# SWEN30006 Assignment 1 - Report

Workshop 09, Team 02
Lucas Fern & Cameron Maddern

Implementing the required changes to the Automail system did not require the creation of any new classes in any of the existing `automail`, `exceptions`, or `simulation` packages. Various attributes and methods were, however, added to the existing classes, and existing classes were modified. These additions will be detailed below, broken down by class. They are also highlighted in yellow on the Design Class Diagram attached at the end of the report.

## 1 Simulation

The `WifiModem` was moved out of the simulation class and into the Robot. This requires the Simulation class to import the Robot, but this slight increase in Coupling is easily justified by the significant increase in Cohesion. This is mentioned again in Section 3.1.

### 1.1 Attributes

The following attributes were added to the Simulation class:

- `private static double billableActivity`
  `+ private double activityUnitsToDeliver`
  `+ private static double activityCost`

  Each of these is a statistic required to be printed at the end of the simulation. Because they aren't used elsewhere it was Cohesive to place them in the Simulation class.

## 2 Automail

The Automail constructor was modified to set an initial service fee for each floor. This incurs an upfront cost that cannot be passed on to the consumers, but is justified, as once an initial fee is set for each floor we can guarantee only one extra call will be used per delivery since a default value now exists.

## 3 Robot

The `operate()` method of the Robot class was modified so that the Robot would attempt to update the service fee of the floor whenever it was completing a delivery. The `moveTowards()` method was also updated so that it added the cost of each movement to the relevant variables of its MailItems.

### 3.1 Attributes

The following attributes were added to the Robot class:

- `public static final double UNITS_PER_FLOOR = 5;`
  `+ public static final double UNITS_PER_LOOKUP = 0.1;`

  These constants are required to update the amount of activity units for the delivery of each MailItem. They were placed here in line with the Information Expert principle, since the robot uses these after taking the relevant action to update the attributes of the items it is carrying.

- `public static WifiModem wModem`

  In this implementation of Automail the Robots are the only classes which interact with the Wi-Fi Modem except for powering it on and off. The Modem was moved into the Robot class to make them the Information Experts and increase Cohesion.

- `private static double[] serviceFees`

  This array stores the service fees most recently retrieved by a robot for each floor. This needs to be accessed by all robots, and isn't specific to each one. It was made `static` to the Robot class to adhere to the High Cohesion and Information Expert patterns.

### 3.2 Methods

The following method was also added to the Robot class:

- `public static int attemptSetServiceFee(int floor)`

  This method interacts with other `static` attributes and methods of the Robot class. It was highly cohesive to place this method here.

# 4 MailItem

The existing `toString()` method of the MailItem was overloaded to take an argument specifying whether to print the charge statistics. The default `toString()` method was overridden to call the overloaded method with backwards compatible behaviour. This was an example of protected variation as the old interface was maintained.

## 4.1 Attributes

The following attributes were added to the MailItem class:

- `private double serviceFee`
  `+ private double activityUnitsToDeliver`

  Each MailItem stores, updates, and operates on its own service fee and required activity units, so these were placed to obey the Information Expert principle and maintain High Cohesion.

- `public static final double ACTIVITY_PRICE = 0.224;`
  `+ public static final double MARKUP_PROP = 0.059;`

  These attributes are used in the calculation of charges upon successful deliveries, as well as in charge estimation for determining priority items. These calculations are performed by the MailItem, so it was highly cohesive to store the constants here. It was also mentioned that the system should be designed so that the weight of the items could be factored into the charge in the future. Since the weight is stored here also, this decision takes into account the cohesion of future updates.

## 4.2 Methods

The following method was also added to the MailItem class:

- `public static int attemptSetServiceFee(int floor)`

  This method interacts with other `static` attributes and methods of the MailItem class. It was highly cohesive to place this method here.

- `public void increaseActivityUnitsToDeliver(double increase)`
  `+ private double calculateTotalActivityUnits()`
  `+ private double estimateActivityToDeliver()`
  `+ public double calculateActivityCost(double activityUnits)`
  `+ private double calculateCost(double activityUnits)`
  `+ public double calculateCharge(double activityUnits)`

  These methods all operate on and make calculations based on an individual MailItem and its properties such as destination and the activity units required to move it there. These methods must take acitivtyUnits as there are multiple ways of calculating this value (estimate and actual value) It was in line with the Information Expert principle to place these methods in the same class.

# 5 MailPool

The `loadItem()` method of the MailPool was modified to support priority dispatch of MailItems with a charge above the threshold defined in the configuration file. By keeping the same interface into the `loadItem()` method, this variation was protected from causing adverse effects elsewhere in the Automail system.

## 5.1 Methods

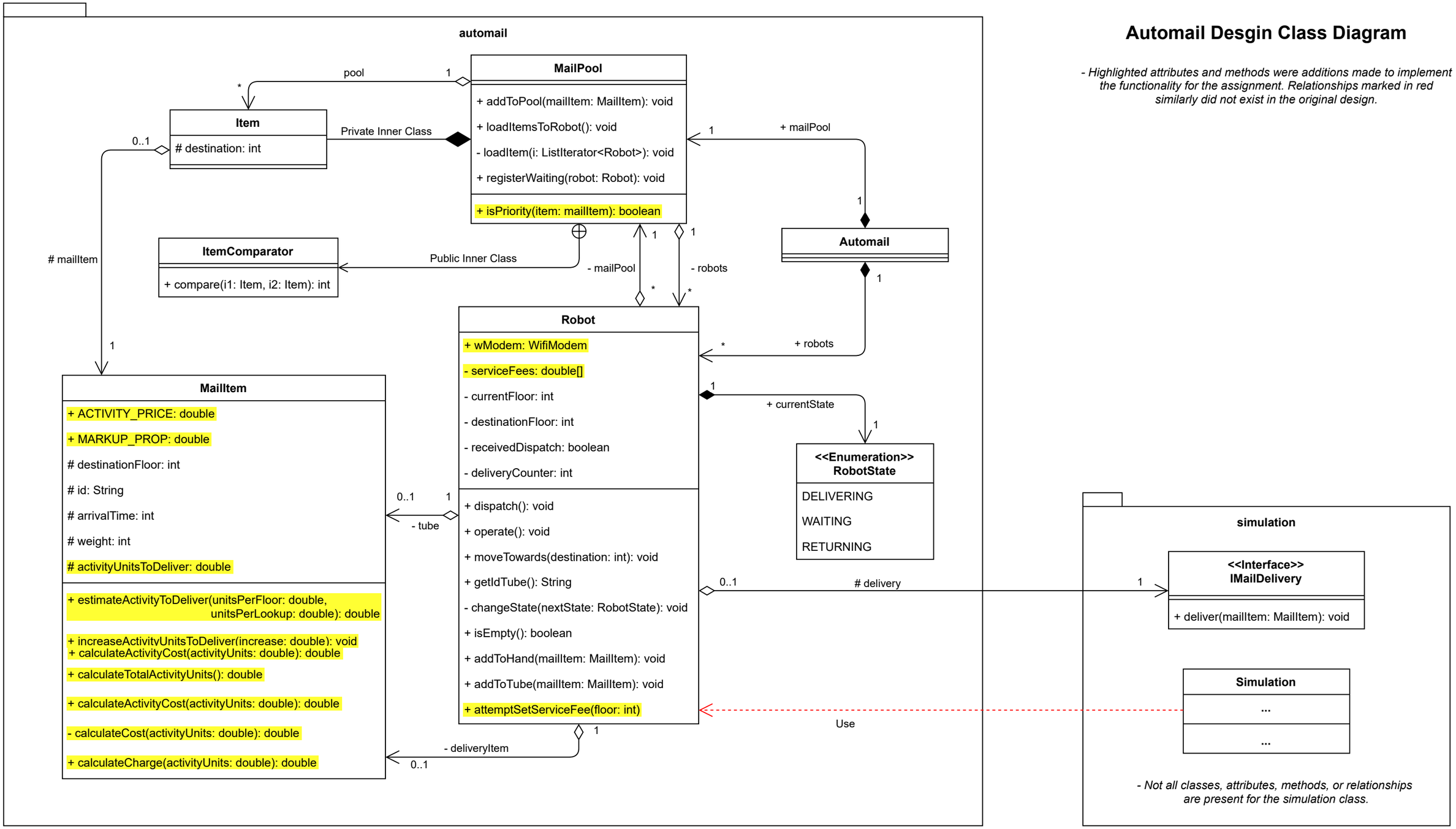
The following method was also added to the MailPool class:

- `private boolean isPriority(MailItem item)`

  This method compares the charge on a mail item to the threshold value for priority mail. It was highly cohesive to place this method here.

# 6 Additional Choices

- It was decided that customers would be charged an amount proportional to the amount of floors that the robot travelled with their item, and if the robot was carrying multiple items both customers would be charged for each movement. This is because in a real life environment customers would feel ripped-off if they usually received a 50% discount from the robot delivering two items at once, and then occasionally were charged the full price when the robot was forced to make a trip with only one item. On top of that, providing no discount is profitable for Delivering Solutions Inc.

# Automail Desgin Class Diagram

*- Highlighted attributes and methods were additions made to implement the functionality for the assignment. Relationships marked in red similarly did not exist in the original design.*