Tori Lentz

Dave Retterer

Design Patterns

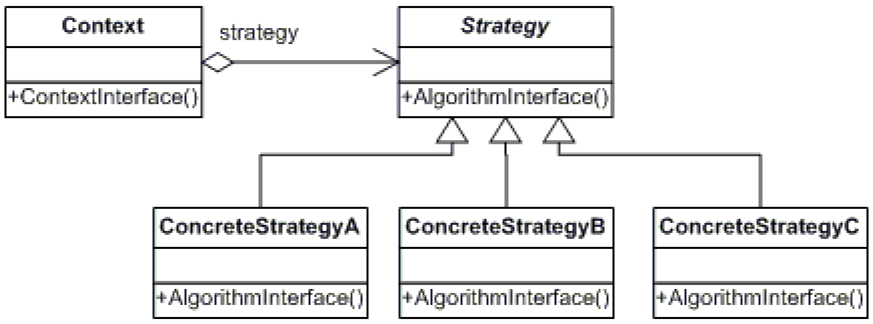
October 19, 2016

Strategy Pattern

Introduction

The goal of this assignment was to successfully implement the Strategy Pattern. The purpose of this pattern is to make it so one can switch between different classes that contain different algorithms (via a method override). The classes themselves are the same but, how they behave is different. After implementing this pattern, the user will be able to select a different strategy at runtime.

The UML Diagram



Strategy	Declares the interface that will be used by all of the classes and objects. The context class uses the interface to call the algorithm defined in the concrete strategy class.
Concrete Strategy	Implements the algorithm defined in the concrete strategy classes using the Strategy class interface.
Context	Contains the objects. Configured with the concrete strategy object, maintains a reference to a strategy class object and it may define an interface that will allow the strategy class to access its data.
http://www.dofactory.com/net/strategy-design-pattern	

The UML diagram displayed above shows the necessary components for this pattern. The Strategy class contains the method Algorithm interface which defines the interface to be used by the different strategy classes, and the context class. The three Concrete Strategy classes are where one overrides the defined method with the different behaviors the user can implement. Within the Context class are the different objects that will be used by the previously mentioned classes.

Code and Description

Strategy Class

```
    public abstract class SortMethods      //This is my Strategy Class
    {
        public abstract void Sort(colorAttribute[] colorList);
    }
```

Above is my Strategy class called SortMethods. It is an abstract class because it is only meant to be used as a base class for other classes, therefor it is left as an incomplete implementation.

Context Class

```
    public class colorAttribute     //This is my Context Class
    {
        public string Name { get; set; }
        public string Brightness { get; set; }
        public string ArgbValue { get; set; }

        private SortMethods _sortmethods;

        public void SetColorSortMethod(SortMethods sortmethods)
        {
             this._sortmethods = sortmethods;
        }

        public void Sort(colorAttribute[] _colorList)
        {
             _sortmethods.Sort(_colorList);

             Console.WriteLine("Executed Sort");
        }
    }
```

To the left is my Context class. Listed within this class are the get set methods needed for my enumerated list, as well as my two methods. The first method, SetColorSortMethod is used to determine which of the three Concrete Strategy classes will be used and then the Sort method implements the different sort strategies.

Concrete Strategy A

```
    public class SortByName : SortMethods   //This is my ConcreteStrategyA
    {
        public override void Sort(colorAttribute[] colorList)
        {
            Global.gv_DisplayData = "";
            IEnumerable<colorAttribute> query1 = colorList.OrderBy(colorattribute
=> colorattribute.Name);
            string paddingstr = "";

            foreach (colorAttribute colorattribute in query1)
            {
                if (colorattribute.Name.ToString().Length <= 8)
                {
                    paddingstr = "\t";
                }
                Global.gv_DisplayData = Global.gv_DisplayData + " Name: " +
colorattribute.Name.ToString() + ";" + paddingstr + "\t \t Brightness: " +
colorattribute.Brightness.ToString() + ";\t \t \t ArgbValue: " +
colorattribute.ArgbValue.ToString() + "\r\n";
                              paddingstr = "";
            }
        }
    }
```

Above is my Concrete Strategy A Pattern. This class inherits the methods of the SortMethods class and overrides that method with its own strategy. This method will sort the values in the enumerated list by their name.

Concrete Strategy B

```
    public class SortByBrightness : SortMethods   //This is my ConcreteStrategyB
    {
        public override void Sort(colorAttribute[] colorList)
        {
            Global.gv_DisplayData = "";
            IEnumerable<colorAttribute> query2 = colorList.OrderBy(colorattribute
=> colorattribute.Brightness);
            string paddingstr = "";

            foreach (colorAttribute colorattribute in query2)
            {
                if (colorattribute.Name.ToString().Length <= 8)
                {
                    paddingstr = "\t";
                }
                Global.gv_DisplayData = Global.gv_DisplayData + " Name: " +
colorattribute.Name.ToString() + ";" + paddingstr + "\t \t Brightness: " +
colorattribute.Brightness.ToString() + ";\t \t \t ArgbValue: " +
colorattribute.ArgbValue.ToString() + "\r\n";
                paddingstr = "";
            }
        }
    }
```

Above is my Concrete Strategy B Pattern. This class inherits the methods of the SortMethods class and overrides that method with its own strategy. This method will sort the values in the enumerated list by their Brightness Values.

Concrete Strategy C

```
    public class SortByArgbValue : SortMethods   //This is my ConcreteStrategyC
    {
        public override void Sort(colorAttribute[] colorList)
        {
            Global.gv_DisplayData = "";
            IEnumerable<colorAttribute> query3 = colorList.OrderBy(colorattribute
=> colorattribute.ArgbValue);
            string paddingstr = "";

            foreach (colorAttribute colorattribute in query3)
            {
                if (colorattribute.Name.ToString().Length <= 8)
                {
                    paddingstr = "\t";
                }
                Global.gv_DisplayData = Global.gv_DisplayData + " Name: " +
colorattribute.Name.ToString() + ";" + paddingstr + "\t \t Brightness: " +
colorattribute.Brightness.ToString() + ";\t \t \t ArgbValue: " +
colorattribute.ArgbValue.ToString() + "\r\n";
                paddingstr = "";
            }
        }
    }
```

Above is my Concrete Strategy C Pattern. This class inherits the methods of the SortMethods class and overrides that method with its own strategy. This method will sort the values in the enumerated list by their Argb Values.

Form

```
    public partial class Form1 : Form
    {
        colorAttribute colorattribute = new colorAttribute();

        //public static string gv_DisplayData = "";
        colorAttribute[] colorList = { new colorAttribute {Name="Red",
Brightness="0.5", ArgbValue="255,255,000,000" },
                                       new colorAttribute {Name="Blue",
Brightness="0.5", ArgbValue="255,000,000,255" },
                                       new colorAttribute {Name="Green",
Brightness="0.2509804", ArgbValue="255,000,255,000" },
                                       new colorAttribute {Name="Orange",
Brightness="0.5", ArgbValue="255,255,165,000" },
                                       new colorAttribute {Name="Purple",
Brightness="0.2509804", ArgbValue="255,128,000,128" },
                                       new colorAttribute {Name="SlateBlue",
Brightness="0.5784314", ArgbValue="255,106,090,205" },
                                       new colorAttribute {Name="Yellow",
Brightness="0.5", ArgbValue="255,255,255,000" },
                                       new colorAttribute {Name="Violet",
Brightness="0.7215686", ArgbValue="255,238,130,238" },
                                       new colorAttribute {Name="Magenta",
Brightness="0.5", ArgbValue="255,255,000,255" } };
```

Above is where I defined my color list, enumerated list and it contains all of the different values needed for the different sorts.

```
public Form1()
        {
            InitializeComponent();
        }

        private void btn_SortByArgbValue_Click(object sender, EventArgs e)
        {

            textBox5.Text = "";

            colorattribute.SetColorSortMethod(new SortByArgbValue());
            colorattribute.Sort(colorList);
            textBox5.Text = "Sorted query3 by Argb Value \r\n";
            textBox5.Text = textBox5.Text + Global.gv_DisplayData;
        }

        private void btn_SortByBrightness_Click(object sender, EventArgs e)
        {
            textBox5.Text = "";

            colorattribute.SetColorSortMethod(new SortByBrightness());
            colorattribute.Sort(colorList);
            textBox5.Text = "Sorted query2 by Brightness \r\n";
            textBox5.Text = textBox5.Text + Global.gv_DisplayData;
        }

        private void btn_SorByName_Click(object sender, EventArgs e)
        {
            textBox5.Text = "";

            colorattribute.SetColorSortMethod(new SortByName());
            colorattribute.Sort(colorList);
            textBox5.Text = "Sorted query1 by Name \r\n";
            textBox5.Text = textBox5.Text + Global.gv_DisplayData;
        }

    }
```

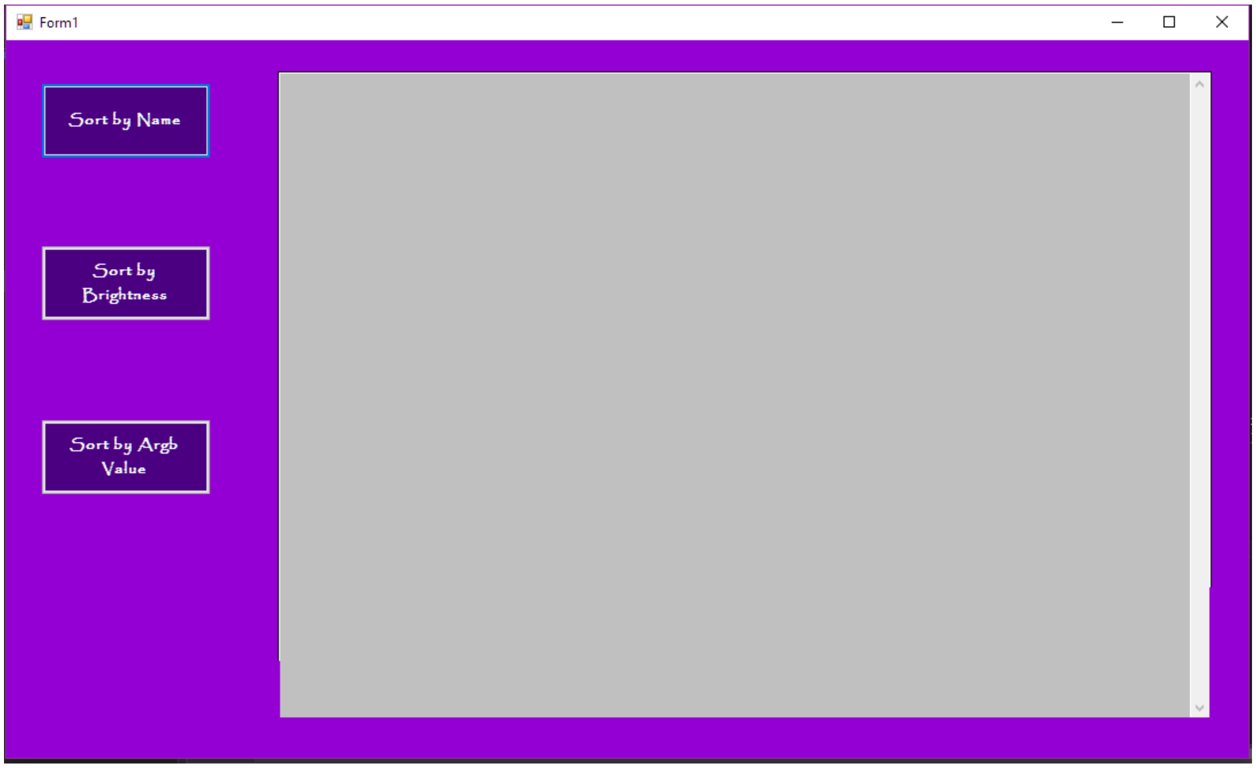
Within each button click, different sort strategies are invoked. The first line will clear out the text box. The next line will set the sort strategy. The third line will call the list and then the next two lines will print out the data.

Global

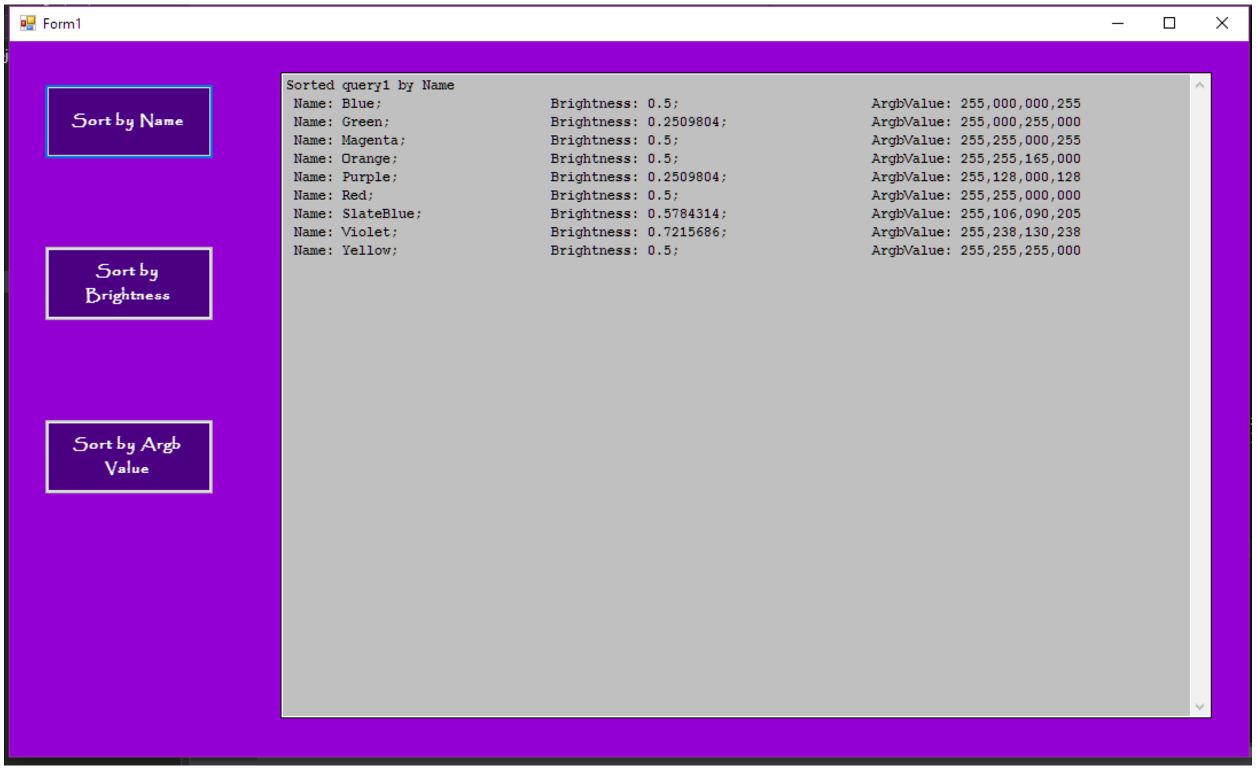
```
    public static class Global
    {
        public static string gv_DisplayData = "";
    }
```

This is a global class that I created so that I can use this global string variable in all of my classes to return display data back to my form.

<u>Screen Shots</u>

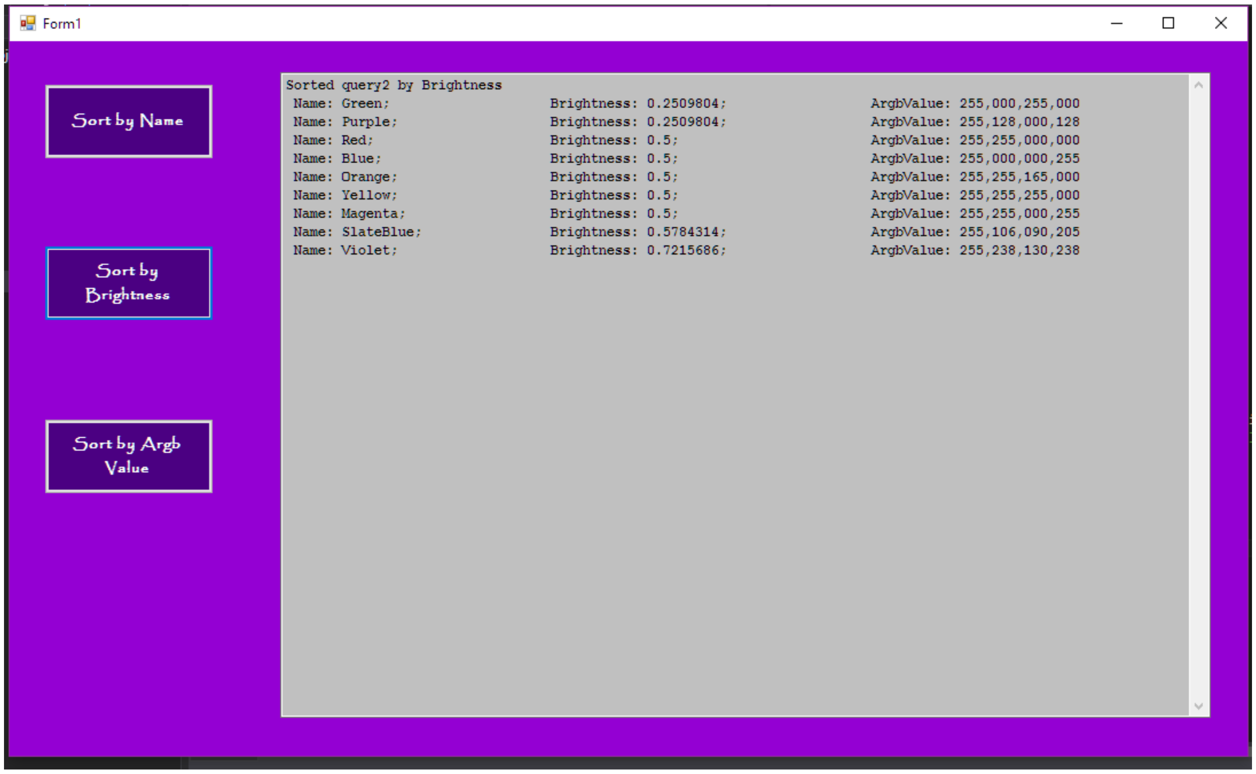


Screen on start up



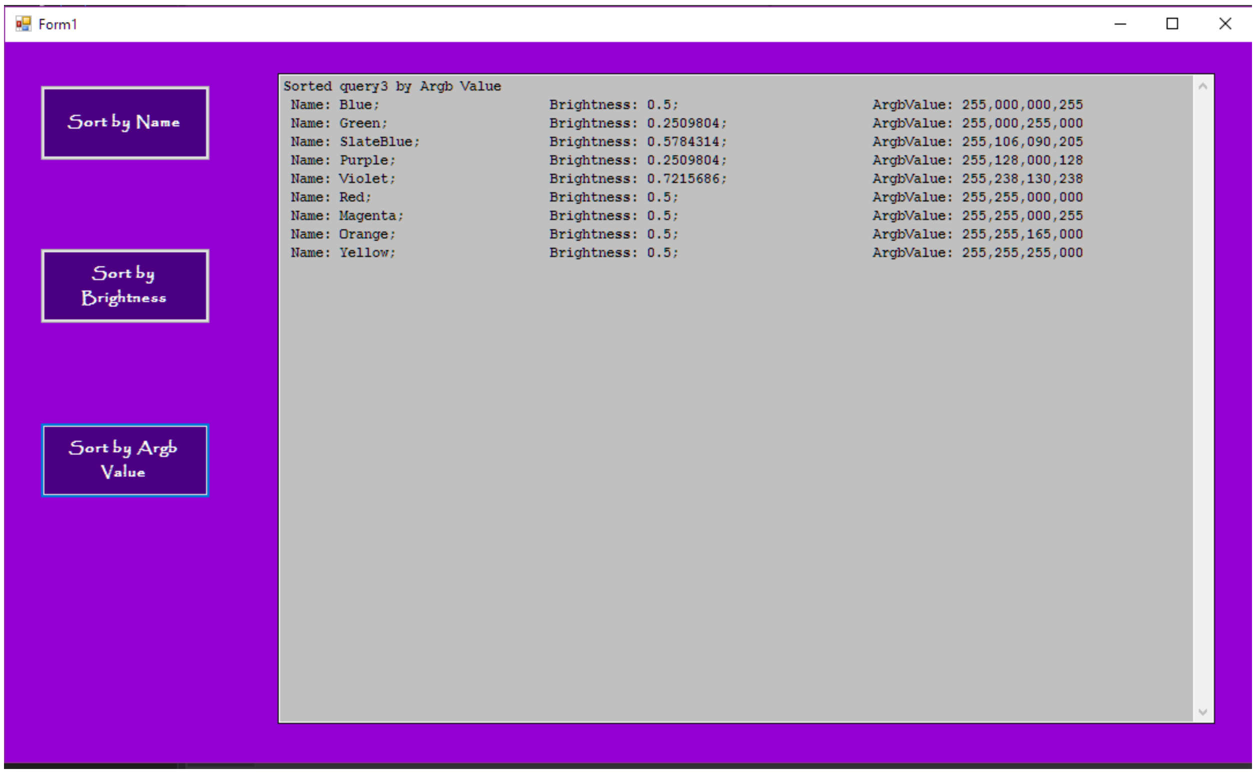
```
Sorted query1 by Name
 Name: Blue;                     Brightness: 0.5;                   ArgbValue: 255,000,000,255
 Name: Green;                    Brightness: 0.2509804;             ArgbValue: 255,000,255,000
 Name: Magenta;                  Brightness: 0.5;                   ArgbValue: 255,255,000,255
 Name: Orange;                   Brightness: 0.5;                   ArgbValue: 255,255,165,000
 Name: Purple;                   Brightness: 0.2509804;             ArgbValue: 255,128,000,128
 Name: Red;                      Brightness: 0.5;                   ArgbValue: 255,255,000,000
 Name: SlateBlue;                Brightness: 0.5784314;             ArgbValue: 255,106,090,205
 Name: Violet;                   Brightness: 0.7215686;             ArgbValue: 255,238,130,238
 Name: Yellow;                   Brightness: 0.5;                   ArgbValue: 255,255,255,000
```

Form after Sort by Name button is clicked



```
Sorted query2 by Brightness
 Name: Green;                    Brightness: 0.2509804;             ArgbValue: 255,000,255,000
 Name: Purple;                   Brightness: 0.2509804;             ArgbValue: 255,128,000,128
 Name: Red;                      Brightness: 0.5;                   ArgbValue: 255,255,000,000
 Name: Blue;                     Brightness: 0.5;                   ArgbValue: 255,000,000,255
 Name: Orange;                   Brightness: 0.5;                   ArgbValue: 255,255,165,000
 Name: Yellow;                   Brightness: 0.5;                   ArgbValue: 255,255,255,000
 Name: Magenta;                  Brightness: 0.5;                   ArgbValue: 255,255,000,255
 Name: SlateBlue;                Brightness: 0.5784314;             ArgbValue: 255,106,090,205
 Name: Violet;                   Brightness: 0.7215686;             ArgbValue: 255,238,130,238
```

Form after Sort by Brightness button is clicked



```
Sorted query3 by Argb Value
 Name: Blue;                Brightness: 0.5;               ArgbValue: 255,000,000,255
 Name: Green;               Brightness: 0.2509804;         ArgbValue: 255,000,255,000
 Name: SlateBlue;           Brightness: 0.5784314;         ArgbValue: 255,106,090,205
 Name: Purple;              Brightness: 0.2509804;         ArgbValue: 255,128,000,128
 Name: Violet;              Brightness: 0.7215686;         ArgbValue: 255,238,130,238
 Name: Red;                 Brightness: 0.5;               ArgbValue: 255,255,000,000
 Name: Magenta;             Brightness: 0.5;               ArgbValue: 255,255,000,255
 Name: Orange;              Brightness: 0.5;               ArgbValue: 255,255,165,000
 Name: Yellow;              Brightness: 0.5;               ArgbValue: 255,255,255,000
```

Form after Sort by Argb button is clicked

Observations

After figuring out how the pattern was supposed to work I think this program went pretty smoothly. In the beginning I ran into a lot of problems with trying to understand how to implement this pattern into my own program. I really struggled with coming up with an idea, but once I did, I was able to find some example code and go from there. The biggest issue I ran into was trying to pass the colors through. Originally I had wanted to use System.Drawing.Color in my list but, no matter how hard I tried and how many different examples I looked up, I could not get those types of colors to work. Instead I used an enumerated list to pass in my data. Overall I really like how this project turned out and with messing around originally with the System.Drawing.Color objects I actually understand how those objects work better now and hopefully in the future I can do more with those kinds of objects.