Tori Lentz

Dave Retterer

Design Patterns

September 2, 2016

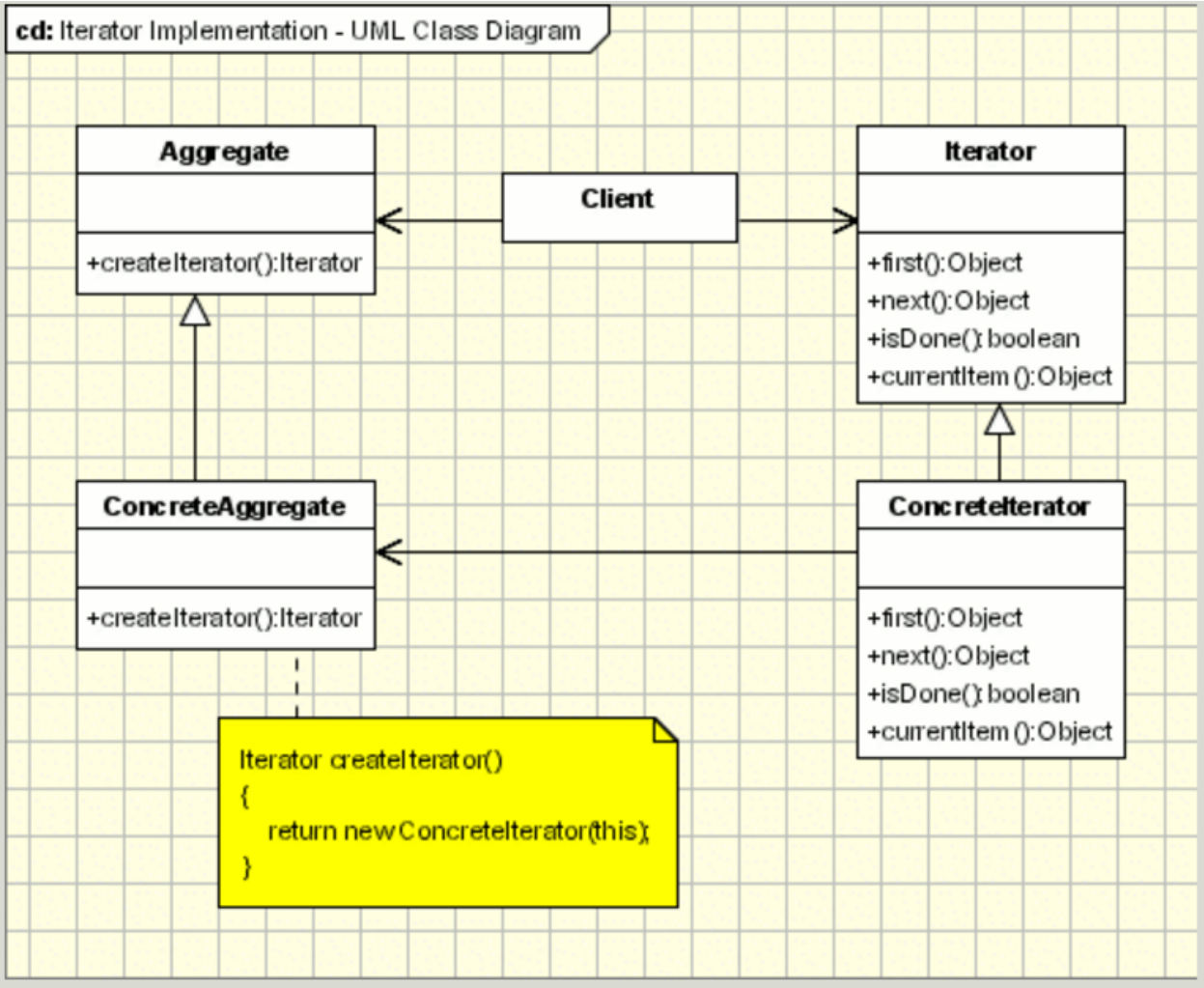
The Iterator Pattern

Introduction

The goal of this assignment was to design and implement the iterator pattern. Through the use of stacks, one can create many different iterations.

The UML Diagram for the Iterator Pattern

The UML diagram shown on the right describes all of the parts necessary for this pattern to meet the specifications described in the assignment.



Aggregate: The Aggregate class was created by creating a public abstract class that implements Aggregate.

```
public abstract class Aggregate // this is the aggregate class
    {
        public List<int> Elements;

        public abstract Iterator createIterator();

    }
```

Iterator: The Iterator class was created by creating a public abstract class that implements Iterator. Within this class is where all of the instructions for the pattern exist. There are 3 objects and a boolian variable. The objects first, next and current item all take care of the placing of an element within a list. The boolian variable then takes care of knowing when an object is finished with its job.

```
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();
    }
```

Iterator Two: As with the above class this second iterator class established the objects needed to run the iteration.

```
public abstract class IteratorTwo
    {
        public abstract object First();
        public abstract object Next();
        public abstract bool IsDone();
        public abstract object CurrentItem();
    }
```

Concrete Aggregate: This class inherits the properties of class Aggregate and expands upon it.

```
public class ConcreteAggregate : Aggregate // this is the
class concrete aggregate
    {
        public ConcreteAggregate()
        {
            Elements = new List<int>();
        }
        public override Iterator createIterator()
        {
            return new ConcreteIterator(this);
        }

    }
```

Concrete Iterator: This class expands upon the Iterator class. Within the Concrete Iterator class are the instructions for the objects mentioned in the Iterator class. Here each command is told what to do and the iteration occurs when run.

```
public class ConcreteIterator : Iterator //this is the
concrete iterator class
    {
        Aggregate aggregate;
        int currentItem;

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

        public override object First() {
            currentItem = 0;
            return CurrentItem();
        }

        public override object Next() {
            if (!IsDone())
                currentItem++;
            return CurrentItem();
        }

        public override bool IsDone() {
            return (currentItem > aggregate.Elements.Count -
1);
        }

        public override object CurrentItem() {
            if (IsDone())
                return null;
            return aggregate.Elements[currentItem];
        }
    }
```

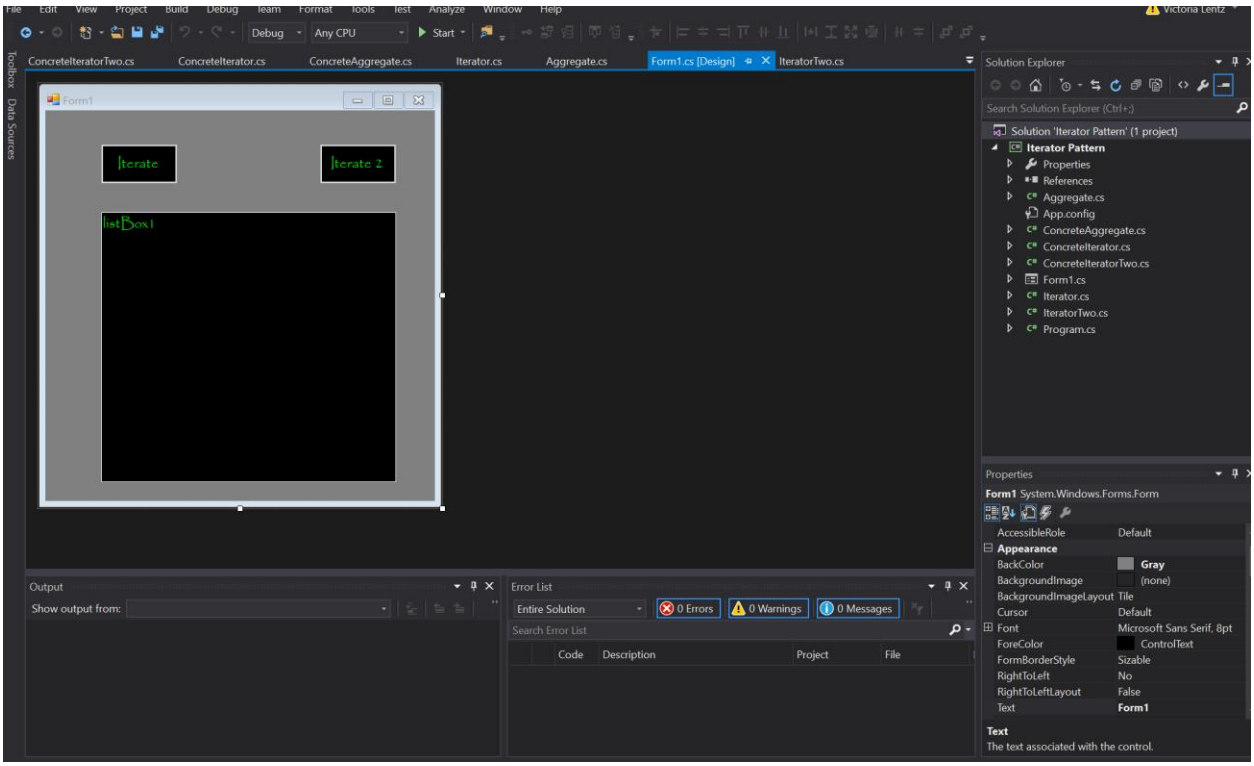
Concrete Iterator : This is the second concrete iterator class that makes it so that a second iteration can be run.

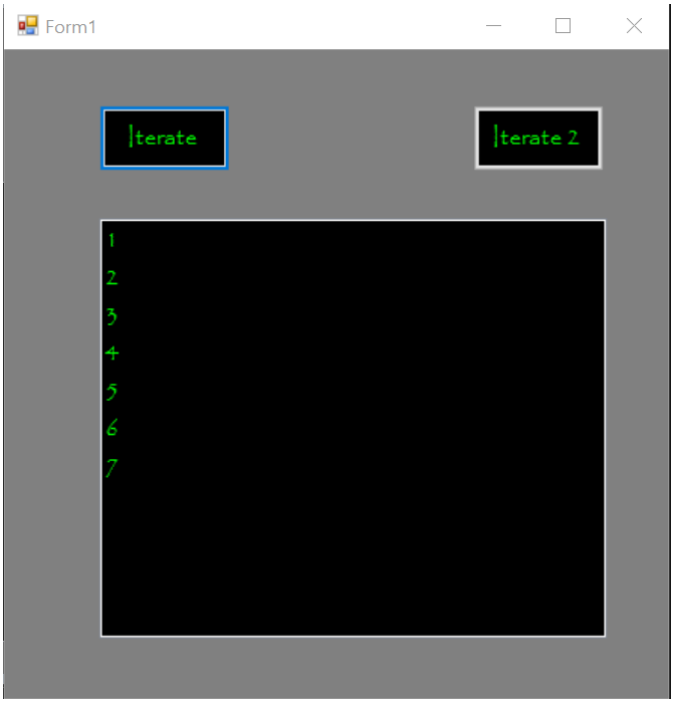
```
public class ConcreteIteratorTwo : IteratorTwo //this is the
concrete iterator class
    {
        Aggregate aggregate;
        int currentItem;

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

        public override object First()
        {
            currentItem = 0;
            return CurrentItem();
        }

        public override object Next()
        {
            if (!IsDone())
                currentItem--;
            return CurrentItem();
        }

        public override bool IsDone()
        {
            return currentItem < 0;
        }

        public override object CurrentItem()
        {
            if (IsDone())
                return null;
            return aggregate.Elements[currentItem];
        }
    }
```

Form: This is the code that comes from the form. The elements to be iterated are established here as well as the code for the buttons.

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

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

        private void PrepareAggWithIter()
        {
           agg.Elements.Add(1);
           agg.Elements.Add(2);
           agg.Elements.Add(3);
           agg.Elements.Add(4);
           agg.Elements.Add(5);
           agg.Elements.Add(6);
           agg.Elements.Add(7);
           iterator = agg.createIterator();
           iteratortwo = agg.createIteratorTwo();
        }

        private void button1_Click(object sender, EventArgs e)
        {

            iterator.First();
            while (!iterator.IsDone())
            {
                listBox1.Items.Add(iterator.CurrentItem());
                iterator.Next();
            }
        }

        private void button2_Click(object sender, EventArgs e)
        {
            iteratortwo.First();
            while (!iteratortwo.IsDone())
            {
                listBox1.Items.Add(iteratortwo.CurrentItem());
                iteratortwo.Next();
            }
        }
    }
```

Screen Shots





Observations

Over all this assignment went well. After seeing the presentations in class I was able to go back into my code and fix what I had done wrong the first time. Originally I was on the right path but had added a few wrong lines of code which then made it so that my code would not run.