Part B - Mailbot 2: Judgment Day

SWEN30006, Semester 1 2018

Overview

Continuing your work as software contractors, your group has been rehired by *Robotic Mailing Solutions Inc. (RMS)* to address further issues with their product AutoMail. As you are aware, RMS uses a simulation framework to test out strategies (for example, for MailPool and Robot operation). However, RMS's experience in improving such strategies (with your help) has made them aware that the simulation framework itself, while functional, is not up to standard from a design perspective. The designers did not consistently apply good design principles, and have ended up with a system which is not sufficiently flexible or modifiable.

You have been assigned three interrelated tasks:

1. Pass judgement on the existing software package, that is, provide an analysis of the package, highlighting poor design decisions.
2. *Refactor* the existing package applying design principles and patterns to improve the flexibility and modifiability of the simulation platform to cope with RMS's changing needs.
3. Extend the design to support some additional functionality and modify the existing simulation to match your extended design.

The Existing Simulation

You are familiar with the existing simulation from your earlier work. Other than the seed for random number generation, all parameters of the existing simulation are fixed in the code. This includes parameters such as the number of building floors, the number of mail items to be delivered, and the type of delivery robots. This does not allow RMS to vary the simulation to try out various combinations of parameters, such as: very tall building with high mail volume and two strong robots.

The Task

We will now break the three components of your task down in detail.

Analysis of Existing Simulation

This task is a design analysis report on the provided simulation. You should use your knowledge of object-oriented (OO) software design and OO code quality to detect design issues, that is, aspects where the design could be improved. We expect five design issues (see note below) to be identified. These issues should be reported in terms of the language of design patterns and principles. (You can use the analysis exercises from workshop 4 as a guideline.) The focus here is on the design, rather than the coding; we don't want to hear, for example, about your preference for a particular Java coding idiom unless it is also relevant to the design.

In this report you should be critical of the existing software, but ensure that you backup your claims and statements with reasoned arguments about the design and, where relevant, better alternatives. Simply stating something is bad is not sufficient for this task. Neither is simply stating that your modified design/code (see below) is better. You are encouraged to refer to or include UML models (or fragments) in your report where helpful to do so. *Your report should not exceed 1500 words.* Diagrams and appendix (see note below) are not counted towards this limit.

Note: The five issues you report should be those you believe most significant. If you detect and intend to fix more than five, feel free to include the additional ones in an appendix to your report. We do not have a list of five you are expected to find to get full marks; we are interested in how strongly you can justify your choices.

Refactored Design and Implementation

Having now critically analysed the provided package, your next task is to provide a refactored software design and implementation that addresses the concerns listed in your report. The refactored implementation must also support using the properties file "automail.properties" provided to you with the source code as a means of setting the parameters specified in the properties file. A start has already been made towards using the properties file, however the code for this, in "Simulation.java", has been commented out. (See also, Monopoly case study code Iteration 4).

Note that, as this is a refactoring, the behaviour of the system should remain unchanged for the combination of one weak robot and one strong robot as supported in the original system, that is, the output of the refactored (an extended) version should be identical to the output of the original for this combination.

Extended Design and Implementation

RMS has bought some new robots to supplement their collection. As well as being able to carry heavy items like the strong robots, these robots have an increased carrying capacity with a tube that holds up to six mail items. These new robots have been dubbed the "big" robots.

With the purchase of these robots, RMS has recognised the need to be able to model different robot configurations. The choice of configuration used in the simulation will be made through the property file by listing the two robot types to be used, selected from the list {weak, strong, big}. *(For simplicity, we will restrict configurations to exactly two robots).* The behaviour of the automail system will depend on the robot types, as follows:

1. In all configurations, one robot (upper) will deliver only to the top half of the building, while the other robot (lower) will deliver to the bottom half as well as throughout the building as described below.
2. The lower robot will deliver all priority items, no matter where in the building they go.
3. Two weak robots is considered an invalid configuration.
4. If one robot is weak, the weak robot will deliver to the upper half of the building and the lower robot will deliver all parcels too heavy for the weak robot.
5. If one robot is big and the other is strong, the big robot will deliver to the upper half of the building.

Note that this behaviour is consistent with that of the provided configuration of one weak robot and one strong robot. *As noted above, the behaviour of this configuration must not change.*

You must provide RMS with support for these specific configurations through the configuration file. However a design/implementation that internally supports additional configurations (e.g. a robot that can't carry heavy items but can carry 6 items) may be possible without much additional overhead.

As part of your final updated and extended design, you need to provide the following diagram:

- A design class diagram (DCD) of all components, complete with visibility modifiers, attributes, associations, and public (at least) methods. 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.

Implementation

Your new implementation must be entirely consistent with your new design, and will be marked on code quality, so should include all appropriate comments, visibility modifiers, and code structure.

As such, your DCD must be consistent with your final updated implementation.

Hints

As stated above, the three required tasks (report, refactor, extend) are interrelated. You are likely to find that considering the changes required to refactor and extend the system will help you identify design issues for your report, and that aspects of the refactoring are best considered at the same time as you consider your approach to extending the system. However, 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.

Building and Running Your Program

Your DCD must be consistent with your final updated implementation. 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 10 marks out of the total 100 available for this subject. These will be broken down as follows:

Criterion	Mark
Design Analysis Report for Provided Simulation	5.0 mark
Implementation of New Design	3.5 marks
Design Class Diagram	1.5 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.

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.

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

Submission

Only one member from your group should submit your project. You should submit one zip file containing the updated code package, the report, and the diagram. *Your report and diagrams should be submitted as PDFs.* You must include your group number in all of your pdf submissions, and as a comment in all source code files provided as part of your project.

Submission Checklist

1. Design Analysis Report (pdf).

2. Class Diagram reflecting your new design and implementation (pdf or png).
3. Your complete updated source code package reflecting your new design and implementation (All Java source: top-level folder called “swen30006”).

Submission Date

This project is due at **11:59 p.m. on Sun 29th of April.** 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 Peter at peter.eze@unimelb.edu.au, before the submission deadline.