Part A - Mailroom Blues

SWEN30006, Semester 1 2016

Overview

You, an independent Software Contractor, have been hired by *Integrated 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 three key components:

- **A Mail Sorting** system, which receives all incoming mail and classifies it based on a set of rules.
- **A Mail Storage** system, which can store a fixed number of packages in a fixed number of containers after they are sorted.
- A fleet of **Delivery Bots**, that retrieve containers of mail from the storage system and deliver the packages throughout the building.

The hardware of this system has been well tested, unfortunately the performance seen so far has been less than optimal. *Integrated 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 sort mail, select mail for delivery and delivery the mail throughout the building are very poor.

Your job is to apply your software engineering knowledge to develop new strategies for sorting, selecting and delivering 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 *Integrated Mailing Solutions*.

The Sample Package

You have been provided with a zip file containing all you will need to start your work for *Integrated 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. To begin, navigate to the root directory of this folder and compile the Simulation program like so:

```
javac com/unimelb/swen30006/mailroom/Simulation.java
```

You can now run the provided sample simulation with:

```
java com.unimelb.swen30006.mailroom.Simulation
```

Or if you want detailed output logging:

```
java com.unimelb.swen30006.mailroom.Simulation detailed
```

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, though you may choose to ignore this if you like. 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 number of steps is a value calculated as a function of the number of floors traversed and the number of floors delivered on. Delivery to a floor counts as 1 step, regardless of how many packages are delivered on that floor. Travelling up one floor on the elevator is considered 2 steps. Therefore, the action of travelling from the first floor, to the 10th floor, and delivering packages on the tenth, before returning to the first would be a total of 37 steps (9 floors up, 1 for delivery, 9 floors down).

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.

The Task

As you can see, the current strategies used for selection of, sorting of and delivery of mail are very simple and not at all optimised for any particular use case. Thankfully, *Integrated 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 selection, sorting and delivery of mail.

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

- 2 classes that implement the `SortingStrategy` interface and use two entirely different algorithms for sorting mail.
- 1 class that implements the `DeliveryStrategy` interface and focuses on minimizing the average floors visited per delivery run.
- 1 class that implements the `SelectionStrategy` interface and focuses on minizing the number of delivery runs required to deliver all mail.

You must include in your code, comments explaining the rationale for your algorithmic choices and how your given algorithms work toward achieving your goals for the `DeliveryStrategy` and `SelectionStrategy` implementations. It is important to note that the strategies your provide *must* be different from those provided to you in the sample package. Likewise, the two provided SortingStrategy classes must also be based on different algorithms with different results. If your strategy gets exactly the same value for a given simulation as a provided simulation, it will be considered equivalent and you will not receive any marks for it.

> **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 and as such, we do not expect optimal planning solutions. You must, however, be able to demonstrate that you have made an attempt to solve the constraint.

You will also be required to modify the existing Simulation class to run a series of simulations based on command line input, as described in the following section. Other than this file (Simulation.java) you **must not** modify any other files provided in the sample package. We will be using our own version of the sample package during marking, if you modify any of the other provided files your solution will likely fail to compile when tested and receive 0 marks.

Simulation Driver

You are required to either modify the existing Simulation driver or write your own new simulation driver to test your new strategies under the following scenarios.

- Large Building: A 200 floor (from 1-200) building with the mail room located on the 2nd floor. This building is served by 20 delivery bots, and has a mail storage room that can hold 50 boxes, each of which can hold 20 units.
- Medium Building: A 50 floor (from 1-50) building with the mail room located on the 20th floor. This building is served by 10 delivery bots and has a mail storage room that can hold 10 boxes, each of which can hold 30 units.
- Small Building: A 10 floor (from 1-10) building with the mail room located on the 10th floor. This building is served by 1 delivery bot and has a mail storage room that can hold 30 boxes each of which can hold 40 units.

Your program should take the type of simulation (as one of `large_building`, `medium_building` and `small_building`) as the first command line argument when running your simulation. You will be responsible for determining which Strategy to use for selection, delivery and sorting for each scenario. The function to run the simulations has been provided to you in the simulation class, but it is your responsibility to parse command line arguments and determine the appropriate arguments to this function.

You must also take an optional second command line argument "random" that determines whether or not a different random sequence of mail items is generated (the default is to repeat the same sequence for each run). Finally, the existing optional command line argument "detailed" (for printing detailed output) can be last, and default to not printing detailed output if this argument is not provided.

The simulation function also provides the functionality to run the simulations n times and average the results. For this project you must run the simulations exactly 10 times. In all simulations you will be required to run a simulation that sorts 1000 mail items. The printing of, and averaging of results is already implemented for you.

Building and Running Your Program

Your program should not rely on any external libraries, and should be able to be compiled using the same command as the sample package:

```
javac com/unimelb/swen30006/mailroom/Simulation.java
```

We will expect to be able to run your program by calling the main class, Simulation, as provided in our sample code. Therefore, your program should run by simply invoking:

```
java com.unimelb.swen30006.mailroom.Simulation big_building random detailed
```

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 7). It is **your responsibility** to ensure you have tested in this environment before your submit your project.

Reflection on Design Suitability

Part of being a good software developer is your ability to discuss and reason about the merits of design choices for a particular problem and present the positives and negatives of those choices so that you, and the team you are working in, are able to make cogent decisions on which to implement. There is often many possible solutions to a given design problem, and the design that we have given you here in the sample package is one possible solution (and not necessarily the best solution) to the Automail design challenge.

As part of your solution, you are required to provide a basic design analysis of what we have provided you, and provide an argument for or against it being a suitable choice to solve the problem. We do not expect a formal argument to be presented (this will come later in the semester), but we do expect to see clear and concise reasoning and justification for your arguments. This can include bullet points (e.g. for pros and cons, or key design issues), and should be no more than 300 words in length. It can be provided as either a PDF or text file.

Implementation Checklist

- Defined 2 different classes that implement the SortingStrategy interface.
- Defined a class that implements the DeliveryStrategy interface.
- Defined a class that implements the SelectionStrategy interface.
- Simulation driver has been modified to appropriately handle command line input arguments.
  - Simulation driver tests the small building scenario.
  - Simulation driver tests the medium building scenario.
  - Simulation driver tests the large building scenario.
  - Simulation driver allows for enabling or disabling random seeds.
- Test 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 handles input arguments appropriately	1 mark
Two correct, working, implementations of the SortingStrategy interface	1 mark
A correct, working, implementation of the DeliveryStrategy interface	1 mark
A correct, working, implementation of the SelectionStrategy interface	1 mark
Reflection on design solution	1 mark

We also reserve the right to award or deduct marks for clever or poor implementations 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 on the large building scenario. We will be judging performance by comparing the total effort of your strategies, that is the average number of steps taken to deliver packages multiplied by the number of delivery runs required.

Further, we expect to see good variable names, well commented functions, 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.

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 **11:59 p.m. on the 27th of March.** Any late submissions will incur a 1 mark penalty per day unless you have supporting documents. If you have any issues with submission, please email Mat at mathew.blair@unimelb.edu.au, before the submission date.