Dylan Storts Design Patterns Professor Retterer

Iterator Pattern

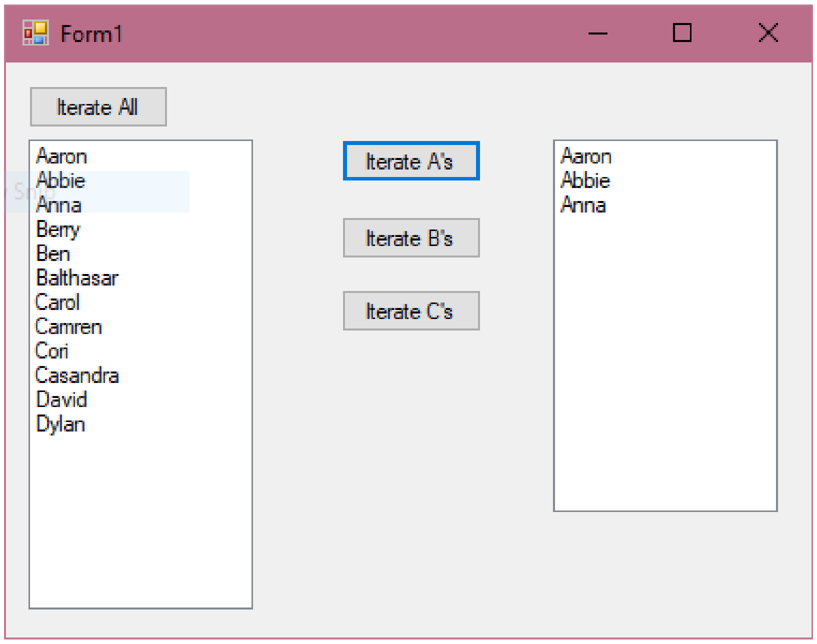
Introduction: For this assignment we were required to make a demo program for the iterator pattern to demonstrate that we really knew what the pattern is and how it works. The definition for the iterator pattern is as follows for those that don't know:

"Provide a way to access the elements of an aggregate object sequentially without exposing its underlying representation." *-dofactory.com*

Analysis: The iterator patterns first objective is simple because it is basically a loop used to access the elements in a collection. The key part that makes it The Iterator Pattern is that you cannot expose the "underlying representation." For this reason abstract classes were needed that defined what the collection was and could be inherited by the aggregate, or collection, class which would in turn "hide" the representation of the collection. I only made my collection an inherited list of strings, like my professor did in his demo, to make my aggregate and the code for that can be seen in the code snippet to the right.

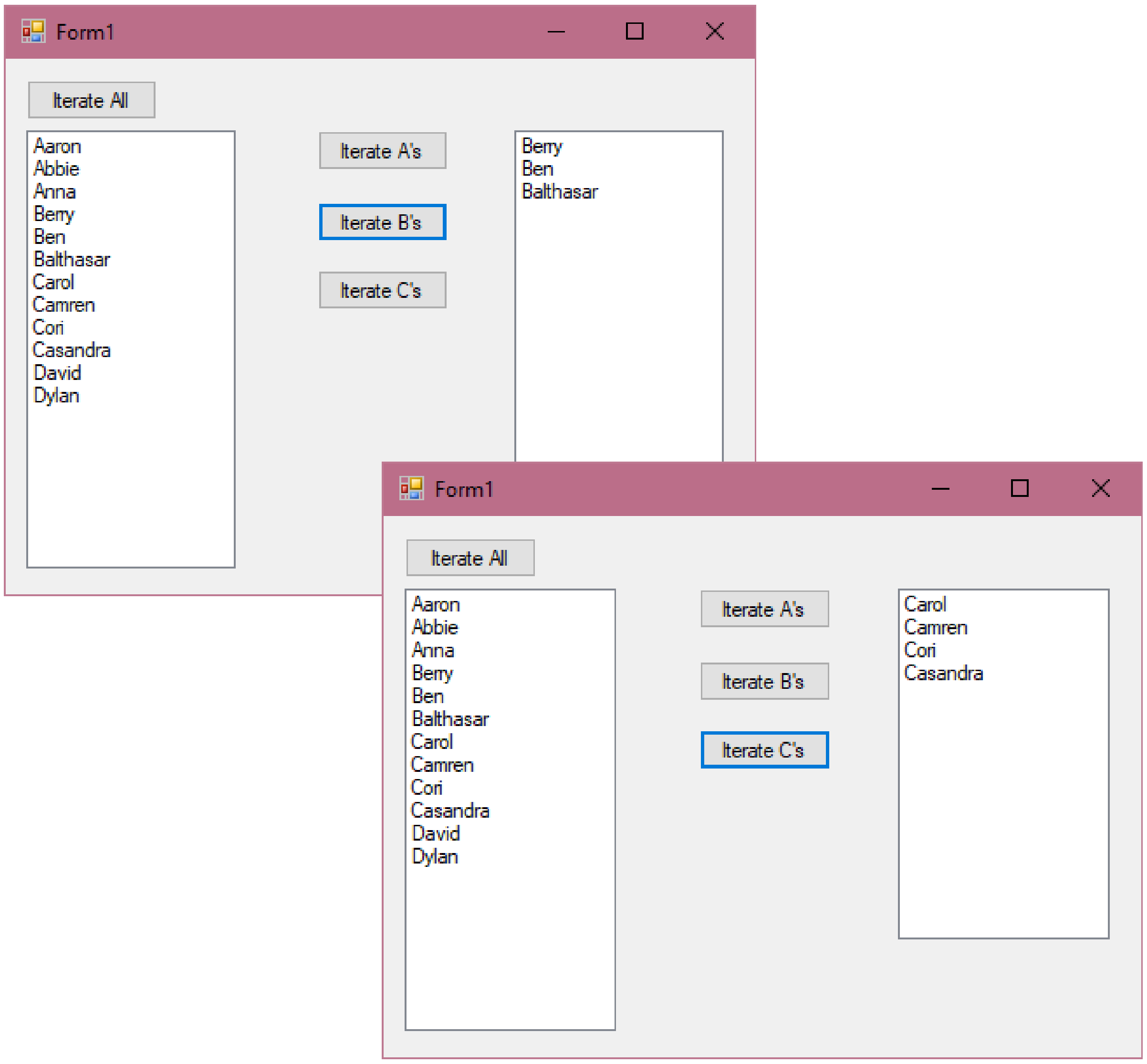
```
public abstract class Aggregate : List<string> {}
```

Then, I used strings of names to populate my aggregate and open up different ways to show my iterator does its job. The names were entered in alphabetical order and so I created an iterator that only stopped on and printed the names that started with a particular letter. To simplify that demo my list was only ten names long and only had names starting with the first few letters of the alphabet. I also included an iterator that stopped to return every element in my aggregate to show the full list to be used to compare if the specified letter iterators work. All of these iterators can be seen in the illustrations below.



```
Aaron
Abbie
Anna
Berry
Ben
Balthasar
Carol
Camren
Cori
Casandra
David
Dylan
```

```
Aaron
Abbie
Anna
```



```
Aaron
Abbie
Anna
Berry
Ben
Balthasar
Carol
Camren
Cori
Casandra
David
Dylan
```

```
Berry
Ben
Balthasar
```

```
Aaron
Abbie
Anna
Berry
Ben
Balthasar
Carol
Camren
Cori
Casandra
David
Dylan
```

```
Carol
Camren
Cori
Casandra
```

The "Iterate All" button runs the iterator that stops and returns all of the items in my collection of names. When it came to creating a `First()` function for the specific letters I set up a while loop to step the current item index counter by one every time the current item, starting from the first, did not contain its specified letter. This code for the "Iterate B's" button can be seen in the text box to the right. This stopped at the first name with 'B' as the first letter and then a special `Next()` function was called, seen in the text box bottom right, that only stepped the current index value to the next item if and only if the next name also started with the letter 'B' and if it did not then the current index would become the size of the collection to force the `isDone()` function to decided that the iteration was done only because we know that the names are in alphabetical order.

```
public override void First()
        {
            current = 0;
            while (agg[current][0] != 'B')
            {
                current += step;
            }
        }
```

```
public override void Next()
        {
          if (agg[current + step][0] == 'B')
              current += step;

          else
              current = agg.Count;
        }
```

Furthermore, the actions that are made to create and iterate through the aggregate can be seen in the text box below along with the rest of my code at the end of this document.

```
private void btn_iterateAll_Click(object sender, EventArgs e)
        {
            ConcreteAggregate agg = new ConcreteAggregate();

            PopulateAgg(agg);

            Iterator iter = 
agg.CreateIterator(IteratorType.completeIterator);

            for(iter.First(); iter.IsDone(); iter.Next())
            {
                listbox_displayAll.Items.Add(iter.CurrentItem() );
            }
        }
```

Reflection: This was one of the hardest programs that I have written thus far because of the use of abstract functions and the fact that I did not know, and still don't fully know, how they worked. My current understanding is that an anything abstract is only required to be a template for some other object to define it later. But I don't get why you use them unless it gets inherited or you're hiding some variables or functions that are shared without revealing exactly what they are. Which is the only purpose I thought the abstract aggregate and iterator did for me in my demo. My Visual Studio Solution for this project and many others can be found on my GitHub at https://github.com/dylanstorts/Design_Patterns_Projects

Form Action Events and Code

```
private void btn_iterateAll_Click(object sender, EventArgs e)
        {
            ConcreteAggregate agg = new ConcreteAggregate();

            PopulateAgg(agg);

            Iterator iter = agg.CreateIterator(IteratorType.completeIterator);

            for(iter.First(); iter.IsDone(); iter.Next())
            {
                listbox_displayAll.Items.Add(iter.CurrentItem() );
            }
        }

        private void PopulateAgg(ConcreteAggregate aggregate)
        {
            aggregate.Add("Aaron");
            aggregate.Add("Abbie");
            aggregate.Add("Anna");
            aggregate.Add("Berry");
            aggregate.Add("Ben");
            aggregate.Add("Balthasar");
            aggregate.Add("Carol");
            aggregate.Add("Camren");
            aggregate.Add("Cori");
            aggregate.Add("Casandra");
```

```
    aggregate.Add("David");
    aggregate.Add("Dylan");
}

private void btn_IterateA_Click(object sender, EventArgs e)
{
    ConcreteAggregate agg = new ConcreteAggregate();

    PopulateAgg(agg);

    Iterator iter = agg.CreateIterator(IteratorType.aIterator);

    listbox_selected.ClearSelected();

    for (iter.First(); iter.IsDone(); iter.Next())
    {

        listbox_selected.Items.Add(iter.CurrentItem());
    }
}

private void btn_IterateB_Click(object sender, EventArgs e)
{
    ConcreteAggregate agg = new ConcreteAggregate();

    PopulateAgg(agg);

    Iterator iter = agg.CreateIterator(IteratorType.bIterator);

    listbox_selected.ClearSelected();

    for (iter.First(); iter.IsDone(); iter.Next())
    {

        listbox_selected.Items.Add(iter.CurrentItem());
    }
}

private void btn_IterateC_Click(object sender, EventArgs e)
{
    ConcreteAggregate agg = new ConcreteAggregate();

    PopulateAgg(agg);

    Iterator iter = agg.CreateIterator(IteratorType.cIterator);

    listbox_selected.ClearSelected();

    for (iter.First(); iter.IsDone(); iter.Next())
    {

        listbox_selected.Items.Add(iter.CurrentItem());
    }
}
```

Code Defining all my Iterators and Aggregates

```
                                    namespace IteratorLibrary
{


    /// The 'Aggregate' that inherits a list of strings thus hiding the the makeup of the
iterator

    public abstract class Aggregate : List<string> {}

    public enum IteratorType {completeIterator, aIterator, bIterator, cIterator }

    /// The 'ConcreteAggregate' class

    public class ConcreteAggregate : Aggregate
    {

        public Iterator CreateIterator(IteratorType iter)
        {
            //return new CompleteIterator(this);

            switch (iter)
            {
                case IteratorType.completeIterator:
                    return new CompleteIterator(this);

                case IteratorType.aIterator:
                    return new AIterator(this);

                case IteratorType.bIterator:
                    return new BIterator(this);

                case IteratorType.cIterator:
                    return new CIterator(this);

                default: return new CompleteIterator(this);
            }

        }
    }

    /// The 'Iterator' interface

    public abstract class Iterator
    {
        abstract public void First();
        abstract public void Next();
        abstract public bool IsDone();
        abstract public string CurrentItem();
    }

    /// The Concrete Iterator that goes over all elements

    public class CompleteIterator : Iterator
    {
        private int current;
        private int step = 1;
        private ConcreteAggregate agg;
```

```csharp
        public CompleteIterator(ConcreteAggregate aggregate)
        {
            this.agg = aggregate;
        }

        public override string CurrentItem()
        {
            if (IsDone())
                return agg[current];
            else
                throw new Exception();
        }

        public override void First()
        {
            current = 0;
        }

        public override bool IsDone()
        {
            return  !( current >= agg.Count );
        }

        public override void Next()
        {
            current += step;
        }

    }

    public class AIterator : Iterator
    {
        private int current;
        private int step = 1;
        private ConcreteAggregate agg;

        public AIterator(ConcreteAggregate aggregate)
        {
            this.agg = aggregate;
        }

        public override string CurrentItem()
        {
            if (IsDone())
                return agg[current];
            else
                throw new Exception();
        }

        public override void First()
        {
            while(agg[current][0] != 'A')
            {
                current += step;
            }
        }

        public override bool IsDone()
        {
            return !(current >= agg.Count);
        }
        public override void Next()
        {
            if (agg[current + step][0] == 'A')
                current += step;

            else
                current = agg.Count;
        }
    }

    public class BIterator : Iterator
    {
        private int current;
        private int step = 1;
        private ConcreteAggregate agg;

        public BIterator(ConcreteAggregate aggregate)
        {
            this.agg = aggregate;
        }

        public override string CurrentItem()
        {
            if (IsDone())
                return agg[current];
            else
                throw new Exception();
        }

        public override void First()
        {
            current = 0;
            while (agg[current][0] != 'B')
            {
                current += step;
            }
        }

        public override bool IsDone()
        {
            return !(current >= agg.Count);
        }

        public override void Next()
        {
            if (agg[current + step][0] == 'B')
                current += step;

            else
                current = agg.Count;
        }
    }

    public class CIterator : Iterator
    {
        private int current;
        private int step = 1;
        private ConcreteAggregate agg;

        public CIterator(ConcreteAggregate aggregate)
        {
            this.agg = aggregate;
        }

        public override string CurrentItem()
        {
            if (IsDone())
                return agg[current];
            else
                throw new Exception();
        }

        public override void First()
        {
            current = 0;
            while (agg[current][0] != 'C')
            {
                current += step;
            }
        }

        public override bool IsDone()
        {
            return !(current >= agg.Count);
        }


        public override void Next()
        {
            if (agg[current + step][0] == 'C')
                current += step;

            else
                current = agg.Count;
        }
    }
}
```