Jackson Spencer

The Abstract Factory Pattern

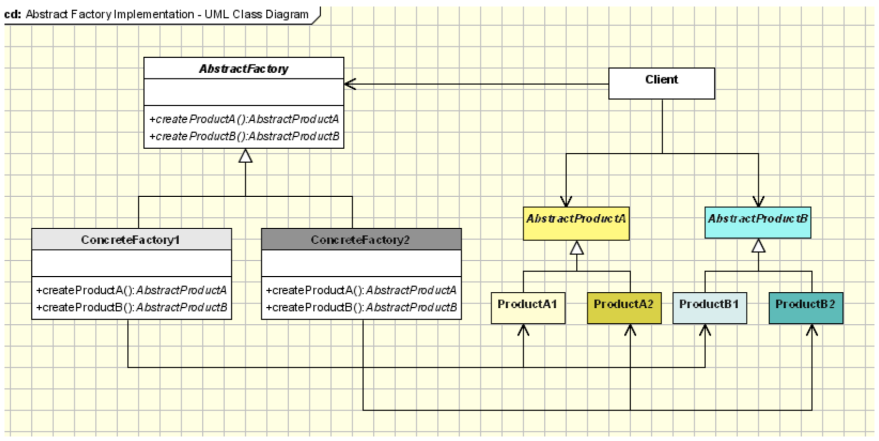
September 28, 2016

ECCS 2411: Design Patterns

Introduction

Our assignment was to do the Abstract Factory Pattern. The purpose of this pattern is to "provide an interface for creating families of related or dependent objects without specifying their concrete classes." In the application I built, a food dish is used to represent the abstract product. I used pasta and sandwiches as two different concrete products and their creation as concrete factories.

UML Diagram


cd: Abstract Factory Implementation - UML Class Diagram
AbstractFactory
+createProductA():AbstractProductA
+createProductB():AbstractProductB
Client
ConcreteFactory1
+createProductA(): AbstractProductA
+createProductB(): AbstractProductB
ConcreteFactory2
+createProductA(): AbstractProductA
+createProductB(): AbstractProductB
AbstractProductA
AbstractProductB
ProductA1
ProductA2
ProductB1
ProductB2

This is the UML diagram for the abstract factory pattern. For the Abstract Factory, I created an interface called CreateDish. I created two Concrete Factory classes: Create Pasta and Create Sandwich. There was one Abstract Product that I called Dish. The code for each product of the Abstract Product can be seen in the next section.

Classes

The interface CreateDish has one purpose and one method: to create a dish to return a dish.

```
interface CreateDish     // Abstract Factory
    {
        Dish createDish();
    }
```

First, however, in order to create the CreateDish interface, the Dish abstract product needed to exist. It declares variables that each concrete product requires. Both sandwiches and pasta need to have substance, toppings, side orders and a method to have their orders taken.

```
public interface Dish  // Abstract Product
    {
        string getSubstance();
        string getToppings();
        string getSide();
        string getOrder();
    }
```

Next are the concrete products. Both the Pasta and Sandwich classes inherit from the Dish interface. The two classes are very similar in the way they are built. Each have toppings, substance, side, and getOrder, but they also have a few elements of their own. Each piece of information is passed through the constructor. There are also methods to return each value that is used. The getOrder method at the end creates the string to be used in the order textbox on the form itself, which can be seen in the screenshot at the end of the code descriptions.

Pasta class

```
public class Pasta : Dish    // Concrete Product
    {
        private string pasta;
        private string sauce;
        private string substance;
        private string toppings;
        private string side;

        public Pasta(string pasta, string sauce, string meat, string extra, string side)
        {
            this.pasta = pasta;
            this.sauce = sauce;
            this.substance = meat;
            this.toppings = extra;
            this.side = side;
        }

        public string getPasta()
        {
            return pasta;
        }

        public string getSauce()
        {
            return sauce;
        }

        public string getSubstance()    //also known as the meat
        {
            return substance;
        }

        public string getToppings()
        {
            return toppings;
        }

        public string getSide()
        {
            return side;
        }

        public string getOrder()
        {
            return "-So your order is a " + getPasta() + " with " + getSauce() + " and "
+ getSubstance() + " with " + getToppings() + " complimented with a side of " + getSide()
+ ".";
        }
    }
```

Sandwich class

```
public class Sandwich : Dish    // Concrete Product
    {
        private string substance;
        private string bread;
        private string cheese;
        private string toppings;
        private string side;

        public Sandwich(string substance, string bread, string cheese, string toppings,
string side)
        {
            this.substance = substance;
            this.bread = bread;
            this.cheese = cheese;
            this.toppings = toppings;
            this.side = side;
        }
        public string getSubstance()
        {
            return substance;
        }

        public string getBread()
        {
            return bread;
        }

        public string getCheese()
        {
            return cheese;
        }

        public string getToppings()
        {
            return toppings;
        }

        public string getSide()
        {
            return side;
        }

        public string getOrder()
        {
            return "-So your order is a " + getSubstance() + " sandwich on fresh " +
getBread() + " with " + getCheese() + " cheese, " + getToppings() + " complimented with a
side of " + getSide() + ".";
        }
    }
```

The CreatePasta and CreateSandwich classes represent concrete factory classes. They inherit from the CreateDish interface. Once again, all of the information is passed through the constructor that is required to create a Pasta or Sandwich object. The createDish method returns a new object of the Pasta or Sandwich class.

CreatePasta

```
class CreatePasta : CreateDish  // Concrete Factory
    {
        private string pasta;
        private string sauce;
        private string meat;
        private string extra;
        private string side;

        public CreatePasta(string pasta, string sauce, string meat, string extra, string
side)
        {
            this.pasta = pasta;
            this.sauce = sauce;
            this.meat = meat;
            this.extra = extra;
            this.side = side;
        }

        public Dish createDish()
        {
            return new Pasta(pasta, sauce, meat, extra, side);
        }
    }
```

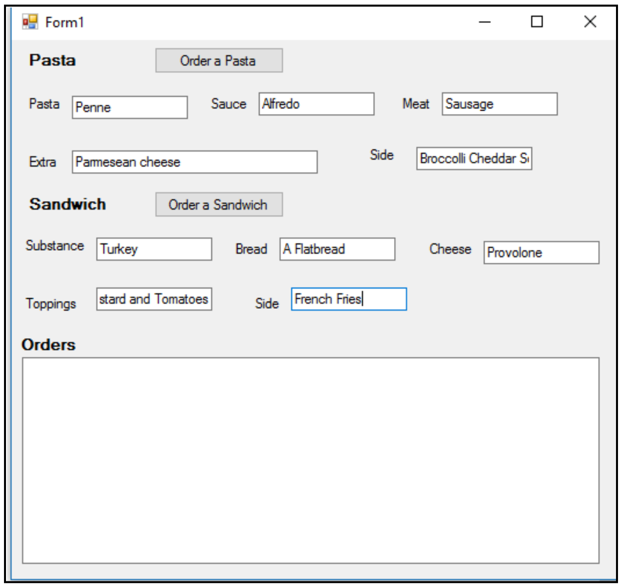
CreateSandwich

```
class CreateSandwich : CreateDish   // Concrete Factory
    {
        private string substance;
        private string bread;
        private string cheese;
        private string toppings;
        private string side;

        public CreateSandwich(string substance, string bread, string cheese, string 
toppings, string side)
        {
            this.substance = substance;
            this.bread = bread;
            this.cheese = cheese;
            this.toppings = toppings;
            this.side = side;
        }

        public Dish createDish()
        {
            return new Sandwich(substance, bread, cheese, toppings, side);
        }
    }
```

Lastly, the form puts it all together. When the pasta creation button is clicked, it creates a new pasta object, assigning it to the factory variable. The factory then calls the createDish() method and assigns it to the pasta. All of the information is entered into the textbox by the last method which retrieves the order from the concrete product classes. The same happens when you try to create a sandwich, the only difference is how the factory variable is assigned.

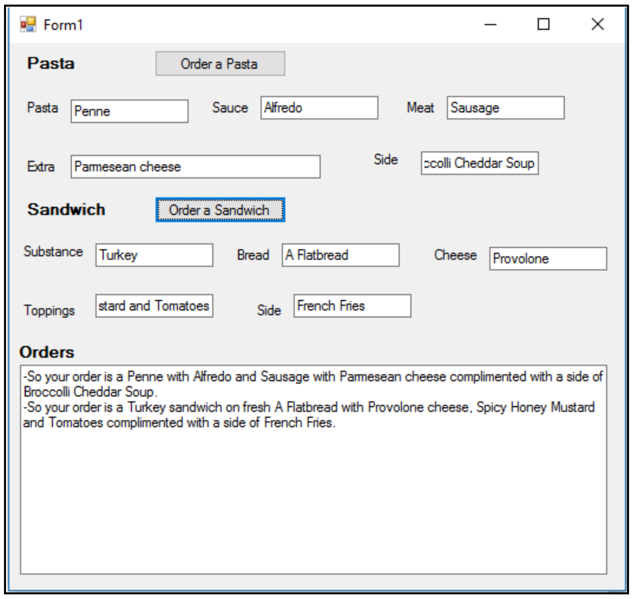
```
public partial class Form1 : Form
    {
        private CreateDish factory;
        private Dish newDish;
        public Form1()
        {
            InitializeComponent();
        }

        private void btn_Pasta_Click(object sender, EventArgs e)
        {
            factory = new CreatePasta(txtPasta.Text, txtSauce.Text, txtMeat.Text, 
txtExtra.Text, txtSideP.Text);
            newDish = factory.createDish();
            txtOrder.Text += newDish.getOrder() + "\n \n";
        }

        private void btn_Sandwich_Click(object sender, EventArgs e)
        {
            factory = new CreateSandwich(txtSub.Text, txtBread.Text, txtCheese.Text,
txtTopp.Text, txtSideS.Text);
            newDish = factory.createDish();
            txtOrder.Text += newDish.getOrder() + "\n \n";
        }

    }
```

Empty Order



Pasta: Penne Sauce: Alfredo Meat: Sausage
Extra: Parmesean cheese Side: Broccolli Cheddar S
Substance: Turkey Bread: A Flatbread Cheese: Provolone
Toppings: stard and Tomatoes Side: French Fries

Execution



Pasta: Penne Sauce: Alfredo Meat: Sausage
Extra: Parmesean cheese Side: :colli Cheddar Soup
Substance: Turkey Bread: A Flatbread Cheese: Provolone
Toppings: stard and Tomatoes Side: French Fries

```
-So your order is a Penne with Alfredo and Sausage with Parmesean cheese complimented with a side of
Broccolli Cheddar Soup.
-So your order is a Turkey sandwich on fresh A Flatbread with Provolone cheese, Spicy Honey Mustard
and Tomatoes complimented with a side of French Fries.
```

Observations

I owe a lot to DoFactory for the help it gave me in understanding this pattern. Having the pattern put into a real world example is useful when first reading about the design pattern. I had some difficulty when I did not realize that the methods in the Abstract Product needed to be used in just about everything, because almost everything inherits from it in one way or another. This was a good experience, and very satisfying when I got it completed and working.