Test Documentation

Enrico Bolzonello

July 2021

Indice

1 ListTester

Test Suite Summary
The test suite test ListAdapter class and verifies that it behaves as declared in the HList interface.

Test Suite Design
Each method is tested in different scenarios to verify that also exceptions are managed without compromising execution.

Test Suite Pre-Condition
Empty HList that will be the subject of our tests.

Test Suite Post-Condition
All tests should run correctly, so the behavior of the ListAdapter class is the same of the HList interface.

Test Suite Execution Variables
none

1.1 Methods

1.1.1 testInitial()

Summary: Test if constructor has correctly instantiated an empty ListAdapter by making sure that size equals to 0 and isEmpty() returns true.

Test Case Design

1. Verify that size() returns 0 and isEmpty() returns true

Test Description: Test initial condition after calling ListAdapter constructor.

Pre-Condition: empty List

Post-Condition: empty List

Expected Results
size() returns 0
isEmpty() returns true
The List is empty.

1.1.2 testAdd_1()

Summary: Test add(o) method of ListAdapter.

Test Case Design

1. Run add(o) method twice with two items of different types
2. Verify that size() returns 2, isEmpty() returns false and get(index) returns the value we've added

Test Description: Test add(o) method by assuring that size() does not return 0, isEmpty() returns false and get(index) returns the value we have added.

Pre-Condition: empty List

Post-Condition: List size = 2, the elements have been added (get(index) returns correct values)

Expected Results
size() returns 2
isEmpty() returns false
get(0) returns "uno"
get(1) returns 1
The element have been added correctly.

1.1.3 testAdd_2()

Summary: Test add(o) method of ListAdapter with a null object.

Test Case Design

1. Run add(o) method
2. Verify that NullPointerException is thrown and that the List remains empty

Test Description: Test add(o) method with a null object.

Pre-Condition: empty List

Post-Condition: NullPointerException, List empty

Expected Results
The element has not been added and the method throws NullPointerException.

1.1.4 testAddIndex_1()

Summary: Test add(index, o) method of ListAdapter with a valid index.

Test Case Design

1. Run add(index, o) method with a valid index (in this case 0)
2. Verify that size() returns 1, isEmpty() returns false and get(index) returns the value we've added

Test Description: Test add(index, o) method with a null object.

Pre-Condition: empty List

Post-Condition: List size = 1, the element has been added

Expected Results
size() returns 1
isEmpty() returns false
get(0) returns the item we added
The element has been added correctly.

1.1.5 testAddIndex_2()

Summary: Test add(index, o) method of ListAdapter with a invalid index.

Test Case Design

1. Run add(index, o) method with a invalid index (in this case 4)
2. Verify that IndexOutOfBounds exception is thrown

Test Description: Test add(index, o) method of ListAdapter with a invalid index.

Pre-Condition: empty List

Post-Condition: IndexOutOfBoundException thrown, List remains empty

Expected Results
IndexOutOfBounds thrown
The element has not been added.

1.1.6 testAddIndex_3()

Summary: Test add(index, o) method of ListAdapter with a null object.

Test Case Design

1. Run add(index, o) method with a null object
2. Verify that NullPointerException is thrown and that the list remains empty

Test Description: Test add(index, o) method with a null object.

Pre-Condition: empty List

Post-Condition: NullPointerException, List empty

Expected Results
The element has not been added and the method throws NullPointerException.

1.1.7 testContains_1()

Summary: Test contains(o) method with different objects (may or may not be in the list)

Test Case Design

1. Add 3 elements (in this case "1", and 3.0)
2. Verify that contains(o) with an item that has been added returns true
3. Verify that contains(o) with an item that has not been added returns false

Test Description: Test contains(o) method with different objects (may or may not be in the list)

Pre-Condition: List with three different elements

Post-Condition: List with three different elements, contains(o) returns correct values

Expected Results
contains(o) returns correct values
List remains unchanged.

1.1.8 testContains_2()

Summary: Test contains(o) method with null objects

Test Case Design

1. Call contains(o) with a null object and verify it throws NullPointerException

Test Description: Test contains(o) method with null objects

Pre-Condition: List with three different elements

Post-Condition: List with three different elements, throws NullPointerException

Expected Results
throws NullPointerException
List remains unchanged.

1.1.9 testIterator()

Summary: Test iterator() and all of HIterator interface methods

Test Case Design

1. Add some elements to the list
2. Call iterator()
3. Verify that iterator.next() returns correct elements
4. Remove that element by calling iterator.remove() and verify it has been removed from the list correctly

Test Description: Test iterator() and all of HIterator interface methods

Pre-Condition: List size = 4

Post-Condition: List is empty

Expected Results
all calls of iterator.next() return expected value
the sum of all calls of iterator.remove() removes all elements from the list, list is empty as a result.

1.1.10 testToArray()

Summary: Test toArray() method

Test Case Design

1. Create a new array of Objects and initialize it by calling toArray(a)
2. Create a new array with all the elements of the list
3. Verify that the two arrays are equal

Test Description: Test toArray() method

Pre-Condition: List has 4 elements, array is empty

Post-Condition: List has 4 elements (unchanged), array has 4 elements (same elements as the list)

Expected Results
array and list have same items
List remains unchanged

1.1.11 testToArrayA_1()

Summary: Test toArray(a) method with the array parameter with the same type and same length as the list

Test Case Design

1. Create a new array of Objects and call toArray(a)
2. Create a new array with all the elements of the list
3. Verify that the two arrays are equal

Test Description: Test toArray(a) method with the array parameter with the same type and same length as the list

Pre-Condition: List has 4 elements, array is empty

Post-Condition: List has 4 elements (unchanged), array has 4 elements (same elements as the list)

Expected Results
array and list have same items
List remains unchanged

1.1.12 testToArrayA_2()

Summary: Test toArray(a) method with the array parameter with the same type and bigger length than the list

Test Case Design

1. Create a new array of Objects with more elements than list has and call toArray(a)
2. Verify that all elements of list are in the array
3. Verify that the last element of the array is null

Test Description: Test toArray(a) method with the array parameter with the same type and same length as the list

Pre-Condition: List has 4 elements, array is empty

Post-Condition: List has 4 elements (unchanged), array has 5 elements (same elements as the list + null object)

Expected Results
all elements of list are in the array
array has a null object after the list elements
List remains unchanged

1.1.13 testRemoveObject_1()

Summary: Test remove(o) method with valid objects

Test Case Design

1. Call remove(o) with o object that is present in the list, verify that returns true and verify that the list does not contains the object anymore
2. Try to call remove with the same element (will return false)

Test Description: Test remove(o) method with valid objects

Pre-Condition: List has 4 elements

Post-Condition: List has 3 elements

Expected Results
remove(o) returns true and removes the element from the list
remove(o) (same object as before) returns false

1.1.14 testRemoveObject_2()

Summary: Test remove(o) method with null object

Test Case Design

1. Call remove(o) with null object and verify it throws NullPointerException

Test Description: Test remove(o) method with null object

Pre-Condition: List has 4 elements

Post-Condition: List has 4 elements

Expected Results
remove(null) throws NullPointerException
List is unchanged

1.1.15 testRemoveIndex_1()

Summary: Test remove(index) method with valid index

Test Case Design

1. Call remove(index) with 0<=index<size , verify that returns the element removed and verify that the list does not contain the object anymore

Test Description: Test remove(index) method with valid index

Pre-Condition: List has 4 elements

Post-Condition: List has 3 elements

Expected Results
remove(index) returns the element removed
List has one less element

1.1.16 testRemoveIndex_2()

Summary: Test remove(index) method with invalid index

Test Case Design

1. Call remove(index) with index<0||index>=size

2. Verify that throws IndexOutOfBoundsException

Test Description: Test remove(index) method with invalid index

Pre-Condition: List has 4 elements

Post-Condition: List has 4 elements

Expected Results
remove(index) throws IndexOutOfBoundsException

1.1.17 testContainsAll_1()

Summary: Test containsAll(c) method with a collection with some elements
Test Case Design

1. Create a new HCollection (in this case a ListAdapter) and add some elements (first only elements that are present in the first list)
2. Verify that containsAll(c) returns true
3. Add an element that is not in the original list
4. Verify that containsAll(c) returns false

Test Description: Test containsAll(c) method with a collection with some elements

Pre-Condition: List has 4 elements, HCollection is empty

Post-Condition: List has 4 elements, HCollection has 3 elements

Expected Results
The first containsAll(c) returns true, because the collection has only elements that are present in the list
The second containsAll(c) (after adding the element that is not present in the list) returns false.

1.1.18 testContainsAll_2()

Summary: Test containsAll(c) method with a null collection

Test Case Design

1. Create a new HCollection (in this case a ListAdapter) and add some elements (first only elements that are present in the first list)
2. Verify that containsAll(c) returns true
3. Add an element that is not in the original list
4. Verify that containsAll(c) returns false

Test Description: Test containsAll(c) method with a null collection

Pre-Condition: List has 4 elements, HCollection is null

Post-Condition: List has 4 elements, HCollection is null, NullPointerException has been thrown

Expected Results
List and HCollection remain unchanged, NullPointerException is thrown.

1.1.19 testAddAll_1()

Summary: Test addAll(c) method with a collection with some elements

Test Case Design

1. Create a new HCollection (in this case a ListAdapter) and add some elements
2. Call addAll(c) and verify it returns true
3. Verify that all the elements are in the list

Test Description: Test addAll(c) method with a collection with some elements

Pre-Condition: List is empty, HCollection has 2 elements

Post-Condition: List has 2 elements, HCollection has 2 elements

Expected Results
addAll(c) returns true
all elements of the HCollection are present in the List

1.1.20 testAddAll_2()

Summary: Test addAll(c) method with a null collection

Test Case Design

1. Create a new null HCollection (in this case a ListAdapter)
2. Call addAll(c) and verify it throws NullPointerException

Test Description: Test addAll(c) method with a null collection

Pre-Condition: List is empty, HCollection is null

Post-Condition: List is empty, HCollection is null, NullPointerException is thrown

Expected Results
List and HCollection remain unchanged, NullPointerException is thrown.

1.1.21 testRemoveAll_1()

Summary: Test removeAll(c) method with a valid collection

Test Case Design

1. Create a new HCollection (in this case a ListAdapter) and add some elements
2. Call removeAll(c)
3. Verify that all the correct elements are removed from the list

Test Description: Test removeAll(c) method with a valid collection

Pre-Condition: List has 4 elements, HCollection has 3 elements.

Post-Condition: List has 2 elements, HCollection has 3 elements.

Expected Results
removeAll(c) returns true
The elements in the HCollection that are present in the List are removed from the List, HCollection remains unchanged

1.1.22 testRemoveAll_2()

Summary: Test removeAll(c) method with a null collection

Test Case Design

1. Create a new null HCollection (in this case a ListAdapter) and add some elements
2. Call removeAll(c)
3. Verify that NullPointerException is thrown and the list and the HCollection remains unchanged

Test Description: Test removeAll(c) method with a null collection

Pre-Condition: List has 4 elements, HCollection is null

Post-Condition: List has 2 elements, HCollection is null, NullPointerException is thrown.

Expected Results
List and HCollection remains unchanged, NullPointerException is thrown

1.1.23 testRetainAll_1()

Summary: Test retainAll(c) method with a valid collection.

Test Case Design

1. Create a new HCollection (in this case a ListAdapter) and add some elements that are contained in the list
2. Call retainAll(c) and verify it returns true
3. Verify that all the elements that are not in the HCollection are removed from the list

Test Description: Test retainAll(c) method with a valid collection.

Pre-Condition: List has 4 elements, HCollection has 2 elements.

Post-Condition: List has 2 elements, HCollection has 2 elements.

Expected Results
retainAll(c) returns true
List contains only elements that are present in the HCollection, HCollection remains unchanged

1.1.24 testRetainAll_2()

Summary: Test retainAll(c) method with a null collection.

Test Case Design

1. Create a new null HCollection
2. Call retainAll(c) and verify it throws NullPointerException

Test Description: Test retainAll(c) method with a null collection.

Pre-Condition: List has 4 elements, HCollection is null.

Post-Condition: List has 2 elements, HCollection is null.

Expected Results
List remains unchanged, HCollection remains unchanged, NullPointerException is thrown

1.1.25 testClear()

Summary: Test clear() method with a not empty List

Test Case Design

1. Verify that the elements are in the list
2. Call clear()
3. Verify that the elements are not in the list anymore

Test Description: Test clear() method with a not empty List

Pre-Condition: List has 2 elements

Post-Condition: List is empty

Expected Results
List is empty

1.1.26 testGet_1()

Summary: Test get(key) with 0<=index<size

Test Case Design

1. Get one element and verify its the correct one

Test Description: Test get(key) with 0<=index<size

Pre-Condition: List has 3 elements

Post-Condition: List has 3 elements

Expected Results
get(key) returns the correct value if the 0<=index<size, List remains unchanged

1.1.27 testGet_2()

Summary: Test get(key) with a invalid index

Test Case Design

1. Call get(index) with an invalid index and verify it throws IndexOutOfBoundsException

Test Description: Test get(key) with a invalid index

Pre-Condition: List has 3 elements

Post-Condition: List has 3 elements

Expected Results
List remains unchanged, get(index) throws IndexOutOfBoundsException

1.1.28 testSet_1()

Summary: Test set(index, element) with a valid index and a valid element

Test Case Design

1. Call set(index, element) with 0<=index<size and a not null element
2. Verify that set returns old element
3. Verify that the new element is set

Test Description: Test set(index, element) with a valid index and a valid element

Pre-Condition: List has 1 element

Post-Condition: List has 1 element, but it is different from the pre-condition

Expected Results
Set(index, element) returns true.
List has still one element, but it is changed.

1.1.29 testSet_2()

Summary: Test set(index, element) with a invalid index and a valid element

Test Case Design

1. Call set(index, element) with index<0||index>=size and a not null element
2. Verify that set throws IndexOutOfBoundsException

Test Description: Test set(index, element) with a invalid index and a valid element

Pre-Condition: List has 1 element

Post-Condition: List has 1 element (unchanged), IndexOutOfBoundsException is thrown

Expected Results
List remains unchanged, IndexOutOfBoundsException is thrown.

1.1.30 testSet_3()

Summary: Test set(index, element) with a valid index and a null element

Test Case Design

1. Call set(index, element) with 0<=index<size and a null element
2. Verify that set throws NullPointerException

Test Description: Test set(index, element) with a valid index and a null element

Pre-Condition: List has 1 element

Post-Condition: List has 1 element (unchanged), NullPointerException is thrown

Expected Results
List remains unchanged, NullPointerException is thrown.

1.1.31 testIndexOf_1()

Summary: Test indexOf(element) with a valid and invalid element (both not null)

Test Case Design

1. Call indexOf(element) with a valid element and verify it returns the correct index
2. Call indexOf(element) with a invalid element and verify it returns -1

Test Description: Test indexOf(element) with a valid and invalid element (both not null)

Pre-Condition: List has 2 elements

Post-Condition: List has 2 elements

Expected Results
indexOf(element)returns the index if it is present, -1 if it is not present

1.1.32 testIndexOf_2()

Summary: Test indexOf(element) with a null element

Test Case Design

1. Call indexOf(element) with a null element and verify it throws NullPointerException

Test Description: Test indexOf(element) with a null element

Pre-Condition: List has 2 elements

Post-Condition: List has 2 elements

Expected Results
indexOf(element) with null element throws NullPointerException

1.1.33 testLastIndexOf_1()

Summary: Test lastIndexOf(element) with a valid and non valid element (both not null)

Test Case Design

1. Call lastIndexOf(element) with different elements and verify it returns the last index if it is present or -1 if it is not.

Test Description: Test lastIndexOf(element) with a valid and non valid element (both not null)

Pre-Condition: List has 3 elements (two are duplicates)

Post-Condition: List has 3 elements

Expected Results
lastIndexOf(element) returns the last index if it is present, -1 if it is not present

1.1.34 testLastIndexOf_2()

Summary: Test lastIndexOf(element) with a null element

Test Case Design

1. Call lastIndexOf(element) with a null element
2. Verify it throws NullPointerException

Test Description: Test lastIndexOf(element) with a null element

Pre-Condition: List has 3 elements (two are duplicates)

Post-Condition: List has 3 elements, NullPointerException is thrown

Expected Results
lastIndexOf(element) throws NullPointerException

1.1.35 testHashCode()

Summary: Test hashCode()

Test Case Design

1. Verify that the same list has the same hashCode
2. Create a new List with an element
3. Verify that the two list have different hashCodes
4. Create a third empty list
5. Verify that the hashCode is equal to the first and different from the second
6. Add to the third list the same element of the second
7. Verify that the hashCode are equals

Test Description: Test hashCode()

Pre-Condition: First and third List are empty, second List has 1 element.

Post-Condition: First list is empty, second and third have 1 element (the same element).

Expected Results
list.hashCode() equals list.hashCode()
list2.hashCode() not equals list.hashCode()

(before adding an element to the second list)
list2.hashCode() not equals list3.hashCode()
list.hashCode() equals list3.hashCode()

(after adding the same element to the third list)
list2.hashCode() equals list3.hashCode()

1.1.36 testListIteratorStart()

Summary: Test listIterator() from the start

Test Case Design

1. Verify that iterator.next() returns correct elements
2. Remove that element by calling iterator.remove() and verify it has been removed from the list correctly

Test Description: Test listIterator() from the start

Pre-Condition: List has 6 elements

Post-Condition: List is empty

Expected Results
iterator.next() returns only elements that are contained in the List and the elements are in the correct posistions
List is empty.

1.1.37 testListIteratorEnd()

Summary: Test listIterator() from the end

Test Case Design

1. Verify that iterator.previous() returns correct elements

Test Description: Test listIterator() from the end

Pre-Condition: List has 6 elements

Post-Condition: List has 6 elements

Expected Results
iterator.previous() returns only elements that are contained in the List and the elements are in the correct posistions

1.1.38 testListIteratorSet()

Summary: Test listIterator() set method

Test Case Design

1. Traverse the list with the listIterator and call set for every cicle

Test Description: Test listIterator() set method

Pre-Condition: List has 6 elements

Post-Condition: List has 6 elements (all different from the start)

Expected Results
All elements in the List are changed.

1.1.39 testListIteratorAdd()

Summary: Test listIterator() add method

Test Case Design

1. Call iterator.next() and then iterator.add(element)
2. Verify the elemenent has been added in the correct position

Test Description: Test listIterator() add method

Pre-Condition: List has 1 elements

Post-Condition: List has 2 elements

Expected Results
List grows as a result of call of iterator.add(element).

1.2 SubList

1.2.1 testSubListConstructor()

Summary: Test SubList constructor with invalid indexes

Test Case Design

1. Call sublist constructor with invalid indexes and verify it throws IndexOutOfBoundsException

Test Description: Test SubList constructor with invalid index

Pre-Condition: List has 6 elements

Post-Condition: List has 6 elements

Expected Results
Sublist constructor throws IndexOutOfBoundsException.

1.2.2 testSubListChanges()

Summary: Test if changes in the List also happen in the SubList

Test Case Design

1. Add/remove object from the parent list
2. Verify that also SubList changed as a consequence

Test Description: Test if changes in the List also happen in the SubList

Pre-Condition: List has 6 elements

Post-Condition: List has 6 elements

Expected Results
List is changed, also SubList is changed as a result.

1.2.3 testSubListSize()

Summary: Test size() and isEmpty() of SubList

Test Case Design

1. Add elements to the list
2. Call sublist(fromIndex, toIndex)
3. Verify that size equals to (toIndex - fromIndex + 1)
4. Verify that isEmpty() returns false

Test Description: Test size() and isEmpty() of SubList

Pre-Condition: List has 6 elements, SubList has 4

Post-Condition: List has 6 elements, SubList has 4

Expected Results
List and SubList remains unchanged, size and isEmpty returns the correct size and false.

1.2.4 testSubListContains_1()

Summary: Test contains(o) of SubList with a valid and invalid element

Test Case Design

1. Call contains(element) with different elements and verify it returns the correct boolean value

Test Description: Test contains(o) of SubList with a valid and invalid element

Pre-Condition: List has 6 elements, SubList has 4 elements.

Post-Condition: List has 6 elements, SubList has 4 elements.

Expected Results
List and SubList remain unchanged, if the element is present contains(o) returns true, if it's not it returns false.

1.2.5 testSubListContains_2()

Summary: Test contains(o) of SubList with a null object

Test Case Design

1. Call contains(element) with null object
2. Verify it throws NullPointerException

Test Description: Test contains(o) of SubList with a null object

Pre-Condition: List has 6 elements, SubList has 4 elements.

Post-Condition: List has 6 elements, SubList has 4 elements.

Expected Results
List and SubList remain unchanged, contains(null) throws NullPointerException.

1.2.6 testSubListToArray()

Summary: Test toArray() of SubList

Test Case Design

1. Call toArray()
2. Compare the array to a test array with the elements that are supposed to be in it and verify they are equals

Test Description: Test toArray() of SubList

Pre-Condition: List has 6 elements, SubList has 4 elements.

Post-Condition: List has 6 elements, SubList has 4 elements.

Expected Results
List and SubList remains unchanged.
The return array and the test array are equals.

1.2.7 testSubListToArrayA_1()

Summary: Test toArray(a) of SubList with a null array

Test Case Design

1. Call toArray(a) with a null array
2. Verify it throws NullPointerException

Test Description: Test toArray(a) of SubList with a null array

Pre-Condition: List has 6 elements, SubList has 4 elements.

Post-Condition: List has 6 elements, SubList has 4 elements, NullPointerException is thrown.

Expected Results
List and SubList remains unchanged, NullPointerException is thrown

1.2.8 testSubListToArrayA_2()

Summary: Test toArray(a) of SubList with an array with the same size as the sublist

Test Case Design

1. Declare a new empty array with the same size as the sublist
2. Call toArray(a)
3. Verify it contains only the same elements as the sublist

Test Description: Test toArray(a) of SubList with an array with the same size as the sublist

Pre-Condition: List has 6 elements, SubList has 4 elements, array is empty.

Post-Condition: List has 6 elements, SubList has 4 elements, array has 4 elements.

Expected Results
List and SubList remains unchanged, array contains all the elements of the sublist.

1.2.9 testSubListToArrayA_3()

Summary: Test toArray(a) of SubList with an array with the size greater than the sublist

Test Case Design

1. Declare a new empty array with the size greater than the sublist
2. Call toArray(a)
3. Verify it contains the same elements as the sublist and a null object at the index sublist.size()

Test Description: Test toArray(a) of SubList with an array with the size greater than the sublist

Pre-Condition: List has 6 elements, SubList has 4 elements, array is empty.

Post-Condition: List has 6 elements, SubList has 4 elements, array has 5 elements.

Expected Results
List and SubList remains unchanged, array contains all the elements of the sublist plus the null element at the end.

1.2.10 testSubListAdd_1()

Summary: Test add(o) of SubList with a valid object

Test Case Design

1. Add an element to the sublist, this will add at the end of the SubList and shift the next elements in the orginial list
2. Verify that the element has been added both to to SubList and the List and the next items have been shifted correctly

Test Description: Test add(o) of SubList with a valid object

Pre-Condition: List has 6 elements, SubList has 4 elements.

Post-Condition: List has 7 elements, SubList has 4 elements.

Expected Results
The element has been added correctly and the items next to that in the list are shifted correcly.

1.2.11 testSubListAdd_2()

Summary: Test add(o) of SubList with a null object

Test Case Design

1. Add a null element to the sublist
2. Verify that it throws NullPointerException

Test Description: Test add(o) of SubList with a null object

Pre-Condition: List has 6 elements, SubList has 4 elements.

Post-Condition: List has 6 elements, SubList has 4 elements.

Expected Results
List and SubList remain unchanged, NullPointerException is thrown.

1.2.12 testSubListAddIndex_1()

Summary: Test add(index, o) of SubList with a valid object and a valid index

Test Case Design

1. Add an element to the sublist, this will add it at the given index of the SubList and shift the next elements
2. Verify that the element has been added both to to SubList and the List and the next items have been shifted correctly

Test Description: Test add(index, o) of SubList with a valid object and a valid index

Pre-Condition: List has 6 elements, SubList has 4 elements.

Post-Condition: List has 7 elements, SubList has 4 elements.

Expected Results
List and SubList changed as a result, List grows, SubList mantains his size.

1.2.13 testSubListAddIndex_2()

Summary: Test add(index, o) of SubList with a valid object and a invalid index

Test Case Design

1. Try adding an element to the sublist with add(index, o) and an invalid index
2. Verify it throws IndexOutOfBoundsException

Test Description: Test add(index, o) of SubList with a valid object and a invalid index

Pre-Condition: List has 6 elements, SubList has 4 elements.

Post-Condition: List has 6 elements, SubList has 4 elements, IndexOutOfBoundsException is thrown.

Expected Results
List and SubList remain unchanged, IndexOutOfBoundsException is thrown.

1.2.14 testSubListAddIndex_3()

Summary: Test add(index, o) of SubList with a null object and a valid index

Test Case Design

1. Try adding an element to the sublist with add(index, o) and an null element
2. Verify it throws NullPointerException

Test Description: Test add(index, o) of SubList with a null object and a valid index

Pre-Condition: List has 6 elements, SubList has 4 elements.

Post-Condition: List has 6 elements, SubList has 4 elements, NullPointerException is thrown.

Expected Results
List and SubList remain unchanged, NullPointerException is thrown.

1.2.15 testSubListContainsAll_1()

Summary: Test containsAll(c) of SubList with a valid collection

Test Case Design

1. Create a new HCollection (in this case ListAdapter) and add only elements contained in the List
2. Call containsAll(c) and verify that all elements are contained
3. Add to the HCollection an element that is not in the List
4. Call containsAll(c) and verify that returns false

Test Description: Test containsAll(c) of SubList with a valid collection

Pre-Condition: List has 6 elements, SubList has 4 elements, HCollection has 2 elements

Post-Condition: List has 6 elements, SubList has 4 elements, HCollection has 3 elements

Expected Results
List, SubList and HCollection remains unchanged. The first containsAll returns true, the second false.

1.2.16 testSubListContainsAll_2()

Summary: Test containsAll(c) of SubList with a null collection

Test Case Design

1. Create a new null HCollection (in this case ListAdapter)
2. Call containsAll(c) and verify it throws NullPointerException

Test Description: Test containsAll(c) of SubList with a null collection

Pre-Condition: List has 6 elements, SubList has 4 elements, HCollection is null

Post-Condition: List has 6 elements, SubList has 4 elements, HCollection is null, NullPointerException is thrown

Expected Results
List, SubList and HCollection remains unchanged. NullPointerException is thrown

1.2.17 testSubListAddAll_1()

Summary: Test addAll(c) of SubList with a valid HCollection

Test Case Design

1. Create a new HCollection (in this case ListAdapter) and add elements
2. Call addAll(c) and verify it returns true
3. Verify that all elements are added and the next elements on the list are shifted

Test Description: Test addAll(c) of SubList with a valid HCollection

Pre-Condition: List has 6 elements, SubList has 4 elements, HCollection has 2 elements.

Post-Condition: List has 8 elements, SubList has 4 elements, HCollection has 2 elements.

Expected Results
List grows as a result, SubList remains the same size but it changes, HCollection remains unchanged.

1.2.18 testSubListAddAll_2()

Summary: Test addAll(c) of SubList with a null HCollection

Test Case Design

1. Create a new null HCollection (in this case ListAdapter)
2. Call addAll(c) and verify it throws NullPointerException

Test Description: Test addAll(c) of SubList with a null HCollection

Pre-Condition: List has 6 elements, SubList has 4 elements, HCollection is null.

Post-Condition: List has 6 elements, SubList has 4 elements, HCollection is null.

Expected Results
List, SubList and HCollection remain unchanged, NullPointerException is thrown.

1.2.19 testSubListAddAllIndex_1()

Summary: Test addAll(index, c) of SubList with a valid HCollection and a valid index

Test Case Design

1. Create a new HCollection (in this case ListAdapter) and add elements
2. Call addAll(index, c) and verify all items are added correctly

Test Description: Test addAll(index, c) of SubList with a valid HCollection and a valid index

Pre-Condition: List has 6 elements, SubList has 4 elements, HCollection has 2 elements.

Post-Condition: List has 8 elements, SubList has 4 elements, HCollection has 2 elements.

Expected Results
HCollection remain unchanged, List grows, SubList doesnt change his size, but it is changed.

1.2.20 testSubListAddAllIndex_2()

Summary: Test addAll(index, c) of SubList with a null HCollection and a valid index

Test Case Design

1. Create a new null HCollection (in this case ListAdapter)
2. Call addAll(index, c) and verify it throws NullPointerException

Test Description: Test addAll(index, c) of SubList with a null HCollection and a valid index

Pre-Condition: List has 6 elements, SubList has 4 elements, HCollection is null.

Post-Condition: List has 6 elements, SubList has 4 elements, HCollection is null.

Expected Results
List, SubList and HCollection remain unchanged, NullPointerException is thrown.

1.2.21 testSubListAddAllIndex_3()

Summary: Test addAll(index, c) of SubList with a valid HCollection and a invalid index

Test Case Design

1. Create a new HCollection (in this case ListAdapter) and add some elements
2. Call addAll(index, c) with an invalid index and verify it throws IndexOutOfBoundsException

Test Description: Test addAll(index, c) of SubList with a valid HCollection and a invalid index

Pre-Condition: List has 6 elements, SubList has 4 elements, HCollection is null, IndexOutOfBoundsException is thrown.

Post-Condition: List has 6 elements, SubList has 4 elements, HCollection is null.

Expected Results
List, SubList and HCollection remain unchanged, IndexOutOfBoundsException is thrown.

1.2.22 testSubListRemoveAll_1()

Summary: Test removeAll(c) of SubList with a valid HCollection

Test Case Design

1. Create a new HCollection (in this case ListAdapter) and add elements that are present in the list
2. Call removeAll(index,c) and verify it returns true
3. Verify that all elements are removed

Test Description: Test removeAll(c) of SubList with a valid HCollection

Pre-Condition: List has 6 elements, SubList has 4 elements, HCollection has 2 elements.

Post-Condition: List has 4 elements, SubList has 4 elements, HCollection has 2 elements.

Expected Results
HCollection is unchanged, List shrinks and SubList changes but mantains his size.

1.2.23 testSubListRemoveAll_2()

Summary: Test removeAll(c) of SubList with a null HCollection

Test Case Design

1. Create a new null HCollection (in this case ListAdapter)
2. Call removeAll(index,c) and verify it throws NullPointerException

Test Description: Test removeAll(c) of SubList with a null HCollection

Pre-Condition: List has 6 elements, SubList has 4 elements, HCollection has 2 elements.

Post-Condition: List has 6 elements, SubList has 4 elements, HCollection has 2 elements.

Expected Results
List, SubList and HCollection remain unchanged, NullPointerException is thrown.

1.2.24 testSubListRetainAll_1()

Summary: Test retainAll(c) of SubList with a valid HCollection.

Test Case Design

1. Create a new HCollection (in this case ListAdapter) and add elements
2. Call retaineAll(index,c) and verify it returns true
3. Verify that all elements that are not in the HCollection are removed

Test Description: Test retainAll(c) of SubList with a valid HCollection.

Pre-Condition: List has 7 elements, SubList has 5 elements, HCollection has 3 elements

Post-Condition: List has 4 elements, SubList has 4 elements, HCollection has 3 elements

Expected Results
HCollection is unchanged, List and SubList change as a result of the call of retainAll.

1.2.25 testSubListRetainAll_2()

Summary: Test retainAll(c) of SubList with a null HCollection.

Test Case Design

1. Create a new null HCollection (in this case ListAdapter)
2. Call retaineAll(index,c) and verify it throws NullPointerException

Test Description: Test retainAll(c) of SubList with a null HCollection.

Pre-Condition: List has 7 elements, SubList has 5 elements, HCollection is null

Post-Condition: List has 7 elements, SubList has 5 elements, HCollection is null

Expected Results
List, SubList and HCollection is unchanged, NullPointerException is thrown.

1.2.26 testSubListClear()

Summary: Test clear() of SubList

Test Case Design

1. Call clear()
2. Verify that items are removed correctly and next one in the original list are shifted

Test Description: Test clear() of SubList

Pre-Condition: List has 8 elements, SubList has 5 elements.

Post-Condition: List has 3 elements, SubList has 3 elements.

Expected Results
after clear() all the elements are removed from the list, as a result the elements after the sublist are shifted.

1.2.27 testSubListHashCode()

Summary: Test hashCode() of SubList

Test Case Design

1. Verify that the same sublist has the same hashCode
2. Create a new SubList with an element
3. Verify that the two Sublist have different hashCodes
4. Create a third Sublist with the same element of the first
5. Verify that the hashCode is equal to the first and different from the second

Test Description: Test hashCode() of SubList

Pre-Condition: List has 7 elements, SubList has 4 elements.

Post-Condition: List has 7 elements, SubList has 4 elements.

Expected Results
sub.hashCode() equals sub.hashCode()
sub.hashCode() not equals sub2.hashCode()
sub2.hashCode() not equals sub3.hashCode()
sub.hashCode() equals sub3.hashCode()

1.2.28 testSubListSet_1()

Summary: Test set(index, o) of SubList with a valid index and a valid object

Test Case Design

1. Call set(index, element) and verify it returns the old element
2. Verify that the new element is set correctly

Test Description: Test set(index, o) of SubList with a valid index and a valid object

Pre-Condition: List has 6 elements, SubList has 4 elements.

Post-Condition: List has 6 elements, SubList has 4 elements, but one of the element has changed.

Expected Results
set(index, element) returns the old element and the list (and sublist) is changed

1.2.29 testSubListSet_2()

Summary: Test set(index, o) of SubList with a invalid index and a valid object

Test Case Design

1. Call set(index, element) with an invalid index and verify it throws IndexOutOfBoundsException

Test Description: Test set(index, o) of SubList with a invalid index and a valid object

Pre-Condition: List has 6 elements, SubList has 4 elements.

Post-Condition: List has 6 elements, SubList has 4 elements, IndexOutOfBoundsException is thrown.

Expected Results
List and SubList remains unchanged, IndexOutOfBoundsException is thrown.

1.2.30 testSubListSet_3()

Summary: Test set(index, o) of SubList with a valid index and a null object

Test Case Design

1. Call set(index, element) with a null object and verify it throws NullPointerException

Test Description: Test set(index, o) of SubList with a valid index and a null object

Pre-Condition: List has 6 elements, SubList has 4 elements.

Post-Condition: List has 6 elements, SubList has 4 elements, NullPointerException is thrown.

Expected Results
List and SubList remain unchanged, NullPointerException is thrown.

1.2.31 testSubListGet_1()

Summary: Test get(index) of SubList with a valid index

Test Case Design

1. Call get(index) with a valid index and verify it returns the correct element

Test Description: Test get(index) of SubList with a valid index

Pre-Condition: List has 6 elements, SubList has 4 elements.

Post-Condition: List has 6 elements, SubList has 4 elements.

Expected Results
List and SubList remain unchanged, get(index) returns the correct element.

1.2.32 testSubListGet_2()

Summary: Test get(index) of SubList with a invalid index

Test Case Design

1. Call get(index) with a invalid index and verify it throws IndexOutOfBoundsException

Test Description: Test get(index) of SubList with a invalid index

Pre-Condition: List has 6 elements, SubList has 4 elements.

Post-Condition: List has 6 elements, SubList has 4 elements, IndexOutOfBoundsException is thrown.

Expected Results
List and SubList remain unchanged, IndexOutOfBoundsException is thrown.

1.2.33 testSubListRemove_1()

Summary: : Test remove(index) of SubList with a valid index

Test Case Design

1. Call remove(index) and verify it returns the old element
2. Verify the next elements shifted

Test Description: Test remove() of SubList with a valid index

Pre-Condition: List has 6 elements, SubList has 4 elements

Post-Condition: List has 5 elements, SubList has 4 elements

Expected Results
remove(index) returns removed element, List and SubList are changed as a result, but SubList doesnt change its size.

1.2.34 testSubListRemove_2()

Summary: : Test remove(index) of SubList with a invalid index

Test Case Design

1. Call remove(index) and verify it throws IndexOutOfBoundsException

Test Description: Test remove(index) of SubList with a invalid index

Pre-Condition: List has 6 elements, SubList has 4 elements

Post-Condition: List has 6 elements, SubList has 4 elements, IndexOutOfBoundsException is thrown

Expected Results
List and SubList remain unchanged, IndexOutOfBoundsException is thrown.

1.2.35 testSubListIndexOf_1()

Summary: Test indexOf(element) of SubList with a valid and invalid object

Test Case Design

1. Call indexOf(element) with a valid object and verify it returns the first index of that element
2. Call indexOf(element) with a invalid object (not null) and verify it returns -1

Test Description: Test indexOf(element) of SubList with a valid and invalid object

Pre-Condition: List has 6 elements, SubList has 4 elements

Post-Condition: List has 6 elements, SubList has 4 elements

Expected Results
List and SubList remain unchanged, indexOf(element) returns first index of the element, returns -1 if the element is not in the list

1.2.36 testSubListIndexOf_2()

Summary: Test indexOf(element) of SubList with a null object

Test Case Design

1. Call indexOf(element) with a null object and verify it throws NullPointerException

Test Description: Test indexOf(element) of SubList with a null object

Pre-Condition: List has 6 elements, SubList has 4 elements

Post-Condition: List has 6 elements, SubList has 4 elements, NullPointerException is thrown

Expected Results
List and SubList remain unchanged, NullPointerException is thrown.

1.2.37 testSubListLastIndexOf_1()

Summary: Test lastIndexOf(element) of SubList with a non null element

Test Case Design

1. Call lastIndexOf(element) with a valid element and verify it returns the last index
2. Call lastIndexOf(element) with a invalid element (not null) and verify it returns -1

Test Description: Test lastIndexOf(element) of SubList with a non null element

Pre-Condition: List has 7 elements, SubList has 4 elements

Post-Condition: List has 7 elements, SubList has 4 elements

Expected Results
List and SubList remain unchanged, indexOf(element) returns last index of the element, returns -1 if the element is not in the list

1.2.38 testSubListLastIndexOf_2()

Summary: Test lastIndexOf(element) of SubList with a null element

Test Case Design

1. Call lastIndexOf(element) with a null element and verify it throws NullPointerException

Test Description: Test lastIndexOf(element) of SubList with a null element

Pre-Condition: List has 7 elements, SubList has 4 elements

Post-Condition: List has 7 elements, SubList has 4 elements, NullPointerException is thrown

Expected Results
List and SubList remain unchanged, NullPointerException is thrown.

1.2.39 testSubListListIterator()

Summary: Test listIterator() of SubList

Test Case Design

1. Verify that all elements returned by iterator1.next() are contained in the SubList and they are in the correct position
2. Verify that all elements returned by iterator2.previous() are contained in the SubList and they are in the correct position
3. Verify that nextIndex() when the cicle is done returns size()
4. Verify that previousIndex() when the cicle is done returns -1

Test Description: Test listIterator() of SubList

Pre-Condition: List has 6 elements, SubList has 4 elements.

Post-Condition: List has 6 elements, SubList has 4 elements.

Expected Results
iterator.next() returns only elements that are contained in the SubList and they are in the correct position
iterator.previous() returns only elements that are contained in the SubList and they are in the correct position

1.2.40 testSubListIterator()

Summary: Test iterator() of SubList

Test Case Design

1. Call iterator()
2. Verify that iterator.next() returns correct elements

Test Description: Test iterator() of SubList

Pre-Condition: List has 6 elements, SubList has 4 elements.

Post-Condition: List has 6 elements, SubList has 4 elements.

Expected Results
iterator.next() returns only elements that are contained in the SubList and they are in the correct position