TestSuite Documentation

PIETRO VALENTE

Foreword

The test suite of the program has been divided into several classes, more precisely one for each public class. In each of these classes, the methods are tested with different tests, each test verifies the correct functioning of a behavior of the method, verified using *Junit's assertEquals* method, the framework based on which the tests were produced.

Tests are run in the *TestRunner.java* file which verifies that they are correct.

What is tested is the behavior described in the *javadoc* documentation of the class, remember that the *ClassCastException* exception is not thrown and that it is the user's responsibility to make sure to enter all data of the same type in the classes. Classes are not *thread safe*.

The **execution variables of** this test suite are:

- Java version: 11.0.7
- JUnit version: 4.13
- Execution Environment Specifications: Linux-Ubuntu a 64 bit, with 8Gb RAM
- CLASSPATH: /usr/share/java/junit4.jar
- PATH: /usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:

Summary

COLLECTION TESTSUITE

boolean equals(Object o);

testEquals1	
Test Summary	Verifies that an instance of *the CollectionAdapter* is *equals* to itself.
Previous condition	An empty *CollectionAdapter* instance has been created.
Next condition	The *CollectionAdapter* instance remains empty.
Test run record	True

testEquals2	
Test Summary	Verifies that an Instance of *SetAdapter* is not *equals* to one of *CollectionAdapter*, even if both are empty.
Previous condition	You have created an empty *CollectionAdapter* instance, an empty *SetAdapter* Instance is created.
Next condition	The two *instances* remain empty.
Test run record	True

testEquals3	
Test Summary	Verifies that two *instances* of *the CollectionAdapter* with different elements are not *equals*.
Previous condition	You have created an empty *CollectionAdapter* instance, another *CollectionAdapter* Instance is created with element 1.
Next condition	The two *instances* remain empty and the other with an item.
Test run record	True

testEquals4	
Test Summary	Verifies that two *instances* of *the CollectionAdapter* with the same elements are *equals*.
Previous condition	You have created an instance of *CollectionAdapter* to which two items are added, another CollectionAdapter Instance is created with the same items.
Next condition	The two *instances* still contain the two elements.
Test run record	True

testEquals5	
Test Summary	Verifies that an instance of *the CollectionAdapter* is not *equals* to a *null* instance. Then the method returns false.
Previous condition	An empty *CollectionAdapter* instance has been created.
Next condition	The *CollectionAdapter* instance remains empty.
Test run record	True

LISTADAPTER TESTSUITE

void add(int index, Object element);

testAddIndex1	
Test Summary	Verifies that when *you insert "null"* as an element, *a NullPointerException* is launched.
Previous condition	An empty *ListAdapter* instance has been created.
Next condition	The exception was launched and the insertion was not successful.
Test run record	True

testAddIndex2	
Test Summary	Verifies that when a negative index is inserted*, indexOutOfBoundsException* is launched.
Previous condition	An empty *ListAdapter* instance has been created.
Next condition	The exception was launched and the insertion was not successful.
Test run record	True

testAddIndex3	
Test Summary	Verifies that when an index is inserted >= *size, indexOutOfBoundsException* is launched.
Previous condition	An empty *ListAdapter* instance has been created.
Next condition	The exception was launched and the insertion was not successful.
Test run record	True

testAddIndex4	
Test Summary	Verifies that when you insert a new item into a valid position, the *size* increases by one.

Previous condition	An empty *ListAdapter* instance has been created.
Next condition	The *ListAdapter* contains the inserted item.
Test run record	True

testAddIndex5	
Test Summary	Verifies that when you insert a new item in a location where one already existed, the item is overwritten.
Previous condition	You have created an instance of the *ListAdapter* with four items.
Next condition	The *ListAdapter* contains the previous items except for the overwritten item.
Test run record	True

boolean add(Object e);

testAdd1	
Test Summary	Verifies that when *"null"* is inserted*, a NullPointerException* is launched.
Previous condition	An empty *ListAdapter* instance has been created.
Next condition	The exception was launched and the insertion was not successful.
Test run record	True

testAdd2	
Test Summary	The *add* function returns *true* when a new item is inserted.
Previous condition	An empty *ListAdapter* instance has been created.
Next condition	The item was successfully added to the *ListAdapter*.
Test run record	True

testAdd3	
Test Summary	The *add* function returns *false* when inserting an element that was already present.
Previous condition	You created an empty *ListAdapter* instance where an item was added.
Next condition	The item is not added to the *ListAdapter* as expected, so an item remains in the *ListAdapter*.
Test run record	True

testAdd4	
Test Summary	The *isEmpty* function returns false if an element has been added.
Previous condition	You have created an empty *ListAdapter* instance to which an item has been added.
Next condition	The *ListAdapter* contains the only item that has been added.
Test run record	True

testAdd5	
Test Summary	When you insert a new element, the *contains* function actually contains that element.
Previous condition	You have created an empty *ListAdapter* instance to which an item has been added.
Next condition	The *ListAdapter* contains the only item that has been added.
Test run record	True

testAdd6	
Test Summary	When you insert a new element, which was already present, the *size* function still increases by one.

Previous condition	You have created an empty *ListAdapter* instance to which an item has been added. And then the same element is added again.
Next condition	The *ListAdapter* contains the two items that have been added.
Test run record	True

boolean addAll(int index, *Collection* c);

testAddAllIndex1	
Test Summary	Verifies that when a *“null" Collection* is inserted, *a NullPointerException* is launched.
Previous condition	An empty *ListAdapter* instance has been created.
Next condition	The exception was launched and the insertion was not successful.
Test run record	True

testAddAllIndex2	
Test Summary	Verifies that when an index is inserted >= *size, indexOutOfBoundsException* is launched.
Previous condition	An empty *ListAdapter* instance has been created.
Next condition	The exception was launched and the insertion was not successful.
Test run record	True

testAddAllIndex3	
Test Summary	Verifies that when you place a *Collection* in a valid index, the *size* increases correctly and the method returns *true*.
Previous condition	You have created an instance of the *ListAdapter* with 10 items. A Collection is created with the same 10 items.
Next condition	The *ListAdapter* contains 20 items.

Test run record	True

testAddAllIndex4	
Test Summary	Verifies that when a *Collection is* placed in a valid index, all items are added to the correct *index*.
Previous condition	You have created an instance of the *ListAdapter* with 10 items. A Collection is created with the same 10 items.
Next condition	The *ListAdapter* contains 20 items.
Test run record	True

testAddAllIndex5	
Test Summary	It verifies that the method works correctly even if the *Collection* has duplicates.
Previous condition	You have created an instance of the *ListAdapter* with 10 items. A Collection is created with the same 10 elements and in addition 2 duplicates.
Next condition	The *ListAdapter* contains 22 items.
Test run record	True

testAddAllIndex6	
Test Summary	It is verified that the method works correctly even if the *Collection* has duplicates and a few fewer elements.
Previous condition	You have created an instance of the *ListAdapter* with 10 items. A Collection is created with some items in common and others not.
Next condition	The *ListAdapter* contains 22 items.
Test run record	True

testAddAllIndex7	
Test Summary	Verifies that if the *Collection* is empty, the method returns *false.*

Previous condition	You have created an instance of the *ListAdapter* with 2 items. An empty Collection is created.
Next condition	The *ListAdapter* contains 2 items.
Test run record	True

boolean addAll(*Collection* c);

testAddAll1	
Test Summary	Verifies that when *"null"* is inserted*, a NullPointerException* is launched.
Previous condition	An empty *ListAdapter* instance has been created.
Next condition	The exception was launched and the insertion was not successful.
Test run record	True

testAddAll2	
Test Summary	When you insert a *Collection* with values identical to those already in the *ListAdapter*, the insertion returns *true.*
Previous condition	An empty *ListAdapter* instance has been created , a second instance of *the ListAdapter* is created. In both, numbers from 0 to 9 are added.
Next condition	The *ListAdapter* contains 20 items.
Test run record	True

testAddAll3	
Test Summary	Verifies that when you insert a *Collection*, all items are added to the end of the *ListAdapter*.
Previous condition	You have created an instance of the *ListAdapter* with 10 items. A Collection is created with the same 10 items.
Next condition	The *ListAdapter* contains 20 items.

Test run record	True

testAddAll4	
Test Summary	It verifies that the method works correctly even if the *Collection* has duplicates.
Previous condition	You have created an instance of the *ListAdapter* with 10 items. A Collection is created with the same 10 elements and in addition 2 duplicates.
Next condition	The *ListAdapter* contains 22 items.
Test run record	True

testAddAll5	
Test Summary	It is verified that the method works correctly even if the *Collection* has duplicates and a few fewer elements.
Previous condition	You have created an instance of the *ListAdapter* with 10 items. A Collection is created with some items in common and others not.
Next condition	The *ListAdapter* contains 22 items.
Test run record	True

boolean contains(Object o);

testContains	
Test Summary	Verifies that the *"null" contains* check launches *a NullPointerException*.
Previous condition	An empty ListAdapter instance has been created.
Next condition	The exception was thrown and the *contains* failed.
Test run record	True

boolean containsAll(*Collection* c);

testContainsAll1	
Test Summary	Verifies that the *containsAll* check for *"null"* launches *a NullPointerException*.
Previous condition	An empty ListAdapter instance has been created.
Next condition	The exception was thrown and the *contains* failed.
Test run record	True

testContainsAll2	
Test Summary	Verifies that if the size of the passed *Collection* is larger than that of the Instance, it returns *false*.
Previous condition	An empty *ListAdapter* instance has been created , a second *ListAdapter* Instance is created. In the second, the numbers 1 and 2 are added.
Next condition	The first instance remained empty.
Test run record	True

testContainsAll3	
Test Summary	The method is verified to work correctly, and then returns *true*.
Previous condition	You have created an instance of the *ListAdapter* that adds the numbers 1,2,3. A second ListAdapter Instance is created, in which only the numbers 1.2 are inserted.
Next condition	The numbers 1,2,3 remain in the *ListAdapter*.
Test run record	True

testContainsAll4	
Test Summary	It is verified that the method works correctly even if the *Collection* has duplicates and a few fewer elements.

Previous condition	You have created an instance of the *ListAdapter* with 10 items. A Collection is created with some items in common and others not.
Next condition	The *ListAdapter* contains 10 items.
Test run record	True

testContainsAll5	
Test Summary	Verifies that the method works correctly with two identical *ListAdapters* .
Previous condition	You have created an instance of the *ListAdapter* with 10 items. Another *ListAdapter* is created with 10 items.
Next condition	Both *ListAdapters* contain 10 items.
Test run record	True

testContainsAll6	
Test Summary	It verifies that the method works correctly even if the *Collection* has duplicates and the same number of items.
Previous condition	You have created an instance of the *ListAdapter* with 10 items. A Collection is created with the same number of elements but with duplicates of range 0-9.
Next condition	The *ListAdapter* contains 10 items.
Test run record	True

boolean equals(Object o);

testEquals1	
Test Summary	Verifies that an Instance of *ListAdapter* is *equals* to itself.
Previous condition	An empty ListAdapter instance has been created.
Next condition	The ListAdapter Instance remains empty.

Test run record	True

testEquals2	
Test Summary	Verifies that an Instance of the *CollectionAdapter* is not *equals* to one of the *ListAdapter*, even if both are empty.
Previous condition	You have created an empty *ListAdapter* instance, an empty *CollectionAdapter* Instance is created.
Next condition	The two *instances* remain empty.
Test run record	True

testEquals3	
Test Summary	Verifies that two *instances* of *the ListAdapter* with different elements are not *equals*.
Previous condition	An empty *ListAdapter* instance has been created, another *ListAdapter* Instance is created with item 1.
Next condition	The two *instances* remain empty and the other with an item.
Test run record	True

testEquals4	
Test Summary	It verifies that two *instances* of *the ListAdapter* with the same items, but different order, are not *equals*.
Previous condition	You have created an instance of *ListAdapter* to which two items are added, another ListAdapter instance is created with the same items.
Next condition	The two *instances* still contain the two elements.
Test run record	True
Test Summary	The method is verified to work correctly when *true* returns.

testEquals5	
Test Summary	The method is verified to work correctly when *true* returns.
Previous condition	You have created two *identical ListAdapter* instances with three items.
Next condition	The two *instances* with the same elements remain.
Test run record	True

testEquals6	
Test Summary	Verifies that an Instance of *ListAdapter* is not *equals* to a *null* instance. Then the method returns false.
Previous condition	An empty *ListAdapter* instance has been created.
Next condition	The ListAdapter Instance remains empty.
Test run record	True

int hashCode();

testHashCode	
Test Summary	Verifies that the *hashCode* value of a *ListAdapter* Instance is equal to the sum of the *hashCodes* of the various elements multiplied by 31.
Previous condition	You created an instance of the *ListAdapter* in which two strings were added.
Next condition	The ListAdapter instance with the two strings remains.
Test run record	True

boolean remove(Object o);

testRemove1	
Test Summary	Verifies that when *"null"* is removed*, a NullPointerException* is launched.
Previous condition	An empty *ListAdapter* instance has been created.

Next condition	The exception was launched and the removal was unsuccessful.
Test run record	True

testRemove2	
Test Summary	Verifies that when an item that is not present is removed, it is returned *false.*
Previous condition	An empty *ListAdapter* instance has been created.
Next condition	The ListAdapter Instance remains empty.
Test run record	True

testRemove3	
Test Summary	Verifies that when you remove an item that is present, *true* is returned.
Previous condition	You have created an instance of the *ListAdapter* to which an item is added.
Next condition	The ListAdapter Instance remains empty.
Test run record	True

testRemove4	
Test Summary	Verifies that when you remove an item that is present, the *size* decreases by one and the item is no longer present in the *ListAdapter.*
Previous condition	You have created an instance of the *ListAdapter* to which 3 items are added.
Next condition	The ListAdapter Instance remains with only 2 items.
Test run record	True

Object remove(int index);

testRemoveIndex1	
Test Summary	Verifies that when a negative index is removed*, the IndexOutOfBoundsException* is launched.
Previous condition	An empty *ListAdapter* instance has been created.
Next condition	The exception was launched and the removal was unsuccessful.
Test run record	True

testRemoveIndex2	
Test Summary	Verifies that when an index is removed >= *size, indexOutOfBoundsException* is launched.
Previous condition	An empty *ListAdapter* instance has been created.
Next condition	The exception was launched and the removal was unsuccessful.
Test run record	True

testRemoveIndex3	
Test Summary	The correct functioning of the method.
Previous condition	You have instantiated the *ListAdapter* with an item.
Next condition	*ListAdapter* is *empty.*
Test run record	True

boolean removeAll(*Collection* c);

testRemoveAll1	
Test Summary	Verifies that when *"null"* is removed*, a NullPointerException* is launched.
Previous condition	An empty *ListAdapter* instance has been created.
Next condition	The exception was launched and the removal was unsuccessful.
Test run record	True

testRemoveAll2	
Test Summary	Verifies that if the *Collection* actually contains values of the *ListAdapter* Instance, the removal is successful and the method returns true.
Previous condition	You have created an Instance of *ListAdapter* and an Instance of *Collection* with the same items.
Next condition	The ListAdapter instance is empty.
Test run record	True

testRemoveAll3	
Test Summary	Verifies that if the *Collection* contains the same items as the *ListAdapter* , the method returns *true* and the *ListAdapter* is *empty*.
Previous condition	You have created an Instance of *ListAdapter* and an Instance of *Collection* with the same items.
Next condition	The ListAdapter instance is empty.
Test run record	True

testRemoveAll4	
Test Summary	It is verified that the method works correctly even if the *Collection* has duplicates and a few fewer elements.

Previous condition	You have created an instance of the *ListAdapter* with 10 items. A Collection is created with some items in common and others not.
Next condition	The *ListAdapter* contains items that were not in the *Collection.*
Test run record	True

testRemoveAll5	
Test Summary	Verifies that if the *Collection* is empty, the method returns *false.*
Previous condition	You have created an instance of the *ListAdapter* with 2 items. An empty Collection is created.
Next condition	The *ListAdapter* contains 2 items.
Test run record	True

boolean retainAll(*Collection* c);

testRetainAll1	
Test Summary	Verifies *that the* retain of *"null"* launches *a NullPointerException*.
Previous condition	An empty *ListAdapter* instance has been created.
Next condition	The exception was launched and the *retain* was not successful.
Test run record	True

testRetainAll2	
Test Summary	Verifies that if the *Collection* and the *ListAdapter* are the same, the method returns *false*.
Previous condition	You have created an instance of *ListAdapter* and instance of *Collection* with the same items.
Next condition	*The ListAdapter* retains the same items it had before the method was called.
Test run record	True

testRetainAll3	
Test Summary	It verifies that if the *Collection* is empty and the *ListAdapter* is not, it becomes *empty.*
Previous condition	You have created an instance of the *ListAdapter* with an item and an empty *Collection* instance.
Next condition	*The ListAdapter* becomes *empty.*
Test run record	True

testRetainAll4	
Test Summary	It verifies that the method works correctly even if the *Collection* has duplicates.
Previous condition	You have created an instance of the *ListAdapter* with 10 items. A Collection is created with the same 10 items. Duplicates are added in both *instances*.
Next condition	The *ListAdapter* its elements that were also contained in the *Collection*.
Test run record	True

Object set(int index, Object element);

testSet1	
Test Summary	Verifies that when *you insert "null"* as an element, *a NullPointerException* is launched.
Previous condition	An empty *ListAdapter* instance has been created.
Next condition	The exception was launched and the insertion was not successful.
Test run record	True

testSet2	
Test Summary	Verifies that when a negative index is inserted*, indexOutOfBoundsException* is launched.

Previous condition	An empty *ListAdapter* instance has been created.
Next condition	The exception was launched and the insertion was not successful.
Test run record	True

List *subList*(int fromIndex, int toIndex);

testSubList1	
Test Summary	Verifies that when negative fromIndex is inserted*, indexOutOfBoundsException* is launched.
Previous condition	An empty *ListAdapter* instance has been created.
Next condition	The exception was thrown out and the *subList* failed.
Test run record	True

testSubList2	
Test Summary	Verifies that when you insert a toIndex greater than fromIndex*, indexOutOfBoundsException* is launched.
Previous condition	An empty *ListAdapter* instance has been created.
Next condition	The exception was thrown out and the *subList* failed.
Test run record	True

testSubList3	
Test Summary	Verifies that when negative toIndex is inserted*, indexOutOfBoundsException* is launched.
Previous condition	An empty *ListAdapter* instance has been created.
Next condition	The exception was thrown out and the *subList* failed.
Test run record	True

test*SubList*4	
Test Summary	Verifies that creating and indexing in a *subList* works.
Previous condition	You have created an instance of the *ListAdapter* with 10 items. A *subList* 2 through 6 is created. Next, an item is added to index 2.
Next condition	Both *the ListAdapter* and the *subList* have an extra element.
Test run record	True

test*SubList*5	
Test Summary	The *subList get* method is checked.
Previous condition	You have created an instance of the *ListAdapter* with 10 items. A *subList* 2 through 6 is created. Next, an item is added.
Next condition	Both *the ListAdapter* and the *subList* have the same items.
Test run record	True

test*SubList*6	
Test Summary	The *add* method of *subList* is checked.
Previous condition	You have created an instance of the *ListAdapter* with 10 items. A *subList* 2 through 6 is created.
Next condition	Both *the ListAdapter* and the *subList* have an extra element.
Test run record	True

test*SubList*7	
Test Summary	The *clear* method of *subList* is checked.
Previous condition	You have created an instance of the *ListAdapter* with 10 items. A *subList* 2 through 6 is created.

Next condition	The *subList* no longer has items, and the *ListAdapter* has the same items minus those that were present in the *subList*.
Test run record	True

test*SubList*8	
Test Summary	The *addAll* method of *subList* is checked.
Previous condition	You have created an instance of the *ListAdapter* with 10 items. A *subList* 2 through 6 is created. An Instance of *Collection* is created with items.
Next condition	Both *subList* and *ListAdapter* add items to the *Collection*.
Test run record	True

test*SubList*9	
Test Summary	The *addAll* method with *subList* index is checked .
Previous condition	You have created an instance of the *ListAdapter* with 10 items. A *subList* 2 through 6 is created. An Instance of *Collection* is created with items.
Next condition	Both *subList* and *ListAdapter* add *Collection* items to their locations.
Test run record	True

test*SubList*10	
Test Summary	The *contains* method of *subList* is checked.
Previous condition	You have created an instance of the *ListAdapter* with 10 items. A *subList* 2 through 6 is created.
Next condition	Both *the ListAdapter* and the *subList* have the same items.
Test run record	True

test*SubList*11	
Test Summary	The *containsAll* method of *subList* is checked.
Previous condition	You have created an instance of the *ListAdapter* with 10 items. A *subList* 2 through 6 is created. An Instance of *Collection* is created with items.
Next condition	Both *the ListAdapter* and the *subList* have the same items.
Test run record	True

test*SubList*12	
Test Summary	The *equals* method of *subList* is checked.
Previous condition	You have created an instance of the *ListAdapter* with 10 items. A *subList* 2 through 6 is created. Another *ListAdapter* Instance is created with the same *subList items*.
Next condition	Both *the ListAdapter* and the *subList* have the same items.
Test run record	True

test*SubList*13	
Test Summary	The first item in the *subList* is correct is checked.
Previous condition	You have created an instance of the *ListAdapter* with 10 items. A *subList* 2 through 6 is created.
Next condition	Both *the ListAdapter* and the *subList* have the same items.
Test run record	True

test*SubList*14	
Test Summary	The *indexOf* method of *subList* is checked.
Previous condition	You have created an instance of the *ListAdapter* with 10 items. A *subList* 2 through 6 is created.
Next condition	Both *the ListAdapter* and the *subList* have the same items.

Test run record	True

testSubList15	
Test Summary	The lastIndexOf method of *subList* is checked.
Previous condition	You have created an instance of the *ListAdapter* with 10 items. A *subList* 2 through 6 is created.
Next condition	Both *the ListAdapter* and the *subList* have the same items.
Test run record	True

testSubList16	
Test Summary	The *remove* method of *subList* is checked.
Previous condition	You have created an instance of the *ListAdapter* with 10 items. A *subList* 2 through 6 is created.
Next condition	Both *the ListAdapter* and the *subList* have the same items, except the one that has been removed.
Test run record	True

testSubList17	
Test Summary	The *remove* method of *subList* is checked.
Previous condition	You have created an instance of the *ListAdapter* with 5 items. A *subList* from 2 to 4 is created.
Next condition	Both *the ListAdapter* and the *subList* have the same items, except those that have been removed.
Test run record	True

test*SubList*18	
Test Summary	The *removeAll* method of *subList* is checked.
Previous condition	You have created an instance of the *ListAdapter* with 10 items. A *subList* 2 through 6 is created. An Instance of *Collection* is created with items.
Next condition	Both *the ListAdapter* and the *subList* have the same items, except those that have been removed.
Test run record	True

test*SubList*19	
Test Summary	The *retainAll* method of *subList* is checked.
Previous condition	You have created an instance of the *ListAdapter* with 10 items. A *subList* 2 through 6 is created. An Instance of *Collection* is created with items.
Next condition	Both *the ListAdapter* and the *subList* have the same items, except those that have been removed.
Test run record	True

test*SubList*20	
Test Summary	The *subList set* method is checked.
Previous condition	You have created an instance of the *ListAdapter* with 10 items. A *subList* 2 through 6 is created.
Next condition	Both *the ListAdapter* and the *subList* have the same items except the modified one.
Test run record	True

test*SubList*21	
Test Summary	The *toArray* method of *subList* is checked.

Previous condition	You have created an instance of the *ListAdapter* with 10 items. A *subList* 2 through 6 is created.
Next condition	Both *the ListAdapter* and the *subList* have the same items.
Test run record	True

Object[] toArray();

testToArray	
Test Summary	Verifies that the method returns an array that contains all the elements.
Previous condition	You have created an instance of the *ListAdapter* with items.
Next condition	The *ListAdapter* still has the same elements.
Test run record	True

Object[] toArray(Object[] a);

testToArrayObjects	
Test Summary	Verifies that the method returns an array that contains all the elements.
Previous condition	You have created an instance of the *ListAdapter* with items.
Next condition	The *ListAdapter* still has the same elements.
Test run record	True

SETADAPTER TESTSUITE

boolean add(Object e);

testAdd1	
Test Summary	Verifies that when *"null"* is inserted*, a NullPointerException* is launched.
Previous condition	An empty *SetAdapter* instance has been created.
Next condition	The exception was launched and the insertion was not successful.
Test run record	True

testAdd2	
Test Summary	The *add* function returns *true* when a new item is inserted.
Previous condition	An empty *SetAdapter* instance has been created.
Next condition	The item was successfully added to the *SetAdapter*.
Test run record	True

testAdd3	
Test Summary	The *add* function returns *false* when inserting an element that was already present.
Previous condition	You created an empty *SetAdapter* instance where an item was added.
Next condition	The item is not added to the *SetAdapter* as expected, so an item remains in the *SetAdapter*.
Test run record	True

testAdd4	
Test Summary	The *isEmpty* function returns false if an element has been added.

Previous condition	You have created an empty *SetAdapter* instance to which an item has been added.
Next condition	The *SetAdapter* contains the only element that has been added.
Test run record	True

testAdd5	
Test Summary	When you insert a new element, the *contains* function actually contains that element.
Previous condition	You have created an empty *SetAdapter* instance to which an item has been added.
Next condition	The *SetAdapter* contains the only element that has been added.
Test run record	True

testAdd6	
Test Summary	When you insert a new element, the *size* function changes from 0 to 1.
Previous condition	You have created an empty *SetAdapter* instance to which an item has been added.
Next condition	The *SetAdapter* contains the only element that has been added.
Test run record	True

boolean addAll(*Collection* c);

testAddAll1	
Test Summary	Verifies that when *"null"* is inserted*, a NullPointerException* is launched.
Previous condition	An empty *SetAdapter* instance has been created.
Next condition	The exception was launched and the insertion was not successful.
Test run record	True

testAddAll2	
Test Summary	When you insert a *Collection* with values identical to those already in the *SetAdapter*, the insertion returns *false*, because the *Set* has not been modified.
Previous condition	An empty *SetAdapter* instance has been created, a second *SetAdapter* Instance is created. In both, numbers from 0 to 9 are added.
Next condition	The *SetAdapter* still contains only elements 0 through 9.
Test run record	True

testAddAll3	
Test Summary	When you insert a *Collection* with values other than those already in the *SetAdapter*, the insertion returns *true*, because the *Set* has been modified.
Previous condition	An empty *SetAdapter* instance has been created , a second *SetAdapter* Instance is created. In both, numbers from 0 to 9 are added. Element 2 is removed from the first.
Next condition	The first Instance also contains 2.
Test run record	True

testAddAll4	
Test Summary	When a *Collection* is inserted , the size changes.
Previous condition	An empty *SetAdapter* instance has been created, a second *SetAdapter* Instance is created. In the second, the numbers from 0 to 9 are added.
Next condition	The *size* of the first Instance is equal to 10.
Test run record	True

testAddAll5	
Test Summary	When a *Collection is* inserted , the size changes, inserting the duplicates only once.

Previous condition	An empty *SetAdapter* instance has been created , a second *SetAdapter* Instance is created. In the second, the numbers from 0 to 9 are added. In addition, 2/3 values are added to the second as duplicates.
Next condition	The *size* of the first Instance is equal to 10. Because duplicates have not been added.
Test run record	True

testAddAll6	
Test Summary	When inserting a *Collection* with duplicates I verify that the elements are exactly what you expect.
Previous condition	An empty *SetAdapter* instance has been created. A Collection is created with duplicates.
Next condition	There are the elements of the *Collection* that were not already present and those of the *SetAdapter* previously present,
Test run record	True

boolean contains(Object o);

testContains	
Test Summary	Verifies that the *"null" contains* check launches *a NullPointerException*.
Previous condition	An empty *SetAdapter* instance has been created.
Next condition	The exception was thrown and the *contains* failed.
Test run record	True

boolean containsAll(*Collection* c);

testContainsAll1	
Test Summary	Verifies that the *containsAll* check for *"null"* launches *a NullPointerException*.
Previous condition	An empty *SetAdapter* instance has been created.

Next condition	The exception was thrown and the *contains* failed.
Test run record	True

testContainsAll2	
Test Summary	Verifies that if the size of the passed *Collection* is larger than that of the Instance, it returns *false*.
Previous condition	An empty *SetAdapter* instance has been created, a second *SetAdapter* Instance is created. In the second, the numbers 1 and 2 are added.
Next condition	The first instance remained empty.
Test run record	True

testContainsAll3	
Test Summary	The method is verified to work correctly, and then returns *true*.
Previous condition	You have created an instance of the *SetAdapter* that adds the numbers 1,2,3. A second Instance of *SetAdapter* is created, in which only the numbers 1.2 are inserted.
Next condition	In the *SetAdapter* remain the numbers 1,2,3.
Test run record	True

testContainsAll4	
Test Summary	Verifies that if the *Collection* has duplicates, it is returned *false*.
Previous condition	You have created an instance of the *SetAdapter* that inserts the numbers from 0 to 9. An Instance of *Collection* with duplicates and range 0-7 is created.
Next condition	The *SetAdapter* still contains the numbers from 0 to 9.
Test run record	True

boolean equals(Object o);

testEquals1	
Test Summary	Verifies that an Instance of *SetAdapter* is *equals* to itself.
Previous condition	An empty *SetAdapter* instance has been created.
Next condition	The SetAdapter Instance remains empty.
Test run record	True

testEquals2	
Test Summary	Verifies that an Instance of *collectionAdapter* is not *equals* to one of *SetAdapter*, even if both are empty.
Previous condition	An empty *SetAdapter* instance has been created, an empty *CollectionAdapter* Instance is created.
Next condition	The two *instances* remain empty.
Test run record	True

testEquals3	
Test Summary	Verifies that two *instances* of *SetAdapter* with different elements are not *equals*.
Previous condition	An empty *SetAdapter* instance has been created, another *SetAdapter* Instance is created with element 1.
Next condition	The two *instances* remain empty and the other with an item.
Test run record	True

testEquals4	
Test Summary	Verifies that two *instances* of *SetAdapter* with the same elements are *equals*.

Previous condition	You have created an instance of *SetAdapter* to which two items are added, another Instance of *SetAdapter* is created with the same elements.
Next condition	The two *instances* still contain the two elements.
Test run record	True

testEquals5	
Test Summary	Verifies that an Instance of *setAdapter* is not *equals* to a *null* instance. Then the method returns false.
Previous condition	An empty *SetAdapter* instance has been created.
Next condition	The SetAdapter Instance remains empty.
Test run record	True

int hashCode();

testHashCode	
Test Summary	Verifies that the *hashCode* value of an Instance of *SetAdapter* is equal to the sum of the *hashCodes* of the various elements.
Previous condition	You have created an instance of the *SetAdapter* in which three strings have been added.
Next condition	The *SetAdapter* instance with the three strings remains.
Test run record	True

boolean remove(Object o);

testRemove1	
Test Summary	Verifies that when *"null"* is removed*, a NullPointerException* is launched.
Previous condition	An empty *SetAdapter* instance has been created.
Next condition	The exception was launched and the removal was unsuccessful.

Test run record	True

testRemove2	
Test Summary	Verifies that when an item that is not present is removed, it is returned *false.*
Previous condition	An empty *SetAdapter* instance has been created.
Next condition	The SetAdapter Instance remains empty.
Test run record	True

testRemove3	
Test Summary	Verifies that when you remove an item that is present, true is returned.
Previous condition	You have created an instance of the *SetAdapter* to which an item is added.
Next condition	The SetAdapter Instance remains empty.
Test run record	True

testRemove4	
Test Summary	Verifies that after a removal, the number of items is the correct one.
Previous condition	You have created an instance of *SetAdapter* with 3 elements and one of them is removed.
Next condition	The *SetAdapter* has 2 elements.
Test run record	True

testRemove5	
Test Summary	Verifies that when you remove an item, it is no longer contained.
Previous condition	You have created an instance of *SetAdapter* with 3 items and one of them is removed.

Next condition	The *SetAdapter* has 2 elements.
Test run record	True

boolean removeAll(*Collection* c);

testRemoveAll1	
Test Summary	Verifies that when *"null"* is removed*, a NullPointerException* is launched.
Previous condition	An empty *SetAdapter* instance has been created.
Next condition	The exception was launched and the removal was unsuccessful.
Test run record	True

testRemoveAll2	
Test Summary	Verifies that if the *Collection* actually contains *setadapter* Instance values, the removal is successful and the method returns true.
Previous condition	You created an Instance of *SetAdapter* and an Instance of *Collection* with the same elements.
Next condition	The *SetAdapter* Instance is empty.
Test run record	True

testRemoveAll3	
Test Summary	Verifies that the method works correctly even if the *setadapter* and *collection* values are not the same.
Previous condition	You have created an instance of *SetAdapter* and an instance of *Collection* with some values that are the same and some that are not.
Next condition	*SetAdapter* contains only those items that were not present in the *Collection*.
Test run record	True

testRemoveAll4	
Test Summary	It is verified that the method works correctly even if there are duplicates in the *Collection*.
Previous condition	You have created an instance of *SetAdapter* and an instance of *Collection* with some values that are the same and some that are not.
Next condition	*SetAdapter* contains only those items that were not present in the *Collection*.
Test run record	True

boolean retainAll(*Collection* c);

testRetainAll1	
Test Summary	Verifies that al *retain* of *"null"* launches *NullPointerException*.
Previous condition	An empty *SetAdapter* instance has been created.
Next condition	The exception was launched and the *retain* was not successful.
Test run record	True

testRetainAll2	
Test Summary	Verifies that if the *Collection* and *setAdapter* are the same, the method returns *false*.
Previous condition	You have created an instance of *SetAdapter* and an instance of *Collection* with the same elements.
Next condition	*SetAdapter* retains the same elements it had before the method was called.
Test run record	True

testRetainAll3	
Test Summary	Verifies that the method works correctly even if the setadapter and collection values are not the same .

Previous condition	You have created an instance of *SetAdapter* and an instance of *Collection* with some values that are the same and some that are not.
Next condition	*SetAdapter* contains only those items that were not present in the *Collection*.
Test run record	True

testRetainAll4	
Test Summary	It is verified that the method works correctly even if there are duplicates in the *Collection*.
Previous condition	You have created an instance of *SetAdapter* and an instance of *Collection* with some values that are the same and some that are not.
Next condition	*SetAdapter* contains only those items that were not present in the *Collection*.
Test run record	True

Object[] toArray();

testToArray	
Test Summary	Verifies that the method returns an array that contains all the elements.
Previous condition	You have created an instance of *setadapter* with elements.
Next condition	The *SetAdapter* still has the same elements.
Test run record	True

Object[] toArray(Object[] a);

testToArrayObjects	
Test Summary	Verifies that the method returns an array that contains all the elements.
Previous condition	You have created an instance of *setadapter* with elements.
Next condition	The *SetAdapter* still has the same elements.

Test run record	True

MAPADAPTER TESTSUITE

boolean containsKey(Object key);

testContainsKey	
Test Summary	Verifies that the *"null" contains a NullPointerException*.
Previous condition	An empty *MapAdapter* instance has been created.
Next condition	The exception was thrown and the *contains* failed.
Test run record	True

boolean containsValue(Object value);

testContainsValues	
Test Summary	Verifies that the *"null" contains a NullPointerException*.
Previous condition	An empty *MapAdapter* instance has been created.
Next condition	The exception was thrown and the *contains* failed.
Test run record	True

Set entrySet();

testEntrySet	
Test Summary	Verifies that the method returns a *Set* of all *Map.Entries* in the *MapAdapter*.
Previous condition	You have created an instance of *MapAdapter* in which 5 pairs have been added.
Next condition	The *MapAdapter* Instance with the 5 pairs remains.
Test run record	True

boolean equals(Object o);

testEquals1	
Test Summary	Verifies that an Instance of *MapAdapter* is *equals* to itself.
Previous condition	An empty *MapAdapter* instance has been created.
Next condition	The *MapAdapter* Instance remains empty.
Test run record	True

testEquals2	
Test Summary	Verifies that a *CollectionAdapter* Instance is not *equals* to a *MapAdapter* Instance, even if both are empty.
Previous condition	You have created an empty *MapAdapter* instance, an empty *CollectionAdapter* Instance is created.
Next condition	The two *instances* remain empty.
Test run record	True

testEquals3	
Test Summary	Verifies that two *instances* of *MapAdapter* with different elements are not *equals*.
Previous condition	An empty *MapAdapter* instance has been created, another *MapAdapter* Instance is created with element 1.
Next condition	The two *instances* remain empty and the other with an item.
Test run record	True

testEquals4	
Test Summary	Verifies that two *instances* of *MapAdapter* with the same elements are *equals*.

Previous condition	You have created an instance of *MapAdapter* to which two elements are added, another MapAdapter Instance is created with the same elements.
Next condition	The two *instances* still contain the two elements.
Test run record	True

testEquals5	
Test Summary	Verifies that an Instance of *MapAdapter* is not equals to another Instance of *MapAdapter* that has the same keys but different values.
Previous condition	You have created an instance of *MapAdapter* to which two elements are added, another MapAdapter Instance is created with elements that have the same key but different values.
Next condition	The *MapAdapter* Instance with its values remains.
Test run record	True

testEquals6	
Test Summary	Verifies that a *MapAdapter* Instance is not *equals* to a *Null* Instance. Then the method returns false.
Previous condition	An empty *MapAdapter* instance has been created.
Next condition	The MapAdapter Instance remains empty.
Test run record	True

testEquals7	
Test Summary	Verifies that an Instance of *MapAdapter* is not *equal to* another Instance of *MapAdapter* that has the same values but different keys.
Previous condition	You have created an instance of *MapAdapter* to which two elements are added, another MapAdapter Instance is created with elements that have the same values but different keys.
Next condition	The *MapAdapter* Instance with its values remains.

Test run record	True

Object get(Object key);

testGet	
Test Summary	Verifies that the null *launch* of *"null"* is *nullPointerException*.
Previous condition	An empty *MapAdapter* instance has been created.
Next condition	The exception was thrown and the get was not successful.
Test run record	True

int hashCode();

testHashCode	
Test Summary	Verifies that the *hashCode* value of a *MapAdapter* Instance is equal to the sum of the *hashCodes* of the various elements.
Previous condition	You have created an instance of *MapAdapter* in which three strings have been added.
Next condition	The *MapAdapter* instance with the three strings remains.
Test run record	True

Set keySet();

testKeySet	
Test Summary	Verifies that the method returns a *Set* of all keys in the *MapAdapter*.
Previous condition	You have created an instance of *MapAdapter* in which 5 pairs have been added.
Next condition	The *MapAdapter* Instance with the 5 pairs remains.
Test run record	True

Object put(Object key, Object value);

testPut1	
Test Summary	Verifies that the *null put* as key launches *NullPointerException*.
Previous condition	An empty *MapAdapter* instance has been created.
Next condition	The exception was thrown and the *put* failed.
Test run record	True

testPut2	
Test Summary	Verifies that the *null put* as value launches *nullPointerException*.
Previous condition	An empty *MapAdapter* instance has been created.
Next condition	The exception was thrown and the *put* failed.
Test run record	True

Object remove(Object key);

testRemove	
Test Summary	Verifies that when *"null"* is removed*, a NullPointerException* is launched.
Previous condition	An empty *MapAdapter* instance has been created.
Next condition	The exception was launched and the removal was unsuccessful.
Test run record	True

Collection values();

testValues	
Test Summary	Verifies that the method returns a *Collection* of all values in the *MapAdapter*.
Previous condition	You have created an instance of *MapAdapter* in which 6 pairs have been added.
Next condition	The *MapAdapter* Instance with the 6 pairs remains.
Test run record	True

HASHITERATORADAPTER TESTSUITE

Object next();

testNext1	
Test Summary	It verifies that the *next* returns the expected key.
Previous condition	An instance of *Hashtable* has been created to which 5 pairs have been inserted. An Instance of *HashIteratorAdapter* is created on the keys for this *Hashtable.*
Next condition	The Instance of *HashIteratorAdapter* remains , and the *Hastable* still has the same values.
Test run record	True

testNext2	
Test Summary	Verifies that the *next* returns the expected value.
Previous condition	An instance of *Hashtable* has been created to which 5 pairs have been inserted. An Instance of *HashIteratorAdapter* is created on the values for this *Hashtable.*
Next condition	The Instance of *HashIteratorAdapter* remains , and the *Hastable* still has the same values.
Test run record	True

testNext3	
Test Summary	It verifies that the *next* returns the expected *Map.Entry*.
Previous condition	An instance of *Hashtable* has been created to which 5 pairs have been inserted. An Instance of *HashIteratorAdapter* is created on the *Map.Entry* for this *Hashtable.*
Next condition	The Instance of *HashIteratorAdapter* remains , and the *Hastable* still has the same values.
Test run record	True

void remove();

testRemove1	
Test Summary	Verifies that twice *remove* throws the *"IllegalStateException" exception.*
Previous condition	An instance of *Hashtable* has been created to which 5 pairs have been inserted. An Instance of *HashIteratorAdapter* is created on the keys for this *Hashtable.*
Next condition	The exception was launched and the second removal was unsuccessful.
Test run record	True

testRemove2	
Test Summary	Verifies that once a key is removed, it is no longer present in the *iterated Hashtable* .
Previous condition	An instance of *Hashtable* has been created to which 5 pairs have been inserted. An Instance of *HashIteratorAdapter* is created on the keys for this *Hashtable.*
Next condition	The Instance of *HashIteratorAdapter* remains , and the *Hastable* still has the same values, minus the removed one.
Test run record	True

testRemove3	
Test Summary	Verifies that once a value is removed, it is no longer present in the *iterated Hashtable*, or there is one less copy.
Previous condition	An instance of *Hashtable* has been created to which 5 pairs have been inserted. An Instance of *HashIteratorAdapter* is created on the values for this *Hashtable.*
Next condition	The Instance of *HashIteratorAdapter* remains , and the *Hastable* still has the same values, minus the removed one.
Test run record	True

testRemove4	
Test Summary	It is verified that once a *Map.Entry* is removed , it is no longer present in the *iterated Hashtable*.
Previous condition	An instance of *Hashtable* has been created to which 5 pairs have been inserted. An Instance of *HashIteratorAdapter* is created on the *Map.Entry* for this *Hashtable.*
Next condition	The Instance of *HashIteratorAdapter* remains , and the *Hastable* still has the same values, minus the removed one.
Test run record	True

LISTITERATORADAPTER TESTSUITE

boolean hasNext();

testHasNext1	
Test Summary	Verifies that the method returns *true* if the iterator still has items.
Previous condition	You have created an instance of *Vector* to which 10 items have been inserted. An Instance of *listIteratorAdapter* is created for this *Vector.*
Next condition	The Instance of *listIteratorAdapter* remains , and the *Vector* still has the same values.
Test run record	True

testHasNext2	
Test Summary	Verifies that the method returns *false* if the iterator has no elements.
Previous condition	You have created an instance of *Vector* to which 10 elements have been inserted. An Instance of *listIteratorAdapter* is created for this *Vector.*
Next condition	The Instance of *listIteratorAdapter* remains , and the *Vector* still has the same values.
Test run record	True

Object next();

testNext1	
Test Summary	Verifies that calling the method without having elements throws the *NoSuchElementException* exception.
Previous condition	An instance of *Hashtable* has been created to which 5 pairs have been inserted. An Instance of *HashIteratorAdapter* is created on the *Map.Entry* for this *Hashtable.*
Next condition	The exception was launched and the *next* one was not successful.
Test run record	True

testNext2	
Test Summary	Verifies that the *next* returns the expected element.
Previous condition	An instance of *Hashtable* has been created to which 5 pairs have been inserted. An Instance of *HashIteratorAdapter* is created on the *Map.Entry* for this *Hashtable.*
Next condition	The Instance of *HashIteratorAdapter* remains , and the *Hastable* still has the same values, minus the removed one.
Test run record	True

boolean hasPrevious();

testHasPrevioust1	
Test Summary	It verifies that the method returns *false* if the iterator has no items before.
Previous condition	You have created an instance of *Vector* to which 10 elements have been inserted. An Instance of *listIteratorAdapter* is created for this *Vector.*
Next condition	The Instance of *listIteratorAdapter* remains , and the *Vector* still has the same values.
Test run record	True

testHasPrevioust2	
Test Summary	Verifies that the method returns *true* if the iterator has items before.
Previous condition	You have created an instance of *Vector* to which 10 elements have been inserted. An Instance of *listIteratorAdapter* is created for this *Vector.*
Next condition	The Instance of *listIteratorAdapter* remains , and the *Vector* still has the same values.
Test run record	True

Object previous();

testPrevious1	
Test Summary	It verifies that calling the method without having elements first throws the *NoSuchElementException*.
Previous condition	You have created an instance of *Vector* to which 10 elements have been inserted. An Instance of *listIteratorAdapter* is created for this *Vector.*
Next condition	The exception was made and the *previous* one was not successful.
Test run record	True

testPrevious2	
Test Summary	The *previous verifies* that the previous returns the expected element.
Previous condition	You have created an instance of *Vector* to which 10 elements have been inserted. An Instance of *listIteratorAdapter* is created for this *Vector.*
Next condition	The Instance of *listIteratorAdapter* remains , and the *Vector* still has the same values.
Test run record	True

void remove();

testRemove1	
Test Summary	Verifies that twice *remove* throws the *IllegalStateException.*
Previous condition	You have created an instance of *Vector* to which 10 elements have been inserted. An Instance of *listIteratorAdapter* is created for this *Vector.*
Next condition	The exception was launched and the second removal was unsuccessful.
Test run record	True

testRemove2	
Test Summary	It is verified that once a key is removed after a *next* it is no longer present in the iterated *Hashtable*.
Previous condition	You have created an instance of *Vector* to which 10 elements have been inserted. An Instance of *listIteratorAdapter* is created for this *Vector.*
Next condition	The *ListIteratorAdapter* Instance remains , and the *Vector* still has the same values, minus the one removed.
Test run record	True

testRemove3	
Test Summary	It verifies that once a key is removed after a *previous* one, it is no longer present in the *iterated Hashtable* .
Previous condition	You have created an instance of *Vector* to which 10 elements have been inserted. An Instance of *listIteratorAdapter* is created for this *Vector.*
Next condition	The *ListIteratorAdapter* Instance remains , and the *Vector* still has the same values, minus the one removed.
Test run record	True

void set(Object e);

testSet1	
Test Summary	It verifies that if it was not previously called neither *next* nor *previous*, the *IllegalStateException* exception is thrown.
Previous condition	You have created an instance of *Vector* to which 10 elements have been inserted. An Instance of *listIteratorAdapter* is created for this *Vector.*
Next condition	The exception was thrown and the *set* failed.
Test run record	True

testSet2	
Test Summary	Verifies that if the set is run, the previous value is no longer present in the *Vector*.
Previous condition	You have created an instance of *Vector* to which 10 elements have been inserted. An Instance of *listIteratorAdapter* is created for this *Vector.*
Next condition	The *ListIteratorAdapter* Instance remains , and the *Vector* still has the same values, except for the modified one.
Test run record	True

testSet3	
Test Summary	Verifies that if the *set* is run, the new value is present in the *Vector.*
Previous condition	You have created an instance of *Vector* to which 10 elements have been inserted. An Instance of *listIteratorAdapter* is created for this *Vector.*
Next condition	The *ListIteratorAdapter* Instance remains , and the *Vector* still has the same values, except for the modified one.
Test run record	True

testSet4	
Test Summary	Verifies that if the *set* is run, the new value is present in the *Vector.*
Previous condition	You have created an instance of *Vector* to which 10 elements have been inserted. An Instance of *listIteratorAdapter* is created for this *Vector.*
Next condition	The *ListIteratorAdapter* Instance remains , and the *Vector* still has the same values, except for the modified one.
Test run record	True

testSet5	
Test Summary	Verifies that if the *set* is run, the previous value is no longer present in the *Vector*.
Previous condition	You have created an instance of *Vector* to which 10 elements have been inserted. An Instance of *listIteratorAdapter* is created for this *Vector.*
Next condition	The *ListIteratorAdapter* Instance remains , and the *Vector* still has the same values, except for the modified one.
Test run record	True

void add(Object e);

testAdd1	
Test Summary	Verifies that the insertion of the item was successful.
Previous condition	You have created an instance of *Vector* to which 10 elements have been inserted. An Instance of *listIteratorAdapter* is created for this *Vector.*
Next condition	The *ListIteratorAdapter* Instance remains , and the *Vector* still has the same values plus the one added.
Test run record	True

testAdd2	
Test Summary	It verifies that the item has been placed at the right index.
Previous condition	You have created an instance of *Vector* to which 10 elements have been inserted. An Instance of *listIteratorAdapter* is created for this *Vector.*
Next condition	The *ListIteratorAdapter* Instance remains , and the *Vector* still has the same values plus the one added.
Test run record	True

testAdd3	
Test Summary	Verifies that if *remove* is called immediately after an *add,* the *IllegalStateException* is thrown.
Previous condition	You have created an instance of *Vector* to which 10 elements have been inserted. An Instance of *listIteratorAdapter* is created for this *Vector.*
Next condition	The exception was launched.
Test run record	True

testAdd4	
Test Summary	Verifies that if *a set* is called immediately after an *add,* the *IllegalStateException* is thrown.
Previous condition	You have created an instance of *Vector* to which 10 elements have been inserted. An Instance of *listIteratorAdapter* is created for this *Vector.*
Next condition	The exception was launched.
Test run record	True

MYENTRY TESTSUITE

Object getKey();

testGetKey	
Test Summary	Verifies that the method returns the correct key.
Previous condition	An instance of *MyEntry* with *key* 2 and *value* "hello" has been created.
Next condition	The *MyEntry* Instance remains unmodified.
Test run record	True

Object getValue();

testGetValue	
Test Summary	Verifies that the method returns the correct value.
Previous condition	An instance of *MyEntry* with *key* 2 and *value* "hello" has been created.
Next condition	The *MyEntry* Instance remains unmodified.
Test run record	True

Object setValue(Object value);

testSetValue	
Test Summary	Verifies that the method changes the value correctly.
Previous condition	An instance of *MyEntry* with *key* 2 and *value* "hello" has been created.
Next condition	The *MyEntry* Instance remains with the new value entered.
Test run record	True

boolean equals(Object o);

testEquals1	
Test Summary	Verifies that an Instance of *MyEntry* is *equals* to itself.
Previous condition	An instance of *MyEntry* with *key* 2 and *value* "hello" has been created.
Next condition	The *MyEntry* Instance remains unmodified.
Test run record	True

testEquals2	
Test Summary	Verifies that a *CollectionAdapter* Instance is not *equals* to a *MyEntry Instance*.
Previous condition	You have created an instance of *MyEntry* with *key* 2 and *value* "hello", an empty *CollectionAdapter* instance is created.
Next condition	The two *instances* remain unchanged.
Test run record	True

testEquals3	
Test Summary	Verifies that two *instances* of *MyEntry* with *key* diverse are not *equals*.
Previous condition	An instance of *MyEntry* with *key* 2 and *value* "hello" has been created, an instance of *MyEntry* is created with *key* 2 and *value* "foo".
Next condition	The two *instances* remain unchanged.
Test run record	True

testEquals4	
Test Summary	Verifies that two *instances* of *MyEntry* with different values are not *equals*.
Previous condition	An instance of *MyEntry* with *key* 2 and *value* "hello" is created, an instance of *MyEntry* with *key 3* and *value* "hello" is created.

Next condition	The two *instances* remain unchanged.
Test run record	True

testEquals5	
Test Summary	Verifies that a *MyEntry* Instance is not *equals* to a *Null* Instance. Then the method returns false.
Previous condition	An instance of *MyEntry* with *key* 2 and *value* "hello" has been created.
Next condition	The *MyEntry* Instance remains unmodified.
Test run record	True

int hashCode();

testHashCode	
Test Summary	Verifies that the method returns the correct *HashCode* value .
Previous condition	An instance of *MyEntry* with *key* 2 and *value* "hello" has been created.
Next condition	The *MyEntry* Instance remains unmodified.
Test run record	True