SWEN30006 Software Modelling and Design Project 1: Automail

Tute: Monday 3:15 - 5:15pm

Finnbar Howard - 1082205

James Hollingsworth - 915178

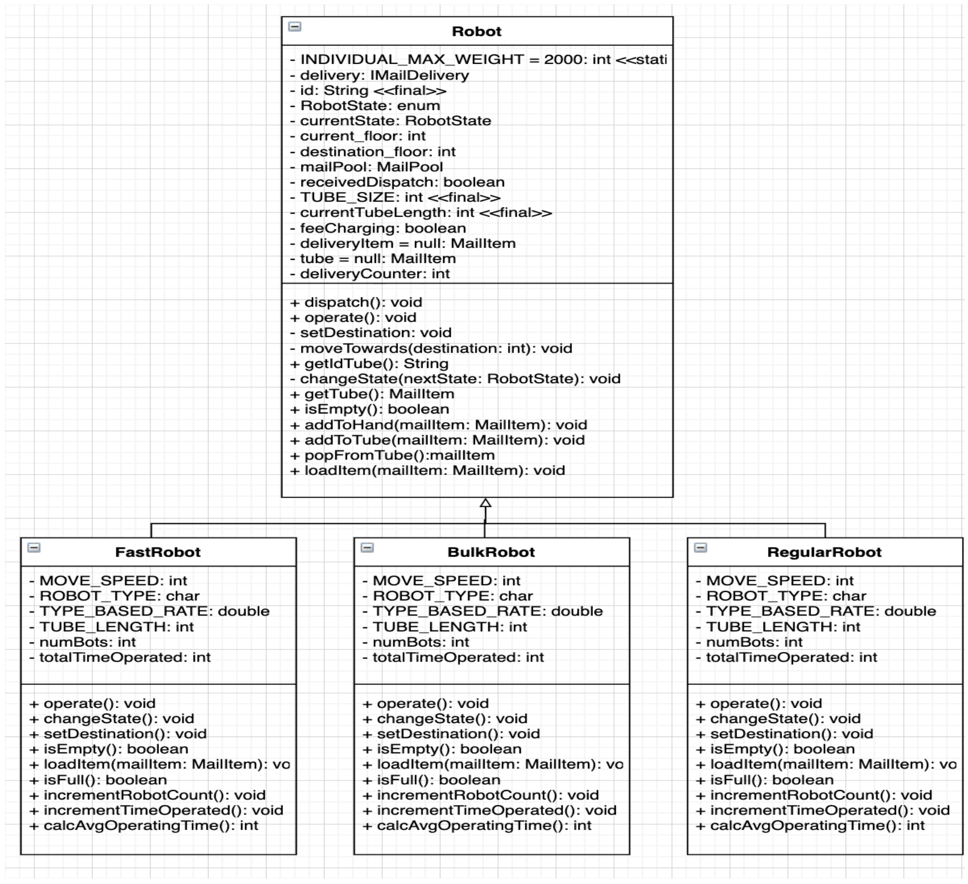
Oliver Bestel de Lezongard - 914956

Introduction

Automail is an automated mail sorting and delivery system incorporated into a large building with a dedicated mailroom. The existing system is composed of two main components. The MailPool and the Delivery Robots. MailPool holds the items to be mailed within the building. The Delivery Robots, currently composed of only “regular robots”, deliver the mail to the specified location. The focus of the project is to update the system with an extensible design that allows for new types of robots, and a charge capability. The purpose of this report is to explain the GRASP principles that have been used to incorporate these two aspects while maintaining an extensible design.

Implementation of New Types of Robots

When considering the implementation of new robots, two aspects must be considered. The updates to allow for these new robot types, as well as the possibility of further types being implemented in the future. To implement the fast and bulk robots as new types, there are required variations in the existing system, and thus the class is a variation point. A focus on an extensible design also requires the class to be considered as an evolution point so as to allow for new types of robots possibly being implemented in the future. Having identified the Robot class as a point of change, and to ensure that these variations of robot types do not have an undesirable impact on other elements, we have implemented the **Protected Variations principle.** This has been done through the implementation of a stable means of access to the types of robots. The Robot class, depicted below, has been changed to act as an abstract parent class to FastRobot, BulkRobot, and RegularRobot. Methods such as operate and changeState, and attributes such as TUBE_LENGTH, TYPE_BASED_RATE, and MOVE_SPEED have been implemented in the child class so as to increase extensibility, as future robots may not operate in the same manner, speeds, or loading capacities as current robots. This use of **Protected Variations** implements **Polymorphism**, to handle alternatives based on type, to create pluggable software components, and to allow low coupling that provides protection at a variation point.

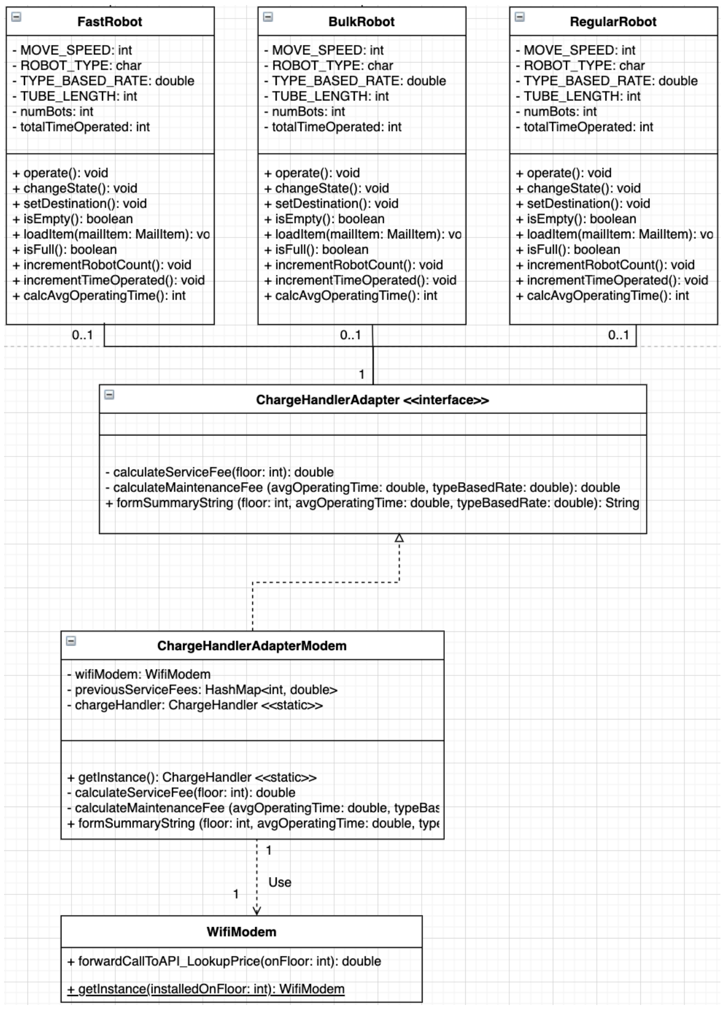


<u>Implementation of Charge Capability</u>

The charge capability is calculated as follows:

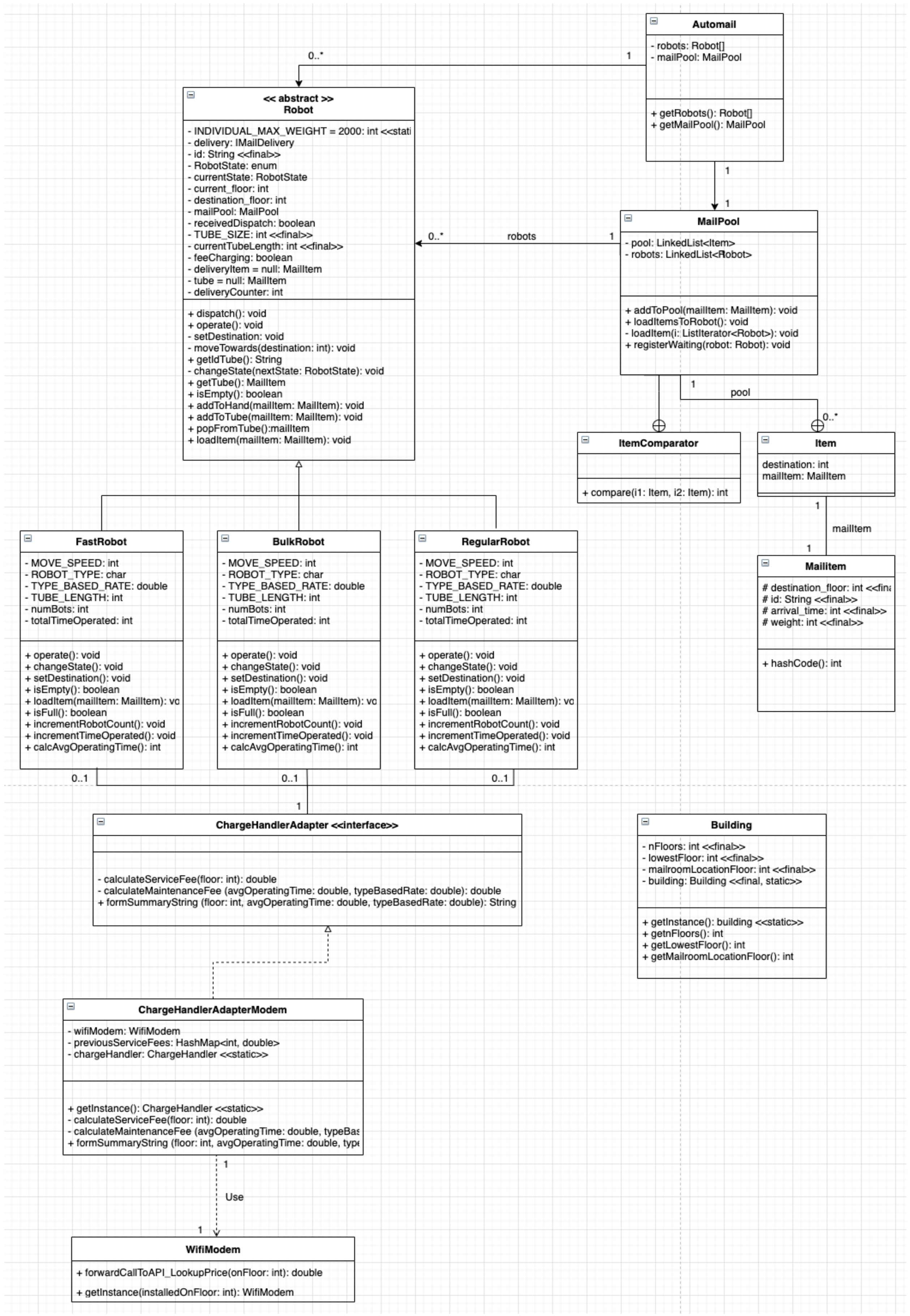
TotalCharge = Service Fee + Est. Maintenance Cost

Thus we can see it requires access to two pieces of information. The service fee, which is specific to the building, and the building floor. The maintenance cost, which is calculated by considering the average operating time of all robots of a specific type within the building Automail system. This requires access to the WifiModem class, as well as the robot type classes. This raises two concerns. Who is responsible for calculating the TotalCharge? And how do we achieve this so as not to violate high cohesion and low coupling? These two concerns can be addressed through a specific type of **Indirection**, namely **Pure Fabrication**. This is achieved through fabricating a ChargeHandlerModem class as a **Singleton**, which calls on WifiModem to get the information pertaining to the building, and building floor. ChargeAdapterModem is a singleton as it needs to capture state in a hash table for when there is possibly no service fee lookup available. The ChargeHandlerModem class is then called within the operate method for a specific type of robot, so as to combine the Est. Maintenance Cost specific to the robot type, and the Service Fee specific to the building, and building floor. Apart from calculating the charge, there are also considerations regarding future changes to the BMS and its WifiModem library. The ChargeHandlerAdapter interface allows a stable connection to, and creation of, a class specific to the method of lookup, and is extensible for implementation of further WifiModems or other means of service fee lookups. Furthermore, the concrete class holds previousServiceFees should the network that connects to the WifiModem fail. From this perspective we have implemented the **Adapter** pattern which uses **Protected Variations** with **Polymorphism** and **Pure Fabrication** so as to handle undesirable impacts around a point of instability, predicted variation, and extensibility while adhering to low coupling.

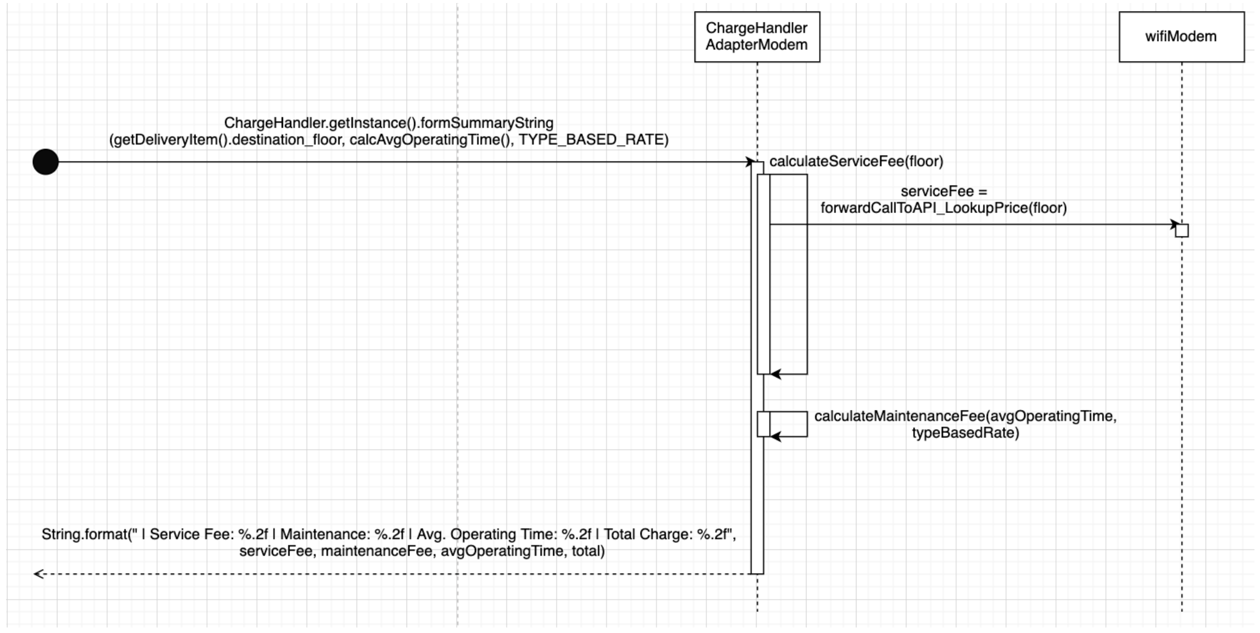


FastRobot
- MOVE_SPEED: int
- ROBOT_TYPE: char
- TYPE_BASED_RATE: double
- TUBE_LENGTH: int
- numBots: int
- totalTimeOperated: int
+ operate(): void
+ changeState(): void
+ setDestination(): void
+ isEmpty(): boolean
+ loadItem(mailItem: MailItem): vo
+ isFull(): boolean
+ incrementRobotCount(): void
+ incrementTimeOperated(): void
+ calcAvgOperatingTime(): int
BulkRobot
- MOVE_SPEED: int
- ROBOT_TYPE: char
- TYPE_BASED_RATE: double
- TUBE_LENGTH: int
- numBots: int
- totalTimeOperated: int
+ operate(): void
+ changeState(): void
+ setDestination(): void
+ isEmpty(): boolean
+ loadItem(mailItem: MailItem): vo
+ isFull(): boolean
+ incrementRobotCount(): void
+ incrementTimeOperated(): void
+ calcAvgOperatingTime(): int
RegularRobot
- MOVE_SPEED: int
- ROBOT_TYPE: char
- TYPE_BASED_RATE: double
- TUBE_LENGTH: int
- numBots: int
- totalTimeOperated: int
+ operate(): void
+ changeState(): void
+ setDestination(): void
+ isEmpty(): boolean
+ loadItem(mailItem: MailItem): vo
+ isFull(): boolean
+ incrementRobotCount(): void
+ incrementTimeOperated(): void
+ calcAvgOperatingTime(): int
0..1
0..1
0..1
1
ChargeHandlerAdapter <<interface>>
- calculateServiceFee(floor: int): double
- calculateMaintenanceFee (avgOperatingTime: double, typeBasedRate: double): double
+ formSummaryString (floor: int, avgOperatingTime: double, typeBasedRate: double): String
ChargeHandlerAdapterModem
- wifiModem: WifiModem
- previousServiceFees: HashMap<int, double>
- chargeHandler: ChargeHandler <<static>>
+ getInstance(): ChargeHandler <<static>>
- calculateServiceFee(floor: int): double
- calculateMaintenanceFee (avgOperatingTime: double, typeBas
+ formSummaryString (floor: int, avgOperatingTime: double, type
1
Use
1
WifiModem
+ forwardCallToAPI_LookupPrice(onFloor: int): double
+ getInstance(installedOnFloor: int): WifiModem

Design Class Diagram


Automail
- robots: Robot[]
- mailPool: MailPool
+ getRobots(): Robot[]
+ getMailPool(): MailPool
0..*
1
<< abstract >>
Robot
- INDIVIDUAL_MAX_WEIGHT = 2000: int <<stati
- delivery: IMailDelivery
- id: String <<final>>
- RobotState: enum
- currentState: RobotState
- current_floor: int
- destination_floor: int
- mailPool: MailPool
- receivedDispatch: boolean
- TUBE_SIZE: int <<final>>
- currentTubeLength: int <<final>>
- feeCharging: boolean
- deliveryItem = null: MailItem
- tube = null: MailItem
- deliveryCounter: int
+ dispatch(): void
+ operate(): void
- setDestination: void
- moveTowards(destination: int): void
+ getIdTube(): String
- changeState(nextState: RobotState): void
+ getTube(): MailItem
+ isEmpty(): boolean
+ addToHand(mailItem: MailItem): void
+ addToTube(mailItem: MailItem): void
+ popFromTube():mailItem
+ loadItem(mailItem: MailItem): void
1
1
0..*
robots
1
MailPool
- pool: LinkedList<Item>
- robots: LinkedList<Robot>
+ addToPool(mailItem: MailItem): void
+ loadItemsToRobot(): void
- loadItem(i: ListIterator<Robot>): void
+ registerWaiting(robot: Robot): void
1
pool
0..*
ItemComparator
+ compare(i1: Item, i2: Item): int
Item
destination: int
mailItem: MailItem
1
mailItem
1
MailItem
# destination_floor: int <<fina
# id: String <<final>>
# arrival_time: int <<final>>
# weight: int <<final>>
+ hashCode(): int
FastRobot
- MOVE_SPEED: int
- ROBOT_TYPE: char
- TYPE_BASED_RATE: double
- TUBE_LENGTH: int
- numBots: int
- totalTimeOperated: int
+ operate(): void
+ changeState(): void
+ setDestination(): void
+ isEmpty(): boolean
+ loadItem(mailItem: MailItem): vo
+ isFull(): boolean
+ incrementRobotCount(): void
+ incrementTimeOperated(): void
+ calcAvgOperatingTime(): int
BulkRobot
- MOVE_SPEED: int
- ROBOT_TYPE: char
- TYPE_BASED_RATE: double
- TUBE_LENGTH: int
- numBots: int
- totalTimeOperated: int
+ operate(): void
+ changeState(): void
+ setDestination(): void
+ isEmpty(): boolean
+ loadItem(mailItem: MailItem): vo
+ isFull(): boolean
+ incrementRobotCount(): void
+ incrementTimeOperated(): void
+ calcAvgOperatingTime(): int
RegularRobot
- MOVE_SPEED: int
- ROBOT_TYPE: char
- TYPE_BASED_RATE: double
- TUBE_LENGTH: int
- numBots: int
- totalTimeOperated: int
+ operate(): void
+ changeState(): void
+ setDestination(): void
+ isEmpty(): boolean
+ loadItem(mailItem: MailItem): vo
+ isFull(): boolean
+ incrementRobotCount(): void
+ incrementTimeOperated(): void
+ calcAvgOperatingTime(): int
0..1
0..1
0..1
1
ChargeHandlerAdapter <<interface>>
- calculateServiceFee(floor: int): double
- calculateMaintenanceFee (avgOperatingTime: double, typeBasedRate: double): double
+ formSummaryString (floor: int, avgOperatingTime: double, typeBasedRate: double): String
Building
- nFloors: int <<final>>
- lowestFloor: int <<final>>
- mailroomLocationFloor: int <<final>>
- building: Building <<final, static>>
+ getInstance(): building <<static>>
+ getnFloors(): int
+ getLowestFloor(): int
+ getMailroomLocationFloor(): int
ChargeHandlerAdapterModem
- wifiModem: WifiModem
- previousServiceFees: HashMap<int, double>
- chargeHandler: ChargeHandler <<static>>
+ getInstance(): ChargeHandler <<static>>
- calculateServiceFee(floor: int): double
- calculateMaintenanceFee (avgOperatingTime: double, typeBas
+ formSummaryString (floor: int, avgOperatingTime: double, type
1
Use
1
WifiModem
+ forwardCallToAPI_LookupPrice(onFloor: int): double
+ getInstance(installedOnFloor: int): WifiModem

Design Sequence Diagram


ChargeHandler AdapterModem
wifiModem
ChargeHandler.getInstance().formSummaryString (getDeliveryItem().destination_floor, calcAvgOperatingTime(), TYPE_BASED_RATE)
calculateServiceFee(floor)
serviceFee = forwardCallToAPI_LookupPrice(floor)
calculateMaintenanceFee(avgOperatingTime, typeBasedRate)
String.format(" | Service Fee: %.2f | Maintenance: %.2f | Avg. Operating Time: %.2f | Total Charge: %.2f", serviceFee, maintenanceFee, avgOperatingTime, total)

Conclusion

Regarding the implementation of new robot types, we have used the **Protected Variations** principle paired with **Polymorphism** to implement new types of robots to the current AutoMail system. **Protected Variations** allows for protection from variation due to new robot types, while **Polymorphism** allows for an extensible design for the possible implementation of new robot types, and low coupling. To implement a charge capability, the **Adapter** pattern uses **Protected Variation, Pure Fabrication** and **Polymorphism** to allow for low coupling, extensibility, and stability around a predicted point of variation due to changes in the BMS WifiModem library, or network failures.