Documentazione ListAdapterTest

Questa suite ha l'obiettivo di testare tutti i metodi di ListAdapter, 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. I metodi di SubList vengono documentati in un file dedicato.
Versione JUnit utilizzata per eseguire tutti i test: junit-4.13
Componenti del frame work utilizzati: assertArrayEquals, assertEquals, assertNotEquals, assertFalse, 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}.

0. b4()

- Riassunto: @Before tramite il quale viene riempita la lista lst1, per essere poi utilizzata.
- Design: vengono aggiunti elementi alla lista elementi di vario tipo.
- 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}.
- Pre condizioni: lista vuota lst1.
- Post condizioni: lista lst1 piena.
- Risultato atteso: lista lst1 piena.

1. constructorNotNull()

- Riassunto: test del costruttore listAdapter() per verificare che l'oggetto listAdapter creato non sia nullo.
- Design: viene creata la lista e verificato che non sia nulla.
- Descrizione: viene creata una nuova lista con il costruttore di default e viene verificato che non sia nulla.
- Pre condizioni: lista piena lst1.
- Post condizioni: lista lst1 torna alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: lst2 non nulla.

2. constructorEquals()

- Riassunto: test del costruttore listAdapter() per verificare due nuove liste vuote siano uguali.
- Design: vengono create due liste vuote e viene verificato che siano uguali.
- Descrizione: vengono create due liste vuote lst2 e lst3 e viene verificato che siano uguali
- Pre condizioni: lista piena lst1.
- Post condizioni: lista lst1 torna alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: lst2 è uguale ad lst3.

3. constructor2IndexOutOfBounds()

- Riassunto: test dell'eccezione IndexOutOfBounds() del costruttore listAdapter(int initialCapacity).
- Design: viene creata una nuova lista con parametro capacità non accettabile.
- Descrizione: viene creata una nuova lista con parametro capacità non accettabile casuale compreso tra -100 e -1.
- Pre condizioni: lista piena lst1.
- Post condizioni: lista lst1 torna alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: eccezione IndexOutOfBounds().

4. constructor2NotNull()

- Riassunto: test del costruttore listAdapter(int initialCapacity) per verificare che l'oggetto listAdapter creato non sia nullo.
- Design: viene creata la lista e verificato che non sia nulla.
- Descrizione: viene creata una nuova lista con il costruttore listAdapter(int initialCapacity) e un parametro accettabile di capacità e viene verificato che non sia nulla.
- Pre condizioni: lista piena lst1.
- Post condizioni: lista lst1 torna alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: lst2 non nulla.

5. constructor3IndexOutOfBounds()

- Riassunto: test dell'eccezione IndexOutOfBounds() del costruttore listAdapter(int initialCapacity, int capacityIncrement).
- Design: viene creata una nuova lista con parametro capacità e incremento capacità non accettabili.
- Descrizione: viene creata una nuova lista con parametro capacità e incremento capacità non accettabili casuali compresi tra -100 e -1.
- Pre condizioni: lista piena lst1.
- Post condizioni: lista lst1 torna alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: eccezione IndexOutOfBounds().

6. constructor3NotNull()

- Riassunto: test del costruttore listAdapter(int initialCapacity, int capacityIncrement) per verificare che l'oggetto listAdapter creato non sia nullo.
- Design: viene creata la lista e verificato che non sia nulla.
- Descrizione: viene creata una nuova lista lst2 con il costruttore listAdapter(int initialCapacity, int capacityIncrement) e un accettabile di capacità e incremento capacità accettabili e viene verificato che non sia nulla.
- Pre condizioni: lista piena lst1.
- Post condizioni: lista lst1 torna alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: lst2 non nulla.

7. 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 null alla lista lst1 in un indice non accettabile casuale compreso tra -100 e -1.
- Pre condizioni: lista piena lst1.
- Post condizioni: lista lst1 torna alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: eccezione IndexOutOfBounds().

8. 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 null alla lista lst1 in un indice non accettabile casuale compreso tra 9 e 100 (valori maggiori della dimensione della lista).
- Pre condizioni: lista piena lst1.
- Post condizioni: lista lst1 torna alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: eccezione IndexOutOfBounds().

9. 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 lst1 in un indice accettabile, e viene controllata la posizione di tutti gli elementi presenti nella lista.
- Pre condizioni: lista piena lst1.
- Post condizioni: lista lst1 torna alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: elementi aggiunti nella posizioni previste ed elementi già presenti spostati di conseguenza.

10. addIndex2()

- Riassunto: test del metodo add(int index, Object element).
- Design: vengono aggiunti 20 elementi a due liste che vengono poi confrontate per verificare che il metodo si comporti allo stesso modo in entrambe le liste.
- Descrizione: dopo aver svuotato lst1, vengono aggiunti interi casuali compresi fra -100 e 100 (ma uguali a due a due) nelle stesse posizioni in lst1 e nella nuova lista lst2. Viene controllato se nelle due liste sono presenti gli stessi elementi nelle stesse posizioni.
- Pre condizioni: lista piena lst1.
- Post condizioni: lista lst1 torna alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: le due liste (lst1 e lst2) contengono gli stessi elementi nelle stesse posizioni.

11. add()

- Riassunto: test del metodo add().
- Design: vengono aggiunti elementi di vario tipo ad una lista vuota.

- Descrizione: dopo aver svuotato lst1, le vengono ri aggiunti i medesimi elementi e ne viene controllata la posizione.
- Pre condizioni: lista piena lst1.
- Post condizioni: lista lst1 torna alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: elementi aggiunti nelle posizioni previste.

12. add2()

- Riassunto: test del metodo add ().
- Design: vengono aggiunti elementi di vario tipo ad una lista vuota.
- Descrizione: dopo aver svuotato lst1, le vengono ri aggiunti i medesimi elementi e ne viene controllata la posizione.
- Pre condizioni: lista piena lst1.
- Post condizioni: lista lst1 torna alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: elementi aggiunti nelle posizioni previste.

13. addAllNullPointer()

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

14. addAll()

- Riassunto: test del metodo addAll(HCollection c).
- Design: viene aggiunta una HCollection ad una lista e viene verificata la corretta aggiunta degli elementi. Successivamente viene eseguita la stessa operazione su un'altra lista e le due liste vengono confrontate.
- Descrizione: vengono create due liste vuote lst2 e lst3. Vengono poi aggiunti ad lst2 degli elementi e l'HCollection lst1. Viene verificata la corretta posizione degli elementi aggiunti. Successivamente vengono aggiunti gli stessi elementi e la stessa HCollection ad lst3. Gli elementi di lst2 ed lst3 vengono poi confrontati per verificare che il metodo funzioni allo stesso modo su liste diverse.
- Pre condizioni: lista piena lst1.
- Post condizioni: lista lst1 torna alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: le due liste (lst3 ed lst2) contengono gli stessi elementi nelle stesse posizioni.

15. addAllIndexNullPointer()

- Riassunto: test dell'eccezione NullPointer del metodo addAll(int index, HCollection c).
- Design: viene aggiunta un'HCollection nulla ad una lista con il metodo addAll(int index, HCollection c).
- Descrizione: viene aggiunta un'HCollection nulla in un indice valido ad lst1 con il metodo addAll(int index, HCollection c).
- Pre condizioni: lista piena lst1.

- Post condizioni: lista lst1 torna alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: eccezione NullPointer.

16. addAllIndexOutOfBounds()

- 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 casuale compreso tra 9 e 100 (valori maggiori della dimensione della lista).
- Pre condizioni: lista piena lst1.
- Post condizioni: lista lst1 torna alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: eccezione IndexOutOfBounds().

17. 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 casuale compreso tra -100 e -1.
- Pre condizioni: lista piena lst1.
- Post condizioni: lista lst1 torna alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: eccezione IndexOutOfBounds().

18. addAllIndex()

- Riassunto: test del metodo addAll(int index, HCollection c).
- Design: viene aggiunta una HCollection ad una lista in un indice valido e viene verificata la corretta aggiunta degli elementi. Successivamente viene eseguita la stessa operazione su un'altra lista e le due liste vengono confrontate.
- Descrizione: vengono create due liste vuote lst2 e lst3. Vengono poi aggiunti ad a lst2 degli elementi e l'HCollection lst1 all'indice 2. Viene verificata la corretta posizione degli elementi aggiunti. Successivamente vengono aggiunti gli stessi elementi e la stessa HCollection all'indice 2 ad lst3. Gli elementi di lst2 ed lst3 vengono poi confrontati per verificare che il metodo funzioni allo stesso modo su liste diverse.
- Pre condizioni: lista piena lst1.
- Post condizioni: lista lst1 torna alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: le due liste (lst3 ed lst2) contengono gli stessi elementi nelle stesse posizioni.

19. clear()

- Riassunto: test del metodo clear().
- Design: viene svuotata la lista lst1 e viene verificato che la sua dimensione sia 0.
- Descrizione: viene svuotata la lista lst1 e viene verificato che la sua dimensione sia 0.
- Pre condizioni: lista piena lst1.
- Post condizioni: lista lst1 torna alla situazione iniziale (vedasi pre condizioni).

- Risultato atteso: lista lst1 vuota, quindi con dimensione uguale a 0.

20. contains()

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

21. 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.
- Post condizioni: lista lst1 torna alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: eccezione NullPointer.

22. containsAll()

- Riassunto: test del metodo containsAll(HCollection c).
- Design: viene verificata la presenza degli elementi di una HCollection in una lista con il metodo containsAll(HCollection c).
- Descrizione: viene creata la lista vuota lst2, a cui vengono aggiunti elementi non presenti in lst1, e successivamente tutti gli elementi di lst1. 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.
- Post condizioni: lista lst1 torna 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.

23. equals()

- Riassunto: test del metodo equals().
- Design: viene testato il metodo equals() confrontando la lista lst1 con liste uguali, diverse ed elementi di tipo diverso.
- Descrizione: viene creata la lista vuota lst2, a cui vengono aggiunti tutti gli elementi di lst1. Viene verificata l'uguaglianza tra lst1 ed lst2 con il metodo equals(). Successivamente viene rimosso un elemento da lst2 e viene verificato tramite equals() che le due liste non sono più uguali. Poi viene creata un'altra lista vuota lst3, che viene riempita con tutti gli elementi di lst1, ma in un ordine diverso.

Viene verificato con equals() che lst1 non è uguale ad lst3, né ad altri elementi di tipo diverso da ListAdapter.

- Pre condizioni: lista piena lst1.
- Post condizioni: lista lst1 torna alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: il metodo restituisce true solo quando le liste confrontate sono effettivamente identiche.

24. 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 casuale compreso tra -100 e -1.
- Pre condizioni: lista piena lst1.
- Post condizioni: lista lst1 torna alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: eccezione IndexOutOfBounds().

25. 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 casuale compreso tra 9 e 100 (valori maggiori della dimensione della lista).
- Pre condizioni: lista piena lst1.
- Post condizioni: lista lst1 torna alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: eccezione IndexOutOfBounds().

26. 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 lst1 tramite get(int index).
- Descrizione: viene testato il metodo get(int index) provando a prendere tutti gli elementi di lst1 tramite get(int index) e viene verificato che il metodo restituisca gli elementi corretti agli indici corretti.
- Pre condizioni: lista piena lst1.
- Post condizioni: lista lst1 torna alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: il metodo restituisce gli elementi corretti agli indici corretti.

27. hashCodeTest()

- Riassunto: test del metodo hashCode().
- Design: vengono confrontati i valori dell'hashcode restituito dall'omonimo metodo con dei valori calcolati esternamente.
- Descrizione: viene creata la lista vuota lst2. Vengono aggiunti degli elementi e man mano che vengono aggiunti, vengono confrontati i valori dell'hashcode restituito dall'omonimo metodo con dei valori calcolati esternamente.
- Pre condizioni: lista piena lst1.

- Post condizioni: lista lst1 torna alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: i valori restituiti dal metodo coincidono con quelli previsti.

28. indexOf()

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

29. isEmpty()

- Riassunto: test del metodo isEmpty().
- Design: viene verificato con il metodo isEmpty() che la lista sia vuota dopo averla svuotata e dopo aver aggiunto elementi di diverso tipo.
- Descrizione: viene creata la lista vuota lst2. Vengono aggiunti e rimossi elementi e viene verificato che la lista sia vuota o meno dopo ogni inserimento/rimozione.
- Pre condizioni: lista piena lst1.
- Post condizioni: lista lst1 torna alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: il metodo restituisce true se la lista è effettivamente vuota, e false altrimenti.

30. iteratorNotNull()

- Riassunto: test del metodo iterator() per verificare che l'oggetto HIterator restituito non sia nullo.
- Design: viene creato l'iteratore e verificato che non sia nullo.
- Descrizione: viene creato un iteratore di lst1 con il metodo iterator() e viene verificato che non sia nullo.
- Pre condizioni: lista piena lst1.
- Post condizioni: lista lst1 torna 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 lst1 con il metodo iterator() e viene verificato che hasNext() funzioni correttamente anche dopo aver aggiunto elementi o aver chiamato il metodo next().
- Pre condizioni: lista piena lst1.
- Post condizioni: lista lst1 torna 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 lista vuota e verificato che chiamando next() venga lanciata l'eccezione NoSuchElementException().
- Descrizione: dopo aver svuotato lst1, viene creato l'iteratore su di essa e chiamato next().
- Pre condizioni: lista piena lst1.
- Post condizioni: lista lst1 torna 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 lista piena e verificato che chiamando next() alla fine della lista, venga lanciata l'eccezione NoSuchElementException().
- Descrizione: viene creato l'iteratore su lst1 e chiamato next() finché l'iteratore non arriva alla fine della lista. Dopodiché viene chiamato nuovamente next.
- Pre condizioni: lista piena lst1.
- Post condizioni: lista lst1 torna 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 lista piena lst1 e verificato che il metodo next() restituisca gli elementi corretti nel giusto ordine.
- Pre condizioni: lista piena lst1.
- Post condizioni: lista lst1 torna 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 lista piena e verificato che chiamando remove() subito dopo la creazione dell'iteratore, venga lanciata l'IllegalStateException().
- Descrizione: viene creato l'iteratore sulla lista piena lst1 chiamato remove() subito dopo.
- Pre condizioni: lista piena lst1.
- Post condizioni: lista lst1 torna 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 lista piena e verificato che chiamando remove() due volte di fila, venga lanciata l'IllegalStateException().
- Descrizione: viene creato l'iteratore sulla lista piena lst1, aggiunto un elemento all'iteratore e chiamato remove() due volte consecutive.
- Pre condizioni: lista piena lst1.
- Post condizioni: lista lst1 torna 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 lista piena lst1, viene chiamato più volte il metodo next() e verificato che il metodo remove() rimuova gli elementi corretti. Viene inoltre verificata la dimensione di lst1.
- Pre condizioni: lista piena lst1.
- Post condizioni: lista lst1 torna 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 lst1 tramite lastIndexOf (). Il metodo viene inoltre testato con un elemento non appartenente a lst1.
- Descrizione: viene testato il metodo lastIndexOf () richiedendo l'indice di tutti gli elementi di lst1 tramite lastIndexOf () e controllando che il metodo restituisca gli indici corretti degli elementi presenti nella lista. Il metodo viene inoltre testato con un elemento non appartenente a lst1.
- Pre condizioni: lista piena lst1.
- Post condizioni: lista lst1 torna alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: il metodo restituisce gli indici corretti degli elementi presenti nella lista, e -1 per gli elementi non presenti nella lista.

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 lst1 con il metodo listIterator() e viene verificato che non sia nullo.
- Pre condizioni: lista piena lst1.
- Post condizioni: lista lst1 torna 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 lst1 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.
- Post condizioni: lista lst1 torna 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 lst1 con il metodo listIterator() e viene verificato che hasPrevious() funzioni correttamente anche dopo aver aggiunto elementi o aver chiamato il metodo next() o remove().
- Pre condizioni: lista piena lst1.
- Post condizioni: lista lst1 torna 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 lista vuota e verificato che chiamando next() venga lanciata l'eccezione NextNoSuchElementException().
- Descrizione: dopo aver svuotato lst1, viene creato l'iteratore su di essa e chiamato next().
- Pre condizioni: lista piena lst1.
- Post condizioni: lista lst1 torna 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 lista piena e verificato che chiamando next() alla fine della lista, venga lanciata l'eccezione NoSuchElementException().
- Descrizione: viene creato l'iteratore su lst1 e chiamato next() finché l'iteratore non arriva alla fine della lista. Dopodiché viene chiamato nuovamente next.
- Pre condizioni: lista piena lst1.
- Post condizioni: lista lst1 torna 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 lista piena lst1 e verificato che il metodo next() restituisca gli elementi corretti nel giusto ordine.
- Pre condizioni: lista piena lst1.
- Post condizioni: lista lst1 torna 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 lista vuota e verificato che chiamando previous() venga lanciata l'eccezione NextNoSuchElementException().
- Descrizione: dopo aver svuotato lst1, viene creato l'iteratore su di essa e chiamato previous().
- Pre condizioni: lista piena lst1.
- Post condizioni: lista lst1 torna 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 lista piena e verificato che chiamando previous() all'inizio della lista, venga lanciata l'eccezione NoSuchElementException().
- Descrizione: viene creato l'iteratore su lst1, viene chiamato next() finché l'iteratore non arriva alla fine della lista. Dopodiché viene chiamato previous finché l'iteratore non torna all'inizio e infine nuovamente previous.
- Pre condizioni: lista piena lst1.
- Post condizioni: lista lst1 torna 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 lst1, viene chiamato next() finché l'iteratore non arriva alla fine della lista. Viene poi verificato che il metodo previous() restituisca gli elementi corretti nel giusto ordine.
- Pre condizioni: lista piena lst1.
- Post condizioni: lista lst1 torna 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 lista piena e verificato che chiamando remove() subito dopo la creazione dell'iteratore, venga lanciata l'IllegalStateException().
- Descrizione: viene creato l'iteratore sulla lista piena lst1 e chiamato remove() subito dopo.
- Pre condizioni: lista piena lst1.
- Post condizioni: lista lst1 torna 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 lista piena e verificato che chiamando remove() due volte di fila, venga lanciata l'IllegalStateException().
- Descrizione: viene creato l'iteratore sulla lista piena lst1, chiamato next(), e poi remove() due volte consecutive.
- Pre condizioni: lista piena lst1.
- Post condizioni: lista lst1 torna 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 lista piena e verificato che chiamando remove() senza aver prima eseguito un next(), venga lanciata l'IllegalStateException().
- Descrizione: viene creato l'iteratore sulla lista piena lst1, aggiunti due elementi all'iteratore e chiamato remove().
- Pre condizioni: lista piena lst1.
- Post condizioni: lista lst1 torna 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 lista piena lst1, viene chiamato più volte il metodo next() e verificato che il metodo remove() rimuova gli elementi corretti. Viene inoltre verificata la dimensione di lst1.
- Pre condizioni: lista piena lst1.
- Post condizioni: lista lst1 torna 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 lista lst1 e viene creato l'iteratore sulla lista vuota lst1. 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.
- Post condizioni: lista lst1 torna 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 lista piena lst1. Viene verificato tramite un ciclo for che nextIndex() restituisca tutti gli indici nell'ordine corretto.
- Pre condizioni: lista piena lst1.
- Post condizioni: lista lst1 torna 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 lista piena lst1. Viene spostato l'iteratore alla fine della lista e verificato tramite un ciclo for che previousIndex() restituisca tutti gli indici nell'ordine corretto.
- Pre condizioni: lista piena lst1.
- Post condizioni: lista lst1 torna 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 lista piena e verificato che chiamando set() subito dopo la creazione dell'iteratore, venga lanciata l'IllegalStateException().
- Descrizione: viene creato l'iteratore sulla lista piena lst1 chiamato set() subito dopo.
- Pre condizioni: lista piena lst1.
- Post condizioni: lista lst1 torna 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 lista piena e verificato che chiamando set() dopo aver chiamato un add(), venga lanciata l'IllegalStateException().
- Descrizione: viene creato l'iteratore sulla lista piena lst1, chiamato next(), aggiunto un elemento all'iteratore e chiamato set().Pre condizioni: lista piena lst1.
- Post condizioni: lista lst1 torna 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 lista 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 lista piena lst1, aggiunto un elemento all'iteratore, ne viene rimosso uno e chiamato set().
- Pre condizioni: lista piena lst1.
- Post condizioni: lista lst1 torna 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 lista piena lst1, 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.
- Post condizioni: lista lst1 torna 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 lst1 con un indice non accettabile casuale compreso tra -100 e -1.
- Pre condizioni: lista piena lst1.
- Post condizioni: lista lst1 torna 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 lst1 con un indice non accettabile casuale compreso tra 9 e 100 (valori maggiori della dimensione della lista).
- Pre condizioni: lista piena lst1.
- Post condizioni: lista lst1 torna 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 lst1 con un indice valido con il metodo listIterator() e viene verificato che non sia nullo.
- Pre condizioni: lista piena lst1.
- Post condizioni: lista lst1 torna 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 lst1 con il metodo listIterator(int ind), e viene verificato che contenga gli elementi corretti nel giusto ordine.
- Pre condizioni: lista piena lst1.
- Post condizioni: lista lst1 torna 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 lst1 in un indice non accettabile casuale compreso tra -100 e -1.
- Pre condizioni: lista piena lst1.
- Post condizioni: lista lst1 torna 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 lst1 in un indice non accettabile casuale compreso tra 9 e 100 (valori maggiori della dimensione della lista).
- Pre condizioni: lista piena lst1.
- Post condizioni: lista lst1 torna 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 lst1 tramite remove(int index).
- Descrizione: viene testato il metodo remove(int index) provando a rimuovere alcuni fra gli elementi di lst1 tramite remove(int index) e viene verificato che il metodo rimuova gli elementi corretti dagli indici corretti. Viene inoltre controllata la dimensione di lst1.
- Pre condizioni: lista piena lst1.
- Post condizioni: lista lst1 torna 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 lst1 tramite remove(Object o ex).
- Descrizione: viene testato il metodo remove(Object o) provando a rimuovere alcuni fra gli elementi di lst1 tramite remove(Object o) e viene verificato che il metodo rimuova gli elementi corretti dagli indici corretti. Viene inoltre controllata la dimensione di lst1.
- Pre condizioni: lista piena lst1.

- Post condizioni: lista lst1 torna 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 lista con il metodo removeAll(HCollection c).
- Descrizione: viene rimosso un elemento nullo da lst1 con il metodo removeAll(HCollection c).
- Pre condizioni: lista piena lst1.
- Post condizioni: lista lst1 torna 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 lista in una lista vuota, in una delle due vengono aggiunti altri elementi e viene rimossa la lista di lunghezza minore da quella con lunghezza maggiore. 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 lst1. A quest'ultima vengono aggiunti altri elementi e ne viene rimosso il contenuto di lst2 tramite removeAll(HCollection c). Viene poi verificata la posizione degli eventuali elementi rimanenti.
- Pre condizioni: lista piena lst1.
- Post condizioni: lista lst1 torna alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: removeAll(HCollection c) ha rimosso tutti e soli gli elementi di lst2 da lst1.

70. removeAllFalse()

- Riassunto: test del metodo removeAll(HCollection c).
- Design: vengono rimossi gli elementi di una lista da un'altra 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 lst1. A quest'ultima viene rimosso il contenuto di lst2 tramite removeAll(HCollection c). Viene poi verificato che la lista lst1 è rimasta immutata.
- Pre condizioni: lista piena lst1.
- Post condizioni: lista lst1 torna alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: removeAll(HCollection c) non ha rimosso alcun elemento da lst1.

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 lst1.
- Pre condizioni: lista piena lst1.
- Post condizioni: lista lst1 torna alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: eccezione NullPointer.

72. retainAllTrue()

- Riassunto: test del metodo retainAll(HCollection c).
- Design: viene aggiunto tutto il contenuto di una lista in una lista vuota, in una delle due vengono aggiunti altri elementi e vengono conservati gli elementi della lista di lunghezza maggiore presenti anche in quella con lunghezza minore. Viene poi verificata la posizione di eventuali elementi rimanenti.
- Descrizione: dopo aver svuotato lst1 e avervi aggiunto degli elementi di vario tipo, viene creata una nuova lista vuota lst2 a cui vengono aggiunti tutti gli elementi di lst1. A quest'ultima vengono aggiunti altri elementi e viene applicato retainAll(HCollection c) con lst2 come parametro. Viene poi verificata la posizione degli eventuali elementi rimanenti.
- Pre condizioni: lista piena lst1.
- Post condizioni: lista lst1 torna alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: retainAll(HCollection c) ha conservato solo gli elementi di lst2 presenti in lst1.

73. retainAllFalse()

- Riassunto: test del metodo retainAll(HCollection c).
- Design: vengono conservati gli elementi di una lista presenti in un'altra tramite retainAll(HCollection c). Le liste hanno tutti gli elementi in comune 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. A quest'ultima viene applicato retainAll(HCollection c) con lst2 come parametro. Viene poi verificato che la lista lst1 è rimasta immutata.
- Pre condizioni: lista piena lst1.
- Post condizioni: lista lst1 torna alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: retainAll(HCollection c) non ha rimosso alcun elemento da lst1.

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 lst1 in un indice non accettabile casuale compreso tra -100 e -1.
- Pre condizioni: lista piena lst1.
- Post condizioni: lista lst1 torna 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 lst1 in un indice non accettabile casuale compreso tra 9 e 100 (valori maggiori della dimensione della lista).
- Pre condizioni: lista piena lst1.
- Post condizioni: lista lst1 torna 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 lst1 tramite set(int index, Object element).
- Descrizione: viene testato il metodo set(int index, Object element) provando a settare alcuni degli elementi di lst1 tramite set(int index, Object element) e viene verificato che il metodo sostituisca gli elementi corretti agli indici corretti.
- Pre condizioni: lista piena lst1.
- Post condizioni: lista lst1 torna 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 lst1 con degli elementi al suo interno e dopo effettuato un clear().
- Pre condizioni: lista piena lst1.
- Post condizioni: lista lst1 torna 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 lista con toArray() e viene verificato che contenga gli stessi elementi della lista nel medesimo ordine.
- Descrizione: viene creato un array con gli elementi di lst1 con toArray() e viene verificato che contenga gli stessi elementi di lst1 nel medesimo ordine.
- Pre condizioni: lista piena lst1.
- Post condizioni: lista lst1 torna alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: l'array ottenuto contiene gli stessi elementi di lst1 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.
- Post condizioni: lista lst1 torna 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 lista di interi. Viene verificato che contenga gli stessi elementi della lista nel medesimo ordine e che l'array sia compatibile col tipo passato.
- Descrizione: viene svuotata lst1 e viene riempita di interi. Viene inoltre creato un array arrInt di interi di dimensione maggiore della lista, che viene riempito di interi. Viene quindi verificato che l'array

restituito dal metodo toArray(Object[] a) su lst1 con arrInt come parametro, sia dello stesso tipo di quello passato (arrInt) e che esso contenga gli stessi elementi di list1 nelle medesime posizioni.

- Pre condizioni: lista piena lst1.
- Post condizioni: lista lst1 torna alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: l'array ottenuto contiene gli stessi elementi di lst1 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 lista, che viene riempito di interi. Viene quindi verificato che l'array restituito dal metodo toArray(Object[] a) su lst1 con arrInt come parametro, restituisca un array di tipo Object[] contenente gli stessi elementi di lst1.
- Pre condizioni: lista piena lst1.
- Post condizioni: lista lst1 torna alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: l'array ottenuto contiene gli stessi elementi di lst1 nelle medesime posizioni.