Lab – ArrayList Implementation

Problems for exercises and homework for the "Programming Advanced: OOP Basics" course from the official "Applied Programmer" curriculum.

You can check your solutions here: https://judge.softuni.bg/Contests/2851/Implementing-Array-List

Static Implementation of a List

Create a new console C# project in Visual Studio named **`ImplementArrayList`**, rename class **`Program.cs`** to **`StartUp.cs`** and add new public class **`CustomArrayList.cs`**.

Implement a **resizable list** in C#. Use the following **structure** for the class:

```
public class CustomArrayList
{
    private object[] arr;

    private int count;

    public int Count
    {
        get { return count; }
        private set { count = value; }
    }

    private static readonly int INITIAL_CAPACITY = 4;

    public CustomArrayList()
    {
        arr = new object[INITIAL_CAPACITY];
        count = 0;
    }

    public void Add(object item)
    {
    }

    public void Insert(int index, object item)
    {
    }

    public int IndexOf(object item)
    {
    }

    public void Clear()
    {
    }

    public bool Contains(object item)
    {
    }

    public object this[int index]
```

```
    {
    }

    public object Remove(int index)
    {
    }

    public int Remove(object item)
    {
    }
}
```

In the provided **structure** we have:

- An **array** of **object type** in which we will **store the list elements**.
- Property **Count**, which will keep **track** of how many **elements** are in the array at the moment.
- A **constructor** that **initializes** our array-based list with **initial capacity** (which is 4 elements by default)

```
private object[] arr;
private const int INITIAL_CAPACITY = 4;
private int count;
public int Count
{
    get
    {
        return count;
    }
    private set { count = value; }
}

public CustomArrayList()
{
    arr = new object[INITIAL_CAPACITY];
    count = 0;
}
```

Since the **`Add(item)`** method is similar to the **`Insert(index, item)`** method, we will write the code, so that **`Add(…)` method calls** the **`Insert(…)` method**. If the array is full, it will allocate **twice as much space** and copy the elements from the old to the new array. This can be done in the method **`Resize()`**.

1. Method Insert(index, item)

Let's implement the **`Insert(index, item)`** method for inserting an element into the array at given position.

Hints

This is how we can insert an element to given position.

- First we check the capacity and resize the underlying array if the capacity is not enough.
- Then we move the elements [index+1…end] on the right.
- Finally, we save the new element at position **`index`**.

This is how our code may look like:

```
public void Insert(int index, object item)
{
    if (Count == arr.Length)
    {
        [redacted]
    }

    for (int i = arr.Length - 1; i > index; i--)
    {
        [redacted]
    }

    [redacted]
}
```

This is how we can resize the array to increase its capacity twice:

```
private void Resize()
{
    object[] copy = new object[arr.Length * 2];
    Array.Copy(arr, copy, arr.Length);
    arr = copy;
}
```

2. Method Add(item)

Adds an element by calling the **Insert** method and uses the **number** of **elements** for the **index**.

- Just invoke the **`Insert()`** method.

3. Method IndexOf(item)

Searches for an item and **returns** its **index** or **-1** if no item is found.

Hints

You will need the Equals method to compare two items.

```
public int IndexOf(object item)
{
    for (int i = 0; i < arr.Length; i++)
    {
        [redacted]
    }
    return -1;
}
```

4. Method Clear()

Deletes the elements in the array and returns it to its initial capacity and count.

5. Method Contains(item)

Checks if the item exists in the list and returns **`true`** / **`false`**.

Hints

You can use a method that you have already written.

```
public bool Contains(object item)
{
    [illegible]

    bool found = (index != -1);

    return found;
}
```

6. Indexer: this[int index]

Used to **access** the **items by** their **index**. Before setting or getting an index of the property, it is necessary to check **whether the index is within the array**. In case an invalid index is submitted, throw new **ArgumentOutOfRangeException** with message: **`"Invalid index: " + index.`**

Hints

```
public object this[int index]
{
    get
    {
        if (index < 0 || index >= Count)
        {
            throw new ArgumentOutOfRangeException ("Invalid index: " + index);
        }
        return arr[index];
    }
    set
    {
        if (index < 0 || index >= Count)
        {
            throw new ArgumentOutOfRangeException ("Invalid index: " + index);
        }
        arr[index] = value;
    }

}
```

7. Method Remove(int index)

Removes an **element** located **on** a **given index and returns** that element. For this purpose, we will first find the required element, remove it, and then **move** the **elements after it** so that there is no space in the corresponding position. This rearrangement can be performed in the **Shift method**. In case the count of the elements in the array is less than ½ of its current size, perform the **Shrink method**, which reduces the size of the array by half, deleting part of the empty cells.

It is necessary to check whether the index is within the array. In case an **invalid index** is submitted, throw new **ArgumentOutOfRangeException** with message: **`"Invalid index: " + index.`**

Hints

```
public object Remove(int index)
{
    if (index >= Count || index < 0)
    {
        [illegible]
    }

    object item = arr[index];

    [illegible]

    if (Count <= arr.Length / 2)
    {
        [illegible]
    }

    return item;
}
```

```
private void Shift(int index)
{
    for (int i = index; i < arr.Length - 1; i++)
    {
        arr[i] = arr[i + 1];
    }

    arr[Count - 1] = null;
}
```

```
private void Shrink()
{
    object[] copy = new object[arr.Length / 2];
    Array.Copy(arr, copy, copy.Length);
    arr = copy;
}
```

8. Method Remove(object item)

Removes an element and **returns the index** on which that element is located.

Hints

```
public int Remove(object item)
{
    int index = [illegible]

    if (index == -1)
    {
        return index;
    }

    Remove(index);

    return index;
}
```

9. Test Your Code

Test the program in the **`Main()`** method of the class **`StartUp.cs`**.

Using an array of object type allows us to **store different data of object type**. For example, in this case we can store the int 7 and the string Tomato.

static void Main()	Output

`CustomArrayList shopingList = new CustomArrayList();`
 	
`shopingList.Add("Tomato");`
`shopingList.Add("Bread");`
`shopingList.Add("Cheese");`
`shopingList.Add("Cucumbers");`
`shopingList.Add("Chocolate");`
`shopingList.Add(7);`
`shopingList.Add("Coke");`
 	
`for (int i = 0; i < shopingList.Count; i++)`
`{`
`  Console.WriteLine(shopingList[i]);`
`}`
 	
`shopingList.Insert(1, "Lemon");`
 	
`Console.WriteLine(shopingList.IndexOf("Chocolate"));`
`Console.WriteLine(shopingList.Contains("Coke"));`
`Console.WriteLine(shopingList[1]);`
`Console.WriteLine(shopingList.Count);`
`shopingList.Remove(3);`
`shopingList.Remove("Tomato");`
`Console.WriteLine(shopingList.Count);`
	Tomato
	Bread
	Cheese
	Cucumbers
	Chocolate
	7
	Coke
	5
	True
	Lemon
	8
	6