**SOFT 252 Java Coursework**

**Uni Systems - A car loan management software Package**

**A group project by**

**Craig Banyard, Daniel Scott and Najim Mazidi**

# Introduction

We were tasked with creating a Java software package that follows the MVC (Model View Controller) Pattern, along with at least two other commonly used design patterns stated in the brief. The brief specified that we should be creating a car loaning system for staff members of a business. Some cars should be allocated to staff members on a long term basis while others can be kept in a pool for day loan use only. We had to create a desktop prototype application to maintain the details of cars, staff members and the loan information, which could all be serialised to disk and then de-serialised on load.

# Design and Implementation of the Application

On the last page of the report you can see an initial UML design which we came up with to outline our classes and their structure and how they relate to one another, with a few methods to give us direction in the early stages of our design. From these early stages of design we decided that we would have a strategy pattern implemented for our loans, to determine whether the loan was a long loan or a day loan, which would then make the objects behave differently depending on what they were. We also decided that we would implement our state pattern on the car objects using a series of Enumerator classes to set the different flags on the car. The application behaves differently depending on these flags which we set upon the car object. Initially we also decided on having a Factory class which created all of our cars and stored them to a garage, however during implementation we believed it would make more sense to have the administrator object handle all of the object creation and storage.

Before the implementation of the system, we quickly realised that the data would need to be observed, so that any changes to the data could be saved to the disk as the changes were made. Our implementation then deviated from our design a little more in that we created an observer system to watch our data as it changes, in order to save the changes. This also lead to the question of how we were going to save our data, which is where we decided on making some singleton classes which host lists of objects. Four things need saving and therefore we created four singleton classes for the Administrators, Cars, Loans and Staff members of the system. Staff were there to loan the cars, Administrators were there to handle the assignment of the loans and the maintenance of the data, Cars just hold properties of specific cars such as their service record and lastly Loans just hold duration data, what car is being loaned and who loaned it.

The administrator object became one of the key elements of the entire system, as we pass around the current administrator object (who is the person logged in to the system) to each form and use that to carry out any tasks which need doing to the data layer of the application.

# Design Principles

## MVC

By the end of our implementation, the design principles that we believe have been implemented well are the following: MVC, as we have our data kept away from the GUI and most of the business layer in singleton classes which hold lists of data which are monitored by their observers. Architecturally we decided that the Administrator should be the only public class which has direct access to the list of objects stored in the application. This is our controller. Once the majority of the data model was in place we started to piece together a GUI taking HCI principles into account while also keeping as closely to the brief as possible.

## Strategy / State pattern

In our system, we implemented the strategy pattern on our loans as the two loaning objects which inherit from the super Loan class. DayLoan and LongLoan act differently depending on which Loan object they take on. We believe that our State pattern has been implemented on various areas of the car as many flags get set through the use of enums to say whether the car is out on loan or awaiting service among other things. These different flags change the way in which we display information to the end user.

## Observer

The observer design principle was implemented to save any changes we made to the data during runtime. This was done in a way that meant every time we create a new object, albeit a Loan, Staff, Administrator or Car, a matching observer is registered to the object, and on the “setter” methods of the objects we tell the observers to call their update method which saves everything to disk.

## Command

To incorporate more of the design patterns listed, we also decided as a last step of our implementation, to implement the command pattern. This allows the user to have access to “undo” and “redo” functionality which in a data entry/maintenance system could be vital if a mistake were to be made.

# Summary

Overall I think we feel as a team that we have incorporated everything which was required of us in the brief. We have also gone beyond the spec in that we have added the capability for the users to be able to log in to the system adding some level of authentication. If we were to do this project again, we may have spent a little more time doing more research into where the design patterns were suitable to improve our current implementations.

