Design Patterns
Prof. Retterer
Iterator Pattern

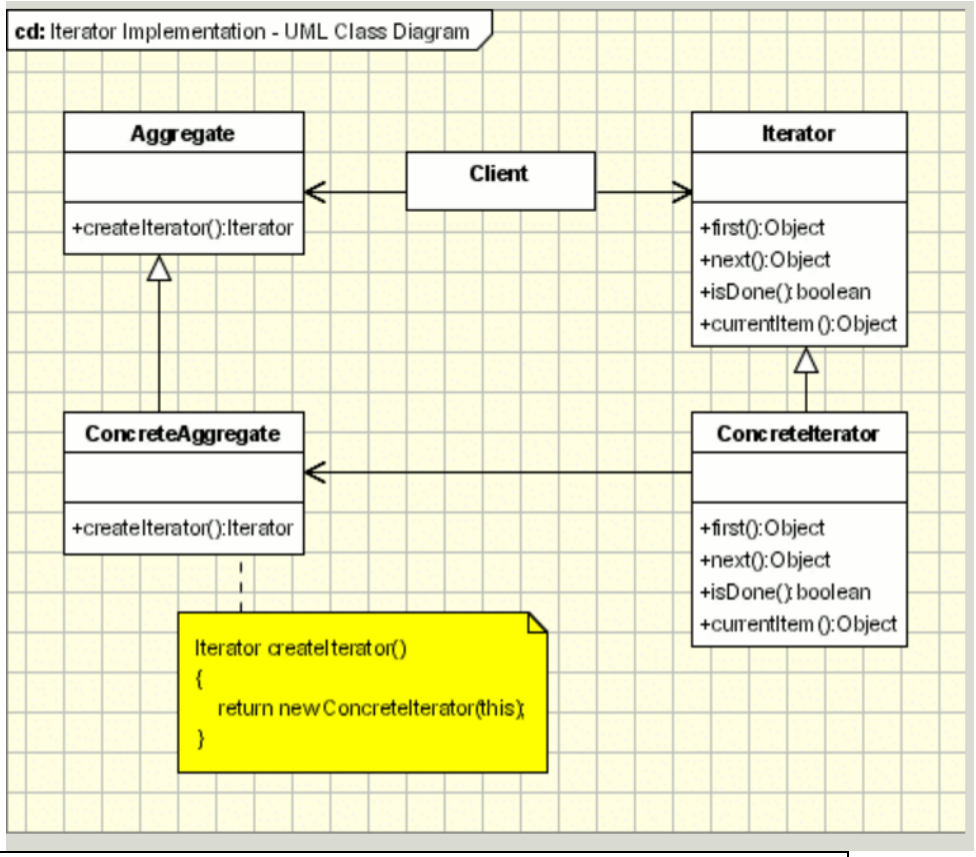
Jeremy Martin
9/1/16

Introduction

This assignment asked us to make a simple application that demonstrated the Iterator Pattern. This assignment also had us using this pattern to iterate through the aggregate in two different ways. This application iterates through a .NET list and displays the elements of the list in order of most important or in order of least importance. All of my code can be seen in my GitHub repository located here https://github.com/jmartin5076/Design-Patterns.

UML Diagram for Iterator

The UML diagram on the right displays the components necessary to satisfy the Iterator Pattern and the relationships between them. Some of my classes are generic in order to satisfy the abstract classes. The table below shows the objects that I determined to be the classes and gives a description of each.



Aggregate	I selected the .NET generic class list<string> to fulfill this condition.
ConcreteAggregate	This class is derived from implementation of the Aggregate class and dictates the type of iterator to use on the list.
Iterator	This is an abstract class that dictates the first, next, isDone, and currentItem methods.
ConcreteIterator	This class is created by the ConcreteAggregate and implements the Iterator class by defining the methods. This also iterates based on the type of Aggregate that is chosen to create it.
Client	Demonstration application.

The code that I show here is modified slightly to allow for the use of two different iterators.The code for the abstract Aggregate class is shown below.

Jeremy Martin


```
public abstract class Aggregate // this is the Aggregate class
    {
        public List<string> Members;

        public abstract Iterator createIterator();
        public abstract Iterator createIteratorLeastImport();
    }
```

The next bit of code shows the ConcreteAggregate class which implements the Aggregate class.

```
public class ConcreteAggregate : Aggregate // this is the ConcreteAggregate class
    {
        public ConcreteAggregate()
        {
            Members = new List<string>();
        }

        public override Iterator createIterator()
        {
            return new ConcreteIterator(this);
        }

        public override Iterator createIteratorLeastImport()
        {
            return new ConcreteIteratorLeastImport(this);
        }
    }
```

The nest piece of code shows the Iterator abstract class.

```
public abstract class Iterator // this is the Iterator class
    {
        public abstract object first();
        public abstract object next();
        public abstract bool isDone();
        public abstract object currentItem();
    }
```

The next two pieces of code are the two ConcreteIterators that both implement the Iterator class. The first piece shows the actual ConcreteIterator class, and the second piece iterates the list in the reverse order.

```
public class ConcreteIterator : Iterator // this is the ConcreteIterator class
    {
        Aggregate aggregate;
        int CurrentIndex;

        public ConcreteIterator(Aggregate agg)
        {
            aggregate = agg;
        }

        public override object currentItem()
        {
            if(isDone())
            {
                return null;
            }
           return aggregate.Members[CurrentIndex];
```

Jeremy Martin


```
        }

        public override object first()
        {
            CurrentIndex = 0;
            return currentItem();
        }

        public override bool isDone()
        {
            return (CurrentIndex > aggregate.Members.Count - 1);
        }

        public override object next()
        {
            if(!isDone())
            {
                CurrentIndex++;
            }
            return currentItem();
        }
    }


public class ConcreteIteratorLeastImport : Iterator
    {
        Aggregate aggregate;
        int CurrentIndex;

        public ConcreteIteratorLeastImport(Aggregate agg)
        {
            aggregate = agg;
        }

        public override object currentItem()
        {
            if (isDone())
            {
                return null;
            }
            return aggregate.Members[CurrentIndex];
        }

        public override object first()
        {
            CurrentIndex = aggregate.Members.Count -1;
            return currentItem();
        }

        public override bool isDone()
        {
            return (CurrentIndex < 0);
        }

        public override object next()
        {
            if (!isDone())
```

```
            {
                CurrentIndex--;
            }
            return currentItem();
        }
    }
```

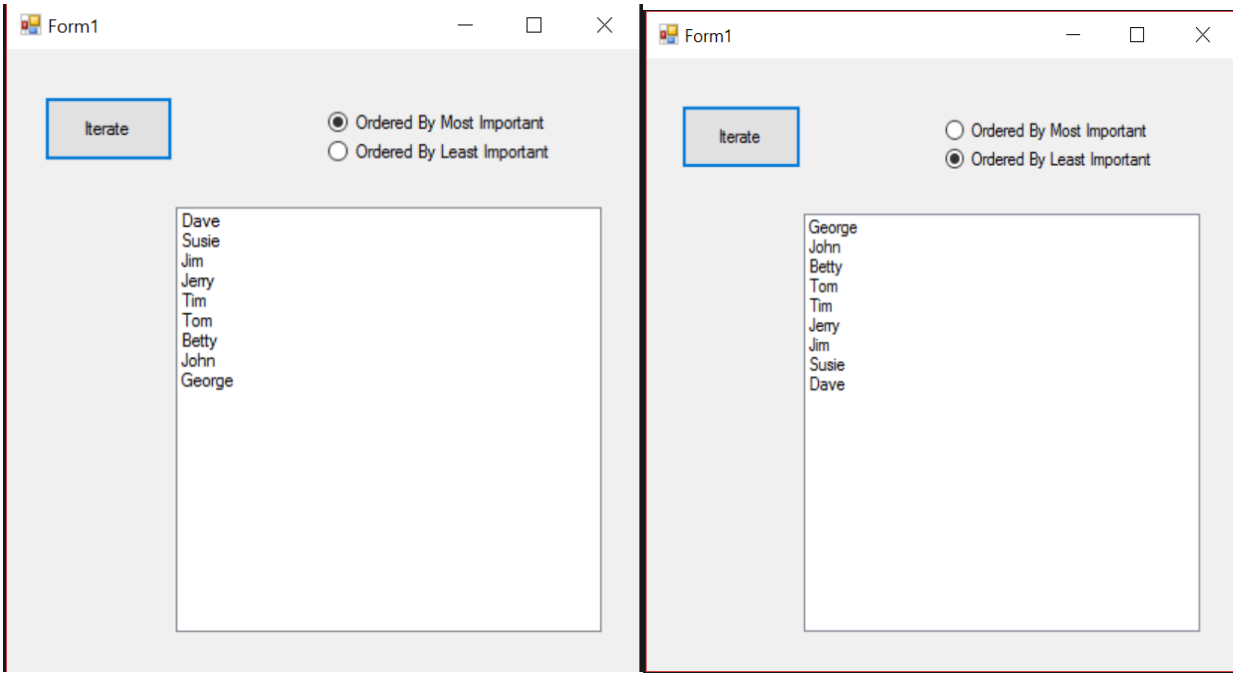
The last piece of code deals with the client and is used to make the form that the user sees.

```
public partial class Form1 : Form
    {
        Aggregate agg = new ConcreteAggregate();
        Iterator iterator;

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

        private void PrepareAggWithIter()
        {
            agg.Members.Add("Dave");
            agg.Members.Add("Susie");
            agg.Members.Add("Jim");
            agg.Members.Add("Jerry");
            agg.Members.Add("Tim");
            agg.Members.Add("Tom");
            agg.Members.Add("Betty");
            agg.Members.Add("John");
            agg.Members.Add("George");
        }

        private void m_btnIterate_Click(object sender, EventArgs e)
        {
            m_lbMembers.Items.Clear();
                iterator.first();
                while (!iterator.isDone())
                {
                    m_lbMembers.Items.Add(iterator.currentItem());
                    iterator.next();
                }
        }

        private void m_rbMostImport_CheckedChanged(object sender, EventArgs e)
        {
            iterator = agg.createIterator();
        }

        private void m_rbLeastImport_CheckedChanged(object sender, EventArgs e)
        {
            iterator = agg.createIteratorLeastImport();
        }
    }
```

Here are some screenshots of my code in action, the first is in the Most Important order and the second is in the Least Important order.

Design Patterns
Prof. Retterer
Iterator Pattern

Jeremy Martin
9/1/16



Final Thoughts

Overall, I thought that this was an interesting and fun project to do. The UML diagram was very concise and easy to understand and the video that you provided ended up helping a great deal with the overall functionality of my code. I wish that I understood a little bit more about the way to iterate through the list and create more complex objects to populate the list with, my original plan dealt with creating a list of names of members in a club and being able to iterate the list so that it would only show officers in the club rather than all the members.