SWEN30006 Part B Design Analysis Report

Issue among MailGenerator, IMailPool, Simulation

In the initial design, MailGenerator class is designed for creating all of the MailItem objects and add them into mailPool, which is the object of IMailPool class that passed in the MailGenerator constructor. Additionally, PriorityMailItem object will be returned into Simulation class for further operation.

By the principle of Information expert, MailGenerator is considered as a class for generating the objects, which contains necessary information for generator only. To solve this issue, MailGenerator is used purely for creating the MailItem objects, meanwhile, as a factory which can avoid complexity. Also, Simulation class handled the work that putting the MailItem objects into MailPool objects, and notify the related class when PriorityMailItem objects arrived.

Without the relationship between MailGenerator and IMailPool, the coupling is reduced at the same time, by assigning the responsibility appropriately.

Issue among Automail, Robot, IMailPool

Originally, Automail class creates Robot and IMailPool objects, and passing the IMailPool object into Robot for handling the case when robot’s state is returning or waiting. The relationship among three classes is high coupling which can be considered as a design problem.

In order to reduce coupling, attribute of MailPool object in Robot class is taken out. Since both returning and waiting states can be considered as that robot is in the mail room, when robot’s state is returning, the MailItem objects remained in the storage tube will be emptied after changing the state to waiting. Automail class will handle this case by implementation, adding an additional method in Automail class using IMailPool object to empty or fill storage tube.

Moreover, Automail is designed as one façade controller for whole mail delivery system, including the MailPool and two Robot objects. Automail class has the responsibility to create both kinds of objects and interact with the system. Additionally, step method in Automail class is added for calling from Simulation, in order to control the robot to call step method after filling the storage tube.

Issue among Robot, StorageTube, MailItem

From the origin design, Robot class contains an attribute called deliveryItem, which is the MailItem object that robot need to deliver next. Since all of the MailItem objects are stored into StorageTube object in Robot class, it is unnecessary for Robot object to store an extra MailItem object for only getting the destination floor information, which also causes high coupling.

To decrease the coupling and satisfy the principle of information expert, deliveryItem attribute is removed from Robot class. Whenever, Robot object needs to know the next destination floor, it will gain the information from StorageTube object, by an additional method getNextDestFloor, which will peek the first MailItem object in StorageTube class and return its destination floor.

Issue in Simulation

Initially, there is an attribute called MAIL\_DELIVERED in Simulation class to record all the delivered MailItem object, in order to count the sum number is satisfied to the MAIL\_TO\_CREATE and check all MailItem objects would not be delivered more than once.

Checking if delivered can be considered as information expert problem. Since the information of checking whether the MailItem object is delivered can be handled by MailItem class itself, an boolean type attribute isDelivered is created, marked as true when the object is delivered.

Also, for counting the delivered MailItem object, a private attribute is added into Simulation class with initial value of zero, and add one value after check isDelivered in deliver method in ReportDelivery class. Without the relationship between Simulation and MailItem class, low coupling is implemented.

Misplaced constants

Constants are placed into inappropriate classes, which is a basic responsibility issue in the original design, related to information expert as well. Two constants are listed below.

LAST\_DELIVERY\_TIME in the Clock class can be considered as misplaced, since the static constant LAST\_DELIVERY\_TIME is unrelated to the Clock class which is only a representation for clocking. Therefore, it is replaced into MailGenerator class, since it is the required information for creating MailItem object. After implementing the PropertyManager, which is an additional class for reading properties from file only once, LAST\_DELIVERY\_TIME is stored in this class as default property. Therefore, final implementation would be MailGenerator class can get the information of last delivery time from PropertyManager.

Meanwhile, MAX\_WEIGHT is misplaced in WeakStrongMailPool class, as it is the constrain for weak type of robot only. In this case, MAX\_WEIGHT should not be existed in any MailPool class, instead, Robot class would be more reasonable and appropriate. By distinguish the type of Robot class by enumeration RobotType, the static constant WEAK\_MAX\_WEIGHT will be used.

PropertyManager - Pure fabrication

In the extended implementation, automail.properties will be read in the Simulation class. A class called PropertyManager is designed for handling all of the properties uniformly. PropertyManager class stores all the default properties in file, and overwrite by the changed file if possible, which satisfy the principle of pure fabrication, like for the pure purpose.