YACA

Yet **A**nother **C**ollection **A**pproach

Introduction

YACA is another approach to introduce Collections to TypeScript / JavaScript like known in Java, C# or other object-oriented programming languages. There are other approaches, but sometimes, small things regarding the convenience are missing.

YACA contains at the moment only **List<T>** as collection type. Further types (e.g. Dictionary *[WIP]* or Stack) are planned.

See the **Change Log** for recent updates.

Installation

```
npm install -S yaca
```

Usage (List)

```
import {List} from 'yaca';

// Default constructor
let numberList: List<number> = new List<number>();

// Constructor with initial value
var stringList: List<string> = new List<string>
("initial value");
```

```
// Constructor with array as initial value
let booelanList: List<boolean> = new List<boolean>
([true, false, true, true]);

// Usage of more complex types
var otherList: List<SomeType> = new List<SomeType>();

numberList.add(22);
numberList.addRange([23,24,25]);
numberList.clear();
```

Constructors

The type **T** is the generic type defined in the constructor (e.g. number, boolean etc.)

Name & Arguments	Description
List<T>()	Default constructor
List<T>(T)	Constructor with one initial value
List<T>(T[])	Constructor with an array of Type T as initial values
List<T> (List<T>)	Constructor with a List of Type T as initial values

Properties

Name	Type	Description
length	number	Gets the length of the list (read-only)

Methods

The type **T** is the generic type defined in the constructor (e.g. number, boolean etc.)

Name & Arguments	Return Type	Description
add(T)	void	Adds an element at the end of the list
addRange(T[])	void	Adds an array of the type T at the end of the list
addRange(List<T>)	void	Adds a List of type T at the end of the list
clear()	void	Removes all elements from the list
contains(T)	boolean	Checks whether the list contains the specified value
copyToArray()	T[]	Copies the whole list into an array of type T
copyToArray(number)	T[]	Copies the list into an array of type T, beginning from the passed

		start index of the list
copyToArray(number, number)	T[]	Copies the list into an array of type T, as range between the passed start and end index of the list
dequeue()	T / undefined	Removes the top element of the list and returns its value (end index). Undefined will be returned if the list is empty
distinct()	void	Removes all duplicates of values in the list. All duplicates after the first occurrence of each value will be removed
		Inserts a new value at the defined index position. All

enqueue(T)	void	values above (index +1) will be shifted to the next higher index. The last item of the list will be shifted to a new value
forEach(*callback*)	void	Implementation of a forEach loop. The callback function gets the values
get()	T	Gets the value of the List at the specified index position
getRange()	List<T>	Copies the whole list into a new list of type T
getRange(number)	List<T>	Copies the list into a new list of type T, beginning from the passed start index of the original list
		Copies the list

getRange(number, number)	List<T>	into a new list of type T, as range between the passed start and end index of the original list
indexOf(T)	number	Gets the index of the first occurrence of the passed value
indicesOf(T)	number[]	Gets an array of the indices of all occurrences of the passed value
indicesOfAsList(T)	List<T>	Gets a list of the indices of all occurrences of the passed value
insertAtIndex(number, T)	void	Inserts a new value at the defined index position. All values above (index +1) will be shifted to the next higher index. The last

		item of the list will be shifted to a new value
lastIndexOf(T)	number	Gets the index of the last occurrence of the passed value
next(any?)	IteratorResult	Method to get the next value of an iterator. If the last item of the list is reached, the returned object indicates that the iterations are finished. Afterwards, the method starts again at index position 0. Calling of the forEach() method will also reset the position to 0.
		Returns the value of the top element of the list without removing it

peek()	T / undefined	(end position / last element). undefined will be returned if the list is empty
pop()	T / undefined	Removes the top element of the list and returns its value (end position / last element). undefined will be returned if the list is empty
push(T)	void	Inserts a new value at the top position of the list (end position / last element). This method is synonymous with add()
remove(T)	boolean	Removes the passed value at the first occurrence in the list. Returns true if an

		element was removed
removeAll(T)	boolean	Removes the passed value at all positions in the List. Returns true if an element was removed
removeAt(T)	void	Removes the value at the defined index. All values above will be shifted one index position down (index - 1)
removeAtIndices(number[])	void	Removes all values at the defined indices. All values above a removes item will be shifted one index position down (index - 1)
		Removes all values at the defined indices.

removeAtIndices(List<number>)	void	All values above a removes item will be shifted one index position down (index - 1)
reverse()	void	Inverts the list
set(number, T)	void	Updates a value of the list at the specified index position
sort(*CompareFunction*)	void	Sorts the list according to the passed comparing function. The function has to compare two values of the type T. If value 1 is smaller than value 2, -1 has to be returned. If value 1 is bigger than value 2, 1 has to be returned. If both values are equal, 0 has to be

		returned.
swapValues(number, number)	void	Swaps the values at the two defined index positions in the list