Blaine Simcox II

CIT 275

Jeff Straw

FacadeWinFormTest

**FacadeWinformTest Overview**

In this application, I built a simple database program that uses the AdventureWorksLT database in the form given by Eugene Seuter in CIT 248. Nothing has been altered in the database for ease of coding, and since I am still a novice to using ADO, I figured the AdventureWorksLT database to give me quite a bit of hassle which it did. We’ll get to that later.

This application uses a façade class to call all of the functions in this application. In the case of binding the form controls to the façade, this step is wholly unnecessary as the same result can be achieved with less code. However, for the sake of upholding the façade hierarchy structure, the control methods activated by the buttons are contained in the façade, not the form. The form only has code for extracting data from the façade layer and printing it to the screen. Even the IteratorCount variable that tells the user how many records are present in the current query, is wholly processed on other layers of the application.

Though this is only a small application detailing the use of a façade, this style is no different than that taught in Westwood’s C++ OOD classes. All classes operate from a central controller class, this hierarchy can really be seen in my Yosemite Final Project, in the application part of my Portfolio. All classes have satellite or intermediary classes where needed to operate their individual functions, but no class operations happen without calls by the controller class. This FaçadeWinformTest demonstrates the same type of operation as Yosemite, albeit on a smaller scale.

Aside from the actual mission of this project (achieving façade application), I learned a great deal about database implementation using C# and SQL together. The biggest issue I ran into while working on this project was passing a null string value from the DB into the app. After hours of trying to find a solution on the internet, I finally came across an answer so easy I don’t know I why it wasn’t my first thought on the matter. The .ToString method will only return empty or full string values, never null, thus solving the problem. Before directly parsing the DB data with the .ToString, I had tried all manners of exception handling to convert the null strings, but the app would crash before the try block could even get the data.

This particular hierarchy seems like it would work very well in video game apps as well as business level, database driven apps used for common office needs. The reason I say this is because of the segregation of the layers. With the exception of the satellite classes (at least that’s what’s I call them) Order and Customers, all data is being fed into and out of the façade layer without any other classes knowing the other’s business. Even the information displayed on the form is simply the result of the DBClass passing List<>() Values into the Façade, simply printing the information to the screen from the Façade’s list<>(). The form actually does no processing other than printing data from the façade’s lists which are in turn filled on the DBClass level, completely anonymous to all other classes not involved in DBClass storage. No class operates without first getting the façade to do it for them.