Design Patterns
Prof. Retterer

Jeremy Martin
11/17/16

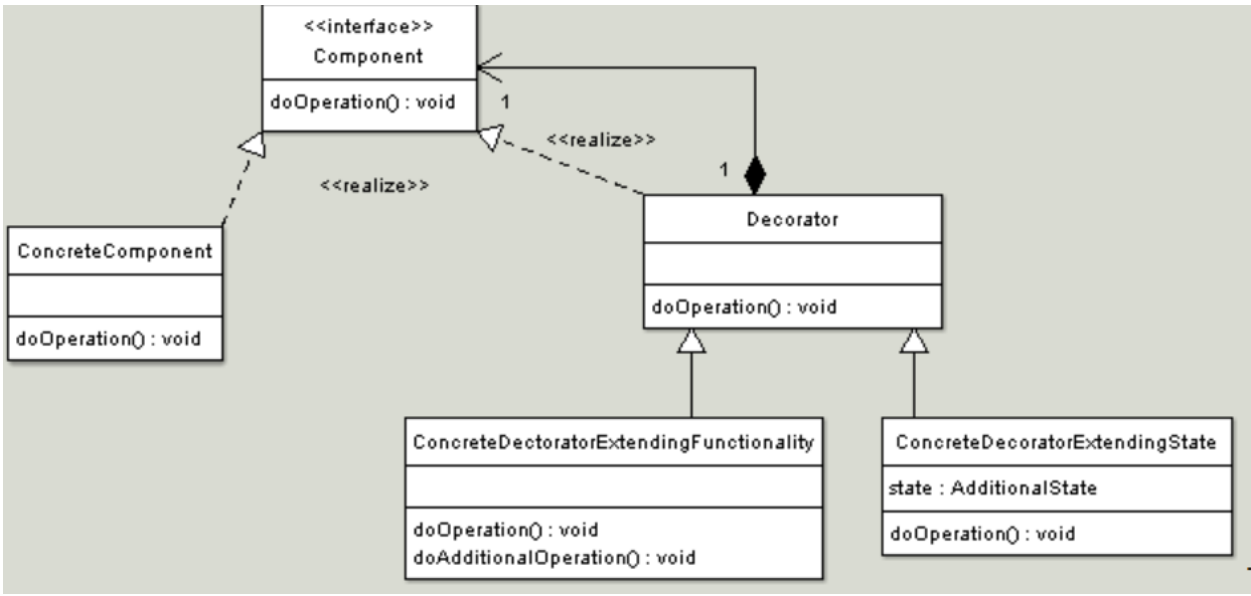
# Decorator Pattern

## Introduction

For this assignment, we were tasked with making an application that demonstrates the Decorator pattern. I chose to make an application that make sundaes and the toppings are the objects used to decorate the sundae.

## Decorator UML

To the right you will see the UML diagram for the Decorator pattern.




In my application the component class is Sundae, ConcreteSundae is the ConcreteComponent class, and Decorator is the Decorator class. The ConcreteDecorators are taken care of by the ice cream flavors and the toppings.

Now we will move on to the code for the application, we will start with the Component class.

```
public abstract class Sundae //this is the Component class
    {
        public abstract string orderSundae(string s);
    }
```

Next, we will move on to the two classes that implement from the Sundae class, ConcreteSundae and Decorator.

```
public class ConcreteSundae : Sundae // this is the ConcreteComponent class
    {
        public override string orderSundae(string s)
        {
            s += " one sundae with";
            return s;
        }
    }

public abstract class Decorator : Sundae // this is the Decorator class
    {
```

```
        public abstract override string orderSundae(string s);
    }
```

Now we will move on to the two sets of ConcreteDecorators that I made for the sundae, the first decorator deals with the flavor of ice cream and the second deals with the toppings. Since most of the code is repeated between the different classes, I will only show a few examples of each class.

```
public class Vanilla : Sundae //these are the ConcreteDecorator classes
    {
        public override string orderSundae(string s)
        {
            s += " vanilla ice cream";
            return s;
        }
    }

    public class Chocolate : Sundae
    {
        public override string orderSundae(string s)
        {
            s += " chocolate ice cream";
            return s;
        }
    }

public class Sprinkles : Decorator // these are the ConcreteDecorator classes
    {
        public override string orderSundae(string s)
        {
            s += " and sprinkles";
            return s;
        }
    }

    public class ChocolateSyrup : Decorator
    {
        public override string orderSundae(string s)
        {
            s += " and chocolate syrup";
            return s;
        }
    }

public class Reeses : Decorator
    {
        public override string orderSundae(string s)
        {
            s += " and Reese's Pieces";
            return s;
        }
    }

    public class Oreo : Decorator
    {
        public override string orderSundae(string s)
        {
            s += " and Oreo pieces";
```

```
            return s;
        }
    }
```

Now for the final piece of code, we will look at the From1 code.

```
public partial class Form1 : Form
    {
        Sundae sundae;
        string s;

        public Form1()
        {
            InitializeComponent();
            s = "You have ordered";
        }

        private void m_btnOrder_Click(object sender, EventArgs e)
        {
            sundae = new ConcreteSundae();
            s = sundae.orderSundae(s);

            if(m_rbVanilla.Checked)
            {
                sundae = new Vanilla();
            }
            else if(m_rbChocolate.Checked)
            {
                sundae = new Chocolate();
            }
            else if(m_rbStrawberry.Checked)
            {
                sundae = new Strawberry();
            }
            else if(m_rbTwist.Checked)
            {
                sundae = new Twist();
            }
            else if(m_rbNeo.Checked)
            {
                sundae = new Neopolitan();
            }
            s = sundae.orderSundae(s);

            if(m_cbSprinkles.Checked)
            {
                sundae = new Sprinkles();
                s = sundae.orderSundae(s);
            }
            if(m_cbChocoSyrup.Checked)
            {
                sundae = new ChocolateSyrup();
                s = sundae.orderSundae(s);
            }
            if(m_cbHotFudge.Checked)
            {
                sundae = new HotFudge();
                s = sundae.orderSundae(s);
            }
```

Decorator Pattern

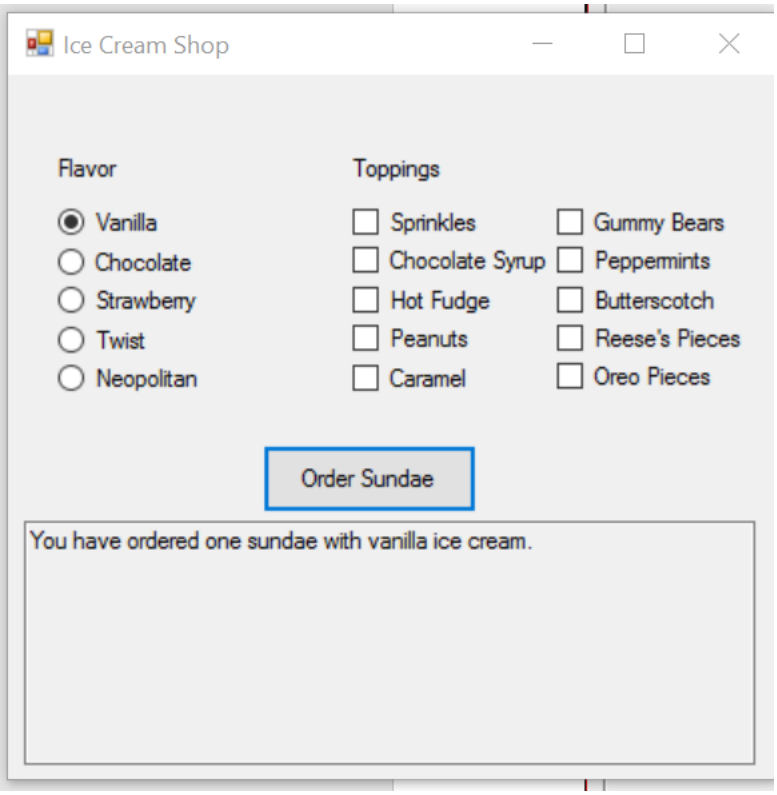
```
            if(m_cbPeanuts.Checked)
            {
                sundae = new Peanuts();
                s = sundae.orderSundae(s);
            }
            if(m_cbCaramel.Checked)
            {
                sundae = new Caramel();
                s = sundae.orderSundae(s);
            }
            if(m_cbGummyBears.Checked)
            {
                sundae = new GummyBears();
                s = sundae.orderSundae(s);
            }
            if(m_cbPeppermints.Checked)
            {
                sundae = new Peppermints();
                s = sundae.orderSundae(s);
            }
            if(m_cbButterscotch.Checked)
            {
                sundae = new Butterscotch();
                s = sundae.orderSundae(s);
            }
            if(m_cbReese.Checked)
            {
                sundae = new Reeses();
                s = sundae.orderSundae(s);
            }
            if(m_cbOreo.Checked)
            {
                sundae = new Oreo();
                s = sundae.orderSundae(s);
            }

            m_tbOrderLog.Text = s+"."+Environment.NewLine+m_tbOrderLog.Text;
            s = "You have ordered";
        }
    }
```

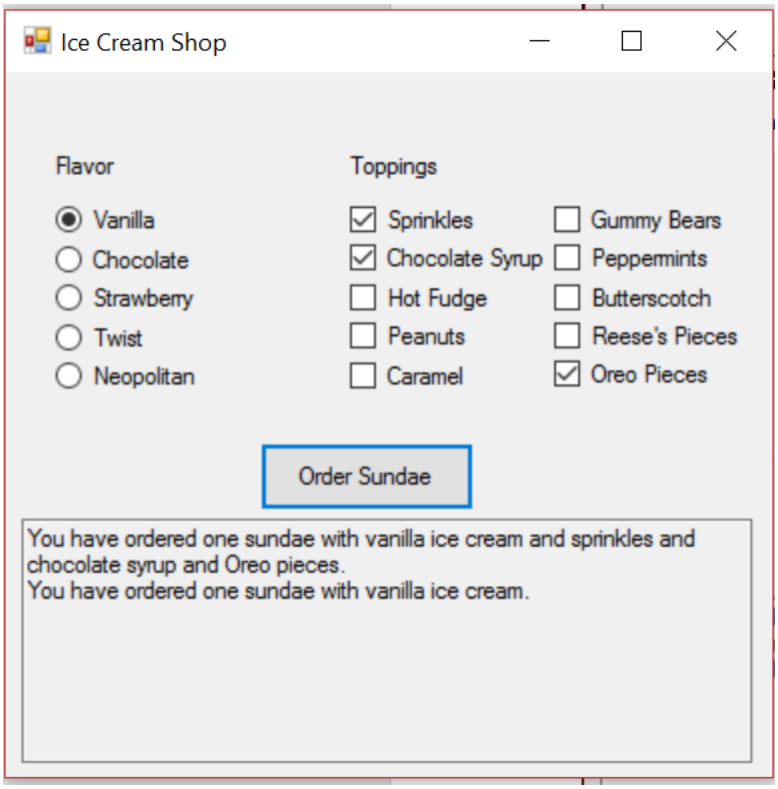
Design Patterns
Prof. Retterer

Jeremy Martin
11/17/16

Decorator Pattern

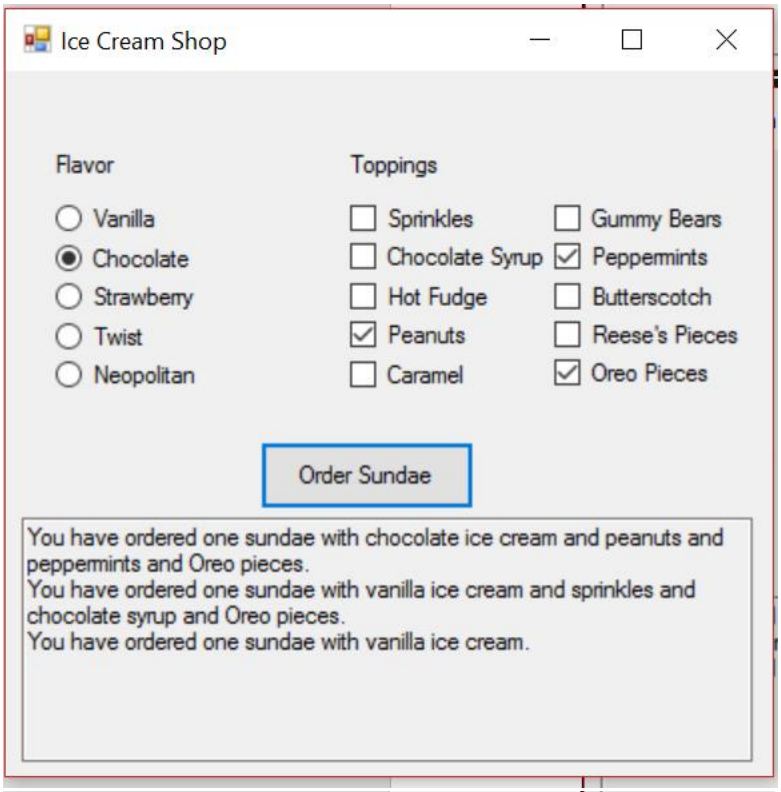
Here are some screenshots of the application running.

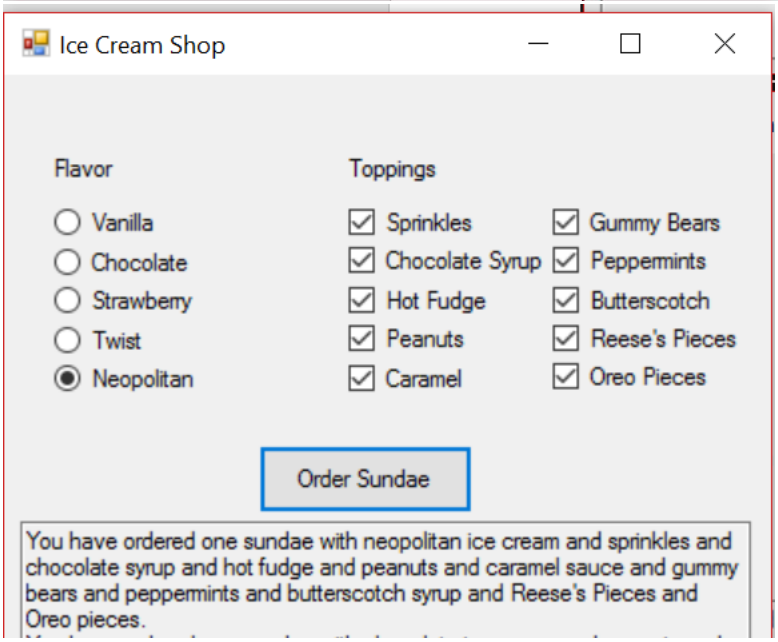


Jeremy Martin


Decorator Pattern




## Conclusion

Overall, this was a very interesting pattern to make an application for. I had so many ideas that would have fit with this pattern, but in the end decided to write the application on ice cream, this is why you shouldn't code when you are hungry.