Nolix

Nolix containers

2020-11-04

Table of contents

1 Introduction

1.1 What Nolix containers are

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

1.2 Why to use Nolix containers

- The provided 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 provided 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 provided LinkedList is optimized to add and remove elements **frequently**.

1.3 Where Nolix containers are

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

1.4 Structure of this document

Chaper 2 introduces the Nolix LinkedList and chapter 3 introduces the Nolix ReadContainer.

2 LinkedList

2.1 Import LinkedList

```
import ch.nolix.common.container.LinkedList;
```

The LinkedList is in the ch.nolix.common.container package.

2.2 Create a new empty LinkedList

```
var LinkedList = new LinkedList<String>();
```

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

2.3 Null elements

Rule

> A LinkedList does not store null elements.

If there is tried to add a null element to a LinkedList, a ArgumentIsNullException will 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 Create a LinkedList with some particular elements

```
var cities = LinkedList.withElements("Berlin", "Hamburg", "München");
```

If one of the given elements is null, the static withElements method will throw a ArgumentIsNullException. If e.g. the 2th of the given elements is null, the error message of the ArgumentIsNullException will be:

"The 2th element is null."

2.5 Create a LinkedList with the elements from a given array

```
String[] citiesArray;

…

var cities = LinkedList.from(citiesArray);
```

If one of the elements of the given array is null, the static from method will throw a ArgumentIsNullException. If e.g. the 2th element of the given array is null, the error message of the ArgumentIsNullException will be:

”The 2th element is null.”

2.6 Add an element at the end of a LinkedList

```
LinkedList<String> cities;

…

cities.addAtEnd(”Berlin”);
```

If the given element is null, the addAtBegin method will throw a ArgumentIsNullException. The error message of the ArgumentIsNullException will be:

”The given element is null.”

2.7 Add several elements at the end to a LinkedList

```
LinkedList<String>() cities;

…

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 ArgumentIsNullException. If e.g. the 2th of the given elements is null, the error message of the ArgumentIsNullException will be:

"The 2th element is null."

2.8 Add the elements of a given array at the end of a LinkedList

```
LinkedList<String() cities;
String[] citiesArray;

…

cities.addAtEnd(citiesArray);
```

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

”The 2th element is null.”

2.9 Add an element at the end of a LinkedList regarding singularity

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

var cities = LinkedList.withElements(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 addAtEndRegarindSingulary method validates that the LinkedList does not contain already the given element.

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

"The current LinkedList contains already the given element."

Attention

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

2.10 Strictness of remove methods

Rule

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

Reason

At default, a remove method is supposed to do actually a remove action. E.g. if the first element of a LinkedList should be removed and the LinkedList is empty, the remove action is not skipped silently. The LinkedList will throw an Exception 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 Remove the first element from a LinkedList

```
var cities1 = LinkedList.withElements("Berlin", "Hamburg", "München");
cities1.removeFirst(); //ok

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

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

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

"The given LinkedList is empty."

2.12 Remove the last element from a LinkedList

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

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

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

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

"The given LinkedList is empty."

2.13 Remove a given element from a LinkedList

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

var cities = LinkedList.withElements(berlin, hamburg, muenchen);

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

The removeFirst method removes the first appearance of the given element from the LinkedList.

If the LinkedList does not contain the given element, the removeFirst method will throw a InvalidArgumentException. The message of the InvalidArgumentException will be:

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

Attention

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

3 ReadContainer

3.1 Import the ReadContainer class

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

The ReadContainer class is in the ch.nolix.common.container package.

3.2 Create a ReadContainer for a given array

```
Person[] personArray;

…

var personReadContainer = ReadContainer.for(personArray);
```

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`
	`  ReadContainer.for(personArray)`
	`  .getRefByMaxInt(Person::getAgeInYears);`
	`}`

The getRefByMaxInt method returns the element from which the given function extracts the biggest Integer. The advantage of a ReadContainer is that search and statistical methods do **not** need to be reimplemented for any given iterable object or array.