Template Method Pattern

Design Patterns

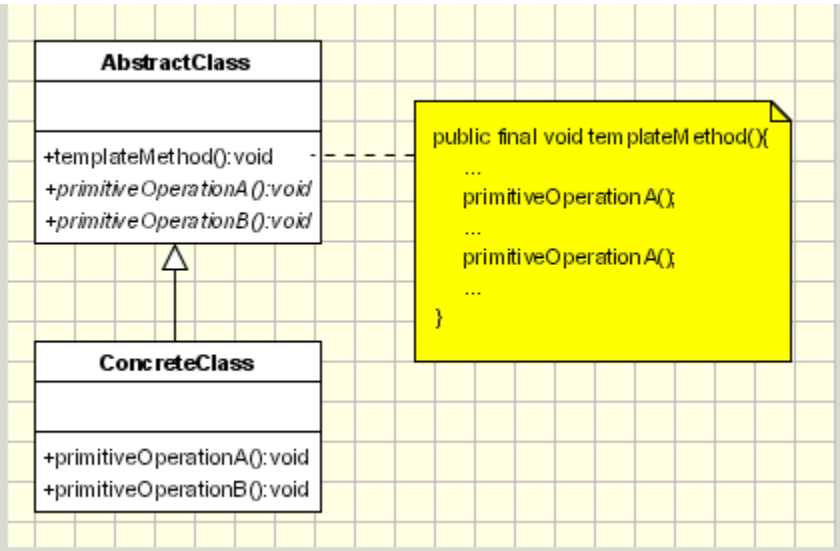
Christopher Doseck

12/2/2016

Introduction

This assignment is an application that I created to show how the template method pattern works. In this application I am using food as a demonstration. Each of the different types of food are the different concrete classes.

The UML Diagram for Template Method



The UML Diagram for the template method pattern, shown on the right, shows the classes that are needed to have the requirements. The AbstractClass for this application is the abstract Food class. The ConcreteClass is implemented by the Pasta, Sandwich, and Soup classes. The table below shows how all of the classes were used.

Food	This is the AbstractClass. All of the ConcreteClasses derive from it.
Pasta, Sandwich, Soup	These are the ConcreteClasses that are children of the AbstractClass.
Form1	This is the client that shows the template method pattern in action.

Narrative

This is the abstract food class. The name is used to hold the specific name of the food. The eat() method is the templateMethod() method from the UML diagram. It uses abstract methods which have not yet been defined when it is called.

```
public abstract class Food
{
    protected string name;
    public string eat()
    {
        return "You eat using" + spoon() + fork() + knife() + hands() + ".";
    }

    public abstract string fork();
```

```
    public abstract string spoon();
    public abstract string knife();
    public abstract string hands();
}
```

These are the abstract methods that will be defined in the child classes.

```
public class Sandwich : Food
{
    public Sandwich(string name)
    {
        this.name = name;
    }
```

This is the concrete Sandwich class which inherits from the Food class. In its constructor, it is passed a name, which is used to make the food more specific.

```
    public override string fork()
    {
        return null;
    }
```

These are the methods that the concrete class defines from the abstract Food class. It returns how the utensils are used, and if they are not used, it returns null.

```
    public override string hands()
    {
        return " your hands to bring the " + name + " sandwich closer to your mouth";
    }

    public override string knife()
    {
        return " a knife to cut the " + name + " sandwich in half so it is easier to eat,
and";
    }

    public override string spoon()
    {
        return null;
    }
}
```

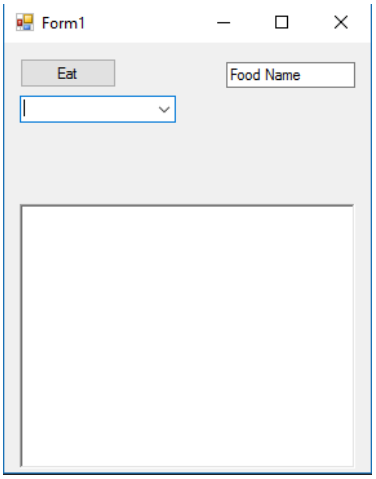
The rest of the concrete classes are not included because they are not necessary.

```
public partial class Form1 : Form
{
    private string foodName;
    private Food food;
    public Form1()
    {
        InitializeComponent();
    }
```

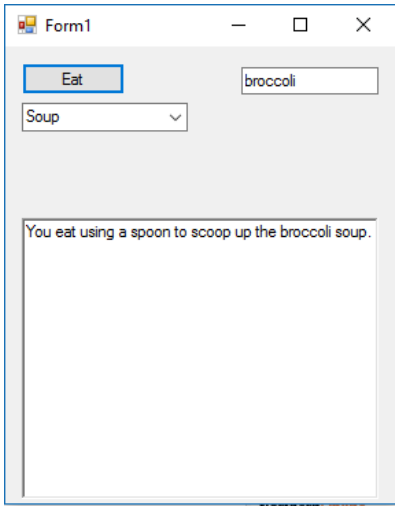
This is the form to display the template method pattern. It has a Food reference within it.

When the eat button is pressed, it creates an object of one of the children of the Food class with the name that is in tbFoodName and assigns the string that the eat() method returns to a rich text box.

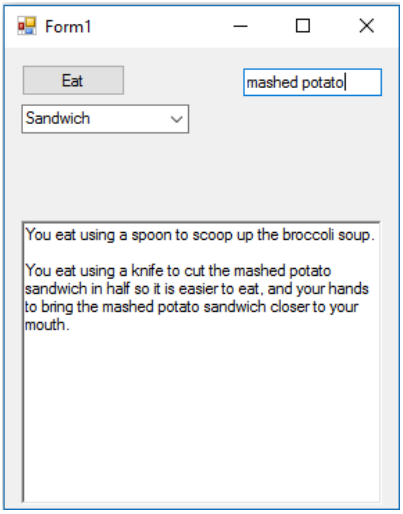
```
    private void btnEat_Click(object sender, EventArgs e)
    {
        foodName = tbFoodName.Text;
        if (cbFoods.Text == "Soup")
            food = new Soup(foodName);
        else if (cbFoods.Text == "Sandwich")
            food = new Sandwich(foodName);
        else
            food = new Pasta(foodName);
        rtbSummary.Text += food.eat() + "\n";
    }
}
```



Initial setup



Brocoli Soup is added.



Mashed Potato Sandwich is added.

Observations

Overall this was a really easy design pattern. There is no part of it that I struggled with. I can see how it is useful to setup a skeleton of an algorithm, letting some steps be defined in child classes. It allows for there to be variations on the algorithm without changing the structure.