SWEN30006 Software Modelling and Design
Project 1: Robomail Revision

School of Computing and Information Systems
University of Melbourne
Semester 2, 2019

Overview

You and your team of independent Software Contractors have been hired by *Robotic Mailing Solutions Inc. (RMS)* to provide some much needed assistance in delivering the latest version of their product *Automail* to market. 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 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:

1. **Delivery Robots** (see Figure 1) which take mail items from the mail room and deliver them throughout the building. Each robot can hold one item in its "hands" and one item in its "tube" (a backpack-like container attached to each robot). 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 (including zero!).
2. A **Mail Pool** 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.

The hardware of this system has been well tested, and the current software seems to perform reasonably well but is not well documented. Currently, the robots offered by *RMS* are only able to move up or down one floor in a building in one step. However, *RMS* recently developed a new capability for their robots which allows them to go into an 'overdrive' mode. This 'overdrive' mode allows the robots to move twice as fast (i.e., two floors up or down in one step) to deliver packages. However, due to the increased load on the robotic hardware when using overdrive, there are two limitations placed on the robots when they use the overdrive mode:

Figure 1: Artistic representation of one of our robots

1. A robot using overdrive mode can only deliver one package. That is, it can carry a package in its hands for delivery, but can not store an additional package in its tube.
2. Once a robot has delivered an item using overdrive, it needs to 'cool down' for 5 steps before it can return to the mail room.

Because of these limitations *RMS* only wants to use the overdrive system to deliver packages of a very high priority. Other packages should be delivered as normal.

Your task is to update the simulation software maintained by *RMS* to show how the overdrive ability could be incorporated into the robot management system. In doing this you should apply your software engineering and patterns knowledge to refactor and extend the system to support this new behaviour. Note that the overdrive behaviour described in this document is just one possible use of this functionality. When designing your solution you should consider that *RMS* may want to incorporate or trial other uses of the overdrive ability in the future.

The behaviour of the system when delivering packages without a very high priority should not change. Additionally, *RMS* expect the behaviour of the system to remain **exactly** the same when the overdrive ability is turned off, or if no very high priority packages arrive.

RMS would also like you to add some statistics tracking to the software so that they can see how the overdrive ability is being used. This will help them to understand the wear and tear that their robots are going to go through. For now, *RMS* would like you to record:

1. The **number** of packages delivered normally (i.e., not using overdrive)
2. The **number** of packages delivered using overdrive
3. The **total weight** of the packages delivered normally (i.e., not using overdrive)
4. The **total weight** of the packages delivered using overdrive

When the simulation ends you should print this information to the console.

As with the overdrive behaviour, you should apply software engineering and patterns knowledge to support this statistics tracking. Your design should consider that *RMS* may want to track additional statistical information relating to robot performance in the future.

Once you have made your changes, your revised system will be benchmarked against the original system to provide feedback on your results to *RMS*.

Other useful information

- The **mailroom** is on the ground floor (floor 0).
- All **mail items** are stamped with their **time of arrival**.
- Some mail items are **priority** mail items, and carry a priority from **10 (low)** to **100 (high)**.
- Priority mail items arrive at the mailpool and are registered one at a time, so **timestamps are unique for priority items**. Normal (non-priority) mail items arrive at the mailpool in batches, so **all items in a normal batch receive the same timestamp**.
- The mailpool is responsible for giving mail items to the robots for delivery.
- A **Delivery Robot** carries at most two items, but it can be sent to deliver mail if it is only carrying one item.
- All mail items have a **weight**. Delivery robots can carry items of any weight, and place items of any weight in their tube.

- The system generates a measure of the effectiveness of the system in delivering all mail items, taking into account time to deliver and priority. You do not need to be concerned about the detail of how this measure is calculated.
- If a priority mail item has a priority of **over 50**, it is considered to have a **very high priority**. These are the mail items that should be delivered using the overdrive ability (if the overdrive ability is enabled).
- A robot using the overdrive ability will move two floors each step, unless it is only one floor away from its destination. Then it will move just one floor each step. So, a robot using overdrive to deliver a package from the mail room (which is on floor 0) to floor 9, it would go to floors 2, 4, 6, 8, then 9.

The Sample Package

You have been provided with an Eclipse project export representing the current system, including an example configuration file. This export includes the full software simulation for the Automail product, which will allow you to implement your approach to supporting the delivery robot's new overdrive functionality. 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 similar to that in Figure 2 showing you the current performance 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 as soon as possible; it is assumed that you will be comfortable with the package provided.

```
T: 439 >   R1(1) changed from WAITING to DELIVERING
T: 439 >   R1(1)-> [Mail Item:: ID:     10 | Arrival:   93 | Destination:  1 | Weight: 1278]
T: 439 > Delivered( 157) [Mail Item:: ID:    101 | Arrival:   46 | Destination:  1 | Weight:  289]
T: 439 >   R2(0) changed from DELIVERING to RETURNING
T: 440 > Delivered( 158) [Mail Item:: ID:     73 | Arrival:   48 | Destination:  1 | Weight: 1188]
T: 440 >   R0(0)-> [Mail Item:: ID:     15 | Arrival:   72 | Destination:  1 | Weight: 1308]
T: 440 > Delivered( 159) [Mail Item:: ID:     10 | Arrival:   93 | Destination:  1 | Weight: 1278]
T: 440 >   R1(0)-> [Mail Item:: ID:    142 | Arrival:  117 | Destination:  1 | Weight: 1859]
T: 440 >   R2(0) changed from RETURNING to WAITING
T: 441 > Delivered( 160) [Mail Item:: ID:     15 | Arrival:   72 | Destination:  1 | Weight: 1308]
T: 441 >   R0(0) changed from DELIVERING to RETURNING
T: 441 > Delivered( 161) [Mail Item:: ID:    142 | Arrival:  117 | Destination:  1 | Weight: 1859]
T: 441 >   R1(0) changed from DELIVERING to RETURNING
T: 442 | Simulation complete!
Final Delivery time: 442
Final Score: 103952.89
```

Figure 2: Sample output

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 outcome, you need to specify the seed. You can do this in the configuration file or by editing the Run Configurations (Arguments tab) under Eclipse and adding an argument. Any integer value will be accepted, e.g. 11111 or 30006.

Project Deliverables

As discussed above, and in order 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 behaviour of the system for configurations where the overdrive ability is disabled and statistical logging is turned off (i.e., Overdrive=false;Statistics=false).
  (Preserve = identical output. We will use a file comparison tool to check this.)
- Add overdrive behaviour to deal with delivering very high priority mail items.
- Add statistics tracking to show the total weight and number of packages delivered with and without the overdrive behaviour.

In support of your updated and extended design, you need to also provide the following:

1. A **domain class diagram** for the robot mail delivery domain, as reflected in the revised simulation. There is no reason you can't be guided by your design class diagram, but be very careful that you end up with a description of the problem space, without reference to this particular solution.
2. A **static design model (design class diagram)** for your submitted system of **all changed and directly related components**, complete with (for the changed components) visibility modifiers, attributes, associations, and public (at least) methods. You should refer to this model in report. You may use a tool to reverse engineer a design class model from your implementation as a basis for your submission. However, your diagram must contain all relevant associations and dependencies (few if any tools do this automatically), and be well laid-out/readable. You should not expect to receive any marks for a diagram that is reverse engineered and submitted unchanged.
3. A **dynamic design model (design sequence diagram)** illustrating how a very high priority mail item is assigned to a robot for delivery, how that robot uses the overdrive ability to deliver the item, and how that robot returns to the mail room. You should self-assess your dynamic model on the basis of how useful it would be in explaining how the system works to a hypothetical new team member. Too little detail and it will have little value, too much detail and it won't add much value over reading the code. The choice of detail level is yours. Again, you should refer to this model in your report.
4. A **brief report**, describing the changes you have made with reference to your diagrams, and providing a rationale for these changes **using design patterns and principles**. Note that to do this well, you should be able to talk about other reasonable options you considered, and why you did not choose these options. Around 2-4 pages should be enough to provide a concise rationale.

Your submitted implementation must be consistent with your design models, and should include all appropriate comments, visibility modifiers, and code structure. As such, your submitted DCD and DSD must be consistent with your submitted implementation.

Hints There are three required tasks here – diagram, code, and report – which are interrelated. You should work on these tasks **as a team**, not separately. Every team members needs to be able to demonstrate their understanding of and contributions to all of the deliverables.

To help understand the task, you can start by putting together your domain model; you can use this project spec and a reverse engineered static design model for the existing system as a basis for this. You can then narrow down which parts of the system you need to understand in detail, and look at the static design and code for those parts. While you are doing this, you should be considering options for the changes you will require to refactor and extend the system: keep track of serious options for your report. When it comes to making changes, separate out the refactoring and extension. Be careful about the changes you make: reordering operations (especially those involving generation of random numbers) will

change the behaviour of the system and therefore the output. If you make such changes and proceed without detecting the problem, recovering may require substantial backtracking on the changes you've made or even reapplying them to the original system. As such we would recommend that you only make refactoring changes that are required to support improving the design.

You do not need to understand or even read all the code provided; you only need to understand the parts you are changing and those they depend on. So, for example, you really don't need to pay any attention to the `MailGenerator`.

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** Your program **must** run on the University lab computers. It is **your responsibility** to ensure you have tested in this environment before your submit your project.

Marking Criteria

This project will account for 12 marks out of the total 100 available for this subject. It will be assessed against the following rubric:

- H1 [10-12] Preserves original behaviour and implements new capability appropriately. Good extended or extendable design/implementation with generally consistent, clear, and helpful diagrams. Clear design rationale in terms of patterns. Few if any minor errors or omissions (no major ones).
- H2B-H2A [8.5-9.5] Preserves original behaviour with at most minor variations in output and implements new capability appropriately. Extended or extendable design/implementation with generally consistent, clear, and helpful diagrams. Fairly clear design rationale in terms of patterns. Few errors or omissions.
- P-H3 [6-8] Similar behaviour to original and implements new capability appropriately. Reasonable design/implementation with generally consistent and clear diagrams. Fairly clear design rationale with some reference to patterns. Some errors or omissions.
- F+ [2.5-5.5] Plausible implementation and reasonable design with related diagrams. Plausible attempt at design rationale with some reference to patterns. Includes errors and/or omissions.
- F [0-2.5] No plausible implementation. Design diagrams and design rationale provided but not particularly plausible/coherent.

> **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 about mail items which have not yet been delivered to the mail pool. It would also not be appropriate to violate the implied physical limitations, for example by having the robots teleport.

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.

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.

For UML diagrams you are expected to use UML 2+ notation.

> **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 a **group** project. More information can be found here: https://academichonesty.unimelb.edu.au/advice.html.

Submission

Detailed submission instructions will be posted on the LMS. You should submit all items shown in the checklist below. You must include your group number in all of your pdf submissions, and as a comment in all changed or new source code files provided as part of your submission.

Submission Checklist

1. Your complete updated source code package reflecting your new design and implementation (all Java source with top-level folder called "swen30006").
2. Design Analysis Report (pdf).
3. Static Domain Model (pdf or png).
4. Static Design Model reflecting your new design and implementation (pdf or png).
5. Dynamic Design Diagram reflecting your new design and implementation (pdf or png).

Submission Date

This project is due at **11:59p.m. on Friday September 13**. 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 Charlotte at charlotte.pierce@unimelb.edu.au **before** the submission deadline.