Generalized Collection Classes

- Array-like data structures
  - ArrayList
  - Queue
  - Stack
  - Hashtable
  - SortedList
- Offer programming convenience for specific access needs.
- Store *objects*
  - Add anything.
  - Typecast on removal.

Generics

- Generics offer typesafe alternatives.
  - New in .NET 2.0
- Generics are generally the preferred alternative.
  - Object collection classes offer advantages in certain unusual cases.

Collection Classes

- To use these classes we must write

```
using System.Collections.Generic;
```

for generics

Included in the default C# template

```
using System.Collections;
```

for object collections

Not included in the default C# template

Generics

- C# generics are like C++ *templates*.
- Classes written with blanks to be filled in by the user (parameters)
  - Cannot be instantiated directly.
  - We create a specialized version for a specific type by supplying the name of the type when the class is used.
    - Not a macro!
- Supplying the parameter effectively creates a new class, which can be instantiated.

The Generic List Class

- List<T> is the generic List class
  - T represents a class name parameter to be supplied in declarations.
- Provides traditional list operations
  - Insert
  - Delete
- Also provides array operations
  - Access by position, using index notation

List<T> Methods

- Add (T item)
  - Add item at end of list
- Insert (int index, T item)
  - Insert item at a specific position
- Remove (T item)
  - Remove first occurance of item
- RemoveAt (int index)
  - Remove item at specified position

List<T> Methods

- Clear()
  - Removes all items from the list
- bool Contains(T item)
  - Determines if item is in the list
- int IndexOf(T item)
  - Returns index of item, or -1 if not in list.
- Sort()

... more

List<T> Indexer

- Array index notation can be used to get or set a specified item.
  - int_list[5] = 17;
  - int temp = int_list[5];
- Throws an exception if int_list[5] does not exist.

List<T> Properties

- Count Number of items in the list

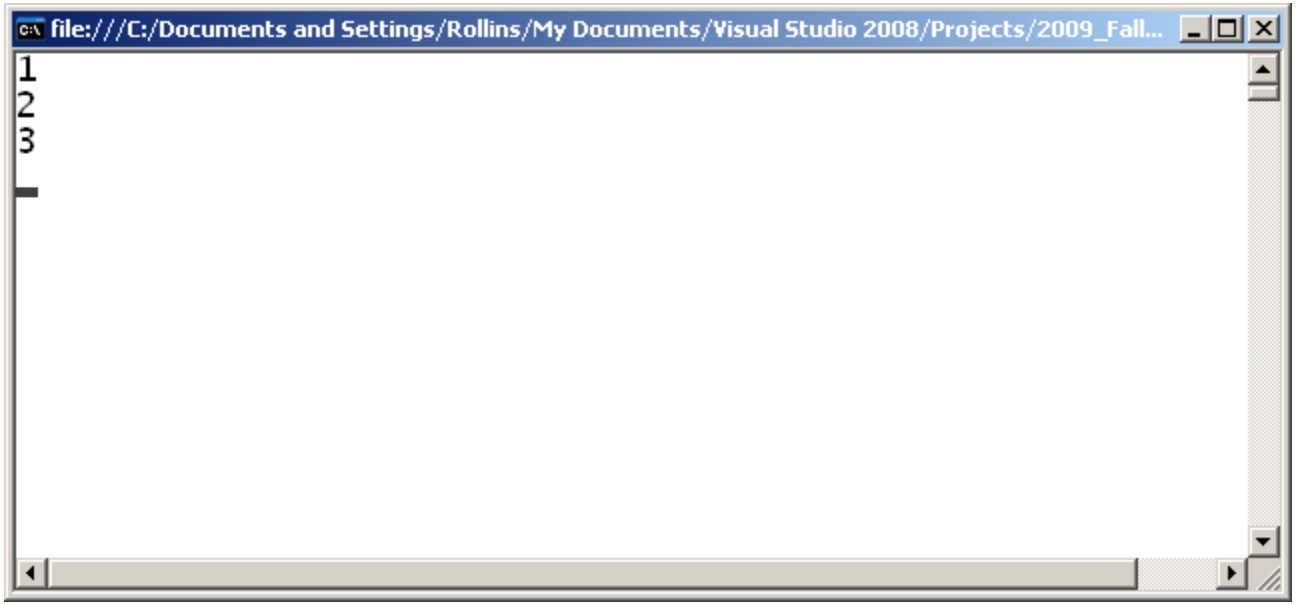
List<T> Example

- Create new C# Console Application project.

List<int> Example

```
static void Main(string[] args)
{
    List<int> ints = new List<int>();

    ints.Add(1);
    ints.Add(2);
    ints.Add(3);

    foreach (int i in ints)
    {
        Console.WriteLine(i.ToString() );
    }
    Console.ReadLine();
}
```

List<int> Example Running



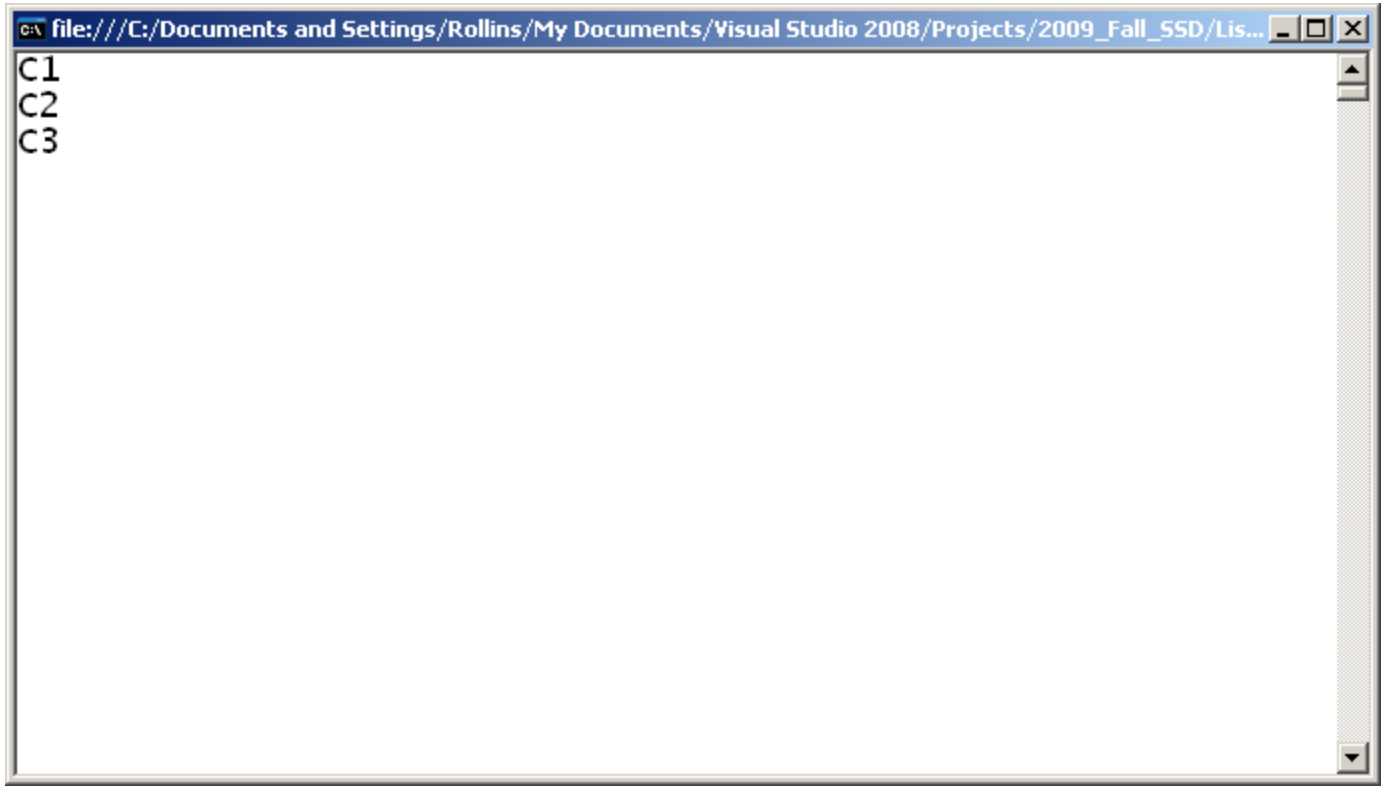
A List of Circles

- Add Circles.cs to the project
- Delete namespaces.

List<Circle> Example

```
using System;
using System.Collections.Generic;
class Program
{
    static void Main(string[] args)
    {
        List<Circle> circles = new List<Circle>();

        circles.Add(new Circle("C1", 1.0));
        circles.Add(new Circle("C2", 2.0));
        circles.Add(new Circle("c3", 3.0));

        foreach (Circle c in circles)
        {
            Console.WriteLine(c.Name());
        }
        Console.ReadLine();
    }
}
```

List<Circle> Example Running



Accessing List by Position

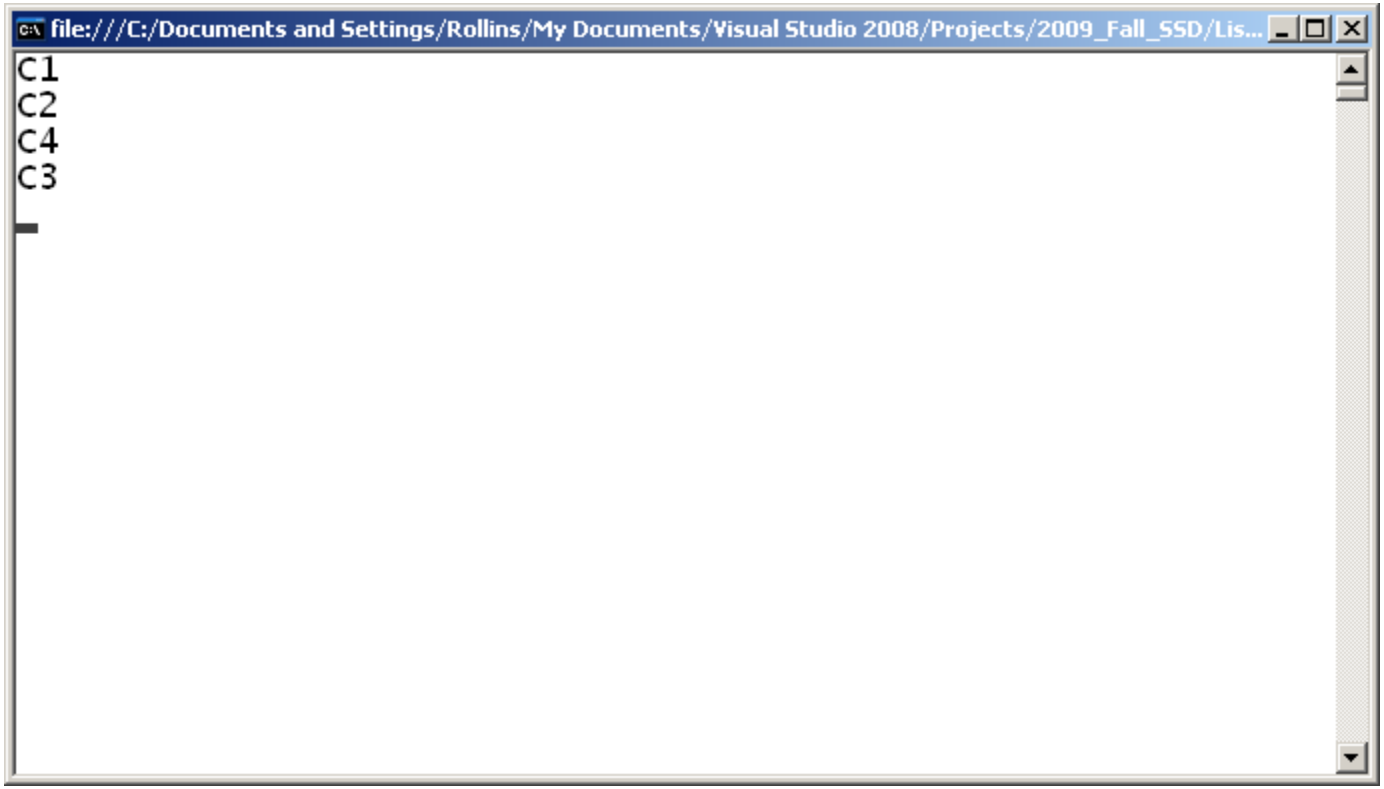
```
static void Main(string[] args)
{
    List<Circle> circles = new List<Circle>();

    circles.Add(new Circle("C1", 1.0));
    circles.Add(new Circle("C2", 2.0));
    circles.Add(new Circle("c3", 3.0));

    for (int i = 0; i < circles.Count; ++i)
    {
        Console.WriteLine(circles[i].Name());
    }
    Console.ReadLine();
}
```

Same result

Inserting

```
static void Main(string[] args)
{
    List<Circle> circles = new List<Circle>();

    circles.Add(new Circle("C1", 1.0));
    circles.Add(new Circle("C2", 2.0));
    circles.Add(new Circle("c3", 3.0));

    Circle c4 = new Circle("C4", 4.0);
    circles.Insert(2, c4);

    for (int i = 0; i < circles.Count; ++i)
    {
        Console.WriteLine(circles[i].Name());
    }
    Console.ReadLine();
}
```

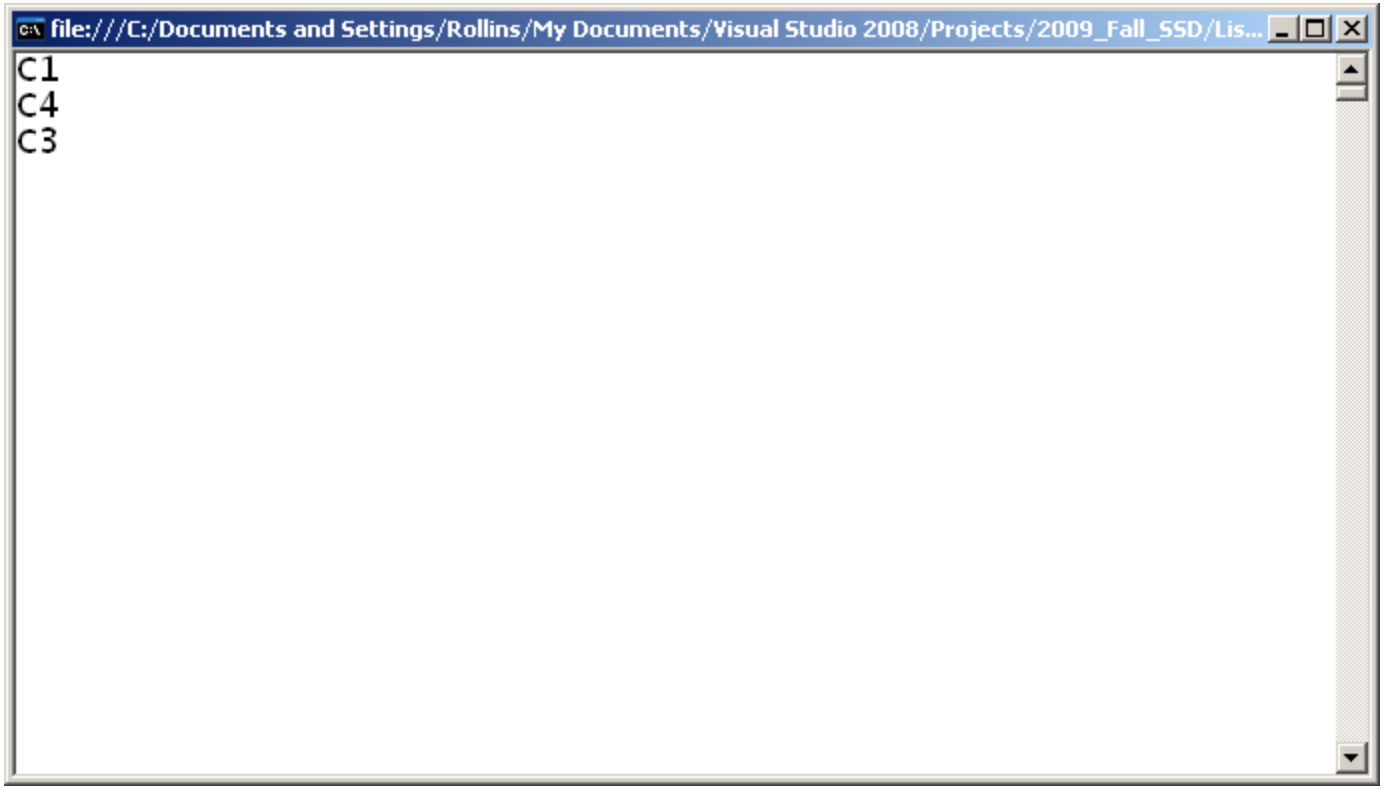
Inserting


file:///C:/Documents and Settings/Rollins/My Documents/Visual Studio 2008/Projects/2009_Fall_SSD/Lis...
C1
C2
C4
C3

Inserting and Deleting

```
static void Main(string[] args)
{
    List<Circle> circles = new List<Circle>();

    circles.Add(new Circle("C1", 1.0));
    circles.Add(new Circle("C2", 2.0));
    circles.Add(new Circle("c3", 3.0));

    Circle c4 = new Circle("C4", 4.0);
    circles.Insert(2, c4);

    circles.RemoveAt(1);

    for (int i = 0; i < circles.Count; ++i)
    {
        Console.WriteLine(circles[i].Name());
    }
    Console.ReadLine();
}
```

Inserting and Deleting



Sorting

- List<T> has a Sort method.
- Parameter class must implement the IComparable<T> interface.

- Example:

```
class Schedule_Record : IComparable<Schedule_Record>
{
    private String college;
    ...
```

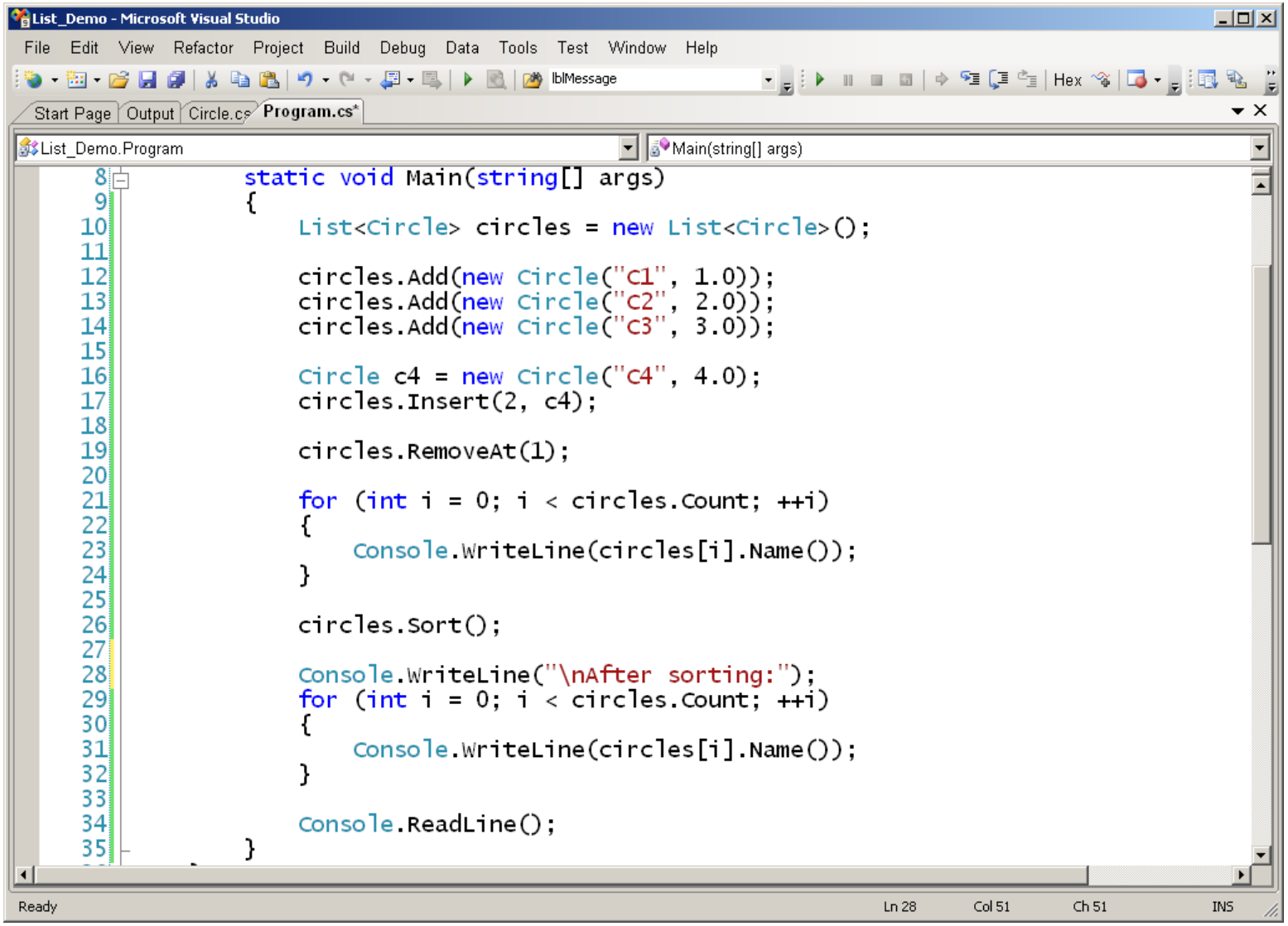
IComparable interface

- Class must implement CompareTo() method, taking an object of the same type as its parameter.
  - Return negative int if this object < other
  - Return 0 if this object == other
  - return positive int if this object > other
    - Same as strcmp() in C

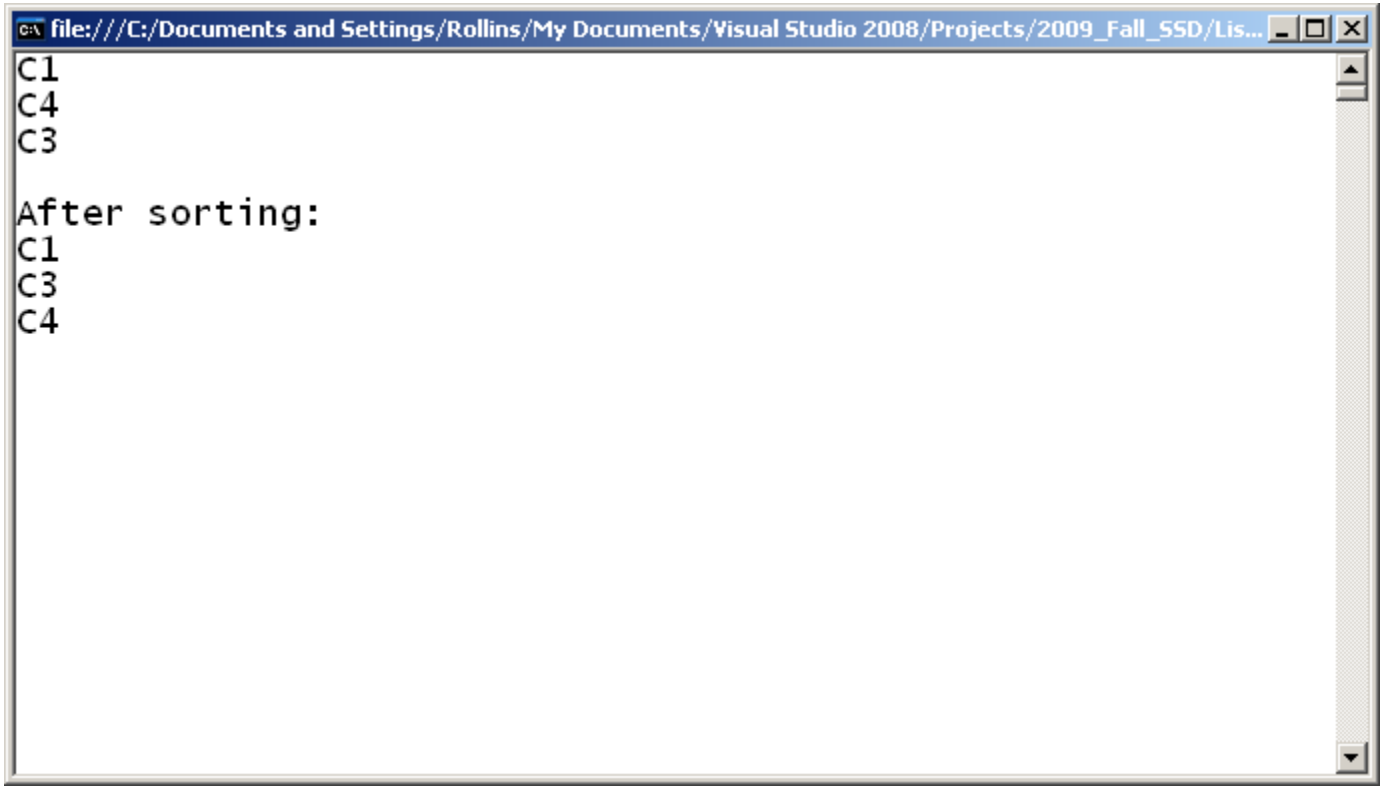
Implementing IComparable

```
class Circle : IComparable<Circle>
{
    private String name;
    private double radius = 0.0;
...
    public int CompareTo(Circle other)
    {
        if (this.radius < other.radius)
            return -1;
        else if (this.radius > other.radius)
            return 1;
        else
            return 0;
    }
```

Using Sort()


List_Demo - Microsoft Visual Studio
File Edit View Refactor Project Build Debug Data Tools Test Window Help
lblMessage
Hex
Start Page Output Circle.cs Program.cs*
List_Demo.Program
Main(string[] args)
8 static void Main(string[] args)
9 {
10 List<Circle> circles = new List<Circle>();
11
12 circles.Add(new Circle("C1", 1.0));
13 circles.Add(new Circle("C2", 2.0));
14 circles.Add(new Circle("C3", 3.0));
15
16 Circle c4 = new Circle("C4", 4.0);
17 circles.Insert(2, c4);
18
19 circles.RemoveAt(1);
20
21 for (int i = 0; i < circles.Count; ++i)
22 {
23 Console.WriteLine(circles[i].Name());
24 }
25
26 circles.Sort();
27
28 Console.WriteLine("\nAfter sorting:");
29 for (int i = 0; i < circles.Count; ++i)
30 {
31 Console.WriteLine(circles[i].Name());
32 }
33
34 Console.ReadLine();
35 }
Ready
Ln 28
Col 51
Ch 51
INS

Program Running



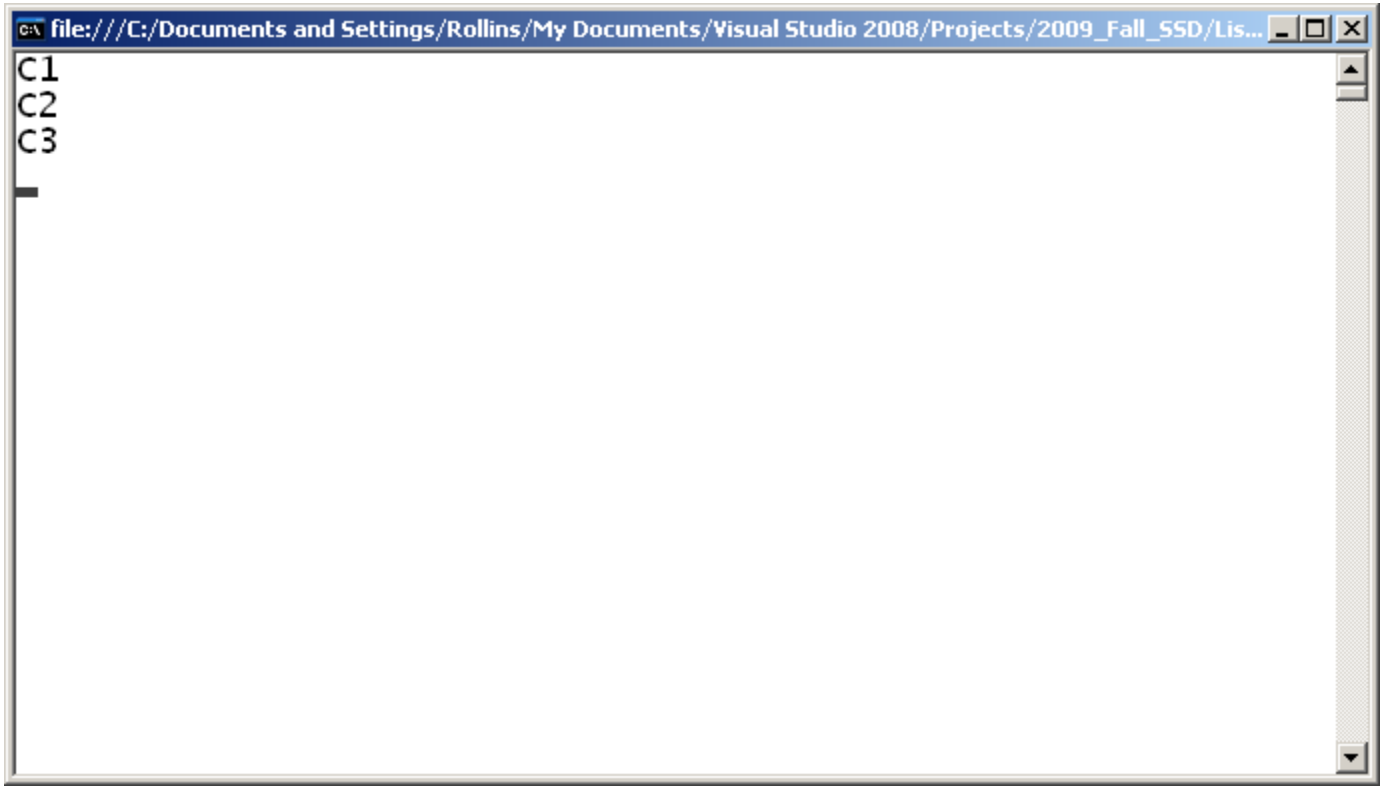
The Generic Queue

- Queue<T>
- Methods
  - Enqueue (T item)
    - Add an item to the end of the queue
  - T Dequeue()
    - Removes and returns object at the head of the queue
  - Clear(), Contains(), Peek(), ... many more

Queue<Circle> Example

```
static void Main(string[] args)
{
    Queue<Circle> circles = new Queue<Circle>();

    circles.Enqueue(new Circle("C1", 1.0));
    circles.Enqueue(new Circle("C2", 2.0));
    circles.Enqueue(new Circle("c3", 3.0));

    while (circles.Count > 0)
    {
        Circle c = circles.Dequeue();
        Console.WriteLine(c.Name());
    }
    Console.ReadLine();
}
```

Queue<Circle> Example Running



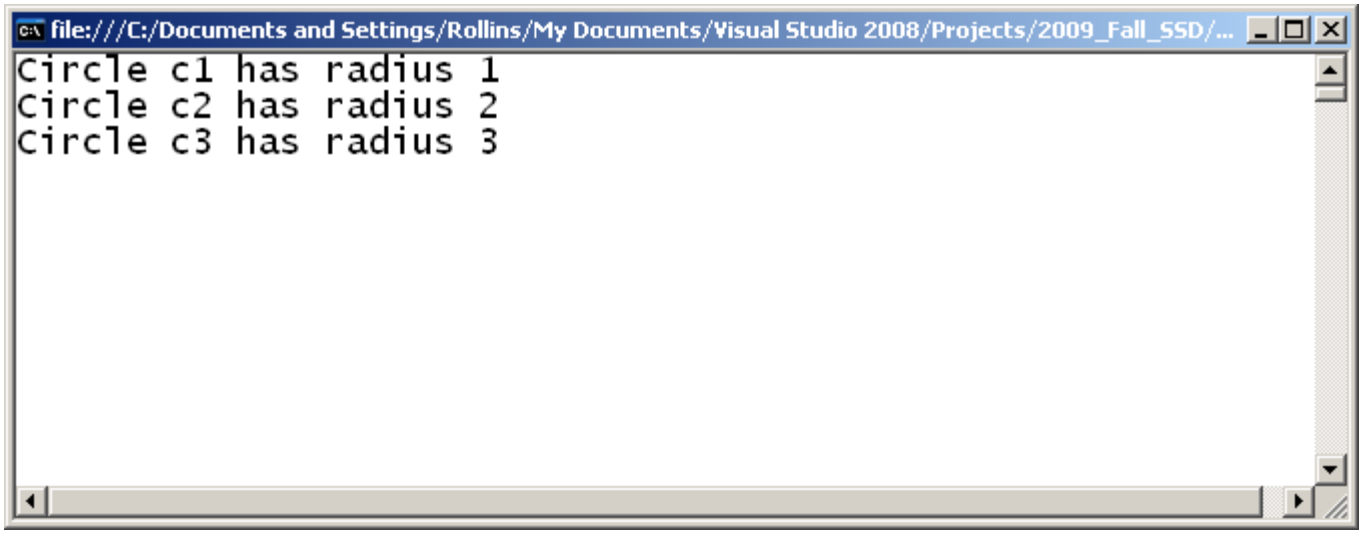
Dictionary<K,V>

- Stores (Key, Value) pairs
  - Class **`KeyValuePair<K,V>`**
- Template parameters
  - K is type of Key
  - V is type of Value

Dictionary<K,V> Example

```
static void Main(string[] args)
{
    Dictionary<String, Circle> circles =
        new Dictionary<String, Circle>();

    circles.Add("c1", new Circle("C1", 1.0));
    circles.Add("c2", new Circle("C2", 2.0));
    circles.Add("c3", new Circle("c3", 3.0));

    foreach (KeyValuePair<String, Circle> kvp in circles)
    {
        String k = kvp.Key;
        Circle c = kvp.Value;
        Console.WriteLine("Circle {0} has radius {1}",
                    k, c.Radius());
    }
    Console.ReadLine();
}
```

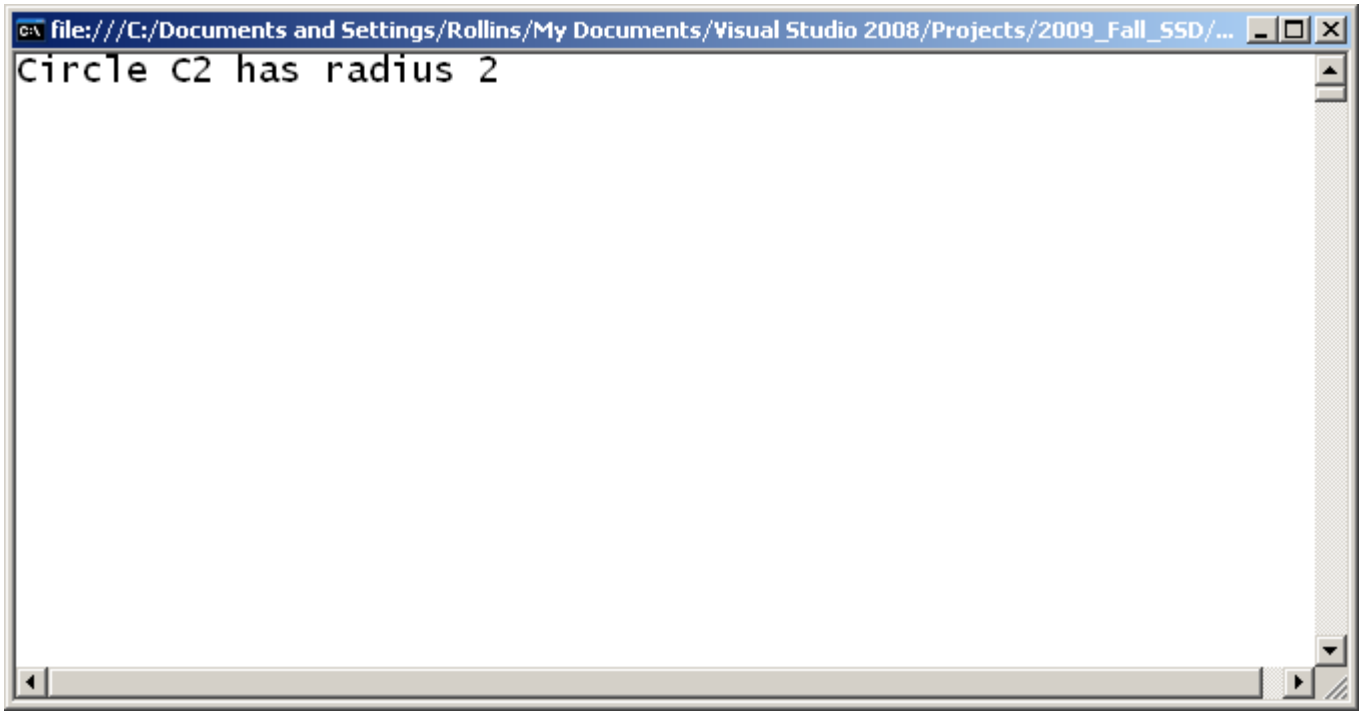
Dictionary<K,V> Example



Using Key as Indexer

```
static void Main(string[] args)
{
    Dictionary<String, Circle> circles =
        new Dictionary<String, Circle>();

    circles.Add("c1", new Circle("C1", 1.0));
    circles.Add("c2", new Circle("C2", 2.0));
    circles.Add("c3", new Circle("c3", 3.0));

    Circle c = circles["c2"];

    Console.WriteLine("Circle {0} has radius {1}",
        c.Name(), c.Radius());

    Console.ReadLine();
}
```

Indexer Example Running

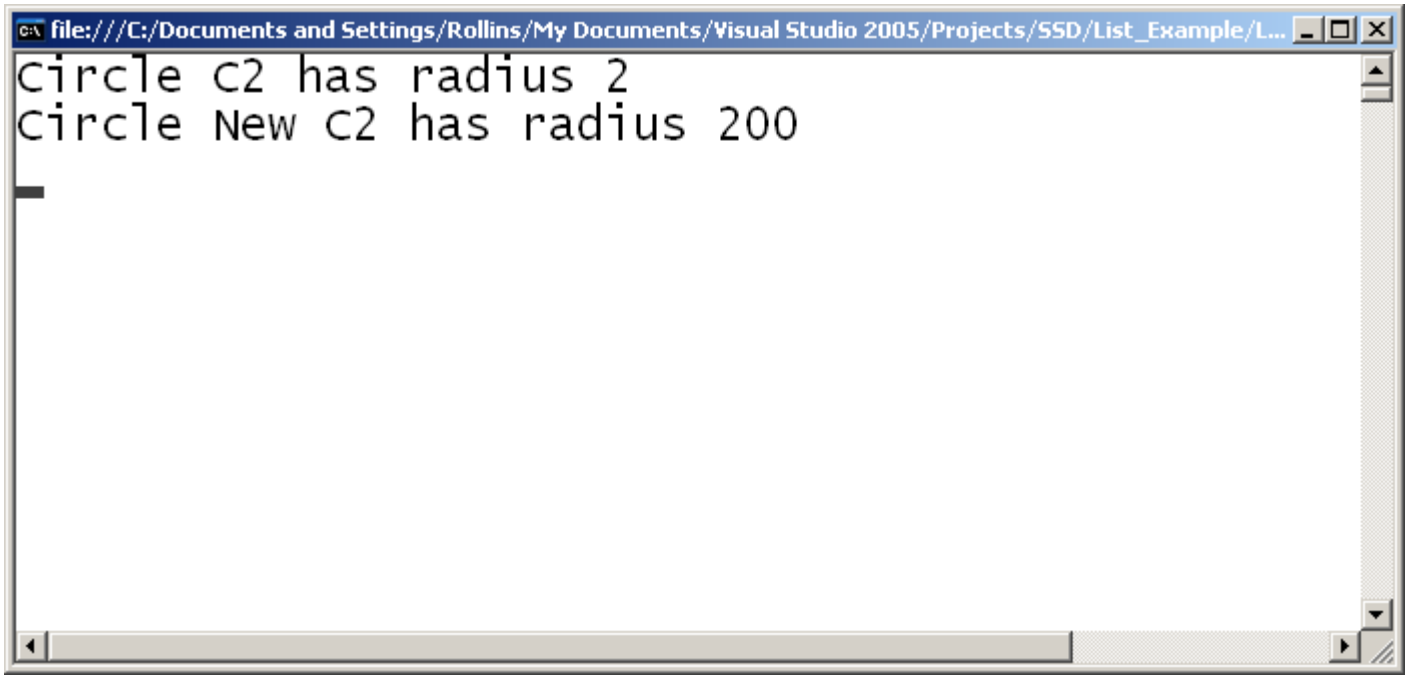


About Dictionary<K,V>

- Key class must have a compare for equality operation.
- Keys must be unique.
  - Attempting to Add an item with a key already in the Dictionary will result in an exception.
  - Can *set* entry with an existing key, using indexer notation.

```
static void Main(string[] args)
{
    Dictionary<String, Circle> circles =
        new Dictionary<String, Circle>();

    circles.Add("c1", new Circle("C1", 1.0));
    circles.Add("c2", new Circle("C2", 2.0));
    circles.Add("c3", new Circle("c3", 3.0));

    Circle c = circles["c2"];

    Console.WriteLine("Circle {0} has radius {1}",
        c.Name(), c.Radius());

    circles["c2"] = new Circle("New C2", 200.0);

    c = circles["c2"];

    Console.WriteLine("Circle {0} has radius {1}",
        c.Name(), c.Radius());

    Console.ReadLine();
}
```

Adding with Existing Key



Other Generic Container Classes

- Linked List (No array operations)
- SortedDictionary
- SortedList
- Stack

See .NET documentation