State Pattern Assignment

Design Patterns

Jacob Hartman

29 November 2016

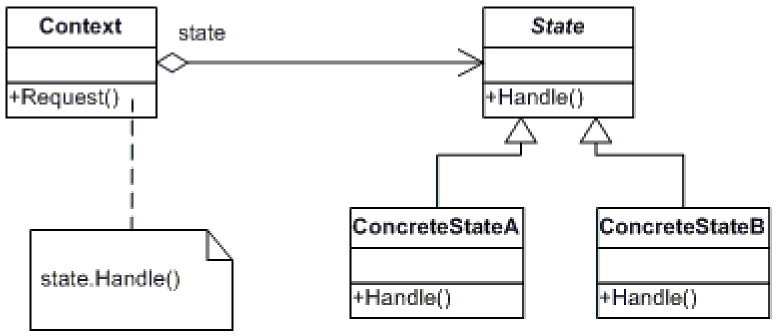
Introduction

This assignment requires an application using the state pattern to be created. The state pattern is used to allow an object's methods to change depending on the circumstances.

The application I wrote for this assignment involves the chemicals that living cells use to store energy. It provides the user with uncharged molecules and allows them to charge the molecules or discharge them after charging.

UML Diagram

image taken from http://www.dofactory.com/net/state-design-pattern



Above is the UML Diagram for the State pattern. It shows the need for four classes. The state class is abstract and has a public handle method. ConcreteStateA and ConcreteStateB implement the state class and its handle method. The context class contains a state object and a public request method that calls the state object's handle method.

My Context class is called Cell. It has a request method called getState, and a List of State objects. For my State class, I wrote the Molecule class and its isEnergized method. My ConcreteState classes are ATP and ADP.

Narrative

I began with the Molecule abstract class.

```
public abstract class Molecule // this is the State class
{
    public abstract bool isEnergized(); // this is the handle method
}
```

The handle method is defined to return a boolean value. Next, I wrote the ATP and ADP classes.

```
public class ATP: Molecule // this is the ConcreteStateA class
{
    public override bool isEnergized() // this is the handle method
    {
        return true;
    }
}

public class ADP : Molecule // this is the ConcreteStateB class
{
    public override bool isEnergized() // this is the handle method
    {
        return false;
    }
}
```

The purpose of the two classes in this program is simply to store a boolean value that can be accessed by the rest of the application. The isEnergized method returns this stored boolean value. Next, I wrote the Cell class.

```
public class Cell
{
    List<Molecule> energy;
    public Cell()
    {
        energy = new List<Molecule>();
        for (int i = 0; i < 10; i++)
        {
            energy.Add(new ADP());
        }
    }

    public String getState(int index) // this is the request method
    {
        if (energy[index].isEnergized())
        {
            return "ATP";
        }
        else
        {
            return "ADP";
        }
    }

    public void setState(int index, bool newState)
    {
        if (newState)
        {
            energy[index] = new ATP();
        }
```

```
            else
            {
                energy[index] = new ADP();
            }
        }
    }
```
The Cell class contains a list of molecules called energy. The list is initially populated with ten ADP molecules, representing the ground state of an example cell. This means there is no energy currently stored in the cell. The getState method checks the molecule at the specified index. A name for the molecule is returned based on whether or not it has energy. The setState method allows the state of a particular molecule to be modified without exposing the underlying structure. A true value charges the molecule, while a false value removes the charge. The rest of the code is found in the main form.

```
    public partial class Form1 : Form
    {
        int currIndex;
        Cell cell;
        public Form1()
        {
            InitializeComponent();
            cell = new Cell();
            for (int i = 0; i < 10; i++)
            {
                lb_energyList.Items.Add("ADP");
            }
            currIndex = 0;
        }
```
The form initially creates a Cell object and allocates an integer to track the index the user has selected. The list is populated with uncharged molecules.

```
        private void lb_energyList_SelectedIndexChanged(object sender, EventArgs e)
        {
            currIndex = lb_energyList.SelectedIndex;
            updateButtons();
        }
```
When the selected index changes, the current index is updated and the updateButtons method is called.

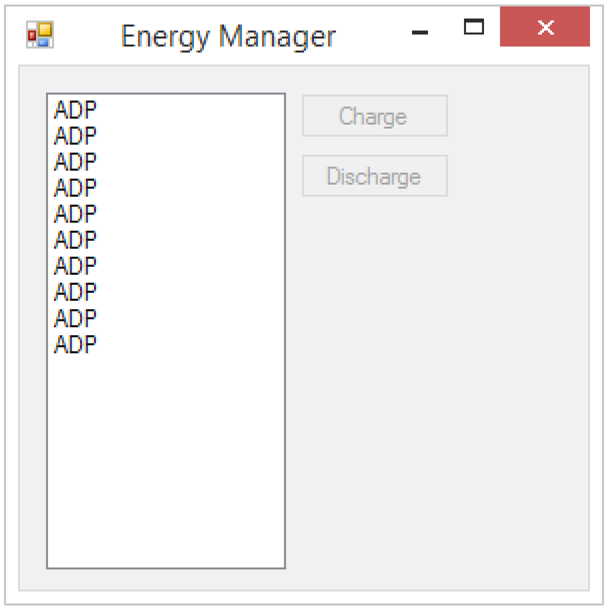
```
        private void btn_charge_Click(object sender, EventArgs e)
        {
            cell.setState(currIndex, true);
            updateList();
            updateButtons();
            lb_energyList.SelectedIndex = currIndex;
        }

        private void btn_discharge_Click(object sender, EventArgs e)
        {
            cell.setState(currIndex, false);
            updateList();
            updateButtons();
            lb_energyList.SelectedIndex = currIndex;
        }
```
When the charge and discharge buttons are clicked, the currently selected molecule is set to the appropriate state and the update methods are called to update the GUI.

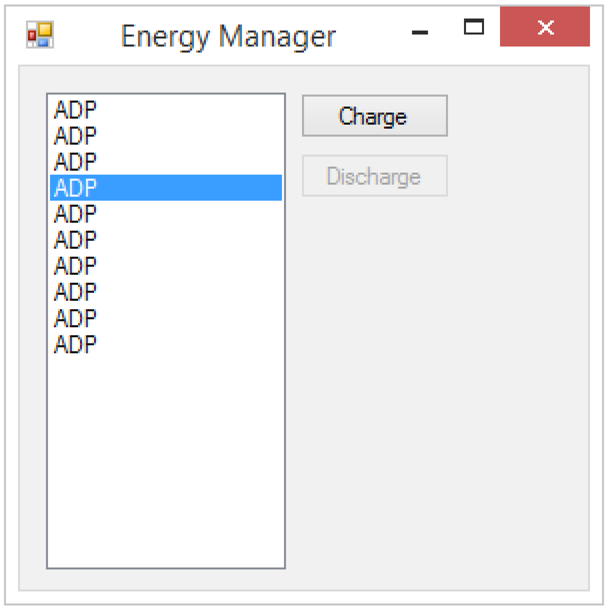
```
        private void updateList()
        {
            lb_energyList.Items.Clear();
            for (int i = 0; i < 10; i++)
            {
                lb_energyList.Items.Add(cell.getState(i));
            }
        }

        private void updateButtons()
        {
            if (cell.getState(currIndex) == "ATP")
            {
                btn_charge.Enabled = false;
                btn_discharge.Enabled = true;
            }
            else
            {
                btn_charge.Enabled = true;
                btn_discharge.Enabled = false;
            }
        }
    }
```

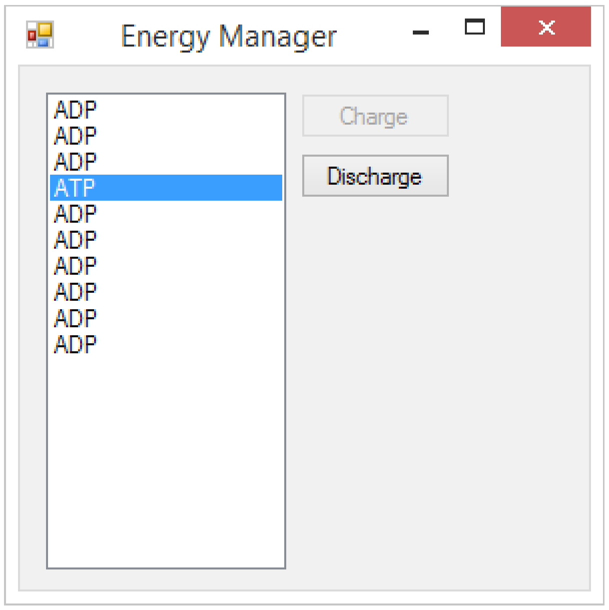
updateList clears the list box and populates it with the new values contained in the cell object. UpdateButtons enables or disables the buttons based on whether or not the selected molecule has charge. These two methods were written separately to avoid unnecessary repetition in the code. Below are some screenshots of the application in use.



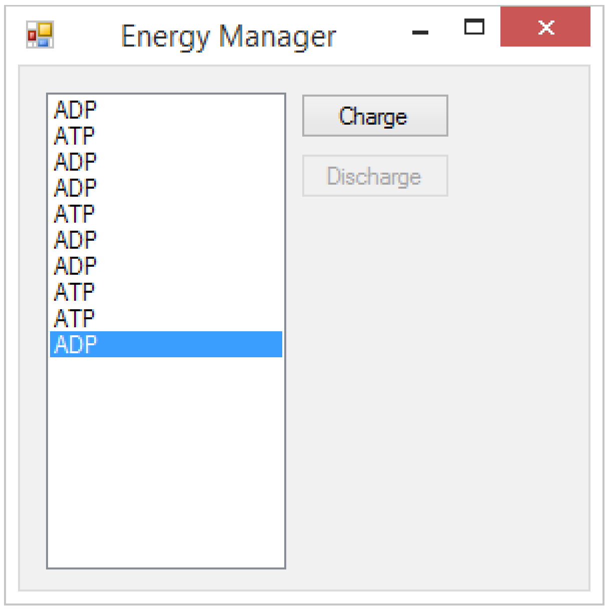
The initial view of the application.



A selected molecule can be charged if it is not.



A charged molecule can be discharged.



Any of the molecules can be altered.

<u>Observations</u>

The state pattern is simple but important. I can see many potential uses for it, far beyond simply storing different values. Entire algorithms could potentially be changed based on the state. I feel that my submission fits the assignment well. It is a very bare-bones demonstration of how the pattern essentially works. With as simple a pattern as this is, I will definitely end up using it again.