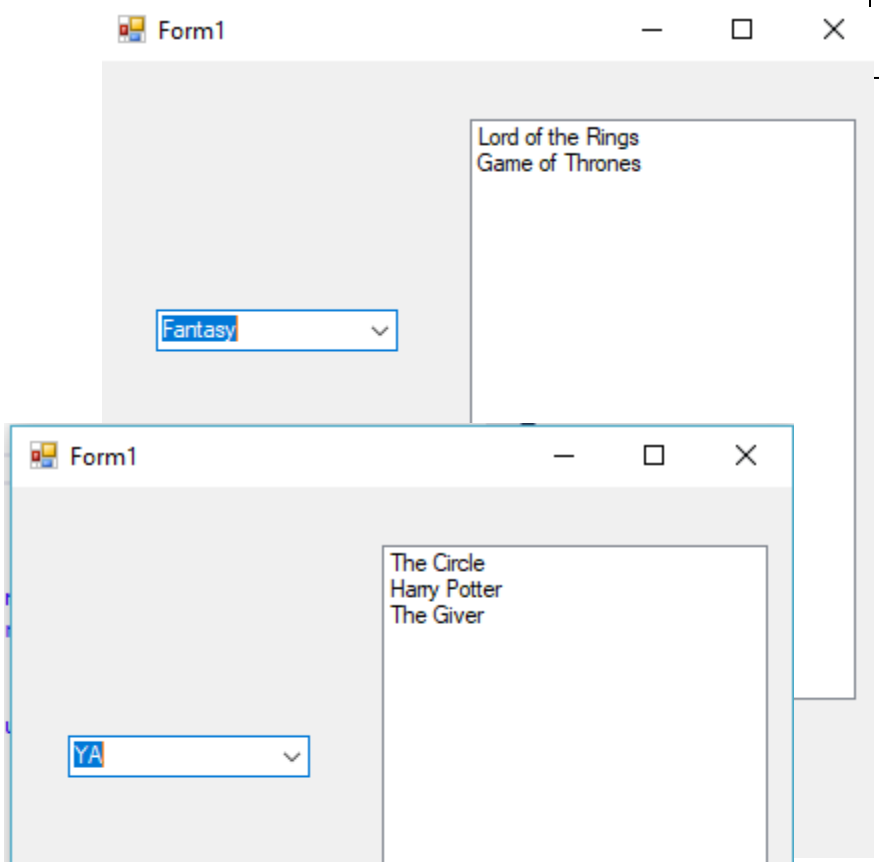
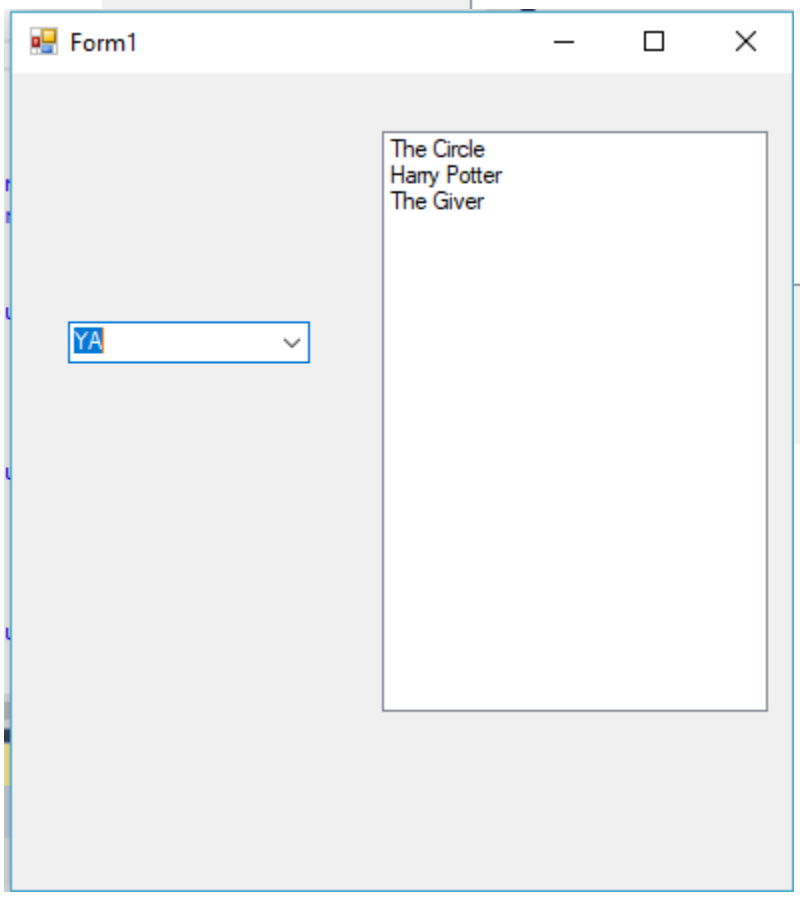
The Iterator Pattern

Introduction:

For this assignment I was asked to do the iterator design pattern. The iterator design pattern provides a way to access the elements of an aggregate object sequentially without exposing it's underlying implementation. I accomplished this task by creating a simple app that iterates through a list of book objects and outputs books based on genre that the user selects.

```
   public abstract class
Iterator
    {
        public abstract object
First();
        public abstract object
Next();
        public abstract bool
isDone();
        public abstract object
CurrentItem();
    }
```





Narrative:

My app allows you to select the genre of book you want to look up, and then outputs the list of that certain genre of book into a listbox. For my aggregate object I used a list of 'book' objects, and created a book class. You can see my iterator class that is based off what is in the UML diagram. My definitions of these abstract methods are in a concrete iterator class, and my program contains three different iterators, for the three different genres. To make use of these classes I made a list in my forms app that all the book objects are added to when the form is initialized. On the next page, you can see the definition for one of my iterators, as well as how the iterator class was used in my form.

Conclusion: I think that this assignment was actually really useful and I learned a lot, especially after re-writing it. After thinking about it a little bit I probably could have only had one iterator and just passed it the genre that it was searching for, but that's not how I ended up doing it even though that would have been a lot easier. Most of the code for

```
class ConcreteIterator : Iterator
{
    int current = 0;
    private ConcreteAggregate concreteAggregate;


    public ConcreteIterator(ConcreteAggregate concreteAggregate)
    {
        this.concreteAggregate = concreteAggregate;
    }

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

    public override void First()
    {
        current = -1;
        Next();
    }
    public override void Next()
    {
        current++;
        while (!(isDone() || ((aggregate[current].Genre == "Fantasy"))))
            current++;
    }

    public override object CurrentItem()
    {
        return aggregate[current].Name;

    }
    public override bool isDone()
    {

        return (current >= aggregate.Count);
    }
}
```

```
listBox1.Items.Clear();
Iterator iter = agg.CreateIterator(IteratorType.iterator);

for (iter.First(); !(iter.isDone()); iter.Next())
{
  listBox1.Items.Add(iter.CurrentItem());
              }
```

all three of the iterators are the same the only thing that's different between them is the string it's checking against. But, overall this assignment was good, I think it gave me a better understanding of the C# language, and how abstract classes work. Also was helpful in learning how to read UML diagrams.