Documentazione subListTest

Questa suite ha l'obiettivo di testare tutti i metodi di ListAdapter.sublist(), alla ricerca di errori per verificarne l'assenza. I metodi vengono testati con variabili di diverso tipo e parametri validi e non, allo scopo di controllare il funzionamento delle eccezioni.
Versione JUnit utilizzata per eseguire tutti i test: junit-4.13
Componenti del frame work utilizzati: assertArrayEquals, assertEquals, assertNotEquals, assertFalse, assertNotNull, assertNull, assertTrue.

Pre condizioni generali:

Prima di ogni test, viene creata una lista vuota lst1 con il costruttore di default, a cui sono stati aggiunti elementi di vario tipo, nell'ordine {null, (double) 3.3333333333333335, (int) -5, (char) L, (string) Test, (int) 7, (double) 3.3333333333333335, (boolean) true}. Viene inoltre istanziata una lista sub, che viene poi inizializzata con il metodo sublist relativa alla porzione di lst1 che va dall'indice 2 (compreso) all'indice 6 (non compreso). Sub quindi conterrà, nell'ordine {(int) -5, (char) L, (string) Test, (int) 7}.

0. b4()

- Riassunto: @Before tramite il quale viene riempita la lista lst1, per essere poi utilizzata e viene inizializzata la HList sub.
- Design: vengono aggiunti elementi alla lista elementi di vario tipo, e viene inizializzata la sublist.
- Descrizione: vengono aggiunti a lst1 elementi di vario tipo: nell'ordine {null, (double) 3.3333333333333335, (int) -5, (char) L, (string) Test, (int) 7, (double) 3.3333333333333335, (boolean) true}. Viene inoltre inizializzata la lista sub con il metodo sublist relativa alla porzione di lst1 che va dall'indice 2 (compreso) all'indice 6 (non compreso). Sub quindi conterrà, nell'ordine {(int) -5, (char) L, (string) Test, (int) 7}.
- Pre condizioni: lista vuota lst1 lista sub non inizializzata.
- Post condizioni: lista lst1 piena, lista sub inizializzata e piena.
- Risultato atteso: lista lst1 piena.

1. constructorNotNull()

- Riassunto: test del costruttore ListAdapter.sublist(int start, int end) per verificare che l'oggetto SubList creato non sia nullo.
- Design: viene creata la sublist e verificato che non sia nulla.
- Descrizione: viene creata una nuova SubList con il metodo ListAdapter.sublist() e viene verificato che non sia nulla.
- Pre condizioni: lista piena lst1, sublist sub piena.
- Post condizioni: lista lst1 e la sublist sub tornano alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: lst1.subList(2, 4) non nulla.

2. constructorEquals()

- Riassunto: test del costruttore listAdapter.sublist(int start, int end) per verificare due nuove sublist siano uguali.

- Design: vengono create due liste piene e viene verificato che due sublist con gli stessi parametri su liste diverse siano uguali.
- Descrizione: viene creata un lista vuota lst2, a cui vengono aggiunti tutti gli elementi di lst1. Successivamente vengono create due sublist con gli stessi parametri su lst1 ed lst2, che vengono poi confrontate.
- Pre condizioni: lista piena lst1, sublist sub piena.
- Post condizioni: lista lst1 e la sublist sub tornano alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: le due nuove sublist sono uguali.

3. constructorIndexOutOfBounds()

- Riassunto: test dell'eccezione IndexOutOfBounds() del metodo ListAdapter. sublist(int start, int end).
- Design: viene creata una nuova sublist con parametro non accettabile.
- Descrizione: viene creata una nuova sublist con il parametro end maggiore della dimensione della lista su cui viene creata.
- Pre condizioni: lista piena lst1, sublist sub piena.
- Post condizioni: lista lst1 e la sublist sub tornano alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: eccezione IndexOutOfBounds().

4. constructorIndexOutOfBounds2()

- Riassunto: test dell'eccezione IndexOutOfBounds() del metodo ListAdapter. sublist(int start, int end).
- Design: viene creata una nuova sublist con parametro non accettabile.
- Descrizione: viene creata una nuova sublist con il parametro start minore di 0.
- Pre condizioni: lista piena lst1, sublist sub piena.
- Post condizioni: lista lst1 e la sublist sub tornano alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: eccezione IndexOutOfBounds().

5. constructorIndexOutOfBounds3()

- Riassunto: test dell'eccezione IndexOutOfBounds() del metodo ListAdapter. sublist(int start, int end).
- Design: viene creata una nuova sublist con parametro non accettabile.
- Descrizione: viene creata una nuova sublist con il parametro start maggiore del parametro end.
- Pre condizioni: lista piena lst1, sublist sub piena.
- Post condizioni: lista lst1 e la sublist sub tornano alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: eccezione IndexOutOfBounds().

6. addIndexOutOfBounds()

- Riassunto: test dell'eccezione IndexOutOfBounds() del metodo add(int index, Object element).
- Design: viene aggiunto un elemento con parametro non accettabile.
- Descrizione: viene aggiunto un elemento alla lista sub in un indice non accettabile minore di 0 (-2).
- Pre condizioni: lista piena lst1, sublist sub piena.
- Post condizioni: lista lst1 e la sublist sub tornano alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: eccezione IndexOutOfBounds().

7. addIndexOutOfBounds2()

- Riassunto: test dell'eccezione IndexOutOfBounds() del metodo add(int index, Object element).
- Design: viene aggiunto un elemento con parametro non accettabile.
- Descrizione: viene aggiunto un elemento alla lista sub in un indice maggiore della sua dimensione (6).
- Pre condizioni: lista piena lst1, sublist sub piena.
- Post condizioni: lista lst1 e la sublist sub tornano alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: eccezione IndexOutOfBounds().

8. addIndex()

- Riassunto: test del metodo add(int index, Object element).
- Design: viene aggiunto un elemento con parametro accettabile.
- Descrizione: viene aggiunto un elemento alla lista sub in un indice accettabile, e viene controllata la posizione di tutti gli elementi presenti nella sublist.
- Pre condizioni: lista piena lst1, sublist sub piena.
- Post condizioni: lista lst1 e la sublist sub tornano alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: elementi aggiunti nella posizioni previste ed elementi già presenti spostati di conseguenza.

9. addIndex2()

- Riassunto: test del metodo add(int index, Object element).
- Design: vengono aggiunti 4 elementi a due sublist che vengono poi confrontate per verificare che il metodo si comporti allo stesso modo in entrambe le sublist.
- Descrizione: viene creata una lista lst2 a cui vengono aggiunti tutti gli elementi di lst1 e ne viene creata una sublist sub2 con gli stessi parametri di sub. Vengono aggiunti interi casuali compresi fra -100 e 100 (ma uguali a due a due) nelle stesse posizioni in sub e nella nuova sublist sub2. Viene controllato se nelle due liste lst1 e lst2 sono presenti gli stessi elementi nelle stesse posizioni.
- Pre condizioni: lista piena lst1, sublist sub piena.
- Post condizioni: lista lst1 e la sublist sub tornano alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: le due liste "originali" (lst1 e lst2) contengono gli stessi elementi nelle stesse posizioni.

10. add()

- Riassunto: test del metodo add().
- Design: vengono aggiunti elementi di vario tipo ad una sublist.
- Descrizione: vengono aggiunti elementi a sub e ne viene controllata la posizione su lst1 e su sub.
- Pre condizioni: lista piena lst1, sublist sub piena.
- Post condizioni: lista lst1 e la sublist sub tornano alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: elementi aggiunti nella posizioni previste.

11. add2()

- Riassunto: test del metodo add(Object element).
- Design: vengono aggiunti 4 elementi a due sublist che vengono poi confrontate per verificare che il metodo si comporti allo stesso modo in entrambe le sublist.

- Descrizione: viene creata una lista lst2 a cui vengono aggiunti tutti gli elementi di lst1 e ne viene creata una sublist sub2 con gli stessi parametri di sub. Vengono aggiunti interi casuali compresi fra -100 e 100 (ma uguali a due a due) nelle stesse posizioni in sub e nella nuova sublist sub2. Viene controllato se nelle due liste lst1 e lst2 sono presenti gli stessi elementi nelle stesse posizioni.
- Pre condizioni: lista piena lst1, sublist sub piena.
- Post condizioni: lista lst1 e la sublist sub tornano alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: le due liste "originali" (lst1 e lst2) contengono gli stessi elementi nelle stesse posizioni.

12. addAllNullPointer()

- Riassunto: test dell'eccezione NullPointer del metodo addAll(HCollection c).
- Design: viene aggiunto un elemento nullo ad una sublist con il metodo addAll(HCollection c).
- Descrizione: viene aggiunto un elemento nullo sub con il metodo addAll(HCollection c).
- Pre condizioni: lista piena lst1, sublist sub piena.
- Post condizioni: lista lst1 e la sublist sub tornano alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: eccezione NullPointer.

13. addAll()

- Riassunto: test del metodo addAll(HCollection c).
- Design: viene aggiunta una HCollection ad una sublist e viene verificata la corretta aggiunta degli elementi.
- Descrizione: viene creata una lista lst2 a cui vengono aggiunti elementi di vario tipo. Vengono poi aggiunti alcuni elementi di lst1 a lst2 tramite l'utilizzo di una sublist e del metodo addAll(HCollection c). viene poi verificata la posizione degli elementi aggiunti a lst2.
- Pre condizioni: lista piena lst1, sublist sub piena.
- Post condizioni: lista lst1 e la sublist sub tornano alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: la lista lst2 contiene gli elementi previsti nelle posizioni attese.

14. addAllIndexNullPointer()

- Riassunto: test dell'eccezione NullPointer del metodo addAll(int index, HCollection c).
- Design: viene aggiunto un elemento nullo ad una sublist con il metodo addAll(int index, HCollection c).
- Descrizione: viene aggiunto un elemento nullo sub con il metodo addAll(int index, HCollection c).
- Pre condizioni: lista piena lst1, sublist sub piena.
- Post condizioni: lista lst1 e la sublist sub tornano alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: eccezione NullPointer.

15. addAllIndexOutOfBounds()

- Riassunto: test dell'eccezione IndexOutOfBounds() del metodo addAll(int index, HCollection c).
- Design: viene aggiunta un'HCollection valida ad una sublist con il metodo addAll(int index, HCollection c) in un indice non accettabile.
- Descrizione: viene creata una lista lst2 e le viene aggiunto un elemento. Essa viene aggiunta a sub in un indice non accettabile maggiore della sua dimensione (8).
- Pre condizioni: lista piena lst1, sublist sub piena.

- Post condizioni: lista lst1 e la sublist sub tornano alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: eccezione IndexOutOfBounds().

16. addAllIndexOutOfBounds2()

- Riassunto: test dell'eccezione IndexOutOfBounds() del metodo addAll(int index, HCollection c).
- Design: viene aggiunta un'HCollection valida ad una lista con il metodo addAll(int index, HCollection c) in un indice non accettabile.
- Descrizione: viene creata una lista lst2 e le viene aggiunto un elemento. Essa viene aggiunta ad lst1 in un indice non accettabile minore di 0 (-1).
- Pre condizioni: lista piena lst1, sublist sub piena.
- Post condizioni: lista lst1 e la sublist sub tornano alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: eccezione IndexOutOfBounds().

17. addAllIndex()

- Riassunto: test del metodo addAll(int index, HCollection c).
- Design: viene aggiunta una HCollection ad una sublist in un indice valido e viene verificata la corretta aggiunta degli elementi.
- Descrizione: viene creata una lista lst2 a cui vengono aggiunti elementi di vario tipo. Vengono poi aggiunti alcuni elementi di lst1 a lst2 tramite l'utilizzo di una sublist e del metodo addAll(int index, HCollection c). Viene poi verificata la posizione degli elementi aggiunti a lst2.
- Pre condizioni: lista piena lst1, sublist sub piena.
- Post condizioni: lista lst1 e la sublist sub tornano alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: le due liste (lst3 ed lst2) contengono gli stessi elementi nelle stesse posizioni.

18. clear()

- Riassunto: test del metodo clear().
- Design: viene svuotata la sublist e viene verificato che la lista madre abbia una dimensione adeguata.
- Descrizione: viene svuotata la sublist sub e viene verificato che la dimensione di lst1 sia uguale alla sua dimensione originale meno la dimensione della sublist, ovvero verifica che siano stati rimossi da lst1 tutti e soli gli elementi di sub.
- Pre condizioni: lista piena lst1, sublist sub piena.
- Post condizioni: lista lst1 e la sublist sub tornano alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: lista lst1 vuota, quindi con dimensione uguale a 0.

19. contains()

- Riassunto: test del metodo contains().
- Design: viene verificata la presenza di elementi in una sublist tramite il metodo contains(). Viene inoltre verificata l'efficienza del metodo controllando la presenza di elementi non facenti parti della sublist.
- Descrizione: viene verificata la presenza degli elementi in sub presenti dalle pre condizioni col metodo contains(). Viene inoltre verificata l'efficienza del metodo controllando la presenza di elementi non facenti parti della sublist.
- Pre condizioni: lista piena lst1, sublist sub piena.
- Post condizioni: lista lst1 e la sublist sub tornano alla situazione iniziale (vedasi pre condizioni).

- Risultato atteso: il metodo contains() ritorna true con gli elementi effettivamente presenti e false con gli elementi non facenti parte della sublist.

20. containsAllNullPointer()

- Riassunto: test dell'eccezione NullPointer del metodo containsAll(HCollection c).
- Design: viene passato un null come parametro al metodo addAll(HCollection c).
- Descrizione: viene passato un null come parametro al metodo addAll(HCollection c).
- Pre condizioni: lista piena lst1, sublist sub piena.
- Post condizioni: lista lst1 e la sublist sub tornano alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: eccezione NullPointer.

21. containsAll()

- Riassunto: test del metodo containsAll(HCollection c).
- Design: viene verificata la presenza degli elementi di una HCollection in una sub con il metodo containsAll(HCollection c).
- Descrizione: viene creata la lista vuota lst2, a cui vengono aggiunti tutti gli elementi presenti in sub. Viene poi verificata tramite il metodo containsAll(HCollection c) la presenza di tutti gli elementi di lst1 in lst2 e viceversa.
- Pre condizioni: lista piena lst1, sublist sub piena.
- Post condizioni: lista lst1 e la sublist sub tornano alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: il metodo riconosce che lst2 contiene tutti gli elementi di lst1 ma che lst1 non contiene tutti gli elementi di lst1.

22. equals()

- Riassunto: test del metodo equals().
- Design: viene testato il metodo equals() confrontando la sublist sub con sublist uguali e diverse.
- Descrizione: viene creata la lista vuota lst2, a cui vengono aggiunti tutti gli elementi di lst1. Viene verificata l'uguaglianza tra sub ed una sublist di lst2 creata con gli stessi parametri di sub con il metodo equals(). Successivamente viene verificata la disuguaglianza tra sub ed una sublist di lst2 creata con parametri diversi da quelli di sub con il metodo equals(). Infine viene aggiunto un elemento a lst2 e viene effettuato di nuovo il primo test, stavolta verificandone la disuguaglianza.
- Pre condizioni: lista piena lst1, sublist sub piena.
- Post condizioni: lista lst1 e la sublist sub tornano alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: il metodo restituisce true solo quando le sublist confrontate sono effettivamente identiche.

23. getIndexOutOfBounds()

- Riassunto: test dell'eccezione IndexOutOfBounds() del metodo get(int index).
- Design: viene utilizzato il metodo get(int index) con un indice non accettabile.
- Descrizione: si prova a prendere un elemento da lst1 in un indice non accettabile minore di 0 (-1).
- Pre condizioni: lista piena lst1, sublist sub piena.
- Post condizioni: lista lst1 e la sublist sub tornano alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: eccezione IndexOutOfBounds().

24. getIndexOutOfBounds2()

- Riassunto: test dell'eccezione IndexOutOfBounds() del metodo get(int index).
- Design: viene utilizzato il metodo get(int index) con un indice non accettabile.
- Descrizione: si prova a prendere un elemento da lst1 in un indice non accettabile maggiore della sua dimensione (6).
- Pre condizioni: lista piena lst1, sublist sub piena.
- Post condizioni: lista lst1 e la sublist sub tornano alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: eccezione IndexOutOfBounds().

25. get(int index)

- Riassunto: test del metodo get(int index)
- Design: viene testato il metodo get(int index) provando a prendere tutti gli elementi di sub tramite get(int index).
- Descrizione: viene testato il metodo get(int index) provando a prendere tutti gli elementi di sub tramite get(int index) e viene verificato che il metodo restituisca gli elementi corretti agli indici corretti.
- Pre condizioni: lista piena lst1, sublist sub piena.
- Post condizioni: lista lst1 e la sublist sub tornano alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: il metodo restituisce gli elementi corretti agli indici corretti.

26. hashCodeTest()

- Riassunto: test del metodo hashCode().
- Design: vengono confrontati i valori dell'hashcode restituito dall'omonimo metodo con dei valori calcolati esternamente.
- Descrizione: viene confrontato il valore dell'hashcode restituito da una sublist contenente solo il valore null con il valore dell'hashcode restituito dall'omonimo metodo con un valore calcolate esternamente. Viene poi verificato che il valore dell'hashcode restituito sia diverso per una sublist differente.
- Pre condizioni: lista piena lst1, sublist sub piena.
- Post condizioni: lista lst1 e la sublist sub tornano alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: i valori restituiti dal metodo coincidono con quelli previsti.

27. indexOf()

- Riassunto: test del metodo indexOf().
- Design: viene testato il metodo indexOf() verificando l'indice di tutti gli elementi di sub tramite indexOf(). Il metodo viene inoltre testato con un elemento non appartenente a sub, ma appartenente a lst1.
- Descrizione: viene testato il metodo indexOf() richiedendo l'indice di tutti gli elementi di sub tramite indexOf() e controllando che il metodo restituisca gli indici corretti degli elementi presenti nella sublist. Il metodo viene inoltre testato con un elemento non appartenente sub, ma appartenente a lst1.
- Pre condizioni: lista piena lst1, sublist sub piena.
- Post condizioni: lista lst1 e la sublist sub tornano alla situazione iniziale (vedasi pre condizioni).

- Risultato atteso: il metodo restituisce gli indici corretti degli elementi presenti nella sublist, e -1 per gli elementi non presenti nella sublist.

28. isEmpty()

- Riassunto: test del metodo isEmpty().
- Design: viene verificato con il metodo isEmpty() che la lista sia vuota dopo averla svuotata.
- Descrizione: viene svuotata sub, e viene verificato che non sia vuota, in quanto sono "scalati" nella sublist elementi appartenenti a lst1 ma con un indice maggiore. Viene poi svuotata di nuovo sub e verificato che stavolta si effettivamente vuota in quanto non persistevano elementi appartenenti a lst1 di indice maggiore.
- Pre condizioni: lista piena lst1, sublist sub piena.
- Post condizioni: lista lst1 e la sublist sub tornano alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: il metodo restituisce true se la sublist è effettivamente vuota, e false altrimenti.

29. isEmpty2()

- Riassunto: test del metodo isEmpty().
- Design: viene verificato con il metodo isEmpty() che la lista sia vuota dopo aver svuotato la lista madre.
- Descrizione: viene svuotata lst1, e viene verificato che il metodo isEmpty() accerti che sia vuota.
- Pre condizioni: lista piena lst1, sublist sub piena.
- Post condizioni: lista lst1 e la sublist sub tornano alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: il metodo restituisce true se la sublist è vuota.

30. iteratorNotNull()

- Riassunto: test del metodo iterator() per verificare che l'oggetto HIterator restituito non sia nullo.
- Design: viene creato l'iteratore su una sublist e verificato che non sia nullo.
- Descrizione: viene creato un iteratore di sub con il metodo iterator() e viene verificato che non sia nullo.
- Pre condizioni: lista piena lst1, sublist sub piena.
- Post condizioni: lista lst1 e la sublist sub tornano alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: iteratore non nullo.

31. iteratorHasNext()

- Riassunto: test del metodo hasNext() di HIterator.
- Design: viene creato l'iteratore e verificato che il metodo hasNext() funzioni correttamente.
- Descrizione: viene creato un iteratore di sub con il metodo iterator() e viene verificato che hasNext() funzioni correttamente chiamandolo finché non si raggiunge la fine della lista. Viene quindi verificato che hasNext() restituisca false.
- Post condizioni: lista lst1 e la sublist sub tornano alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: hasNext() restituisce true solo se ci sono elementi successivi.

32. iteratorNextNoSuchElementException()

- Riassunto: test dell'eccezione NoSuchElementException() del metodo next() di HIterator.

- Design: viene creato l'iteratore su una sublist vuota e verificato che chiamando next() venga lanciata l'eccezione NoSuchElementException().
- Descrizione: dopo aver svuotato lst1, viene creato l'iteratore su sub e chiamato next().
- Pre condizioni: lista piena lst1, sublist sub piena.
- Post condizioni: lista lst1 e la sublist sub tornano alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: eccezione NoSuchElementException().

33. iteratorNextNoSuchElementException2()

- Riassunto: test dell'eccezione NoSuchElementException() del metodo next() di HIterator.
- Design: viene creato l'iteratore su una sublist piena e verificato che chiamando next() alla fine della sublist, venga lanciata l'eccezione NoSuchElementException().
- Descrizione: viene creato l'iteratore su sub e chiamato next() finché l'iteratore non arriva alla fine della lista. Dopodiché viene chiamato nuovamente next.
- Pre condizioni: lista piena lst1, sublist sub piena.
- Post condizioni: lista lst1 e la sublist sub tornano alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: eccezione NoSuchElementException().

34. iteratorNext()

- Riassunto: test del metodo next() di HIterator.
- Design: viene creato l'iteratore e verificato che il metodo next() funzioni correttamente.
- Descrizione: viene creato l'iteratore sulla sublist piena sub e verificato che il metodo next() restituisca gli elementi corretti nel giusto ordine.
- Pre condizioni: lista piena lst1, sublist sub piena.
- Post condizioni: lista lst1 e la sublist sub tornano alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: next() restituisce gli elementi corretti nel giusto ordine.

35. iteratorRemoveIllegalStateException()

- Riassunto: test dell'eccezione IllegalState() del metodo remove() di HIterator.
- Design: viene creato l'iteratore su una sublist piena e verificato che chiamando remove() subito dopo la creazione dell'iteratore, venga lanciata l'IllegalStateException().
- Descrizione: viene creato l'iteratore sulla sublist piena sub chiamato remove() subito dopo.
- Pre condizioni: lista piena lst1, sublist sub piena.
- Post condizioni: lista lst1 e la sublist sub tornano alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: eccezione IllegalStateException().

36. iteratorRemoveIllegalStateException2()

- Riassunto: test dell'eccezione IllegalState() del metodo remove() di HIterator.
- Design: viene creato l'iteratore su una sublist piena e verificato che chiamando remove() due volte di fila, venga lanciata l'IllegalStateException().
- Descrizione: viene creato l'iteratore sulla sublist piena sub, aggiunto un elemento all'iteratore e chiamato remove() due volte consecutive.
- Pre condizioni: lista piena lst1, sublist sub piena.
- Post condizioni: lista lst1 e la sublist sub tornano alla situazione iniziale (vedasi pre condizioni).

- Risultato atteso: eccezione IllegalStateException().

37. iteratorRemove()

- Riassunto: test del metodo remove() di HIterator.
- Design: viene creato l'iteratore e verificato che il metodo remove() funzioni correttamente.
- Descrizione: viene creato l'iteratore sulla sublist piena sub, viene chiamato più volte il metodo next() e verificato che il metodo remove() rimuova gli elementi corretti. Viene inoltre verificata la dimensione di sub.
- Pre condizioni: lista piena lst1, sublist sub piena.
- Post condizioni: lista lst1 e la sublist sub tornano alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: remove() rimuove gli elementi corretti.

38. lastIndexOf()

- Riassunto: test del metodo lastIndexOf().
- Design: viene testato il metodo lastIndexOf () verificando l'indice di tutti gli elementi di sub tramite lastIndexOf (). Il metodo viene inoltre testato con elementi non appartenenti a sub.
- Descrizione: dopo aver aggiunto a sub un elemento già presente, viene testato il metodo lastIndexOf () richiedendo l'indice di tutti gli elementi di sub tramite lastIndexOf () e controllando che il metodo restituisca gli indici corretti degli elementi presenti nella sublist. Il metodo viene inoltre testato con elementi non appartenente a sub.
- Pre condizioni: lista piena lst1, sublist sub piena.
- Post condizioni: lista lst1 e la sublist sub tornano alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: il metodo restituisce gli indici corretti degli elementi presenti nella sublist, e -1 per gli elementi non presenti nella sublist.

39. listIteratorNotNull()

- Riassunto: test del metodo listIterator() per verificare che l'oggetto ListIterator restituito non sia nullo.
- Design: viene creato l'iteratore e verificato che non sia nullo.
- Descrizione: viene creato un iteratore di sub con il metodo listIterator() e viene verificato che non sia nullo.
- Pre condizioni: lista piena lst1, sublist sub piena.
- Post condizioni: lista lst1 e la sublist sub tornano alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: iteratore non nullo.

40. listIteratorHasNext()

- Riassunto: test del metodo hasNext() di ListIterator.
- Design: viene creato l'iteratore e verificato che il metodo hasNext() funzioni correttamente.
- Descrizione: viene creato un iteratore di sub con il metodo listIterator() e viene verificato che hasNext() funzioni correttamente anche dopo aver aggiunto elementi o aver chiamato il metodo next().
- Pre condizioni: lista piena lst1, sublist sub piena.
- Post condizioni: lista lst1 e la sublist sub tornano alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: hasNext() restituisce true solo se ci sono elementi successivi.

41. listIteratorHasPrevious()

- Riassunto: test del metodo hasPrevious() di ListIterator.
- Design: viene creato l'iteratore e verificato che il metodo hasPrevious() funzioni correttamente.
- Descrizione: viene creato un iteratore di sub con il metodo listIterator() e viene verificato che hasPrevious() funzioni correttamente anche dopo aver svuotato la sublist, aggiunto elementi o aver chiamato il metodo next() o remove().
- Pre condizioni: lista piena lst1, sublist sub piena.
- Post condizioni: lista lst1 e la sublist sub tornano alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: hasprevious() restituisce true solo se ci sono elementi precedenti.

43. listIteratorNextNoSuchElementException()

- Riassunto: test dell'eccezione NoSuchElementException() del metodo next() di ListIterator.
- Design: viene creato l'iteratore su una sublist vuota e verificato che chiamando next() venga lanciata l'eccezione NextNoSuchElementException().
- Descrizione: dopo aver svuotato lst1, viene creato l'iteratore su sub e chiamato next().
- Pre condizioni: lista piena lst1, sublist sub piena.
- Post condizioni: lista lst1 e la sublist sub tornano alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: eccezione NextNoSuchElementException().

44. listIteratorNextNoSuchElementException2()

- Riassunto: test dell'eccezione NoSuchElementException() del metodo next() di ListIterator.
- Design: viene creato l'iteratore su una sublist piena e verificato che chiamando next() alla fine della sublist, venga lanciata l'eccezione NoSuchElementException().
- Descrizione: viene creato l'iteratore su sub e chiamato next() finché l'iteratore non arriva alla fine della sublist. Dopodiché viene chiamato nuovamente next.
- Pre condizioni: lista piena lst1, sublist sub piena.
- Post condizioni: lista lst1 e la sublist sub tornano alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: eccezione NextNoSuchElementException().

45. listIteratorNext()

- Riassunto: test del metodo hasNext() di ListIterator.
- Design: viene creato l'iteratore e verificato che il metodo next() funzioni correttamente.
- Descrizione: viene creato l'iteratore sulla sublist piena sub e verificato che il metodo next() restituisca gli elementi corretti nel giusto ordine.
- Pre condizioni: lista piena lst1, sublist sub piena.
- Post condizioni: lista lst1 e la sublist sub tornano alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: next() restituisce gli elementi corretti nel giusto ordine.

46. listIteratorPreviousNoSuchElementException()

- Riassunto: test dell'eccezione NoSuchElementException() del metodo previous() di ListIterator.

- Design: viene creato l'iteratore su una sublist vuota e verificato che chiamando previous() venga lanciata l'eccezione NextNoSuchElementException().
- Descrizione: dopo aver svuotato lst1, viene creato l'iteratore su sub e chiamato previous().
- Pre condizioni: lista piena lst1, sublist sub piena.
- Post condizioni: lista lst1 e la sublist sub tornano alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: eccezione NoSuchElementException().

47. listIteratorPreviousNoSuchElementException2()

- Riassunto: test dell'eccezione NoSuchElementException() del metodo previous() di ListIterator.
- Design: viene creato l'iteratore su una sublist piena e verificato che chiamando previous() all'inizio della lista, venga lanciata l'eccezione NoSuchElementException().
- Descrizione: viene creato l'iteratore su sub, viene chiamato next() finché l'iteratore non arriva alla fine della sublist. Dopodiché viene chiamato previous finché l'iteratore non torna all'inizio e infine nuovamente previous.
- Pre condizioni: lista piena lst1, sublist sub piena.
- Post condizioni: lista lst1 e la sublist sub tornano alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: eccezione NoSuchElementException().

48. listIteratorPrevious()

- Riassunto: test del metodo previous() di ListIterator.
- Design: viene creato l'iteratore e verificato che il metodo previous() funzioni correttamente.
- Descrizione: viene creato l'iteratore su sub, viene chiamato next() finché l'iteratore non arriva alla fine della sublist. Viene poi verificato che il metodo previous() restituisca gli elementi corretti nel giusto ordine.
- Pre condizioni: lista piena lst1, sublist sub piena.
- Post condizioni: lista lst1 e la sublist sub tornano alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: previous() restituisce gli elementi corretti nel giusto ordine.

49. listIteratorRemoveIllegalStateException()

- Riassunto: test dell'eccezione IllegalState() del metodo remove() di LIterator.
- Design: viene creato l'iteratore su una sublist piena e verificato che chiamando remove() subito dopo la creazione dell'iteratore, venga lanciata l'IllegalStateException().
- Descrizione: viene creato l'iteratore sulla sublist piena sub e chiamato remove() subito dopo.
- Pre condizioni: lista piena lst1, sublist sub piena.
- Post condizioni: lista lst1 e la sublist sub tornano alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: eccezione IllegalStateException().

50. listIteratorRemoveIllegalStateException2()

- Riassunto: test dell'eccezione IllegalState() del metodo remove() di ListIterator.
- Design: viene creato l'iteratore su una sublist piena e verificato che chiamando remove() due volte di fila, venga lanciata l'IllegalStateException().
- Descrizione: viene creato l'iteratore sulla sublist piena sub, chiamato next(), e poi remove() due volte consecutive.

- Pre condizioni: lista piena lst1, sublist sub piena.
- Post condizioni: lista lst1 e la sublist sub tornano alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: eccezione IllegalStateException().

51. listIteratorRemoveIllegalStateException3()

- Riassunto: test dell'eccezione IllegalState() del metodo remove() di ListIterator.
- Design: viene creato l'iteratore su una sublist piena e verificato che chiamando remove() senza aver prima eseguito un next(), venga lanciata l'IllegalStateException().
- Descrizione: viene creato l'iteratore sulla sublist piena sub, aggiunti due elementi all'iteratore e chiamato remove().
- Pre condizioni: lista piena lst1, sublist sub piena.
- Post condizioni: lista lst1 e la sublist sub tornano alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: eccezione IllegalStateException().
-

52. listIteratorRemove()

- Riassunto: test del metodo remove() di ListIterator.
- Design: viene creato l'iteratore e verificato che il metodo remove() funzioni correttamente.
- Descrizione: viene creato l'iteratore sulla sublist piena sub, viene chiamato più volte il metodo next() e verificato che il metodo remove() rimuova gli elementi corretti.
- Pre condizioni: lista piena lst1, sublist sub piena.
- Post condizioni: lista lst1 e la sublist sub tornano alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: remove() rimuove gli elementi corretti.

53. listIteratorAdd()

- Riassunto: test del metodo add() di ListIterator.
- Design: viene creato l'iteratore e verificato che il metodo add() funzioni correttamente.
- Descrizione: viene svuotata la sublist sub e viene creato l'iteratore sulla sublist vuota sub. Vengono aggiunti all'iteratore diversi elementi tramite il metodo add() e viene verificato con il metodo previous() che gli elementi siano stati aggiunti tutti e nel giusto ordine.
- Pre condizioni: lista piena lst1, sublist sub piena.
- Post condizioni: lista lst1 e la sublist sub tornano alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: add() aggiunge tutti gli elementi nell'ordine corretto.

54. listIteratorNextIndex()

- Riassunto: test del metodo nextIndex() di ListIterator.
- Design: viene creato l'iteratore e verificato che il metodo nextIndex() funzioni correttamente.
- Descrizione: viene creato l'iteratore sulla sublist piena sub. Viene verificato tramite un ciclo for che nextIndex() restituisca tutti gli indici nell'ordine corretto.
- Pre condizioni: lista piena lst1, sublist sub piena.
- Post condizioni: lista lst1 e la sublist sub tornano alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: nextIndex() restituisce tutti gli indici nell'ordine corretto.

55. listIterator PreviousIndex ()

- Riassunto: test del metodo previousIndex() di ListIterator.
- Design: viene creato l'iteratore e verificato che il metodo previousIndex() funzioni correttamente.
- Descrizione: viene creato l'iteratore sulla sublist piena sub. Viene spostato l'iteratore alla fine della sublist e verificato tramite un ciclo for che previousIndex() restituisca tutti gli indici nell'ordine corretto.
- Pre condizioni: lista piena lst1, sublist sub piena.
- Post condizioni: lista lst1 e la sublist sub tornano alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: previousIndex() restituisce tutti gli indici nell'ordine corretto.

56. listIteratorSetIllegalStateException()

- Riassunto: test dell'eccezione IllegalState() del metodo set() di LIterator.
- Design: viene creato l'iteratore su una sublist piena e verificato che chiamando set() subito dopo la creazione dell'iteratore, venga lanciata l'IllegalStateException().
- Descrizione: viene creato l'iteratore sulla sublist piena sub chiamato set() subito dopo.
- Pre condizioni: lista piena lst1, sublist sub piena.
- Post condizioni: lista lst1 e la sublist sub tornano alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: eccezione IllegalStateException().

57. listIteratorSetIllegalStateException2()

- Riassunto: test dell'eccezione IllegalState() del metodo set() di ListIterator.
- Design: viene creato l'iteratore su una sublist piena e verificato che chiamando set() dopo aver chiamato un add(), venga lanciata l'IllegalStateException().
- Descrizione: viene creato l'iteratore sulla sublist piena sub, chiamato next(), aggiunto un elemento all'iteratore e chiamato set().
- Pre condizioni: lista piena lst1, sublist sub piena.
- Post condizioni: lista lst1 e la sublist sub tornano alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: eccezione IllegalStateException().

58. listIteratorSetIllegalStateException3()

- Riassunto: test dell'eccezione IllegalState() del metodo set() di ListIterator.
- Design: viene creato l'iteratore su una sublist piena e verificato che chiamando set() senza aver prima eseguito un next() o un previous(), venga lanciata l'IllegalStateException().
- Descrizione: viene creato l'iteratore sulla sublist piena sub, aggiunto un elemento all'iteratore, ne viene rimosso uno e chiamato set().
- Pre condizioni: lista piena lst1, sublist sub piena.
- Post condizioni: lista lst1 e la sublist sub tornano alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: eccezione IllegalStateException().

59. listIteratorSet()

- Riassunto: test del metodo set() di ListIterator.
- Design: viene creato l'iteratore e verificato che il metodo set() funzioni correttamente.

- Descrizione: viene creato l'iteratore sulla sublist piena sub, vengono chiamati più volte i metodi next() e previous() e viene verificato che il metodo set() sostituisca gli elementi corretti nella posizione giusta.
- Pre condizioni: lista piena lst1, sublist sub piena.
- Post condizioni: lista lst1 e la sublist sub tornano alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: set() sostituisce gli elementi corretti nella posizione giusta.

60. listIteratorIndexOutOfBounds()

- Riassunto: test dell'eccezione IndexOutOfBounds() del metodo listIterator(int ind).
- Design: viene utilizzato il metodo listIterator(int ind) con un indice non accettabile.
- Descrizione: si prova a creare un ListIterator di sub con un indice non accettabile casuale compreso tra -100 e -1.
- Pre condizioni: lista piena lst1, sublist sub piena.
- Post condizioni: lista lst1 e la sublist sub tornano alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: eccezione IndexOutOfBounds().

61. listIteratorIndexOutOfBounds2()

- Riassunto: test dell'eccezione IndexOutOfBounds() del metodo listIterator(int ind).
- Design: viene utilizzato il metodo listIterator(int ind) con un indice non accettabile.
- Descrizione: si prova a creare un ListIterator di sub con un indice non accettabile casuale compreso tra 9 e 100 (valori maggiori della dimensione della sublist).
- Pre condizioni: lista piena lst1, sublist sub piena.
- Post condizioni: lista lst1 e la sublist sub tornano alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: eccezione IndexOutOfBounds().

62. listIteratorIndexNotNull()

- Riassunto: test del metodo listIterator(int ind) per verificare che l'oggetto ListIterator restituito non sia nullo.
- Design: viene creato l'iteratore e verificato che non sia nullo.
- Descrizione: viene creato un iteratore di sub con un indice valido con il metodo listIterator(int ind) e viene verificato che non sia nullo.
- Pre condizioni: lista piena lst1, sublist sub piena.
- Post condizioni: lista lst1 e la sublist sub tornano alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: iteratore non nullo.

63. listIteratorIndex()

- Riassunto: test del metodo listIterator(int ind).
- Design: viene creato l'iteratore tramite il metodo listIterator(int ind) con indice accettabile e viene verificato che vi siano gli elementi corretti nel giusto ordine.
- Descrizione: viene creato un listIterator di sub con il metodo listIterator(int ind), e viene verificato che contenga gli elementi corretti nel giusto ordine.
- Pre condizioni: lista piena lst1, sublist sub piena.
- Post condizioni: lista lst1 e la sublist sub tornano alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: eccezione IndexOutOfBounds().

64. removeIndexOutOfBounds()

- Riassunto: test dell'eccezione IndexOutOfBounds() del metodo remove(int index).
- Design: viene utilizzato il metodo remove(int index) con un indice non accettabile.
- Descrizione: si prova a rimuovere un elemento da sub in un indice non accettabile casuale compreso tra -100 e -1.
- Pre condizioni: lista piena lst1, sublist sub piena.
- Post condizioni: lista lst1 e la sublist sub tornano alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: eccezione IndexOutOfBounds().

65. removeIndexOutOfBounds2()

- Riassunto: test dell'eccezione IndexOutOfBounds() del metodo remove(int index).
- Design: viene utilizzato il metodo remove (int index) con un indice non accettabile.
- Descrizione: si prova a rimuovere un elemento da sub in un indice non accettabile casuale compreso tra 9 e 100 (valori maggiori della dimensione della sublist).
- Pre condizioni: lista piena lst1, sublist sub piena.
- Post condizioni: lista lst1 e la sublist sub tornano alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: eccezione IndexOutOfBounds().

66. remove(int index).

- Riassunto: test del metodo remove(int index).
- Design: viene testato il metodo remove(int index) provando a rimuovere alcuni fra gli elementi di sub tramite remove(int index).
- Descrizione: viene testato il metodo remove(int index) provando a rimuovere alcuni fra gli elementi di sub tramite remove(int index) e viene verificato che il metodo rimuova gli elementi corretti dagli indici corretti. Viene inoltre controllata la dimensione di sub.
- Pre condizioni: lista piena lst1, sublist sub piena.
- Post condizioni: lista lst1 e la sublist sub tornano alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: il metodo rimuove gli elementi corretti dagli indici corretti.

67. removeObject(Object o).

- Riassunto: test del metodo remove(Object o).
- Design: viene testato il metodo remove(Object o) provando a rimuovere alcuni fra gli elementi di sub tramite remove(Object o ex).
- Descrizione: viene testato il metodo remove(Object o) provando a rimuovere alcuni fra gli elementi di sub tramite remove(Object o) e viene verificato che il metodo rimuova gli elementi corretti dagli indici corretti. Viene inoltre controllata la dimensione di sub.
- Pre condizioni: lista piena lst1, sublist sub piena.
- Post condizioni: lista lst1 e la sublist sub tornano alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: il metodo rimuove gli elementi corretti dagli indici corretti.

68. removeAllNullPointer()

- Riassunto: test dell'eccezione NullPointer del metodo removeAll(HCollection c).

- Design: viene rimosso un elemento nullo ad una sublist con il metodo removeAll(HCollection c).
- Descrizione: viene rimosso un elemento nullo da sub con il metodo removeAll(HCollection c).
- Pre condizioni: lista piena lst1, sublist sub piena.
- Post condizioni: lista lst1 e la sublist sub tornano alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: eccezione NullPointer.

69. removeAllTrue()

- Riassunto: test del metodo removeAll(HCollection c).
- Design: viene aggiunto tutto il contenuto di una sublist in una lista vuota, dopodiché viene rimosso il contenuto della lista dalla sublist. Viene poi verificata la posizione di eventuali elementi rimanenti.
- Descrizione: viene creata una nuova lista vuota lst2 a cui vengono aggiunti tutti gli elementi di sub. A quest'ultima viene rimosso il contenuto di lst2 tramite removeAll(HCollection c). Viene poi verificata la posizione degli eventuali elementi rimanenti e della diemnsione di sub.
- Pre condizioni: lista piena lst1, sublist sub piena.
- Post condizioni: lista lst1 e la sublist sub tornano alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: removeAll(HCollection c) ha rimosso tutti e soli gli elementi di lst2 da sub.

70. removeAllFalse()

- Riassunto: test del metodo removeAll(HCollection c).
- Design: vengono rimossi gli elementi di una lista da una sublist tramite removeAll(HCollection c). Le liste non hanno elementi in comune perciò ci si aspetta che non ne vengano tolti.
- Descrizione: viene creata una nuova lista vuota lst2 a cui vengono aggiunti alcuni elementi non contenuti in sub. A quest'ultima viene rimosso il contenuto di lst2 tramite removeAll(HCollection c). Viene poi verificato che la sublist sub è rimasta immutata.
- Pre condizioni: lista piena lst1, sublist sub piena.
- Post condizioni: lista lst1 e la sublist sub tornano alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: removeAll(HCollection c) non ha rimosso alcun elemento da sub.

71. retainAllNullPointer()

- Riassunto: test dell'eccezione NullPointer del metodo retainAll(HCollection c).
- Design: viene passato un elemento nullo al metodo retainAll(HCollection c).
- Descrizione: viene passato un elemento nullo al metodo retainAll(HCollection c) su sub.
- Pre condizioni: lista piena lst1, sublist sub piena.
- Post condizioni: lista lst1 e la sublist sub tornano alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: eccezione NullPointer.

72. retainAllTrue()

- Riassunto: test del metodo retainAll(HCollection c).
- Design: vengono aggiunti alcuni degli elementi di una sublist in una lista vuota, vengono quindi conservati solo gli elementi della lista nella sublist. Viene poi verificata la posizione di eventuali elementi rimanenti nella sublist.

- Descrizione: viene creata una nuova lista vuota lst2 a cui vengono aggiunti alcuni degli elementi di sub. A quest'ultima viene applicato retainAll(HCollection c) con lst2 come parametro. Viene poi verificata la posizione degli eventuali elementi rimanenti.
- Pre condizioni: lista piena lst1, sublist sub piena.
- Post condizioni: lista lst1 e la sublist sub tornano alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: retainAll(HCollection c) ha conservato solo gli elementi di lst2 presenti in sub.

73. retainAllFalse()

- Riassunto: test del metodo retainAll(HCollection c).
- Design: vengono conservati gli elementi di una sublist presenti in una lista tramite retainAll(HCollection c). La lista contiene tutti gli elementi della sublist, perciò ci si aspetta che non ne vengano tolti.
- Descrizione: viene creata una nuova lista vuota lst2 a cui vengono aggiunti tutti gli elementi contenuti in lst1. Viene quindi applicato retainAll(HCollection c) a sub con lst2 come parametro. Viene poi verificato che la lista sub è rimasta immutata.
- Pre condizioni: lista piena lst1, sublist sub piena.
- Post condizioni: lista lst1 e la sublist sub tornano alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: retainAll(HCollection c) non ha rimosso alcun elemento da sub.

74. setIndexOutOfBounds()

- Riassunto: test dell'eccezione IndexOutOfBounds() del metodo set(int index, Object element).
- Design: viene utilizzato il metodo set(int index, Object element) con un indice non accettabile.
- Descrizione: si prova a sostituire un elemento da sub in un indice non accettabile casuale compreso tra -100 e -1.
- Pre condizioni: lista piena lst1, sublist sub piena.
- Post condizioni: lista lst1 e la sublist sub tornano alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: eccezione IndexOutOfBounds().
-

75. setIndexOutOfBounds2()

- Riassunto: test dell'eccezione IndexOutOfBounds() del metodo set(int index, Object element).
- Design: viene utilizzato il metodo set(int index, Object element)) con un indice non accettabile.
- Descrizione: si prova a sostituire un elemento da sub in un indice non accettabile casuale compreso tra 9 e 100 (valori maggiori della dimensione della sublist).
- Pre condizioni: lista piena lst1, sublist sub piena.
- Post condizioni: lista lst1 e la sublist sub tornano alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: eccezione IndexOutOfBounds().

76. set()

- Riassunto: test del metodo set(int index, Object element).
- Design: viene testato il metodo set(int index, Object element) provando a sostituire alcuni elementi di sub tramite set(int index, Object element).

- Descrizione: viene testato il metodo set(int index, Object element) provando a settare alcuni degli elementi di sub tramite set(int index, Object element) e viene verificato che il metodo sostituisca gli elementi corretti agli indici corretti.
- Pre condizioni: lista piena lst1, sublist sub piena.
- Post condizioni: lista lst1 e la sublist sub tornano alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: il metodo sostituisce gli elementi corretti agli indici corretti.

77. size()

- Riassunto: test del metodo size().
- Design: viene testato il metodo size().
- Descrizione: viene testato size() su sub con degli elementi al suo interno e dopo effettuato un clear().
- Pre condizioni: lista piena lst1, sublist sub piena.
- Post condizioni: lista lst1 e la sublist sub tornano alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: il restituisce la dimensione corretta.

78. toArray()

- Riassunto: test del metodo toArray().
- Design: viene creato un array con gli elementi di una sublist con toArray() e viene verificato che contenga gli stessi elementi della sublist nel medesimo ordine.
- Descrizione: viene creato un array con gli elementi di sub con toArray() e viene verificato che contenga gli stessi elementi di sub nel medesimo ordine.
- Pre condizioni: lista piena lst1, sublist sub piena.
- Post condizioni: lista lst1 e la sublist sub tornano alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: l'array ottenuto contiene gli stessi elementi di sub nel medesimo ordine.

79. toArrayNullPointer()

- Riassunto: test dell'eccezione NullPointer del metodo toArray(Object[] a).
- Design: viene passato un null come parametro al metodo toArray(Object[] a).
- Descrizione: viene passato un null come parametro al metodo toArray(Object[] a).
- Pre condizioni: lista piena lst1, sublist sub piena.
- Post condizioni: lista lst1 e la sublist sub tornano alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: eccezione NullPointer.

80. toArrayObjInt()

- Riassunto: test del metodo toArray(Object[] a).
- Design: viene passato un array di Int al metodo toArray(Object[] a) che agisce su una sublist di interi. Viene verificato che contenga gli stessi elementi della sublist nel medesimo ordine e che l'array sia compatibile col tipo passato.
- Descrizione: viene svuotata lst1 e viene riempita di interi, dopodiché viene nuovamente inizializzata sub con paramentri (3, 12). Viene inoltre creato un array arrInt di interi di dimensione maggiore della sublist, che viene riempito di interi. Viene quindi verificato che l'array restituito dal metodo toArray(Object[] a) su sub con arrInt come parametro, sia array dello stesso tipo di quello passato (arrInt) e che esso contenga gli stessi elementi di sub nelle medesime posizioni.
- Pre condizioni: lista piena lst1, sublist sub piena.

- Post condizioni: lista lst1 e la sublist sub tornano alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: l'array ottenuto contiene gli stessi elementi di sub nelle medesime posizioni ed è dello stesso tipo di quello passato.

81. toArrayObj()

- Riassunto: test del metodo toArray(Object[] a).
- Design: vengono confrontati un array creato con il metodo toArray() e uno creato con il metodo toArray(Object[] a).
- Descrizione: viene inoltre creato un array arrInt di interi di dimensione minore della sublist, che viene riempito di interi. Viene quindi verificato che l'array restituito dal metodo toArray(Object[] a) su sub con arrInt come parametro, restituisca un array di tipo Object[] contenente gli stessi elementi di sub.
- Pre condizioni: lista piena lst1, sublist sub piena.
- Post condizioni: lista lst1 e la sublist sub tornano alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: l'array ottenuto contiene gli stessi elementi di sub nelle medesime posizioni.