**Custom List Class Project (out of 120 points)**

**User Stories**

The built-in List<T> class is a generic class that acts as a wrapper over the array class. **You cannot use built-in List or Array methods.**

**(20 points)** As a developer, I want to use Test Driven Development (TDD), so that I can write tests for my methods to pass to ensure proper functionality within my application.

**(10 points):** As a developer, I want to use a custom-built list class that stores its values in an array, so that I can store any data type in my collection.

**(10 points):** As a developer, I want the ability to add an object to an instance of my custom-built list class.

**(10 points):** As a developer, I want the ability to remove an object from an instance of my custom-built list class.

**(10 points):** As a developer, I want the custom list class to be iterable.

**(10 points):** As a developer, I want to be able to override the ToString method that converts the contents of the custom list to a string.

**(10 points):** As a developer, I want to be able to overload the + operator, so that I can add two instances of the custom list class together.

**(10 points):** As a developer, I want to be able to overload the – operator, so that I can subtract one instance of a custom list class from another instance of a custom list class.

**(10 points):** As a developer, I want a Count property implemented on the custom-built list class, so that I can get a count of the number of elements in my custom list class instance.

**(10 points):** As a developer, I want the ability to Enumerable.Zip() two custom list class instances together.

**(10 points):** As a developer, I want to use C# best practices, SOLID design principles, and good naming conventions on the project.

**(Bonus 5 points):** As a developer, I want the ability to sort an instance of my custom-built list class. To be eligible for the bonus points, you may not use Array.Sort() that is already built in and you must tell us what sorting algorithm you used.

**NOTICE: get your unit tests (test methods) checked off by an instructor before you begin writing your methods to ensure you are on the correct path.**

**public class GenericPrint<T>**

**{**

**T value;**

**public GenericPrint(T test)**

**{**

**value = test;**

**}**

**public void Print()**

**{**

**Console.WriteLine(value);**

**}**

**}**

**// Use the generic class Test with an int type parameter.**

**GenericPrint<int> genericTestInt = new GenericPrint<int>(16);**

**Console.WriteLine("Generic Class Test used with an Int");**

**genericTestInt.Print();**

**// Use the generic class Test with a string type parameter.**

**GenericPrint<string> genericTestString = new GenericPrint<string>("Packers");**

**Console.WriteLine("Generic Class Test used with a string");**

**genericTestString.Print();**

**public class GenericSwap**

**{**

**public void SwapFunction<T>(ref T item1, ref T item2)**

**{**

**T temp;**

**temp = item1;**

**item1 = item2;**

**item2 = temp;**

**}**

**}**

**//Use the generic swap method**

**GenericSwap swap = new GenericSwap();**

**Console.WriteLine("Using Generic Swap Method");**

**int FirstIntForSwap = 4;**

**int SecondIntForSwap = 12;**

**Console.WriteLine("First Int For Swap Value: {0} \r\n", FirstIntForSwap);**

**Console.WriteLine("Second Int For Swap Value: {0} \r\n", SecondIntForSwap);**

**Console.WriteLine("Using Generic Swap Method to Swap Ints");**

**swap.SwapFunction<int>(ref FirstIntForSwap, ref SecondIntForSwap);**

**Console.WriteLine("After the Int Swap ");**

**Console.WriteLine("First Int For Swap Value: {0} \r\n", FirstIntForSwap);**

**Console.WriteLine("Second Int For Swap Value: {0} \r\n", SecondIntForSwap);**

**string FirstStringForSwap = "This is the first string";**

**string SecondStringForSwap = "This is the second string";**

**Console.WriteLine("Before the String Swap ");**

**Console.WriteLine("First String For Swap Value: {0} \r\n", FirstStringForSwap);**

**Console.WriteLine("Second String For Swap Value: {0} \r\n", SecondStringForSwap);**

**swap.SwapFunction<string>(ref FirstStringForSwap, ref SecondStringForSwap);**

**Console.WriteLine("After The String Swap");**

**Console.WriteLine("First String for Swap Value: {0} \r\n", FirstStringForSwap);**

**Console.WriteLine("Second String for Swap Value: {0} \r\n", SecondStringForSwap);**

//OverloadAddOperator obj1 = new OverloadAddOperator();

//obj1.Num1 = 10;

//obj1.Str1 = "OPERATOR";

//OverloadAddOperator obj2 = new OverloadAddOperator();

//obj2.Num1 = 10;

//obj2.Str1 = "OVERLOAD!";

//OverloadAddOperator obj3 = new OverloadAddOperator();

//obj3 = obj1 + obj2;

//public static void Main(string[] args)

//{

// OverloadAddOperator obj1 = new OverloadAddOperator();

// obj1.Num1 = 10;

// obj1.Str1 = "OPERATOR";

// OverloadAddOperator obj2 = new OverloadAddOperator();

// obj2.Num1 = 10;

// obj2.Str1 = "OVERLOAD!";

// OverloadAddOperator obj3 = new OverloadAddOperator();

// obj3 = obj1 + obj2;

//}

//public class OverloadAddOperator

//{

// public int Num1 = 0;

// public string Str1 = "";

// public static OverloadAddOperator operator +(OverloadAddOperator obj1, OverloadAddOperator obj2)

// {

// OverloadAddOperator obj3 = new OverloadAddOperator();

// obj3.Num1 = obj3.Num1 + obj2.Num1;

// obj3.Str1 = obj1.Str1 + obj2.Str1;

// return obj3;

// }

//}

// //public static void OverrideToStringConversion()

// {

// int myNumber = 1;

// Console.WriteLine(myNumber.ToString());

// CustomList<int> myValue = new CustomList<int>();

// myValue.value1 = 1;

// myValue.value2 = 2;

// Console.WriteLine(myValue.ToString());

// }

// Console.ReadKey();