Tori Lentz

Dave Retterer

Design Patterns

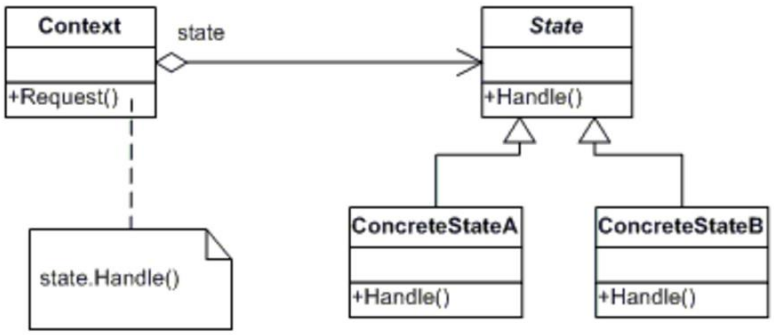
November 27, 2016

State Pattern

Introduction

The purpose of this assignment was to design and implement the state pattern. The state pattern is a very interesting pattern that allows an object to alter its behavior when something in its internal state changes. The object will then appear to change its class.

UML Diagram



The state pattern is made up of several different classes including context, state, and concrete states. The context class defines the interface that will be used by the other classes. Along with these components it also maintains an instance of one of the concrete state subclasses and it uses this as its default state. The state class would be either an abstract class or an interface and it defines an interface for encapsulating the behavior that is associated with one of the states of the context class. Finally, are the concrete classes. These classes are subclasses that each implement a different behavior that is associated with the state of the context class.

Code and Description

State

```
// The 'State' abstract class
public abstract class State
{
    protected StudentGrades grade;
    protected double rawgradescore;

    protected double bonuspoints;
    protected double lowerLimit;
    protected double upperLimit;
```

Here is my state class which is abstract and within it, all of my methods are defined. I also create an object for my context class which I can then use in my methods as well as my subclasses.

```
        // Properties
        public StudentGrades Grades
        {
            get { return grade; }
            set { grade = value; }
        }

        public double RawGradeScore
        {
            get { return rawgradescore; }
            set { rawgradescore = value; }
        }

        public abstract string PostGrade(double amount, string assignment);
    }
```

Context

```
    // The 'Context' class
    public class StudentGrades
    {
        private State _state;
        private string _student;
        private string _quiz;
        private string _midterm;
        private string _final;


        // Constructor
        public StudentGrades(string student)
        {
            // New grades are 'B' by default
            this._student = student;
            this._state = new BStudentState(0.0, this);
        }

        // Properties
        public double RawGradeScore
        {
            get { return _state.RawGradeScore; }
        }

        public State State
        {
            get { return _state; }
            set { _state = value; }
        }

        public string Quiz
        {
            get { return _quiz; }
            set { _quiz = value; }
        }

        public string Midterm
        {
            get { return _midterm; }
            set { _midterm = value; }
```

This is my context class which expands upon the methods that were defined in my state class. Just as I make an object for my context class in my state class, I do the same here with the state class and make an object for it that I used in different methods. The post grade method is where I output all of the information that the other methods get, which I then call in my form through a textbox. The raw grade score is what determines which state the context will be in. This is a grader program so, depending on how high a point score one gets, determines which grade they receive. In other words, this determines if the student will be in the A, B or C state.

```
        }

        public string Final
        {
            get { return _final; }
            set { _final = value; }
        }

        public string PostGrade(double amount, string assignment)
        {
            _state.PostGrade(amount, assignment);
            return "PostGrade: " + amount.ToString() + "\r\n" + " Assignment: " +
assignment.ToString() + "\r\n" + " RawGradeScore = " + this.RawGradeScore + "\r\n" + "
GradeStatus = " + this.State.GetType().Name + "\r\n";
        }
    }
```

Concrete State A

```
    // A 'ConcreteState' class
    class AStudentState : State
    {
        // Overloaded constructors
        public AStudentState(State state)
          : this(state.RawGradeScore, state.Grades)
        {
        }

        public AStudentState(double rawgradescore, StudentGrades grade)
        {
            this.rawgradescore = rawgradescore;
            this.grade = grade;
            Initialize();
        }

        private void Initialize()
        {
            if (grade.Quiz == "Y")
            {
                lowerLimit = 0;
                upperLimit = 0;
            }
            else if (grade.Midterm == "Y")
            {
                lowerLimit = 0;
                upperLimit = 0;
            }
            else if (grade.Final == "Y")
            {
                lowerLimit = 0;
                upperLimit = 0;
            }
        }

        public override string PostGrade(double amount, string assignment)
        {
            rawgradescore += amount;
            return StateChangeCheck();
```

My first concrete state class is the A Student state. There are three different grades to be entered so, depending on the grade the student gets on these assignments, determines which state the context class will be in. Each grade will concatenate onto the previous score and so, if the student starts with a C, then they can still end up with an A, as long as they get high scores on the midterm and final.

```
        }

        private string StateChangeCheck()
        {
            if (grade.Quiz == "Y")
            {
                lowerLimit = 3.01;
                upperLimit = 4.0;
            }
            else if (grade.Midterm == "Y")
            {
                lowerLimit = 6.01;
                upperLimit = 8.0;
            }
            else if (grade.Final == "Y")
            {
                lowerLimit = 9.01;
                upperLimit = 12.0;
            }


            if (rawgradescore < lowerLimit/2)
            {
                grade.State = new CStudentState(this);

            }
            else if (rawgradescore < lowerLimit)
            {
                grade.State = new BStudentState(this);

            }
            return grade.State.ToString();
        }
    }
```

Concrete State B

```
   // A 'ConcreteState' class
    class BStudentState : State
    {
        // Overloaded constructors
        public BStudentState(State state) :
          this(state.RawGradeScore, state.Grades)
        {
        }

        public BStudentState(double rawgradescore, StudentGrades grade)
        {
            this.rawgradescore = rawgradescore;
            this.grade = grade;
            Initialize();
        }

        private void Initialize()
        {
            if (grade.Quiz == "Y")
            {
                lowerLimit = 0;
```

This class looks almost exactly like the A and C state classes, with the exceptions of the point score that the grades fall in and the logic that determines which state the context class should be in.

```
            upperLimit = 0;
        }
        else if (grade.Midterm == "Y")
        {
            lowerLimit = 0;
            upperLimit = 0;
        }
        else if (grade.Final == "Y")
        {
            lowerLimit = 0;
            upperLimit = 0;
        }
    }

    public override string PostGrade(double amount, string assignment)
    {
        rawgradescore += amount;
        return StateChangeCheck();
    }

    private string StateChangeCheck()
    {
        if (grade.Quiz == "Y")
        {
            lowerLimit = 2.01;
            upperLimit = 3.0;
        }
        else if (grade.Midterm == "Y")
        {
            lowerLimit = 4.01;
            upperLimit = 6.0;
        }
        else if (grade.Final == "Y")
        {
            lowerLimit = 6.01;
            upperLimit = 9.0;
        }


        if (rawgradescore < lowerLimit)
        {
            grade.State = new CStudentState(this);
        }
        else if (   rawgradescore > upperLimit)
        {
            grade.State = new AStudentState(this);
        }
        return grade.State.ToString();
    }
}
```

Concrete State C

```
// A 'ConcreteState' class
class CStudentState : State
{
    // Constructor
    public CStudentState(State state)
```

```
        {
            this.rawgradescore = state.RawGradeScore;
            this.grade = state.Grades;
            Initialize();
        }

        private void Initialize()
        {
            if (grade.Quiz == "Y")
            {
                lowerLimit = 0;
                upperLimit = 0;
            }
            else if (grade.Midterm == "Y")
            {
                lowerLimit = 0;
                upperLimit = 0;
            }
            else if (grade.Final == "Y")
            {
                lowerLimit = 0;
                upperLimit = 0;
            }
        }

        public override string PostGrade(double amount, string assignment)
        {
            rawgradescore += amount;
            return StateChangeCheck();
        }

        private string StateChangeCheck()
        {
            if (grade.Quiz == "Y")
            {
                lowerLimit = 1.01;
                upperLimit = 2.0;
            }
            else if (grade.Midterm == "Y")
            {
                lowerLimit = 2.01;
                upperLimit = 4.0;
            }
            else if (grade.Final == "Y")
            {
                lowerLimit = 3.01;
                upperLimit = 6.0;
            }


            if (rawgradescore > upperLimit)
            {
                grade.State = new BStudentState(this);
            }
            return grade.State.ToString();
        }
    }
```

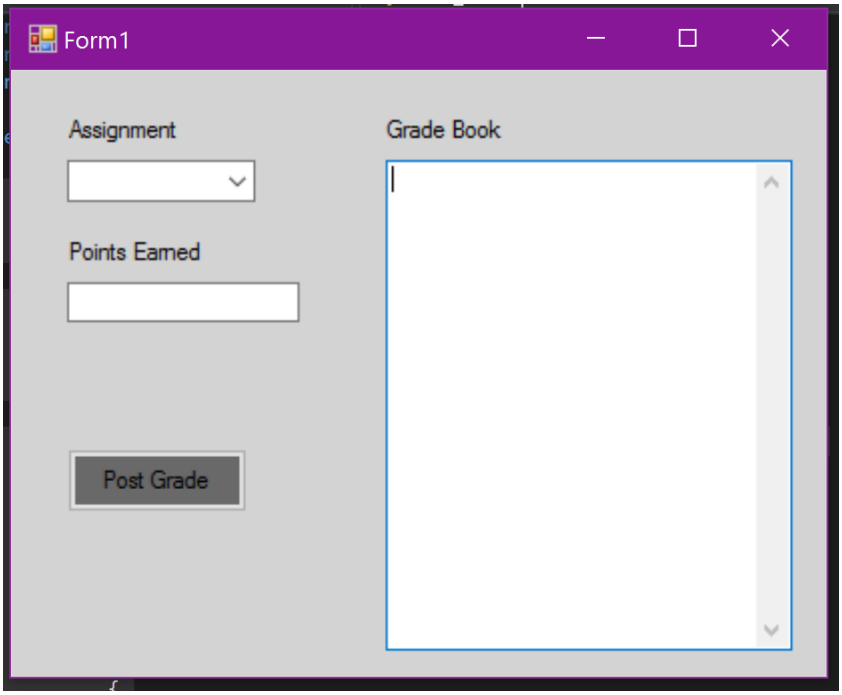
Form

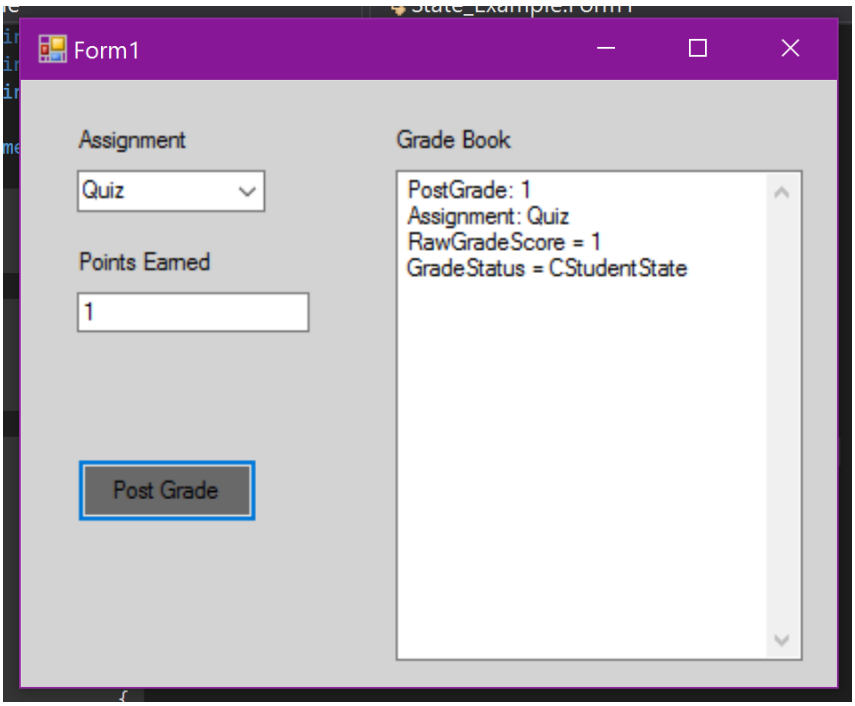
```
    public partial class Form1 : Form
    {
        StudentGrades grade = new StudentGrades("Jim Johnson");

        public Form1()
        {
            InitializeComponent();
        }

        private void btn_GradeBookEntry_Click(object sender, EventArgs e)
        {
            if (cb_Assignments.Text == "Quiz" )
            {
                grade.Quiz = "Y";
                grade.Midterm = "N";
                grade.Final = "N";
            }
            else if (cb_Assignments.Text == "Midterm")
            {
                grade.Quiz = "N";
                grade.Midterm = "Y";
                grade.Final = "N";
            }
            else if (cb_Assignments.Text == "Final")
            {
                grade.Quiz = "N";
                grade.Midterm = "N";
                grade.Final = "Y";
            }
            tb_GradeBook.Text = tb_GradeBook.Text + " " +
grade.PostGrade(Convert.ToDouble(tb_Grade.Text), cb_Assignments.Text) + "\r\n";
        }
    }
```

Within my form, I have a combobox. Depending on which item is selected, determines the code that will be executed. If I want to enter the score for quiz, then the others will be set to false so I can execute the right calls. After all of this logic, I output the information to my textbox.

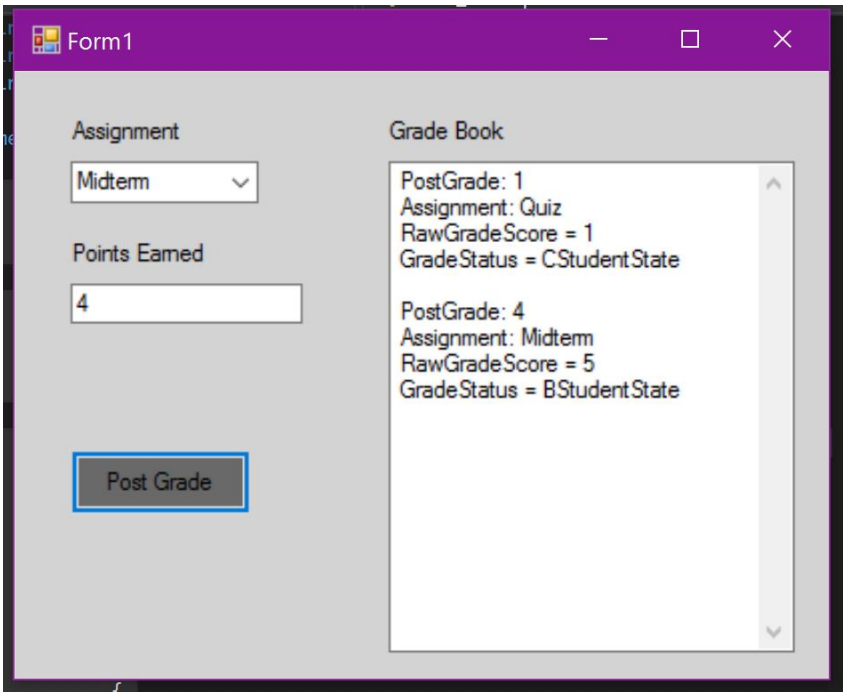
Screen Shots

Form on load

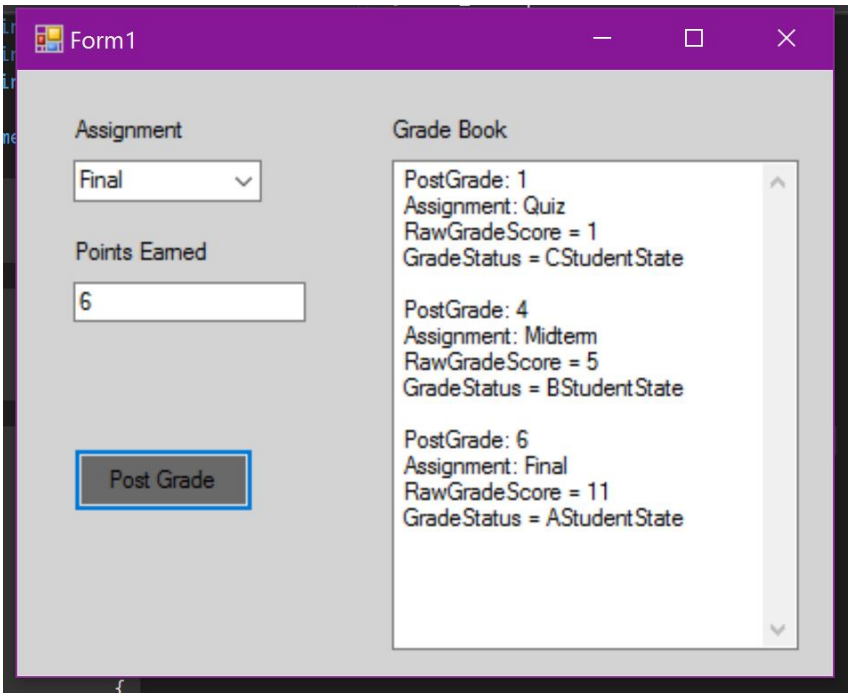




Form after Quiz score entered



Form after Midterm and Quiz entered



Form after Quiz, Midterm and the Final have been entered.

Observations

This assignment went very well and I thought it was a very interesting pattern to do. I thought it was interesting to see how this pattern would change its state depending on only a few different things. I had some issues with getting my combobox to call the right code but, I was finally able to figure out what it was doing and therefore was able to fix it. Once this problem was resolved, I ran into another one which had to do with getting my calls to print to a textbox. Again, I had to step through my code to understand what was wrong. After a few short fixes, however, I was able to get my information to print out correctly and all the state changes worked.