Tori Lentz

Dave Retterer

Programming Environments

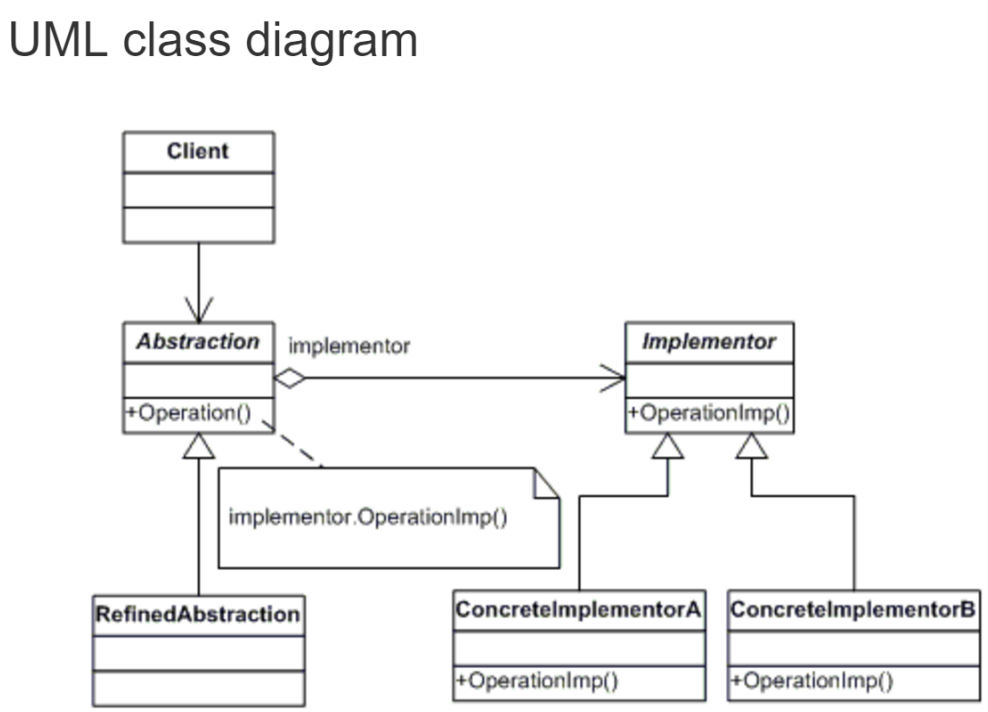
November 8, 2016

Bridge Pattern

Introduction

The purpose of this assignment was to implement the bridge pattern. In some cases people might want to have an abstraction that has different implementations. This pattern is intended to decouple abstraction from implementation so that the two can vary independently.

UML Diagram



Abstraction	This class defines the interface to be used by the abstraction and it also maintains a reference to an Implementor type object.
Refined Abstraction	This class will extend the interface that is defined by the abstraction class.
Implementor	This class defines the interface for all of the implementation classes.
Concrete Implementor	This class implements the implementer interface and defines its concrete implementation.
http://www.dofactory.com/net/bridge-design-pattern	

This whole pattern is made up of five different parts: the client, Abstraction, Implementor, Refined Abstraction, and the Concrete Implementors. The client is the form used by the user where they can call upon the classes and methods defined within those classes. The Refined Abstraction class expands the Implementor and within this class one can create more methods that the user can call upon. The Implementor class can be an Interface or an abstract class and within this class is where all of the methods are initiated for later use. The Abstraction class is where an Implementor object is passed in and it is used to call upon the methods that are defined in the Concrete Implementor class. This Concrete Implementor class is where all of the methods used by the pattern are defined.

Code and Description

Implementor Class

```
abstract class Implementor
{
    public abstract int NextMovie();
    public abstract int PriorMovie();
    public abstract string AddMovie(string name);
    public abstract string ShowMovie();
    public abstract string ShowAllMovies();
    public abstract int FirstMovie();
    public abstract int LastMovie();
    public abstract int MovieCount();
}
```

Here is my Implementor class. It is an abstract class, meaning that it is a class that is closely related to interfaces and are classes that cannot be instantiated. Abstract classes may implement an unlimited number of interfaces but can only inherit from one abstract class. Within this class I have eight different methods that are initiated here so that other classes can use them later on.

Abstraction Class

```
class Abstraction
{
    private Implementor _dataObject;
    protected string group;

    public Abstraction(string group)
    {
        this.group = group;
    }

    public Implementor Genre
    {
        set { _dataObject = value; }
        get { return _dataObject; }
    }

    public virtual int Next()
    {
       return _dataObject.NextMovie();
    }

    public virtual int Prior()
    {
        return _dataObject.PriorMovie();
    }

    public virtual string Add(string movies)
```

Here is the Abstraction class. Within it is an Implementor object that is used as the bridge between the Abstract Implementor and the Concrete Implementor class. Different methods are made for the methods that are to be defined in the Concrete Implementor class and the object calls upon those methods within the new methods.

```
        {
            return _dataObject.AddMovie(movies);
        }

        public virtual string Show()
        {
            return _dataObject.ShowMovie();
        }

        public virtual string ShowAll()
        {
            return _dataObject.ShowAllMovies();
        }

        public virtual int First()
        {
            return _dataObject.FirstMovie();
        }

        public virtual int Last()
        {
            return _dataObject.LastMovie();
        }

        public virtual int Count()
        {
            return _dataObject.MovieCount();
        }
    }
```

Refined Abstraction Class

```
    class RefinedAbstraction : Abstraction
    {
        // Constructor
        public RefinedAbstraction(string group)
          : base(group)
        {
        }

        public override string ShowAll()
        {
            string _MovieList = "";
            _MovieList = base.ShowAll();
            return _MovieList;
        }
    }
```

This is the Refined Abstraction class. Within this class different methods can be defined and used. The reference base is used within this class so that one can call a method that has been overridden by another method.

Concrete Implementor Class

```
    class ConcreteImplementor : Implementor
    {
        public List<string> _movies = new List<string>();
        private int _current = 0;

        public ConcreteImplementor()
        {
            _movies.Add("Howl's Moving Castle");
```

This is the Concrete Implementor class. It inherits from the Implementor class and within it is the main body of the pattern. All of my methods are defined here as well as my list of strings.

```
    _movies.Add("Spirited Away");
    _movies.Add("Star Trek");
    _movies.Add("A Knights Tale");
    _movies.Add("Troy");
    _movies.Add("Stardust");
}

public override int NextMovie()
{
    if (_current <= _movies.Count - 1)
    {
        _current++;
    }
    return _current;
}

public override int PriorMovie()
{
    if (_current > 0)
    {
        _current--;
    }
    return _current;
}

public override string AddMovie(string movies)
{
    _movies.Add(movies);
    return movies;
}

public override string ShowMovie()
{
    return _movies[_current];
}

public override string ShowAllMovies()
{
    string MovieList = "";
    foreach (string movies in _movies)
    {
        MovieList = MovieList + movies + "\r\n";
    }
    return MovieList;
}

public override int FirstMovie()
{
    _current = 0;
    return _current;
}

public override int LastMovie()
{
    _current = _movies.Count - 1;
    return _current;
}
```

My Next Movie method will make it so that the next movie in the list is displayed. My Prior Movie method will display the movie prior to the current movie. Add Movie adds movies to the list that are entered in the textbox. Show All Movies will show the entire list of movies. The First Movie method will display the first movie in the list and the Last Movie method will show the last movie in the list. My Movie Count and Show movie methods are methods that are used internally for other methods and are not ones that will be displayed by form button calls.

```
        public override int MovieCount()
        {
            _current = _movies.Count;
            return _current;
        }
    }
```

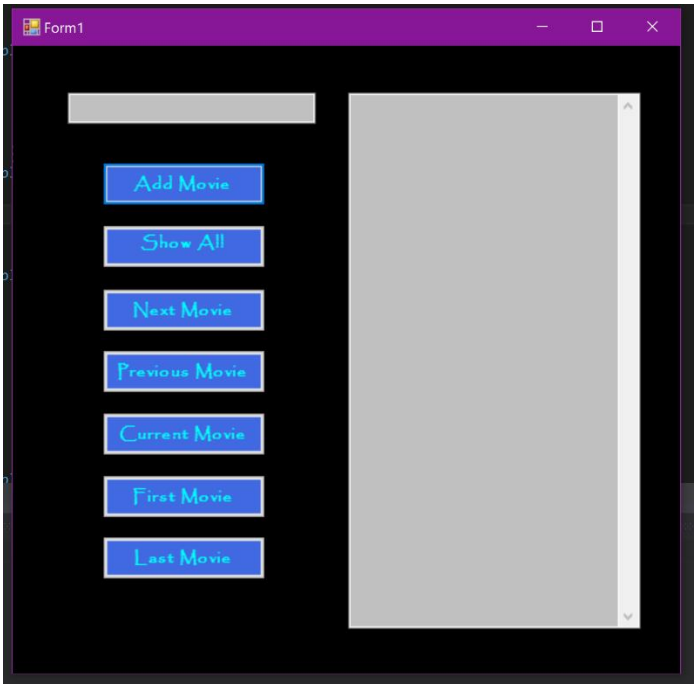
Client

The client for my program is my form. In order to call my methods I used different button calls and two different textboxes to display my results. Within the buttons I called my methods and then using my Show method I displayed the results to the desired textbox.

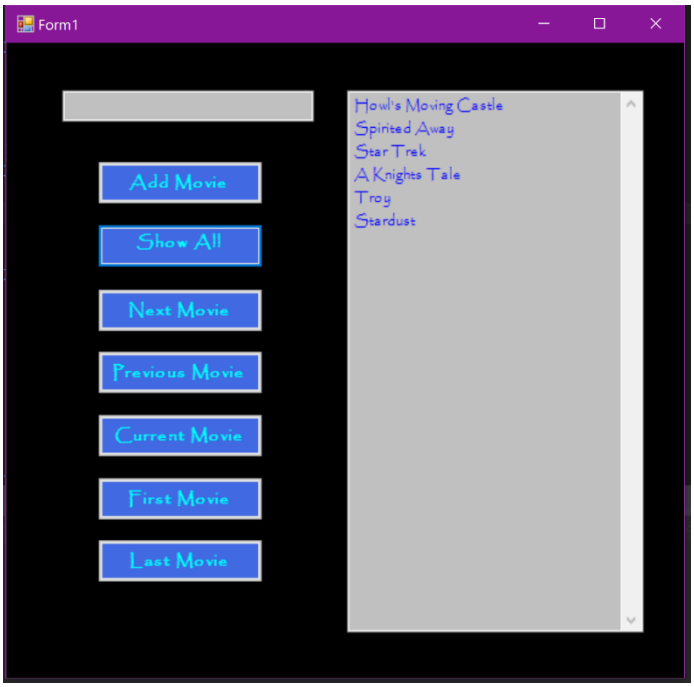
```
    public partial class Form1 : Form
    {

        RefinedAbstraction movies = new
RefinedAbstraction("Fantasy");

        public Form1()
        {
            InitializeComponent();

            movies.Genre = new ConcreteImplementor();
        }

        private void btn_AddMovie_Click(object sender, EventArgs e)
        {
            movies.Add(tb_movie.Text);
            movies.Last();
            tb_movie.Text = movies.Show();
        }

        private void btn_NextMovie_Click(object sender, EventArgs e)
        {
            movies.Next();
            tb_movie.Text = movies.Show();
        }

        private void btn_PreviousMovie_Click(object sender, EventArgs e)
        {
            movies.Prior();
            tb_movie.Text = movies.Show();
        }

        private void btn_ShowAll_Click(object sender, EventArgs e)
        {
            tb_list.Text = movies.ShowAll();
        }

        private void btn_CurrentMovie_Click(object sender, EventArgs e)
        {
            tb_movie.Text =  movies.Show();
        }

        private void btn_First_Click(object sender, EventArgs e)
        {
            movies.First();
            tb_movie.Text = movies.Show();
        }

        private void btn_Last_Click(object sender, EventArgs e)
```

```
        {
            movies.Last();
            tb_movie.Text = movies.Show();
        }
    }
```

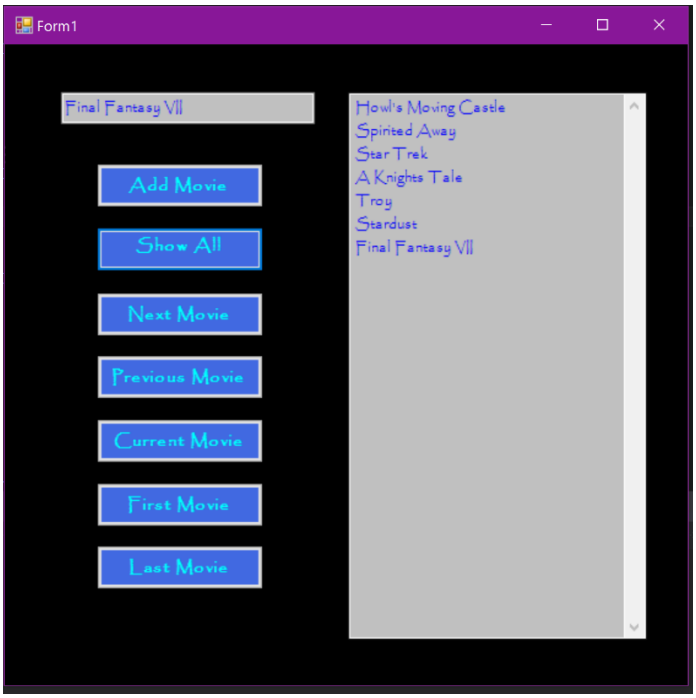
Screen Shots



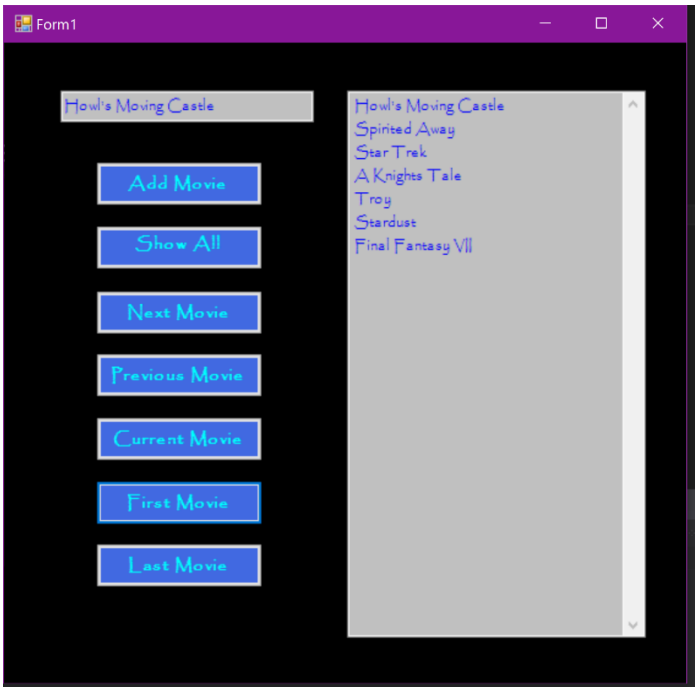
This is the form when upon initiation.



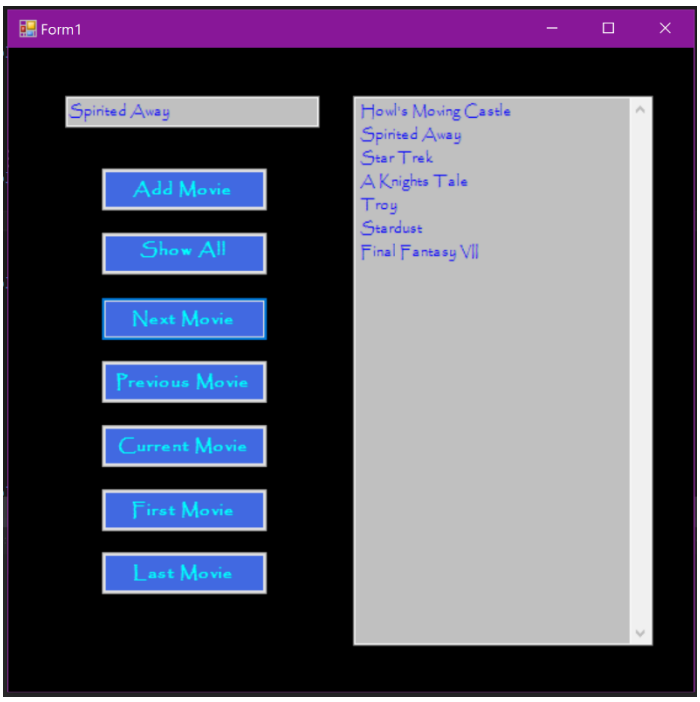
This is the form after the Show All button has been clicked.



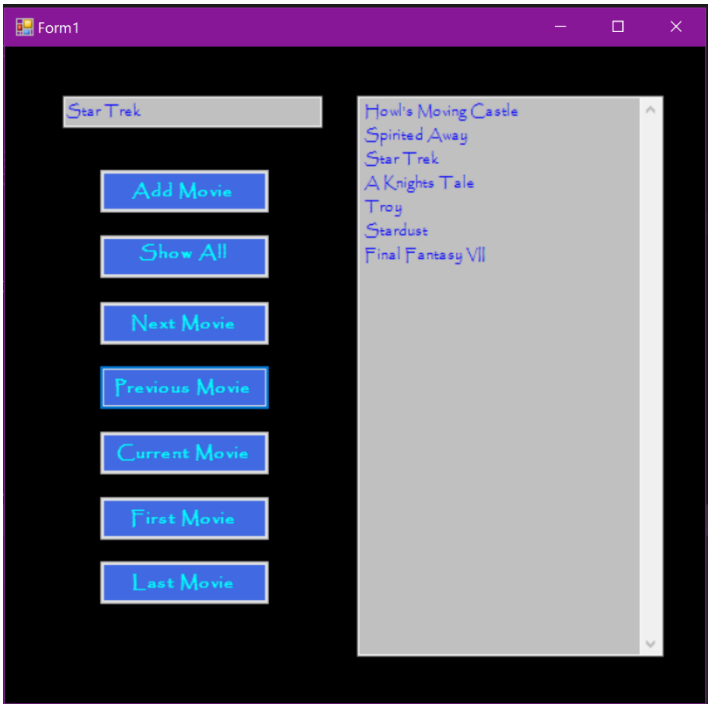
This is the form after the Add Movie button has been clicked.



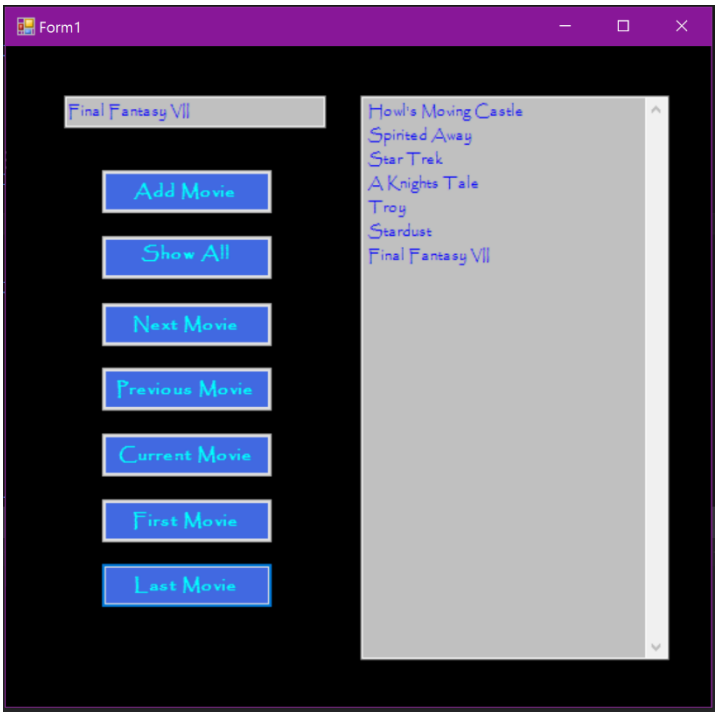
This is the form after the First Movie button has been clicked.



This is the form after the Next Movie button is clicked.



This is the form after the Previous Movie button is clicked.



This is the form after the Last Movie button is clicked.

Observations

Overall I thought this pattern went well. It took me a little while to wrap my head around how it needs to work and why but, once I was able to figure it out everything went smoothly. I wasn't really sure what to do for this assignment in the beginning and just ended up doing a movie list program. I know this pattern can be used for a lot more complicated applications but I couldn't think of what to use it for. I would like to know more about it and see more examples of it so that I can further know how it can be applied.