SWEN30006 Project 1

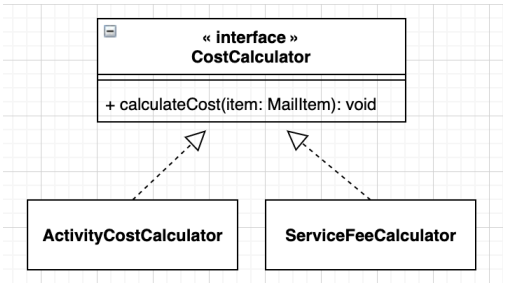
Workshop nn, Team mm

Name1 Student#
Name2 Student#
(2 member team)

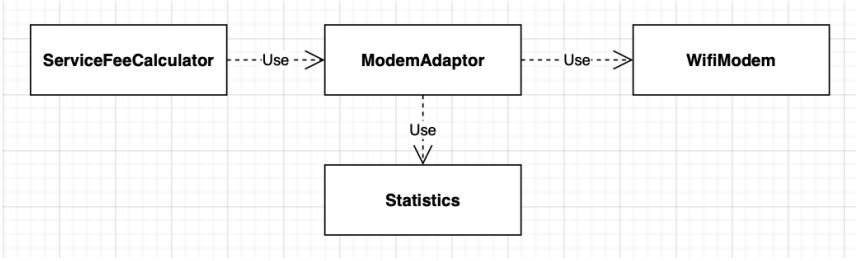
To preserve the functionality of the program, we implemented the new features by adding new components to the given framework. Our goal was to make minimal modifications to the existing framework to avoid overloading the original classes with responsibilities and reducing their cohesion. The following modules were created to accommodate these new functionalities: Cost, CostCalculator (ServiceFeeCalculator & ActivityCostCalculator), ModemAdaptor and Statistics.

The **Cost** class was created to manage the *Charge*-related functionalities for a delivery that would otherwise be naturally implemented in the MailItem class (as a *Mail Item* naturally has a *Charge*). This creation of Cost adheres to the principle of Pure Fabrication and increases the cohesion of MailItem by reducing its responsibilities. MailItem was delegated the role of Creator for Cost, and also stores the reference to Cost, as it was determined that the MailItem represents an item being delivered, and Cost would not exist without that construct (i.e. MailItem compositely aggregates Cost). Cost provides access to a complete breakdown of all the elements that make up a *Charge* for a delivery, however, the specific calculations of these elements are delegated to separate components called CostCalculator's to increase the cohesion of the system.

The **CostCalculator** interface sets the rule that any CostCalculator object in the program must have a polymorphic operation calculateCost() which calculates its respective cost/fee for a certain purpose. This method takes a MailItem as input, hence, future implementations that make use of item weight or any other metric for calculations can easily be implemented. For these given specifications, an ActivityCostCalculator and a ServiceFeeCalculator were created, both of which are delegated responsibilities as their name suggests. The CostCalculator interface allows future implementations to create extra components for calculating any other fees that may be required, thus allowing the design to be extendable and polymorphic.



The **ModemAdaptor** class interacts with the WifiModem class that is provided. This new class was designed with the Protected Variation principle in mind, where WifiModem is considered an external component whose functionality may change in the future. Hence, the design makes use of the Indirection principle to create a single point of interaction with this external source, meaning any interaction to the modem must be made through the ModemAdaptor class. This helps to minimise the risk of instability from such a component, while reducing the coupling between this component and the rest of the system. By Information Expert, ModemAdaptor has access to lookup outcomes from the modem, therefore, it is assigned the responsibility of updating Statistics with lookup successes and failures.



The **Statistics** class is responsible for keeping track of the various statistics required by the specifications such as items delivered, total activity, total service fees and breakdown of lookups. This class was designed following a Singleton pattern to ensure that only one instance of it would exist, as the statistics are calculated for the entire simulation. Furthermore, the global nature of the design allows for the easy collection of information from different components, while leaving flexibility for any additional collection of statistics that may be implemented in the future.

Additional design decisions/changes:

- The functionality of comparing priorities between items was added to the ItemComparator class as a natural extension to this existing comparator class. Furthermore, items within the same priority category are sorted using the existing algorithm (furthest destination first) as it was assumed to generally be a quicker way to deliver all the mail (smaller trips can be completed while a big trip is taking place).
- The logging of *Charge* information was integrated into the ReportDelivery class since it already has the responsibility of outputting the delivery log.
- Since the Simulation class does not perform any calculations relating to *Charge*, the *Charge_Threshold* was moved into the Cost class where all other *Charge*-related information is kept, increasing the cohesion of both classes.
- The lookup to check for service fee is done twice, once upon arrival at the MailPool and once upon delivery of the item, this is to keep the cost of lookups down. If the lookup fails, the previous lookup value for that floor is used. We reasoned that if a customer is making significant loss because of poor wifi connection, it would be wiser in the long run to upgrade the wifi rather than to do more lookups.

Below is the design model containing the components that we created.