Lab7: Collections and Generics

Name: Gondrala Mani Sai
Id num: 2100031545

To create generic type **CustomArray** – one dimensional array with random index range

CustomArray is a collection – array of random type values with fixed length and with original index that is specified by user.

Example 1: array of 20 elements length, array values – symbols, index starts with 18.
Example 2: array of 10 elements length, array values – objects of class Animals, index starts with -5

Values of random type can be located in array, custom first index and the number of elements in array should be specified while creating. The length and range of indexes cannot be changed after creating. Values of array elements can be set while creating the array and later with the help of indexer.

Initial and finite index, array length, array elements in the form of standard array Array that starts with 0 can be obtained from array.

CustomArray should be able to use operator foreach and other constructions that are oriented to the presence of numerator in class.

The task has two levels of complexity: Low and Advanced.

Low level tasks require implementation of the following functionality:

- Creating of empty user array and the one based on standard existing
- Receiving first, last indexes, length, values in form of standard array with 0.
- Access to writing and reading element based on predetermined correct index

Advanced level tasks require implementation of the following functionality:

- All completed tasks of Low level
- Creating of array based on values params
- Generating exceptions, specified in xml-comments to class methods
- Receiving numerator from array for operator foreach

Solution:

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
```

```
namespace lab_7
{
    public class CustomArray<T> : IEnumerable<T>
    {
        private T[] array;
        private int startIndex;

        // Constructor to create an empty user array
        public CustomArray(int length, int startIndex)
        {
            if (length <= 0)
                throw new ArgumentException("Length must be greater than zero.");

            this.array = new T[length];
            this.startIndex = startIndex;
        }

        // Constructor to create an array based on values params
        public CustomArray(int startIndex, params T[] values)
        {
            if (values == null)
                throw new ArgumentNullException(nameof(values));

            this.array = values;
            this.startIndex = startIndex;
        }

        // Method to get the first index
        public int GetFirstIndex()
        {
            return startIndex;
        }

        // Method to get the last index
        public int GetLastIndex()
        {
            return startIndex + array.Length - 1;
        }

        // Method to get the length of the array
        public int GetLength()
        {
            return array.Length;
        }

        // Indexer to access elements based on predetermined correct index
        public T this[int index]
        {
            get
            {
                ValidateIndex(index);
                return array[index - startIndex];
            }
            set
            {
                ValidateIndex(index);
                array[index - startIndex] = value;
            }
        }

        // Method to get values in form of standard array with 0
        public T[] GetStandardArray()
        {
```

```
            return array;
        }

        // Method to validate the index
        private void ValidateIndex(int index)
        {
            if (index < startIndex || index >= startIndex + array.Length)
                throw new IndexOutOfRangeException($"Index '{index}' is out of
range.");
        }

        // Implementation of IEnumerable<T> interface for using operator foreach
        public IEnumerator<T> GetEnumerator()
        {
            foreach (T item in array)
            {
                yield return item;
            }
        }

        // Implementation of IEnumerable interface for using operator foreach
        IEnumerator IEnumerable.GetEnumerator()
        {
            return GetEnumerator();
        }
    }

}
```

Output: