Jackson Spencer

The Factory Method Pattern

September 19, 2016

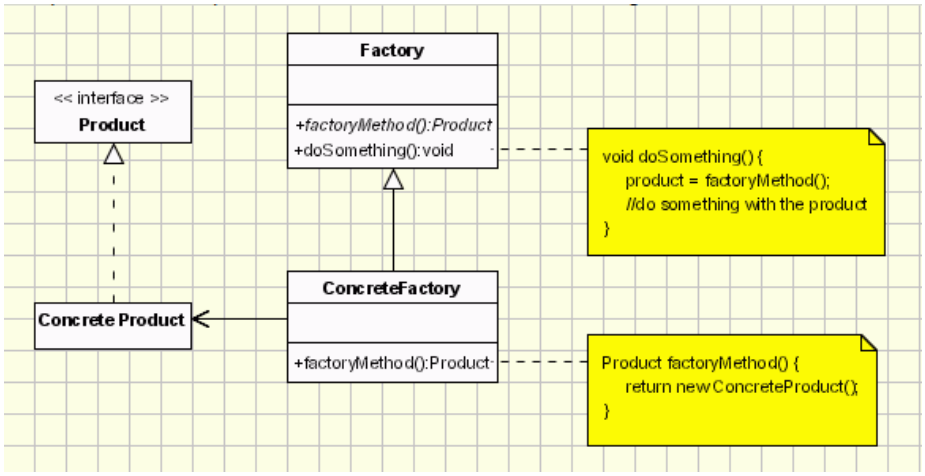
ECCS 2411: Design Patterns

**Introduction**

The assignment required an application to demonstrate the Factory Method design pattern. We were to implement the pattern using classes and an interface. The application simply creates a text file, using the interface and subclasses. The other components are shown in the UML diagram.

**UML Diagram**

This diagram shows the classes required to implement the design pattern. The Concrete Product in my case was called TextFile, because it created a text file. The factory class shows what is required to create the file. The Product class shows what the file's components are. The TextFile class mentioned earlier, is a subclass of Product.




**Classes**

The Product class is an interface, which contains methods that each file will use.

```
public interface Product
    {
        string getFileName();
        string getFileType();
        void generateFile();
        void writeFile(string text);
```

```
    }
```

The Factory class is responsible for the creation of files. The method generateFile() creates a file on the user's local machine. The newFile() method is an abstract method that creates an object of the Product class.

```
public abstract class Factory
    {
        public void generateFile(string filetype, string filename, string text)
        {
            Product file = newFile(filetype, filename);
            file.generateFile();
            file.writeFile(text);
        }

        public abstract Product newFile(string filetype, string filename);
    }
```

The Concrete Factory class determines whether or not the file is a text file, and returns null if otherwise.

```
class ConcreteFactory : Factory
    {
        public override Product newFile(string filetype, string filename)
        {
            if (filetype == "Text File")
                return new TextFile(filename);
            return null;
        }
    }
```

The TextFile class is a subclass of Product. It contains methods, getFileType() and getFileName(), to get the name and type of file. The generateFile() method creates the file with a name and type. The writeFile() method writes a string to be passed to the generated file.

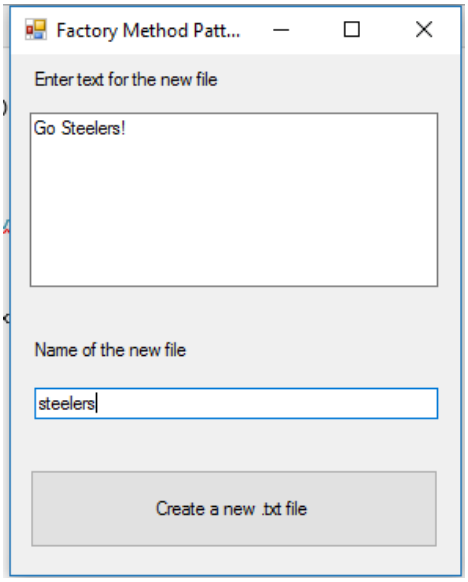
```
class TextFile : Product
    {
        private string name;
        private string fileType = ".txt";
        public TextFile(string name)
        {
            this.name = name;
        }

        public void generateFile()
        {
            System.IO.File.Create("C:\\Temp\\" + name + fileType).Close();
        }
```

```
        public void writeFile(string text)
        {
            System.IO.File.WriteAllText("C:\\Temp\\" + name + fileType, text);
        }

        public string getFileType()
        {
            return fileType;
        }

        public string getFileName()
        {
            return name;
        }
    }
```

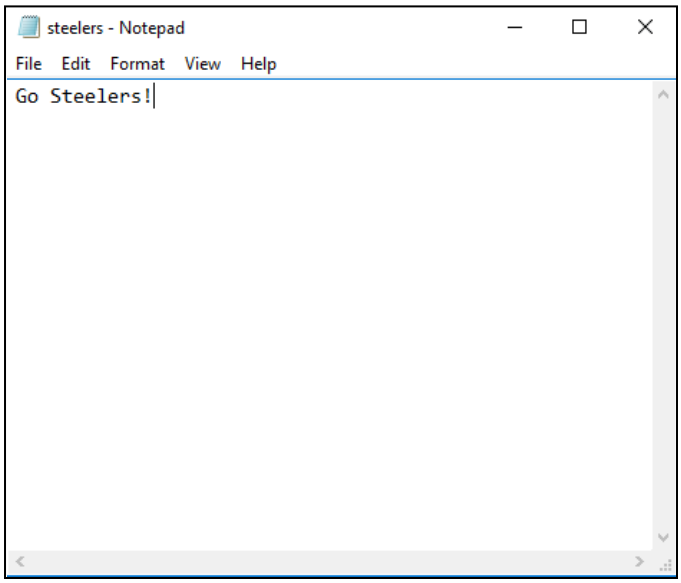
The form implements the Concrete Factory class. It simply controls the action when the button is clicked. The name of the file is retrieved as well as the type of file.

```
public partial class Form1 : Form
    {
        string name;
        ConcreteFactory factory = new ConcreteFactory();
        public Form1()
        {
            InitializeComponent();
        }
        private void btn_Create_Click(object sender, EventArgs e)
        {
            name = txt_Name.Text;
            string text = txt_EnterText.Text;
            factory.generateFile("Text File", name, text);
        }
    }
```

**Form**




**Product**



**Observations**

This pattern was difficult to produce. I understood what to do, however I felt as though I had too little of time to complete it. In fact, I was forced to turn it in late due to my not being able to complete it on time. The functionality I chose to use in this project was an interesting idea that was initially taught to us in Programming 1. I feel as though I have a fair idea of how the pattern works. I only wish that I had been able to flesh out the application more.