The IComparable Interface

Array.Sort()----This method uses the QuickSort algorithm.

The System.IComparable interface specifies a behavior that allows an object to be sorted based on some

specified key. Here is the formal definition:

// This interface allows an object to specify its

// relationship between other like objects.

public interface IComparable

{

int CompareTo(object o);

}

**Note** The generic version of this interface (IComparable<T>) provides a more type-safe manner to handle

comparisons between objects.

Let’s assume you have a new Console Application named ComparableCar that defines the following

updated Car class (notice that we have basically just added a new property to represent a unique ID for

each car and a modified constructor):

public class Car

{

...

public int CarID {get; set;}

public Car(string name, int currSp, int id)

{

CurrentSpeed = currSp;

PetName = name;

CarID = id;

}

...

}

Now assume you have an array of Car objects as follows:

static void Main(string[] args)

{

Console.WriteLine("\*\*\*\*\* Fun with Object Sorting \*\*\*\*\*\n");

// Make an array of Car objects.

Car[] myAutos = new Car[5];

myAutos[0] = new Car("Rusty", 80, 1);

myAutos[1] = new Car("Mary", 40, 234);

myAutos[2] = new Car("Viper", 40, 34);

myAutos[3] = new Car("Mel", 40, 4);

myAutos[4] = new Car("Chucky", 40, 5);

Console.ReadLine();

}

The System.Array class defines a static method named Sort(). When you invoke this method on an

array of intrinsic types (int, short, string, etc.), you are able to sort the items in the array in

numeric/alphabetic order, as these intrinsic data types implement IComparable. However, what if you

were to send an array of Car types into the Sort() method as follows?

// Sort my cars? Not yet!

Array.Sort(myAutos);

The System.Array class defines a **static** method named **Sort().** When you invoke this method on an

array of intrinsic types (int, short, string, etc.), you are able to sort the items in the array in

numeric/alphabetic order, **as these intrinsic data types implement IComparable**. However, what if you

were to send an array of Car types into the Sort() method as follows?

// Sort my cars? Not yet!

Array.Sort(myAutos);

If you run this test, you would get a runtime exception, as the Car class does not support the

necessary interface. When you build custom types, you can **implement IComparable** to allow arrays of

your types to be sorted. When you flesh out the details of CompareTo(), it will be up to you to decide what

the baseline of the ordering operation will be. For the Car type, the internal CarID seems to be the logical candidate:

// The iteration of the Car can be ordered

// based on the CarID.

public class Car : IComparable

{

...

// IComparable implementation.

int IComparable.CompareTo(object obj)

{

Car temp = obj as Car;

if (temp != null)

{

if (this.CarID > temp.CarID)

return 1;

if (this.CarID < temp.CarID)

return -1;

else

return 0;

}

else

throw new ArgumentException("Parameter is not a Car!");

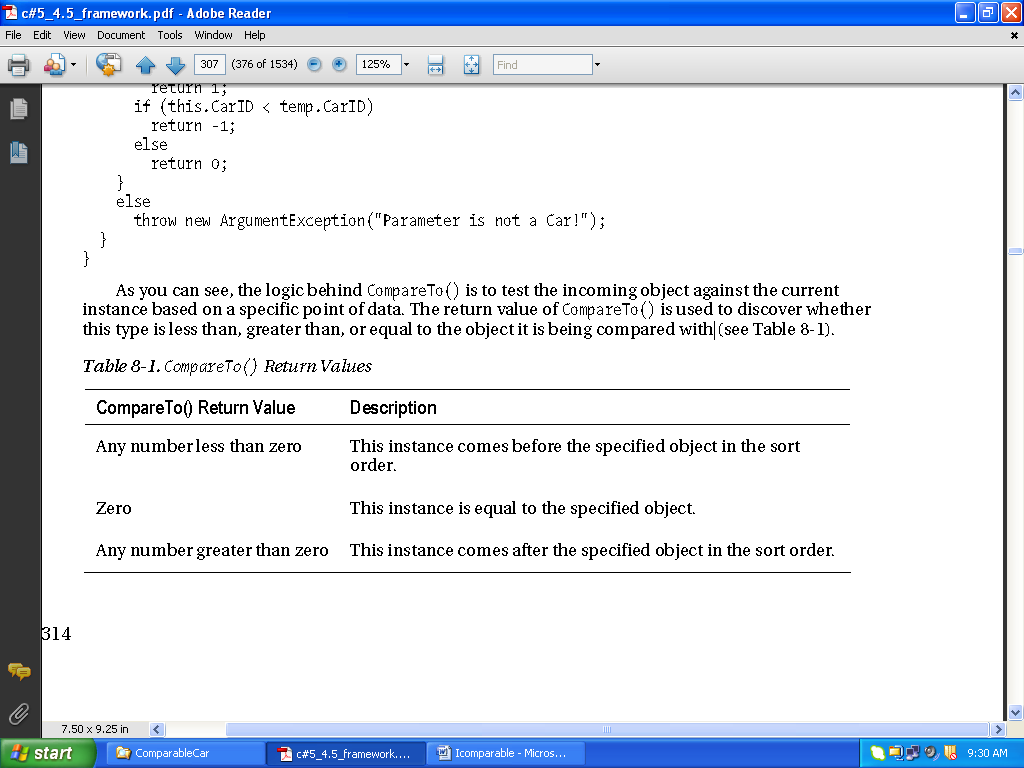
}

}

As you can see, the logic behind CompareTo() is to test the incoming object against the current

instance based on a specific point of data. The return value of CompareTo() is used to discover whether

this type is less than, greater than, or equal to the object it is being compared with



We can streamline the previous implementation of CompareTo() given the fact that the C# int data

type (which is just a shorthand notation for the CLR System.Int32) implements IComparable. You could

implement the Car’s CompareTo() as follows:

int IComparable.CompareTo(object obj)

{

Car temp = obj as Car;

if (temp != null)

return this.CarID.CompareTo(temp.CarID);

else

throw new ArgumentException("Parameter is not a Car!");

}

In either case, so that your Car type understands how to compare itself to like objects, you can write

the following user code:

// Exercise the IComparable interface.

static void Main(string[] args)

{

// Make an array of Car objects.

...

// Display current array.

Console.WriteLine("Here is the unordered set of cars:");

foreach(Car c in myAutos)

Console.WriteLine("{0} {1}", c.CarID, c.PetName);

// Now, sort them using IComparable!

Array.Sort(myAutos);

Console.WriteLine();

// Display sorted array.

Console.WriteLine("Here is the ordered set of cars:");

foreach(Car c in myAutos)

Console.WriteLine("{0} {1}", c.CarID, c.PetName);

Console.ReadLine();

}

Here is the output from the previous Main() method:

\*\*\*\*\* Fun with Object Sorting \*\*\*\*\*

Here is the unordered set of cars:

1 Rusty

234 Mary

34 Viper

4 Mel

5 Chucky

Here is the ordered set of cars:

1 Rusty

4 Mel

5 Chucky