Nolix

Nolix containers

2025-03-29

Table of contents

1 Introduction

1.1 What Nolix containers are

Nolix containers are data structures with many comfort methods.

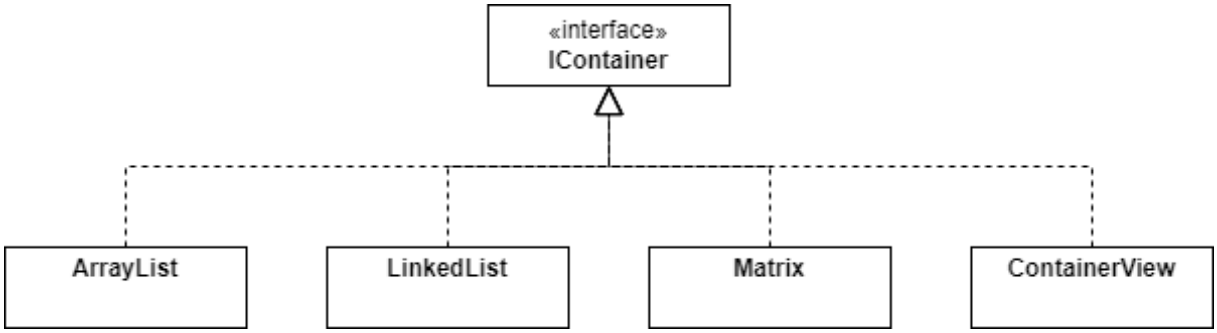
1.2 Why use Nolix containers

- Nolix containers are data structures that provide **various** search and aggregation functions.
- The provided ContainerViews can read on **any** iterable object or array. ContainerViews provide search and aggregation 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 themselves.

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.

2 Overview



All Nolix containers implement the IContainer interface. The IContainer interface declares only methods for accessing the elements of the IContainer. So, an IContainer does not need to be mutable. The IContainer interface is in the ch.nolix.coreapi.containerapi.baseapi package.

IContainer	Description
ArrayList	Optimized for access elements by their index.
LinkedList	Optimized for adding elements at the beginning or end.
Matrix	Organizes elements in rows and columns.
ContainerView	Readable only. Can access arrays or any Iterables as if they were IContainers themselves.

3 LinkedList

3.1 Import LinkedList

```
import ch.nolix.core.container.linkedlist.LinkedList;
```

The LinkedList is in the ch.nolix.core.container.linkedlist package.

3.2 Create a new empty LinkedList

```
LinkedList<String> linkedList = LinkedList.create();
```

The static create method creates a new empty LinkedList. The created LinkedList can store elements of the declared type.

3.3 Create a LinkedList with given elements

```
LinkedList<String> cities =
LinkedList.withElement("Berlin", "Hamburg", "München");
```

The static withElement method can take an arbitrary number of elements.

If one of the given elements is null, the static withElement method will throw a ArgumentIsNullException. For example, if the 2th of the given elements is null, the error message of the ArgumentIsNullException will be:

The 2th element is null.

3.4 Create a LinkedList with the elements from a given array

```
String[] citiesArray = …
LikedList<String> cities = LinkedList.fromArray(citiesArray);
```

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

The 2th element is null.

3.5 Add an element at the end to a LinkedList

```
LinkedList<String> cities = …
cities.addAtEnd("Amsterdam");
```

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.

3.6 Add several elements at the end to a LinkedList

```
LinkedList<String>() cities = …
cities.addAtEnd("Amsterdam", "Berlin", "Hamburg");
```

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. For example, if the 2th of the given elements is null, the error message of the ArgumentIsNullException will be:

The 2th element is null.

3.7 Add the elements of an array at the end to 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. For example, if the 2th element of the given array is null, the error message of the ArgumentIsNullException will be:

The 2th element is null.

3.8 Remove the first element from a LinkedList strictly

```
LinkedList<String> cities1 =
LinkedList.withElement("Berlin", "Hamburg", "München");
cities1.removeFirstStrictly(); //ok

LinkedList<String> cities2 = LinkedList.createEmpty();
cities2.removeFirstStrictly(); //error
```

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

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

The given LinkedList is empty.

3.9 Remove the last element from a LinkedList strictly

```
LinkedList<String> cities1 =
LinkedList.withElement("Berlin", "Hamburg", "München");
cities1.removeLastStrictly(); //ok

LinkedList<String> cities2 = new LinkedList.createEmpty();
cities2.removeLastStrictly(); //error
```

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

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

The given LinkedList is empty.

3.10 Remove an element from a LinkedList

```
String amsterdam = "Amsterdam";
String berlin = "Berlin";
String hamburg = "Hamburg";
String koeln = "Köln";
var cities = LinkedList.withElement(Amsterdam, berlin, hamburg);

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

The removeFirstOccurence method removes the **first** occurence of the given element from the LinkedList.

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

The given LinkedList does not contain the given element.

Warning

The removeFirst method removes only **same** elements. The method does not remove elements that only equals a given element.

4 ContainerView

4.1 Import ContainerView

```
import ch.nolix.core.container.containerView.ContainerView;
```

The ContainerView is in the ch.nolix.core.container.containerview package.

4.2 Create a ContainerView for a given array

```
Person[] personArray = …
IContainer<Person> personContainerView =
ContainerView.forArray(personArray);
```

The created ContainerView can access the elements of the given array. The ContainerView does **not** mutate the given array. The given array can be empty.

If the given array is null, the static forArray method will throw an ArgumentIsNullException. The message of the ArgumentIsNullException will be:

The given array is null.

4.3 Use ContainerView

Common approach	`public Person getOldestPerson(Person[] persons) {`
	`  var oldestPerson = persons[0];`
	`  for (var p : persons) {`
	`  if (`
	`  p.getAgeInYears() > oldestPerson.getAgeInYears()`
	`  ) {`
	`  oldestPerson = p;`
	`  }`
	`  }`
	`  return oldestPerson;`
	`}`

ContainerView approach	`public Person getOldestPerson(Person[] persons) {`
	`  return`
	`  ContainerView`
	`  .forArray(persons)`
	`  .getStoredByMax(Person::getAgeInYears);`
	`}`

The getStoredByMax method returns the element from which the given function extracts the biggest value. The big advantage of a ContainerView is that common search and statistical methods do **not need** to be reimplemented for iterable objects or array.