4. Übungsblatt, Thema: Erweiterbare Listen.

Einleitung:

Um erweiterbare Datenstruktur zu verstehen ist am Besten selber diese zu implementieren. Wir wollen eine Klasse namens *TextCollection* schreiben. Diese Klasse soll in der Lage sein, beliebige viele Elemente zu verwalten. Dabei verwaltet es strings als Werte. Wenn von einer Liste gesprochen wird, dann ist damit eine Instance der Klasse TextCollection gemeint.

1. Implementierung von Method Append und GetArray

Beschreibung:

Zu Append:

Diese Methode soll einen string als neuen Wert der List entgegen nehmen. Nach der Ausführung dieser Methode soll sich dieser Wert am Ende der List befinden. Dabei soll diese Liste nicht wie ein Array von der Größe beschränkt sein. Man kann also beliebig viele Werte zu dieser List hinzufügen.

Zu GetArray:

Diese Methode soll ein Array zurückgeben. Dieses Array soll alle Werte aus der Liste enthalten und nur so groß sein wie die Anzahl aller Elemente aus der Liste. Falls die Liste leer ist soll auch das Array leer sein.

Tipp:

- Es ist ratsam eine innere Klasse der Klasse TextCollection zu schreiben. Diese innere Klasse sollte ein Element der Liste verwalten. Damit kann man dan auch Vorgänger und Nachfolger eines Elements in der Liste mit abspeichern.
- Es kann sehr hilfreich sein den Anfang der Liste als Variable zu speichern.

Declaration:

```
public void Append(string newValue)
public String[] GetArray()
```

Wichtig nach der Implementierung der 1. Aufgabe:

Füge folgende Code ein in die Klasse, wo auch der Test Code aus dieses Aufgabenblatt liegt.

```
static void PrintList(TextCollection collection)
{
    string[] array = collection.GetArray();

    if (array != null)
    {
        int count = 0;
        foreach (string element in collection.GetArray())
        {
            Console.WriteLine($"Element {count}: {element}");
            count++;
        }
        Console.WriteLine(new String('=', 30));
    }
```

```
    else
    {
        Console.WriteLine("List is empty");
    }
}
```

Test Program:

```
Console.WriteLine("Testing Appending");
TextCollection collection = new TextCollection();
collection.Append("0");
collection.Append("1");
collection.Append("2");

PrintList(collection);
```

Erwarteter Output:

Testing Appending
Element 0: 0
Element 1: 1
Element 2: 2
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

2. Implementierung von Method Pop

Beschreibung:

Diese Methode soll das letzte Element aus der Liste zurückgeben. Nach der Ausführung der Methode soll das letzte Element auch aus der Liste entfernt werden.

Randbedingung:

Falls sich kein Element mehr in der Liste befindet, soll der Wert null zurückgegeben werden. Dabei soll keine Exception geworfen werden.

Declaration:

```
public string Pop()
```

Test Program:

```
Console.WriteLine("Testing Pop");
TextCollection collection = new TextCollection();
Console.WriteLine($"collection.Pop() = {collection.Pop()}");
PrintList(collection);

collection.Append("0");
collection.Append("1");
collection.Append("2");

Console.WriteLine($"collection.Pop() = {collection.Pop()}");
PrintList(collection);

Console.WriteLine($"collection.Pop() = {collection.Pop()}");
Console.WriteLine($"collection.Pop() = {collection.Pop()}");
PrintList(collection);

Console.WriteLine($"collection.Pop() = {collection.Pop()}");
PrintList(collection);
```

Erwarteter Output:

Testing Pop
collection.Pop() =
List is empty
collection.Pop() = 2
Element 0: 0
Element 1: 1
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
collection.Pop() = 1
collection.Pop() = 0
List is empty
collection.Pop() =
List is empty

3. Implementierung von Property Count

Beschreibung:

Dieser Property ist ein reiner Getter. Dieser Getter gibt die Anzahl aller Elemente der Liste zurück.

Declaration:

```
public int Count
```

Test Program:

```
Console.WriteLine("Testing Count");
TextCollection collection = new TextCollection();
Console.WriteLine($"collection.Count = {collection.Count}");
collection.Append("0");
collection.Append("1");
collection.Append("2");
Console.WriteLine($"collection.Count = {collection.Count}");
collection.Pop();
Console.WriteLine($"collection.Count = {collection.Count}");
```

Erwarteter Output:

Testing Count
collection.Count = 0
collection.Count = 3
collection.Count = 2

4. Implementierung von einem Indexer

Beschreibung:

Wie beim Array soll man in der Lage sein, einen Wert in der Liste mit Hilfe eines Index abzurufen oder zu verändern können.

Randbedingung: Falls die Liste leer ist, der Parameter als Index negative ist oder der Index größer als alle Elemente der Liste ist, soll folgendes passieren: Null als Wert soll zurück gegeben werden, falls ein Wert abgerufen wird. Es soll nichts passieren und die Liste soll unverändert sein, falls ein Wert verändert werden soll. Dabei soll keine Exception geworfen werden.

Tipp:

Hier ein Link wie Indexer funktionieren https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/indexers/

Declaration:

```
public string this[int index]
```

Test Program:

```
Console.WriteLine("Testing Indexer");
TextCollection collection = new TextCollection();

collection.Append("0");
collection.Append("1");
collection.Append("2");

Console.WriteLine($"collection[-2] = {collection[-2]}");
Console.WriteLine($"collection[0] = {collection[0]}");
Console.WriteLine($"collection[1] = {collection[1]}");
Console.WriteLine($"collection[2] = {collection[2]}");
Console.WriteLine($"collection[3] = {collection[3]}");

collection[-1] = "-10";
collection[1] = "-1";
collection[3] = "10";

PrintList(collection);
```

Erwarteter Output:

Testing Indexer
collection[-2] =
collection[0] = 0
collection[1] = 1
collection[2] = 2
collection[3] =
Element 0: 0
Element 1: -1
Element 2: 2
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

5. Implementierung von Method RemoveAt

Beschreibung:

Die Methode nimmt einen Index als Parameter entgegen. Nach Ausführung der Methode soll das Element an der Stelle, durch den Index angeben, in der Lister entfernt sein.

Randbedingung:

Falls der Parameter negative ist oder größer ist als die Anzahl aller Element in der Liste, soll nichts passieren und die Liste unverändert bleiben. Dabei soll keine Exception geworfen werden.

Declaration:

```
public void RemoveAt(int index)
```

Test Program:

```
Console.WriteLine("Testing RemoveAt");
TextCollection collection = new TextCollection();

collection.Append("0");
collection.Append("1");
collection.Append("2");

collection.RemoveAt(-1);
collection.RemoveAt(-3);

PrintList(collection);

collection.RemoveAt(1);

PrintList(collection);

collection.RemoveAt(0);

PrintList(collection);

collection.RemoveAt(0);
collection.RemoveAt(0);

PrintList(collection);
```

Erwarteter Output:

Testing RemoveAt \ Element 0: 0 \ Element 1: 1
Element 2: 2
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
Element 0: 0
Element 1: 2
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
Element 0: 2
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
List is empty