SWEN30006 Project 1

Workshop 1 Team 13

Name1: Name2:

1- Refactoring and Analysis of Current Design

In this report, we will analyze the current design of the Automail system and refactor the system based on GRASP (General Responsibility Assignment Software Patterns). There are several concerns based on GRASP principles, leading to scalability and maintainability issues.

1.1 Understanding the current design

There are seven classes and two test properties in the framework. First, we should understand their responsibilities and roles in the system. I will show my understanding of some of the classes.

1.1.1 MailRoom class

The MailRoom class is the core part of the system. Some different robots are allocated in this class such as idle, active and deactivated robots. First, it will find the floor number to which the earliest arriving letter will be delivered. The mailroom then dispatches idle robots to deliver all the letters on that floor, from left to right. Once the delivery is complete, the active robot returns to the mailroom, becomes idle again, and continues the process above if there are still letters to be delivered.

1.1.2 Robot class

Based on the MailRoom class, the Robot class implements the robot’s movement and delivery process.

1.1.3 Letter class

The Letter class acts as a recorder that records the floor number, room number to be delivered, and arrival time.

1.1.4 Other class

The Main class serves as the entry point for the program and reads the test properties to run the program.

The Simulation class simulates the operational flow of the system, manages the passage of time, etc.

The Building class simulates the building structures and manages the layout, number of floors and rooms. Also, it defines some methods to determine if a robot occupies the room and implements the robot’s basic movement logic. Meanwhile, in the Building class, it defines a singleton pattern to ensure there is only one building in the system.

The BuildingGrid class displays the layout of a building through grids, each of which corresponds to a room or location. It vividly shows the floor number, room number, ladder, and mailroom.

1.2 Analyze the current design

1.2.1 MailRoom class

The MailRoom class takes on many responsibilities. It is not only responsible for robot dispatch but also for managing item information and directly controlling robots to perform tasks, which may violate High Cohesion, Low Coupling and Controller principles.

For example, someItems(), floorWithEarliestItem(), arrive(List<Letter> items), robotReturn(Robot robot), loadRobot(int floor, Robot robot), I think these methods are not the responsibilities of the MailRoom class, we should ensure it only takes its responsibility: dispatch robots according to GRASP principles through assigning these tasks to other classes, such as the Letter class and Robot class.

1.2.2 Letter class

The Letter class only stores information about letters including the floor number, room number and arrival time. However, according to the Information Expert principle, it should be responsible for processing all information related to letters, and not rely on external classes to obtain or process this information. We should put some of the methods from the MailRoom class into the Letter class, such as someItems(), floorWithEarliestItem(), and arrive(List<Letter> items). In the Letter class, we can find the earliest arriving letter, corresponding floor number, and process letters.

Moreover, according to the Polymorphism principle, the Letter class lacks scalability. In cycling mode, we only have letters to deliver, but in flooring mode, we have parcels, and in the future, we may have more different items to deliver. The best way to achieve it is to make it a parent, abstract class or interface so that other robots can extend or implement it.

1.2.3 Robot class

According to the Information Expert principle, the Robot class mainly controls the robot's movement, but not the other responsibilities he is supposed to do. For example, the Robot class should control his return to the MailRoom and load letters.

Meanwhile, according to the Polymorphism principle, the Robot class should be easily scalable. In cycling mode, we only have one type of robot, but in flooring mode, we have two different types of robots, and in the future, we may have more different kinds of robots. So, we can’t make the Robot class so specific. We should make it a parent, abstract class or interface so that other robots can extend or implement it.

3- Future Development Plan

3.1 More items to deliver

We have considered the delivery of letters and parcels, but many more items will need to be delivered in the future. In this system design, we have considered the item's weight, but many other factors should be taken into account, such as the item's size.

In addition, we can prioritize the delivery of items, not just by the arrival time of delivery. For example, if some essential documents or items need to be delivered, we should deliver these high-priority items first.

3.2 More delivering modes

We have completed the cycling and flooring modes in this AutoMail system design, but there may be more in the future. We have only two ladders in the building for the flooring mode, and we design two column robots. However, in a more complex building environment, we must plan the best robot delivery path.

Moreover, the robot in our design only considered the delivery of items on the same floor at a time; in future development, we can also design the robot to simultaneously deliver items on different floors.

As mentioned above, the items to be delivered may have a priority, so we should design a mode that delivers items according to their priority.