Final Project Report

**Project Overview**

* Brief description: Car Construction will take in specs given by a customer and the program will build them their custom car. We aren’t talking about building a supercharged Toyota Corolla or a turbocharged V8 Chevy Tahoe SUV. This will build a **custom** car from the specs.

**What is the purpose?**

* Whenever someone buys a car, they always ask for features. With those features they desire, they wind up getting features that they don’t really need. This program solves that problem. A customer can enter what specs they want (style of car, powerTrain (Gas, Electric, Hybrid), Transmission, exterior material, size of wheels, size of brakes, interior material, seats (amount, material))
* We are assuming that nothing is impossible. For example, a user wants a subcompact hybrid V12 manual transmission 24 inch wheels, 20 inch brakes, 8 leather seats, real Rose Wood Trim interior. These are possible. There are no default values in this program.

For this project, I used the Builder, and Singleton creational patterns, Observer and the Iterator behavioral patterns. Unfortunately, no structural patterns were used. Originally, I planned to use the Decorator structural pattern for dealing with the details of the car, but I have realized that I can just use the builder for dealing with the details.

The Builder pattern uses the director for taking in the customer specs, which will then pass them onto the Builder. The Builder will then be constructing the car throughout the process. The Singleton pattern is used to create one instance of the Database (SQLite3). If there is a scenario where a particular item is low on stock, the program calls the Observer pattern to monitor and order more stock for that particular item. The Iterator pattern may not be explicitly used, but because I am using a map<string, int> and a vector<vector<string>>, the iterator will sequentially go through the elements in the particular list or map.

I believe that Builder is more suited for this task over Factory, because keep in mind, building a car requires several complex steps. Plus, Builder is more flexible than the Factory pattern.

At the stage where I was near completion, I realized that maybe I could have used different patterns. For example, I could have used the prototype pattern, which would ultimately create the same model from the style of car, powertrain, transmission, exterior material, interior material, size of wheels, brakes, interior cabin material, seat material and number of seats), except with just different values for each spec. Trade offs: I could use either Composite, Proxy, or Decorator for this project. Decorator allows me to add responsibilities to other objects without creating too many subclasses.