Part A - Mailbot Blues

SWEN30006, Semester 1 2017

Overview

You, an independent Software Contractor, have been hired by *Robotic Mailing Solutions Inc.* to provide some much needed assistance in delivering their latest product to market, Automail. The Automail is an automated mail sorting and delivery system designed to operate in large buildings that have dedicated Mail rooms. It offers end to end sorting, storage and delivery of all mail within the large building, and can be tweaked to fit many different installation environments. The system consists of four key components:

- **A Mail Pool** system which can hold a number of packages that has just arrived.
- A **Delivery Robot** (see figure below) which delivers mail items from the mail pool throughout the building.
- A **Storage Unit**, that is, a 'backpack' which is attached to the delivery robot. It can contain at most four units.
- **A Mail Sorting** system which decides what packages should go into a robot's backpack for delivery.

Figure 1: Artistic representation of our robot

The hardware of this system has been well tested. Unfortunately, the performance seen so far has been less than optimal. *Robotic Mailing Solutions* has traditionally been a hardware company, and as such do not have much software development experience. As a result, the strategies that they are using to organise the mail and select the mail for delivery are very poor.

Your job is to apply your software engineering knowledge to develop new strategies for collecting and sorting the mail to improve the performance of their system. Once you have created your strategies, you must benchmark your strategies and provide feedback on your performance to *Robotic Mailing Solutions*.

Other useful information

- A **mail item** is labelled with a **priority level (low, medium or high)**.
- A **mail item** has a **size (large, medium or small)**.
  - **Large** mail corresponds to **four** units
  - **Medium** mail item corresponds to **two** units
  - **Small** mail item corresponds to **one** unit
- A **Storage Unit** can contain **at most four units in total**.
- The **Delivery Robot** can be sent to deliver mail even if the **Storage Unit** is not full.

The Sample Package

You have been provided with a zip file containing all you will need to start your work for *Robotic Mailing Solutions*. This zip file includes the full software simulation for the Automail product, which will allow you to test your strategies in an experimental environment. This zip file provides you the application as an Eclipse project. To begin:

1. Extract the zip files's contents to your desired location
2. Open Eclipse
3. Import a new Eclipse project (File >> Import ... >> General >> Existing Projects into Workspace)
4. Select the unzipped folder as the directory (and Next as required to complete the import)
5. Try running `Simulation.java`, you should see an output similar to this:

```
Simulation complete!
Final Delivery time: 367
Final Score: 406.5
```

Figure 2: sample results

This will run the provided simulation and print output showing you the current performance of the Automail system. This simulation should be used as a starting point in developing your benchmark program for your strategies. Along with the code you have also been provided with compiled versions of the Javadoc for this package, located in the `doc` folder. You should use this as your guide in developing your solution.

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 as soon as possible; it is assumed knowledge that you will be comfortable with the package provided. In particular, you should note the scoring function used to calculate the cost of a delivery.

On Calculation of Stats

The sample package has the capability to run simulations and provide a stats summary after the simulation has finished. It is important that you understand how these statistics are calculated so that there is no misunderstanding that leads you to incorrect assumptions about the behaviour of your strategies. In particular, the "Final Score" is the performance measure you need to improve: lower for this score is better.

> **Note** By default, the simulation will run without a fixed seed, meaning the results will be random on every run. In order to have a consistent test case, it would be good to specify the seed. You can do this by editing the Run Configurations under Eclipse and adding an argument. Any integer value will be accepted, i.e. 30006

Strategies Package Diagram

Seen below is a diagram of the strategies package. There are other classes included in this diagram as well. Be aware that this does **not** represent all the classes in Simulation. This diagram is provided below to aid in the understanding of the application.

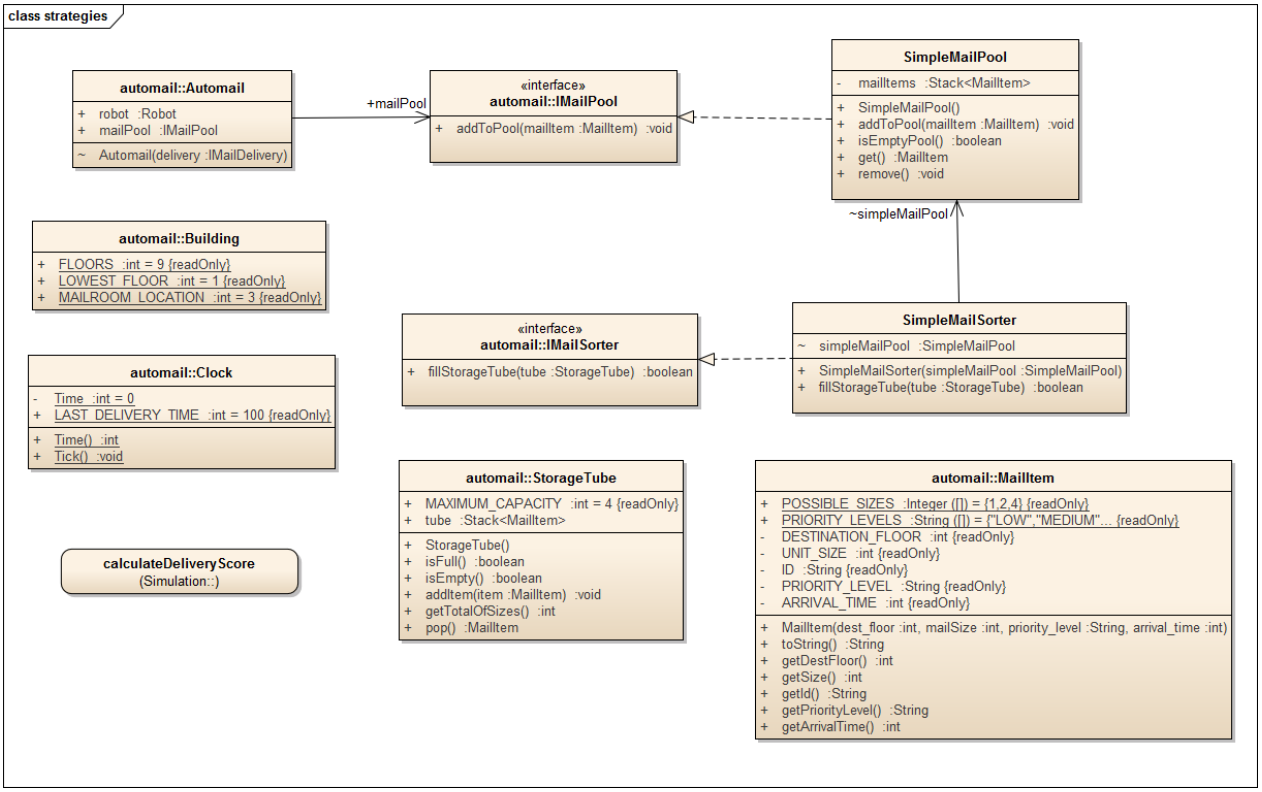

Figure 3: Sample Diagram

The Task

As you can see, the current strategies used for the collection and sorting of mail are very simple and not at all optimised for any particular use case. Thankfully, *Robotic Mailing Solutions Inc.* has made use of the Strategy pattern (see reference **here**) to make their software more flexible and extensible in the future, by creating a common interface for the collecting and sorting of mail.

Your task is to develop new collection and sorting strategies for mail within the Automail system. *Robotic Mailing Solutions inc.* is particularly interested in improvements to the `MailSorter`, as internal testing has shown this to be a significant factor on current performance. Specifically you must:

1. Create a class in the strategies package that implements the `IMailPool` interface.
2. Create a class in the strategies package that implement the `IMailSorter` interface.
3. Modify the class Automail (that is, revise Automail.java) so that it uses your MailPool and MailSorter strategies

You must include in your code, comments explaining the rationale for your data structure and algorithm choices and how these work toward achieving your goals for the `MailSorter` and `MailPool` implementations. It is important to note that the strategies you provide *must* be different from those provided to you in the sample package, and must achieve a better (lower) value for the "Final Score" statistic.

> **Note** You will not be marked on the quality of your algorithms (that is, how well you achieve the stated goal). This is not an algorithms subject so we do not expect optimal planning solutions. You must, however, be able to demonstrate that you have made an attempt to address the problem by organising the incoming mail in the mailpool and using that organisation in selecting mail in the mailsorter. In addition, you must not violate the principle of the simulation by using information that would not be available to simulated system, for example by using information about mail items which have not yet been delivered to the mail pool.

We will be using our own version of the sample package during marking in which we will use our own fixed seed. You will be submitting **only** the three files listed above, which must work with the rest of the sample package. You should also test with different values, but should not otherwise modify the sample package.

Submission Verification

On the LMS, there is a file called build.zip which you need to extract to a folder and include two other files in them.

- **Part A.zip** (which is the original project file you downloaded off the LMS)
- **submission.zip** (a zip file which includes the **three** files you need to submit)

Refer to the README.txt file in build.zip for more details on building and running your program on the University lab computers. In particular note the fact that you should develop your project solution in Eclipse, not use the script for development!

If we cannot run your program you will receive at most 1 mark out of the 5 available for this project.

> **Note** Your program **must** run on the University lab computers (using Java 8). It is **your responsibility** to ensure you have tested in this environment before your submit your project.

Implementation Checklist

- Defined a class that implements the MailPool interface.
- Defined a class that implements the MailSorter interface
- Modified Automail.java so that the simulation uses the above classes
- All submitted code commented and of good quality (see below)
- Tested your program on a University computer to ensure compatibility

Marking Criterion

This project will account for 5 marks out of the total 100 available for this subject. These will be broken down as follows:

Criterion	Mark
Simulation compiles and runs correctly as per specification above	2.5 marks
Code quality, as described below.	2.5 marks

We expect to see good variable names, well commented functions, and inline comments for complicated code. We also expect good object oriented design principles and functional decomposition. If we find any significant issues with code quality we may deduct further marks.

We also reserve the right to award or deduct marks for clever or poor code quality on a case by case basis outside of the prescribed marking scheme.

There is also a *bonus mark* available for submissions in the top 5% of performance. We will be judging performance by comparing the "Final Score" statistic for your strategy. (The lower the score the better).

On Plagiarism

We take plagiarism very seriously in this subject. You are not permitted to submit the work of others under your own name. This is an **individual** project. More information can be found here: (https://academichonesty.unimelb.edu.au/advice.html).

Submission Date

This project is due at **11:59pm on the 22nd of March.** Any late submissions will incur a 1 mark penalty per day unless you have an appropriate reason with supporting documents. If you have any issues with submission, please email William at tiow@unimelb.edu.au, before the submission date.