RMS System Designing Analysis Report

Project Group 16
Shuyuan Dang 840992
Feng Zhao 838219
Hongyi Chen 822666

Introduction

The given Java project mainly provide a simulation of AutoMail Robot Delivery system. However, the modelling of program, that is association among classes have some shortcomings, as a result, the coupling among classes is high and cohesion within each class is low. This report will identify the responsibility of each class, then analyse where the association among those classes are not fit GRASP properly and propose new design solution of each issue.

Responsibilities of each class

This section will identify the responsibilities of each class, point out what roles the class plays in the whole architectural level.

ExcessiveDeliveryException, *MailAlreadyDeliveredException*, *TubeFullException*, *Building and Clock*

These 5 classes will not be analyzed for they are error exceptions or static variable carrier which would not mightily affect the design of program.

Simulation

This class works as a driver with a main function. From GRASP perspective, even though there is no object of this class created, it implements the functions of running whole program, thus it not only play the role of controller among the whole program but also initiating the whole program. Also, it is a creator for many objects, such as *Automail* and *MailGenerator*.

Furthermore, *ReportDelivery* class which implements *IMailDelivery* interface is defined in *Simulation* as an inner class. The *ReportDelivery* takes the responsibilities of marking a *MaiiItem* as a delivered *MailItem* and updating the total score.

MailGenerator

This class takes the responsibility of creating *MailItems* and add them into an instance of an *MailPool* received from *Simulation*. In the given design, *MailGenerator* works as an information expert with high cohesion and creator to create list of *MailItems*.

Robot

This class works as a controller major in controlling the actions and states of mail delivery robots by implementing a state machine describing different actions for different triggers and guards. It receives several parameters, such as *ReportDelivery*, *IMailPool* and *IRobotBehaviour* to be initialized. Meanwhile, it is also the creator of *StorageTube* for carrying mail items.

StorageTube

This class describes the storage-tube carried by robot, and works as an information expert for storing and manipulating mail items, such as popping mail items and pushing new mail items.

Automail

This class works as creator, which creates three objects of class: *IMailPool*, *IRobotBehaviour* and *Robot*. It plays the role of an intermedia between Simulation and other components.

MailPool, *IMailPool*

MailPool implements IMailPool interface and works as an information expert for storing and sorting mail items. It is integrated with high cohesion.

SimpleRobotBehaviour, *SmartRobotBehaviour*, *IRobotBehaviour*

The former two classes describe different behaviors of robot by implementing the *IRobotBehaviour* interface. *IRobotBehaviour* is a modifiable interface and it describes the specific steps of each actions of a robot.

MailItem

This class describes a basic unit used in this program. *MailItem* is widely use in nearly all other classes. *PriorityMailItem* is a subclass of *MailItem*, with more features, such as priority level.

Analysis, Modification and reasons for modification.

This section will argue weather a certain class is suitable for its responsibilities or not and where the inappropriate design happened, then proposes our solutions.

Simulation & ReportDelivery

On one hand, *Simulation* is not only the driver with the main method but also suitable to be the main controller as it control the flow of the whole program. In a high level, it only do four tasks, which include initialization components, mail generation, starting the delivery process and displaying the final score. On the other hand, it is reasonable for assigning the role of creator to *Simulation* class because it contains the main function and need to create some object to control.

However, it is inappropriate to implement *ReportDelivery* class in *Simulation* as *ReportDelivery* is more detailed component of robot. *ReportDelivery* does not live as the same level as *Simulation*. Also *ReportDelivery* takes the responsibilities of marking a *MailItem* as a delivered *MailItem* and updating the total score. The two responsibilities should be separated in to two objects so that they can keep high cohesion.

We intend to modify two corners for this issue. The first is extracting the *ReportDelivery* from *Simulation* so that keeping the *Simulation* class only focus on high level tasks and improving cohesion of *Simulation*. The second is setting up a new interface and assign the responsibility of updating total score to it (name it *IScoreCalculator*). That can divide the responsibilities of *ReportDelivery* and make it high cohesion. Also the new interface enhance the expandability of the program for we can change the function of score calculation more flexible.

Furthermore, to enhance the expandability of our system, it is better to set parameters in a configuration file instead of fixing it with code and argument input. Considering that *Simulation* is the direct or indirect creator of most instances and we hope to read all configurations at the same time, we assign the responsibilities of reading and distributing configurations to *Simulation*. That will make our program more flexible and extendable.

MailGenerator

This class is an information expert for generating mail items with high cohesion. There is not obvious problem with this class at design level. However, it should receive parameter form configurations for enhancing flexibility

.

StorageTube

StorageTube is a high cohesion class. Nevertheless, the capability of storage tube is fixed in code, four in given program. It should have the ability of changing it capability to adapt different behaviors, for example, the common robot can only carrying four mail items but the big second-hand robot can carry up to six mail items.

Robot

Robot describes the actions of all robots in a high level. So, assigning it as the controller of the robot sub-system is suitable.

Automail

Automail is the creator of *IMailPool*, *Robot* and *IRobotBehaviour*. However, except for creating those objects, *Automail* does not have other functions. It seems more like an intermedia between Simulation and *IMailPool*, *Robot* and *IRobotBehaviour. Automail* reduces coupling of the whole design.

The design of *Automail* is good. However, we consider that *Robot* should not be created by *Automail* because it increases coupling between packages (Robot belongs to automail package but *Automail* belongs to strategies package). Based on this reason, we intend to move the creation of Robot from *Automail* to *Simulation*. Although this modification increases coupling and complexity of *Simulation*, it make sure that if we extend the strategies package, we could only focus on the robot behaviors and have no concern about how the robot be created.

In addition, *Automail* should know which behaviour it should create. So, it is necessary to modify the *Automail* in order to equip the ability of receiving a key from configuration, and creating corresponding behaviour. This modification will enhance the extensibility of the whole project. By doing this, it is easier to add and switch behaviours in this system.

MailPool, *IMailPool*

IMailPool and MailPool are with high cohesion and it highly covers all functions that a mail pool needs.

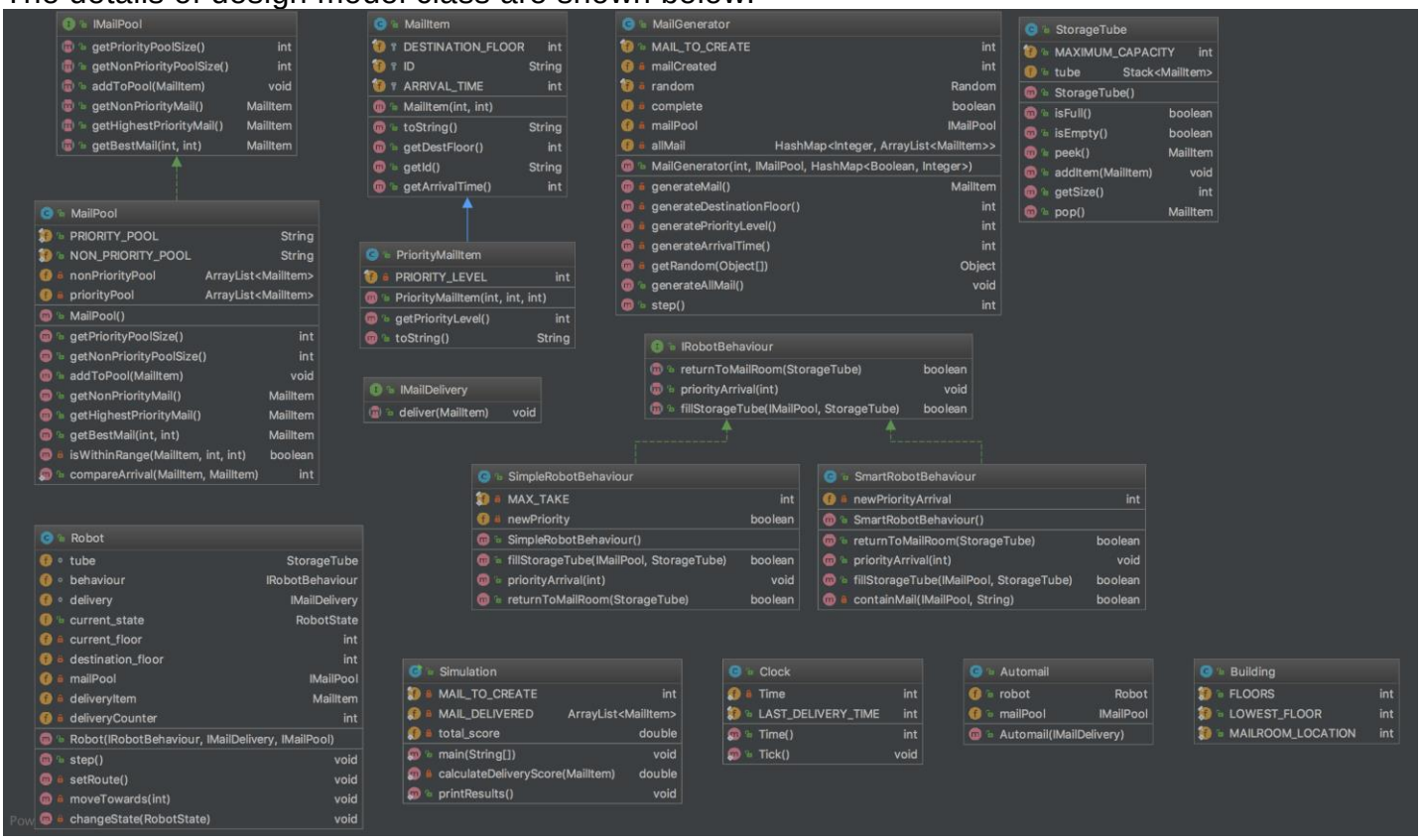
SimpleRobotBehaviour, *SmartRobotBehaviour*, *IRobotBehaviour*

This three classes defined the functionality of filling storage tube, deciding whether the robot is ready to return and how the robot reacts toward the signal of the arrival of priority mail items, with high cohesion.

Design model diagram

Because of the excessively large size of the picture, the class details and association within design model are separated.

The details of design model class are shown below.



The associations of design model are shown below.