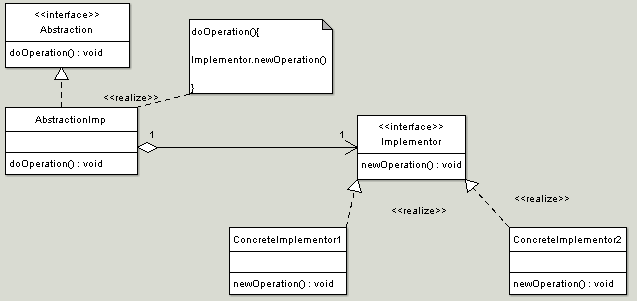
Bridge Pattern

Design Patterns

Christopher Doseck

11/9/2016

Introduction

This assignment is an application that I created to show how the bridge pattern works. In this application I am using warming up food to represent the bridge pattern. I use microwaves and toasters to heat foods.

The UML Diagram for Bridge

The UML Diagram for the bridge pattern, shown on the right, shows the classes that are needed to have the requirements. The Food interface represents the Abstraction interface and ConcreteFood represents AbstractionImp. The Heater interface represents the Implementer, and the Toaster and Microwave implement the Heater like the ConcreteImplementers. The table below shows how all of the classes were used.

|  |  |
| --- | --- |
| Food | This is the interface that represents the Abstraction interface. |
| ConcreteFood | This is the class that represents the AbstractionImp class. It inherits from Food. |
| Heater | This interface represents the Implementer interface. |
| Microwave | This class is one of the ConcreteImplementer classes. It inherits from the Heater interface. |
| Toaster | This class is one of the ConcreteImplementer classes. It inherits from the Heater interface. |
| Form1 | This is the client that shows the bridge pattern in action |

Narrative

This is the interface for Heating up the food.

public interface Heater

{

void heat(string food);

}

public class Microwave : Heater

This is the microwave class. It inherits from the Heater interface. It shows a message saying that it microwaved the food.

{

public void heat(string food)

{

MessageBox.Show("Enjoy your microwaved " + food);

}

}

This is the toaster class. It inherits from the Heater interface. It shows a message saying that it toasted the food.

public class Toaster : Heater

{

public void heat(string food)

{

MessageBox.Show("Enjoy your wonderfully toasted " + food);

}

}

public interface Food

This is the Food interface.

{

void warm(Heater heater);

}

public class ConcreteFood : Food

This is the ConcreteFood class. It inherits from the Food interface. In the constructor, it is passed the name of the food. The warm function is passed a heater object, in which it calls the heater.heat() method.

{

private string name;

public ConcreteFood(string name)

{

this.name = name;

}

public void warm(Heater heater)

{

heater.heat(name);

}

}

public partial class Form1 : Form

{

Food food;

This is the Form. It contains a Food object.

public Form1()

{

InitializeComponent();

When the toaster button is clicked, it creates a new Concrete food with the textbox’s text, and then calls the food.warm() method with a new Toaster object.

}

private void toastBtn\_Click(object sender, EventArgs e)

{

food = new ConcreteFood(tbFood.Text);

food.warm(new Toaster());

}

private void microwaveBtn\_Click(object sender, EventArgs e)

When the microwave button is clicked, it creates a new Concrete food with the textbox’s text, and then calls the food.warm() method with a new Microwave object.

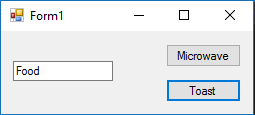
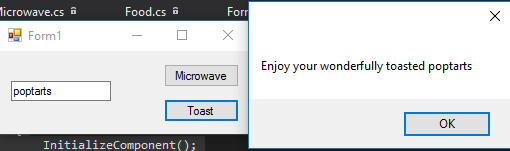
{

food = new ConcreteFood(tbFood.Text);

food.warm(new Microwave());

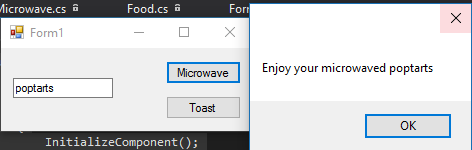
}

}



Poptarts entered into the textbox, and toast is pressed.

Initial setup



Poptarts entered into microwave, and microwave is pressed.

Conclusion

I did not see how this pattern is useful. I ended up doing a very simple program because I could not think of an application for it. It took me a while to understand how the pattern worked, but I think that I figured it out.