

### Charge Display:

A Class called CostCalculator is created. It calculates the charge of a delivery including service cost and activity costs, the markup percentage and the activity cost per unit is configurable using different creators or setters if needed in future.

CostCalculator is a static object in Simulation class initialized; upon a successful delivery, the function ReportDelivery(MailItem) is called in Simulation. The costCalculator functions then can be called within Simulation. The CostCalculator then takes the MailItem as input, and calculates the costs using the formula given in project spec. If the CHARGE\_DISPLAY== true, the output now includes the costs, else the system prints the original outputs.

The class is created for the purpose of high cohesion, the class is focused on calculating the costs of delivery, thus avoiding assigning more responsibility to different classes, which enhances understanding of the code. CostCalculator does not originally exist, and in order to maintain low coupling and high cohesion, the class is fabricated regarding to the GRASP pure fabrication and it does not change the structure of the original code, it calculates the costs, and returns the costs only.

### Charge Threshold:

In order to implement the priority delivering if the expected costs exceeds the charge threshold. The MailPool class now has a new static attribute CostCalculator, a new static double attribute CHARGE\_THRESHOLD, both is passed on from Simulation by calling their setter functions.

To keep track of the expected cost of a mailitem, the expectedCost is now a double attribute of the class MailItem, default at 0. The expected cost is calculated when a Mailitem is added to the pool (using the function addToPool(MailItem mailItem)) using the costCalculator and stores it in MailItem.expectedCost. Since the MailPool is sorted everytime a new mailitem enters the pool using a Comparator<Item>, by editing the comparator function, the item that exceeds the charge threshold can be thrown to the front of the pool. In the comparator function, When the CHARGE\_THRESHOLD is above 0, it checks both mailitem's expected costs, if mailitem is expectedCharge exceeds the CHARGE\_THRESHOLD, that item should be in front of the pool, thus returns (smaller), if both exceeds or not exceeds the expected cost, the items are compared as original manner.

This way of implementing the priority function adheres to the original structure of the code, minimal changes were made, and it only turns on if ChargeThreshold is > 0;.

### Statistics Tracking:

Since the required statistics relate to deliveries, and the CostCalculator class has access to the relevant information, the statistics are attributes of the CostCalculator class. At the start of the simulation the statistics will have a default value (0) and are updated as deliveries are made.

Each time cost is calculated upon delivery, each statistic value is modified accordingly. With the exception of the number of lookups, all of the statistics are able to be modified correctly from information that CostCalculator already has. In the case of number of lookups, the number of attempts for lookups and failed lookups are retrieved from the wifi log.

The statistics are stored as attributes in the CostCalculator class in accordance with the high cohesion, low coupling GRASP principle. The CostCalculator class' purpose is to perform relevant calculations as items are delivered. The main calculation is of course the cost of individual deliveries, but making auxiliary calculations such as the running statistics relevant to deliveries and costs are still in line with the class' purpose. Low coupling is achieved as almost all of the required information for the statistics is already available to the CostCalculator class, and it is only dependent on the wifi log to look up values for the number of lookups statistics.

In a similar vein, the principle of information expert also informed the decision to store the statistics in the CostCalculator class, as the required information for those statistics is already available there.

The fact that a fabricated class exists to keep track of relevant statistics means that it is easy to modify the formulas for the current statistics, or easily add in more statistics in future. The stattrak functionality can be turned on or off by editing the automail.properties, simply by setting StatTrack=true or false.