SWEN30006 Software Modelling and Design
Project 1: Automail
- Project Specification -

School of Computing and Information Systems
University of Melbourne
Semester 2, 2021

Background: Automail

Delivering Solutions Inc. (DS) has recently developed and provided a Robotic Mail Delivery system called *Automail* to the market. Automail is an automated mail sorting and delivery system designed to operate in a large building that has a dedicated mail room. The system offers end-to-end receipt and delivery of mail items within the building and can be tweaked to fit many different installation environments. The system consists of two key components:

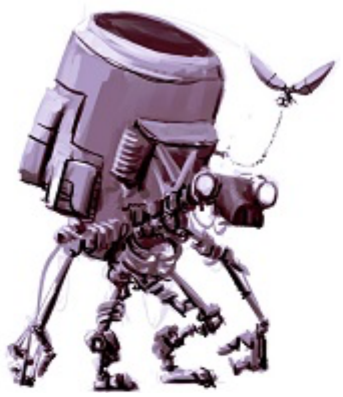


Figure 1: Artistic representation of one of DS robots

- A **MailPool** subsystem which holds mail items after their arrival at the building's mail room. The mail pool decides the order in which mail items should be delivered.

- **Delivery Robots** which take mail items from the mail room and deliver them throughout the building. The currently used robot has two hands and one tube, i.e., a backpack-like container attached to each robot for carrying items (see Figure 1). The robot can hold one item in its hands (i.e., two hands carry one item) and one item in its tube. If a robot is holding two items (i.e., one in its hands and one it its tube) it will always deliver the item in its hands first. An installation of Automail can manage a team of delivery robots of any reasonable size.

DS provides a **simulation** subsystem to show that Automail can operate to deliver mail items within a building. The subsystem runs a simulation based on a property file and shows the delivery log of the robots and delivery statistics, e.g., the elapsed time before each mail item is delivered and how long it has taken for all the mail items to be delivered. The system generates a measure of the effectiveness of the system in delivering all mail items, considering the time to deliver mail item as well as the types of mail items. *You do not need to be concerned about the detail of how this measure is calculated.*

The simulation subsystem uses a clock to simulate operations of the mail pool and robot subsystems. Broadly speaking, for each tick of the clock (i.e. one unit of time), the mail pool subsystem will load items to the robots if there are robots available at the mailroom; and the robots will either move to deliver an item (if there are items in their hands or tubes), deliver an item, or move to return to the mailroom (if all items are delivered). Currently, the robots offered by *DS* will *take one unit of time when moving one step* (i.e., moving up or down one floor in a building) and *one unit of time to deliver an item*. For example, if a mail destination is 4 floors from the mailroom, a robot will take 4 units of time for moving to the delivery floor, plus 1 unit of time for delivery, plus 4 units of time for returning to the mailroom.

You can assume that the robot hardware has been well tested and will work with Automail. The current version of Automail seems to perform reasonably well. However, the implementation and design of Automail are not well documented.

Your Task

Due to the COVID-19 and the high demand of delivery services, *DS* wants to update their Automail to support (1) new robot types, and (2) a charge capability. To pilot a change towards new capabilities, *DS* has hired your team to extend the latest version of Automail to include the ability to have multiple types of robots and to charge tenants upon successful delivery of mail items.

(Task 1) New robot types: *DS* has bought two new types or robots: ***Fast*** robots and ***Bulk*** robots in addition to the regular robots that DS is currently using.

- Fast robots can carry only 1 item for a delivery (it doesn't have tube). However, a fast robot moves ***three times*** faster than the regular robot, i.e., the fast robot can move *at most* 3 floors for 1 unit of time.
    - For example: If a mail destination is 4 floors from the mail room, a fast robot will take 2 units of time for moving to the delivery floor (i.e., 1 unit for 3 floors and 1 unit for the remaining 1 floor), plus 1 unit of time for delivery, plus 2 units of time for returning to the mailroom.
- Bulk robots can carry 5 items for a delivery (all 5 items are in the tube; no hands). The bulk robots move with the same speed as the regular robot (1 unit of time per floor). The items are placed on top of each other in their tubes, so the delivery order is last in first out.

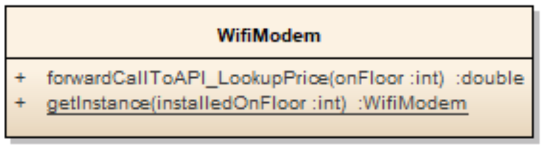
There is no difference in the mail assignment for each type of robot. Also, mail items are assigned to robots on a first come first served basis, i.e., the first robot (regardless of its type) in the waiting queue is assigned mail item(s) and is out for delivery. Note that a robot should be assigned mail items as much as it can (according to its capacity) for delivery unless no mails left in the mailroom at the loading time.

(Task 2) A charge capability: The government now allows building owners and *DS* to charge additional service fees to tenants. Therefore, *DS* wants to extend Automail to calculate a total charge from (1) a service fee specified by their customers (i.e., the building owners) and (2) a robot maintenance cost.
The charge is formulated as follows:

TotalCharge = Service Fee + Est. Maintenance Cost

Service Fee is varied by servicing floors (i.e., the destination floor of a mail item) and controlled by the building owner. Thus, Automail needs to perform remote lookups to an external Building Management System (***BMS***) at the time of delivery to retrieve the most recent service fee.

For security reasons, the building owner will provide a binary library (a jar file) to connect to their modem for looking up a service fee calculated by BMS.


WifiModem
+ forwardCallToAPI_LookupPrice(onFloor :int) :double
+ getInstance(installedOnFloor :int) :WifiModem

- ***getInstance(installedOnFloor: int)*** initiates the connection with the modem and returns an instance of WifiModem, where ***installedOnFloor*** is the floor number where the mailroom is located.

- **f*orwardCallToAPI_LookupPrice(onFloor: int*)** returns the service fee for the floor as a double value, where ***onFloor*** is the floor number that the robot is delivering a mail item.

Note that:

- Service fees are not static; they can increase (but not decrease) during a delivery session.
- Lookups depend on network connectivity and so the lookup request can fail. The failed request is indicated by a *negative* value returned by the forwardCallToAPI_LookupPrice function.
- If the lookup request fails, the most recent service fee retrieved by any other robot for the same floor should be used. In the case of no service fee retrieved for this floor before the value of the service fee should be 0.
- *DS* requires the Automail update be built for fault tolerance: the delivery **must** be successful with fees charged.
- As Automail is responsible for the infrastructure, it is only reasonable to charge the tenant one such service fee per mail item delivery.

Est. Maintenance cost is calculated based on the average robot usage based on a robot type using the following formula. Note that the maintenance cost is calculated ***after*** a service fee is successfully retrieved from BMS.

*Est. Maintenance cost = Type-based rate * Avg. operating time.*

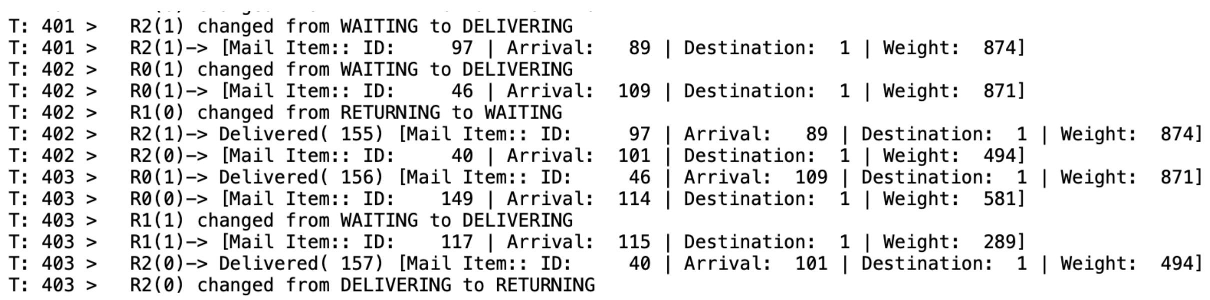
- Type-based rate = $0.025 for a regular robot, $0.05 for a fast robot, and $0.01 for a bulk robot
- Avg. operating time is the average number of steps that all the robots with the same type as the current robot have operated in the past (counting all units of times that the robots are in the delivering and returning states, and not when the robots are in the waiting state).

For example: A regular robot A is about to delivery an item to a tenant. The charge calculation process is as follow:

- Looks up a service fee from BMS. Let's assume that the service fee is $20.
- Asks all regular robots for their operating time. Let's assume that two regular bots are used: a regular robot A has operated for 200 units, and a regular robot B has operated for 100 units. Hence, Avg. maintenance cost = $0.025 * (200+100)/2 = $3.75
- TotalCharge = $20 + $3.75 = $23.75

Output Log

Figure 2 shows a sample log output of the current version. *DS* would also like you to **adjust** the log so that they can see how the new capabilities are being used.



```
T: 401 >   R2(1) changed from WAITING to DELIVERING
T: 401 >   R2(1)-> [Mail Item:: ID:     97 | Arrival:   89 | Destination:  1 | Weight:  874]
T: 402 >   R0(1) changed from WAITING to DELIVERING
T: 402 >   R0(1)-> [Mail Item:: ID:     46 | Arrival:  109 | Destination:  1 | Weight:  871]
T: 402 >   R1(0) changed from RETURNING to WAITING
T: 402 >   R2(1)-> Delivered( 155) [Mail Item:: ID:     97 | Arrival:   89 | Destination:  1 | Weight:  874]
T: 402 >   R2(0)-> [Mail Item:: ID:     40 | Arrival:  101 | Destination:  1 | Weight:  494]
T: 403 >   R0(1)-> Delivered( 156) [Mail Item:: ID:     46 | Arrival:  109 | Destination:  1 | Weight:  871]
T: 403 >   R0(0)-> [Mail Item:: ID:    149 | Arrival:  114 | Destination:  1 | Weight:  581]
T: 403 >   R1(1) changed from WAITING to DELIVERING
T: 403 >   R1(1)-> [Mail Item:: ID:    117 | Arrival:  115 | Destination:  1 | Weight:  289]
T: 403 >   R2(0)-> Delivered( 157) [Mail Item:: ID:     40 | Arrival:  101 | Destination:  1 | Weight:  494]
T: 403 >   R2(0) changed from DELIVERING to RETURNING
```

Figure 2: Sample log and output of the current version.

Adjust #1: Add Robot type to the log

The original version records the robot id and the number of items in the tube (see the number in parenthesis):

`R0(0)` means a regular robot (id 0) have no item in the tube.

`R1(1)` means a regular robot (id 1) has one item in the tube.

For the fast and bulk robots, the log should show the records in the following format:

`F2(0)` means a fast robot (id 2) has no item in the tube.

Note: the number of items in the tube for the fast robot is always 0 because it doesn't have a tube.

`B3(5)` means a bulk robot (id 3) has five items in the tube.

Note that the numerical id of the robot is incremental across all robots regardless of their types.

Adjust #2: Add charge, a fee, and a maintenance cost to the log
Additional information should be appended to the current log item **upon a successful delivery** as shown below in **blue** for the following example**:**

```
T: 402 >   R2(1)-> Delivered( 155) [Mail Item:: ID:     97 | Arrival:   89 | Destination:  1 | Weight:
874 | Service Fee: 20.00 | Maintenance: 3.75 | Avg. Operating Time: 150.00 | Total Charge: 23.75]
```

Where

- Service Fee is the value retrieved from BMS lookup.
- Maintenance is an average maintenance cost (see the calculation above).
- Avg. Operating Time is the average operating time.
- Total Charge = Service Fee + Est Maintenance Cost
- Double values should be rounded to the nearest hundredth (2 digits after the decimal point).

Your task is to update the latest version of Automail developed by *DS* to show how the charge capability could be incorporated into the robot management system. In doing this, you should apply your software engineering and patterns/principles knowledge to refactor and extend the system to support this new behaviour. Note that the behaviour described in this document is just one possible use of this functionality. When designing your solution, you should consider that DS may want to extend their capabilities in the future. In addition, BMS and its WifiModem library may be altered or even replaced in the future. Hence, your design should be able to accommodate for the future DS capabilities as well as changes in the external systems without requiring major changes.

The Base Package

You have been provided with an Eclipse project export representing the current version of the system, including an example configuration file. This export includes the full simulation software for the Automail product, which will allow you to implement your approach to support the charge capability.

To begin:

1. Import the project zip file as you did for Workshop 1.
2. Try running by right clicking on the project and selecting **Run as... Java Application**.
3. You should see an output like the one in Figure 2 o the Eclipse console showing you the current behaviour of the Automail system.

This simulation should be used as a starting point. Please carefully study the sample package and ensure that you are confident you understand how it is set up and functions before continuing. If you have any questions, please make use of the discussion board or ask your tutor directly in the workshop as soon as possible; it is assumed that you will be comfortable with the package provided.

Note: By default, the simulation will run and generate mails randomly, meaning the results will be different on every run. To have a consistent test outcome, you need to specify the seed. You can do this in the configuration file (*automail.properties*). Any integer value will be accepted, e.g. 30006.

Configuration and Project Deliverables
(1) Extended Automail: As discussed above, and for the users of Automail to have confidence that changes have been made in a controlled manner, you are required to preserve the Automail simulation's existing behaviour. Your extended design and implementation must account for the following:

- Preserve the existing behaviour of the system for configurations where the additional capabilities is turned off in the configuration file (*automail.properties*), i.e., `FastRobots=0`, `BulkRobots=0`, and `FeeCharging=false`. Note that "preserve" implies identical output. We will use a file comparison tool to check this.
- Add the handling and delivering behaviour for the charge capability as described above.

It's recommended that you understand the high-level design of current system so that you can effectively identify & update relevant parts. And, you also don't need to refactor the whole system.

(2) Report: In addition to the extended Automail, *DS* also wants you to provide a report to document your design changes and justification of your design. More detail of the report is available on the LMS submission page.

Note: Your implementation must not violate the principle of the simulation by using information that would not be available in the system being simulated. For example, it would not be appropriate to use information from the simulation package (e.g., mail items which have not yet been delivered to the mail pool). We also reserve the right to award or deduct marks for clever or very poor code quality on a case-by-case basis outside of the prescribed marking scheme.

Testing Your Solution
We will be testing your application programmatically, so we need to be able to build and run your program without using an integrated development environment. *The entry point must remain as "swen30006.automail.Simulation.main()". You must not change the names of properties in the provided property file or require the presence of additional properties*.

Note: It is **your team's responsibility** to ensure that the team has thoroughly tested their software before submission.

Submission
Detailed submission instructions will be posted on the LMS. *You must include your team number in all your pdf submissions, and as a comment in all changed or new source code files provided as part of your submission.*