

Tori Lentz

Design Patterns

Dave Retterer

November 11, 2016

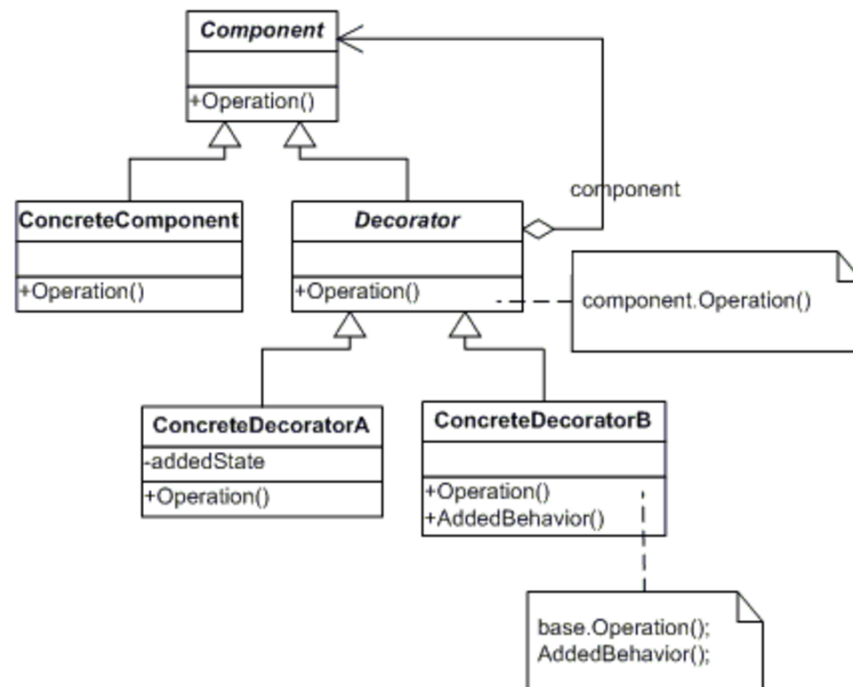
## Decorator Pattern

### Introduction

The purpose of this assignment is to implement the Decorator Pattern which is used to extend an objects functionality dynamically. An objects functionality can be extended statically as well meaning, at compile time, the functions will be run. With dynamic functionality, however, a variety of other options can be created and added to the program.

### UML Diagram

## UML class diagram



Component	This class defines the interface for the objects of this pattern.
Concrete Component	This class defines the objects to which additional responsibilities can be attached.
Decorator	The decorator class defines an interface that conforms to the Component classes interface, as well as, maintaining a reference to a Component object.
Concrete Decorator	This class implements the new responsibilities to the components.

Shown above is the UML diagram for the decorator pattern. The entire pattern is made up of four different classes: Component, Concrete Component, Decorator and Concrete Decorator. The Component defines the interface to be used by all the objects contained within the program. It is an abstract class so, it is meant to be used as a base. The next part is the Concrete Component and within this class all the objects are defined. Thirdly, is the Decorator class which defines the interface that will conform to the interface already defined in the Component class. This class will also maintain references to the Component object. Lastly is the Concrete Decorator class which is used to attach all the responsibilities to the components that have been defined in the other classes.

### Code and Description

#### **Component**

```
public abstract class CharacterComponent
{
    public abstract string GetCharacter(string charactername);
}

public abstract class CostumeComponent
{
    public abstract string GetName(string name);
    public abstract double GetPrice(double price);
    public abstract string GetSize(string size);
}
```

This is my component class. Within it are two different classes which define the two parts of my program, the character component and the costume component. The character component only deals with those objects that have to do with the character, whereas, the costume component has three objects within it which will get the name, price and size of the clothes picked.

#### **Concrete Component 1**

```
class CostumeBase : CostumeComponent
{
    private string v;

    public CostumeBase(string v)
    {
        this.v = v;
    }

    public override string GetName(string name)
    {
        return name;
    }

    public override double GetPrice(double price)
    {
        return price;
    }

    public override string GetSize(string size)
    {
        return size;
    }
}
```

This is my first concrete class which deals with all the costume components as defined in the component class. The methods here will return the values for name, price and size that will be further defined in the decorator classes. This class inherits from the public abstract class component.

```
}
```

## Concrete Component 2

```
class CostumeCharacter : CharacterComponent
{
    private string v;

    public CostumeCharacter(string v)
    {
        this.v = v;
    }

    public override string GetCharacter(string charactername)
    {
        return charactername;
    }
}
```

This is my other concrete component class which, again, inherits from my component class. This class only deals with my character components and the method defined as GetCharacter will return the character name that will be picked.

## Decorator

```
class ClothingDecorator : CostumeComponent
{
    CostumeComponent m_BaseComponent = null;

    protected string m_Name = "Undefined Decorator";
    protected double m_Price = 0.0;
    protected string m_Size = "Undefined Decorator";

    protected ClothingDecorator(CostumeComponent baseComponent)
    {
        m_BaseComponent = baseComponent;
    }

    public override string GetName(string charactername)
    {
        return string.Format("{0}, {1}", m_BaseComponent.GetName(m_Name),
charactername);
    }

    public override double GetPrice(double price)
    {
        return price + m_BaseComponent.GetPrice(m_Price);
    }

    public override string GetSize(string size)
    {
        return size + m_BaseComponent.GetSize(m_Size);
    }
}
```

This is my decorator class which will decorate my character with different clothes. Here, the methods for name, price and size are further defined and, the actual functions are returned.

## Concrete Decorator 1

```
class CharacterMaskDecorator : ClothingDecorator
{
    public CharacterMaskDecorator(CostumeComponent baseComponent)
```

This is my first concrete decorator class which deals with the option of decorating the chosen character with a mask.

```

        : base(baseComponent)
    {
        this.m_Name = "Head Mask";
        this.m_Price = 45.0;
        this.m_Size = "";
    }
}

```

## Concrete Decorator 2

```

class CharacterShoesDecorator : ClothingDecorator
{
    public CharacterShoesDecorator(CostumeComponent baseComponent)
        : base(baseComponent)
    {
        this.m_Name = "Shoes";
        this.m_Price = 20.0;
        this.m_Size = "";
    }
}

```

My second concrete decorator class deals with the option of decorating the chosen character with shoes.

## Form

```

public partial class CharacterDesignForm : Form
{
    public CharacterDesignForm()
    {
        InitializeComponent();
    }

    private void CharacterDesignForm_Load(object sender,
EventArgs e)
    {
        button1.Enabled = false;
    }

    private void comboBox1_SelectedIndexChanged(object sender,
EventArgs e)
    {
        string displayText = "";

        if (comboBox1.Text != "" && comboBox2.Text != "" &&
comboBox3.Text != "")
        {
            button1.Enabled = true;
        }
        else {
            button1.Enabled = false;
        }
        textBox1.Text = displayText;
    }

    private void button1_Click(object sender, EventArgs e)
    {
        string displayText = textBox1.Text;
        CostumeCharacter baseCharacter = new CostumeCharacter(comboBox1.Text);

        String s = comboBox2.Text;
    }
}

```

Displayed here is my form code. Upon the form load, the button will be disabled and as the user picks options, the button will then be enabled. The button will get all the chosen selections and will return those selections to the text box, as well as, the other information that has been defined in the decorator classes and concrete decorator classes.

```

String searchString = ":";
int startIndex = s.IndexOf(searchString) + 1;
int endIndex = s.Length - startIndex;
String substring = s.Substring(startIndex, endIndex);
double selectedPrice = Convert.ToDouble(substring);
string selectedTransaction = s.Substring(0, startIndex - 1);
String selectedSize = comboBox3.Text;

// Pants Base
CostumeBase basecharacter = new CostumeBase(selectedTransaction);
displayText = "Selected Character: " +
baseCharacter.GetCharacter(comboBox1.Text).ToString() + "\r\n";
displayText = displayText + "Purchase/Rent: " +
basecharacter.GetName(selectedTransaction).ToString() + "\r\n";
displayText = displayText + "Selected Size: " +
basecharacter.GetSize(selectedSize).ToString() + "\r\n";
displayText = displayText + "Total Price: $" +
basecharacter.GetPrice(selectedPrice).ToString() + "\r\n\r\n\r\n";

if (checkBox1.Checked == true && checkBox2.Checked == true)
{
    // Mask Decorator Concrete
    CharacterMaskDecorator maskCharacter1 = new
CharacterMaskDecorator(basecharacter);
    displayText = displayText + "Selected Character: " +
baseCharacter.GetCharacter(comboBox1.Text).ToString() + "\r\n";
    displayText = displayText + "Purchase/Rent: " +
maskCharacter1.GetName(selectedTransaction).ToString() + "\r\n";
    displayText = displayText + "Selected Size: " +
maskCharacter1.GetSize(selectedSize).ToString() + "\r\n";
    displayText = displayText + "Total Price: $" +
maskCharacter1.GetPrice(selectedPrice).ToString() + "\r\n\r\n\r\n";

    // Shoes Decorator Concrete
    CharacterShoesDecorator shoesCharacter1 = new
CharacterShoesDecorator(maskCharacter1);
    displayText = displayText + "Selected Character: " +
baseCharacter.GetCharacter(comboBox1.Text).ToString() + "\r\n";
    displayText = displayText + "Purchase/Rent: " +
shoesCharacter1.GetName(selectedTransaction).ToString() + "\r\n";
    displayText = displayText + "Selected Size: " +
shoesCharacter1.GetSize(selectedSize).ToString() + "\r\n";
    displayText = displayText + "Total Price: $" +
shoesCharacter1.GetPrice(selectedPrice).ToString() + "\r\n\r\n\r\n";
}
else if (checkBox1.Checked == true && checkBox2.Checked == false)
{
    // Mask Decorator Concrete
    CharacterMaskDecorator maskCharacter = new
CharacterMaskDecorator(basecharacter);
    displayText = displayText + "Selected Character: " +
baseCharacter.GetCharacter(comboBox1.Text).ToString() + "\r\n";
    displayText = displayText + "Purchase/Rent: " +
maskCharacter.GetName(selectedTransaction).ToString() + "\r\n";
    displayText = displayText + "Selected Size: " +
maskCharacter.GetSize(selectedSize).ToString() + "\r\n";
    displayText = displayText + "Total Price: $" +
maskCharacter.GetPrice(selectedPrice).ToString() + "\r\n\r\n\r\n";
}

```

```

    }
    else if (checkBox1.Checked == false && checkBox2.Checked == true)
    {
        // Shoes Decorator Concrete
        CharacterShoesDecorator shoesCharacter = new
CharacterShoesDecorator(basecharacter);
        displayText = displayText + "Selected Character: " +
baseCharacter.GetCharacter(comboBox1.Text).ToString() + "\r\n";
        displayText = displayText + "Purchase/Rent: " +
shoesCharacter.GetName(selectedTransaction).ToString() + "\r\n";
        displayText = displayText + "Selected Size: " +
shoesCharacter.GetSize(selectedSize).ToString() + "\r\n";
        displayText = displayText + "Total Price: $" +
shoesCharacter.GetPrice(selectedPrice).ToString() + "\r\n\r\n\r\n\r\n";
    }

    textBox1.Text = displayText;
}

private void comboBox2_SelectedIndexChanged(object sender, EventArgs e)
{
    string displayText = "";

    if (comboBox1.Text != "" && comboBox2.Text != "" && comboBox3.Text != "")
    {
        button1.Enabled = true;
    }
    else {
        button1.Enabled = false;
    }
    textBox1.Text = displayText;
}

private void checkBox1_CheckedChanged(object sender, EventArgs e)
{
}

private void checkBox2_CheckedChanged(object sender, EventArgs e)
{
    string displayText = "";

    if (comboBox1.Text != "" && comboBox2.Text != "" && comboBox3.Text != "")
    {
        button1.Enabled = true;
    }
    else {
        button1.Enabled = false;
    }
    textBox1.Text = displayText;
}

private void checkBox1_CheckedChanged_1(object sender, EventArgs e)
{
    string displayText = "";

    if (comboBox1.Text != "" && comboBox2.Text != "" && comboBox3.Text != "")

```

```

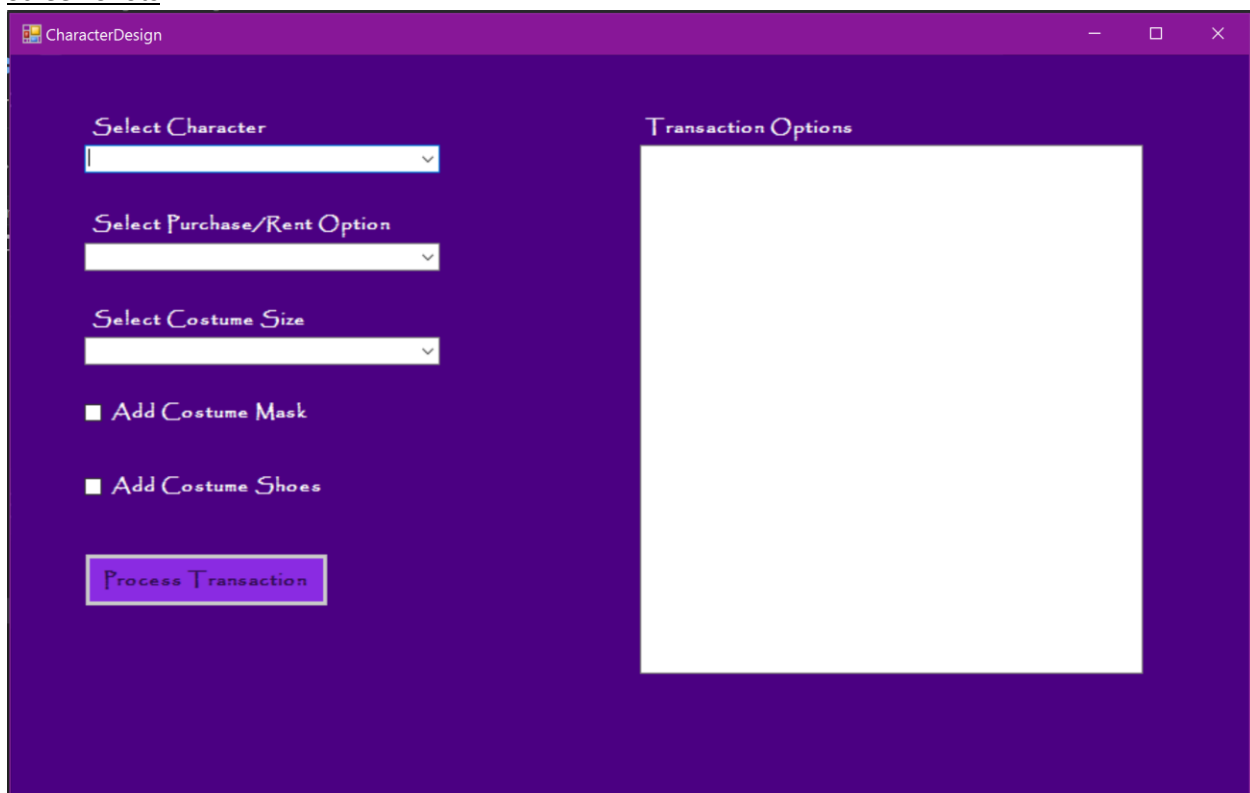
    {
        button1.Enabled = true;
    }
    else {
        button1.Enabled = false;
    }
    textBox1.Text = displayText;
}

private void comboBox3_SelectedIndexChanged(object sender, EventArgs e)
{
    string displayText = "";

    if (comboBox1.Text != "" && comboBox2.Text != "" && comboBox3.Text != "")
    {
        button1.Enabled = true;
    }
    else {
        button1.Enabled = false;
    }
    textBox1.Text = displayText;
}
}

```

### Screen Shots



The screenshot shows a Windows application window titled "CharacterDesign". The interface has a dark blue background. On the left side, there are three white dropdown menus labeled "Select Character", "Select Purchase/Rent Option", and "Select Costume Size". Below these are two checkboxes: "Add Costume Mask" and "Add Costume Shoes", both of which are currently unchecked. At the bottom left is a white button labeled "Process Transaction". On the right side, there is a large white rectangular area titled "Transaction Options".

This is the form upon load

This is the form when only add costume mask is checked

The screenshot shows the 'CharacterDesign' application window. On the left, there are three dropdown menus: 'Select Character' with 'Belle' selected, 'Select Purchase/Rent Option' with 'Purchase Base: 75.00' selected, and 'Select Costume Size' with 'Small' selected. Below these are two checkboxes: 'Add Costume Mask' (checked) and 'Add Costume Shoes' (unchecked). A 'Process Transaction' button is at the bottom left. On the right, a 'Transaction Options' box displays the following text:

Selected Character: Belle  
Purchase/Rent: Purchase Base  
Selected Size: Small  
Total Price: \$75

Selected Character: Belle  
Purchase/Rent: Head Mask, Purchase Base  
Selected Size: Small  
Total Price: \$120

The screenshot shows the 'CharacterDesign' application window with the same settings as the previous one, but with both 'Add Costume Mask' and 'Add Costume Shoes' checked. The 'Transaction Options' box now displays three entries:

Selected Character: Belle  
Purchase/Rent: Purchase Base  
Selected Size: Small  
Total Price: \$75

Selected Character: Belle  
Purchase/Rent: Head Mask, Purchase Base  
Selected Size: Small  
Total Price: \$120

Selected Character: Belle  
Purchase/Rent: Head Mask, Shoes, Purchase Base  
Selected Size: Small  
Total Price: \$140

This is the form when both add costume shoes and add costume mask are checked



This is the form when only add costume shoes is checked

The screenshot shows a Windows application window titled "CharacterDesign". The interface has a dark blue background. On the left side, there are three dropdown menus: "Select Character" with "Belle" selected, "Select Purchase/Rent Option" with "Purchase Base: 75.00" selected, and "Select Costume Size" with "Small" selected. Below these are two checkboxes: "Add Costume Mask" (unchecked) and "Add Costume Shoes" (checked). At the bottom left is a blue button labeled "Process Transaction". On the right side, there is a white rectangular area titled "Transaction Options" which displays the following text: "Selected Character: Belle", "Purchase/Rent: Purchase Base", "Selected Size: Small", and "Total Price: \$75". Below this, it shows "Selected Character: Belle", "Purchase/Rent: Shoes, Purchase Base", "Selected Size: Small", and "Total Price: \$95".

### Observations

Over all, this pattern went well. I had some trouble understanding how it was supposed to work the first few times of looking at it because it is so much different than the patterns we had previously done but, after looking at some examples, I finally understood how the pattern worked and was able to go from there. I was able to incorporate several different things into this project, such as, checkboxes, drop downs and several other things that I had not previously incorporated into other projects so, I enjoyed figuring these things out.