Nathan Bemus

Design Patterns

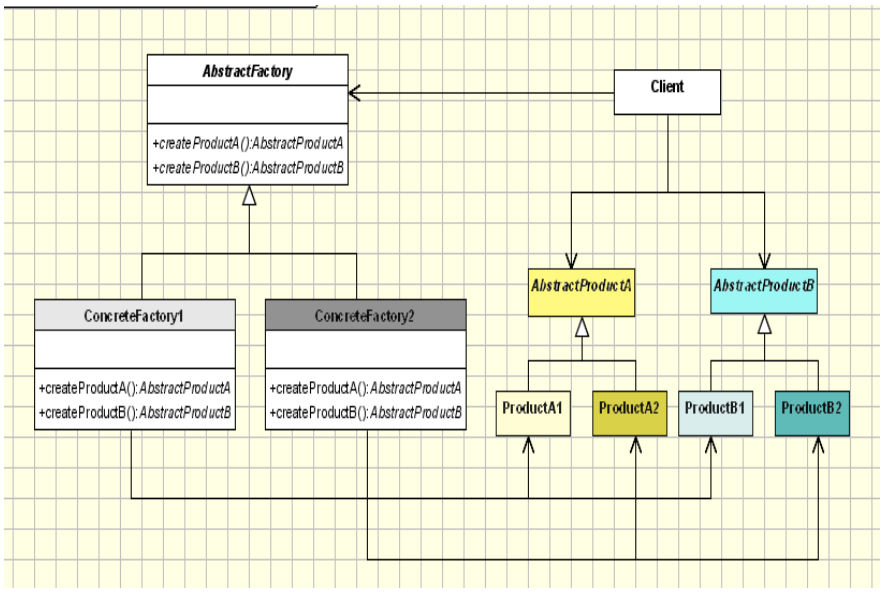
Abstract Factory Pattern

9/28/2016

Abstract Factory Pattern

Introduction

The purpose of this pattern is to provide an interface for creating a group of objects that are loosely related, but without specific initializations for the classes. This particular pattern creates a technological product based off of the company. There are three products to choose from per company.



UML Diagram Analysis

The diagram shows that the factory method pattern is divided into four different sections. In it lies the AbstractFactory Class, the ConcreteFactory Class, the AbstractProduct, and the Product Class. The client accesses the AbstractFactory and calls the function that creates the factory which creates the product.

Program Walkthrough

AbstractFactory Class

```
public abstract class AbstractFactory
    {
        public abstract PCComputer
createPCComputer(string Company, string elecType,
string OS);
```

This class is used to declare an interface for abstract products. The interface takes in three string arguments that are passed on throughout the code.

```
        public abstract PCTablet createPCTablet(string Company, string elecType, string
OS);
        public abstract PCPhone createPCPhone(string Company, string elecType, string
OS);

        public abstract MacComputer createMacComputer(string Company, string elecType,
string OS);
        public abstract MacTablet createMacTablet(string Company, string elecType, string
OS);
        public abstract MacPhone createMacPhone(string Company, string elecType, string
OS);
    }
```

PC Class (ConcreteFactory Class)

This class is used to create the constructor for the product. There is another class that is almost identical except for the Apple products.

```
//concrete factory
    public class PC : AbstractFactory
    {
        public override PCComputer
createPCComputer(string Company, string elecType, string OS)
        {
            return new PCComputer(Company, elecType, OS);
        }

        public override PCTablet createPCTablet(string Company, string elecType, string
OS)
        {
            return new PCTablet(Company, elecType, OS);
        }

        public override PCPhone createPCPhone(string Company, string elecType, string OS)
        {
            return new PCPhone(Company, elecType, OS);
        }

        public override MacComputer createMacComputer(string Company, string elecType,
string OS) { throw new NotImplementedException(); }
        public override MacTablet createMacTablet(string Company, string elecType, string
OS) { throw new NotImplementedException(); }
        public override MacPhone createMacPhone(string Company, string elecType, string
OS) { throw new NotImplementedException(); }
    }
```

Computer Class (Abstract Class)

This class creates the actuall product based off of which kind of computer (Mac vs. PC). The product that is created is a message box that displays the technological product and all of its details.

```
public class PCComputer
    {
        public PCComputer() { }

        public PCComputer(string Company, string
elecType, string OS)
        {
            MessageBox.Show("Company Brand: " + Company + '\n' + "Electronic Type: " +
elecType +
```

```
                '\n' + "Operating System: " + OS, "Product Created");
        }
    }

    public class MacComputer
    {
        public MacComputer() { }

        public MacComputer(string Company, string elecType, string OS)
        {
            MessageBox.Show("Company Brand: " + Company + '\n' + "Electronic Type: " +
elecType +
                    '\n' + "Operating System: " + OS, "Product Created");
        }
    }
```

Form1 Code (Client)

This is the main class where the form is generated and is called to the screen. All of the class variables are declared in this form. The form then fires specific events based off of which button is pressed. The buttons click event assigns values to the string variables that get passed to the other classes. Each individual button calls a separate function to create the unique message box (product).

```
public partial class Form1 : Form
    {
        PC factory1;
        Mac factory2;
        PCComputer pcc;
        PCTablet pct;
        PCPhone pcp;
        MacComputer mc;
        MacTablet mt;
        MacPhone mp;

        string company;
        string elecType;
        string OS;

        public Form1()
        {
            InitializeComponent();
            factory1 = new PC();
            factory2 = new Mac();
            pcc = new PCComputer();
            pct = new PCTablet();
            pcp = new PCPhone();
            mc = new MacComputer();
            mt = new MacTablet();
            mp = new MacPhone();
        }

        private void btnPCComputer_Click(object sender, EventArgs e)
        {
            company = "Microsoft";
            elecType = "Computer";
            OS = "Windows 10";
            factory1.createPCComputer(company, elecType, OS);
        }

        private void btnPCTablet_Click(object sender, EventArgs e)
        {
```

```
            company = "Microsoft";
            elecType = "Tablet";
            OS = "Windows 10";
            factory1.createPCTablet(company, elecType, OS);
        }

        private void btnPCPhone_Click(object sender, EventArgs e)
        {
            company = "Microsoft";
            elecType = "Phone";
            OS = "Windows 10 Mobile";
            factory1.createPCPhone(company, elecType, OS);
        }

        private void btnMacComputer_Click(object sender, EventArgs e)
        {
            company = "Apple";
            elecType = "Computer";
            OS = "OSX";
            factory2.createMacComputer(company, elecType, OS);
        }

        private void btnMacTablet_Click(object sender, EventArgs e)
        {
            company = "Apple";
            elecType = "Tablet";
            OS = "iOS 10";
            factory2.createMacTablet(company, elecType, OS);
        }

        private void btnMacPhone_Click(object sender, EventArgs e)
        {
            company = "Apple";
            elecType = "Phone";
            OS = "iOS 10";
            factory2.createMacPhone(company, elecType, OS);
        }
    }
```

Reflection

The abstract factory pattern uses the AbstractFactory Class to call functions that create the interface for the product. The classes also use inheritance to pass along values to each other to finish the product.

Conclusion

Overall I think I have a general grasp of how this pattern works and how it could be used in the real world in a light example. I feel this assignment was simple at times due to how similar it was the Factory Method, however it had enough twists in the code

that still made it a challenge. I think I have learned enough about this pattern to explain a general concept to someone else.