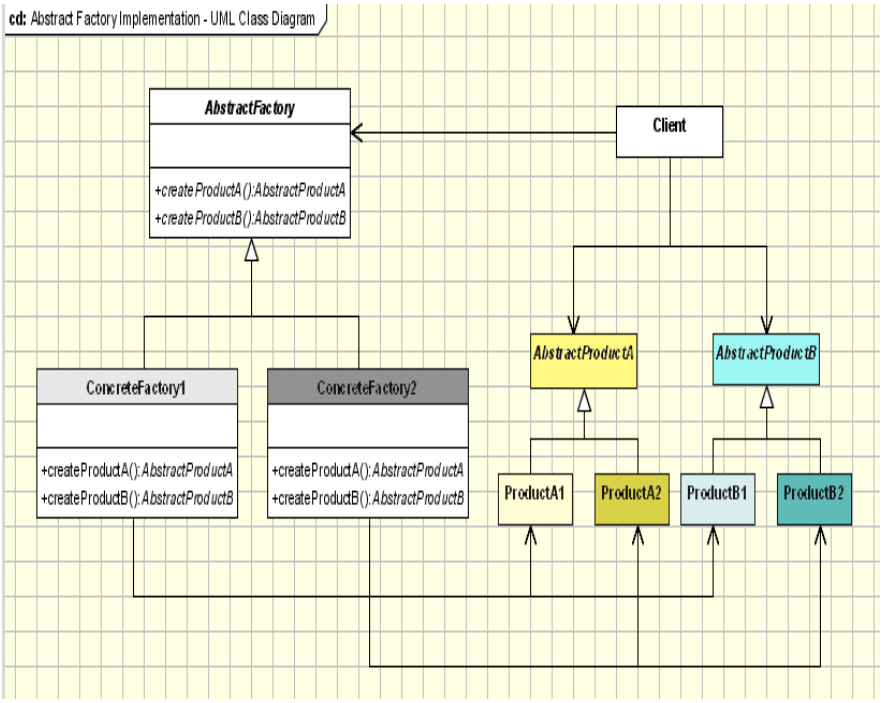
Introduction

For this program I had to create a program that offered an interface to create a family of related objects without explicitly specifying their class. The UML diagram on the side illustrates this. My subject for the pattern is a computer factory that when a button is clicked, a form pops up to show the specs of the desired model.



Abstract Factory

To start this project, I first created an abstract factory that I used to then create my two concrete factories. The two concrete factories then created the products based on the methods created by the abstract products. I'll start with the code for the abstract factory and work my way through.

```
   public abstract class AbstractFactory
    {
        public abstract Desktop createDesktop(ConcretePC.CPU m_CPUPart, ConcretePC.GPU
m_GPUPart, ConcretePC.Ram m_RamPart);

        public abstract Laptop createLaptop(ConcretePC.CPU m_CPUPart, ConcretePC.GPU
m_GPUPart, ConcretePC.Ram m_RamPart);

        public abstract Macbook createMacbook(ConcreteMac.CPU m_CPUPart, ConcreteMac.GPU
m_GPUPart, ConcreteMac.Ram m_RamPart);

        public abstract iMac createiMac(ConcreteMac.CPU m_CPUPart, ConcreteMac.GPU
m_GPUPart, ConcreteMac.Ram m_RamPart);
    }
```

The code above is the abstract factory class creating the abstract factories that the concrete factories will pull from later when producing the product.

```
public class ConcretePC : AbstractFactory
    {
        public enum CPU { i7, i5}
        public enum GPU { gtx970, integrated}
        public enum Ram { sixteen, four}

        private CPU _CPUPart;
        private GPU _GPUPart;
```

```
        private Ram _RamPart;

        public CPU m_CPUPart
        {
            get { return _CPUPart; }
            set
            {
                _CPUPart = value;
            }
        }

        public GPU m_GPUPart
        {
            get { return _GPUPart; }
            set
            {
                _GPUPart = value;
            }
        }

        public Ram m_RamPart
        {
            get { return _RamPart; }
            set
            {
                _RamPart = value;
            }
        }

        public override Desktop createDesktop(ConcretePC.CPU m_CPUPart, ConcretePC.GPU
m_GPUPart, ConcretePC.Ram m_RamPart)
        {
            return new Desktop(m_CPUPart,m_GPUPart,m_RamPart);
        }

        public override Laptop createLaptop(ConcretePC.CPU m_CPUPart, ConcretePC.GPU
m_GPUPart, ConcretePC.Ram m_RamPart)
        {
            return new Laptop(m_CPUPart, m_GPUPart, m_RamPart);
        }

        public override iMac createiMac(ConcreteMac.CPU m_CPUPart, ConcreteMac.GPU
m_GPUPart, ConcreteMac.Ram m_RamPart)
        {
            throw new NotImplementedException();
        }

        public override Macbook createMacbook(ConcreteMac.CPU m_CPUPart, ConcreteMac.GPU
m_GPUPart, ConcreteMac.Ram m_RamPart)
        {
            throw new NotImplementedException();
        }
    }
```

This code is just one of the concrete factories. The other concrete factory, which is called ConcreteMac, has the same code, just reworded for the Mac side of it. The enums set up what can actually be in the

text boxes for the second form. The private variables are set so that they can be used in this class, which then returns them as a value to be taken by the Desktop class, shown below.

```
    public class Desktop : AbstractDesktop
    {
        public Desktop()
        {

        }

        public Desktop(ConcretePC.CPU m_CPUPart, ConcretePC.GPU m_GPUPart, ConcretePC.Ram
m_RamPart)
        {
            Form2 f2 = new Form2(m_CPUPart,m_GPUPart,m_RamPart);
            f2.Visible = true;
        }
    }
    public class iMac : AbstractDesktop
    {
        public iMac()
        {

        }

        public iMac(ConcreteMac.CPU m_CPUPart, ConcreteMac.GPU m_GPUPart, ConcreteMac.Ram
m_RamPart)
        {
            Form2 f2 = new Form2(m_CPUPart, m_GPUPart, m_RamPart);
            f2.Visible = true;
        }
    }
```

The Desktop class seen above has both the ConcretePC and ConcreteMac products in it. But as can be seen, they have very similar set ups. The public Desktop constructor calls the arguments from the abstract factory to open and populate the second form with the desired specs.

```
    public partial class Form1 : Form
    {
        ConcretePC concretePC;
        ConcreteMac concreteMac;
        Desktop desktop;
        Laptop laptop;
        iMac iMac;
        Macbook macbook;

        public Form1()
        {
            InitializeComponent();
            concretePC = new ConcretePC();
            concreteMac = new ConcreteMac();
            desktop = new Desktop();
            laptop = new Laptop();
            iMac = new iMac();
            macbook = new Macbook();
        }

        private void btnPC_Click(object sender, EventArgs e)
```

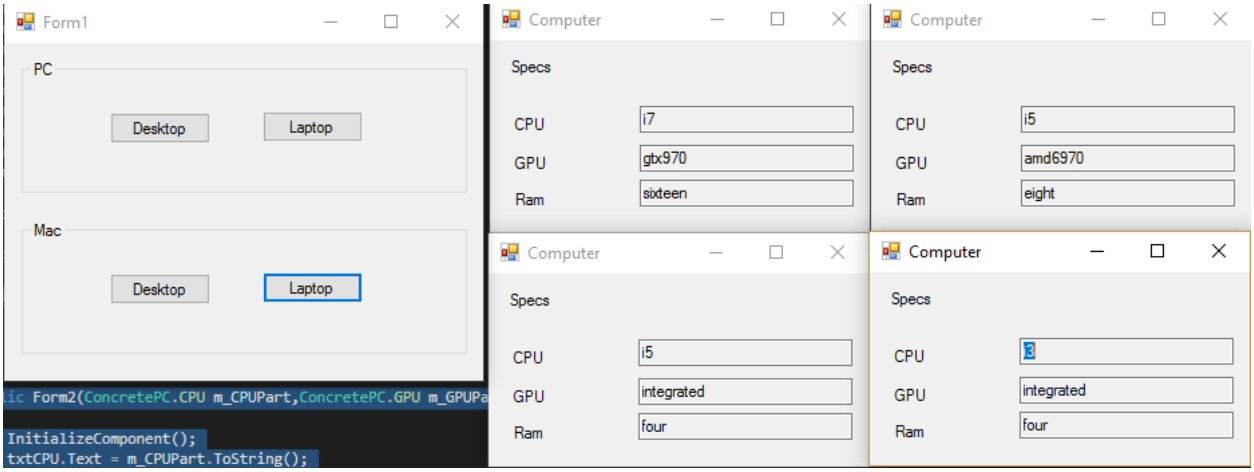
```
        {
            concretePC.m_CPUPart = ConcretePC.CPU.i7;
            concretePC.m_GPUPart = ConcretePC.GPU.gtx970;
            concretePC.m_RamPart = ConcretePC.Ram.sixteen;
            concretePC.createDesktop(concretePC.m_CPUPart, concretePC.m_GPUPart,
concretePC.m_RamPart);
        }

        private void btnPCTop_Click(object sender, EventArgs e)
        {
            concretePC.m_CPUPart = ConcretePC.CPU.i5;
            concretePC.m_GPUPart = ConcretePC.GPU.integrated;
            concretePC.m_RamPart = ConcretePC.Ram.four;
            concretePC.createLaptop(concretePC.m_CPUPart, concretePC.m_GPUPart,
concretePC.m_RamPart);
        }

        private void btnMac_Click(object sender, EventArgs e)
        {
            concreteMac.m_CPUPart = ConcreteMac.CPU.i5;
            concreteMac.m_GPUPart = ConcreteMac.GPU.amd6970;
            concreteMac.m_RamPart = ConcreteMac.Ram.eight;
            concreteMac.createiMac(concreteMac.m_CPUPart, concreteMac.m_GPUPart,
concreteMac.m_RamPart);
        }

        private void btnMacTop_Click(object sender, EventArgs e)
        {
            concreteMac.m_CPUPart = ConcreteMac.CPU.i3;
            concreteMac.m_GPUPart = ConcreteMac.GPU.integrated;
            concreteMac.m_RamPart = ConcreteMac.Ram.four;
            concreteMac.createMacbook(concreteMac.m_CPUPart, concreteMac.m_GPUPart,
concreteMac.m_RamPart);
        }
    }
```

Above is form 1 and below is form 2.

```
        public Form2()
        {

        }

        public Form2(ConcretePC.CPU m_CPUPart,ConcretePC.GPU m_GPUPart,ConcretePC.Ram
m_RamPart)
        {
            InitializeComponent();
            txtCPU.Text = m_CPUPart.ToString();
            txtGPU.Text = m_GPUPart.ToString();
            txtRam.Text = m_RamPart.ToString();
        }

        public Form2(ConcreteMac.CPU m_CPUPart,ConcreteMac.GPU m_GPUPart, ConcreteMac.Ram
m_RamPart)
        {
            InitializeComponent();
            txtCPU.Text = m_CPUPart.ToString();
            txtGPU.Text = m_GPUPart.ToString();
```

```
            txtRam.Text = m_RamPart.ToString();
        }
    }
```

While the above code looks really complicated, especially form 1, it is in fact just setting up the buttons to do their desired functions. And form 2 is set up to take the values it is given, turn it into a string and put it into the text boxes. When it is all done, you get the finished program seen below.



This is the program running with each of the buttons pressed and their corresponding forms open.

Observations

When I first started this program I didn't quite understand how it was different than the factory pattern. But after working on it for a while and going through the struggles, it makes a lot more sense. Given the opportunity, I would like to make this app more complicated and with more information. But this will work for demo purposes.