C# Indexer

This contains some basic information about indexers in C#.

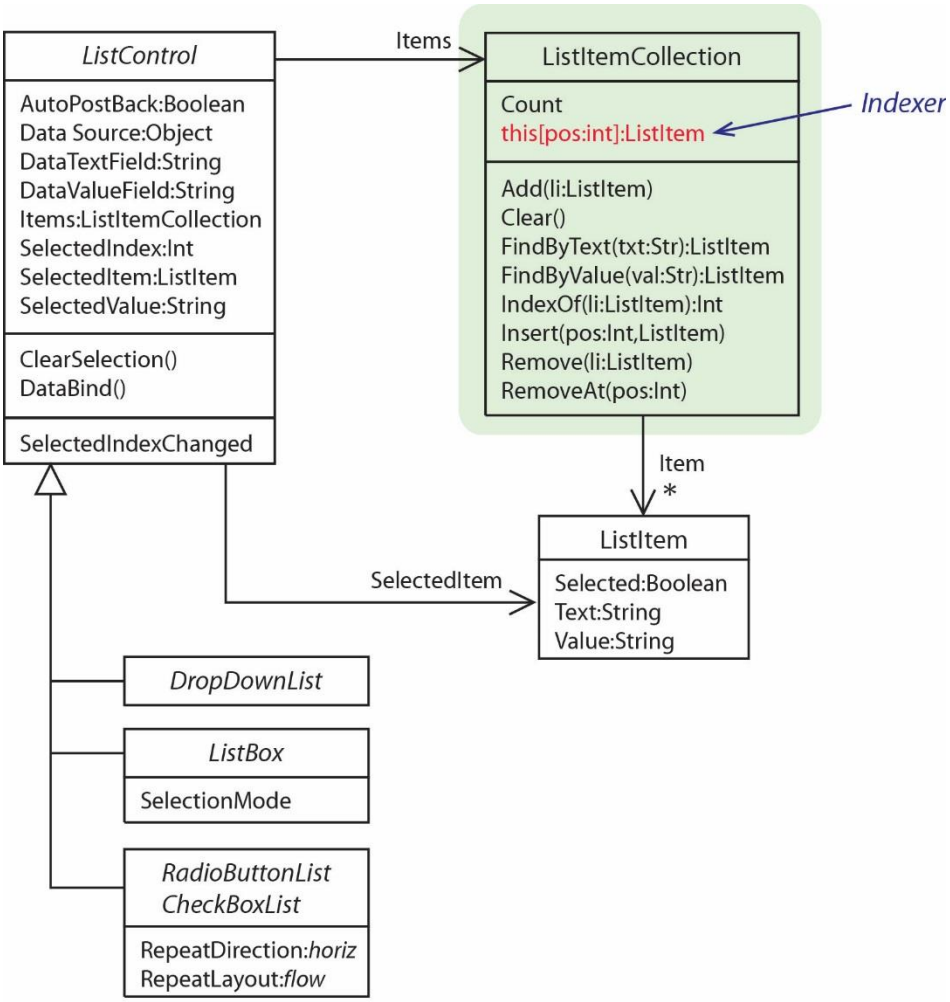
Introduction

1. The *ListControl* class has an *Items* property which is a *ListItemCollection.* The *ListItemCollection* class utilizes an *indexer* which provides access to the *ListItems* using array-like notation. For example:

```
for(int i=0; i<lbxNames.Items.Count; i++) {
        ListItem li = lbxNames.Items[i];
}
```

   In Java, we would write the statement inside the *for* like this:

```
ListItem li = lbxNames.Items.Get(i);
```



2. In general, an *indexer* is a C# technique that allows client code access to the elements in a private collection using an array-like notation.

   An indexer is a *default* property (in the C# sense) for a class that allows access to a private collection.

Example 1

3. Consider the *Exams* class on the right which holds an array of exam scores and can calculate the average. An indexer is defined by declaring a property whose name is *this*.

```
public class Exams {
    private int[] exams;

    public Exams(int num) {
        this.exams = new int[num]; }

    // Indexer
    public int this[int i] {
        get { return exams[i]; }
        set { exams[i] = value; }
    }

    public double Average() { ... }
}
```

4. To use the class and indexer:

```
Exams exams = new Exams(3);
exams[0] = 80;
exams[1] = 90;
exams[2] = 90;

double score = exams[1];
double avg = exams.Average();
```

Example 2

5. We can use a List to back the Indexer. In this case we will need a *Count* method (or property) to tell how many things are in the List.

```
public class Exams2 {
    List<double> exams;

       public Exams2() { exams = new List<double>(); }

    public double this[int i] {
        get {
            // List has an Items field that is an Indexer(!)
            return exams[i];
        }
        set { exams.Insert(i, value); }
    }

    public int Count() { return exams.Count; }

    public double Average() {
        double sum = 0.0;
        foreach (int score in exams) sum += score;
        return sum / exams.Count;
    }
}
```

Example 3

6. We can overload the indexer. In this case we overload the indexer to search for a Customer with a particular *name*.

```
public class Customers {
    List<Customer> customers;

    public Customers() { customers = new List<Customer>(); }

    public Customer this[int i] {
        get { return customers[i]; }
        set { customers.Insert(i, value); }
    }

    public Customer this[string name] {
        get {
            foreach (Customer c in customers)
                if (c.Name.Equals(name))
                    return c;
            return null;
        }
    }

    public void Add(Customer c) { customers.Add(c); }

    public int Count() { return customers.Count; }

    public override string ToString() { ... }
}
```

Which we can use like this:

```
Customers customers = new Customers();

Customer c = new Customer("Dave", "123 Nevins");
customers[0] = c;
c = new Customer("Tom", "555 Slater");
customers.Add(c);
c = new Customer("Linda", "77 Ridgeway");
customers.Add(c);

c = customers["Tom"];
if (c != null)
    txtMsg.Text += "Customer found: " + c + Environment.NewLine;
else
    txtMsg.Text += "Customer not found" + Environment.NewLine;

c = customers["Renny"];
if (c != null)
    txtMsg.Text += "Customer found: " + c + Environment.NewLine;
else
    txtMsg.Text += "Customer not found" + Environment.NewLine + Environment.NewLine;
```