SWEN30006 Software Modelling and Design
Project 1: Robomail Revision

School of Computing and Information Systems
University of Melbourne
Semester 1, 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. 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 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:

- **Delivery Robots** (see Figure 1) which deliver mail items from the mail pool throughout the building. The robot can hold a single item in its "hands" and another in its *tube*, that is, the 'backpack' which is attached to each delivery robot. This tube can hold a single additional mail item, for delivery after the one held by the robot.
- A **MailPool** subsystem which holds mail items after their arrival at the building and which 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. In addition, the current system has a weight limit based on what a single robot can carry. However, the robots are capable of coordinating in pairs or triples to carry heavier items, although more slowly (one third of the speed; see details below). *RMS* wants to upgrade the software to coordinate robot teams to carry these heavier items. However, they do not want the behaviour of the system to change if there are no heavier packages as they are just starting to build trust with their customer base, and want this change to be a clear improvement. In addition, they still expect behaviour in handling the lighter mail items to be similar in the revised system, that is, the handling can't be universally poor just because of the presence of heavier items.

Figure 1: Artistic representation of one of our robots

Note that *RMS* don't expect optimal behaviour in the presence of heavier mail items (no fancy algorithm or optimisation is required); they just want heavier items delivered without degrading the current capability.

Your job is to apply your software engineering and patterns knowledge to refactor the system to more easily support this variation in behaviour (the individual/team behaviour), and then add the new team capability in a maintainable and modifiable form[1]. 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.
- All **mail items** are stamped with their **time of arrival**.
- Some mail items are **priority** mail items, and carry a priority from **100 high** to **10 low**.
- 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; **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. A Delivery Robot can be sent to deliver mail if its tube is empty.
- All mail items have a **weight** from **200 grams** up to a limit set as a system parameter. An item can be carried by one to three robots (one only in the current system). The weight capacities are represented in the code by the constants (see `Robot`):
  - INDIVIDUAL_MAX_WEIGHT = 2000
  - PAIR_MAX_WEIGHT = 2600
  - TRIPLE_MAX_WEIGHT = 3000
- 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.
- A team of robots (whether made up of two or three robots) moves at one third the speed of an individual robot; where an individual robot moves one floor every time step (robot.step()), the team will wait for two time steps before moving on the third time step.

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 coordinating robots to deal with heavy packages. To begin:

1. import the project zip file as you did for Workshop 1.
2. Try running by right clicking on the project `SWEN30006_2019S1_P1` and selection `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 (or an exception, depending on the particular run).

This simulation should be used as a starting point in developing your benchmark program for your strategies.

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.

[1] As you are aware, simply coding up lots of complex conditions does not result in maintainable, modifiable code.

```
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.

The Task

Extended Design and Implementation 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 of where the maximum mail item weight is limited to that transportable by a single robot. (Preserve = identical output. We will use a file comparison tool to check this.)
- Add team behaviour to deal with heavier mail items.

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 the design class diagram (as per below), 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, and as such it should be considered as part of your 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 robots in your system transition from operating individually to working as a team and back to working individuals again. You should self-assess your dynamic model on the basis of how useful it would be in explaining how the transition 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 the requested models, 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 reject, and why. Around 2-4 pages should be enough to provide a concise rationale.

Your submitted implementation must be consistent with your design models, and will be marked on code quality, so 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–report, refactor, and extend–which are interrelated. 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 exiting 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 (esp. 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 so that you can add the new team capability. Accordingly, 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`.

Building and Running Your Program We will be testing your application using the desktop environment, and need to be able to build and run your program automatically. 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 by using information about mail items which have not yet been delivered to the mail pool, or by violating implied physical limitations, for example by having the robots teleport or deliver as a team when they are on different floors.

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 **11:59 p.m. on Fri 3rd of May.** 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.
