# SORTING A COLLECTION

# .SORT

#### List<T>.Sort Method ()

Sorts the elements in the entire List<T> using the default comparer.

Jody

Sava

Nikita

```
// The example displays the following output:
using System;
                                            List in unsorted order:
using System.Collections.Generic;
                                     //
                                               Samuel
                                                         Dakota Koani
                                                                         Saya Vanya Yiska Yuma
                                     //
public class Example
                                            List in sorted order:
                                     //
                                                              Koani Nikita Samuel
                                     //
                                               Dakota
                                                       Jody
  public static void Main()
     String[] names = { "Samuel", "Dakota", "Koani", "Saya", "Vanya",
                       "Yiska", "Yuma", "Jody", "Nikita" };
     var nameList = new List<String>();
     nameList.AddRange(names);
     Console.WriteLine("List in unsorted order: ");
     foreach (var name in nameList)
        Console.Write(" {0}", name);
     Console.WriteLine(Environment.NewLine);
     nameList.Sort();
     Console.WriteLine("List in sorted order: ");
     foreach (var name in nameList)
        Console.Write(" {0}", name);
     Console.WriteLine();
```

# .SORT

- This method uses the default comparer <u>Comparer<T>. Default</u> for type *T* to determine the order of list elements.
- The <u>Comparer<T>.Default</u> property checks whether type *T* implements the <u>IComparable<T></u> generic interface and uses that implementation, if available.
- If not, <u>Comparer<T>.Default</u> checks whether type *T* implements the <u>IComparable</u> interface.
- If type *T* does not implement either interface, <u>Comparer<T>.</u>

  <u>Default</u> throws an exception (an <u>InvalidOperationException</u>)

# .SORT

• Example

```
Where T = String
```

#### **Syntax**

```
C# C++ F# JScript VB

[SerializableAttribute]
public sealed class String : IComparable
```

#### **Syntax**

```
C# C++ F# VB

public interface IComparable<in T>
```

|            | Name             | Description  |
|------------|------------------|--|
| <b>≡</b> ₩ | CompareTo(T<br>) | Compares the current instance with another object of the same type and returns an integer that indicates whether the current instance precedes, follows, or occurs in the same position in the sort order as the other object. |

# OVER TO YOU ...

• Create a list of integers and print them in order.

## **SORTING A LIST OF CUSTOM OBJECTS**

- Say we have a list of student object ???
- Assume there is a class Student with fields name, knumber and mark
- If the collection of class objects is to be sortable based on the value of a single field e.g mark in the objects, then the class Student should implement the IComparable interface
- This will be the default comparison
- Sorting will be executed by list.Sort();

#### ICOMPARABLE<T>

- This interface is implemented by types whose values can be ordered or sorted and provides a strongly typed comparison method for ordering members of a generic collection object.
- For example, one number can be larger than a second number, and one string can appear in alphabetical order before another.
- It requires that implementing types define a single method, CompareTo(T), that indicates whether the position of the current instance in the sort order is before, after, or the same as a second object of the same type.
- Typically, the method is not called directly from developer code. Instead, it is called automatically by methods such as <u>List<T>.Sort()</u> and <u>Add</u>.

#### ICOMPARABLE<T>

- Typically, types that provide an IComparable<T> implementation also implement the <a href="IEquatable<T>">IEquatable<T></a> interface.
- The <u>lEquatable<T></u> interface defines the <u>Equals</u> method, which determines the equality of instances of the implementing type.
- The implementation of the <a href="CompareTo(T)">CompareTo(T)</a> method must return an <a href="Int32">Int32</a> that has one of three values, as shown in the following table.

| Value             | Meaning   |
|-------------------|---|
| Less than zero    | This object precedes the object specified by the CompareTo method in the sort order.  |
| Zero              | This current instance occurs in the same position in the sort order as the object specified by the CompareTo method argument. |
| Greater than zero | This current instance follows the object specified by the CompareTo method argument in the sort order.                        |

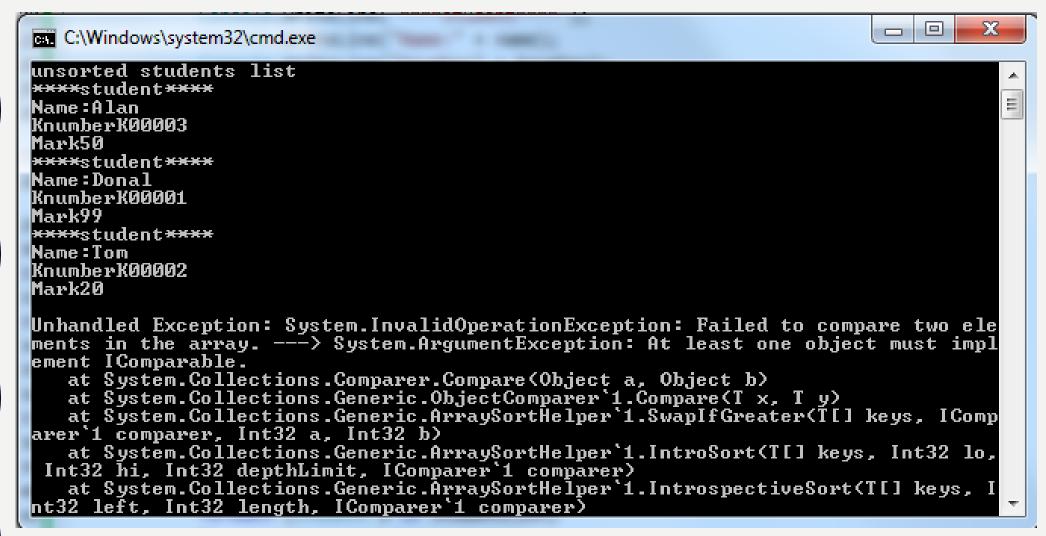
#### **EXAMPLE**

```
class Student
    2 references
    public string name { get; set; }
    2 references
    public string knumber { get; set; }
    2 references
    public int mark { get; set;}
    3 references
    public Student(String n, String kn, int m){
        name = n;
        knumber = kn;
        mark = m;
    1 reference
    public void print(){
        Console.WriteLine("****student****");
        Console.WriteLine("Name:" + name);
        Console.WriteLine("Knumber" + knumber);
        Console.WriteLine("Mark" + mark);
0 references
```

#### **EXAMPLE**

```
class Program
   0 references
    static void Main(string[] args)
       List<Student> studentList = new List<Student>();
        Student a = new Student("Alan", "K00003", 50);
        Student b = new Student("Donal", "K00001", 99);
        Student c = new Student("Tom", "K00002", 20);
        studentList.Add(a);
        studentList.Add(b);
        studentList.Add(c);
       Console.WriteLine("unsorted students list");
        foreach (Student s in studentList)
            s.print();
        // sort and print
        studentList.Sort();
```

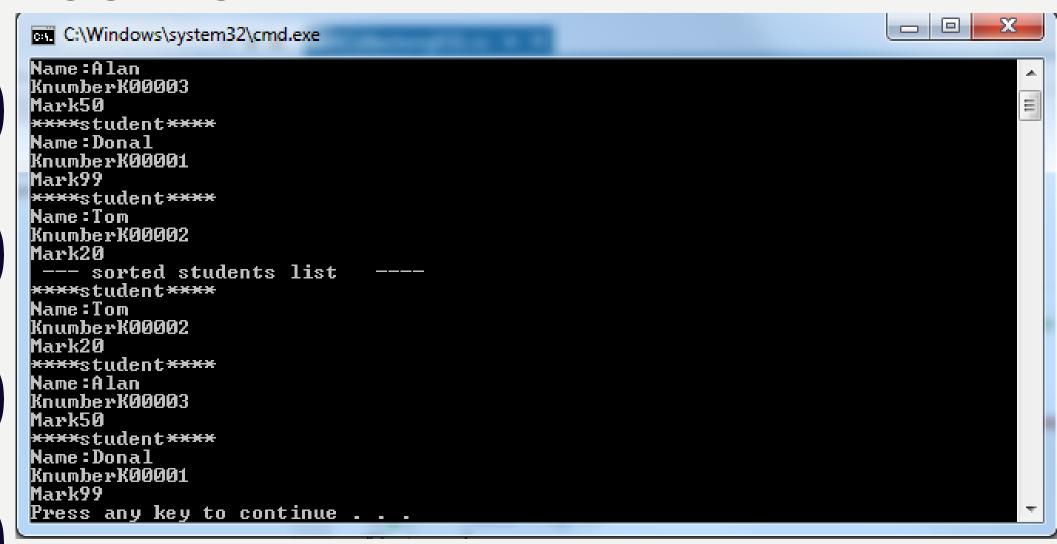
#### **ERROR**



```
class Student:IComparable
    2 references
    public string name { get; set; }
    2 references
    public string knumber { get; set; }
    4 references
    public int mark { get; set;}
    3 references
    public Student(String n, String kn, int m){
        name = n;
        knumber = kn;
        mark = m;
    2 references
    public void print(){
        Console.WriteLine("****student****");
        Console.WriteLine("Name:" + name);
        Console.WriteLine("Knumber" + knumber);
        Console.WriteLine("Mark" + mark);
    0 references
    public int CompareTo(Object obj)
         if (obj is Student){
            Student s = obj as Student;
            return (this.mark - s.mark);
            //return (s.mark - this.mark); //to reverse sort order
         else
             throw new ArgumentException("Object to compare is not a Student object");
```

```
class Program
   0 references
   static void Main(string[] args)
        List<Student> studentList = new List<Student>();
        Student a = new Student("Alan", "K00003", 50);
        Student b = new Student("Donal", "K00001", 99);
        Student c = new Student("Tom", "K00002", 20);
        studentList.Add(a);
        studentList.Add(b);
        studentList.Add(c);
        Console.WriteLine("unsorted students list");
        foreach (Student s in studentList)
            s.print();
        // sort and print
        studentList.Sort();
        Console.WriteLine(" --- sorted students list ----");
        foreach (Student s in studentList)
            s.print();
```

## OUTPUT

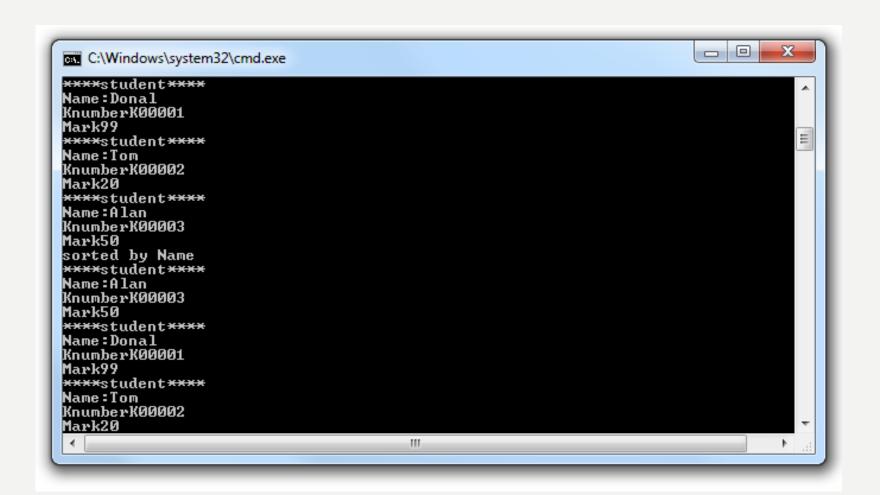


# bbb

• If the collection of class objects is to be sortable based on an alternative field • e.g. knumber in the student class and not or student marks

```
public class StudentComparerKnumber : IComparer<Student>{
1 reference
public int Compare(Student S1, Student S2)
    return String.Compare(S1.knumber,S2.knumber);
1 reference
public class StudentComparerName : IComparer<Student>
    1 reference
    public int Compare(Student S1, Student S2)
        return String.Compare(S1.name, S2.name);
```

```
//sort by knumber
studentList.Sort(new StudentComparerKnumber());
Console.WriteLine("sorted by knumber");
foreach (Student s in studentList)
    s.print();
//sort by name
studentList.Sort(new StudentComparerName());
Console.WriteLine("sorted by Name");
foreach (Student s in studentList)
    s.print();
```



## PP OUTPUT ...

```
// sort and print
studentList.Sort();

Console.WriteLine(" --- sorted students ----");
foreach (Student s in studentList)
{
    s.print();
}
```

#### ANOTHER EXAMPLE

```
public class Box : IComparable<Box>
    public Box(int h, int l, int w)
        this.Height = h;
        this.Length = 1;
        this.Width = w;
    public int Height { get; private set; }
    public int Length { get; private set; }
    public int Width { get; private set; }
    public int CompareTo(Box other)
       // Compares Height, Length, and Width.
        if (this.Height.CompareTo(other.Height) != 0)
            return this.Height.CompareTo(other.Height);
        else if (this.Length.CompareTo(other.Length) != 0)
            return this.Length.CompareTo(other.Length);
        else if (this.Width.CompareTo(other.Width) != 0)
            return this.Width.CompareTo(other.Width);
        else
            return 0;
```

```
public class BoxLengthFirst : Comparer<Box>
   // Compares by Length, Height, and Width.
   public override int Compare(Box x, Box y)
       if (x.Length.CompareTo(y.Length) != 0)
            return x.Length.CompareTo(y.Length);
        else if (x.Height.CompareTo(y.Height) != 0)
            return x.Height.CompareTo(y.Height);
        else if (x.Width.CompareTo(y.Width) != 0)
            return x.Width.CompareTo(y.Width);
        else
            return 0;
```

```
static void Main(string[] args)
   List<Box> Boxes = new List<Box>();
    Boxes.Add(new Box(4, 20, 14));
    Boxes.Add(new Box(12, 12, 12));
    Boxes.Add(new Box(8, 20, 10));
    Boxes.Add(new Box(6, 10, 2));
    Boxes.Add(new Box(2, 8, 4));
    Boxes.Add(new Box(2, 6, 8));
    Boxes.Add(new Box(4, 12, 20));
    Boxes.Add(new Box(18, 10, 4));
    Boxes.Add(new Box(24, 4, 18));
    Boxes.Add(new Box(10, 4, 16));
    Boxes.Add(new Box(10, 2, 10));
    Boxes.Add(new Box(6, 18, 2));
    Boxes.Add(new Box(8, 12, 4));
    Boxes.Add(new Box(12, 10, 8));
    Boxes.Add(new Box(14, 6, 6));
    Boxes.Add(new Box(16, 6, 16));
    Boxes.Add(new Box(2, 8, 12));
    Boxes.Add(new Box(4, 24, 8));
    Boxes.Add(new Box(8, 6, 20));
    Boxes.Add(new Box(18, 18, 12));
   // Sort by an Comparer<T> implementation that sorts
   // first by the length.
    Boxes.Sort(new BoxLengthFirst());
    Console.WriteLine("H - L - W");
    Console.WriteLine("======");
    foreach (Box bx in Boxes)
        Console.WriteLine("{0}\t{1}\t{2}",
            bx.Height.ToString(), bx.Length.ToString(),
            bx.Width.ToString());
```