PROJECT

Nathan Cheng

Vanier College 420-SF2-RE sect. 00001

Contents

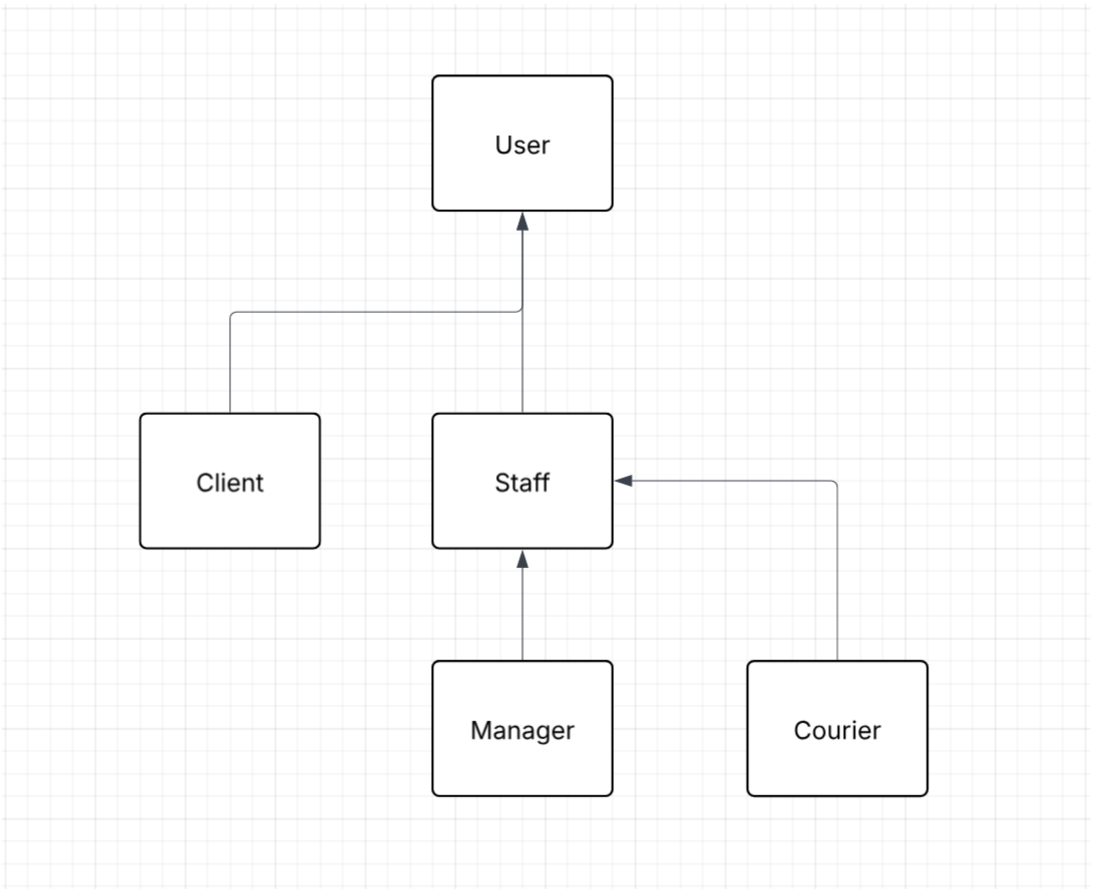
Scenario

The application project focuses on a scenario similar to a delivery system of a post-office center. In this case, the project will encapsulate functionalities such as managing and monitoring deliveries, monitoring requests and tickets, listing requests and deliveries, and establishing roles and responsibilities.

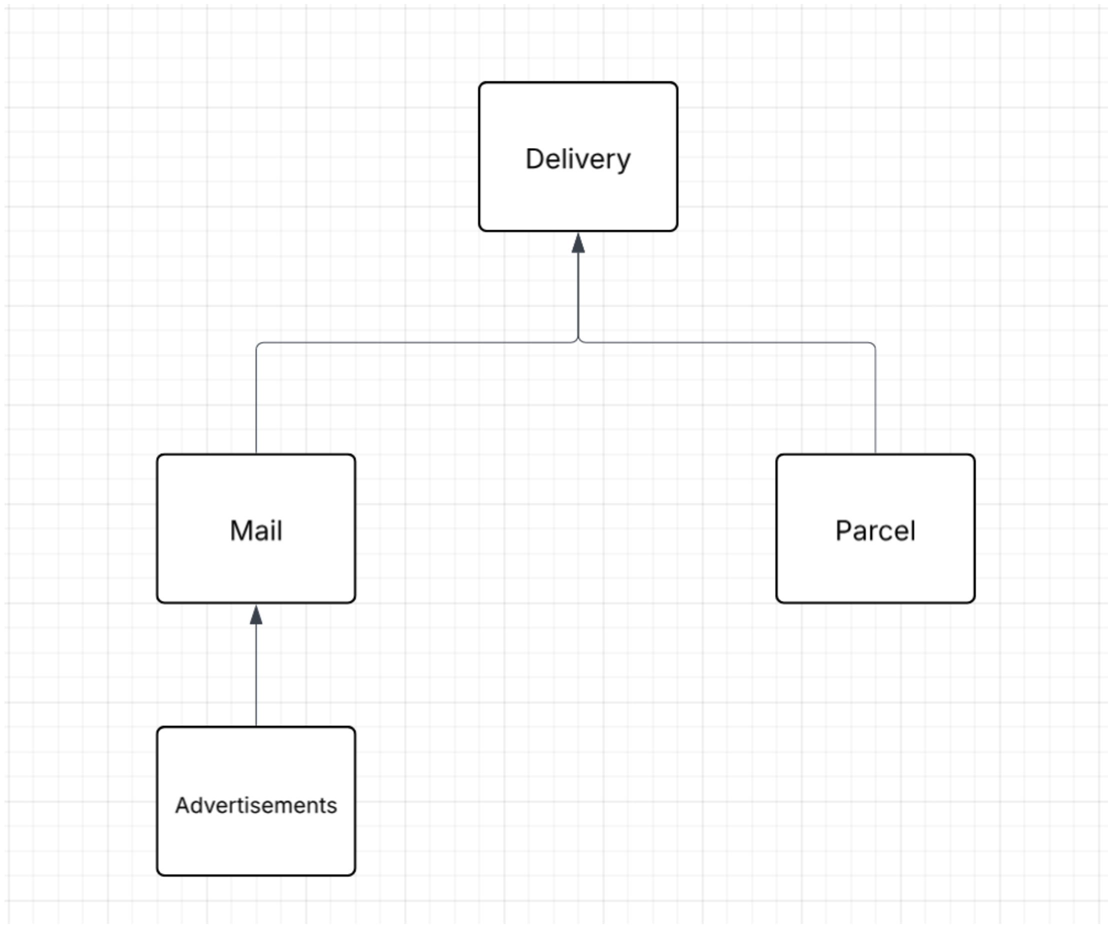
Hierarchical classes

The project will include User and Delivery hierarchy.

On one hand, the user family will be separated to clients and staffs, which can also descend to manager or courier. In the user family, the User abstract class can register their information, which can also be modified. For the client subclass, the clients are able to create, view, and manage their tickets and requests. The staff subclass consists of the abilities to manage deliveries, tickets, and requests. In terms of the Manager class, the manager can enroll new staffs and monitor all requests and deliveries. The courier class are given the functionalities to complete deliveries and return unwanted deliveries.



On the other hand, the Delivery hierarchy is split between mails and parcels. In the Delivery class, it contains the information of the client and their address. For the mail subclass, mails only hold messages in form of descriptions or details. For the parcel subclass, it contains an item with its information, as well as a description of the item. The Mail subclass also have an Advertisement subclass that contains descriptions of advertising product according to associated companies.



Functionalities

Design:

The design of the project encapsulates several demonstrative functionalities:

- The ability to register their shared information to the system as a client or staff member

- The creation of tickets such as bug reports, item refunds or cancellations through parcel, support, and mailing requests
- The action to monitor or view certain deliveries as a client, staff member, or manager
- The management of deliveries by assigning them to staffs and couriers, completing them, or removing old or cancelled deliveries
- The ability to create deliveries and associate them by address
- The ability to change or modify assigned roles or information

Expected Result:

In the end, the delivery application should allow the clients to perform certain actions:

- Registering their share information on the platform
- Changing their share information
- Creating a ticket for bug report and support
- View their associated deliveries
- Managing their own deliveries and delivery request such as cancelling, sending, sorting, monitoring, and receiving their mails or items
- Returning & refunding received packages after a certain amount of time
- Creating mails and packages to send to others via address
- *Possibly view a catalogue to buy certain items in associated store*
- *Add to cart system that will automatically create a parcel to their address*

Interfaces

User-defined:

- Refundable: For the Parcel and Item class, the interface allows to check if the parcel, or item, can be still refunded or not according to the date.
- Expirable: For the PostOffice class, it will remove all mails after a certain amount of time (assuming that certain mails are abundant and are not perishable).
- Processing: For the Staff and Manager classes, the interface reserves the abilities to manage, complete, remove, and label tickets, requests, and deliveries to these members while the Courier class is dedicated to only delivering their current deliveries.

Sorting

Comparator:

- Staff class
- Parcel class
- Mail class
- Item class

Comparable

- Client class
- Advertisement class

Methods

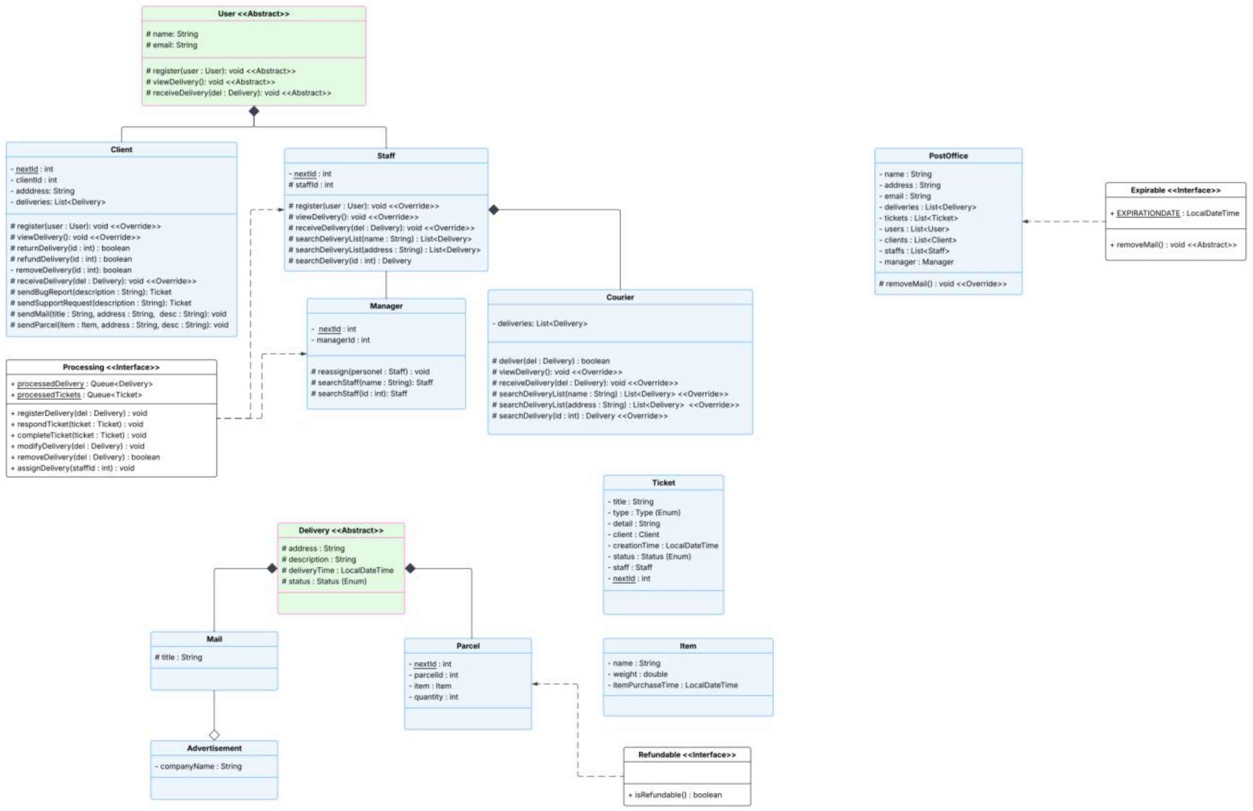
Polymorphism:

- The viewDelivery() function that allows the User classes to view their deliveries. It is different between clients and staff members. It is also different between couriers and staffs.
- The receiveDelivery() function gives the ability to store the received delivery in their inventory, which can vary between clients and staff members.
- The register() function allow the users to register their share information, however the information will reflect whether if it is a staff member or a client.

TextIO:

- The User class and the Delivery class both need to implement the TextIO functionality in order to store the given information inside the local resource file
- The Item class can also have the implementation of the TextIO functions for easier access to available registered items.

Diagram


User <<Abstract>>
# name: String
# email: String
# register(user : User): void <<Abstract>>
# viewDelivery(): void <<Abstract>>
# receiveDelivery(del : Delivery): void <<Abstract>>
Client
- nextId : int
- clientId : int
- adddress: String
- deliveries: List<Delivery>
# register(user : User): void <<Override>>
# viewDelivery(): void <<Override>>
# returnDelivery(id : int) : boolean
# refundDelivery(id : int) : boolean
- removeDelivery(id : int): boolean
# receiveDelivery(del : Delivery): void <<Override>>
# sendBugReport(description : String): Ticket
# sendSupportRequest(description : String): Ticket
# sendMail(title : String, address : String, desc : String): void
# sendParcel(item : Item, address : String, desc : String): void
Staff
- nextId : int
# staffId : int
# register(user : User): void <<Override>>
# viewDelivery(): void <<Override>>
# receiveDelivery(del : Delivery): void <<Override>>
# searchDeliveryList(name : String) : List<Delivery>
# searchDeliveryList(address : String) : List<Delivery>
# searchDelivery(id : int) : Delivery
Manager
- nextId : int
- managerId : int
# reassign(personel : Staff) : void
# searchStaff(name : String): Staff
# searchStaff(id : int): Staff
Courier
- deliveries: List<Delivery>
# deliver(del : Delivery) : boolean
# viewDelivery(): void <<Override>>
# receiveDelivery(del : Delivery): void <<Override>>
# searchDeliveryList(name : String) : List<Delivery> <<Override>>
# searchDeliveryList(address : String) : List<Delivery> <<Override>>
# searchDelivery(id : int) : Delivery <<Override>>
Processing <<Interface>>
+ processedDelivery : Queue<Delivery>
+ processedTickets : Queue<Ticket>
+ registerDelivery(del : Delivery) : void
+ respondTicket(ticket : Ticket) : void
+ completeTicket(ticket : Ticket) : void
+ modifyDelivery(del : Delivery) : void
+ removeDelivery(del : Delivery) : boolean
+ assignDelivery(staffId : int) : void
PostOffice
- name : String
- address : String
- email : String
- deliveries : List<Delivery>
- tickets : List<Ticket>
- users : List<User>
- clients : List<Client>
- staffs : List<Staff>
- manager : Manager
# removeMail() : void <<Override>>
Expirable <<Interface>>
+ EXPIRATIONDATE : LocalDateTime
+ removeMail() : void <<Abstract>>
Ticket
- title : String
- type : Type (Enum)
- detail : String
- client : Client
- creationTime : LocalDateTime
- status : Status (Enum)
- staff : Staff
- nextId : int
Delivery <<Abstract>>
# address : String
# description : String
# deliveryTime : LocalDateTime
# status : Status (Enum)
Mail
# title : String
Advertisement
- companyName : String
Parcel
- nextId : int
- parcelId : int
- item : Item
- quantity : int
Item
- name : String
- weight : double
- itemPurchaseTime : LocalDateTime
Refundable <<Interface>>
+ isRefundable() : boolean

**Note: It can still be modified. This is only a template of the overall design. **

Deliverable 2

In the second deliverable, the focus for the project will be starting with the implementation of the User family, the Delivery hierarchy, the Item class, a PostOffice class, their relationships, and the beginning of the implementation of certain functions like viewDeliviery() and TextIO functions, as well as the basic methods in each class.