Test Suite – Test Cases di ListAdapter

Sommario

Test Suite – ListAdapter

La seguente test suite è stata realizzata per testare tutti i metodi presenti nella classe ListAdapter.java per dimostrarne la correttezza. Per quasi la totalità dei metodi, esclusi quelli che non presentano possibili eccezioni, sono stati realizzati due metodi separati, per poter così dividere i test riguardanti il funzionamento da quelli riguardanti il non funzionamento dei vari metodi, quindi il lancio delle eccezioni.

La test suite in sé è suddivisa in due differenti sezioni che dimostrano la correttezza di classi sempre più interne alla classe ListAdapter, rispettivamente ListAdapter e ListAdapter.SubList.

Per la totalità dei metodi presenti in questa test suite è presente una struttura dati (oggetto di tipo ListAdapter) inizialmente vuota. Su tale struttura verranno eseguiti tutti i test, supportati da alcune helper function (fillList(): riempie la lista, printList(): stampa a schermo la lista, printSubList(): stampa a schermo la SubList), funzioni private che forniscono il riempimento e stampa dell'oggetto stesso. A seguito dell'esecuzioni di ogni test, l'oggetto ListAdapter viene svuotato, permettendo così ad ogni test di essere indipendente dagli altri.

Test Cases - ListAdapter

1. initializationTest()

Sommario: questo test è finalizzato a verificare le condizioni che dovranno essere veritiere all'inizio di ogni metodo di test. In particolare si verifica la corretta creazione dell'oggetto ListAdapter e il corretto funzionamento delle helper function utilizzate in tutti i test.

Descrizione test: Controllo che l'oggetto ListAdapter sia stato creato, riempimento di esso tramite il metodo fillList(), controllo del corretto riempimento dell'oggetto. Tale test prevede l'avvenuta corretta creazione dell'oggetto ListAdapter per poter funzionare.

Pre-condizioni: l'oggetto ListAdapter deve essere stato correttamente creato con dimensione uguale a zero.

Post-condizioni: l'oggetto ListAdapter deve essere svuotato dei suoi elementi e la dimensione deve tornare a zero.

Valori attesi: L'oggetto ListAdapter deve essere creato correttamente e la sua dimensione deve essere pari a zero. A seguito dell'inserimento la dimensione dell'oggetto deve diventare pari al totale di elementi inseriti.

2. addTest()

Sommario: questo test è finalizzato a verificare la correttezza del metodo add(int index, Object element). Viene verificato la corretta aggiunta di un elemento ad un desiderato indice della lista quando essa contiene già elementi e non.

Design del caso di test: la verifica del metodo viene eseguita in due successivi test, rispettivamente il numero 2. ed il numero 3.

Descrizione test: aggiunta di un elemento a lista vuota, controllo corretto aumento della dimensione della lista e dell'inserimento nella posizione corretta dell'oggetto precedentemente aggiunto. Riempimento lista, aggiunta di due elementi, controllo corretto aumento della dimensione della lista e dell'inserimento nella posizione corretta degli oggetti precedentemente aggiunti. Tale test prevede l'avvenuta corretta creazione dell'oggetto ListAdapter per poter funzionare.

Pre-condizioni: l'oggetto ListAdapter deve essere stato correttamente creato con dimensione uguale a zero.

Post-condizioni: l'oggetto ListAdapter deve essere svuotato dei suoi elementi e la dimensione deve tornare a zero.

Valori attesi: Il metodo deve essere capace di inserire gli oggetti nella posizione corretta indicata, facendo scalare tutti i risultati presenti, compreso quello nella posizione indicata, alla sua destra. Deve correttamente aumentare di 1 la dimensione della lista.

3. addTestExe()

Sommario: questo test è finalizzato a verificare la correttezza del metodo add(int index, Object element). Viene verificato il corretto funzionamento delle eccezioni.

Design del caso di test: la verifica del metodo viene eseguita in due successivi test, rispettivamente il numero 2. ed il numero 3.

Descrizione test: Riempimento lista, aggiunta di due elementi in posizioni non valide, rispettivamente -1 e size()+1, controllo corretto lancio eccezioni. Aggiunta di un elemento nullo, controllo corretto lancio eccezioni. Tale test prevede l'avvenuta corretta creazione dell'oggetto ListAdapter per poter funzionare.

Pre-condizioni: l'oggetto ListAdapter deve essere stato correttamente creato con dimensione uguale a zero.

Post-condizioni: l'oggetto ListAdapter deve essere svuotato dei suoi elementi e la dimensione deve tornare a zero.

Valori attesi: Il metodo deve essere capace di lanciare IndexOutOfBoundException in caso di utilizzo di indici non validi durante l'inserimento, quindi {]-inf, 0[&]size, +inf[}, lanciare NullPointerException in caso di inserimento di parametri non validi { null }.

4. basicAddTest()

Sommario: questo test è finalizzato a verificare la correttezza del metodo add(Object element). Viene verificato la corretta aggiunta di un elemento desiderato nella lista in ultima posizione quando essa contiene già elementi e non.

Design del caso di test: la verifica del metodo viene eseguita in due successivi test, rispettivamente il numero 4. ed il numero 5.

Descrizione test: aggiunta di un elemento a lista vuota, controllo corretto aumento della dimensione della lista e dell'inserimento nella posizione corretta dell'oggetto precedentemente aggiunto. Riempimento lista, aggiunta di un elemento, controllo corretto aumento della dimensione della lista e dell'inserimento nella posizione corretta degli oggetti precedentemente aggiunti. Tale test prevede l'avvenuta corretta creazione dell'oggetto ListAdapter per poter funzionare.

Pre-condizioni: l'oggetto ListAdapter deve essere stato correttamente creato con dimensione uguale a zero.

Post-condizioni: l'oggetto ListAdapter deve essere svuotato dei suoi elementi e la dimensione deve tornare a zero.

Valori attesi: Il metodo deve essere capace di inserire gli oggetti in ultima posizione. Deve correttamente aumentare di 1 la dimensione della lista.

5. basicAddTestExc()

Sommario: questo test è finalizzato a verificare la correttezza del metodo add(Object element). Viene verificato il corretto funzionamento delle eccezioni nel caso di inserimento di elementi nulli.

Design del caso di test: la verifica del metodo viene eseguita in due successivi test, rispettivamente il numero 4. ed il numero 5.

Descrizione test: Riempimento lista, aggiunta di un elemento nullo { null } e controllo del corretto lancio di NullPointerException. Tale test prevede l'avvenuta corretta creazione dell'oggetto ListAdapter per poter funzionare.

Pre-condizioni: l'oggetto ListAdapter deve essere stato correttamente creato con dimensione uguale a zero.

Post-condizioni: l'oggetto ListAdapter deve essere svuotato dei suoi elementi e la dimensione deve tornare a zero.

Valori attesi: Il metodo deve essere capace di lanciare eccezioni nel caso vengano inseriti valori non validi { null }.

6. addAllTest()

Sommario: questo test è finalizzato a verificare la correttezza del metodo addAll(HCollection c). Viene verificato la corretta aggiunta di una Collection di elementi alla fine della lista quando essa contiene già elementi e non.

Design del caso di test: la verifica del metodo viene eseguita in due successivi test, rispettivamente il numero 6. ed il numero 7.

Descrizione test: creazione di una Collection di due elementi, aggiunta della Collection alla lista vuota, controllo corretto aumento della dimensione della lista e successivo svuotamento di essa. Riempimento lista, aggiunta di una Collection, controllo corretto aumento della dimensione della lista. Tale test prevede l'avvenuta corretta creazione dell'oggetto ListAdapter per poter funzionare e la corretta creazione della Collection.

Pre-condizioni: l'oggetto ListAdapter deve essere stato correttamente creato con dimensione uguale a zero, la Collection deve essere correttamente creata con due valori.

Post-condizioni: l'oggetto ListAdapter deve essere svuotato dei suoi elementi e la dimensione deve tornare a zero.

Valori attesi: Il metodo deve essere capace di inserire tutti gli oggetti di una collezione in ultima posizione. Deve correttamente aumentare di due elementi, ovvero la dimensione della Collection creata.

7. addAllTestExc()

Sommario: questo test è finalizzato a verificare la correttezza del metodo addAll(HCollection c). Viene verificato il corretto funzionamento delle eccezioni nel caso di inserimento di una Collection nulla o contenente elementi nulli.

Design del caso di test: la verifica del metodo viene eseguita in due successivi test, rispettivamente il numero 6. ed il numero 7.

Descrizione test: creazione Collection nulla, controllo che l'aggiunta di tale Collection alla lista restituisca NullPointerException, creazione di una Collection con un solo elemento nullo, controllo che l'aggiunta di tale collezione alla lista restituisca NullPointerException. Tale test prevede l'avvenuta corretta creazione dell'oggetto ListAdapter per poter funzionare.

Pre-condizioni: l'oggetto ListAdapter deve essere stato correttamente creato con dimensione uguale a zero.

Post-condizioni: l'oggetto ListAdapter deve essere svuotato dei suoi elementi e la dimensione deve tornare a zero.

Valori attesi: Il metodo deve essere capace di lanciare NullPointerException nel caso che la Collection inserita sia nulla o contenga valori nulli { null }.

8. addAllIndexTest()

Sommario: questo test è finalizzato a verificare la correttezza del metodo addAllIndex(int index, HCollection c). Viene verificato la corretta aggiunta di una Collection di elementi ad un desiderato indice della lista con l'eventuale spostamento degli elementi presenti in quell'indice e superiori per lasciar spazio agli elementi della Collection quando essa contiene già elementi e non.

Design del caso di test: la verifica del metodo viene eseguita in due successivi test, rispettivamente il numero 8. ed il numero 9.

Descrizione test: creazione di una Collection di due elementi, aggiunta della Collection alla lista vuota, controllo corretto aumento della dimensione della lista e successivo svuotamento di essa. Riempimento lista, aggiunta di una Collection, controllo corretto aumento della dimensione della lista. Tale test prevede l'avvenuta corretta creazione dell'oggetto ListAdapter per poter funzionare e la corretta creazione della Collection.

Pre-condizioni: l'oggetto ListAdapter deve essere stato correttamente creato con dimensione uguale a zero, la Collection deve essere correttamente creata.

Post-condizioni: l'oggetto ListAdapter deve essere svuotato dei suoi elementi e la dimensione deve tornare a zero.

Valori attesi: Il metodo deve essere capace di inserire tutti gli oggetti di una nella posizione desiderata, gestendo di conseguenza la posizione degli altri elementi già presenti nella lista. La dimensione della lista deve correttamente aumentare di due elementi, ovvero la dimensione della Collection creata ed inserita come spiegato nella descrizione.

9. addAllIndexTestExc()

Sommario: questo test è finalizzato a verificare la correttezza del metodo addAllIndex(int index, HCollection c). Viene verificato il corretto funzionamento delle eccezioni.

Design del caso di test: la verifica del metodo viene eseguita in due successivi test, rispettivamente il numero 8. ed il numero 9.

Descrizione test: creazione Collection nulla, riempimento lista, controllo che l'aggiunta di tale Collection alla lista restituisca NullPointerException, creazione di una Collection con un solo elemento nullo, controllo che l'aggiunta di tale collezione alla lista restituisca NullPointerException. Creazione Collection non nulla, controllo che l'inserimento di tale Collection in un indice non valido {]-inf, 0[&]size, +inf[} lanci IndexOutOfBoundException.

Pre-condizioni: l'oggetto ListAdapter deve essere stato correttamente creato con dimensione uguale a zero, la Collection deve essere correttamente creata.

Post-condizioni: l'oggetto ListAdapter deve essere svuotato dei suoi elementi e la dimensione deve tornare a zero.

Valori attesi: Il metodo deve essere capace di lanciare IndexOutOfBoundException in caso di utilizzo di indici non validi durante l'inserimento, quindi {]-inf, 0[&]size, +inf[}, lanciare NullPointerException in caso di inserimento di una Collection nulla o che presenti oggetti nulli al suo interno { null }.

10. clearTest()

Sommario: questo test è finalizzato a verificare la correttezza del metodo clear(). Viene verificato la corretta rimozione di tutti gli elementi presenti nella lista.

Design del caso di test: la verifica del metodo viene eseguita interamente all'interno di questo test.

Descrizione test: riempimento lista, rimozione degli elementi tramite il metodo clear(), controllo che la dimensione della lista sia pari a zero.

Pre-condizioni: l'oggetto ListAdapter deve essere stato correttamente creato con dimensione uguale a zero.

Post-condizioni: l'oggetto ListAdapter deve essere svuotato dei suoi elementi e la dimensione deve tornare a zero.

Valori attesi: dopo l'utilizzo del metodo clear() la lista deve essere vuota, quindi di dimensione zero.

11. containsTest()

Sommario: questo test è finalizzato a verificare la correttezza del metodo contains(Object o). Viene verificata con questo metodo l'esistenza di uno specifico oggetto o all'interno della lista.

Design del caso di test: la verifica del metodo viene eseguita in due successivi test, rispettivamente il numero 11. ed il numero 12.

Descrizione test: riempimento lista, controllo dell'esistenza o meno di due particolari valori all'interno della lista, uno dei quali è effettivamente presente, l'altro no. Tale test prevede l'avvenuta corretta creazione dell'oggetto ListAdapter per poter funzionare.

Pre-condizioni: l'oggetto ListAdapter deve essere stato correttamente creato con dimensione uguale a zero.

Post-condizioni: l'oggetto ListAdapter deve essere svuotato dei suoi elementi e la dimensione deve tornare a zero.

Valori attesi: Il metodo deve restituire { true } nel caso l'elemento cercato fosse presente nella lista, ovvero in una delle due verifiche effettuate, { false } nel caso contrario.

12. containsTestExc()

Sommario: questo test è finalizzato a verificare la correttezza del metodo contains(Object o). Viene verificato che lanci effettivamente eccezioni nei casi non validi.

Design del caso di test: la verifica del metodo viene eseguita in due successivi test, rispettivamente il numero 11. ed il numero 12.

Descrizione test: viene cercato all'interno della lasta il valore nullo { null }, tale inserimento deve causare il lancio dell'eccezione NullPointerException. Tale test prevede l'avvenuta corretta creazione dell'oggetto ListAdapter per poter funzionare.

Pre-condizioni: l'oggetto ListAdapter deve essere stato correttamente creato con dimensione uguale a zero.

Post-condizioni: l'oggetto ListAdapter deve essere svuotato dei suoi elementi e la dimensione deve tornare a zero.

Valori attesi: La ricerca all'interno della lista del valore nullo { null } deve causare il lancio dell'eccezione NullPointerException.

13. containsAllTest()

Sommario: questo test è finalizzato a verificare la correttezza del metodo containsAll(HCollection c). Viene verificato la corretta ricerca di ogni elemento di una Collection di elementi all'interno della lista.

Design del caso di test: la verifica del metodo viene eseguita in due successivi test, rispettivamente il numero 13. ed il numero 14.

Descrizione test: riempimento lista, creazione di una Collection di due elementi esistenti all'interno della lista, ricerca della Collection nella lista, aggiunta di un elemento non presente nella lista all'interno della Collection, ricerca della Collection nella lista.

Pre-condizioni: l'oggetto ListAdapter deve essere stato correttamente creato con dimensione uguale a zero, la Collection deve essere correttamente creata con due valori.

Post-condizioni: l'oggetto ListAdapter deve essere svuotato dei suoi elementi e la dimensione deve tornare a zero.

Valori attesi: Il metodo deve essere capace di verificare l'esistenza di tutti gli elementi di una Collection all'interno di una lista. Uno dei due tentativi porterà il metodo a restituire { true }, l'altro porterà il metodo a restituire { false }.

14. containsAllTestExc()

Sommario: questo test è finalizzato a verificare la correttezza del metodo containsAll(HCollection c). Viene verificato il corretto funzionamento delle eccezioni nel caso di ricerca di una Collection nulla o contenente elementi nulli.

Design del caso di test: la verifica del metodo viene eseguita in due successivi test, rispettivamente il numero 13. ed il numero 14.

Descrizione test: creazione Collection nulla, riempimento lista, controllo che la ricerca di tale Collection nella lista restituisca NullPointerException, creazione di una Collection con un solo elemento nullo, controllo che la ricerca di tale collezione nella lista restituisca NullPointerException. Tale test prevede l'avvenuta corretta creazione dell'oggetto ListAdapter per poter funzionare.

Pre-condizioni: l'oggetto ListAdapter deve essere stato correttamente creato con dimensione uguale a zero e che la Collection venga correttamente creata.

Post-condizioni: l'oggetto ListAdapter deve essere svuotato dei suoi elementi e la dimensione deve tornare a zero.

Valori attesi: Il metodo deve essere capace di lanciare NullPointerException nel caso che la Collection ricercata sia nulla o contenga valori nulli { null }.

15. equalsTest()

Sommario: questo test è finalizzato a verificare la correttezza del metodo equals(Object o). Viene verificato la corrispondenza di due ListAdapter tra loro.

Design del caso di test: la verifica del metodo viene eseguita interamente all'interno di questo test.

Descrizione test: riempimento lista, creazione nuova lista identica alla precedente, verifica che le due liste siano uguali, aggiunta ad una delle due liste di un elemento, verifica che le due liste non siano più uguali. Controllo uguaglianza con null, controllo uguaglianza con una stringa. Tale test prevede l'avvenuta corretta creazione di due oggetti ListAdapter per poter funzionare.

Pre-condizioni: l'oggetto ListAdapter deve essere stato correttamente creato con dimensione uguale a zero e un secondo oggetto ListAdapter uguale al primo deve essere creato.

Post-condizioni: l'oggetto ListAdapter deve essere svuotato dei suoi elementi e la dimensione deve tornare a zero.

Valori attesi: per la prima verifica sono state realizzate due liste contenenti gli stessi valori tra loro, quindi il risultato della funzione equals() dovrà essere { true }, dopo l'aggiunta di un nuovo valore all'interno di una lista, esse non saranno più uguali, facendo sì che il risultato della funzione equals() sia { false }.

16. getTest()

Sommario: questo test è finalizzato a verificare la correttezza del metodo get(int index). Viene verificato che passando al metodo un certo indice venga restituito il valore corretto corrispondente.

Design del caso di test: la verifica del metodo viene eseguita in due successivi test, rispettivamente il numero 16. ed il numero 17.

Descrizione test: riempimento lista, utilizzo del metodo get() e controllo dalla corretta restituzione dell'elemento corrispondente. Tale test prevede l'avvenuta corretta creazione di un ListAdapter per poter funzionare.

Pre-condizioni: l'oggetto ListAdapter deve essere stato correttamente creato con dimensione uguale a zero.

Post-condizioni: l'oggetto ListAdapter deve essere svuotato dei suoi elementi e la dimensione deve tornare a zero.

Valori attesi: ci si aspetta che la lista mantenga una corrispondenza tra la posizione degli elementi e l'ordine con cui essi sono stati inseriti all'interno della lista. Ad esempio al primo oggetto inserito spetterà la posizione nell'indice 0 della lista.

17. getTestExc()

Sommario: questo test è finalizzato a verificare la correttezza del metodo getTest(int index). Viene verificato che lanci effettivamente eccezioni nei casi non validi.

Design del caso di test: la verifica del metodo viene eseguita in due successivi test, rispettivamente il numero 16. ed il numero 17.

Descrizione test: tentativo di utilizzo del metodo get() in due posizioni non valide, rispettivamente -1 e 100, avendo come insieme di indici validi [0, 9], e successivo controllo corretto lancio eccezioni. Tale test prevede l'avvenuta corretta creazione dell'oggetto ListAdapter per poter funzionare.

Pre-condizioni: l'oggetto ListAdapter deve essere stato correttamente creato con dimensione uguale a zero.

Post-condizioni: l'oggetto ListAdapter deve essere svuotato dei suoi elementi e la dimensione deve tornare a zero.

Valori attesi: Il metodo deve essere capace di lanciare IndexOutOfBoundException in caso di utilizzo di indici non validi durante l'inserimento, quindi {]-inf, 0[&]size, +inf[}.

18. hashCodeTest()

Sommario: questo test è finalizzato a verificare la correttezza del metodo hashCode(). Viene verificato la corrispondenza tra gli hashCode di due ListAdapter tra loro, ovvero che il metodo restituisca lo stesso valore nel caso che venga chiamato in liste uguali, come richiesto dalla documentazione.

Design del caso di test: la verifica del metodo viene eseguita interamente all'interno di questo test.

Descrizione test: riempimento lista, creazione nuova lista identica alla precedente, verifica che gli hashCode delle due liste siano uguali. Tale test prevede l'avvenuta corretta creazione di due oggetti ListAdapter per poter funzionare.

Pre-condizioni: l'oggetto ListAdapter deve essere stato correttamente creato con dimensione uguale a zero e un secondo oggetto ListAdapter uguale al primo deve essere creato.

Post-condizioni: l'oggetto ListAdapter deve essere svuotato dei suoi elementi e la dimensione deve tornare a zero.

Valori attesi: come il metodo hashCode stabilisce, la chiamata di esso su due liste uguali deve poter fornire lo stesso valore. Dal test quindi ci si attende di ottenere una corrispondenza tra i due risultati.

19. indexOfTest()

Sommario: questo test è finalizzato a verificare la correttezza del metodo indexOf(Object o). Viene verificato che passando al metodo un certo oggetto venga restituito il primo indice della sua occorrenza all'interno della lista o -1 ne caso esso non sia presente.

Design del caso di test: la verifica del metodo viene eseguita in due successivi test, rispettivamente il numero 19. ed il numero 20.

Descrizione test: riempimento lista, utilizzo del metodo indexOf() e controllo dalla corretta restituzione dell'indice corrispondente. Vengono cercati due oggetti, uno presente effettivamente all'interno della lista e uno non presente, in modo tale da verificare anche la corretta restituzione del valore -1. Tale test prevede l'avvenuta corretta creazione di due oggetti ListAdapter per poter funzionare.

Pre-condizioni: l'oggetto ListAdapter deve essere stato correttamente creato con dimensione uguale a zero.

Post-condizioni: l'oggetto ListAdapter deve essere svuotato dei suoi elementi e la dimensione deve tornare a zero.

Valori attesi: ci si aspetta che il metodo possa riconoscere la presenza degli oggetti cercati, restituendo l'indice della prima occorrenza o agendo di conseguenza nel caso l'oggetto non sia presente.

20. indexOfTestExc()

Sommario: questo test è finalizzato a verificare la correttezza del metodo getTest(int index). Viene verificato che lanci effettivamente eccezioni nei casi non validi.

Design del caso di test: la verifica del metodo viene eseguita in due successivi test, rispettivamente il numero 19. ed il numero 20.

Descrizione test: riempimento lista, tentativo di utilizzo del metodo indexOf() passando un oggetto nullo, viene successivamente atteso il lancio dell'eccezione NullPointerException. Tale test prevede l'avvenuta corretta creazione dell'oggetto ListAdapter per poter funzionare.

Pre-condizioni: l'oggetto ListAdapter deve essere stato correttamente creato con dimensione uguale a zero.

Post-condizioni: l'oggetto ListAdapter deve essere svuotato dei suoi elementi e la dimensione deve tornare a zero.

Valori attesi: Il metodo deve essere capace di lanciare NullPointerException in caso di utilizzo di Oggetti non validi { null } durante l'inserimento.

21. isEmptyTest()

Sommario: questo test è finalizzato a verificare la correttezza del metodo isEmpty(). Viene verificato la restituzione del corretto valore booleano nel caso la lista abbia elementi e non.

Design del caso di test: la verifica del metodo viene eseguita interamente all'interno di questo test.

Descrizione test: riempimento lista, controllo tramite isEmpty() che la lista non sia vuota, rimozione degli elementi tramite il metodo clear(), controllo tramite isEmpty() che la lista sia vuota. Tale test prevede l'avvenuta corretta creazione dell'oggetto ListAdapter per poter funzionare.

Pre-condizioni: l'oggetto ListAdapter deve essere stato correttamente creato con dimensione uguale a zero.

Post-condizioni: l'oggetto ListAdapter deve essere svuotato dei suoi elementi e la dimensione deve tornare a zero.

Valori attesi: dopo l'utilizzo del metodo isEmpty() la lista deve essere restituire { true } nel caso la lista fosse vuota, { false } nel caso non lo fosse.

22. iteratorTest()

Sommario: questo test è finalizzato a verificare la correttezza del metodo iterator(). Viene verificata la corretta restituzione di un iteratore, i metodi ad esso associati e il corretto funzionamento delle eccezione.

Design del caso di test: la verifica del metodo viene ripetuta con successivi controlli anche successivamente, si tratta dei test numero 22. e numero 57.

Descrizione test: riempimento lista, creazione di un iteratore, controllo del corretto funzionamento del metodo next(), sia che restituisca il valore corretto, sia che non restituisca un valore errato, controllo del corretto funzionamento del metodo hasNext(), utilizzato per spostarsi all'ultimo elemento e controllando che quello sia effettivamente l'ultimo elemento chiamando next() e controllando che lanci NullPointerException. Tale test prevede l'avvenuta corretta creazione dell'oggetto ListAdapter per poter funzionare.

Pre-condizioni: l'oggetto ListAdapter deve essere stato correttamente creato con dimensione uguale a zero.

Post-condizioni: l'oggetto ListAdapter deve essere svuotato dei suoi elementi e la dimensione deve tornare a zero.

Valori attesi: ci si aspetta che il metodo restituisca un iteratore, e che l'iteratore restituito funzioni correttamente.

23. lastIndexOfTest()

Sommario: questo test è finalizzato a verificare la correttezza del metodo lastIndexOf(Object o). Viene verificato che passando al metodo un certo oggetto venga restituito l'ultimo indice della sua occorrenza all'interno della lista o -1 ne caso esso non sia presente.

Design del caso di test: la verifica del metodo viene eseguita in due successivi test, rispettivamente il numero 23. ed il numero 24.

Descrizione test: riempimento lista, inserimento di altri due elementi alla sua coda, utilizzo del metodo lastIndexOf() e controllo dalla corretta restituzione dell'indice corrispondente. Vengono cercati due oggetti, uno presente effettivamente all'interno della lista e uno non presente, in modo tale da verificare anche la corretta restituzione del valore -1. Tale test prevede l'avvenuta corretta creazione dell'oggetto ListAdapter per poter funzionare.

Pre-condizioni: l'oggetto ListAdapter deve essere stato correttamente creato con dimensione uguale a zero.

Post-condizioni: l'oggetto ListAdapter deve essere svuotato dei suoi elementi e la dimensione deve tornare a zero.

Valori attesi: ci si aspetta che il metodo possa riconoscere la presenza degli oggetti cercati, restituendo l'indice dell'ultima occorrenza o agendo di conseguenza nel caso l'oggetto non sia presente.

24. lastIndexOfTestExc()

Sommario: questo test è finalizzato a verificare la correttezza del metodo lastIndexOfTestExc (Object o). Viene verificato che lanci effettivamente eccezioni nei casi non validi.

Design del caso di test: la verifica del metodo viene eseguita in due successivi test, rispettivamente il numero 23. ed il numero 24.

Descrizione test: riempimento lista, tentativo di utilizzo del metodo lastIndexOf() passando un oggetto nullo, viene successivamente atteso il lancio dell'eccezione NullPointerException. Tale test prevede l'avvenuta corretta creazione dell'oggetto ListAdapter per poter funzionare.

Pre-condizioni: l'oggetto ListAdapter deve essere stato correttamente creato con dimensione uguale a zero.

Post-condizioni: l'oggetto ListAdapter deve essere svuotato dei suoi elementi e la dimensione deve tornare a zero.

Valori attesi: Il metodo deve essere capace di lanciare NullPointerException in caso di utilizzo di Oggetti non validi { null } durante l'inserimento.

25. listIteratorTest()

Sommario: questo test è finalizzato a verificare la correttezza del metodo listIterator(). Viene verificata la corretta restituzione di un iteratore, i metodi ad esso associati e il corretto funzionamento delle eccezione.

Design del caso di test: la verifica del metodo viene effettuata completamente in questo test.

Descrizione test: riempimento lista, creazione di un iteratore, controllo del corretto funzionamento del metodo next(), nextIndex(), previous(), previousIndex(), spostandosi avanti e indietro all'interno della lista e controllando le corrispondenze tra posizione attuale, indice restituito e valori attesi. Si test poi il metodo remove() e il metodo add(), controllano la successiva esistenza dell'oggetto aggiunto e che sia stato inserito nella posizione corretta. Successivamente si verifica il corretto funzionamento di hasPrevious(), utilizzato per spostarsi al primo elemento e controllando che quello sia effettivamente il primo elemento chiamando previous() e controllando che lanci NullPointerException. Tale test prevede l'avvenuta corretta creazione dell'oggetto ListAdapter per poter funzionare.

Pre-condizioni: l'oggetto ListAdapter deve essere stato correttamente creato con dimensione uguale a zero.

Post-condizioni: l'oggetto ListAdapter deve essere svuotato dei suoi elementi e la dimensione deve tornare a zero.

Valori attesi: ci si aspetta che il metodo restituisca un iteratore, e che l'iteratore restituito funzioni correttamente.

26. listIteratorIndexTest()

Sommario: questo test è finalizzato a verificare la correttezza del metodo listIteratorIndex(int index). Viene verificata la corretta restituzione di un iteratore, i metodi ad esso associati e il corretto funzionamento delle eccezione.

Design del caso di test: la verifica del metodo viene ripetuta con successivi controlli anche successivamente, si tratta dei test numero 26. e numero .

Descrizione test: riempimento lista, creazione di un iteratore passando un indice al costruttore, controllo che il successivo valore restituito da next() sia quello aspettato, controllo che il passaggio al costruttore di un indice non valido restituisca IndexOutOfBoundException. Tale test prevede l'avvenuta corretta creazione dell'oggetto ListAdapter per poter funzionare.

Pre-condizioni: l'oggetto ListAdapter deve essere stato correttamente creato con dimensione uguale a zero.

Post-condizioni: l'oggetto ListAdapter deve essere svuotato dei suoi elementi e la dimensione deve tornare a zero.

Valori attesi: ci si aspetta che il metodo restituisca un iteratore, e che l'iteratore restituito funzioni correttamente.

27. removeTest()

Sommario: questo test è finalizzato a verificare la correttezza del metodo remove(int index). Viene verificato la corretta rimozione di un elemento ad un desiderato indice della lista, la corretta restituzione di tale elemento e la diminuzione della dimensione della lista.

Design del caso di test: la verifica del metodo viene eseguita in due successivi test, rispettivamente il numero 27. ed il numero 28.

Descrizione test: riempimento lista, rimozione di un elemento tramite remove() e controllo corretta restituzione dell'elemento corrispondente all'indice rimosso, controllo corretta diminuzione della dimensione della lista. Tale test prevede l'avvenuta corretta creazione dell'oggetto ListAdapter per poter funzionare.

Pre-condizioni: l'oggetto ListAdapter deve essere stato correttamente creato con dimensione uguale a zero.

Post-condizioni: l'oggetto ListAdapter deve essere svuotato dei suoi elementi e la dimensione deve tornare a zero.

Valori attesi: Il metodo deve essere capace di rimuovere gli oggetti nella posizione corretta indicata, facendo scalare tutti i risultati presenti alla sua sinistra. Deve correttamente diminuire di 1 la dimensione della lista.

28. removeTestExe()

Sommario: questo test è finalizzato a verificare la correttezza del metodo remove(int index). Viene verificato il corretto funzionamento delle eccezioni.

Design del caso di test: la verifica del metodo viene eseguita in due successivi test, rispettivamente il numero 27. ed il numero 28.

Descrizione test: rimozione di un elemento in posizione size() avendo la lista vuota e controllo corretto lancio IndexOutOfBoundException, riempimento lista, rimozione di un elemento in posizione size(), controllo corretto lancio IndexOutOfBoundException. Tale test prevede l'avvenuta corretta creazione dell'oggetto ListAdapter per poter funzionare.

Pre-condizioni: l'oggetto ListAdapter deve essere stato correttamente creato con dimensione uguale a zero.

Post-condizioni: l'oggetto ListAdapter deve essere svuotato dei suoi elementi e la dimensione deve tornare a zero.

Valori attesi: Il metodo deve essere capace di lanciare IndexOutOfBoundException in caso di utilizzo di indici non validi durante la rimozione, quindi {]-inf, 0[&]size, +inf[}.

29. removeObjTest()

Sommario: questo test è finalizzato a verificare la correttezza del metodo remove(Object o). Viene verificato la corretta rimozione di un booleano che descriva la rimozione o meno di un elemento.

Design del caso di test: la verifica del metodo viene eseguita in due successivi test, rispettivamente il numero 29. ed il numero 30.

Descrizione test: riempimento lista, rimozione di un elemento presente nella lista tramite remove(Object o) e controllo corretta restituzione di { true } e della corretta diminuzione della dimensione della lista. Tentativo di rimozione di un oggetto non presente nella lista, controllo che il metodo restituisca { false } e che la dimensione della lista non sia cambiata. Tale test prevede l'avvenuta corretta creazione dell'oggetto ListAdapter per poter funzionare.

Pre-condizioni: l'oggetto ListAdapter deve essere stato correttamente creato con dimensione uguale a zero.

Post-condizioni: l'oggetto ListAdapter deve essere svuotato dei suoi elementi e la dimensione deve tornare a zero.

Valori attesi: Il metodo deve essere capace di rimuovere gli oggetti indicati, facendo scalare tutti i risultati presenti alla sua sinistra. Deve correttamente diminuire di 1 la dimensione della lista.

30. removeObjTestExe()

Sommario: questo test è finalizzato a verificare la correttezza del metodo remove(Object o). Viene verificato il corretto funzionamento delle eccezioni.

Design del caso di test: la verifica del metodo viene eseguita in due successivi test, rispettivamente il numero 29. ed il numero 30.

Descrizione test: rimozione di un elemento nullo { null } avendo la lista vuota e controllo corretto lancio NullPointerException, riempimento lista, rimozione di un elemento nullo { null }, controllo corretto lancio

NullPointerException. Tale test prevede l'avvenuta corretta creazione dell'oggetto ListAdapter per poter funzionare.

Pre-condizioni: l'oggetto ListAdapter deve essere stato correttamente creato con dimensione uguale a zero.

Post-condizioni: l'oggetto ListAdapter deve essere svuotato dei suoi elementi e la dimensione deve tornare a zero.

Valori attesi: Il metodo deve essere capace di lanciare NullPointerException in caso di utilizzo di oggetti non validi durante la rimozione, quindi { null }.

31. removeAllTest()

Sommario: questo test è finalizzato a verificare la correttezza del metodo removeAll(HCollection c). Viene verificato la corretta rimozione di una Collection di elementi dalla lista.

Design del caso di test: la verifica del metodo viene eseguita in due successivi test, rispettivamente il numero 31. ed il numero 32.

Descrizione test: creazione di una Collection di due elementi entrambi presenti nella lista, riempimento lista, rimozione della Collection, controllo corretta diminuzione della dimensione della lista. Aggiunta di un elemento non presente nella lista alla Collection, pulizia lista e nuovo riempimento di essa, rimozione della Collection dalla lista e controllo corretta diminuzione della dimensione della lista. Tale test prevede l'avvenuta corretta creazione dell'oggetto ListAdapter per poter funzionare e la corretta creazione della Collection.

Pre-condizioni: l'oggetto ListAdapter deve essere stato correttamente creato con dimensione uguale a zero, la Collection deve essere correttamente creata con due valori.

Post-condizioni: l'oggetto ListAdapter deve essere svuotato dei suoi elementi e la dimensione deve tornare a zero.

Valori attesi: Il metodo deve essere capace di rimuovere tutti gli oggetti di una Collection. Deve correttamente ridurre la dimensione della lista di due elementi, ovvero la dimensione degli elementi della Collection creata che fanno parte anche della lista.

32. removeAllTestExc()

Sommario: questo test è finalizzato a verificare la correttezza del metodo removeAll(HCollection c). Viene verificato il corretto funzionamento delle eccezioni nel caso di rimozione di una Collection nulla.

Design del caso di test: la verifica del metodo viene eseguita in due successivi test, rispettivamente il numero 31. ed il numero 32.

Descrizione test: creazione Collection nulla, controllo che la rimozione di tale Collection dalla lista restituisca NullPointerException. Tale test prevede l'avvenuta corretta creazione dell'oggetto ListAdapter per poter funzionare.

Pre-condizioni: l'oggetto ListAdapter deve essere stato correttamente creato con dimensione uguale a zero.

Post-condizioni: l'oggetto ListAdapter deve essere svuotato dei suoi elementi e la dimensione deve tornare a zero.

Valori attesi: Il metodo deve essere capace di lanciare NullPointerException nel caso che la Collection inserita sia nulla o contenga valori nulli { null }.

33. retainAllTest()

Sommario: questo test è finalizzato a verificare la correttezza del metodo retainAll(HCollection c). Viene verificato la corretta rimozione di tutti gli elementi non presenti in una Collection di elementi dalla lista.

Design del caso di test: la verifica del metodo viene eseguita in due successivi test, rispettivamente il numero 33. ed il numero 34.

Descrizione test: creazione di una Collection di due elementi entrambi presenti nella lista, riempimento lista, utilizzo del metodo retainAll(), controllo corretta diminuzione della dimensione della lista. Aggiunta di un elemento non presente nella lista alla Collection, pulizia lista e nuovo riempimento di essa, utilizzo del metodo retainAll() e controllo corretta diminuzione della dimensione della lista. Tale test prevede l'avvenuta corretta creazione dell'oggetto ListAdapter per poter funzionare e la corretta creazione della Collection.

Pre-condizioni: l'oggetto ListAdapter deve essere stato correttamente creato con dimensione uguale a zero, la Collection deve essere correttamente creata con due valori.

Post-condizioni: l'oggetto ListAdapter deve essere svuotato dei suoi elementi e la dimensione deve tornare a zero.

Valori attesi: Il metodo deve essere capace di rimuovere tutti gli oggetti non presenti nella Collection. Deve correttamente ridurre la dimensione della lista a 2, ovvero la dimensione degli elementi della lista presenti nella Collection creata.

34. retainAllTestExc()

Sommario: questo test è finalizzato a verificare la correttezza del metodo retainAll(HCollection c). Viene verificato il corretto funzionamento delle eccezioni nel caso di utilizzo di una Collection nulla.

Design del caso di test: la verifica del metodo viene eseguita in due successivi test, rispettivamente il numero 33. ed il numero 34.

Descrizione test: creazione Collection nulla, controllo che l'utilizzo di retainAll() con tale Collection dalla lista restituisca NullPointerException. Tale test prevede l'avvenuta corretta creazione dell'oggetto ListAdapter per poter funzionare.

Pre-condizioni: l'oggetto ListAdapter deve essere stato correttamente creato con dimensione uguale a zero.

Post-condizioni: l'oggetto ListAdapter deve essere svuotato dei suoi elementi e la dimensione deve tornare a zero.

Valori attesi: Il metodo deve essere capace di lanciare NullPointerException nel caso che la Collection inserita sia nulla o contenga valori nulli { null }.

35. setTest()

Sommario: questo test è finalizzato a verificare la correttezza del metodo set(int index, Object element). Viene verificato la corretta sostituzione di un elemento ad un desiderato indice della lista.

Design del caso di test: la verifica del metodo viene eseguita in due successivi test, rispettivamente il numero 35. ed il numero 36.

Descrizione test: riempimento lista, sostituzione di un elemento, controllo corretta presenza nella lista dell'elemento appena inserito. Tale test prevede l'avvenuta corretta creazione dell'oggetto ListAdapter per poter funzionare.

Pre-condizioni: l'oggetto ListAdapter deve essere stato correttamente creato con dimensione uguale a zero.

Post-condizioni: l'oggetto ListAdapter deve essere svuotato dei suoi elementi e la dimensione deve tornare a zero.

Valori attesi: Il metodo deve essere capace di sostituire gli oggetti nella posizione corretta indicata. Deve far rimanere invariata la dimensione della lista.

36. setTestExe()

Sommario: questo test è finalizzato a verificare la correttezza del metodo set(int index, Object element). Viene verificato il corretto funzionamento delle eccezioni.

Design del caso di test: la verifica del metodo viene eseguita in due successivi test, rispettivamente il numero 35. ed il numero 36.

Descrizione test: Riempimento lista, aggiunta di due elementi in posizioni non valide, rispettivamente -1 e size(), controllo corretto lancio eccezioni. Aggiunta di un elemento nullo, controllo corretto lancio eccezioni. Tale test prevede l'avvenuta corretta creazione dell'oggetto ListAdapter per poter funzionare.

Pre-condizioni: l'oggetto ListAdapter deve essere stato correttamente creato con dimensione uguale a zero.

Post-condizioni: l'oggetto ListAdapter deve essere svuotato dei suoi elementi e la dimensione deve tornare a zero.

Valori attesi: Il metodo deve essere capace di lanciare IndexOutOfBoundException in caso di utilizzo di indici non validi durante l'inserimento, quindi {]-inf, 0[&]size, +inf[}, lanciare NullPointerException in caso di inserimento di parametri non validi { null }.

37. sizeTest()

Sommario: questo test è finalizzato a verificare la correttezza del metodo size(). Viene verificato la restituzione del corretto valore intero corrispondente al numero di elementi della lista.

Design del caso di test: la verifica del metodo viene eseguita interamente all'interno di questo test.

Descrizione test: controllo se con la lista vuota il metodo size() restituisce 0, riempimento lista, controllo tramite size() che la dimensione della lista corrisponda a 10, rimozione di un elemento tramite il metodo remove(), controllo tramite size() che la lista abbia dimensione 9. Tale test prevede l'avvenuta corretta creazione dell'oggetto ListAdapter per poter funzionare.

Pre-condizioni: l'oggetto ListAdapter deve essere stato correttamente creato con dimensione uguale a zero.

Post-condizioni: l'oggetto ListAdapter deve essere svuotato dei suoi elementi e la dimensione deve tornare a zero.

Valori attesi: il metodo size() deve restituire il corretto numero di elementi in ogni momento.

38. subListTest()

Sommario: questo test è finalizzato a verificare la correttezza del metodo subList(int from, int to). Viene verificata la corretta restituzione di una SubList, i metodi ad esso associati verranno verificati successivamente.

Design del caso di test: la verifica del metodo viene eseguita in due successivi test, rispettivamente il numero 38. ed il numero 39. La verifica dei metodi di SubList viene eseguita successivamente con i test dal numero 43. al numero 71.

Descrizione test: riempimento lista, creazione di una SubList, controllo che la dimensione della lista creata sia corretta. Creazione di un'altra SubList, questa volta con gli indici from e to coincidenti e controllo della dimensione della SubList creata. Creazione di un'altra SubList, questa volta con gli indici coincidenti agli estremi della lista originale, controllo che le dimensioni siano coincidenti. Tale test prevede l'avvenuta corretta creazione dell'oggetto ListAdapter per poter funzionare.

Pre-condizioni: l'oggetto ListAdapter deve essere stato correttamente creato con dimensione uguale a zero.

Post-condizioni: l'oggetto ListAdapter deve essere svuotato dei suoi elementi e la dimensione deve tornare a zero.

Valori attesi: ci si aspetta che il metodo restituisca un SubList funzionante dalla dimensione fornita tramite gli indici from e tu presenti nel costruttore.

39. subListTestExe()

Sommario: questo test è finalizzato a verificare la correttezza del metodo subList(int from, int to). Viene verificato il corretto funzionamento delle eccezioni.

Design del caso di test: la verifica del metodo viene eseguita in due successivi test, rispettivamente il numero 38. ed il numero 39.

Descrizione test: creazione di due SubList con indici in posizioni non valide, la prima (to = 100) e la seconda (from = -2), controllo corretto lancio eccezioni. Tale test prevede l'avvenuta corretta creazione dell'oggetto ListAdapter per poter funzionare.

Pre-condizioni: l'oggetto ListAdapter deve essere stato correttamente creato con dimensione uguale a zero.

Post-condizioni: l'oggetto ListAdapter deve essere svuotato dei suoi elementi e la dimensione deve tornare a zero.

Valori attesi: Il metodo deve essere capace di lanciare IndexOutOfBoundException in caso di utilizzo di indici non validi durante l'inserimento, quindi {]-inf, 0[&]size, +inf[}.

40. toArrayTest()

Sommario: questo test è finalizzato a verificare la correttezza del metodo toArray(). Viene verificata la corretta restituzione di un array contenente gli elementi della lista nello stesso ordine.

Design del caso di test: la verifica del metodo viene eseguita interamente all'interno di questo test.

Descrizione test: riempimento lista, creazione di un array tramite il metodo toArray(), controllo che tutti gli elementi dell'array coincidano in valore e posizione a tutti gli elementi della lista originaria. Tale test prevede l'avvenuta corretta creazione dell'oggetto ListAdapter per poter funzionare.

Pre-condizioni: l'oggetto ListAdapter deve essere stato correttamente creato con dimensione uguale a zero.

Post-condizioni: l'oggetto ListAdapter deve essere svuotato dei suoi elementi e la dimensione deve tornare a zero.

Valori attesi: ci si aspetta che il metodo restituisca un array dalla dimensione, valori e posizioni degli elementi identiche a quelle della lista originale.

41. toArrayObjTest()

Sommario: questo test è finalizzato a verificare la correttezza del metodo toArray(Object[] a). Viene verificata la corretta restituzione di un array contenente gli elementi della lista nello stesso ordine utilizzando un array a come destinazione finale.

Design del caso di test: la verifica del metodo viene eseguita in due successivi test, rispettivamente il numero 41. ed il numero 42.

Descrizione test: riempimento lista, creazione di un array a due elementi e utilizzo del metodo toArray() passando come parametro l'array creato, controllo che tutti gli elementi dell'array coincidano in valore e posizione a tutti gli elementi della lista originaria. Creazione di un array a venti elementi e utilizzo del metodo toArray() passando come parametro l'array creato, controllo che tutti gli elementi dell'array coincidano in valore e posizione a tutti gli elementi della lista originaria. Tale test prevede l'avvenuta corretta creazione dell'oggetto ListAdapter per poter funzionare.

Pre-condizioni: l'oggetto ListAdapter deve essere stato correttamente creato con dimensione uguale a zero.

Post-condizioni: l'oggetto ListAdapter deve essere svuotato dei suoi elementi e la dimensione deve tornare a zero.

Valori attesi: ci si aspetta che il metodo restituisca un array con valori e posizioni degli elementi identiche a quelle della lista originale.

42. toArrayObjTestExe()

Sommario: questo test è finalizzato a verificare la correttezza del metodo toArray(Object[] o). Viene verificato il corretto funzionamento delle eccezioni.

Design del caso di test: la verifica del metodo viene eseguita in due successivi test, rispettivamente il numero 41. ed il numero 42.

Descrizione test: riempimento lista, creazione di un array nullo e passaggio di esso tramite il metodo toArray(), controllo corretto lancio NullPointerException. Tale test prevede l'avvenuta corretta creazione dell'oggetto ListAdapter per poter funzionare.

Pre-condizioni: l'oggetto ListAdapter deve essere stato correttamente creato con dimensione uguale a zero.

Post-condizioni: l'oggetto ListAdapter deve essere svuotato dei suoi elementi e la dimensione deve tornare a zero.

Valori attesi: Il metodo deve essere capace di lanciare lanciare NullPointerException in caso di inserimento di array non validi { null }.

Test Cases – ListAdapter.SubList

43. subListAddTest()

Sommario: questo test è finalizzato a verificare la correttezza del metodo add(Object element). Viene verificato la corretta aggiunta di un elemento desiderato nella SubList in ultima posizione.

Design del caso di test: la verifica del metodo viene eseguita in due successivi test, rispettivamente il numero 43. ed il numero 44.

Descrizione test: riempimento lista, creazione SubList, aggiunta di due elementi, controllo corretto aumento della dimensione della lista e dell'inserimento nella posizione corretta degli oggetti precedentemente aggiunti. Tale test prevede l'avvenuta corretta creazione dell'oggetto ListAdapter e della SubList per poter funzionare.

Pre-condizioni: l'oggetto ListAdapter deve essere stato correttamente creato con dimensione uguale a zero, creazione di una SubList.

Post-condizioni: l'oggetto ListAdapter deve essere svuotato dei suoi elementi e la dimensione deve tornare a zero.

Valori attesi: Il metodo deve essere capace di inserire gli oggetti in ultima posizione. Deve correttamente aumentare di 1 la dimensione della lista.

44. subListAddTestExc()

Sommario: questo test è finalizzato a verificare la correttezza del metodo add(Object element). Viene verificato il corretto funzionamento delle eccezioni nel caso di inserimento di elementi nulli.

Design del caso di test: la verifica del metodo viene eseguita in due successivi test, rispettivamente il numero 43. ed il numero 44.

Descrizione test: riempimento lista, aggiunta di un elemento nullo { null } e controllo del corretto lancio di NullPointerException . Tale test prevede l'avvenuta corretta creazione dell'oggetto ListAdapter e della SubList per poter funzionare.

Pre-condizioni: l'oggetto ListAdapter deve essere stato correttamente creato con dimensione uguale a zero, creazione di una SubList.

Post-condizioni: l'oggetto ListAdapter deve essere svuotato dei suoi elementi e la dimensione deve tornare a zero.

Valori attesi: Il metodo deve essere capace di lanciare eccezioni nel caso vengano inseriti valori non validi { null }.

45. subListAddAllTest()

Sommario: questo test è finalizzato a verificare la correttezza del metodo addAll(HCollection c). Viene verificato la corretta aggiunta di una Collection di elementi alla fine della SubList.

Design del caso di test: la verifica del metodo viene eseguita in due successivi test, rispettivamente il numero 45. ed il numero 46.

Descrizione test: creazione di una Collection di due elementi, riempimento lista, creazione SubList, aggiunta della Collection alla SubList, controllo corretto aumento della dimensione della lista, controllo corretta posizione degli elementi appena giunti alla lista. Tale test prevede l'avvenuta corretta creazione dell'oggetto ListAdapter per poter funzionare e la corretta creazione della Collection.

Pre-condizioni: l'oggetto ListAdapter deve essere stato correttamente creato con dimensione uguale a zero, la Collection deve essere correttamente creata con due valori.

Post-condizioni: l'oggetto ListAdapter deve essere svuotato dei suoi elementi e la dimensione deve tornare a zero, creazione di una SubList.

Valori attesi: Il metodo deve essere capace di inserire tutti gli oggetti di una collezione in ultima posizione della SubList. Deve correttamente aumentare di due elementi, ovvero la dimensione della Collection creata.

46. subListAddAllTestExc()

Sommario: questo test è finalizzato a verificare la correttezza del metodo addAll(HCollection c). Viene verificato il corretto funzionamento delle eccezioni nel caso di inserimento di una Collection nulla o contenente elementi nulli.

Design del caso di test: la verifica del metodo viene eseguita in due successivi test, rispettivamente il numero 45. ed il numero 46.

Descrizione test: creazione Collection nulla, riempimento lista, creazione SubList, controllo che l'aggiunta di tale Collection alla SubList restituisca NullPointerException, creazione di una Collection con un solo elemento nullo, controllo che l'aggiunta di tale collezione alla SubList restituisca NullPointerException. Tale test prevede l'avvenuta corretta creazione dell'oggetto ListAdapter per poter funzionare.

Pre-condizioni: l'oggetto ListAdapter deve essere stato correttamente creato con dimensione uguale a zero, creazione di una SubList.

Post-condizioni: l'oggetto ListAdapter deve essere svuotato dei suoi elementi e la dimensione deve tornare a zero.

Valori attesi: Il metodo deve essere capace di lanciare NullPointerException nel caso che la Collection inserita sia nulla o contenga valori nulli { null }.

47. subListSetTest()

Sommario: questo test è finalizzato a verificare la correttezza del metodo get(int index). Viene verificato che passando al metodo un certo indice venga restituito il valore corretto corrispondente.

Design del caso di test: la verifica del metodo viene eseguita in due successivi test, rispettivamente il numero 47. ed il numero 48.

Descrizione test: riempimento lista, creazione SubList, utilizzo del metodo get() e controllo dalla corretta restituzione dell'elemento corrispondente. Tale test prevede l'avvenuta corretta creazione dell'oggetto ListAdapter per poter funzionare.

Pre-condizioni: l'oggetto ListAdapter deve essere stato correttamente creato con dimensione uguale a zero, creazione di una SubList.

Post-condizioni: l'oggetto ListAdapter deve essere svuotato dei suoi elementi e la dimensione deve tornare a zero.

Valori attesi: ci si aspetta che la SubList mantenga una corrispondenza tra la posizione degli elementi e l'ordine con cui essi sono stati inseriti all'interno della lista. Ad esempio al primo oggetto inserito spetterà la posizione nell'indice 0 della lista.

48. subListSetTestExc()

Sommario: questo test è finalizzato a verificare la correttezza del metodo getTest(int index). Viene verificato che lanci effettivamente eccezioni nei casi non validi.

Design del caso di test: la verifica del metodo viene eseguita in due successivi test, rispettivamente il numero 47. ed il numero 48.

Descrizione test: tentativo di utilizzo del metodo get() in due posizioni non valide, rispettivamente -1 e 3, avendo come insieme di indici validi [2, 4], e successivo controllo corretto lancio eccezioni. Tale test prevede l'avvenuta corretta creazione dell'oggetto ListAdapter per poter funzionare.

Pre-condizioni: l'oggetto ListAdapter deve essere stato correttamente creato con dimensione uguale a zero, creazione di una SubList.

Post-condizioni: l'oggetto ListAdapter deve essere svuotato dei suoi elementi e la dimensione deve tornare a zero.

Valori attesi: Il metodo deve essere capace di lanciare IndexOutOfBoundException in caso di utilizzo di indici non validi durante l'inserimento, quindi {]-inf, 0[&]size, +inf[}.

49. subListClearTest()

Sommario: questo test è finalizzato a verificare la correttezza del metodo clear(). Viene verificato la corretta rimozione di tutti gli elementi presenti nella SubList.

Design del caso di test: la verifica del metodo viene eseguita interamente all'interno di questo test.

Descrizione test: riempimento lista, creazione SubList, rimozione degli elementi tramite il metodo clear(), controllo che la dimensione della SubList sia pari a zero, controllo che la dimensione della lista originale sia 7, ovvero { list.size() – SubList.size() } quando sono state appena create.

Pre-condizioni: l'oggetto ListAdapter deve essere stato correttamente creato con dimensione uguale a zero, creazione di una SubList.

Post-condizioni: l'oggetto ListAdapter deve essere svuotato dei suoi elementi e la dimensione deve tornare a zero.

Valori attesi: dopo l'utilizzo del metodo clear() la SubList deve essere vuota, quindi di dimensione zero, mentre la lista originaria deve essere privata dagli elementi della SubList.

50. subListContainsTest()

Sommario: questo test è finalizzato a verificare la correttezza del metodo contains(Object o). Viene verificata con questo metodo l'esistenza di uno specifico oggetto o all'interno della SubList.

Design del caso di test: la verifica del metodo viene eseguita in due successivi test, rispettivamente il numero 50. ed il numero 51.

Descrizione test: riempimento lista, creazione SubList, controllo dell'esistenza o meno di due particolari valori all'interno della lista, uno dei quali è effettivamente presente, l'altro no. Tale test prevede l'avvenuta corretta creazione dell'oggetto ListAdapter per poter funzionare.

Pre-condizioni: l'oggetto ListAdapter deve essere stato correttamente creato con dimensione uguale a zero, creazione di una SubList.

Post-condizioni: l'oggetto ListAdapter deve essere svuotato dei suoi elementi e la dimensione deve tornare a zero.

Valori attesi: Il metodo deve restituire { true } nel caso l'elemento cercato fosse presente nella SubList, ovvero in una delle due verifiche effettuate, { false } nel caso contrario.

51. subListContainsTestExc()

Sommario: questo test è finalizzato a verificare la correttezza del metodo contains(Object o). Viene verificato che lanci effettivamente eccezioni nei casi non validi.

Design del caso di test: la verifica del metodo viene eseguita in due successivi test, rispettivamente il numero 50. ed il numero 51.

Descrizione test: riempimento lista, creazione SubList, viene cercato all'interno della SubList il valore nullo { null }, tale inserimento deve causare il lancio dell'eccezione NullPointerException. Tale test prevede l'avvenuta corretta creazione dell'oggetto ListAdapter per poter funzionare.

Pre-condizioni: l'oggetto ListAdapter deve essere stato correttamente creato con dimensione uguale a zero, creazione di una SubList.

Post-condizioni: l'oggetto ListAdapter deve essere svuotato dei suoi elementi e la dimensione deve tornare a zero.

Valori attesi: La ricerca all'interno della SubList del valore nullo { null } deve causare il lancio dell'eccezione NullPointerException.

52. subListContainsAllTest()

Sommario: questo test è finalizzato a verificare la correttezza del metodo containsAll(HCollection c). Viene verificato la corretta ricerca di ogni elemento di una Collection di elementi all'interno della SubList.

Design del caso di test: la verifica del metodo viene eseguita in due successivi test, rispettivamente il numero 52. ed il numero 53.

Descrizione test: riempimento lista, creazione SubList, creazione di una Collection di due elementi esistenti all'interno della SubList, ricerca della Collection nella SubList, aggiunta di un elemento non presente nella SubList all'interno della Collection, ricerca della Collection nella SubList.

Pre-condizioni: l'oggetto ListAdapter deve essere stato correttamente creato con dimensione uguale a zero, la Collection deve essere correttamente creata con due valori, creazione di una SubList.

Post-condizioni: l'oggetto ListAdapter deve essere svuotato dei suoi elementi e la dimensione deve tornare a zero.

Valori attesi: Il metodo deve essere capace di verificare l'esistenza di tutti gli elementi di una Collection all'interno di una SubList. Uno dei due tentativi porterà il metodo a restituire { true }, l'altro porterà il metodo a restituire { false }.

53. subListContainsAllTestExc()

Sommario: questo test è finalizzato a verificare la correttezza del metodo containsAll(HCollection c). Viene verificato il corretto funzionamento delle eccezioni nel caso di ricerca di una Collection nulla o contenente elementi nulli.

Design del caso di test: la verifica del metodo viene eseguita in due successivi test, rispettivamente il numero 52. ed il numero 53.

Descrizione test: creazione Collection nulla, riempimento lista, creazione SubList, controllo che la ricerca di tale Collection nella SubList restituisca NullPointerException, creazione di una Collection con un solo elemento nullo, controllo che la ricerca di tale collezione nella SubList restituisca NullPointerException. Tale test prevede l'avvenuta corretta creazione dell'oggetto ListAdapter per poter funzionare.

Pre-condizioni: l'oggetto ListAdapter deve essere stato correttamente creato con dimensione uguale a zero e che la Collection venga correttamente creata, creazione di una SubList.

Post-condizioni: l'oggetto ListAdapter deve essere svuotato dei suoi elementi e la dimensione deve tornare a zero.

Valori attesi: Il metodo deve essere capace di lanciare NullPointerException nel caso che la Collection ricercata sia nulla o contenga valori nulli { null }.

54. subListEqualsTest()

Sommario: questo test è finalizzato a verificare la correttezza del metodo equals(Object o). Viene verificato la corrispondenza di due ListAdapter.SubList tra loro.

Design del caso di test: la verifica del metodo viene eseguita interamente all'interno di questo test.

Descrizione test: riempimento lista, creazione nuova lista identica alla precedente, verifica che le due SubList siano uguali, aggiunta ad una delle due SubList di un elemento, verifica che le due SubList non siano più uguali. Tale test prevede l'avvenuta corretta creazione di due oggetti ListAdapter per poter funzionare.

Pre-condizioni: l'oggetto ListAdapter deve essere stato correttamente creato con dimensione uguale a zero e un secondo oggetto ListAdapter uguale al primo deve essere creato, creazione di due SubList.

Post-condizioni: l'oggetto ListAdapter deve essere svuotato dei suoi elementi e la dimensione deve tornare a zero.

Valori attesi: per la prima verifica sono state realizzate due SubList contenenti gli stessi valori tra loro, quindi il risultato della funzione equals() dovrà essere { true }, dopo l'aggiunta di un nuovo valore all'interno di una SubList, esse non saranno più uguali, facendo sì che il risultato della funzione equals() sia { false }.

55. subListHashCodeTest()

Sommario: questo test è finalizzato a verificare la correttezza del metodo hashCode(). Viene verificato la corrispondenza tra gli hashCode di due ListAdapter.SubList tra loro, ovvero che il metodo restituisca lo stesso valore nel caso che venga chiamato in SubList uguali, come richiesto dalla documentazione.

Design del caso di test: la verifica del metodo viene eseguita interamente all'interno di questo test.

Descrizione test: riempimento lista, creazione SubList, creazione nuova SubList identica alla precedente, verifica che le due SubList siano uguali, aggiunta ad una delle due SubList di un elemento, verifica che gli hashCode delle due SubList siano uguali. Tale test prevede l'avvenuta corretta creazione di due oggetti ListAdapter per poter funzionare.

Pre-condizioni: l'oggetto ListAdapter deve essere stato correttamente creato con dimensione uguale a zero e un secondo oggetto ListAdapter uguale al primo deve essere creato, creazione di due SubList.

Post-condizioni: l'oggetto ListAdapter deve essere svuotato dei suoi elementi e la dimensione deve tornare a zero.

Valori attesi: come il metodo hashCode stabilisce, la chiamata di esso su due SubList uguali deve poter fornire lo stesso valore. Dal test quindi ci si attende di ottenere una corrispondenza tra i due risultati.

56. subListIsEmptyTest()

Sommario: questo test è finalizzato a verificare la correttezza del metodo isEmpty(). Viene verificato la restituzione del corretto valore booleano nel caso la SubList abbia elementi e non.

Design del caso di test: la verifica del metodo viene eseguita interamente all'interno di questo test.

Descrizione test: riempimento lista, creazione SubList, controllo tramite isEmpty() che la SubList non sia vuota, rimozione degli elementi tramite il metodo clear(), controllo tramite isEmpty() che la SubList sia vuota. Tale test prevede l'avvenuta corretta creazione dell'oggetto ListAdapter per poter funzionare.

Pre-condizioni: l'oggetto ListAdapter deve essere stato correttamente creato con dimensione uguale a zero, creazione di due SubList.

Post-condizioni: l'oggetto ListAdapter deve essere svuotato dei suoi elementi e la dimensione deve tornare a zero.

Valori attesi: dopo l'utilizzo del metodo isEmpty() la SubList deve essere restituire { true } nel caso la SubList fosse vuota, { false } nel caso non lo fosse.

57. subListIteratorTest()

Sommario: questo test è finalizzato a verificare la correttezza del metodo iterator(). Viene verificata la corretta restituzione di un iteratore sulla SubList, i metodi ad esso associati e il corretto funzionamento delle eccezione.

Design del caso di test: la verifica del metodo viene ripetuta con successivi controlli anche successivamente, si tratta dei test numero 22. e numero 57.

Descrizione test: riempimento lista, creazione SubList, creazione di un iteratore, controllo del corretto funzionamento del metodo next(), sia che restituisca il valore corretto, sia che non restituisca un valore errato, controllo del corretto funzionamento del metodo hasNext(), utilizzato per spostarsi all'ultimo elemento e controllando che quello sia effettivamente l'ultimo elemento chiamando next() e

controllando che lanci NullPointerException. Tale test prevede l'avvenuta corretta creazione di dell'oggetto ListAdapter per poter funzionare.

Pre-condizioni: l'oggetto ListAdapter deve essere stato correttamente creato con dimensione uguale a zero, creazione di una SubList.

Post-condizioni: l'oggetto ListAdapter deve essere svuotato dei suoi elementi e la dimensione deve tornare a zero.

Valori attesi: ci si aspetta che il metodo restituisca un iteratore, e che l'iteratore restituito funzioni correttamente.

58. subListRemoveTest()

Sommario: questo test è finalizzato a verificare la correttezza del metodo remove(Object o). Viene verificato la corretta rimozione di un booleano che descriva la rimozione o meno di un elemento.

Design del caso di test: la verifica del metodo viene eseguita in due successivi test, rispettivamente il numero 58. ed il numero 59.

Descrizione test: riempimento lista, creazione SubList, rimozione di un elemento presente nella SubList tramite remove(Object o) e controllo corretta restituzione di { true } e della corretta diminuzione della dimensione della SubList. Tentativo di rimozione di un oggetto non presente nella SubList, controllo che il metodo restituisca { false } e che la dimensione della SubList non sia cambiata. Tale test prevede l'avvenuta corretta creazione dell'oggetto ListAdapter per poter funzionare.

Pre-condizioni: l'oggetto ListAdapter deve essere stato correttamente creato con dimensione uguale a zero, creazione di due SubList.

Post-condizioni: l'oggetto ListAdapter deve essere svuotato dei suoi elementi e la dimensione deve tornare a zero.

Valori attesi: Il metodo deve essere capace di rimuovere gli oggetti indicati, facendo scalare tutti i risultati presenti alla sua sinistra. Deve correttamente diminuire di 1 la dimensione della SubList.

59. subListRemoveTestExe()

Sommario: questo test è finalizzato a verificare la correttezza del metodo remove(Object o). Viene verificato il corretto funzionamento delle eccezioni.

Design del caso di test: la verifica del metodo viene eseguita in due successivi test, rispettivamente il numero 58. ed il numero 59.

Descrizione test: creazione SubList vuota, rimozione di un elemento nullo { null } avendo la SubList vuota e controllo corretto lancio NullPointerException, riempimento SubList, rimozione di un elemento nullo { null }, controllo corretto lancio NullPointerException. Tale test prevede l'avvenuta corretta creazione dell'oggetto ListAdapter per poter funzionare.

Pre-condizioni: l'oggetto ListAdapter deve essere stato correttamente creato con dimensione uguale a zero, creazione di una SubList.

Post-condizioni: l'oggetto ListAdapter deve essere svuotato dei suoi elementi e la dimensione deve tornare a zero.

Valori attesi: Il metodo deve essere capace di lanciare NullPointerException in caso di utilizzo di oggetti non validi durante la rimozione, quindi { null }.

60. subListRemoveAllTest()

Sommario: questo test è finalizzato a verificare la correttezza del metodo removeAll(HCollection c). Viene verificato la corretta rimozione di una Collection di elementi dalla SubList.

Design del caso di test: la verifica del metodo viene eseguita in due successivi test, rispettivamente il numero 60. ed il numero 61.

Descrizione test: creazione di una Collection di due elementi entrambi presenti nella lista, riempimento lista, creazione SubList, rimozione della Collection dalla SubList, controllo corretta diminuzione della dimensione della SubList. Aggiunta di un elemento non presente nella SubList alla Collection, pulizia lista e nuovo riempimento di essa, rimozione della Collection dalla SubList e controllo corretta diminuzione della dimensione della SubList. Tale test prevede l'avvenuta corretta creazione dell'oggetto ListAdapter per poter funzionare e la corretta creazione della Collection.

Pre-condizioni: l'oggetto ListAdapter deve essere stato correttamente creato con dimensione uguale a zero, la Collection deve essere correttamente creata con due valori, creazione di una SubList.

Post-condizioni: l'oggetto ListAdapter deve essere svuotato dei suoi elementi e la dimensione deve tornare a zero.

Valori attesi: Il metodo deve essere capace di rimuovere tutti gli oggetti di una Collection. Deve correttamente ridurre la dimensione della SubList di due elementi, ovvero la dimensione degli elementi della Collection creata che fanno parte anche della SubList.

61. subListRemoveAllTestExc()

Sommario: questo test è finalizzato a verificare la correttezza del metodo removeAll(HCollection c). Viene verificato il corretto funzionamento delle eccezioni nel caso di rimozione di una Collection nulla.

Design del caso di test: la verifica del metodo viene eseguita in due successivi test, rispettivamente il numero 60. ed il numero 61.

Descrizione test: riempimento lista, creazione SubList, creazione Collection nulla, controllo che la rimozione di tale Collection dalla SubList restituisca NullPointerException. Tale test prevede l'avvenuta corretta creazione dell'oggetto ListAdapter per poter funzionare.

Pre-condizioni: l'oggetto ListAdapter deve essere stato correttamente creato con dimensione uguale a zero, creazione di una SubList.

Post-condizioni: l'oggetto ListAdapter deve essere svuotato dei suoi elementi e la dimensione deve tornare a zero.

Valori attesi: Il metodo deve essere capace di lanciare NullPointerException nel caso che la Collection inserita sia nulla o contenga valori nulli { null }.

62. subListRetainAllTest()

Sommario: questo test è finalizzato a verificare la correttezza del metodo retainAll(HCollection c). Viene verificato la corretta rimozione di tutti gli elementi non presenti in una Collection di elementi dalla SubList.

Design del caso di test: la verifica del metodo viene eseguita in due successivi test, rispettivamente il numero 62. ed il numero 63.

Descrizione test: creazione di una Collection di due elementi entrambi presenti nella SubList, riempimento lista, creazione SubList, utilizzo del metodo retainAll(), controllo corretta diminuzione della dimensione della SubList. Aggiunta di un elemento non presente nella SubList alla Collection, pulizia lista e nuovo riempimento di essa, utilizzo del metodo retainAll() e controllo corretta diminuzione della dimensione della SubList. Tale test prevede l'avvenuta corretta creazione dell'oggetto ListAdapter per poter funzionare e la corretta creazione della Collection.

Pre-condizioni: l'oggetto ListAdapter deve essere stato correttamente creato con dimensione uguale a zero, la Collection deve essere correttamente creata con due valori, creazione di una SubList.

Post-condizioni: l'oggetto ListAdapter deve essere svuotato dei suoi elementi e la dimensione deve tornare a zero.

Valori attesi: Il metodo deve essere capace di rimuovere tutti gli oggetti non presenti nella Collection. Deve correttamente ridurre la dimensione della SubList a due, ovvero la dimensione degli elementi della lista presenti nella Collection creata.

63. subListRetainAllTestExc()

Sommario: questo test è finalizzato a verificare la correttezza del metodo retainAll(HCollection c). Viene verificato il corretto funzionamento delle eccezioni nel caso di utilizzo di una Collection nulla.

Design del caso di test: la verifica del metodo viene eseguita in due successivi test, rispettivamente il numero 62. ed il numero 63.

Descrizione test: creazione Collection nulla, riempimento lista, creazione SubList, controllo che l'utilizzo di retainAll() con tale Collection dalla SubList restituisca NullPointerException. Tale test prevede l'avvenuta corretta creazione dell'oggetto ListAdapter per poter funzionare.

Pre-condizioni: l'oggetto ListAdapter deve essere stato correttamente creato con dimensione uguale a zero, creazione SubList.

Post-condizioni: l'oggetto ListAdapter deve essere svuotato dei suoi elementi e la dimensione deve tornare a zero.

Valori attesi: Il metodo deve essere capace di lanciare NullPointerException nel caso che la Collection inserita sia nulla o contenga valori nulli { null }.

64. subListSizeTest()

Sommario: questo test è finalizzato a verificare la correttezza del metodo size(). Viene verificato la restituzione del corretto valore intero corrispondente al numero di elementi della SubList.

Design del caso di test: la verifica del metodo viene eseguita interamente all'interno di questo test.

Descrizione test: creazione di una SubList vuota, controllo se con la SubList vuota il metodo size() restituisce 0, riempimento lista, controllo tramite size() che la dimensione della SubList corrisponda a 3. Tale test prevede l'avvenuta corretta creazione dell'oggetto ListAdapter per poter funzionare.

Pre-condizioni: l'oggetto ListAdapter deve essere stato correttamente creato con dimensione uguale a zero, creazione di una SubList.

Post-condizioni: l'oggetto ListAdapter deve essere svuotato dei suoi elementi e la dimensione deve tornare a zero.

Valori attesi: il metodo size() deve restituire il corretto numero di elementi in ogni momento.

65. subListToArrayTest()

Sommario: questo test è finalizzato a verificare la correttezza del metodo toArray(). Viene verificata la corretta restituzione di un array contenente gli elementi della SubList nello stesso ordine.

Design del caso di test: la verifica del metodo viene eseguita interamente all'interno di questo test.

Descrizione test: riempimento lista, creazione SubList, creazione di un array tramite il metodo toArray(), controllo che tutti gli elementi dell'array coincidano in valore e posizione a tutti gli elementi della SubList originaria. Tale test prevede l'avvenuta corretta creazione dell'oggetto ListAdapter per poter funzionare.

Pre-condizioni: l'oggetto ListAdapter deve essere stato correttamente creato con dimensione uguale a zero, creazione di una SubList.

Post-condizioni: l'oggetto ListAdapter deve essere svuotato dei suoi elementi e la dimensione deve tornare a zero.

Valori attesi: ci si aspetta che il metodo restituisca un array dalla dimensione, valori e posizioni degli elementi identiche a quelle della SubList.

66. subListToArrayObjTest()

Sommario: questo test è finalizzato a verificare la correttezza del metodo toArray(Object[] a). Viene verificata la corretta restituzione di un array contenente gli elementi della SubList nello stesso ordine utilizzando un array a come destinazione finale.

Design del caso di test: la verifica del metodo viene eseguita in due successivi test, rispettivamente il numero 66. ed il numero 67.

Descrizione test: riempimento lista, creazione SubList, creazione di un array a due elementi e utilizzo del metodo toArray() passando come parametro l'array creato, controllo che tutti gli elementi dell'array coincidano in valore e posizione a tutti gli elementi della SubList originaria. Creazione di un array a venti elementi e utilizzo del metodo toArray() passando come parametro l'array creato, controllo che tutti gli elementi dell'array coincidano in valore e posizione a tutti gli elementi della SubList originaria. Tale test prevede l'avvenuta corretta creazione dell'oggetto ListAdapter per poter funzionare.

Pre-condizioni: l'oggetto ListAdapter deve essere stato correttamente creato con dimensione uguale a zero, creazione di una SubList.

Post-condizioni: l'oggetto ListAdapter deve essere svuotato dei suoi elementi e la dimensione deve tornare a zero.

Valori attesi: ci si aspetta che il metodo restituisca un array con valori e posizioni degli elementi identiche a quelle della SubList originale.

67. subListToArrayObjTestExe()

Sommario: questo test è finalizzato a verificare la correttezza del metodo toArray(Object[] o). Viene verificato il corretto funzionamento delle eccezioni.

Design del caso di test: la verifica del metodo viene eseguita in due successivi test, rispettivamente il numero 66. ed il numero 67.

Descrizione test: riempimento lista, creazione SubList, creazione di un array nullo e passaggio di esso tramite il metodo toArray(), controllo corretto lancio NullPointerException. Tale test prevede l'avvenuta corretta creazione dell'oggetto ListAdapter per poter funzionare.

Pre-condizioni: l'oggetto ListAdapter deve essere stato correttamente creato con dimensione uguale a zero, creazione di una SubList.

Post-condizioni: l'oggetto ListAdapter deve essere svuotato dei suoi elementi e la dimensione deve tornare a zero.

Valori attesi: Il metodo deve essere capace di lanciare lanciare NullPointerException in caso di inserimento di array non validi { null }.

68. subListIndexOfTest()

Sommario: questo test è finalizzato a verificare la correttezza del metodo indexOf(Object o). Viene verificato che passando al metodo un certo oggetto venga restituito il primo indice della sua occorrenza all'interno della SubList o -1 ne caso esso non sia presente.

Design del caso di test: la verifica del metodo viene eseguita in due successivi test, rispettivamente il numero 68. ed il numero 69.

Descrizione test: riempimento lista, creazione SubList, utilizzo del metodo indexOf() e controllo dalla corretta restituzione dell'indice corrispondente. Tale test prevede l'avvenuta corretta creazione dell'oggetto oggetti ListAdapter per poter funzionare.

Pre-condizioni: l'oggetto ListAdapter deve essere stato correttamente creato con dimensione uguale a zero, creazione di una SubList.

Post-condizioni: l'oggetto ListAdapter deve essere svuotato dei suoi elementi e la dimensione deve tornare a zero.

Valori attesi: ci si aspetta che il metodo possa riconoscere la presenza degli oggetti cercati, restituendo l'indice della prima occorrenza o agendo di conseguenza nel caso l'oggetto non sia presente.

69. subListIndexOfTestExc()

Sommario: questo test è finalizzato a verificare la correttezza del metodo getTest(int index). Viene verificato che lanci effettivamente eccezioni nei casi non validi.

Design del caso di test: la verifica del metodo viene eseguita in due successivi test, rispettivamente il numero 68. ed il numero 69.

Descrizione test: riempimento lista, creazione SubList, tentativo di utilizzo del metodo indexOf passando un oggetto nullo, viene successivamente atteso il lancio dell'eccezione NullPointerException. Tale test prevede l'avvenuta corretta creazione dell'oggetto ListAdapter per poter funzionare.

Pre-condizioni: l'oggetto ListAdapter deve essere stato correttamente creato con dimensione uguale a zero, creazione di una SubList.

Post-condizioni: l'oggetto ListAdapter deve essere svuotato dei suoi elementi e la dimensione deve tornare a zero.

Valori attesi: Il metodo deve essere capace di lanciare NullPointerException in caso di utilizzo di Oggetti non validi { null } durante l'inserimento.

70. subListLastIndexOfTest()

Sommario: questo test è finalizzato a verificare la correttezza del metodo lastIndexOf(Object o). Viene verificato che passando al metodo un certo oggetto venga restituito l'ultimo indice della sua occorrenza all'interno della SubList o -1 ne caso esso non sia presente.

Design del caso di test: la verifica del metodo viene eseguita in due successivi test, rispettivamente il numero 70. ed il numero 71.

Descrizione test: riempimento lista, creazione SubList, inserimento di altri due elementi alla sua coda, utilizzo del metodo lastIndexOf() e controllo dalla corretta restituzione dell'indice corrispondente. Vengono cercati due oggetti, uno presente effettivamente all'interno della SubList e uno non presente, in modo tale da verificare anche la corretta restituzione del valore -1. Tale test prevede l'avvenuta corretta creazione dell'oggetto ListAdapter per poter funzionare.

Pre-condizioni: l'oggetto ListAdapter deve essere stato correttamente creato con dimensione uguale a zero, creazione di una SubList.

Post-condizioni: l'oggetto ListAdapter deve essere svuotato dei suoi elementi e la dimensione deve tornare a zero.

Valori attesi: ci si aspetta che il metodo possa riconoscere la presenza degli oggetti cercati, restituendo l'indice dell'ultima occorrenza o agendo di conseguenza nel caso l'oggetto non sia presente.

71. subListLastIndexOfTestExc()

Sommario: questo test è finalizzato a verificare la correttezza del metodo lastIndexOfTestExc (Object o). Viene verificato che lanci effettivamente eccezioni nei casi non validi.

Design del caso di test: la verifica del metodo viene eseguita in due successivi test, rispettivamente il numero 70. ed il numero 71.

Descrizione test: riempimento lista, creazione SubList tentativo di utilizzo del metodo lastIndexOf() passando un oggetto nullo, viene successivamente atteso il lancio dell'eccezione NullPointerException. Tale test prevede l'avvenuta corretta creazione dell'oggetto ListAdapter per poter funzionare.

Pre-condizioni: l'oggetto ListAdapter deve essere stato correttamente creato con dimensione uguale a zero, creazione di una SubList.

Post-condizioni: l'oggetto ListAdapter deve essere svuotato dei suoi elementi e la dimensione deve tornare a zero.

Valori attesi: Il metodo deve essere capace di lanciare NullPointerException in caso di utilizzo di Oggetti non validi { null } durante l'inserimento.