Lab: Linear Data Structures

Problems for exercises and homework for the "Data Structures" course @ SoftUni.

You can check your solutions here: https://judge.softuni.bg/Contests/Compete/Index/550#0.

1. ArrayList<T>

Implement a data structure **`ArrayList<T>`** that holds a sequence of elements of generic type **`T`**. It should hold a **sequence of items in an array**. The structure should have **capacity** that **grows twice** when it is filled, **always starting at 2**. The list should support the following operations:

- **`int Count`** → returns the number of elements in the structure
- **`T this[int index]`** → the indexer should access the elements by **index** (in range **`0`** … **`Count-1`**) in the reverse order of adding
- **`void Add(T item)`** → adds an element to the sequence (grow twice the underlying array to extend its capacity in case the capacity is full)
- **`T RemoveAt(int index)`** → removes an element by **index** (in range **`0`** … **`Count-1`**) and returns the element

Be sure to **test implemented operations** whenever possible before moving to the next

Examples

```
static void Main(string[] args)
{
    ArrayList<int> list = new ArrayList<int>();
    list.Add(5);
    list[0] = list[0] + 1;
    int element = list.RemoveAt(0);
}
```

Solution

Declare the class **`ArrayList<T>`**

```
public class ArrayList<T>
{
    private const int Initial_Capacity = 2;

    private T[] items;

    public ArrayList()
    {
        this.items = new T[Initial_Capacity];
    }
}
```

Start with **`Count`** and **`Indexer`**

```
public int Count { get; private set; }

public T this[int index]
{
    get...

    set...
}
```

Implement **get** by index

```
get
{
    if (index >= this.Count)
    {
        throw new ArgumentOutOfRangeException();
    }

    return this.items[index];
}
```

And **set** by index should be

```
set
{
    if (index >= this.Count)
    {
        throw new ArgumentOutOfRangeException();
    }

    this.items[index] = value;
}
```

Implement **Add** and **Resize** methods

```
public void Add(T item)
{
    if (this.Count == this.items.Length)
    {
        this.Resize();
    }

    this.items[this.Count++] = item;
}
```

```
private void Resize()
{
    T[] copy = new T[this.items.Length * 2];
    for (int i = 0; i < this.items.Length; i++)
    {
        copy[i] = this.items[i];
    }

    this.items = copy;
}
```

Finally, implement **`RemoveAt`**, **`Shrink`** and Shift methods

```
public T RemoveAt(int index)
{
    if (index >= this.Count)
    {
        throw new ArgumentOutOfRangeException();
    }

    T element = this.items[index];
    this.items[index] = default(T);
    this.Shift(index);
    this.Count--;

    if (this.Count <= this.items.Length / 4)
    {
        this.Shrink();
    }

    return element;
}
```

```
private void Shift(int index)
{
    for (int i = index; i < this.Count; i++)
    {
        this.items[i] = this.items[i + 1];
    }
}
```

```
private void Shrink()
{
    T[] copy = new T[this.items.Length / 2];
    for (int i = 0; i < this.Count; i++)
    {
        copy[i] = this.items[i];
    }

    this.items = copy;
}
```