Strategy Pattern Assignment

Design Patterns

Jacob Hartman

20 October 2016

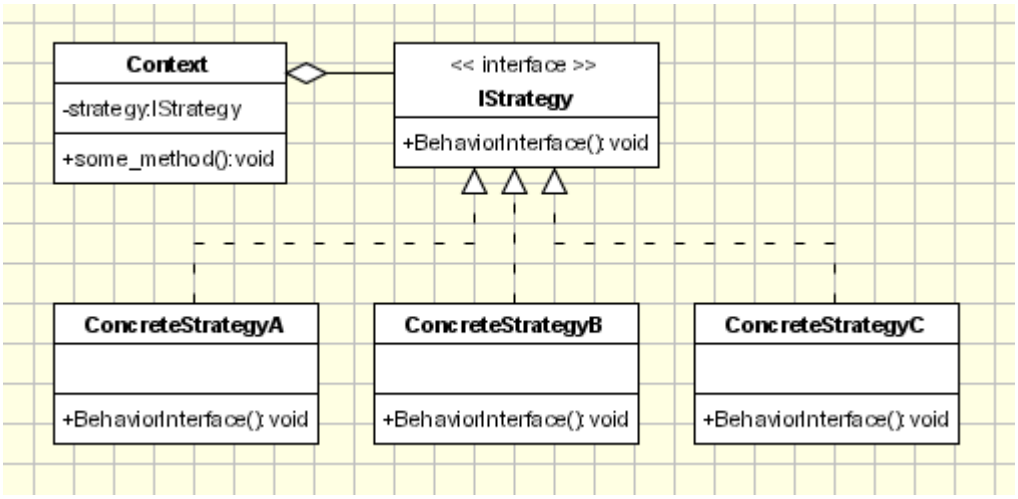
Introduction

This assignment required the creation of a program implementing the strategy pattern. The strategy pattern allows subsystems to select which algorithm to use for handling data based on the circumstances.

The application I wrote for this assignment handles the sorting of integer arrays. It implements three common sorting algorithms: insertion sort, selection sort, and bubble sort. The user can input a list of numbers and select an algorithm. The list is then sorted using the appropriate algorithm.

UML Diagram

image taken from http://www.oodesign.com/strategy-pattern.html



Above is the UML Diagram for the strategy pattern. It shows the need for four classes. An interface called IStrategy must have a BehaviorInterface method. Three ConcreteStrategy classes must implement the interface and its method. Lastly, the Context class must instantiate the interface and use its method via a some_method method.

My IStrategy class is an interface called Sort. For my ConcreteStrategy classes, I have InsertionSort,

SelectionSort, and BubbleSort. The Context class is part of my main form.

Narrative

The first class I wrote was the Sort interface.

```
public interface Sort // this is the IStrategy interface
{
    // this is the behaviorInterface method
    List<int> sortList(List<int> list);
}
```

The interface simply declares the need for a sortList method which accepts an integer list as input and returns an integer list. This method is the BehaviorInterface method. Its signature is declared so that implementing classes will know the structure to use.

```
public class InsertionSort : Sort // this is the ConcreteStrategyA class
{
    public List<int> sortList(List<int> list)
    {
        int limit = list.Count;
        int smallList = limit;
        int least = list[0];
        List<int> newList = new List<int>();
        for (int j = 0; j < limit; j++)
        {
            for (int i = 0; i < smallList; i++)
            {
                if (list[i] > least)
                {
                    least = list[i];
                }
            }
            newList.Add(least);
            list.Remove(least);
            smallList -= 1;
            least = 0;
        }
        newList.Reverse();
        return newList;
    }
}
```

The first ConcreteStrategy class is the InsertionSort class. It's sortList method sorts the input integer list using the insertion sort algorithm. This algorithm finds the smallest value in the original list and inserts it into a new list, doing this repeatedly until the new list is a sorted version of the original.

```
public class SelectionSort : Sort // this is the ConcreteStrategyB class
{
    public List<int> sortList(List<int> list)
    {
        int limit = list.Count;
        int temp = list[0];
```

```
            for (int i = 0; i < limit; i++)
            {
                for (int j = 0; j < limit; j++)
                {
                    if (list[j] > temp)
                    {
                        temp = list[j];
                    }
                }
                list.Remove(temp);
                list.Insert(i, temp);
            }
            return list;
        }
    }
```

The second ConcreteStrategy class is the SelectionSort class. This class also uses the sortList method to sort an input integer list. However, it does so using the selection sort algorithm. Through this algorithm, the smallest value is checked for and placed at the beginning of the list. The next-least value is then found and placed in the second position. This is repeated until the entire list is sorted.

```
    public class BubbleSort : Sort // this is the ConcreteStrategyC class
    {
        public List<int> sortList(List<int> list)
        {
            int limit = list.Count;
            bool hasSwapped = false;
            int temp = list[0];
            for (int i = 0; i < limit; i++)
            {
                if (list[i] < temp)
                {
                    list[i - 1] = list[i];
                    list[i] = temp;
                    hasSwapped = true;
                }
                else
                {
                    temp = list[i];
                }
            }
            if (hasSwapped)
            {
                return sortList(list);
            }
            else{
                return list;
            }
        }
    }
```

The final ConcreteStrategy class is the BubbleSort class. It, too, uses the sortList method to sort an input list of integers. The algorithm it uses is called bubble sort. This algorithm goes through the list in pairs, swapping the numbers in each pair so that they are in decreasing order. It tracks if any swaps had to be made during a pass. The method operates recursively, calling itself for each pass. Once no swaps

are made in a pass, the final list is returned.

```
public partial class Form1 : Form
{
    private List<int>     list;
    private Sort          sorter;
    private InsertionSort inserter;
    private SelectionSort selector;
    private BubbleSort    bubble;
    public Form1()
    {
        InitializeComponent();
        cb_sortType.SelectedIndex = 0;
        list     = new List<int>();
        inserter = new InsertionSort();
        sorter   = inserter;
        selector = new SelectionSort();
        bubble   = new BubbleSort();
    }
```

The Form1 class includes the code necessary for the Context class. When created, it creates an integer list and a Sort object which can hold any of the three strategies that are also instantiated in the constructor. The menu's combo box is also initialized on the first option, and the sorter object is set to point to an InsertionSort object, which corresponds to the first option.

```
    private void displayList()
    {
        lb_numList.Items.Clear();
        foreach (int a in list)
        {
            lb_numList.Items.Add("" + a);
        }
    }

    private void sortList(List<int> aList)
    {
        list = sorter.sortList(aList);
        displayList();
    }
```

The displayList method allows the integer list to be conveniently displayed in the list box on the form. The sortList method is the some_method method, calling the sortList method of the sorter object. This will sort the list using whatever algorithm the sorter is set to point to. It then calls the displayList method to show the user the sorted list.

```
    private void btn_add_Click(object sender, EventArgs e)
    {
        int a = (int)nud_newNum.Value;
        list.Add(a);
        displayList();
    }
```

There is a button on the main form next to a number spinner. The button allows the user to add the

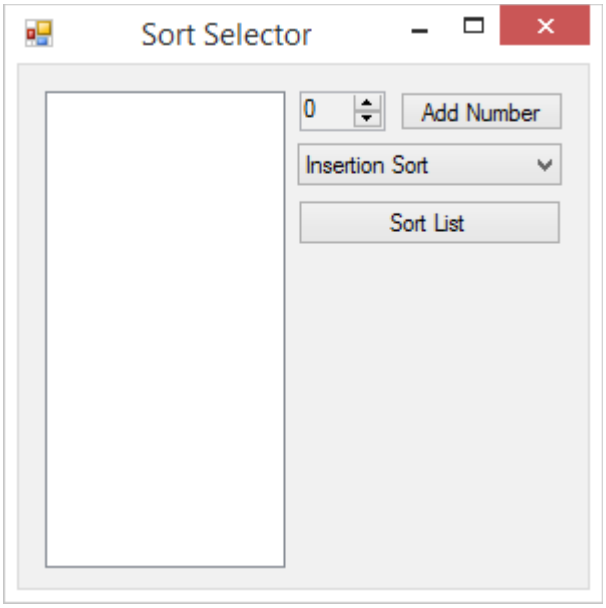
value of the spinner to the integer list. This is then displayed for the user to see.

```
        private void cb_sortType_SelectedIndexChanged(object sender, EventArgs e)
        {
            int a = cb_sortType.SelectedIndex;
            if (a == 0)
            {
                sorter = inserter;
            }
            else if (a == 1)
            {
                sorter = selector;
            }
            else
            {
                sorter = bubble;
            }
        }
```

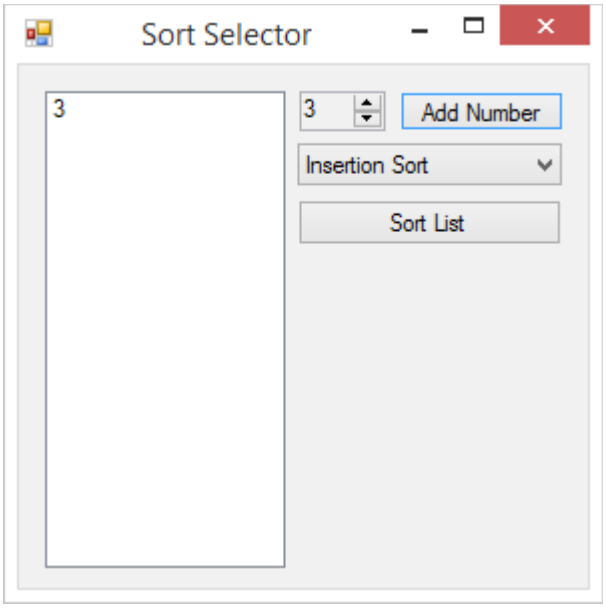
The combo box on the form allows the user to select an algorithm. Any time the selected index changes, this event handler is called. It points the sorter object to the appropriate algorithm based on what the user has selected.

```
        private void btn_sort_Click(object sender, EventArgs e)
        {
            if (list.Count > 0)
            {
                sortList(list);
            }
        }
    }
```

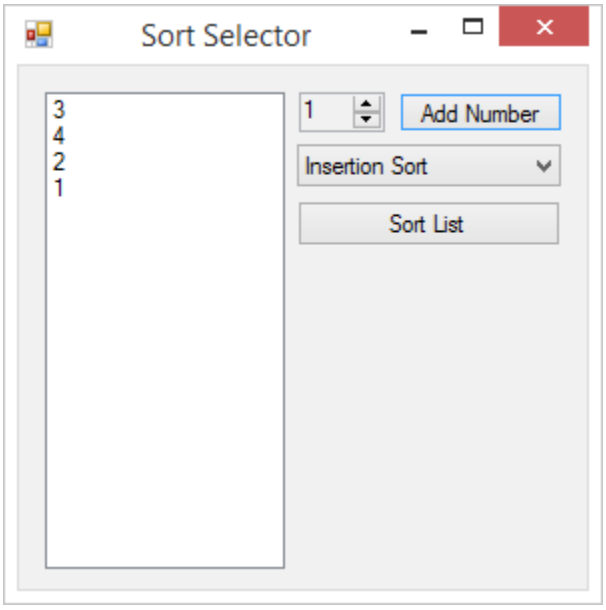
The sort button is where the pattern is truly utilized. If the integer list is not empty, the sortList method is called when the button is clicked. This allows the algorithm currently selected to be called and used to sort the list, which is then displayed. Below are screenshots of the program in use.



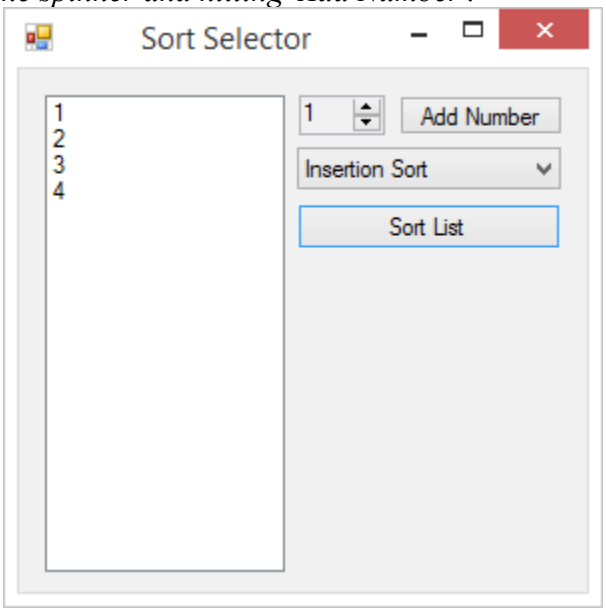
The initial view of the application.



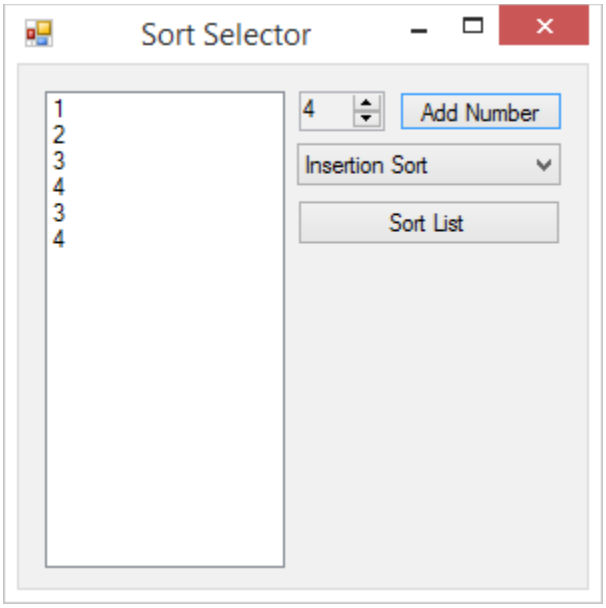
Integers can be added by selecting a value in the spinner and hitting 'Add Number'.



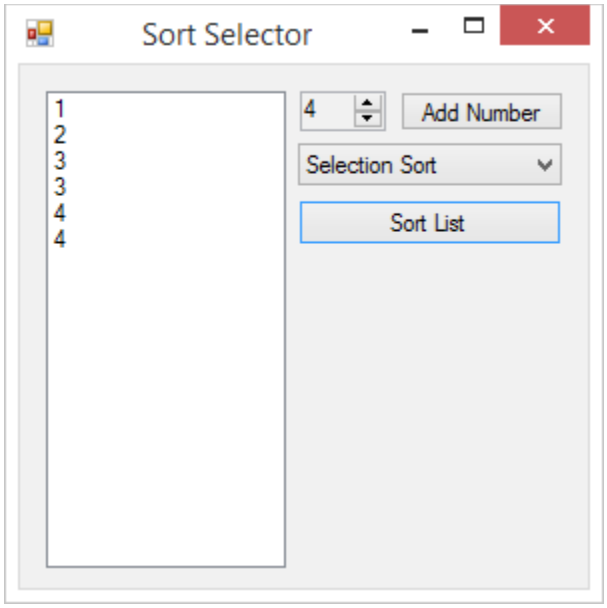
Integers can be added in any order.



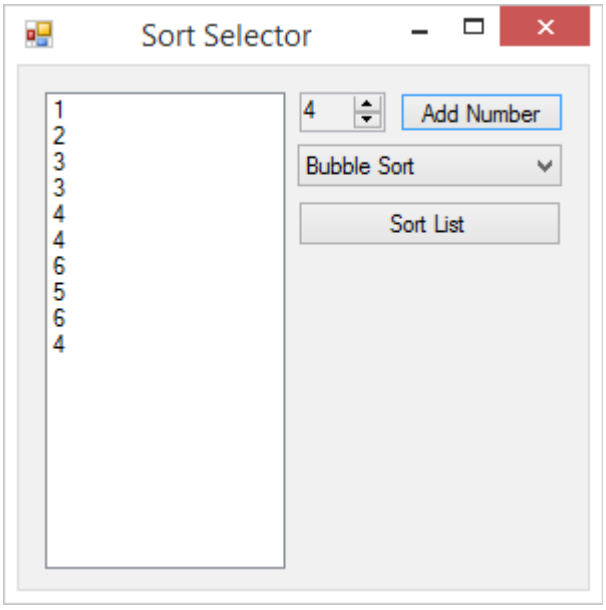
When Insertion Sort is selected and the 'Sort' button is clicked, the list is sorted using the insertion sort algorithm.



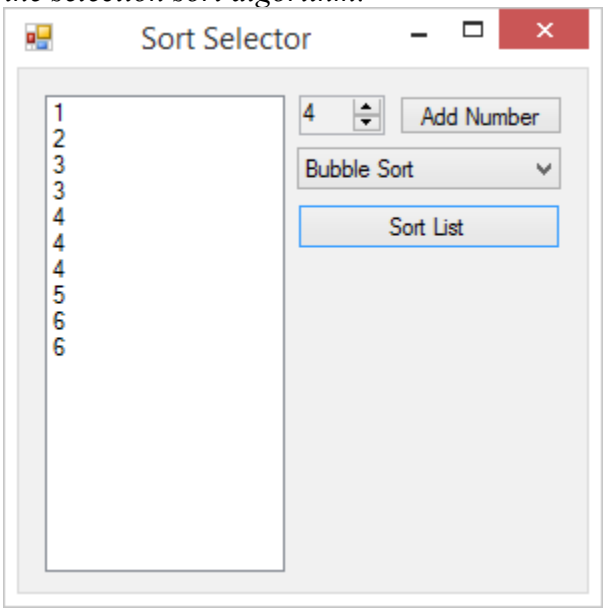
More numbers are added to demonstrate use.



Choosing Selection Sort will sort the list with the selection sort algorithm.



More numbers are added to demonstrate use.



Selecting Bubble Sort will sort the list with the bubble sort algorithm.

Observations

The strategy pattern is fairly simple, but it could potentially help avoid a lot of headaches. The subject matter I chose actually hints at a potential use for the pattern. Different sorting algorithms vary in efficiency based on the input. A program could be set up to choose the most efficient algorithm based on the data. I feel that my submission fits the assignment well. The UML Diagram was followed, and the resulting program functions properly. This assignment reinforced the idea that code can be reusable. I know this pattern will be helpful in the future.