Nolix

Nolix containers

2019-02-01

Table of contents

1 Introduction

1.1 What Nolix containers are

Nolix containers are containers that implement the **'IContainer'** interface from the Nolix library.

Nolix containers store elements in the memory. The Nolix containers provide many search and statistical methods.

1.2 Why to use Nolix containers

- The given ReadContainer can read on **any** iterable object or array. The ReadContainer provides many search and statistical methods that are not available on common iterable objects or arrays.
- The given Matrix stores elements in rows and columns like a 2-dimensional array. The rows and columns of a Matrix themselves can be accessed comfortably **like containers**.
- The given List is optimized to addAtBegin and remove elements **frequently**.

1.3 Where Nolix containers are

Nolix containers are defined in the Nolix library. To use Nolix containers, import the Nolix library into your project.

2 Lists

2.1 How to import the List class

```
import ch.nolix.core.container.List;

…

var listClass = List.class;

…
```

The List class can be found in the package ‘ch.nolix.core.container’.

2.2 How to create an empty List

```
var list = new List<String>();
```

The created List can store Strings. The List is empty at the beginning.

2.3 Null elements

Rule

> A List does not store null elements.

If there is tried to add a null element to a List, a NullArgumentException wil be thrown.

Reason

Storing null elements in a container has several disadvantages. A null element does not serve as element of the type it has been declared as. A statistical analysis on a container with null elements will either fail or deliver a blurred result. E.g. what should be the average of 10 BigDecimals and 5 null elements?

2.4 How to create a List containing some given elements

```
var cities = new List<String>(”Berlin”, ”Hamburg”, ”München”);
```

If one of the given elements would be null, the constructor of the List threw a NullArgumentException. If e.g. the 2th of the given elements would be null, the error message of the NullArgumentException was:

”The 2th of the given elements is null.”

2.5 How to create a List containing the elements of a given array

```
String[] citiesArray;

…

var cities = new List<String>(citiesArray);
```

If one of the elements of the given array is null, the List’s constructor will throw a NullArgumentException. If e.g. the 2th element of the given array is null, the error message of the NullArgumentException will be:

”The 2th element is null.”

2.6 How to add an element at the end of a List

```
var cities = new List<String>();

cities.addAtEnd"Berlin");
```

If the given element would be null, the ‘addAtBegin’ method threw a NullArgumentException. The error message of the NullArgumentException was:

"The given element is null."

2.7 How to add several elements at the end to a List at once

```
var cities = new List<String>();

cities.addAtEnd("Berlin", "Hamburg", "München");
```

The ‘addAtBegin‘ method can take an arbitrary number of elements. If one of the given elements is null, the ‘addAtBegin’ method will throw a NullArgumentException. If e.g. the 2th of the given elements is null, the error message of the NullArgumentException will be:

"The 2th element is null."

2.8 How to add the elements of a given array at the end of a List

```
var cities = new List<String();

…

String[] citiesArray;

…

cities.addAtEnd(citiesArray);
```

If the given array is null, the ‘addAtBegin’ method will throw a NullArgumentException.

If one of the elements of the given array is null, the ‘addAtBegin’ method will throw a NullArgumentException. If e.g. the 2th element of the given array is null, the error message of the NullArgumentException will be:

"The 2th element is null."

2.9 How to add an element at the end of a List regarding singularity

```
var berlin = "Berlin";
var hamburg = "Hamburg";
var muenchen = "München";
var koeln = "Köln";

var cities = new List<String>(berlin, hamburg, muenchen);

cititesaddAtEnd(berlin); //ok
cities.addAtEndRegardingSingularity(koeln); //ok
cities.addAtEndRegardingSingularity(berlin); //error
```

Sometimes, you want to make sure that a container does not contain an element several times. E.g. when a container of cities should store the same city only once. The method 'addAtEndRegarindSingulary' validates that the List does not contain already the given element.

If the given element is already in the List, the method 'addAtEndRegardingSingularity' will throw an InvalidArgumentException. The error message of the InvalidArgumentException will be:

"The current list contains already the given element."

Attention

The method 'addAtEndRegardingSingularity' only checks if a List contains already the **same** element. The given element is allowed to equal another element of the List.

2.10 Strictness of remove methods

Rule

> At default, the remove methods of a List will throw an Exception if the elements, that are **supposed to be removed**, are not in the List.

Reason

At default, a remove method is supposed to do actually a remove action. E.g. if the first element of a List should be removed and the List is empty, the remove action is not skipped silently. An exception will be thrown instead. This takes the programmer **aware** to call remove methods **not by chance**. At default, a remove method always causes a change or throws an exception.

2.11 How to remove the first element from a List

```
var cities1 = new List<String>("Berlin", "Hamburg", "München");
cities1.removeFirst(); //ok

var cities2 = new List<String>();
cities2.removeFirst(); //error
```

The method 'removeFirst' removes the element at the **begin** of a List.

If a List is empty, the 'removeFirst' method will throw an EmptyArgumentException. The message of the EmptyArgumentException will be:

"The given list is empty."

2.12 How to remove the last element from a List

```
var cities1 = new List<String>("Berlin", "Hamburg", "München");
cities1.removeLast(); //ok

var cities2 = new List<String>();
cities2.removeLast(); //error
```

The method 'removeLast' removes the element at the **end** of a List.

If a List is empty, the 'removeLast' method will throw an EmptyArgumentException. The message of the EmptyArgumentException will be:

"The given list is empty."

2.13 How to remove a given element from a List

```
var berlin = "Berlin";
var hamburg = "Hamburg";
var muenchen = "München";
var koeln = "Köln";

var cities = new List<String>(berlin, hamburg, muenchen);

cities.removeFirst(berlin); //ok
cities.removeFirst(koeln); //error
```

The method 'removeFirst' removes the first appearance of the given element from the List. If the List does not contain the given element, the 'removeFirst' method will throw an InvalidArgumentException. The message of the InvalidArgumentException will be:

"The given list does not contain the given element."

Attention

The method 'removeFirst' only removes the **same** element. The method does not remove an element from the List, that only equals the given element.

3 ReadContainers

3.1 How to import the ReadContainer class

```
import ch.nolix.core.container.ReadContainer;

…

var readContainerClass = ReadContainer.class

…
```

The ReadContainer class is in the package ‘ch.nolix.core.container’.

3.2 How to create a ReadContainer for a given array

```
Person[] personArray;

…

var personReadContainer = new ReadContainer<Person>(personArray);
```

If the given person array is null, the constructor of the ReadContainer will throw a NullArgumentException.

The created ReadContainer can access the elements of the given array. The ReadContainer does **not** mutate the given array.

3.3 Advantages of a ReadContainer

Common approach

```
…

public Person getOldestPerson() {

  person oldestPerson = personArray[0];

  for (var p : personArray) {
    if (
      p.getAgeInYears() >
      oldestPerson.getAgeInYears()
    ) {
      oldestPerson = p;
    }
  }

  return oldestPerson;
}

…
```

ReadContainer approach

```
…

public Person getOldestPerson() {
  return
  new ReadContainer(personArray)
  .getRefByMaxInt(p -> p.getAgeInYears());
}

…
```

A ReadContainer provides comfortable methods like the shown ‘getRefByMaxInt’ method. The ‘getRefByMaxInt’ method returns the element from which the given anonymous function extracts the biggest integer. The advantage of a ReadContainer is that search and statistical methods do **not need to be reimplemented** for given arrays or for any given iterable object.