Design Patterns

Decorator Pattern

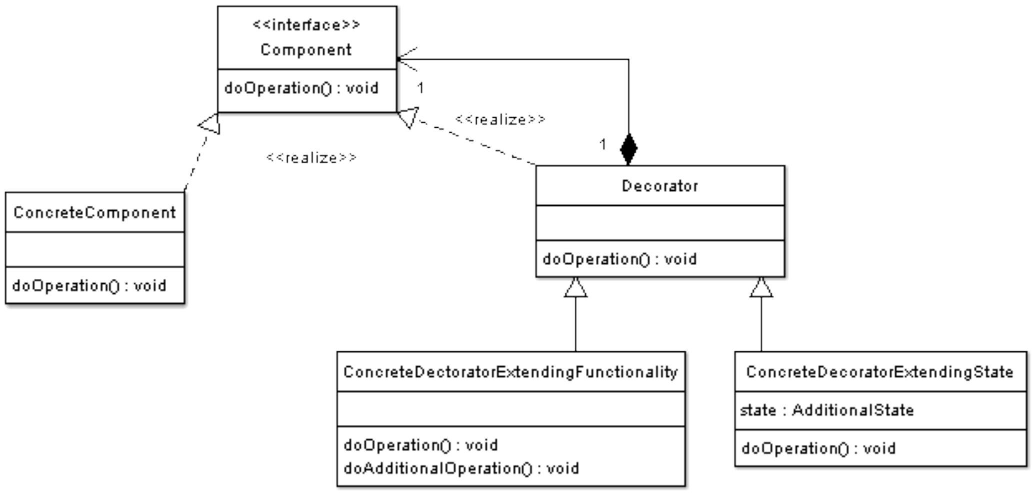
Collin Kemner

November 17, 2016

<u>Introduction</u>

The purpose of this assignment is to demonstrate the Decorator Pattern by looking at its code and using an application that uses the pattern. In this particular assignment, I created a pizza maker. The app can create not only the pizza, but it also shows what the price of the particular pizza is.

<u>UML Diagram</u>



According to OODesign.com, the Decorator Pattern "add(s) additional responsibilities dynamically to an object." The Component interface is an interface that holds objects with responsibilities while the ConcreteComponent defines an object that the responsibilities can be added to. The Decorator maintains a reference to a component object and defines an interface similar to the Component's while the ConcreteDecorator extend the functionality of the component by adding a state or behavior. In the case of this application, the Component and ConcreteComponent work together to create the pizza itself while the Decorator and ConcreteDecorator gets the price of the pizza that is selected.

<u>Application Code: Component Interface</u>

```
public interface Component
    {
        string displayPizza();
    }
```

The Component simply contains a string method "displayPizza", which will display the physical contents of the pizza that is created.

ConcreteComponent Class

```
public class ConcreteComponent : Component
    {
        public string _crustType;
        public string _topping1;
        public string _topping2;
        public string _topping3;

        public ConcreteComponent(string crustType, string Topping1, string Topping2,
string Topping3)
        {
            this._crustType = crustType;
            this._topping1 = Topping1;
            this._topping2 = Topping2;
            this._topping3 = Topping3;

            if (_topping1 != null)
            {
                _topping1 = _topping1 + " ";
            }
            if (_topping2 != null)
            {
                _topping2 = _topping2 + " ";
            }
            if (_topping3 != null)
            {
                _topping3 = _topping3 + " ";
            }
        }

        public string displayPizza()
        {
            return "You have selected to consume a " + _topping1 + _topping2 + _topping3
+ "pizza with " + _crustType + ".";
        }
    }
```

The ConcreteComponent starts off by creating three strings for three toppings that will be sent from the Form as well as the crust type, which will also be sent from the Form. These toppings and the crust type are sent into the ConcreteComponent constructor so that the values are all set. Beneath the initializations, there are if statements that are checking the strings for the toppings are there, and if they are, to add a space next to the name of the topping itself. Lastly, the string method “displayPizza” is at the bottom to return a statement containing the information about the makeup of the pizza.

Decorator Class

```
public class Decorator : Component
    {
        protected Component _comp;

        public Decorator(Component comp)
        {
            this._comp = comp;
        }

        public string displayPizza()
        {
            return _comp.displayPizza();
        }

        public string totalPrice(double
       crustPrice, double toppingPrice)
        {
            double totalprice = crustPrice + toppingPrice;
            string priceString = totalprice.ToString();
            return "The price of your newly created pizza is: $" + priceString;
        }
    }
```

The Decorator class extends the functionality of the Component and also maintains a reference to the interface. With help from the Decorator constructor, the public string method "displayPizza" returns the value from the "displayPizza" method from the Component. Below that is the "totalPrice" string method, which is what adds more functionality. The purpose of the method is to return the price of the pizza. In a restaurant, knowing the price of a pizza is a good bit of information, so it seemed like it was necessary to add.

ConcreteDecorator Class

```
public class ConcreteDecorator : Decorator
    {
        public ConcreteDecorator(Component _comp)
            : base(_comp)
        {

        }

        public string displayPizza()
        {
            return _comp.displayPizza();
        }
        public string totalPrice(double crustPrice, double toppingPrice)
        {
            double totalprice = crustPrice + toppingPrice;
            string priceString = totalprice.ToString();
            return "The price of your newly created pizza is: $" + priceString;
        }
    }
```

The ConcreteDecorator class contains a base which is the Component object (so that the reference will remain) and the same methods from the Decorator class, however, these will be the methods that the Form will call.

Form1 Code

```
public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        #region Variable Initializations
        double BaconPrice = 4.09;
        double HamPrice = 4.19;
        double SausagePrice = 5.00;

        double GreenPeppersPrice = 2.50;
        double BananaPeppersPrice = 2.50;
        double PineapplePrice = 3.00;
        double GreenOlivesPrice = 3.09;
        double GarlicPrice = 2.40;
        double MushroomsPrice = 4.00;

        double ThinCrustPrice = 6.00;
        double ThickCrustPrice = 5.50;
        double FlatbreadPrice = 6.50;
        double FocacciaPrice = 7.30;

        double meatPrice;
        double veggiePt1Price;
        double veggiePt2Price;
        double crustPrice;
        #endregion

        List<string> Toppings = new List<string>()
        {
            "", "", "", ""
        };
```

Lastly in the app is the Form code. As the Form is quite large, it will be explained in separate chunks. Firstly, in the code is variable initializations. There are price numbers for the toppings and crust (the prices are not supposed to be accurate). There is then a list entitiled "Toppings". This list will contain the toppings of the pizza in it so that sending the strings over to the ConcreteComponent will be a little easier.

```
        #region Create Pizza event
        private void m_btnCreatePizza_Click(object sender, EventArgs e)
        {
            string meatTopping;
            string veggiePt1Topping;
            string veggiePt2Topping;
            string crustType;

            double totalToppingPrice;

            if (m_rbBacon.Checked == true)  //meat topping set up
            {
                meatTopping = m_rbBacon.Text;
                Toppings[0] = meatTopping;
                meatPrice = BaconPrice;
            }
```

Next is the button event for creating the pizza. A few strings are initialized for when a radio button is selected. For example, in the if-else statement for Ham topping, if the radio button for it has been checked, then the “meatTopping” string from above is set to the text in the radio button (which is Ham) and that string is then set to the first indice of the list. Then, the “meatPrice” string from above is set to the price of the Ham (“HamPrice”). This same process goes on for the other topping groups and the crusts.

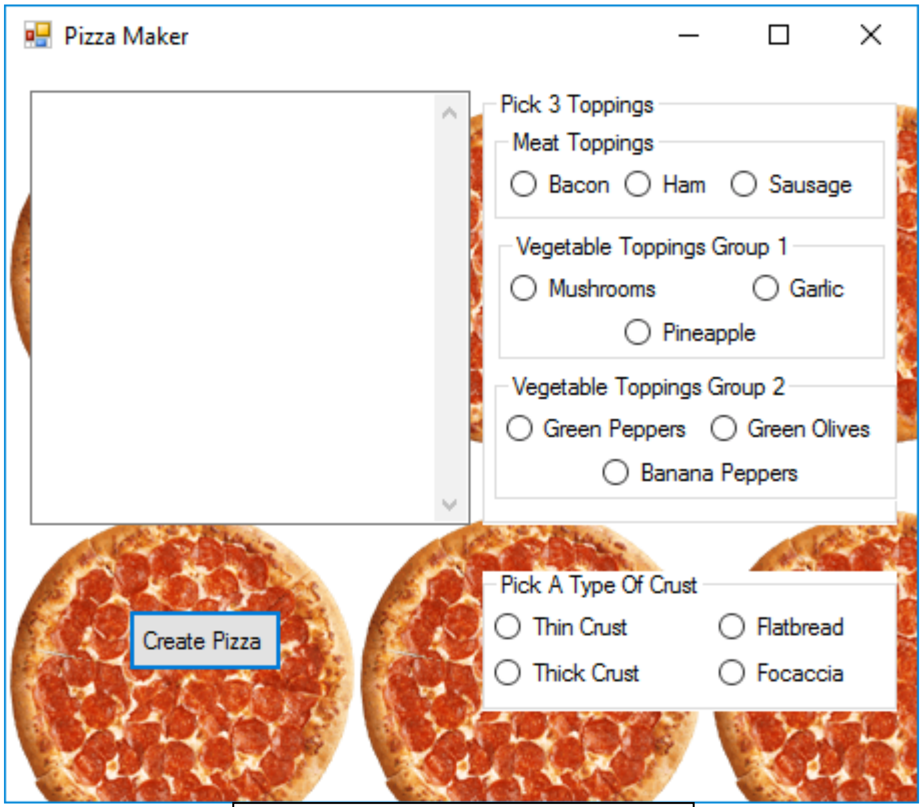
```
else if (m_rbHam.Checked == true)
{
    meatTopping = m_rbHam.Text;
    meatTopping = Toppings[0];
    meatPrice = HamPrice;
}
else if (m_rbSausage.Checked == true)
{
    meatTopping = m_rbSausage.Text;
    Toppings[0] = meatTopping;
    meatPrice = SausagePrice;
}

if (m_rbMushrooms.Checked == true)   //veggies group 1 set up
{
    veggiePt1Topping = m_rbMushrooms.Text;
    Toppings[1] = veggiePt1Topping;
    veggiePt1Price = MushroomsPrice;
}
else if (m_rbGarlic.Checked == true)
{
    veggiePt1Topping = m_rbMushrooms.Text;
    Toppings[1] = veggiePt1Topping;
    veggiePt1Price = GarlicPrice;
}
else if (m_rbPineapple.Checked == true)
{
    veggiePt1Topping = m_rbPineapple.Text;
    Toppings[1] = veggiePt1Topping;
    veggiePt1Price = PineapplePrice;
}

if (m_rbGreenOlives.Checked == true)    //veggies group 2 set up
{
    veggiePt2Topping = m_rbGreenOlives.Text;
    Toppings[2] = veggiePt2Topping;
    veggiePt2Price = GreenOlivesPrice;
}
else if (m_rbGreenPeppers.Checked == true)
{
    veggiePt2Topping = m_rbGreenPeppers.Text;
    Toppings[2] = veggiePt2Topping;
    veggiePt2Price = GreenPeppersPrice;
}
else if (m_rbBananaPeppers.Checked == true)
{
    veggiePt2Topping = m_rbBananaPeppers.Text;
    Toppings[2] = veggiePt2Topping;
    veggiePt2Price = BananaPeppersPrice;
}

if (m_rbThinCrust.Checked == true)    //crust type set up
{
    crustType = m_rbThinCrust.Text;
    Toppings[3] = crustType;
    crustPrice = ThinCrustPrice;
}
else if (m_rbThickCrust.Checked == true)
```

```
            {
                crustType = m_rbThickCrust.Text;
                Toppings[3] = crustType;
                crustPrice = ThickCrustPrice;
            }
            else if (m_rbFlatbread.Checked == true)
            {
                crustType = m_rbFlatbread.Text;
                Toppings[3] = crustType;
                crustPrice = FlatbreadPrice;
            }
            else if (m_rbFocaccia.Checked == true)
            {
                crustType = m_rbFocaccia.Text;
                Toppings[3] = crustType;
                crustPrice = FocacciaPrice;
            }
```

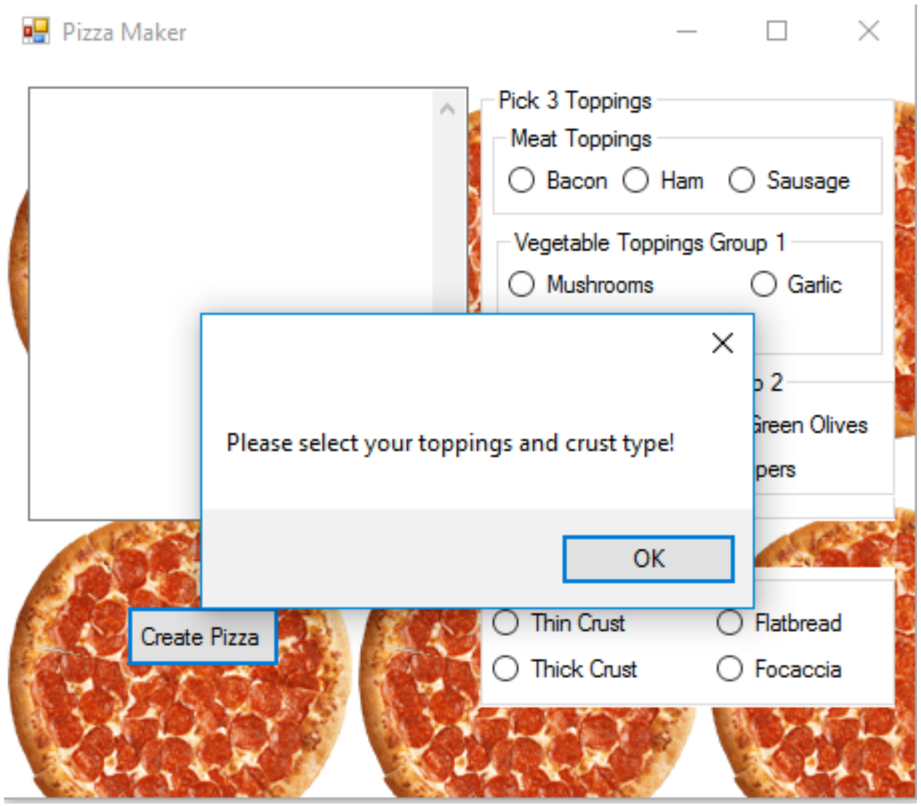
Next in the form is the setting of the “totalToppingPrice” double variable, which adds up all of the toppings to one value. Next, a ConcreteComponent object is created which has the list indices as parameters since they are what makes up the pizza. Then, a ConcreteDecorator object is created with the ConcreteComponent object as its parameter. Next, an if statement checks to see if there are no radio buttons that are checked, and if they are not, then a message box appears telling the user to select a radio button. If there are radio buttons selected, then the pizza information is sent to the textbox on the form.

```
            totalToppingPrice = meatPrice + veggiePt1Price + veggiePt2Price;

            ConcreteComponent conccomp = new ConcreteComponent(Toppings[3], Toppings[0],
Toppings[1], Toppings[2]);
            ConcreteDecorator concdec = new ConcreteDecorator(conccomp);

            if (!m_rbBacon.Checked && !m_rbHam.Checked && !m_rbSausage.Checked &&
!m_rbGreenOlives.Checked && !m_rbGreenPeppers.Checked &&
                !m_rbGarlic.Checked && !m_rbBananaPeppers.Checked &&
!m_rbMushrooms.Checked && !m_rbPineapple.Checked &&
                !m_rbThinCrust.Checked && !m_rbThickCrust.Checked &&
!m_rbFlatbread.Checked && !m_rbFocaccia.Checked)
            {
                MessageBox.Show("Please select your toppings and crust type!");
            }
            else
            {
                m_tbDisplayPizza.Text += conccomp.displayPizza() +
System.Environment.NewLine;
                m_tbDisplayPizza.Text += concdec.totalPrice(crustPrice,
totalToppingPrice);
                m_tbDisplayPizza.Text += System.Environment.NewLine +
System.Environment.NewLine;
            }
        }
        #endregion
    }
```

Screenshots of Application Running

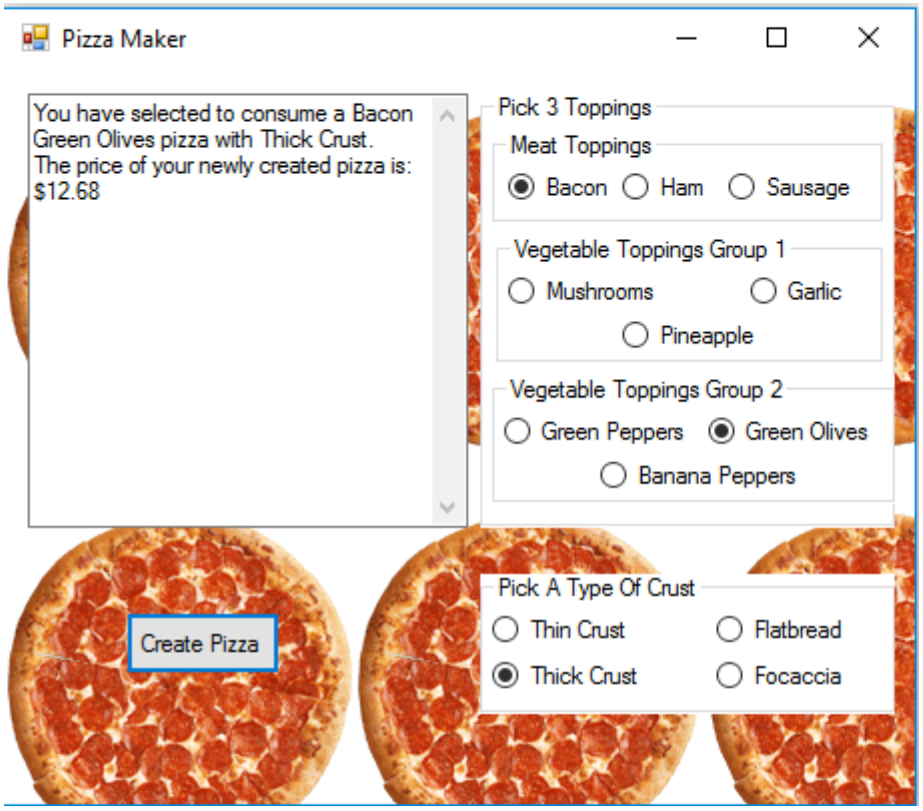


Application when first opened.



Please select your toppings and crust type!

No buttons pressed when Create Pizza button is pressed.



```
You have selected to consume a Bacon
Green Olives pizza with Thick Crust.
The price of your newly created pizza is:
$12.68
```

Pick 3 Toppings

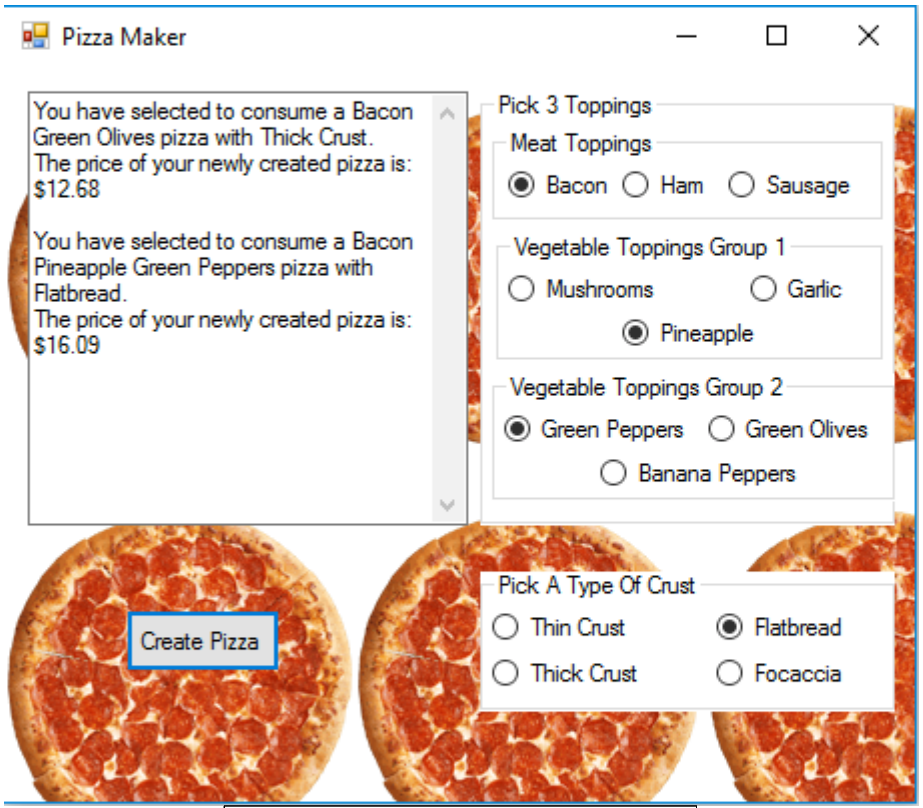
Meat Toppings: [✓ Bacon] Ham Sausage

Vegetable Toppings Group 1: Mushrooms Garlic Pineapple

Vegetable Toppings Group 2: Green Peppers [✓ Green Olives] Banana Peppers

Pick A Type Of Crust: Thin Crust Flatbread [✓ Thick Crust] Focaccia

Bacon Green Olives with Thick Curst pizza.



```
You have selected to consume a Bacon
Green Olives pizza with Thick Crust.
The price of your newly created pizza is:
$12.68

You have selected to consume a Bacon
Pineapple Green Peppers pizza with
Flatbread.
The price of your newly created pizza is:
$16.09
```

Pick 3 Toppings

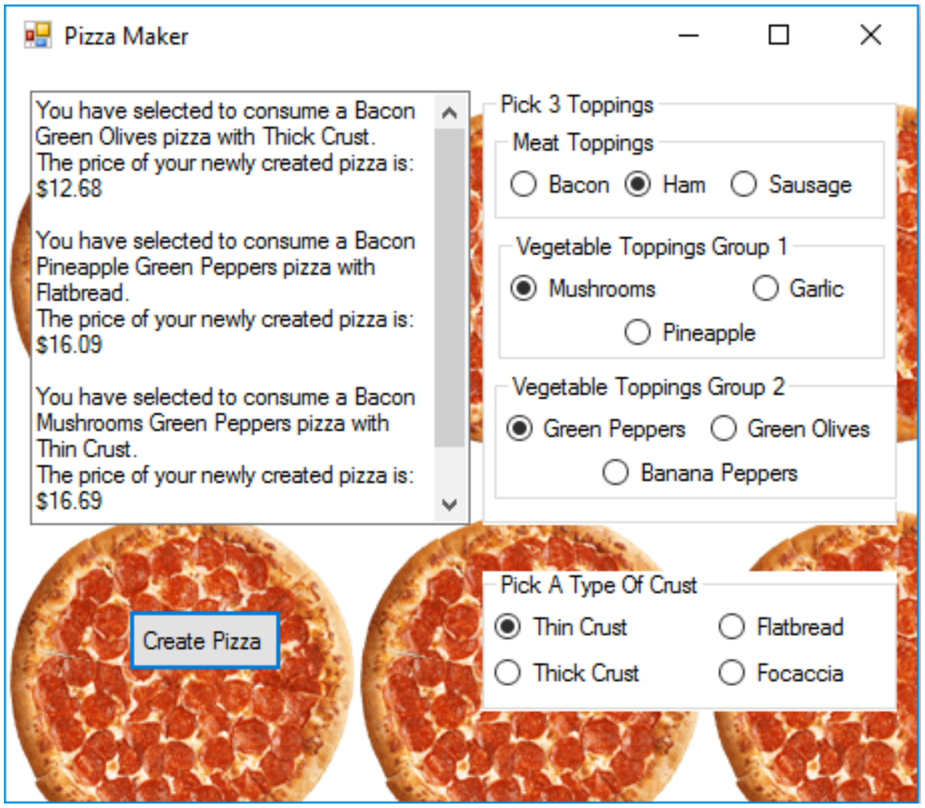
Meat Toppings: [✓ Bacon] Ham Sausage

Vegetable Toppings Group 1: Mushrooms Garlic [✓ Pineapple]

Vegetable Toppings Group 2: [✓ Green Peppers] Green Olives Banana Peppers

Pick A Type Of Crust: Thin Crust [✓ Flatbread] Thick Crust Focaccia

Bacon Pineapple Green Peppers with Flatbread pizza.



Pizza Maker

```
You have selected to consume a Bacon
Green Olives pizza with Thick Crust.
The price of your newly created pizza is:
$12.68

You have selected to consume a Bacon
Pineapple Green Peppers pizza with
Flatbread.
The price of your newly created pizza is:
$16.09

You have selected to consume a Bacon
Mushrooms Green Peppers pizza with
Thin Crust.
The price of your newly created pizza is:
$16.69
```

Pick 3 Toppings

Meat Toppings: Bacon [✓ Ham] Sausage

Vegetable Toppings Group 1: [✓ Mushrooms] Garlic Pineapple

Vegetable Toppings Group 2: [✓ Green Peppers] Green Olives Banana Peppers

Pick A Type Of Crust: [✓ Thin Crust] Flatbread Thick Crust Focaccia

Ham Mushrooms Green Peppers with Thin Crust pizza.

Observations and Reflections

This pattern was fun for me to do once I figured out what to do. I was having some trouble at first figuring out how to check if any of the buttons are selected, but once I figured that out, I sped through the code. This is a different application for me as I have a background picture for it! I'm hoping that I did do the pattern right as I see in the UML diagram that the interface needs a void method rather than a string. So, I'm hoping that doesn't change anything as I'm feeling pretty good about how I did.