Design Patterns
Prof. Retterer
Factory Method

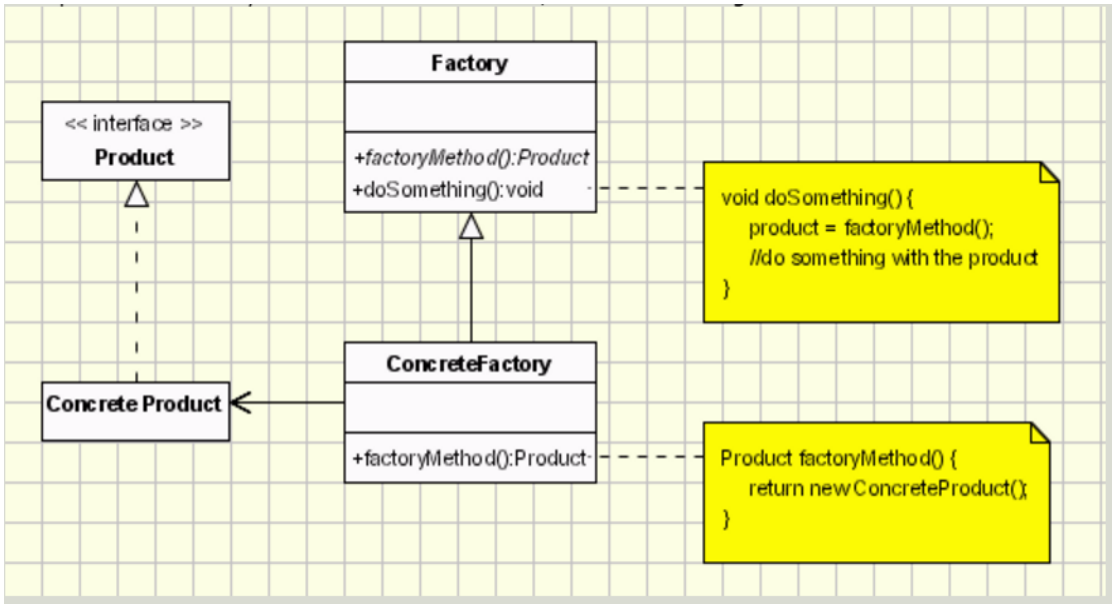
Jeremy Martin
9/15/16

Introduction

For this assignment we were told to implement the Factory Method pattern. I used the code from my Facade pattern to help me achieve this. The program will now make an object that holds all the commands that the party will do when the tactic is selected.

UML Diagram for Factory Method

The UML diagram is shown to the right, and displays the fact that the factory class and concreteFactory class must create a new product.



Now I will show you the new pieces of code that I made to implement this into my facade pattern and the changed code from the original program.

The first piece of code shows the Factory abstract class.

```
public abstract class Factory //this is the Factory class
    {
        public void createParty(Warrior.WarriorState m_WarriorState,
Rogue.RogueState m_RogueState, White_Mage.WhiteMageState m_WhiteMageState,
Black_Mage.BlackMageState m_BlackMageState)
        {
            Party party =
factoryMethod(m_WarriorState,m_RogueState,m_WhiteMageState,m_BlackMageState);
        }

        protected abstract Party factoryMethod(Warrior.WarriorState
m_WarriorState, Rogue.RogueState m_RogueState, White_Mage.WhiteMageState
m_WhiteMageState, Black_Mage.BlackMageState m_BlackMageState);
    }
```

The next piece shows the ConcreteFactory class.

```
  public class ConcreteFactory : Factory //this is the ConcreteFactory class
    {
```

```
        protected override Party factoryMethod(Warrior.WarriorState
m_WarriorState, Rogue.RogueState m_RogueState, White_Mage.WhiteMageState
m_WhiteMageState, Black_Mage.BlackMageState m_BlackMageState)
        {
            return new
ConcreteParty(m_WarriorState,m_RogueState,m_WhiteMageState,m_BlackMageState);
        }
    }
```

The next piece shows the Product form.

```
public partial class Party : Form //this is the Product class
    {
        public Party()
        {

        }

        public Party(Warrior.WarriorState m_WarriorState, Rogue.RogueState
m_RogueState, White_Mage.WhiteMageState m_WhiteMageState,
Black_Mage.BlackMageState m_BlackMageState)
        {
            InitializeComponent();
            m_tbWarrior.Text = m_WarriorState.ToString();
            m_tbRogue.Text = m_RogueState.ToString();
            m_tbWhiteMage.Text = m_WhiteMageState.ToString();
            m_tbBlackMage.Text = m_BlackMageState.ToString();
        }
    }
```

The next piece shows the ConcreteProduct class.

```
public class ConcreteParty : Party //this is the ConcreteProduct class
    {
        public Party party;

        public ConcreteParty()
        {
        }

        public ConcreteParty(Warrior.WarriorState m_WarriorState, Rogue.RogueState
m_RogueState, White_Mage.WhiteMageState m_WhiteMageState,
Black_Mage.BlackMageState m_BlackMageState)
        {
            party = new
Party(m_WarriorState,m_RogueState,m_WhiteMageState,m_BlackMageState);
            party.Visible = true;
        }
    }
```

The last few pieces of code are changes that I made to the basic form for the facade pattern.

```
      Warrior war;
       Rogue rg;
       White_Mage wm;
       Black_Mage bm;
      ConcreteParty concreteParty;

Factory factory; public Form1()
```

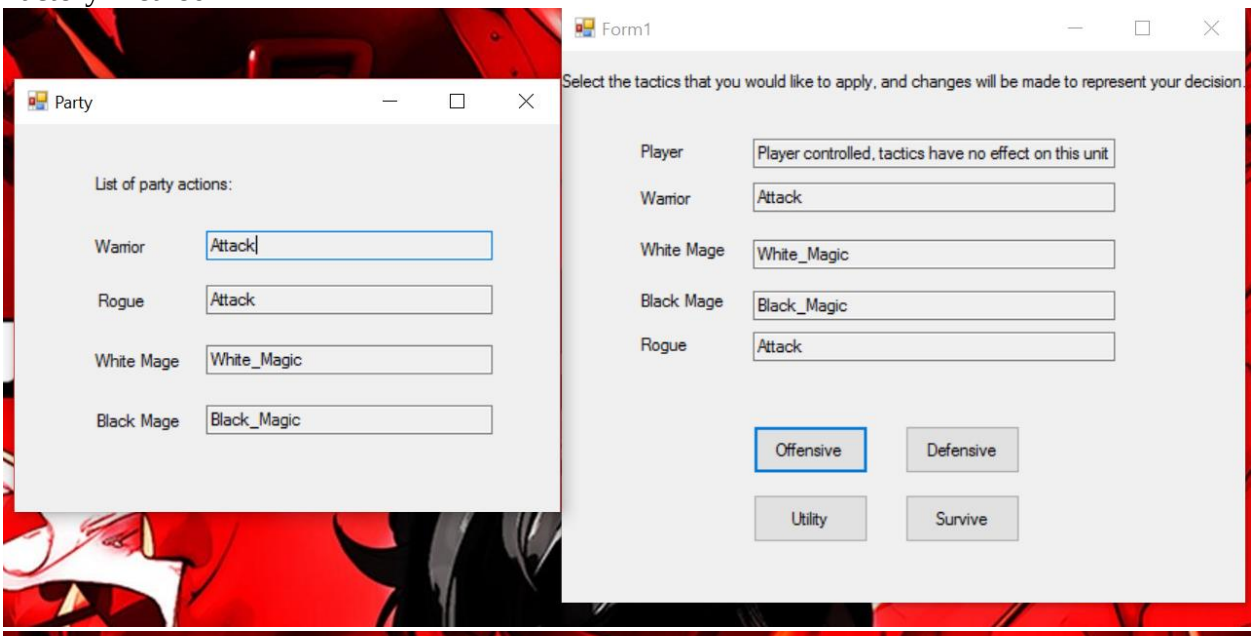
```
        {
            InitializeComponent();
            war = new Warrior();
            rg = new Rogue();
            wm = new White_Mage();
            bm = new Black_Mage();
            concreteParty = new ConcreteParty();
            factory = new ConcreteFactory();
            war.WarriorStateChanged += new
WarriorStateChangedEventHandler(war_WarriorStateChanged);
            rg.RogueStateChanged += new
RogueStateChangedEventHandler(rg_RogueStateChanged);
            wm.WhiteMageStateChanged += new
WhiteMageStateChangedEventHandler(wm_WhiteMageStateChanged);
            bm.BlackMageStateChanged += new
BlackMageStateChangedEventHandler(bm_BlackMageStateChanged);
        }

factory.createParty(war.m_WarriorState, rg.m_RogueState, wm.m_WhiteMageState,
bm.m_BlackMageState);
```

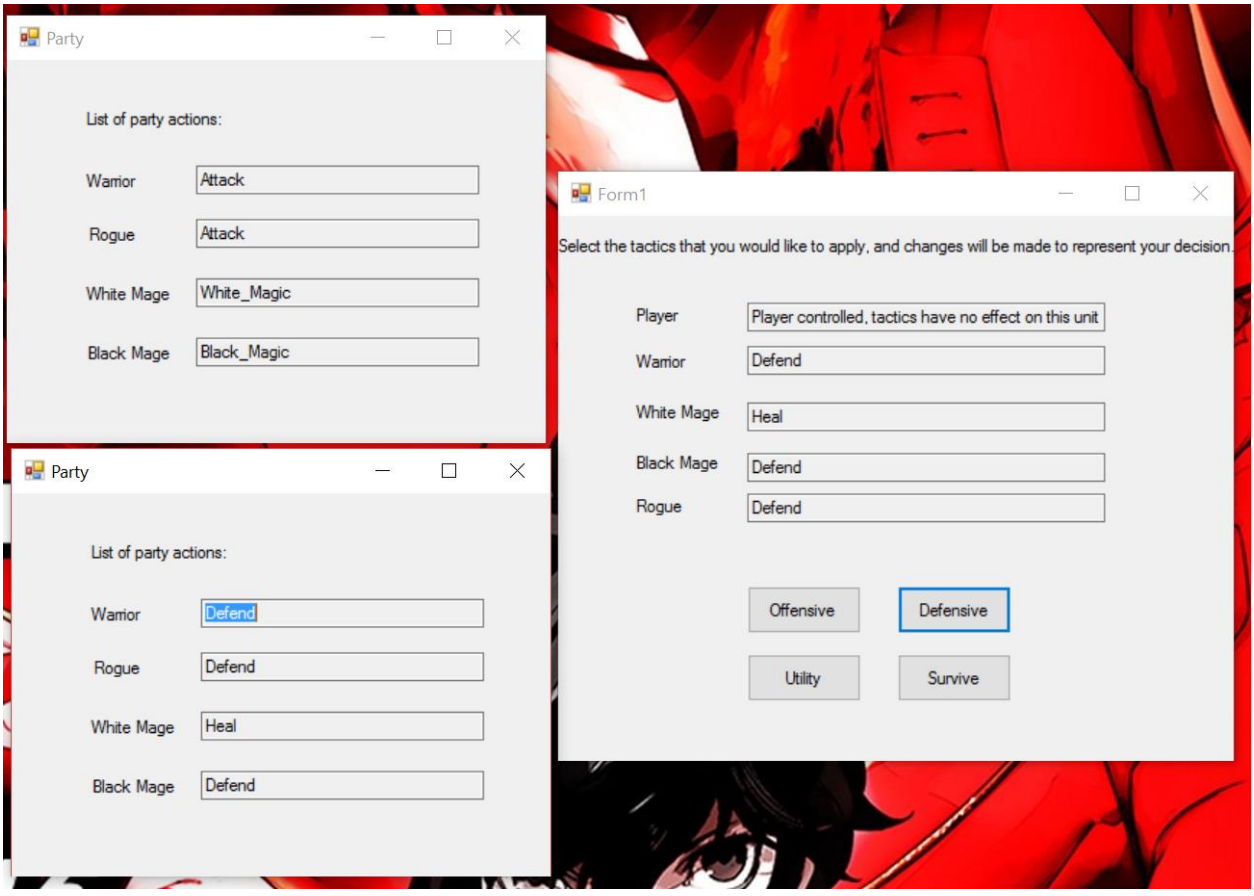
The last line of code was added to all of the buttons that changed the states of the other classes in the facade.

Here are some screenshots of the new forms being created when a button is clicked.

Design Patterns
Prof. Retterer
Factory Method

Jeremy Martin
9/15/16





Design Patterns
Prof. Retterer
Factory Method
Conclusions

Jeremy Martin
9/15/16

Overall I feel like I should've just spent the extra time to implement this to the original code in the first place. I spent several hours trying to figure out how to make this code work and was just about ready to give up on it. I finally figured out that my main problem was the fact that I wasn't initializing the Factory object in the right place in the form code. Once I found the right place to initialize it, the code worked without a problem, but figuring out what was wrong with the code was a lot harder than I expected it to be. I'm just really glad to be done with this project now.