SWEN30006 - Project 1: Automail

Student: Dyno Wibowo - 1002801
Student: Thomas Morrison - 1080352
Student: George Troupis - 830 930

WifiModemQuerier
As a means of accessing the WifiModem class, we implemented a class called the WifiModemQuerier. This class is a Singleton (Gamma, 1995) class instantiated by Simulation. Under the information expert (Larman, 2004) principle, Simulation has the necessary resource to create the WifiModemQuerier instance — it needs a WifiModem instance which is stored in Simulation. The reason for the WifiModemQuerier class is to avoid direct coupling between classes that need access to WifiModem's methods and WifiModem itself. It serves as an intermediary under the indirection (Larman, 2004) principle. Under the protected variation (Larman, 2004) principle, it serves as an interface to handle the instability of WifiModem's lookup methods. As mentioned earlier, WifiModemQuerier is a Singleton class. We decided to do this because we would need only one instance of WifiModemQuerier. The instance would also have to be accessible to multiple different classes. If it was not a Singleton, then a particular instance of the class would have to be passed into all of the classes that need to use it. It also carries the risk of creating more querier than what was needed. The alternative to our approach would be to directly access WifiModem for lookup calls and store every lookup information in Simulation. We decided to go against this because it leaves little separation between WifiModem class — of which we do not have any control over — and all other classes that require it. It was done to prevent direct coupling.

Charge
In terms of the Charge functionality, we debated whether it should be its own class or a set of attributes and methods in MailItem. We decided Charge should be its own class, as the alternative would lead to low cohesion (Larman, 2004) due to MailItem becoming too bloated. The Creator principle (Larman, 2004) dictated that MailItem should create Charge as it stored it and had the relevant information for creation and update.

Charge accounts for future changes in a number of ways. It contains an extra cost variable that we have set to 0, but can be repurposed in the future via a new method to account for weight and delay penalties. In addition, we have made activity unit price and markup percentage configurable by adding "ActivityUnitPrice" and "MarkupPercentage" to the list of properties Simulation looks for in the properties file. If not found, they revert to their default values.

The main purpose of the Charge class is to calculate the Charge, as shown by Figure 1. This is done multiple times, at the beginning and end of the delivery process. Thus, lookups are generally done more than once per item, in addition to lookups at the start of the process. This implementation only charges the tenant for one lookup despite potentially doing more, as the design brief outlined it would be unfair to pass these costs to the tenant. We determined this was a fair trade-off in order to ensure the System is fully operational and deliveries can be completed with near certainty.

Statistics
In order to add the new statistics feature, we decided to implement them within the Simulation class. Simulation is the perfect fit for this feature — under the information expert principle — as it has access to all the information necessary to function. Simulation also already handles all the printing and keeps

track of a statistic for Delay; we found that it may be better to just extend the existing class instead of creating an entirely new one.

We added new functionalities to keep track of and calculate the number of items delivered, total billable activity, total activity cost, total service cost, and total number of lookups. Total billable activity is the sum of all chargeable activity units deriving from the delivery of items. Total activity cost and total service cost are the accumulation of all of all the costs from all billed charges. The total number of lookups is the number of times WifiModemQuerier has been called for a lookup function.
The alternative of doing statistics would be to create a completely separate class. We avoided this because it will lead to coupling — this class would have to be connected with many other classes in order to function.

MailPool
The pricing mechanism also required implementation of priority mail. Any MailItems which have an estimated Charge of greater than the ChargeThreshold are classified as priority items and should be delivered first. We decided to handle this distinction of priority mail within the MailPool class. This was decided using the Information Expert principle which assigns responsibility to the MailPool because it already has all the relevant information and methods. For example, the MailPool class already has methods to handle loading the items into a robot.

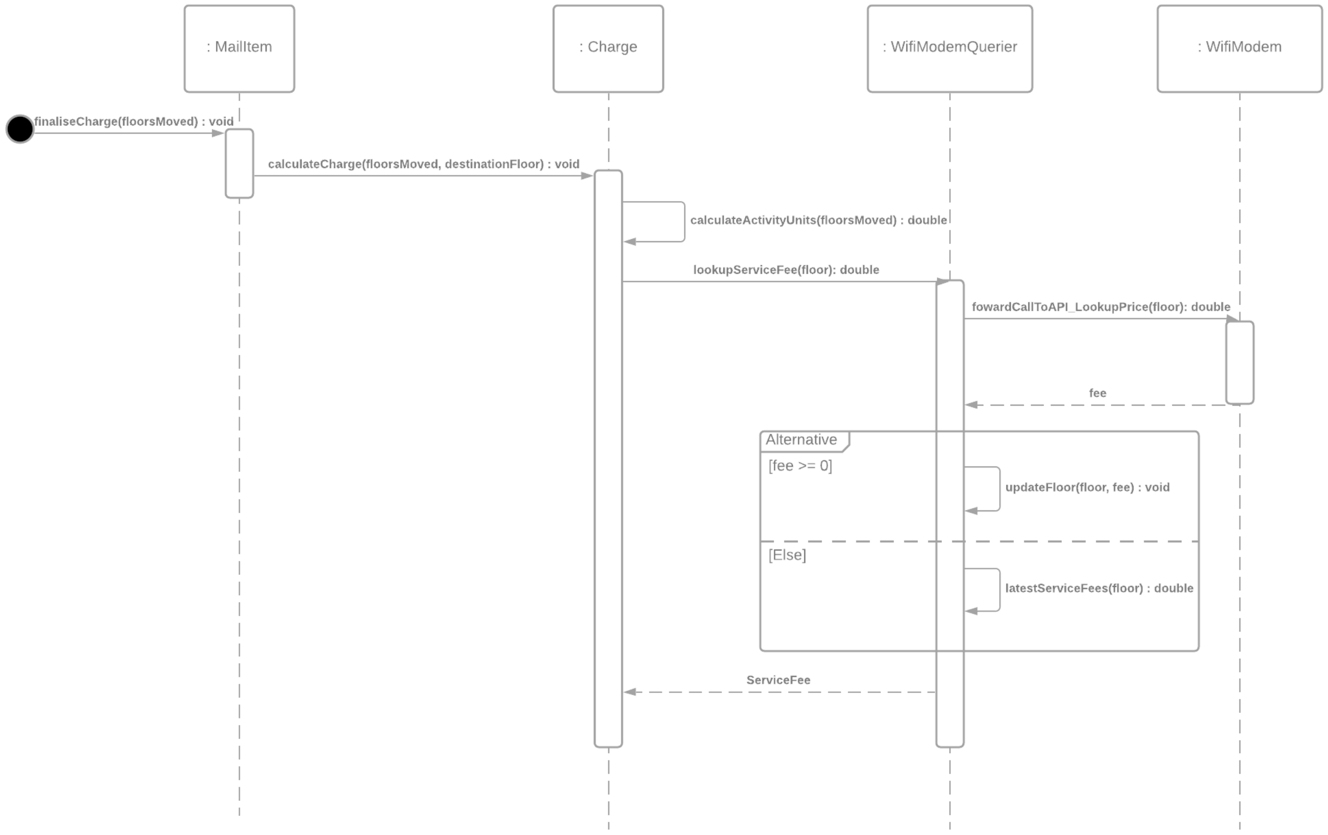
An alternative to this would be creating a separate class which *just* handles priority items. This would support low coupling between objects by creating a greater distinction between priority mail and non-priority mail. However, an important contradiction of the Pure Fabrication (Larman, 2004) principle is that objects should not have too many responsibilities that are *not* located with the information that is relevant to them. This means there is a tradeoff between 1) having a simple design with relevant information and methods stored close to each other and 2) creating multiple distinct classes which will all have high dependencies on one another. In the future, if it is decided that there should be multiple priority levels, it may be important to abstract the classes to support low coupling. However, there is no indication that this is in-scope for the project and therefore has not been factored into this design.

REFERENCES

Gamma, E. (1995). Design patterns : elements of reusable object-oriented software. Addison-Wesley.

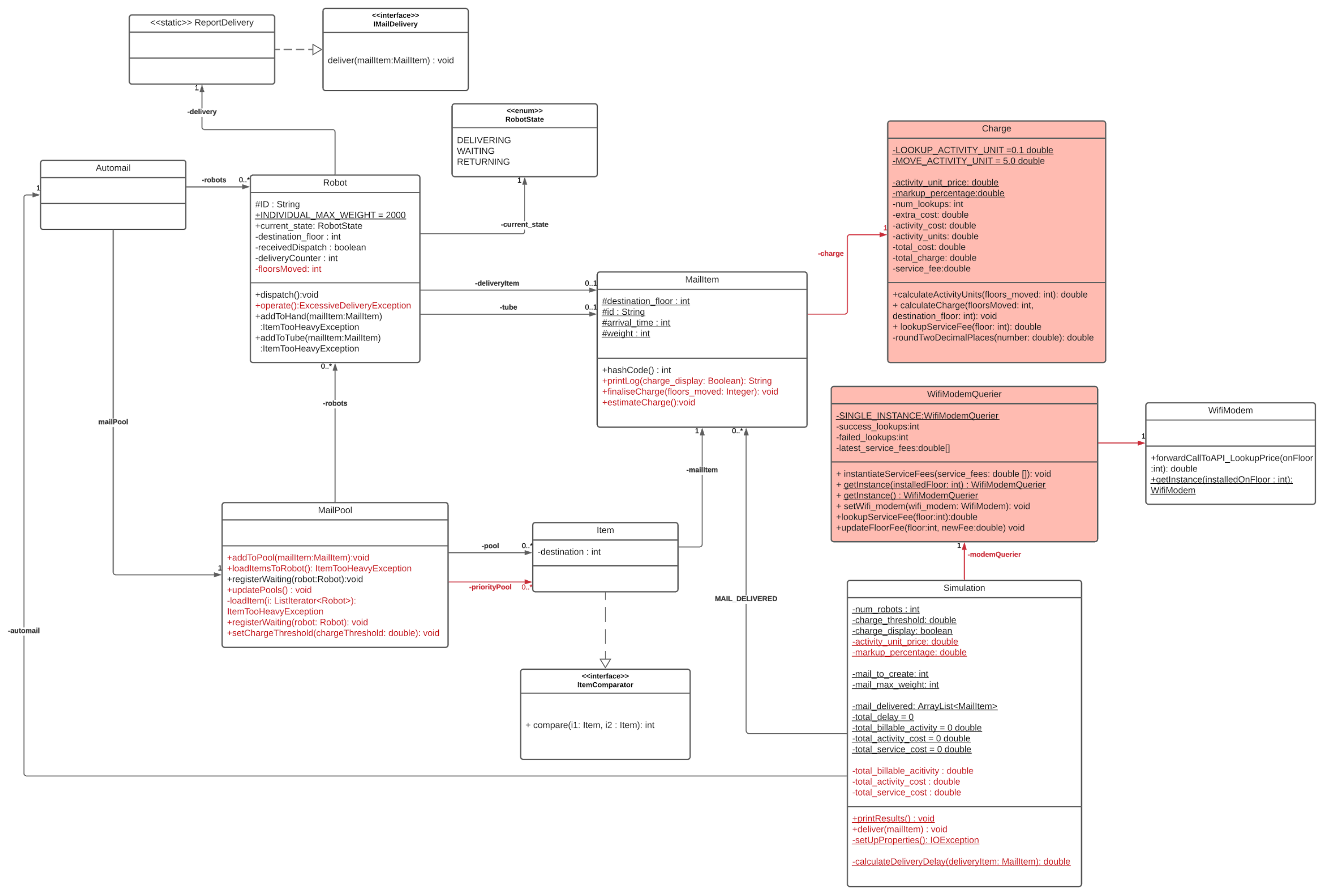
Larman, C. (2004). Applying UML and patterns : an introduction to object-oriented analysis and design and iterative development (3rd ed.). Prentice Hall Professional Technical Reference.

DIAGRAMS



: MailItem
: Charge
: WifiModemQuerier
: WifiModem
finaliseCharge(floorsMoved) : void
calculateCharge(floorsMoved, destinationFloor) : void
calculateActivityUnits(floorsMoved) : double
lookupServiceFee(floor): double
fowardCallToAPI_LookupPrice(floor): double
fee
Alternative
[fee >= 0]
updateFloor(floor, fee) : void
[Else]
latestServiceFees(floor) : double
ServiceFee

(Figure 1) - Design Sequence Diagram for calculating the final Charge


<<static>> ReportDelivery
<<interface>> IMailDelivery
deliver(mailItem:MailItem) : void
<<enum>> RobotState
DELIVERING
WAITING
RETURNING
Automail
-robots
0..*
Robot
#ID : String
+INDIVIDUAL_MAX_WEIGHT = 2000
+current_state: RobotState
-destination_floor : int
-receivedDispatch : boolean
-deliveryCounter : int
-floorsMoved: int
+dispatch():void
+operate():ExcessiveDeliveryException
+addToHand(mailItem:MailItem) :ItemTooHeavyException
+addToTube(mailItem:MailItem) :ItemTooHeavyException
-delivery
-current_state
-deliveryItem
-tube
0..1
MailItem
#destination_floor : int
#id : String
#arrival_time : int
#weight : int
+hashCode() : int
+printLog(charge_display: Boolean): String
+finaliseCharge(floors_moved: Integer): void
+estimateCharge():void
-charge
Charge
-LOOKUP_ACTIVITY_UNIT =0.1 double
-MOVE_ACTIVITY_UNIT = 5.0 double
-activity_unit_price: double
-markup_percentage:double
-num_lookups: int
-extra_cost: double
-activity_cost: double
-activity_units: double
-total_cost: double
-total_charge: double
-service_fee:double
+calculateActivityUnits(floors_moved: int): double
+ calculateCharge(floorsMoved: int, destination_floor: int): void
+ lookupServiceFee(floor: int): double
-roundTwoDecimalPlaces(number: double): double
WifiModemQuerier
-SINGLE_INSTANCE:WifiModemQuerier
-success_lookups:int
-failed_lookups:int
-latest_service_fees:double[]
+ instantiateServiceFees(service_fees: double []): void
+ getInstance(installedFloor: int) : WifiModemQuerier
+ getInstance() : WifiModemQuerier
+ setWifi_modem(wifi_modem: WifiModem): void
+lookupServiceFee(floor:int):double
+updateFloorFee(floor:int, newFee:double) void
WifiModem
+forwardCallToAPI_LookupPrice(onFloor :int): double
+getInstance(installedOnFloor : int): WifiModem
-modemQuerier
-robots
mailPool
-mailItem
MailPool
+addToPool(mailItem:MailItem):void
+loadItemsToRobot(): ItemTooHeavyException
+registerWaiting(robot:Robot):void
+updatePools() : void
-loadItem(i: ListIterator<Robot>): ItemTooHeavyException
+registerWaiting(robot: Robot): void
+setChargeThreshold(chargeThreshold: double): void
-pool
-priorityPool
Item
-destination : int
<<interface>> ItemComparator
+ compare(i1: Item, i2 : Item): int
MAIL_DELIVERED
-automail
Simulation
-num_robots : int
-charge_threshold: double
-charge_display: boolean
-activity_unit_price: double
-markup_percentage: double
-mail_to_create: int
-mail_max_weight: int
-mail_delivered: ArrayList<MailItem>
-total_delay = 0
-total_billable_activity = 0 double
-total_activity_cost = 0 double
-total_service_cost = 0 double
-total_billable_acitivity : double
-total_activity_cost : double
-total_service_cost : double
+printResults() : void
+deliver(mailItem) : void
-setUpProperties(): IOException
-calculateDeliveryDelay(deliveryItem: MailItem): double

(Figure 2) - Design Class Diagram

Diagram Source:
https://lucid.app/lucidchart/invitations/accept/inv_67fda2c7-b710-4698-858a-1fa4d8a4d8b2