Seminar 2

Object-Oriented Design, IV1350

Ossian Dillner – ossiand@kth.se

2017-05-02

Contents

1 Introduction

In preparation for this seminar the student was tasked with designing a program capable of handling the scenario given in the previous seminar. The design most follow the standard guidelines for object oriented design, such as making sure the program has high cohesion, low coupling and good encapsulation. The student most also make sure to follow the MVC (Model-View-Controller) pattern when designing the program.

I choose to perform this seminar task by myself.

2 Method

Before I started designing the previously mentioned program I made sure to carefully examine the domain and system sequence diagram from the previous seminar in order to the best of my ability follow the naming and association standards established in the analysis of the given scenario.

I then laid the foundation for the class diagram by first creating the necessary packages, carefully following the MVC pattern while doing so. I then started filling these packages with classes, these classes were heavily inspired by the aforementioned analysis of the scenario where applicable.

Finally I added the necessary associations while constantly redesigning the program to ensure optimal cohesion, coupling and encapsulation.

After laying the foundation of the design by creating a comprehensive class diagram I continued the design process by creating several communication diagrams detailing the various operations contained in the program. The already created class diagram made this process very easy by simply adding the already existing classes and methods into the corresponding diagrams, making sure to keep the program flow intact.

3 Result

Since the package/class diagram is quite hard to read when inserted directly into the report, as seen in fig 3.1, here's a link to a page containing an image of the complete diagram.

The aforementioned diagram is quite self explanatory, it contains all the relevant classes and the methods contained in those classes as well as how the different classes depend on each other and which packages they are divided into.

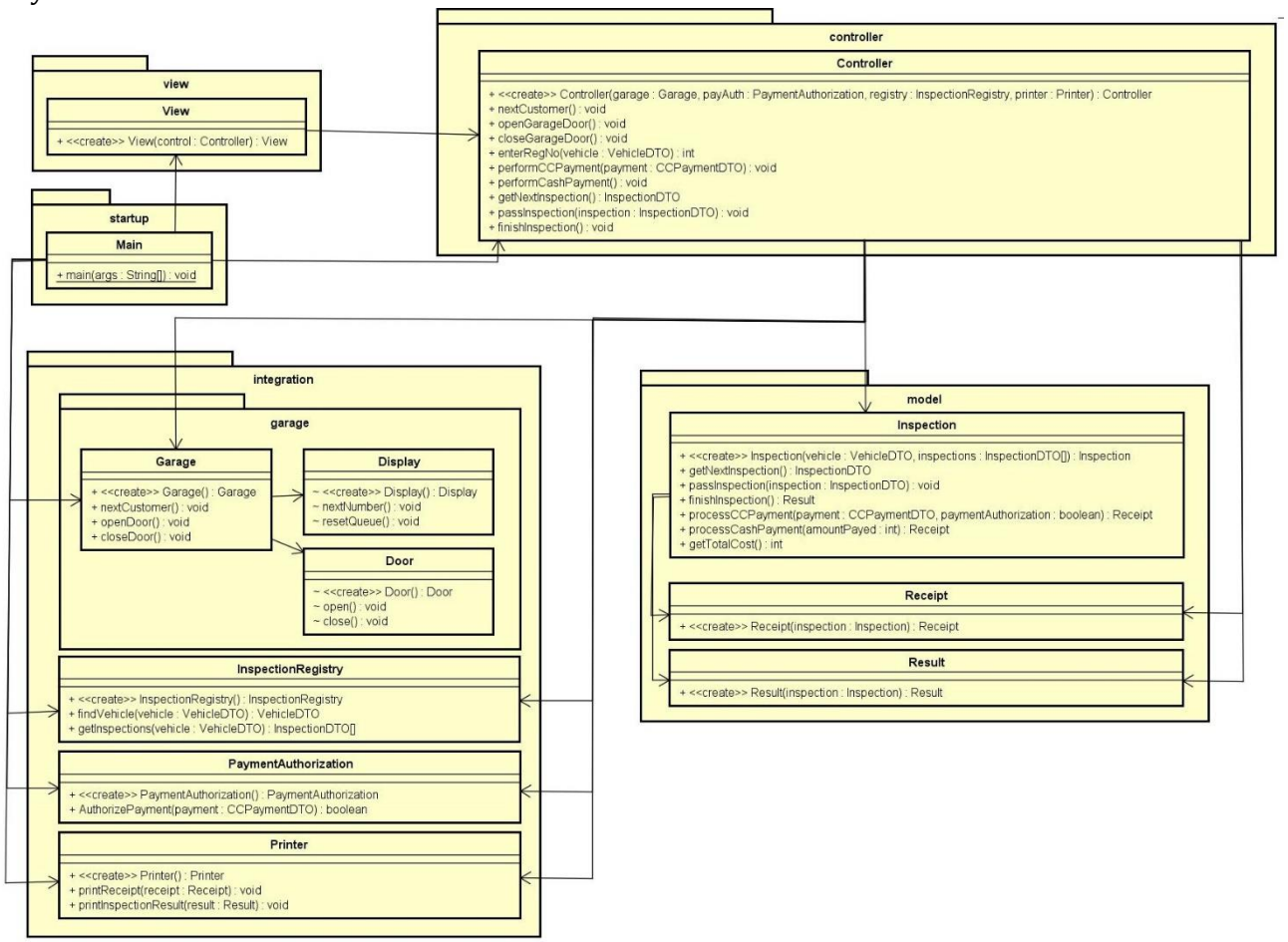

fig 3.1

The start sequence in the main function, as illustrated below in fig 3.2, starts by initializing the different external systems represented in the integration package, it then initializes the controller, passing along the aforementioned external systems as parameters and finally it creates the view using the previously created controller object.

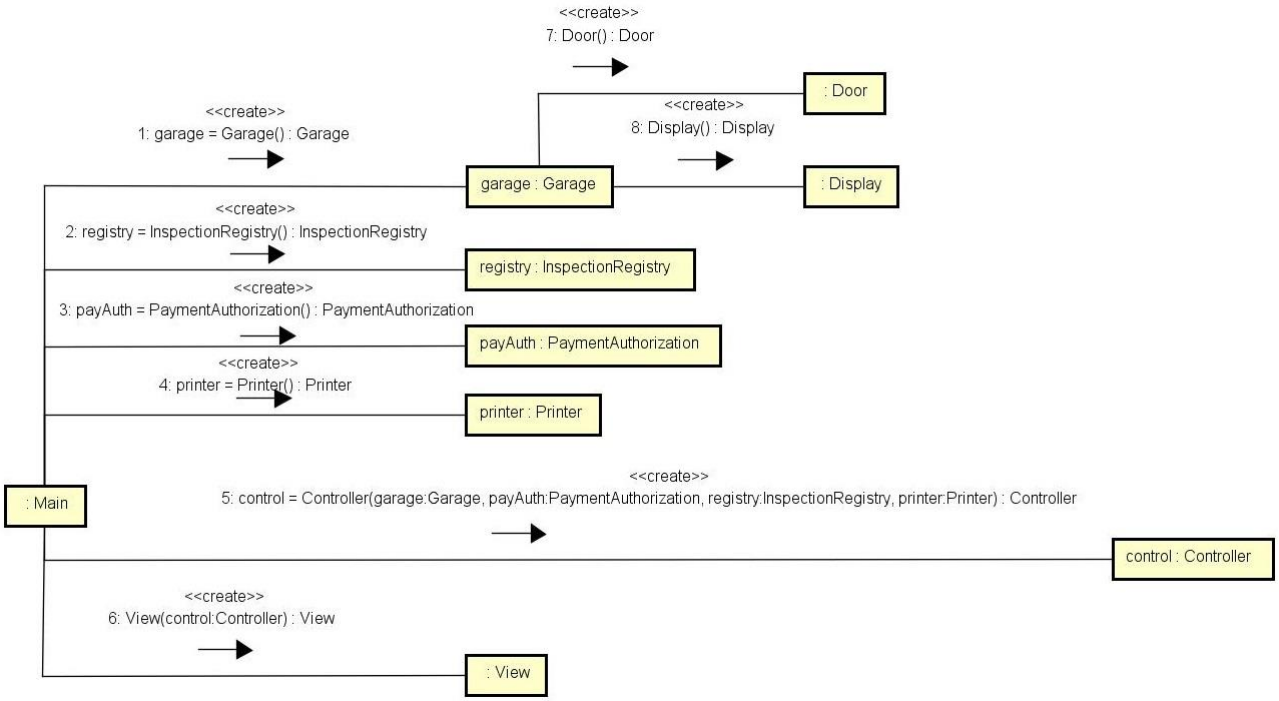

fig 3.2

After running the startup sequence the operations relating the given scenario can start. The first step is the instructor starting a new inspection by instructing the system to do so, which makes the system open the garage door and increment the displayed number on the queue display, as illustrated below in fig 3.3.

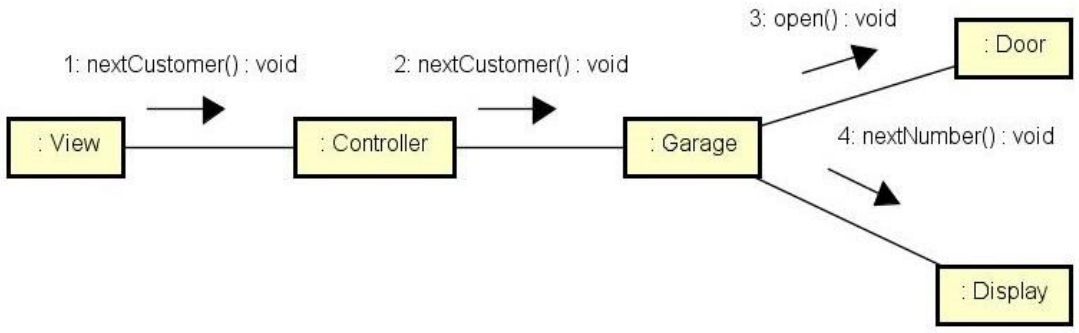

fig 3.3

The next step in the scenario is registering the customers vehicle in the system, this operation is illustrated in fig 3.4 below. The inspector enters the registration number into the system, which then contacts the registry containing vehicles and corresponding inspections, requesting it to return the searched for vehicle. If such a vehicle is found the system then requests all inspections that needs to be performed on the vehicle and feeds this information to the Inspection class which handles most of the business logic during the inspection process. Finally the controller calls a method contained the Inspection class that calculates the total cost of the whole inspection process and returns it to the view, allowing for the scenario to progress further.

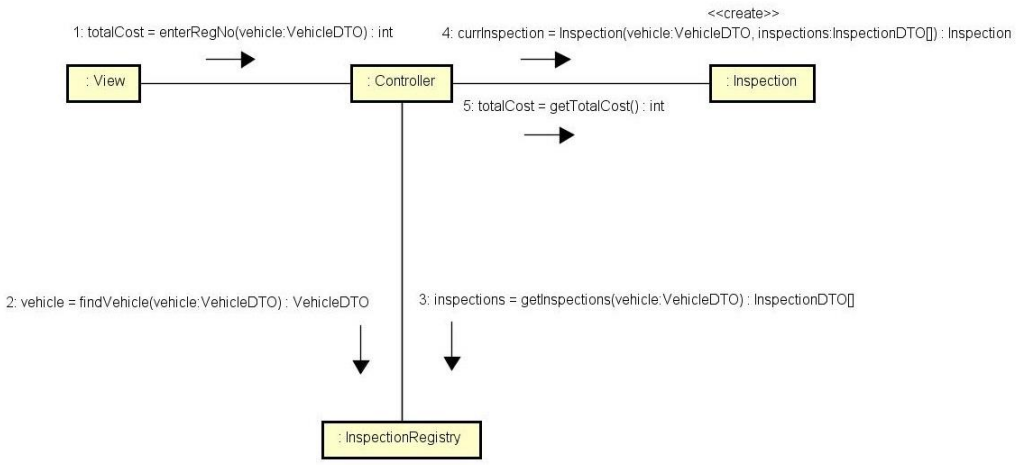

fig 3.4

After the vehicle has been registered in the system the next step in the scenario is the payment for the inspection, as illustrated below in fig 3.5. The inspector inputs the payment information into the view, which passes it along to the controller which then acquires authorization from the external authorization system before passing the gathered information along to the Inspection class where the business logic is handled. The Inspection class then returns a receipt for the controller to pass along to the printer to print.

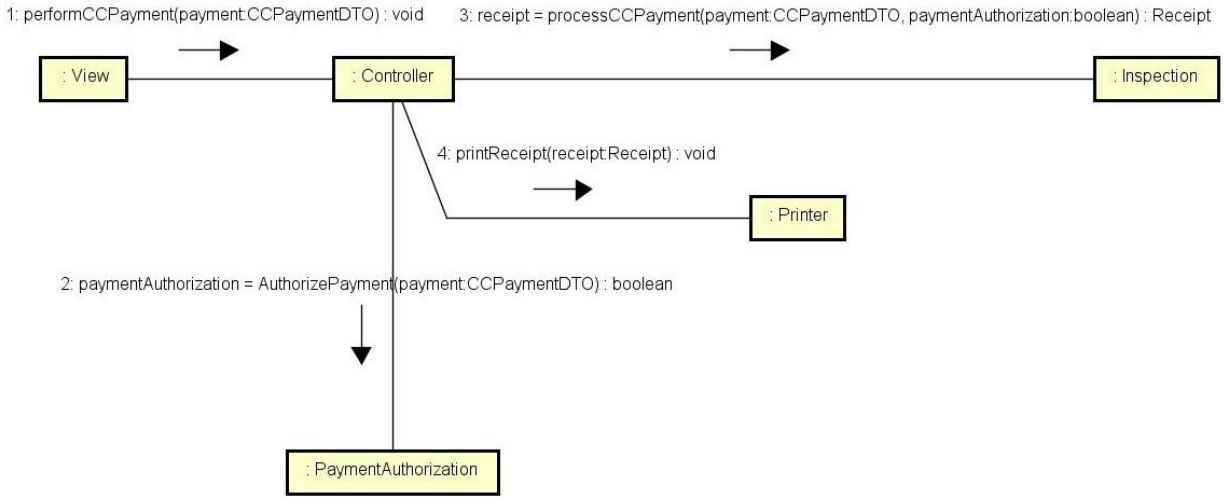

fig 3.5

The final part of the scenario is the inspection process, illustrated below in fig 3.6. The inspector simply asks the system for an inspection to perform, performs the aforemention inspection and tells the system the result. This is repeated until there are no more inspections left to perform at which point the inspector tells the system that the inspection is finished, which causes the system to tell the printer to print the results of the inspection process.

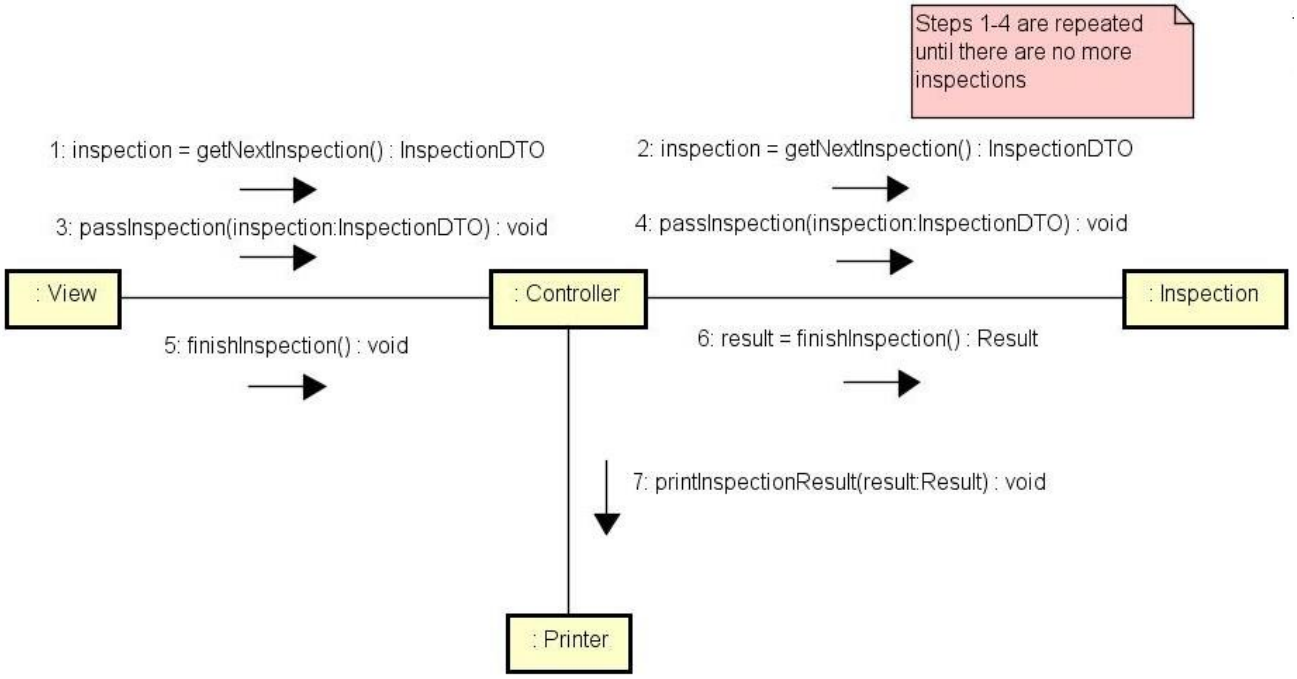

fig 3.6

4 Discussion

I find that my design fulfills most of the criteria. At first when I started this course I was very doubtful when it came to these diagrams but after designing and during the later seminars implementing these designs I've found them extremely helpful and that's for a program with quite a limited scope, I can't even imagine how useful the design process will be in the future when I will be required to create actual programs for bigger systems.