**Seminar 1**

Object-Oriented Design, IV1350

Ossian Fjordefalk

[Ossian.f@gmail.com](mailto:Ossian.f@gmail.com)

2/5-2017

**Contents**

1 Introduction 3

2 Method 4

3 Result 5

4 Discussion 7

1. **Introduction**

The task of seminar 3 was to create a program from the design models made in previous seminars. We then created tests for the program using Junit. I did all the work on my own and then discussed my solutions with class mate Anders Pettersson.

1. **Method**

I started of by creating all the classes and packages included in the class diagram created in seminar 2. I then looked at communication diagrams from seminar 2 and implemented the necessary methods into the classes. Some translations from model to actual program was very straightforward, I just created the methods mentioned in the communication diagrams and used the same parameters. But some things had to be changed to work in the real program.

1. **Result**

Github link: <https://github.com/KrokusMorning/Seminarie3>

The program simulates a vehicle inspection. Data is inputted through the view class which then calls methods in the controller class.

The method newInspection is located in the controller and calls a method in Garage which in turn opens the garage door and increases the que number on the display.

The method closeDoor is located in the controller and calls a method in Garage which in turn closes the garage door.

The method costForInspection is located in the controller. It takes vehicle as a parameter and calls a method in database manager which creates and returns an instance of InspectionChecklist which is the used to create an inspection instance. Inspection then calculates the cost based on the information about which inspections to be done in the inspection checklist. costForInspection then returns the inspection checklist. The cost for the inspection can then accessed through a getter. In the model the int value cost was returned from the method. In the program I made the function update information in an object instead as I found it more convenient.

In the model this method was called enterRegistartionNumber as a primitive value containing a registration number was used in the method call. In the program I choose to create an instance of vehicle and use that in the method call instead.

The payWithCard method is located in the controller. It takes an instance of CreditCard and two primitive values as parameters and calls the cardPayment method in Payment. The payment authorization class is then called to verify the payment and returns a Boolean value The method in payment then creates an instance of CardReceipt resembling a receipt which is then passed on in a method call to the printCardReceipt method in the Printer class. The printer writes out all the necessary information about the payment.

In the model I had the payment class create the credit card, but that included passing long list of parameter through multiple method calls. I solved it by creating the credit card from the view and then pass the object instead.

The whatInspectRequest method located in the controller updates the checklist with the result for a specified inspection. It takes the result vehicle and checklist to update as parameters. The data base manager checks if it’s the right vehicle and updates the first inspection that is not passed and returns. One method call updates one inspection result.

The inspectionComplete method in controller calls a method in database manager which creates an instance of inspectionResult which copys all the checklist results and generates a final result. The database manager then calls the printer which prints out the result.

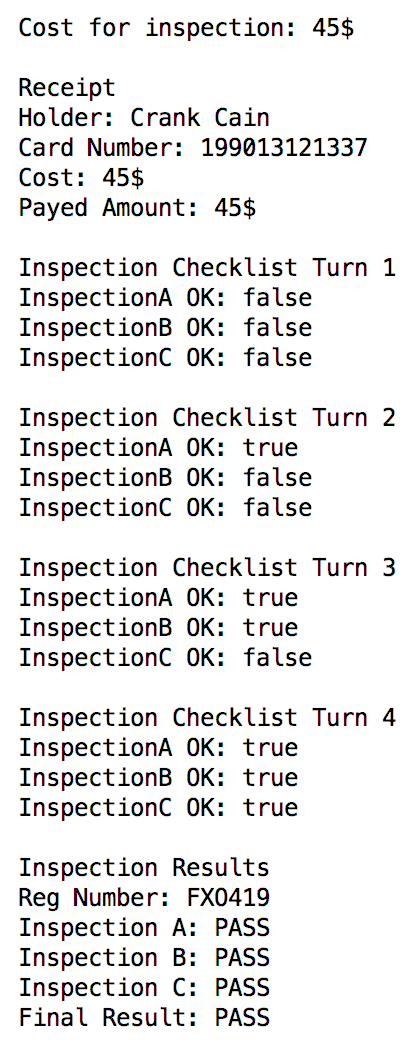


Figure Sample run

1. **Discussion**