        - Per order for merchant, the app should gain 5%-10% of the earning (reason: so far focused on basic logic)

Final Class Diagram & Comparison

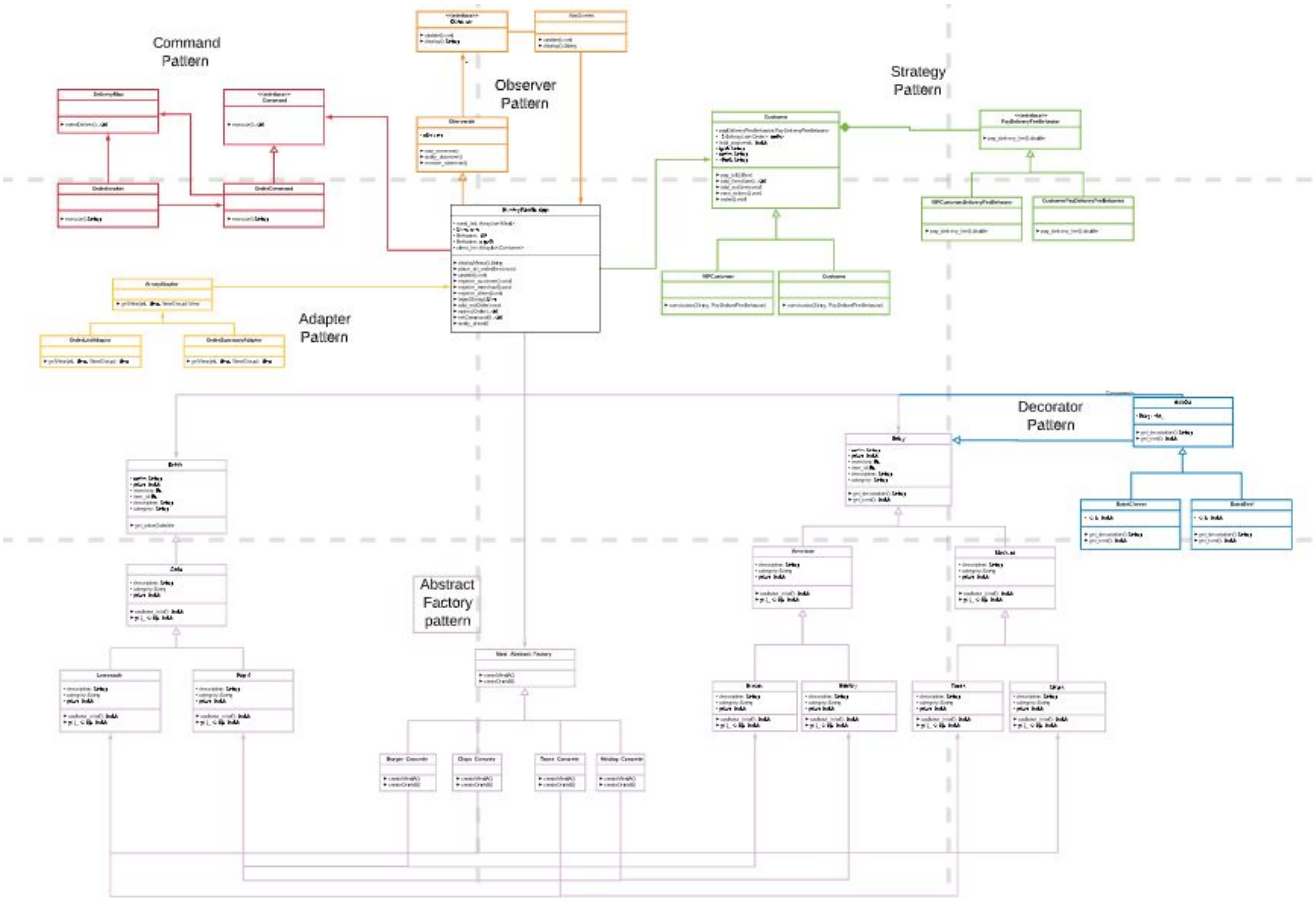
- Project 4 Class Diagram
  (https://www.lucidchart.com/invitations/accept/4e930327-5ed3-4080-a110-b3ac9ab866f4)



- Project 5 Class Diagram
  (https://www.lucidchart.com/invitations/accept/e1b69e88-4faf-4a27-beb4-8790c64dab6f)



- Final Class Diagram
  (https://www.lucidchart.com/invitations/accept/9753f9ff-c653-43aa-8472-fbb4e6a5403f)



Third-Party code vs. Original code

- Third-Party code

  We watched a lot of youtube tutorial and materials on Android Studio, but the whole implementation are done by ourselves

  Here are a list of links we've used for reference

  - https://www.youtube.com/watch?v=fn5OlqQuOCk (playlist)
  - https://www.youtube.com/watch?v=6ow3L39Wxmg (playlist)
  - https://www.youtube.com/watch?v=Ob4vSoWud9k (playlist)
  - https://www.youtube.com/watch?v=EknEIzswvC0&list=PLS1QulWo1RIbb1cYyzZpLFCKvdYV_yJ-E(playlist)
  - https://www.youtube.com/watch?v=jcvkHpPbVMk&list=PLfNDfEpg5eOPU8r_KqagzqVC_IXYbgcw2&index=4(playlist)
  - https://www.youtube.com/watch?v=bgIUdb-7Rqo(playlist)
  - https://www.youtube.com/watch?v=vMnCU6KKHd4&list=PLrnPJCHvNZuDrSqu-dKdDi3Q6nM-VUyxD(playlist)
  - https://www.youtube.com/watch?v=nBaL78HC0ls(playlist)
  - https://www.youtube.com/watch?v=E6vE8fqQPTE(playlist)
  - https://developer.android.com/guide/topics/ui/controls/checkbox
  - https://developer.android.com/guide/topics/ui/controls/spinner
  - https://en.proft.me/2017/02/23/command-pattern-java-and-python/
  - https://stackoverflow.com/questions/29880992/why-wont-emulator-window-close-in-android-studio-1-1-0-when-i-select-stop-from
  - https://stackoverflow.com/questions/39604889/how-to-fix-you-need-to-use-a-theme-appcompat-theme-or-descendant-with-this-a/39604946
  - https://mkyong.com/android/android-imagebutton-example/

- https://www.dev2qa.com/passing-data-between-activities-android-tutorial/
- https://firebase.google.com/docs/firestore/quickstart
- https://www.tutorialspoint.com/android/android_list_view.htm
- https://developer.android.com/reference/android/widget/ArrayAdapter
- https://developer.android.com/reference/android/os/Parcel.html
- https://stackoverflow.com/questions/11452859/android-hashmap-in-bundle

- Original code

  "All codes are written by ourselves, some of the code follows the tutorial or examples on the resources listed above." - Mengchen & Tianli

OOAD process (3 key design process elements)

1. Agile methodology. It is the right type of methodology for this project, it will be so hard if used waterfall methodology. For the whole project, we split it into many different parts, which are ground interface implementation, abstract factory pattern implementation, decorator pattern implementation, observer pattern implementation, adapter pattern implementation, strategy pattern implementation, and command pattern implementation.

   By using Agile methodology, we gradually fulfill the requirements of the whole project. We focused on one part and moved forward step by step. By using the emulator from Android studio, we are able to test and see if our app is functioning well. Agile also worked well when collaborating with different team members that we are able to focus on different parts in the process, combine work together or work together for the next stage.

2. We have designed UML, Architecture diagram, Activity diagram, use cases, UI diagram in project 4, which made it possible to distribute work to team members working in parallel. We also designed the app logo and sketched out interfaces using Figma that provided a good guideline for building the app in Android studio.

   The former work in project 4 provided us a good understanding of what we wanted to accomplish for this project, so worked well as a guideline in the later process. It was important to help us understand the requirements of the project, and made the big picture of the project clear rather than just an idea floating in our brain. Without project4, the later stages will be way more difficult.

3. We got an asynchronous issue. In the PlaceAnOrder page,we need to load cart data from the database first. Once loading is completed, put updated cart data back to the database. However, we are unaware that the loading method we call is asynchronous which makes a series of issues.