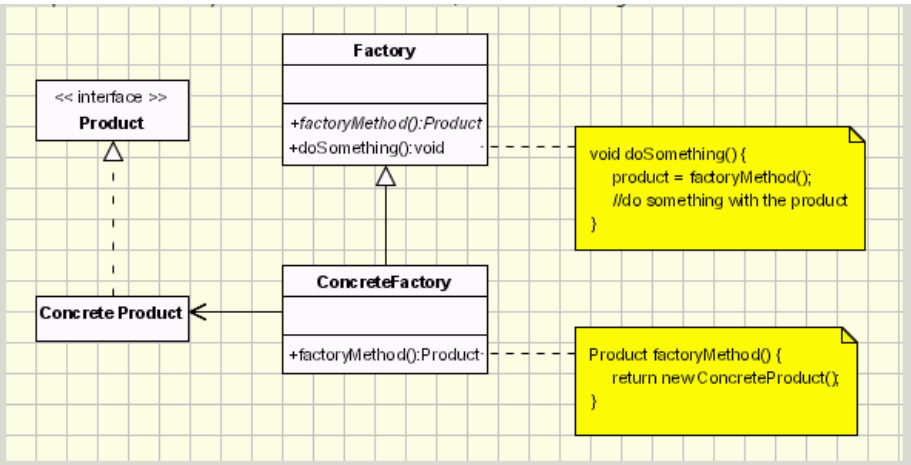
Introduction

For this assignment, the Factory Pattern, I had to define an interface for creating objects, but let the subclasses decide which class to instantiate. I also had to refer to the newly created object through a common interface. I ended up adding onto my past Façade pattern by adding in the needed classes as shown on the UML diagram which is shown to the right.



Assignment

As stated above, after trying to just make a factory pattern from scratch, I decided to just add on to my façade pattern application, the universal remote. I had to add four classes, Factory, ConcreteFactory, ConcreteProduct, and Product. The factory class is where I set up the arguments for the MakeSettings method, which is what I used to populate the new form. The code below is the factory class.

```
  public abstract class Factory
    {
        public void MakeSettings(TV.TVPowerState PowerState, TV.TVInputState InputState,
SurroundSound.SoundState SoundState, DVD.DVDState DVDState, Game.GameState GameState)
        {
          Product product = factoryMethod(PowerState, InputState, SoundState, DVDState,
GameState);
        }

        internal abstract Product factoryMethod(TV.TVPowerState PowerState,
TV.TVInputState InputState, SurroundSound.SoundState SoundState, DVD.DVDState DVDState,
Game.GameState GameState);
    }
```

Next I made the ConcreteFactory class. This class is what I used to actually make the concrete product. It derived from the factory class, as in used the same arguments, in order to return the ConcreteProduct.

```
public class ConcreteFactory : Factory
    {
        internal override Product factoryMethod(TV.TVPowerState PowerState,
TV.TVInputState InputState, SurroundSound.SoundState SoundState, DVD.DVDState DVDState,
Game.GameState GameState)
        {
            return new ConcreteProduct(PowerState, InputState, SoundState, DVDState,
GameState);
        }
    }
```

Since the ConcreteFactory makes the ConcreteProduct, that is where I went next. This is what makes the product so that the product class can return it.

```
    public class ConcreteProduct : Product
    {
        public Product product;

        public ConcreteProduct()
        {

        }

        public ConcreteProduct(TV.TVPowerState PowerState, TV.TVInputState InputState,
SurroundSound.SoundState SoundState, DVD.DVDState DVDState, Game.GameState GameState)
        {
            product = new Product(PowerState, InputState, SoundState, DVDState,
GameState);
            product.Visible = true;
        }

    }
```

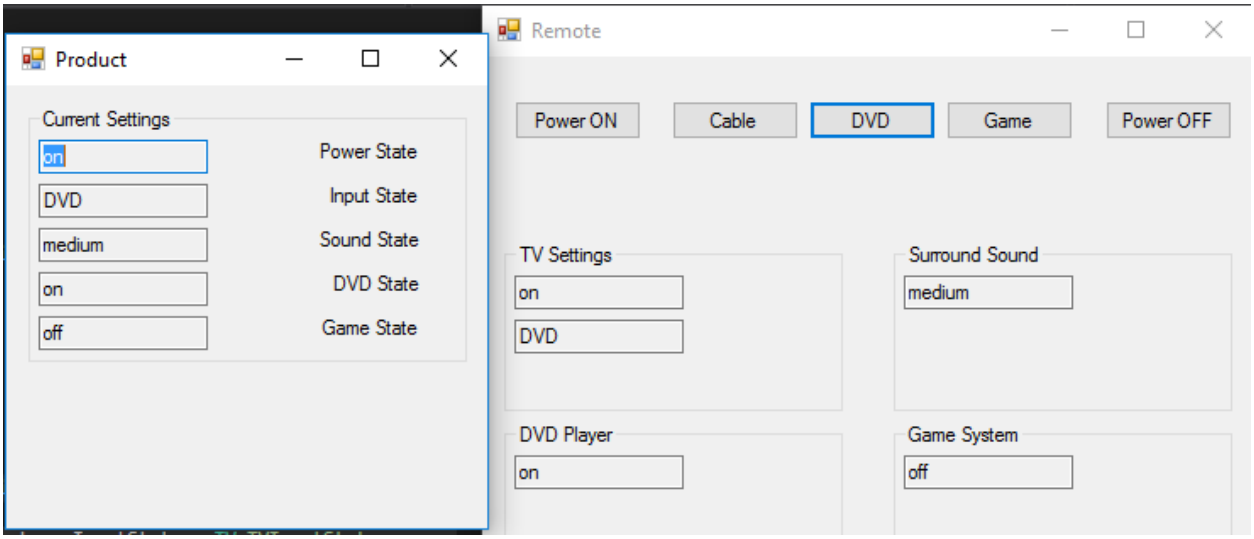
The above code uses the previously established arguments to set up the product. The product class is then able to take the information given by the original façade pattern and by the new factory class and turn it into strings to populate the new form textboxes. The code below shows this.

```
public partial class Product : Form
    {
        public Product()
        {

        }

        public Product(TV.TVPowerState PowerState, TV.TVInputState InputState,
SurroundSound.SoundState SoundState, DVD.DVDState DVDState, Game.GameState GameState)
        {
            InitializeComponent();
            pwrState.Text = PowerState.ToString();
            inpState.Text = InputState.ToString();
            sndState.Text = SoundState.ToString();
            dvdState.Text = DVDState.ToString();
            gameState.Text = GameState.ToString();
        }
    }
```

After these new classes have been made and coded, I had to add code to the buttons on the original form in order to make the new code actually be called.

```
    private void GameBtn_Click(object sender, EventArgs e)
        {
            tv.m_InputState = TV.TVInputState.game;
            gs.m_GameState = Game.GameState.on;
            dvd.m_DVDState = DVD.DVDState.off;
            ss.m_SSState = SurroundSound.SoundState.loud;
            factory.MakeSettings(tv.m_State, tv.m_InputState, ss.m_SSState,
dvd.m_DVDState, gs.m_GameState);
        }
```

The last line added to the button calls the MakeSettings method, which opens a new form with new textboxes that are then populated with the current settings.



Observations

I actually had a lot of trouble with this pattern. Mostly just grasping the concept of what I needed to do. I started out just trying to make a new program but I kept getting lost and messing up. Eventually I just decided to add onto the façade pattern which I knew worked and had compatibility with the factory pattern. I may revisit this program later and streamline it, clean up the code a bit. But this is what I have so far.