Object Oriented Final Project

Challenges and Lessons

Amy Wilson

Lewis University

10/22/2020

Table of Contents

[The Project 2](#_Toc53866958)

[Challenges 2](#_Toc53866959)

[Lessons 4](#_Toc53866960)

# The Project

To satisfy the requirement of utilizing four of the design patterns learned in this class, I decided to implement a baseball line-up manager application. This application provides the user with the capability of setting a line-up using bullpen, line-up, and field views. The application loads players into the system and the user can select a pitcher from the bullpen, add players to the line-up from the line-up manager, and see those players on the field portion of the application.

The patterns used to create this system include the observer, factory, decorator, and model-view-controller (MVC) patterns. These can be found in the appropriately named packages Observer, Factories, PitchDecorator, Controller, Views, and Models. The Observer is used to gather data from my player and pitcher views and report changes to my controller so it can update the models and repopulate the data. It is also used in the main view to update the view visibility in the frame. I have pitch and player factories that supply the data used in the views. I use the pitch decorator to add applicable pitches to each pitcher’s repertoire of pitches. For the MVC pattern, I have views, models, and a controller. There is also a test folder that contains tests for the factory and decorator patterns.

Architecting a system that is maintainable, flexible, reusable, and extendable is always a challenging exercise. Design patterns allow us to pursue these goals by using researched and recommended strategies. So, the usage of design patterns reduced the challenges in designing this system. However, I still ran into a few issues along the way.

## Challenges

While designing my final project, the patterns that I used were selected by running into problems that the patterns solved. I used factories to bridge my data provider and my view controller. The decorator pattern allowed me to add pitches to my pitcher’s repertoires. MVC allowed me to create separation between the view and the business logic. Then the observer pattern enabled transferring data from various views. At the end, this all seemed obvious to me, but I did struggle initially.

One of my issues involved trying to use the decorator pattern where it did not fit. I started with a player decorator and tried to decorate my players with different property values and positions. After I had spent some time on that, I realized that while this could work, all the players had the same properties that just had different values. While a decorator can modify property values, its purpose is to add functionality. It was not suited for my purpose of modifying property values. Property modification can be done more simply so, I decided on using a factory with a parameterized constructor.

Then, I was trying to use the factory pattern to obtain pitches. While this worked, once it was implemented, I realized how much better the decorator pattern fit this function. Each pitcher had a different repertoire of pitches. So, it made sense to have a repertoire of pitches that could be added on to by the different pitch decorators. So, I ended up redoing my pitch factory after I had already put it in.

Lastly, I struggled implementing my decorator with a collection as opposed to the string example in the book. In the first assignment, I had attempted to add to a collection using decorators. After some number of failures during the week I reverted to the simple string append. This time, I really needed to add to a collection since I was going to use the objects in a table view. After doing some research, I realized I needed to return an empty collection in the instantiation object. After that, my pitch decorator unit tests began to pass, and I had successfully decorated my collection.

## Lessons

One of my biggest take-aways of this project is that forethought will save me time later. I initially tried using two patterns that were not optimally suited to address the problem I was trying to solve. So, I spent time implementing a pattern or partially implementing a pattern that I just had to redo later after I had thought about it more. I could have spent more time on the forefront designing my system and saved myself time on the implementation side.

Another lesson that was important for me is that I need the ability to apply these patterns in a wide variety of ways. The application that is outlined in the book is just one scenario, but programming large systems can require applying a pattern in different ways. The book has outlined decorating a string or int, but real life may require decorating a collection or a hash map. In fact, having worked on numerous large systems, manipulating more complexed data types of a frequent occurrence. So, putting in the time to learn additional application tactics will more than likely prove useful.

This class has reinforced the notion that architecture and well thought out design are integral in creating systems that survive and thrive. This architecture of this system is maintainable, flexible, reusable, and extendable due to the use of patterns. MVC allows for the separation of the user interface and business logic in an ever-changing world of GUI’s. Adding and updating the view became very simple because of this pattern. The observer pattern helps to increase separation and allows for more flexibility. Extending the scope of updating the line-up was simple because of this pattern in conjunction with MVC. The decorator allows for the ease of adding additional pitches. Adding a knuckleball or a spitball would just require more decorators and handling them. Lastly, the factories increase flexibility by separating object creation and requests for the objects.