Factory Method Pattern

Design Patterns

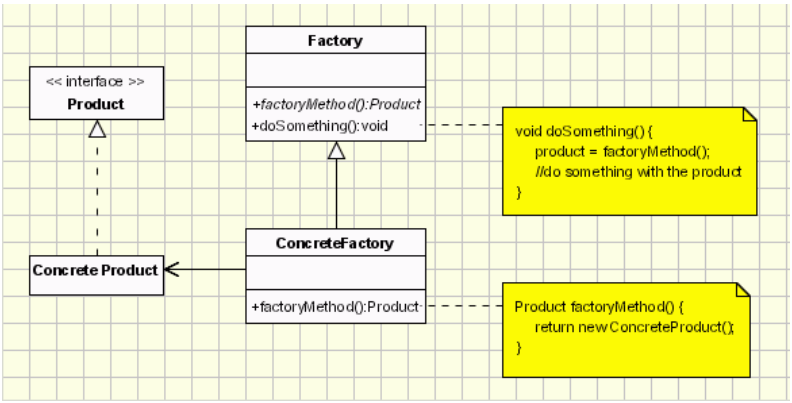
Christopher Doseck

9/16/2015

Introduction

This assignment is an application that I created to show how the factory method pattern works. In this application I am using an interface to represent a basic file, and am using subclasses of that to represent a csv file and a text file. Another class that I used was the factory class, and the factory method class, which created files and wrote text to those files.

The UML Diagram for Factory Method



The UML diagram for the factory method pattern, shown on the right, shows the classes that are needed to have the requirements. I had the file interface as the product interface, the csv file and text file as two concrete products, the factory class as the factory class, and the concrete factory as the concrete factory. The table below shows how each of the classes were used.

Factory	This class shows what would be required to create a file in general, no matter what file it is.
Concrete Factory	This class made it so that you could specify which type of file you were creating.
File	This is a class that shows what all files need, no matter what type of file they are. It represents a file that has no specifications.
Text File	This class represents a text file, and has all of the attributes of a text file. It is a subclass of the File class.
CSV File	This class represents a csv file, and has all of the attributes of a csv file. It is a subclass of the File Class

Form 1	This is the graphical user interface, which shows the factory method actually working.

Narrative

File

```
public interface File
{
    string getName();
    string getFileType();
    void createFile();
    void writeToFile(string text);
}
```

This is the setup of the file class. All files will have these methods in common.

TextFile

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

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

    }

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

    public string getFileType()
    {
        return fileType;
    }

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

This is a text file class. It is a subclass of the file class. It has methods to get the name and fileType. Create file creates the file with that fileType and closes it after creation. Write to File writes the string that it is passed to the file.

CSVFile

```
class CSVFile : File
{
    private string fileType = ".csv";
    private string name;

    public CSVFile(string name)
    {
        this.name = name;
```

This is the CSV file; it also is a subclass of the file class. It is very similar to the text file class.

```
    }

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

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

    public string getFileType()
    {
        return fileType;
    }

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

Factory

```
public abstract class Factory
{
    public void createFile(string type, string name, string text)
    {
        File file = createNewFile(type, name);
        file.createFile();
        file.writeToFile(text);
    }
    public abstract File createNewFile(string type, string name);
}
```

This is the factory class. It is responsible for creating files, even if it doesn't know what the type of the file it is. It has the method createFile which will create a file on the computer, and write text to it. It also has the createNewFile method, which is an abstract method to create an object of the file class.

ConcreteFactory

```
class ConcreteFactory : Factory
{
    public override File createNewFile(string type, string name)
    {
        if (type == "Text File")
            return new TextFile(name);
        else if (type == "CSV File")
            return new CSVFile(name);
        return null;
    }
}
```

This is the concrete factory class. It determines which type of file to create, and returns that file when called.

Form1

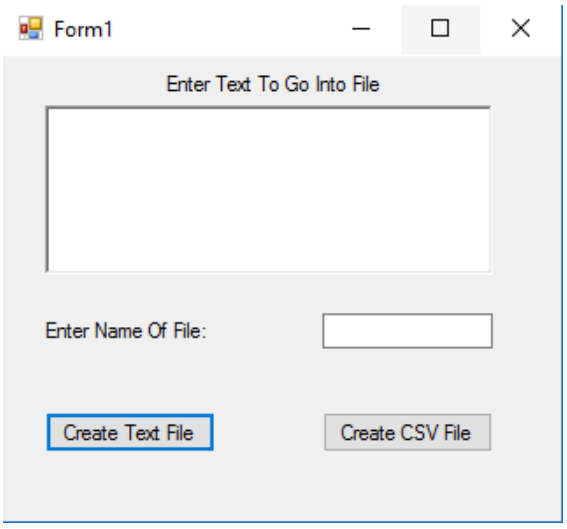
```
public partial class Form1 : Form
{
    string fileName;
    ConcreteFactory factory = new ConcreteFactory();
    public Form1()
    {
        InitializeComponent();
    }
```

This is the form that implements the concrete factory class.

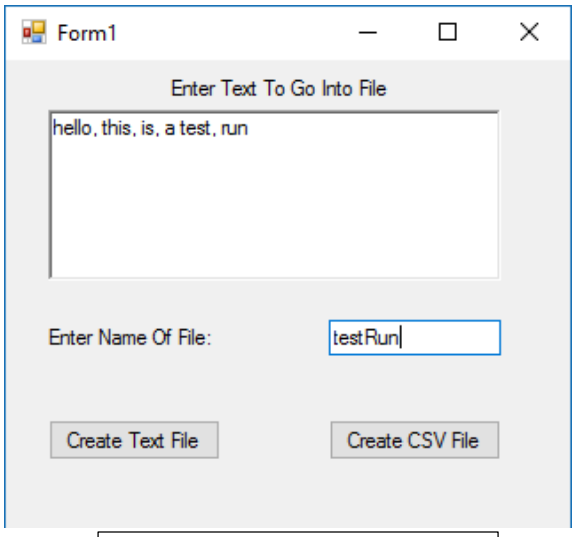
```
    private void btnCreateTextFile_Click(object sender, EventArgs e)
    {
        fileName = tbFileName.Text;
        string text = tbUserText.Text;
        factory.createFile("Text File", fileName, text);
    }

    private void btnCSVFile_Click(object sender, EventArgs e)
    {
        fileName = tbFileName.Text;
        string text = tbUserText.Text;
        factory.createFile("CSV File", fileName, text);
    }
}
```

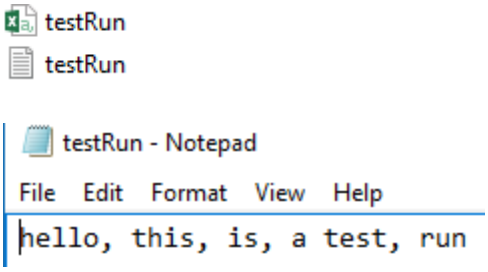
This shows how the form gets the information to the concrete factory class and to creating a file on the computer.



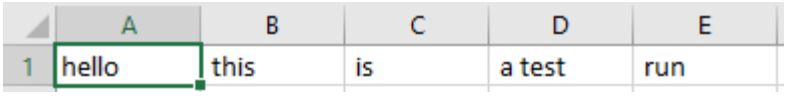
This is a Blank GUI



GUI with information filled in



These are the files that were created after hitting create text file, and create csv file, and what is stored in them



	A	B	C	D	E
1	hello	this	is	a test	run

Observations

This was a rather confusing design pattern to figure it out. I felt that the professor did not go over it enough in class. Even after finishing this pattern, I do not really feel like it is that important of a pattern.