Alex Lang

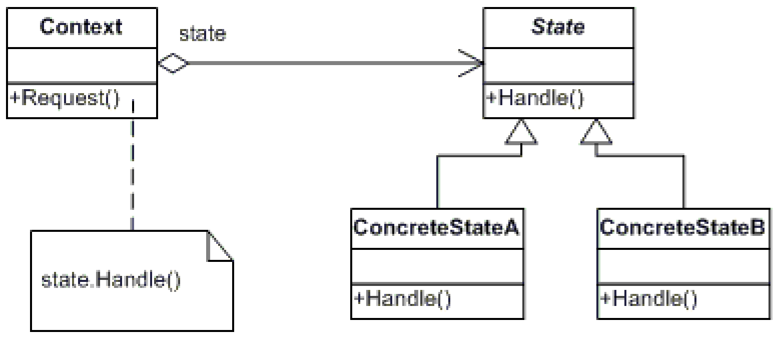
The State Pattern

Design Patterns

11/29/16

For this assignment, we were told to do the State Pattern. The State Pattern implements each individual state as a derived class of the state pattern interface, and uses common methods to alter some object.

Here is the UML diagram from DoFactory on what the Decorator Pattern consists of and its implementation.



For my application, I decided to create a factory, with a machine that has three different states, an on, off and maintenance state. For my abstract State class, I created an abstract MachineState class, and made OnState, OffState and RepairState my concrete state classes.

The first class I created was the MachineState class. For my application, when the state of the machine is changed, a string is updated that displays the current state of the machine.

```
public abstract class MachineState
    {
        public abstract string changeState();
    }
```

This abstract class contains the changeState method, which for this app is the only method each class contains.

Next, I created the concrete state class, which implement MachineState

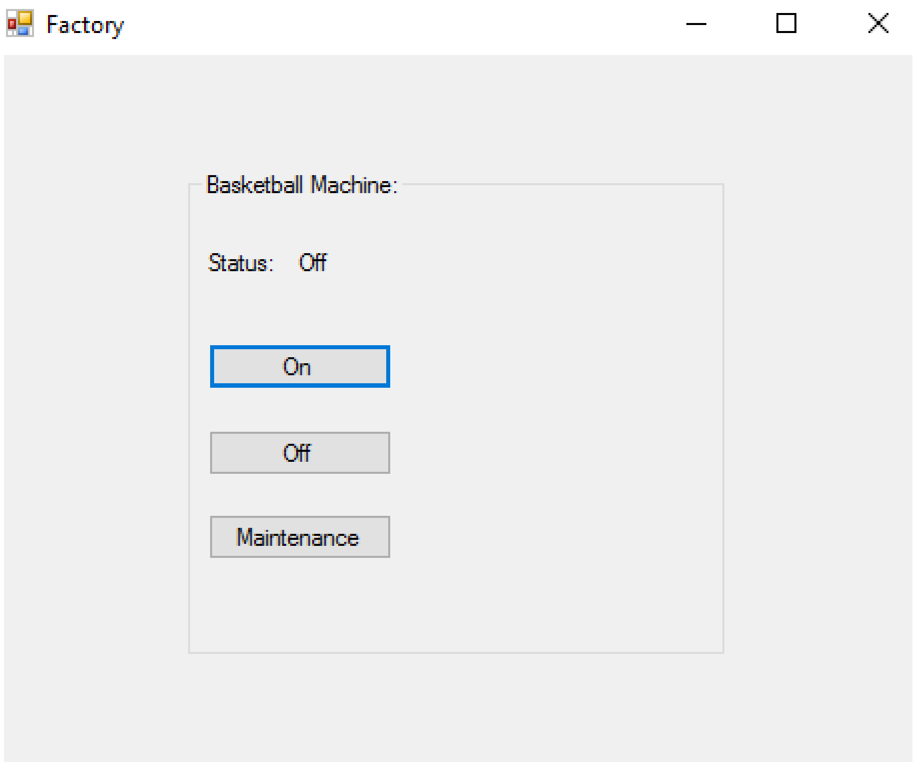
```
public class OnState : MachineState
    {
        public override string changeState()
        {
            return "On";
        }
    }

public class OffState : MachineState
    {
        public override string changeState()
        {
            return "Off";
        }
    }

public class RepairState : MachineState
    {
        public override string changeState()
        {
            return "Machine under maintenance";
        }
    }
```

Each of these methods here return a string to match the state that it represents the machine is in.

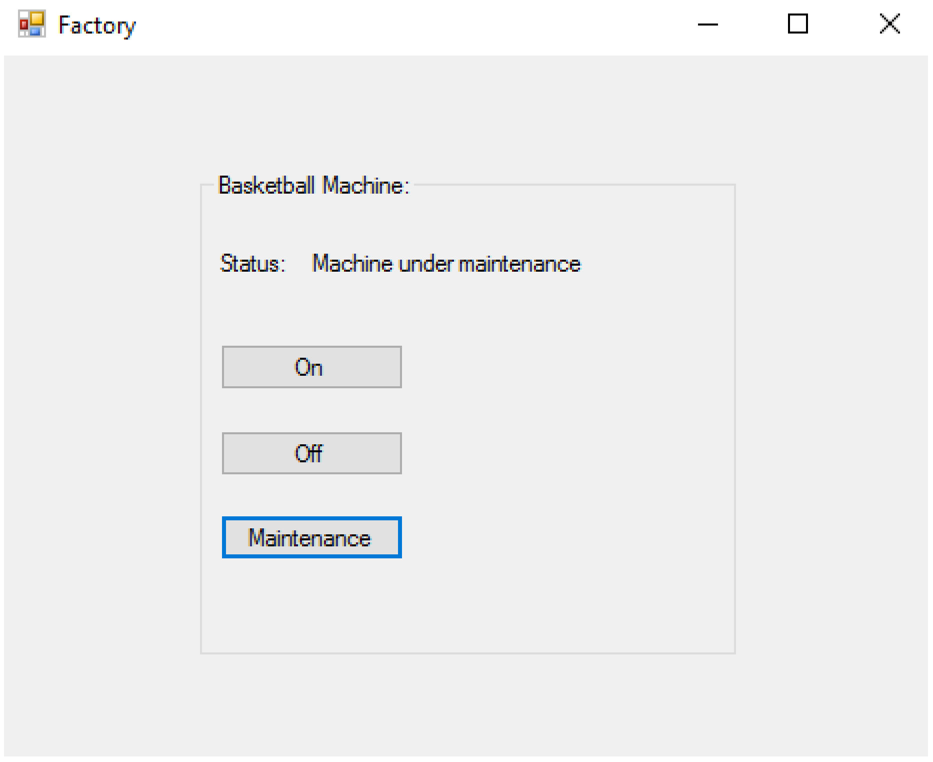
Finally, I put everything together in the main form.

```
public partial class Form1 : Form
    {
        MachineState machine;

        public Form1()
        {
            InitializeComponent();
        }

        private void bOnBtn_Click(object sender, EventArgs e)
        {
            machine = new OnState();
            bballStatLbl.Text = machine.changeState();
        }

        private void bOffBtn_Click(object sender, EventArgs e)
        {
            machine = new OffState();
            bballStatLbl.Text = machine.changeState();
        }

        private void bMaitBtn_Click(object sender, EventArgs e)
        {
            machine = new RepairState();
            bballStatLbl.Text = machine.changeState();
        }
    }
```

For this form, I first created a abstract machine object. When the state buttons are clicked, the machine object becomes a machine object but with the corresponding state. After the button is pressed, the string representing the state is returned and is updated to the label.



Status: Off



Status: Machine under maintenance

Conclusion: I believe that this pattern is by far one of the easiest to understand and write an application on. As soon as I heard what this design pattern did, I knew exactly wanted to create and I feel that this application, while being extremely simple, does a good job demonstrating the state pattern.