Purpose: This program explores sorting using comparers and the natural ordering of classes.

It is common to want to be able to sort objects of the classes we create into different orders using the sort methods of the **Array** class or the **List** class. In order to define the natural order for objects of a class, one must implement the **IComparable** interface. This will allow objects to be compared to one another to determine if one is less than, greater than, or equal to another. To create additional sort orders (beyond this default order), one must create instances of the **IComparer** interface and use them with the sort methods to establish a different sorting order. Typically, this is done by extending class **Comparer** instead of directly implementing the **IComparer** class. You may refer to the following article from MSDN for more information on these interfaces: Comparer(T) Class Example.

For another example, I have created a project that demonstrates these interfaces being used with the **Time2** class from Chapter 10 of the text. It is attached to this program description in the file *SortingDemo.zip*. I recommend that you follow this example closely when solving this assignment.

- Using the **Parcel** hierarchy from your instructor's solution to Program 1A, rewrite the **Parcel** class to support the **IComparable<T>** interface.
- Make the default sort order for parcels be ascending by cost.
- In addition, create another class that extends the **Comparer<T>** class (thus implementing the **IComparer<T>** interface) and will allow parcels to be put in descending order by destination zip code.
- Write a simple test program that creates a **List** of at least 10 **Parcel** objects (use all the various item types).
- Display the list of items in the original order then sort the list using the default ordering (ascending by cost).
- Display the list again.
- Next, sort the list using your comparer to create descending by destination zip code and display the newly ordered list.
- Your test application can be a simple console application (as in the attached *SortingDemo*).
- **Extra Credit** - For 10 points of extra credit, for create (and test) a new sort order that will first order by Parcel type (ascending) and then cost (descending) within each grouping.
- So, all **GroundPackage** items should be together followed by the **Letter** items, then **NextDayAirPackage** items, etc.
- You will create another class that extends the **Comparer<T>** class to create this new sort order. The comparer that you create should allow the class hierarchy to be extended without requiring modification. It should not matter what the concrete classes of **Parcel** hierarchy actually are.
- So, in other words, don't hard code **is** tests for each class type. **Hint**: You may want to explore what method **GetType().ToString()** returns (as we used in Prog1B). It may prove very helpful for creating your groupings. BTW, this is a multi-level sort ordering, much like the **Time2** demo sorted first by hour, then minute, and finally second. CompareTo() Method in Example's Structure… If Else Hour = Ascending Type, Minute = Descending Cost

Be sure to add appropriate comments in your code for each file, including your **Grading ID** (not name nor student ID number), program number, due date, course section, and description of each file's class. Each variable used in your program needs a comment describing its purpose. These requirements are expected for every program and are listed in the syllabus. Preconditions and postconditions are now required, as well. So, for each constructor, method, get property, and set property a pair of comments describing the precondition and postcondition must be provided. Please review the PowerPoint presentation (under Course Documents) for further details about preconditions and postconditions.

As with our labs, I'm asking you to upload a compressed ZIP archive of the entire project. The steps for doing this will vary somewhat based on the ZIP utility being used. Before you upload this .ZIP file, it's a good idea to make sure that everything was properly zipped. Make sure your code is present and you can run your file.

Once you have verified everything, return to the *Assignments, Programs* area of Blackboard. Click on "Program 4" and then click on *View Assessment* and the assignment description will appear. Click *Add Content* button to browse the system for your file. Click on *Insert Content* icon (+) at right-end of nav-bar, and then choose *Insert Local Files* from the submenu.

Browse to the location of your .ZIP file and select it. Note, multiple files may be attached by repeating the Insert Content sequence. For this assignment, we just need the "Prog4.zip" file. Make sure everything is correct in the form and then click *Submit* to complete the assignment and upload your file to be graded.