Robert C. Oliver, Jr.

03/12/2021

Dr. Nowak

Software Engineering

Write Up for Final Project

This project was a menu ordering system for a fake dessert shop, called cozyDesserts. This menu ordering application had four design patterns that will be explained later in this write-up.

The project is made up of seven packages, including the default package. They are as follows: Cremebrulee, Cremebrulee.CremebruleeDecorator, Drink, Drink.Decorator, IceCream, IceCream.IceCreamDecorator, OODFinal, and default package. In IceCream and Cremebrulee lie the objects themselves, name IceCream.java and Cremebrulee.java. The drink object lies in Drink> Drink.java. The Drink object implements the Template Design Pattern, as the cold drinks get ice put into them, and the hot drinks have various add-ons other than ice. Patissier.java and Server.java are the workers of the shop, and they lie in OODFinal. The Patissier has a method, getOrder, that takes in the type of object being sent from the Factory Pattern, and depending on what it gets sent (Drink, Ice Cream, or Cremebrulee) it creates a new FactoryPattern object, that sends back the type of dessert they picked as a new object. That object then gets decorated, and the cost of each dessert gets returned as well.

In the project rests four design patterns. First one would be the Command Design Pattern, which lies in the OODFinal package. Within the same package is the Pâtissier.java (pastry chef) and the Server.java file, who act as the receiver and invoker of the pattern, respectively. Also, within the OODFinal package is the Factory Design Pattern, which lives in three files: CremebruleeFactoryPattern.java, IceCreamFactoryPattern.java, and DrinkFactoryPattern.java. The Template Design Pattern lies within the Drink package, in the ColdDrink.java and HotDrink.java files. The last Design Pattern in the program currently is the Decorator Design Pattern. This pattern is in the following packages: Cremebrulee.CremebruleeDecorator and IceCream.IceCreamDecorator. The respective associated files are as follows: FruitDecorator.java, OtherDecorator.java, ConfectioneryDecorator.java, FruitDecorator.java (in the IceCream.IceCreamDecorator package), and SnacksAndCandyDecorator.java.

The application starts with a menu that greets you and asks the user if they want to order ice cream, a crème brulee, or a drink to go with it. If ice cream or crème brulee is picked, you are taken to a secondary menu, offering you flavors for each desert (as of now, you can only pick one flavor). After a flavor is selected, the Decorator Pattern comes into play. For each dessert, you are offered certain types of toppings- for ice cream, there are three decorators: ConfectioneryDecorator, SnacksAndCandyDecorator, and FruitDecorator; for crème brulee, you there are two: FruitDecorator and OthersDecorator. Confectioneries have caramel, fudge, and so forth. Fruit has mixed berries and bananas, Snacks and Candy consist of Cookie Dough, Oreo’s, and so forth. The actual decorator method lies in the Patissier.java file. After they are done topping their dessert, an array is made for each decorator, and an “order” is placed (IceCreamOrder, DrinkOrder, or CremebruleeOrder) and set to the server to deliver via the Command Pattern. The customer is then greeted with the total for the item they just customized, along with the same menu from the opening application. This continues until the user presses 0, which tells the server that the customer is leaving the shop now. The customer is given an exit message.

I did have to make changes from what I originally thought I would use. I stuck with the Decorator and Factory Design Patterns, as it was a no-brainer in terms of using the Decorator Pattern for choosing toppings on the desserts. I wanted to use the Front Control Design Pattern, and the Model View Controller Design Pattern, but I did not get to implementing a front end yet. In the future, the MVC Design Pattern would be another pattern implemented into the overall project. These changes came late on, but I was able to adjust accordingly.

The biggest difficulty faced was the time frame. This was hard to keep up with. It was important to do development each week, and I feel like I fell off the mark with it. I overcame this difficulty by setting a reminder on my calendar to develop the program for at least an hour per day. If I could not do this, I would make sure to read up more on the Design Patterns I was using, to get familiar a bit more. In the end this paid off, as I did not have to cram too much in the end, although I should not have been cramming anything at all. A better design plan should have been from the start, rather than making changes as I went. This led to many “false leads” in terms of functionalities that I did not end up implementing. I eventually overcame this by sticking to what I knew in terms of added features. This kept me slowly moving forward with development, and it made my timeline a lot less sporadic.

Another difficulty I had was sending the Drink and Cremebrulee object to their respective Factory Patterns. I kept getting a NullPointerException in the Patissier getOrder method. The Ice Cream object was being sent just fine, and I created both objects the same way, so I knew there was something wrong with my syntax. I went debugging for a couple of hours, adding breakpoints in the Patissier, Server, CremebruleeOrder, IceCreamOrder, DrinkOrder, CremebruleeFactoryPattern, DrinkFactoryPattern, and IceCreamFactoryPattern. I first followed the Ice Cream object through the many files of my code, watching it bounce back and forth, here and there. I watched it through a few times, and at each breakpoint, I looked at my variables and the state of each variable. I noticed the pâtissier object itself was holding null Drink, Ice Cream, and Cremebrulee objects, but the Ice Cream object was working fine, so I know the problem did not lie in what the pâtissier object was holding. The exception kept popping up at the statement where a new object is created and sent to their FactoryPattern, so I thought the issue might lie in the factory pattern files. I looked through them and they were all correct, so I decided to look at the actual Cremebrulee object itself. Each object has a getDecsription() method that has a String of the name of the product. I noticed that I made a custom description for each cremebrulee and drink object, and I made the name of the product description in the customer menu. Basically, the object was looking for (as an example) “Pepsi” as the type of drink. My description of Pepsi was not “Pepsi,” instead it was “A classic soda.” This results in an object not being created, and an error returning a null object. The Factory Pattern method could not locate the “Pepsi” object, as the Pepsi object was technically the “A classic soda” object. Once I resolved this issue, the rest of development was smoother, and I used my newfound debugging knowledge for the rest of the project.

Overall, I enjoyed the project, and I am looking forward to doing this as a career in the “real world.” I also enjoyed the love and hate for development throughout these past eight weeks. There were some days where everything went right, and some days where nothing went right. I realized that this is all part of the development process, and if you are persistent enough, you end up with a project that is a unique piece of art. This is what I truly enjoy about Software Engineering!