**Lab: Linear-Data-Structures**

This document defines the lab for the ["Data Structures – Fundamentals (C#)" course @ Software University](https://softuni.bg/trainings/3672/data-structures-fundamentals-with-csharp-february-2022).

## List

Your task is to implement the **ADS** **IAbstractList<T>** inside the **List<T>** class provided.

You have to implement **all the methods** to solve the problem, however, you are free to add more methods with any access modifier you want.

### ****void Add(****T item)

### Adds an element at the end of the sequence.

### This method should in addition increase the size of the structure and ensure that there is enough space for the addition to work.

### If needed you will have to resize the array.

### int Count

### Returns the number of elements.

### void Insert(int index, T item)

### Inserts the passed element at the specified index in the sequence (if possible).

### If the index is outside of the sequence bounds, throw an IndexOutOfRangeException.

### Get (indexer)

### Returns the element at the given index and does not remove it from the collection.

### If the index is invalid throw IndexOutOfRangeException with a proper message of your choice (the message itself is not subjected to testing).

### Set (indexer)

### Sets the element at the given index. Again you should validate the index and throw IndexOutOfRangeException if the validation fails.

### void RemoveAt(int index)

### Removes the element at specified index.

### If the index is outside of the sequence bounds, throw an IndexOutOfRangeException.

* **bool Contains (T item)**
  + Returns **true** or **false** if the element is **present** inside the structure.

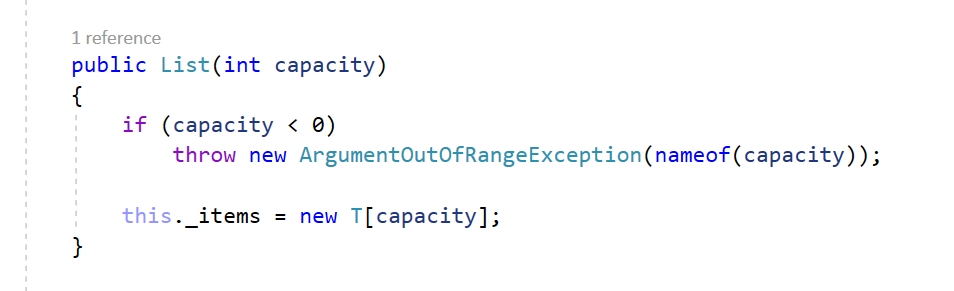
### int IndexOf(T item)

### Returns the index of an element if the element is not present in the structure then returns -1 as an invalid array index.

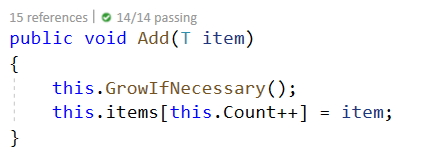
* **bool Remove (T item)**
  + Removes the first occurrence of the passed item and returns **true**.
  + If it is not present in the list, return **false**.

### Solution:

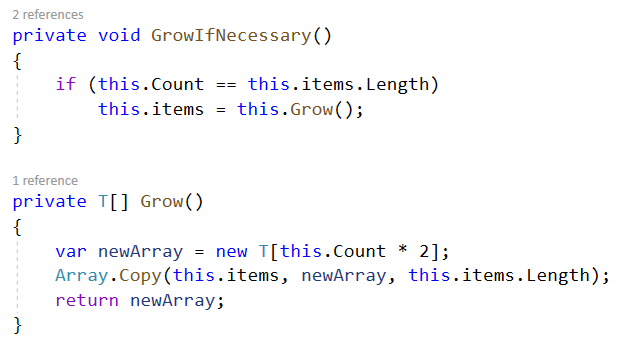
##### ****public List**(int capacity)**



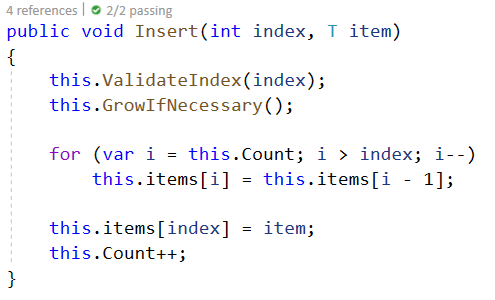
##### ****void Add(****T item)



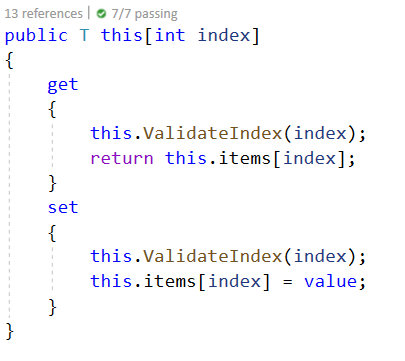
##### **Grow() helper methods**



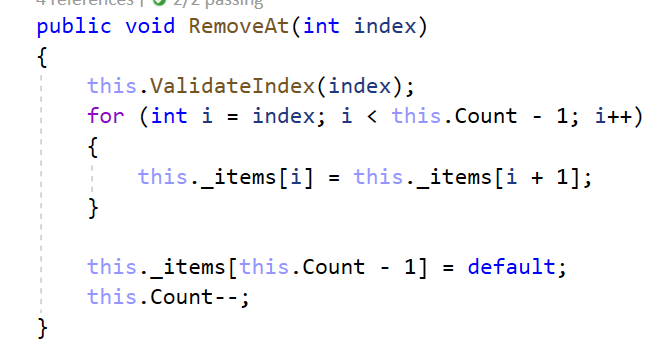
##### void Insert(int index, T item)



##### Get and set through indexer

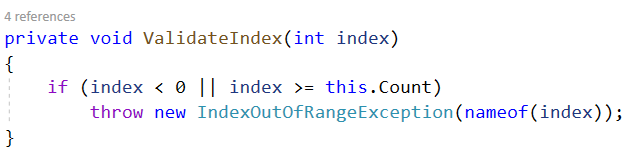


##### void RemoveAt(int index)

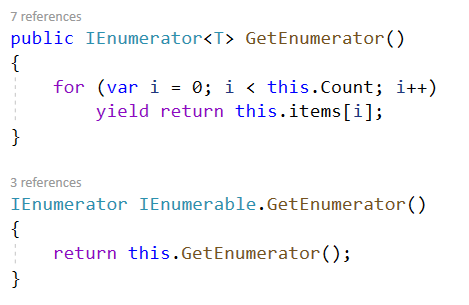


##### Take a look at those additional helper methods, you can reuse them whenever needed:

##### The ValidateIndex() method, feel free to add a message.



##### GetEnumerator()



All of the **other** **methods** are **easy** and **straightforward** to be **implemented** so you won't need any help. If it doesn't work the first time **simply try a different approach**.

## Stack

### Your task is to implement the ADS IAbstractStack<Т> inside the Stack<Т> class provided.

### You have to implement all the methods to solve the problem, however, you are free to add more methods with any access modifier you want.

### void Push(T item)

### Adds an element at the top of the stack and increases the size.

### T Pop()

### Removes an element at the current top of the stack and returns it.

### If the stack is empty throw InvalidOperationException with the appropriate message.

### T Peek()

### Returns the element at the current top of the stack without removing it.

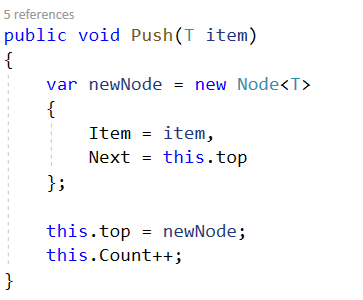
### If the stack is empty throw InvalidOperationException with the appropriate message.

### int Count

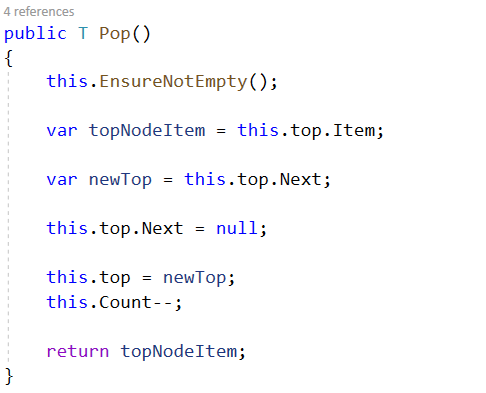
### Returns the number of elements inside the stack.

### Solution:

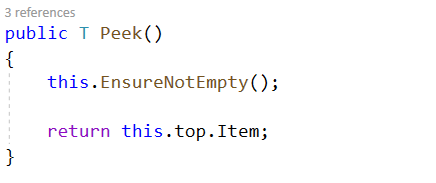
### void Push (T item)



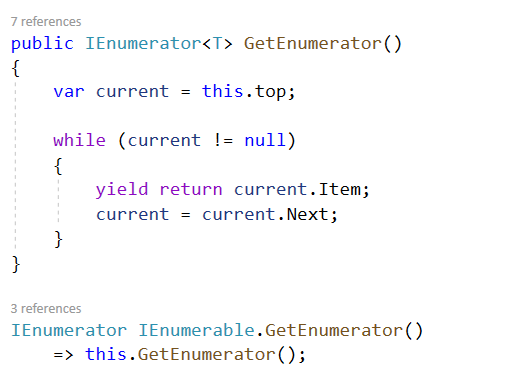
### T Pop ()



### T Peek ()



### GetEnumerator()



All of the **other** **methods** are **easy** and **straightforward** to be **implemented**. If it doesn't work the first time **simply try a different approach**.

## Queue

Your task is to implement the **ADS** **IAbstractQueue<T>** inside the **Queue<T>** class provided.

You have to implement all the methods to solve the problem, however, you are free to add more methods with any access modifier you want.

### void Enqueue(T item)

* + **Adds** an **element** at the **end** of the **queue** and increases the size.

### T Dequeue()

##### Removes and returns the first element at the queue also decreases the size and performs a check if this method is called upon empty collection.

##### If so throw InvalidOperationException with the message of your choice.

### T Peek()

##### Returns the element at the current front of the queue. If the collection is empty throw InvalidOperationException with the appropriate message.

### int Count

##### Returns the number of elements inside the stack.

### Solution:

As you can see a lot of the operations **described** **above** are a lot like those we did on the **Stack** problem so think about **how** you can **reuse** and **modify** those.

Now you can see a **slightly** **different** way of adding the elements - in the **stack** **implementation** we had a **pointer** to the **top** **element** here we have **the** **first** pointer so you need to **find** the **last** **element** so you can **enqueue** the **new** **node**.

## SinglyLinkedList

Your task is to implement the ADS **IAbstractLinkedList<T>** inside the **SinglyLinkedList<T>** class provided.

You have to implement all the methods to solve the problem, however, you are free to add more methods with any access modifier you want.

### void AddFirst(T item)

### Adds an element in front of the collection and increases the size.

### void AddLast(T item)

### Adds an element after the last element of the collection and increases the size.

### T RemoveFirst()

### Removes and returns the first element of the collection if it is such if no then throw InvalidOperationException with the appropriate message.

### T RemoveLast()

### Removes and returns the last element of the collection if it is such if no then throw InvalidOperationException with the appropriate message.

### T GetFirst()

### Returns but does not remove the first element of the collection if it is such if no then throw InvalidOperationException with the appropriate message.

### T GetLast()

### Returns but does not remove the last element of the collection if it is such if no then throw InvalidOperationException with the appropriate message.

### int Count

### Returns the number of elements inside the collection.

### Solution:

Here comes the **tricky** **part**, **all** of the **operations** above **are** **like** the **previous** **ones** you have implemented combined and modified of course. But in really small and simple matters so try to solve those on your own.

**Good Luck!** And **remember** if something gets **too** **complicated** or **unclear** and **does** **not** **work** you can **always start again by choosing a different approach.**