Lab Exercise: Advanced Operations with C# Generic Collections

This lab exercise is designed to deepen your understanding of C# generic collections by exploring their more advanced features and capabilities. Through these tasks, you'll learn to manipulate collections in powerful ways, improving the efficiency and maintainability of your C# applications.

Objectives:

- Gain a deeper understanding of generic collections in C# and their advanced features.

- Explore the use of custom comparers for sorting and searching within generic collections.

- Practice converting between collection types and manipulating collections with LINQ.

- Implement custom extension methods for generic collections to add new functionalities.

Prerequisites:

- Basic knowledge of C# and familiarity with generic collections (`List<T>`, `Dictionary<TKey, TValue>`, etc.).

- Visual Studio or a similar IDE that supports C# development.

Tasks:

Task 1: Custom Comparers for Sorting

1. Define a `Person` class with properties `Name` (string) and `Age` (int).

2. Create a `List<Person>` and populate it with several `Person` objects.

3. Implement a custom comparer that sorts `Person` objects by age in descending order. Use this comparer to sort your list and display the sorted list to the console.

Task 2: Converting Collections

1. Using the same `List<Person>` from Task 1, convert it to a `Dictionary<int, Person>` where the key is a unique identifier for each person.

2. Demonstrate how to efficiently convert a `Dictionary<int, Person>` back to a `List<Person>`, extracting only persons over a certain age.

Task 3: Implementing Custom Extension Methods

1. Create a static class `CollectionExtensions` that will store your extension methods.

Trainer: VenkatShiva.Reddy@gmail.com

2. Implement an extension method `Shuffle<T>` for `IList<T>` that randomly shuffles the elements within the list.

3. Implement an extension method `DeepCopy<T>` for `List<T>` where `T` is a class with a public parameterless constructor. This method should create a deep copy of the list and its elements.

Bonus Challenge:

- Create a generic method `ConvertAll<TSource, TDestination>` that converts a collection of one type to another type, given a conversion function, and demonstrate its use by converting a `List<int>` to a `List<string>`.

Tips:

- For the custom comparer, implement the `IComparer<T>` interface.

- When converting collections, explore the use of `ToDictionary` and `ToList` LINQ methods.

- For the custom extension methods, remember that extension methods must be defined in a static class and the first parameter must include the `this` modifier.