Project 1 Design Report - Team: Workshop 10 Group 9

Aidan Western (1083407), Yunhao Fang (1067868), Mahee Hossain (1080102)

To extend the Automail system and include the required additional functionality, multiple changes were made to the source code. To implement the ability to charge residents, a new function (calculateCharge) was written to calculate the charge for the delivery of an item, with an additional helper function (calculateActivityCost). By the Information Expert GRASP Pattern, the responsibility for operating those functions was placed in the MailItem class. This implementation of the calculateCharge function also allows for the extension of the formula to account for the weight of an item and the delay in its delivery, as under the current implementation, both can be accessed directly within the function, making alterations to the input parameters and output type unnecessary.

The calculateCharge function is used for charging residents, and estimating the delivery charge for the purposes of determining delivery priority. The delivery priority system is implemented as a pair of linked lists, highPriority and lowPriority, with delivery items sorted into them based on their estimated delivery charge compared to the threshold charge, and each list is sorted in the same manner as in the original implementation, based on the proximity of the destination floor. This allows the original functionality to be maintained when the charge threshold is set to 0, as then all items will be placed in the highPriority list and sorted based on how close their destination floor is.

Calculating the delivery charge requires determining the service fee of the floor, which is handled via the api call function in the modem class. The system will repeat this process if the connection fails, until it succeeds.

We used the GRASP Polymorphism Pattern to handle the implementation of the altered delivery output log, via the creation of an alternate ChargeReportDelivery through the IMailDelivery Interface, which prints the extra information for the altered output, and is instantiated in the place of the original ReportDelivery class if the input properties file states that the extra information in the output log is requested.

For the implementation of the total statistics tracking, we created the artificial class StatisticTracker, thus following the Pure Fabrication GRASP pattern. StatisticTracker held all the static variables tracking the total statistics, and also held methods to update and print all statistics, thus making it a highly cohesive class, and also ensuring that the low coupling principle is not violated.

In our implementation of statistics tracking, the total activity cost output includes the cost for all failed lookups that occur during attempts to get the service fee of a floor, despite the fact that we do not pass the cost of failed lookups on to the customer. The failed lookups are included in the total costs as though it may be company policy to not charge the customer for these lookups, the company would naturally have to pay for these lookups, thus they should be tracked in the final cost. Additionally, the cost of any lookups done by Automail to calculate the expected cost of a particular delivery item, for the purposes of determining if it should have priority or not, though not associated with a delivery, is included in the final cost. This is done to track the total operating costs of the system, which the company running the system will likely want to know, even if the costs of that are not passed on the recipients of the delivery.

When calculating the delivery charge, our robot includes the amount of activity units it will accrue if it simply had to return to the mail room immediately after the delivery. This was done because the activity cost is supposed to include the cost of the round trip, if the currently delivered item was the only one being delivered, so accounting for the return trip is necessary.

High cohesion was maintained across the project as seen through the MailItem and StatisticTracker classes. MailItem contained all the responsibilities around calculating the charge for each delivery, and thus remained a cohesive class. Furthermore, StatisticTracker contained all the variables and methods related to the total statistics, therefore also being a cohesive class. Through conserving high cohesion, future system changes are accounted for, as high cohesion contributes to ease of adaptation (if a change is needed to charge or statistics tracking, we just have to change MailItem or StatisticTracker). Through the realisation of high cohesion, and the use of GRASP Patterns such as Polymorphism, the principle of Low Coupling was also maintained, as changes to classes would make minimal impact on other classes.