Visentin Francesco matricola 1216419

Test suite della classe ListAdapter.java

La seguente test suite è stata ideata per testare i metodi della classe ListAdapter.java e si riferisce ai test presenti all'interno delle due test classes ListTester.java ed SubListTester.java.

Le due classi ListTester ed SubListTester contengono gli stessi test ma differiscono per l'oggetto su cui i vari test vengono eseguiti:

- la classe ListTester ha a disposizione un oggetto di tipo ListAdapter configurato per lavorare come una "lista di base."
- La classe SubListTester lavora invece su un oggetto ListAdapter configurato come "sublist".

Il ripetersi dei test, per ListAdapter configurati in entrambe le due "possibili modalità" (vedi nota sui metodi 53-64 a pagina nuova), vuole andare a verificare la totale trasparenza tra queste ad un utilizzatore esterno e garantire il corretto funzionamento sia in una modalità che nell'altra.

La suite è formata da un totale di 64 test, ripetuti due volte, descritti nelle pagine a seguire, ed riportati secondo l'ordine con cui questi appaiono all'interno delle interfacce della API di J2SE 1.42 che la classe ListAdapter.java va ad implementare.

Tutti i metodi pubblici, e tutte le funzionalità esposte della classe, sono state testati.

Per i metodi che possono lanciare eccezioni sono stati solitamente definiti due test distinti: un test in cui si verifica il corretto funzionamento del metodo quando questo viene usato come previsto dall'interfaccia, un test in cui si verifica che le eccezioni vengano lanciate e gestite in modo adeguato quando il metodo viene invocato in modo improprio.

Ogni test ha a disposizione un oggetto di tipo ListAdapter correttamente inizializzato (in base alla modalità richiesta) ed inizialmente vuoto. Tutti gli altri oggetti necessari per effettuare il test vengono creati all'interno di esso. Per favorire questo processo all'interno delle due classi è stato messo a disposizione il metodo makeCollection() che restituisce una HCollection di otto elementi, tutti diversi da null, di cui: i primi quattro di tipo "String", i secondi quattro di tipo "int".

Ogni test case è focalizzato sul verificare il corretto comportamento di un solo metodo, all'interno di questo si è perciò tentato di rendere minime (per quanto possibile) le dipendenze e le interconnessioni con altri metodi. Queste interconnessioni sono però spesso inevitabili per cui i vari metodi, diversi da quello in esame, presenti all'interno di un test case vengono considerati correttamente funzionanti come precondizione.

Nota sui test 53-64:

Come sopra citato e come descritto nella propria documentazione, la classe ListAdapter.java è stata implementata per poter essere utilizzata in “due modalità” stabilite in fase di costruzione:

- Una modalità “lista base” in cui può lavorare su tutti i valori contenuti al suo interno.
- Una modalità “sublist” in cui l’oggetto ListAdapter è legato ad una seconda lista ed il range di utilizzo è limitato ad degli indici definiti in fase di costruzione.

Alcuni metodi sono quindi stati progettati per poter lavorare in maniera leggermente diversa in base alla modalità in cui l’oggetto ListAdapter è stato creato. Il corretto funzionamento di questi metodi in entrambe le modalità è garantito dalla ridondanza dei test effettuati tra le classi ListTester.java ed SubListTester.java

I test da 53 a 64 sono quindi focalizzati, oltre che nel garantire ancora più nel dettaglio il corretto funzionamento dei metodi che presentano “doppia modalità”, nel verificare la corretta implementazione del backing tra liste e che le modifiche strutturali applicate alla sublist vengano correttamente trasferite anche nella lista di base.

Nota: in fase di testing è stata utilizzata la versione di jUnit presentata a lezione.

1° AddIndexTest():

- Test del metodo public void add(int index, Object o)
- **Summary / Description**: vengono inseriti quattro elementi, diversi da null, in varie posizioni valide della lista. Il primo inserimento testa il corretto inserimento su una lista vuota indicando l'indice zero. I successivi inserimenti verificano che gli elementi della lista vengano opportunamente spostati quando viene inserito un nuovo elemento prima di essi. Mentre i vari elementi vengono inseriti si verifica che la dimensione della lista venga aumentata congruentemente e che gli oggetti inseriti occupino effettivamente la posizione richiesta
- **Design**: verifica che l'inserimento di un elemento, diverso da null, all'interno della lista avvenga correttamente modificando correttamente gli indici di eventuali elementi già presenti nella lista.
- **Pre-Condition**: un oggetto ListAdapter vuoto e correttamente inizializzato. Parametro "o" inizializzato con valore diverso da null. Parametro "index" appartenente al range [0, list.size()]
- **Post-Condition**: la lista dovrà contenere gli elementi, inseriti correttamente, nessuna eccezione dovrà essere generata.
- **Expected results**: una lista contenente quattro elementi, nell'ordine desiderato e senza il lancio di nessuna eccezione.

2° AddIndexExceptionsTest():

- Test del metodo public void add(int index. Object o)
- **Summary / Description**: entrambe le eccezioni vengono inizialmente testate su una lista vuota e si verifica che:
    - null su una lista vuota ad un indirizzo valido genera NullPointerException.
    - un elemento valido su una lista vuota ad un indice non valido genera IndexOutOfBoundException. (testato con più indici)
    - null su una lista vuota ad un indirizzo invalido genera IndexOutOfBoundException.

  Si verifica che la dimensione della lista non è stata modificata (non è stato effettivamente inserito nessun elemento).

  Viene poi aggiunto un elemento valido alla lista e si ripetono i test in maniera analoga a prima, ma su una lista non vuota, verificando che l'elemento inserito in precedenza non subisca modifiche.

- **Design**: verifica che:
    - il metodo lanci l'eccezione NullPointerException qualora si tentasse di inserire un oggetto inizializzato a null nella lista, come specificato dall'interfaccia. Questo deve verificarsi sia quando la lista è vuota sia quando questa contiene già altri elementi.
    - Il metodo lanci l'eccezione IndexOutOfBoundException qualora si tentasse di inserire un elemento al di fuori del range [0, list.size()].
- **Pre-Condition**: un oggetto ListAdapter vuoto e correttamente inizializzato.
- **Post-Condition**: il lancio delle eccezioni non modifica il contenuto della lista. Dimensione ed eventuali valori già presenti rimangono inalterati.
- **Expected results**: il lancio delle eccezioni NullPointerException ed IndexOutOfBoundException ed l'inserimento corretto di un elemento nella lista. Nessun altra eccezione deve essere generata.

3° AddTest():

- Test del metodo public boolean add(Object o)
- **Summary / Description**: tre oggetti vengono aggiunti alla lista e si verifica che:
    - gli oggetti vengano correttamente inseriti in coda alla lista con indice e valore corretto.
    - la dimensione della lista aumenti di tre.
    - il temine dell'operazione gli oggetti sia effettivamente contenuti all'interno della lista.
- **Design**: si effettuano più inserimenti verificando che dopo ciascuno modifichi congruentemente la dimensione della lista. Si verifica inoltre che i vari oggetti siano stati correttamente inseriti all'indice corretto e con il valore desiderato.
- **Pre-Condition**: un oggetto ListAdapter vuoto e correttamente inizializzato. Parametro "o" inizializzato con valore diverso da null.
- **Post-Condition**: la lista dovrà contenere i tre elementi, inseriti correttamente, nessuna eccezione dovrà essere generata.
- **Expected results**: dimensione della lista pari a tre, i tre elementi sono stati correttamente inseriti, ognuno al previsto indice ed inizializzati al corretto valore. Il metodo ritorna "true" ad ogni chiamata in quanto modifica la dimensione della lista.

4° AddExceptionsTest():

- Test del metodo public boolean add(Object o)
- **Summary / Description**: viene inizialmente verificato il corretto lancio dell'eccezione quando si inserisce null su una lista vuota, viene poi inserito un elemento valido e a seguire un secondo elemento null che genera nuovamente un eccezione. Si verifica che l'elemento valido inserito precedentemente non è stato modificato ed è ancora presente all'indice corretto.
- **Design**: verifica che il metodo lanci l'eccezione NullPointerException qualora si tentasse di inserire un oggetto inizializzato a null in coda alla lista, come specificato dall'interfaccia. Questo deve verificarsi sia quando la lista è vuota sia quando questa contiene già altri elementi.
- **Pre-Condition**: un oggetto ListAdapter vuoto e correttamente inizializzato.
- **Post-Condition**: il lancio dell'eccezione non modifica il contenuto della lista. Dimensione ed eventuali valori già presenti rimangono inalterati.
- **Expected results**: il lancio di due eccezioni NullPointerException ed l'inserimento corretto di un elemento nella lista. Nessun altra eccezione deve essere generata.

5° AddAllTest():

- Test del metodo public boolean addAll(HCollection c)
- **Summary / Description**: viene aggiunta alla lista, inizialmente vuota, una HCollection di 8 oggetti, tutti diversi da null. Si verifica che questi elementi vengano correttamente inseriti in coda alla lista e che la dimensione della lista sia aggiornata congruentemente. Viene poi ripetuto lo stesso processo partendo però questa volta da una lista non vuota. Viene inoltre verificato il corretto comportamento quando si aggiunge una HCollection vuota.
- **Design**: verifica che l'aggiunta degli elementi della HCollection in fondo alla lista avvenga correttamente senza modificare gli elementi ed inserendoli congruentemente all'ordine con cui questi sono restituiti dall'iteratore della collection. Questo deve essere verificato sia se la lista è vuota, sia se contiene già altri elementi.
- **Pre-Condition**: un oggetto ListAdapter vuoto e correttamente inizializzato. Parametro "c", istanza di classe che implementa HCollection, inizializzato con valore diverso da null.
- **Post-Condition**: la lista dovrà contenere gli elementi della HCollection, inseriti correttamente (primi otto elementi ed gli ultimi otto), nessuna eccezione dovrà essere generata.
- **Expected results**: elementi correttamente inseriti nella lista, ognuno al previsto indice ed inizializzati al corretto valore. La dimensione della lista è stata aggiornata adeguatamente. L'aggiunta di una HColletion vuota non comporta nessuna modifica alla lista.

6° AddAllExceptionTest():

- Test del metodo public boolean addAll(HCollection c)
- **Summary / Description**: viene inizialmente verificato il corretto lancio dell'eccezione quando si inserisce null su una lista vuota, viene poi inserito un elemento valido e a seguire un secondo elemento null che genera nuovamente un eccezione. Si verifica che l'elemento valido inserito precedentemente non è stato modificato ed è ancora presente all'indice corretto.
- **Design**: verifica che il metodo lanci l'eccezione NullPointerException qualora si tentasse di inserire un oggetto inizializzato a null in coda alla lista, come specificato dall'interfaccia. Questo deve verificarsi sia quando la lista è vuota sia quando questa contiene già altri elementi.
- **Pre-Condition**: un oggetto ListAdapter vuoto e correttamente inizializzato.
- **Post-Condition**: il lancio dell'eccezione non modifica il contenuto della lista. Dimensione ed eventuali valori già presenti rimangono inalterati.
- **Expected results**: il lancio di due eccezioni NullPointerException ed l'inserimento corretto di un elemento nella lista. Nessun altra eccezione deve essere generata.

7° AddAllIndexTest():

- Test del metodo public boolean addAll(int index, HCollection c)
- **Summary / Description**: viene aggiunta alla lista, inizialmente vuota, una HCollection di 8 oggetti, tutti diversi da null. Si verifica che questi elementi vengano correttamente inseriti in coda alla lista e che la dimensione della lista sia aggiornata congruentemente. Viene poi ripetuto lo stesso processo aggiungendo però gli elementi a partire dall'indice 3. Viene inoltre verificato il corretto comportamento quando si aggiunge una HCollection vuota nel mezzo ed in coda alla lista.
- **Design**: verifica che l'aggiunta degli elementi della HCollection, all'indice indicato della lista, avvenga correttamente senza modificare gli elementi ed inserendoli congruentemente all'ordine con cui questi sono restituiti dall'iteratore della collection e spostando eventuali elementi già presenti. Questo deve essere verificato sia se la lista è vuota, sia se contiene già altri elementi.
- **Pre-Condition**: un oggetto ListAdapter vuoto e correttamente inizializzato. Parametro "c", istanza di classe che implementa HCollection, inizializzato con valore diverso da null. Parametro "index" appartenente al range [0,list.size()].
- **Post-Condition**: la lista dovrà contenere gli elementi della HCollection, inseriti correttamente (relativamente agli indici specificati), nessuna eccezione dovrà essere generata.
- **Expected results**: elementi correttamente inseriti nella lista, ognuno al previsto indice ed inizializzati al corretto valore. La dimensione della lista è stata aggiornata adeguatamente. L'aggiunta di una HColletion vuota non comporta nessuna modifica alla lista.

8° AddAllIndexExceptionsTest():

- Test del metodo public boolean addAll(int index, HCollection c)
- **Summary / Description**: entrambe le eccezioni vengono inizialmente testate su una lista vuota e si verifica che:
    - null su una lista vuota ad un indirizzo valido genera NullPointerException.
    - un elemento valido su una lista vuota ad un indice non valido genera IndexOutOfBoundException. (testato con più indici)
    - null su una lista vuota ad un indirizzo invalido genera IndexOutOfBoundException.

  Si verifica che la dimensione della lista non è stata modificata (non è stato effettivamente inserito nessun elemento).

  Viene poi aggiunto un elemento valido alla lista e si ripetono i test in maniera analoga a prima, ma su una lista non vuota, verificando che l'elemento inserito in precedenza non subisca modifiche.

- **Design**: verifica che:
    - il metodo lanci l'eccezione NullPointerException qualora si tentasse di inserire un oggetto inizializzato a null nella lista, come specificato dall'interfaccia. Questo deve verificarsi sia quando la lista è vuota sia quando questa contiene già altri elementi.
    - Il metodo lanci l'eccezione IndexOutOfBoundException qualora si tentasse di inserire un elemento al di fuori del range [0, list.size()].
- **Pre-Condition**: un oggetto ListAdapter vuoto e correttamente inizializzato.
- **Post-Condition**: il lancio delle eccezioni non modifica il contenuto della lista. Dimensione ed eventuali valori già presenti rimangono inalterati.
- **Expected results**: il lancio delle eccezioni NullPointerException ed IndexOutOfBoundException ed l'inserimento corretto di un elemento nella lista. Nessun altra eccezione deve essere generata.

9° ClearTest():

- Test del metodo public void clear()
- **Summary / Description**: il metodo viene inizialmente chiamato su una lista vuota, aspettandosi che non porti a nessuna modifica, la lista viene poi riempita ed il metodo chiamato nuovamente per svuotare la lista.
- **Design**: verifica che tutti gli elementi della lista vengano effettivamente eliminati e che la dimensione della lista venga impostata a zero.
- **Pre-Condition**: un oggetto ListAdapter vuoto inizializzato correttamente.
- **Post-Condition**: la lista dovrà tornare ad essere vuota, nessuna eccezione dovrà essere generata.
- **Expected results**: una lista vuota con dimensione uguale a zero.

10° ContainsTest():

- Test del metodo public boolean contains(Object o)
- **Summary / Description**: si testa inizialmente il corretto funzionamento su una lista vuota, vengono poi aggiunti degli elementi alla lista e si verifica che questi vengano effettivamente "individuati" dal metodo contains. Questi elementi vengono poi eliminati dalla lista e si verifica che ora non sia più visibili al metodo contains.
- **Design**: verifica che il metodo ritorni:
    - "true" quando si passa come parametro un oggetto effettivamente contenuto nella lista. Questo deve essere vero a prescindere dalla posizione che questo occupa nella lista.
    - "false" altrimenti.

  Il metodo deve lavorare correttamente anche su una lista vuota (ritorna sempre false).

- **Pre-Condition**: un oggetto ListAdapter vuoto correttamente iniziallizato, un parametro "o" diverso da null.
- **Post-Condition**: la lista torna ad essere vuota ed il metodo contains continuerà a ritornare false ad ogni invocazione. Nessun eccezione deve essere generata.
- **Expected results**: il metodo ritorna il valore true se e solo se l'oggetto è presente nella lista.

11° ContainsExceptionsTest():

- Test del metodo public boolean contains(Object o)
- **Summary / Description**: si effettua prima un test su una lista vuota poi su una lista contenente altri elementi.
- **Design**: verifica che il metodo lanci NullPointerException qualora si passasse come parametro il valore null.
- **Pre-Condition**: un oggetto ListAdapter vuoto e correttamente inizializzato.
- **Post-Condition**: il lancio delle eccezioni non modifica il contenuto della lista. Dimensione ed eventuali valori già presenti rimangono inalterati.
- **Expected results**: il lancio dell' eccezione NullPointerException ed l'inserimento corretto di un elemento nella lista. Nessun altra eccezione deve essere generata.

12° ContainsAllTest():

- Test del metodo public boolean containsAll(HCollection c)
- **Summary / Description**: si verifica prima che il metodo funzioni correttamente:
    - su una HCollection vuota.
    - su una HCollection di elementi che sono stati aggiunti alla lista.
    - sulla stessa HCollection ma dopo aver rimosso alcuni dei sui valori valori dalla lista.
    - aggiungendo nuovamente la stessa collection alla lista (verifica il funzionamento a prescindere dall'ordine degli elementi all'interno della lista).
- **Design**: verifica che il metodo ritorni:
    - "true" quando tutti gli oggetti della HCollection passata come parametro sono effettivamente contenuti nella lista. Questo deve essere vero a prescindere dalla posizione che questi occupa nella lista.
    - "false" altrimenti.

  Il metodo deve lavorare correttamente anche su una lista vuota (ritorna sempre false).

- **Pre-Condition**: un oggetto ListAdapter vuoto correttamente inizializzato, una HCollection c correttamente inizializzata con valore diverso da null.
- **Post-Condition**: la lista torna ad essere vuota ed il metodo containsAll continuerà a ritornare false ad ogni invocazione. Nessun eccezione deve essere generata.
- **Expected results**: il metodo ritorna il valore true se e solo se tutti gli oggetti della HCollection sono presenti nella lista (questo è vero anche quando la HCollection è vuota).

13° ContainsAllExceptionTest():

- Test del metodo public boolean containsAll(HCollection c)
- **Summary / Description**: si effettua prima un test su una lista vuota poi su una lista contenente altri elementi.
- **Design**: verifica che il metodo lanci NullPointerException qualora si passasse come parametro una HCollection c pari al valore null.
- **Pre-Condition**: un oggetto ListAdapter vuoto e correttamente inizializzato.
- **Post-Condition**: il lancio delle eccezioni non modifica il contenuto della lista. Dimensione ed eventuali valori già presenti rimangono inalterati.
- **Expected results**: il lancio dell' eccezione NullPointerException ed l'inserimento corretto di un elemento nella lista. Nessun altra eccezione deve essere generata.

14° EqualsTest():

- Test del metodo public boolean equals(Object o)
- **Summary / Description**: verifica il corretto funzionamento del metodo equals:
    - quando viene passato come parametro null.
    - quando viene passato come parametro una ListAdapter vuota.
    - quando viene passato un oggetto di una classe diversa.
    - quando si confrontano due ListAdapter effettivamente uguali.
    - Quando si confrontano due ListAdapter avente gli stessi elementi ma in ordine diverso.
- **Design**: verifica il funzionamento del metodo tentando di confronto fra oggetti diversi e non.
- **Pre-Condition**: un oggetto ListAdapter correttamente inizializzato.
- **Post-Condition**: la lista conterrà gli elementi inseriti dal metodo (corrispondenti a quelli della HCollection inserita) senza nessuna modifica a questi e senza generare nessuna eccezione.
- **Expected results**: restituisce true se e solo se il parametro “o” è anch’esso un oggetto ListAdapter contenente gli stessi oggetti, nello stesso ordine, della mia lista.

15° GetTest():

- Test del metodo public Object get(int index)
- **Summary / Description**: viene inserita una HCollection nella lista. Viene invocato il metodo get su più indici e si verifica che il valore restituito corrisponda con al valore atteso. Viene poi eliminato un elemento dalla lista e si verifica che l'elemento prima ispezionato occupa ora l'indice precedente.
- **Design**: verifica la correttezza del metodo tramite accessi ad indici diversi, modificando fra questi la struttura della lista.
- **Pre-Condition**: un oggetto ListAdapter correttamente inizializzato. Un indice appartenente al range [0,list.size()-1].
- **Post-Condition**: la lista contiene gli elementi della HCollection, accessibili congruentemente al loro indice.
- **Expected results**: il valore restituito dal metodo è uguale, per ogni indice, all'elemento corrispondente nella lista di riferimento.

16° GetExceptionsTest():

- Test del metodo public Object get(int index)
- **Summary / Description**: si verifica il lancio dell'eccezione effettuano:
    - un accesso su una lista vuota
    - un accesso ad un indice negativo
    - un accesso fuori range della lista
- **Design**: si genera IndexOutOfBoundException tentando di accedere ad indici non validi.
- **Pre-Condition**: un oggetto ListAdapter correttamente inizializzato.
- **Post-Condition**:il lancio delle eccezioni non modifica il contenuto della lista. Dimensione ed eventuali valori già presenti rimangono inalterati.
- **Expected results**: il lancio dell' eccezione IndexOutOfBoundException ed l'inserimento corretto di un elemento nella lista. Nessun altra eccezione deve essere generata.

17° HashCodeTest():

- Test del metodo public int hashCode()
- **Summary / Description**: alla lista vengono aggiunti gli elementi di una HColletion, gli stessi elementi vengono aggiunti anche ad un secondo oggetto ListAdapter. Si verifica che i due hashCode sono uguali. Viene poi modificata la seconda lista e si verifica i due hashCode sono diversi. Verificato il funzionamento anche su liste vuote.
- **Design**: verifico che il metodo hashCode rispetti funzioni sempre e che rispetti la condizione: list1.equals(list2) => list1.hashCode() == list2.hashCode().
- **Pre-Condition**: una oggetto ListAdapter inizializzato correttamente.
- **Post-Condition**: il metodo hashCode non modifica la struttura della lista. Nessuna eccezione viene generata.
- **Expected results**: il metodo hashCode funziona sempre e rispetta la condizione posta dal metodo equals.

18° IndexOfTest():

- Test del metodo public int indexOf(Object o)
- **Summary / Description**: aggiungo alla lista una HCollection di cui conosco la posizione degli elementi, si verifica che gli indici degli elementi nel ListAdapter corrispondano a quelli della collection di riferimento. Si inserisce poi un elemento duplicato e si verifica che venga restituito l'indice minore corrispondete a tale elemento. Si testa infine il comportamento su elementi non presenti nella lista.
- **Design**: verifica del funzionamento del metodo confrontando gli indici restituiti con quelli di una lista di riferimento, anche in caso di elementi duplicati o lista vuota.
- **Pre-Condition**: un oggetto ListAdapter inizializzato correttamente, un parametro "o" diverso da null.
- **Post-Condition**: la lista torna ad essere vuota, il metodo indexOf(Object o) restituirà sempre -1.
- **Expected results**: verifica che il metodo restituisca l'indice, all'interno della lista, dell'oggetto richiesto. Ritorna -1 se tale oggetto non è presente.

19° IndexOfExceptionsTest():

- Test del metodo public int indexOf(Object o)
- **Summary / Description**: si effettua prima un test su una lista vuota poi su una lista contenente altri elementi.
- **Design**: verifica che il metodo lanci NullPointerException qualora si passasse come parametro un oggetto “o” inizializzato al valore null.
- **Pre-Condition**: un oggetto ListAdapter vuoto e correttamente inizializzato.
- **Post-Condition**: il lancio delle eccezioni non modifica il contenuto della lista. Dimensione ed eventuali valori già presenti rimangono inalterati.
- **Expected results**: il lancio dell’ eccezione NullPointerException ed l’inserimento corretto di un elemento nella lista. Nessun altra eccezione deve essere generata.

20° IsEmptyTest():

- Test del metodo public boolean isEmpty()
- **Summary / Description**: prima si verifica su una lista di dimensione pari a zero che il metodo restituisca true. Viene poi aggiunto un elemento alla lista e si verifica che il metodo ritorni false. Viene infine nuovamente svuotata la lista e chiamato nuovamente il metodo aspettandosi di ricevere true come risultato.
- **Design**: si verifica che il metodo restituisca true quando la lista è vuota.
- **Pre-Condition**: un oggetto ListAdapter inizializzato correttamente, un parametro “o” diverso da null.
- **Post-Condition**: la lista torna ad essere vuota, il metodo isEmpty() restituirà sempre true.
- **Expected results**: verifica che il metodo restituisca true se e solo se la lista è vuota.

21° IteratorHasNextTest():

- Test del metodo public boolean hasNext() di Iter implements HIteraor
- **Summary / Description**: su una lista vuota, ricavo un iteratore tramite il metodo iterator*****, si verifica che il metodo hasNext() restituisca false. Viene poi inserita una HCollection all'interno della lista e viene ricreato nuovamente l'iteratore. Avanzando sulla lista si verifica che il metodo hasNext() restituisce true esattamente tante volte quante sono gli elementi della collection.
- **Design**: verifica il corretto funzionamento chiamando il metodo su un iteratore che scorre sulla lista.
- **Pre-Condition**: una oggetto ListAdapter correttamente inizializzato.
- **Post-Condition**: la lista contiene gli elementi della HCollection inserita, l'iteratore si trova alla fine del processo di iterazione.
- **Expected results**: verifica che il metodo resistituisce true se e solo se l'iteratore deve ancora visitare alcuni elementi della lista.

(*****) **Nota**:
il metodo public HIterator iterator() non è stato singolarmente testato in quanto restituisce semplicemente una nuova istanza della classe privata Iter che implementa HIterator, non richiede parametri né può generare eccezioni. Una chiamata a tale metodo non fallisce mai.
I test 21°, 22°, 23° ed 24° a seguire sono incentrati sui metodi di tale classe.

22° IteratorNextTest():

- Test del metodo public Object next() di Iter implements HIterator
- **Summary / Description**: viene aggiunta una HCollection ad una lista vuota, si crea un iteratore e si verifica che il metodo next() restituisce gli elementi della HCollection di riferimento nello stesso ordine in cui sono stati inseriti nella lista. Arrivati alla fine della lista si verifica che un ulteriore chiamata al metodo genera l'eccezione NoSuchElementException.
- **Design**: verifica della corrispondenza tra valori restituiti e valori di riferimento mentre si effettuano chiamate ripetute al metodo finché si itera lungo la lista.
- **Pre-Condition**: una oggetto ListAdapter correttamente inizializzato.
- **Post-Condition**: la lista contiene gli elementi della HCollection inserita, l'iteratore si trova alla fine del processo di iterazione.
- **Expected results**: verifica che il metodo restituisce gli oggetti coerentemente con la loro posizione nella lista.

23° IteratorRemoveTest():

- Test del metodo public Object remove() di Iter implements HIterator
- **Summary / Description**: viene inserita una HCollection all'interno della lista e viene creato un iteratore su di essa. Viene alternata una chiamata ad next con una chiamata ad remove e si verifica che l'oggetto rimosso dalla lista corrisponde all'ultimo oggetto restituito dal metodo next. Si verifica che alla fine del processo la lista sia vuota.
- **Design**: chiamate ripetute al metodo (alternate da chiamate a next) per verificarne il giusto comportamento eliminando tutti gli elementi della lista
- **Pre-Condition**: una oggetto ListAdapter correttamente inizializzato.
- **Post-Condition**: la lista è tornata ad essere vuota, l'iteratore si trova alla fine del processo di iterazione.
- **Expected results**: verifica che il metodo elimina dalla lista il valore restituito dalla precedente chiamata al metodo next.

24° IteratorRemoveExceptionsTest():

- Test del metodo public Object remove() di Iter implements HIterator
- **Summary / Description**: si verifica che il metodo generi l'eccezione quando:
    - viene invocato su una lista vuota
    - viene invocato su una lista che è stata modificata durante un processo di iterazione da parte di altri metodi (undefined behaviour)
    - quando viene invocato prima del metodo next
    - quando sono invocate due chiamate a remove di fila.
- **Design**: verifica che il metodo lancia eccezioni quando non viene intervallato da chiamate a next, viene chiamato su liste vuote.
- **Pre-Condition**: un oggetto ListAdapter vuoto e correttamente inizializzato.
- **Post-Condition**: il lancio delle eccezioni non modifica il contenuto della lista. Dimensione ed eventuali valori già presenti rimangono inalterati.
- **Expected results**: il lancio dell' eccezione IllegalStateException quando si tenta di rimuovere più di un elemento per volta.

25° LastIndexOfTest():

- Test del metodo public int lastIndexOf(Object o)
- **Summary / Description**: vengono aggiunti in ordine gli elementi di una HCollection alla lista, e si verifica che il valore ritornato dal metodo su due elementi di riferimento corrisponda a quello della HCollection. Viene poi nuovamente aggiunta la stessa HCollection e si verifica ora che l'indice restituito, ricercando lo stesso elemento, sia ora maggiore del valore precedente e corrispondente alla posizione del nuovo elemento duplicato nella parte finale della lista. Viene poi effettuata una ricerca su un elemento non presente ed una su una lista vuota.
- **Design**: verifica il corretto comportamento del metodo invocandolo su degli elementi di riferimento di cui si conosce l'indice che vengono confrontati con i valori restituiti dal metodo.
- **Pre-Condition**: un oggetto ListAdapter vuoto e correttamente inizializzato, un parametro "o" diverso da null.
- **Post-Condition**: la lista torna ad essere vuota. Il metodo ora restituirà sempre -1.
- **Expected results**: il metodo restituisce l'indice dell'ultima occorrenza del parametro cercato all'interno della lista. Se questo non è presente restituisce -1.

26° LastIndexOfExceptionsTest():

- Test del metodo public int lastIndexOf(Object o)
- **Summary / Description**: si effettua prima un test su una lista vuota poi su una lista contenente altri elementi.
- **Design**: verifica che il metodo lanci NullPointerException qualora si passasse come parametro un oggetto “o” inizializzato al valore null.
- **Pre-Condition**: un oggetto ListAdapter vuoto e correttamente inizializzato.
- **Post-Condition**: il lancio delle eccezioni non modifica il contenuto della lista. Dimensione ed eventuali valori già presenti rimangono inalterati.
- **Expected results**: il lancio dell’ eccezione NullPointerException ed l’inserimento corretto di un elemento nella lista. Nessun altra eccezione deve essere generata.

27° ListIteratorExceptionsTest():

- Test del metodo public HListIterator listIterator(int index)
- **Summary / Description**: si verifica il lancio dell'eccezione effettuano:
    - un accesso su una lista vuota
    - un accesso ad un indice negativo
    - un accesso fuori range della lista
- **Design**: si genera IndexOutOfBoundException tentando di creare un HListIterator inizializzato ad indici non validi.
- **Pre-Condition**: un oggetto ListAdapter correttamente inizializzato.
- **Post-Condition**:il lancio delle eccezioni non modifica il contenuto della lista. Dimensione ed eventuali valori già presenti rimangono inalterati.
- **Expected results**: il lancio dell' eccezione IndexOutOfBoundException ed l'inserimento corretto di un elemento nella lista. Nessun altra eccezione deve essere generata.

Nota:
A seguire sono riportati i test dei metodi della classe ListIter extends Iter implements HListIterator di cui in istanza è restituita dai metodi listIterator() ed listIterator(int index) di ListAdapter.
I 28°, 29°, 30°, 31°, 32°, 33°, 34°, 35°, 36° test vanno a verificare il corretto comportamento della classe sia quando questa viene inizializzata con un offset iniziale sia quando l'iteratore parte dall'inizio della lista.
I metodo che vengono direttamente ereditati dalla classe Iter non sono stati testati nuovamente. (vedi test 21°, 22°, 23° ed 24°)

28° ListIteratorAddTest():

- Test del metodo public void add(Object o) di ListIter implements HListIterator
- **Summary / Description**: creo un iteratore su una lista vuota ed aggiungo un nuovo elemento, verifico che la dimensione della lista sia stata aumentata, che l'elemento sia stato inserito correttamente e che l'indice dell'iteratore sia stato spostato in maniera corretta. Aggiungo poi una HCollection alla lista e creo un nuovo iteratore questa volta partendo da un offset iniziale. Ripeto le verifiche precedenti aggiungendo due nuovi oggetti.
- **Design**: verifica la correttezza del metodo aggiungendo elementi all'interno della lista e verificando che il loro indice corrisponda a quello desiderato.
- **Pre-Condition**: un oggetto ListAdapter vuoto correttamente inizializzato.
- **Post-Condition**: la lista contiene gli elementi della collection più gli elementi inseriti dall'iteratore posizionati agli indici corretti.
- **Expected results**: verifica che il metodo aggiunga correttamente un elemento nella lista aggiornando gli indici in modo che hasNext, next ed hasPrevius abbiano il giusto comportamento.

29° ListIteratorAddExceptionsTest():

- Test del metodo public void add(Object o) di ListIter implements HListIterator
- **Summary / Description**: si effettua prima un test su una lista vuota poi su una lista contenente altri elementi. Si effettua anche il test del comportamento del metodo quando si tenta di modificare tramite iteratore una lista che ha subito modifiche concorrenti.
- **Design**: verifica che il metodo lanci NullPointerException qualora si passasse come parametro un oggetto “o” inizializzato al valore null.
- **Pre-Condition**: un oggetto ListAdapter vuoto e correttamente inizializzato.
- **Post-Condition**: il lancio delle eccezioni non modifica il contenuto della lista. Dimensione ed eventuali valori già presenti rimangono inalterati.
- **Expected results**: il lancio dell’ eccezione NullPointerException ed l’inserimento corretto di un elemento nella lista. Nessun altra eccezione deve essere generata.

30° ListIteratorHasPreviousTest():

- Test del metodo public boolean hasPrevious() di ListIter implements HListIterator
- **Summary / Description**: viene creato un iteratore su una lista vuota e si verifica che il metodo restituisca false. Viene poi aggiunta una HCollection alla lista e creato un iteratore all'ultima posizione della lista e si verifica che, facendo scorrere l'iteratore all'indietro, il metodo restituisce true fino a quando non si raggiunge la prima posizione sulla lista.
- **Design**: verifica la correttezza del metodo testandolo su una lista di cui si conoscono dimensioni e riferimenti.
- **Pre-Condition**: un oggetto ListAdapter correttamente inizializzato.
- **Post-Condition**: la lista contiene gli elementi della HCollection, l'iteratore è stato portato all'indice iniziale: hasPrevius da sempre false, hasNext da sempre true.
- **Expected results**: il metodo ritorna true se e solo se l'iteratore non si trova nella posizione iniziale della lista.

31° ListIteratorNextTest():

- Test del metodo public Object next() di ListIter implements HListIterator
- **Summary / Description**: viene aggiunta una HCollection ad una lista vuota, si crea un iteratore e si verifica che il metodo next() restituisce gli elementi della HCollection di riferimento nello stesso ordine in cui sono stati inseriti nella lista. Arrivati alla fine della lista si verifica che un ulteriore chiamata al metodo genera l'eccezione NoSuchElementException.
- **Design**: verifica della corrispondenza tra valori restituiti e valori di riferimento mentre si effettuano chiamate ripetute al metodo finché si itera lungo la lista.
- **Pre-Condition**: una oggetto ListAdapter correttamente inizializzato.
- **Post-Condition**: la lista contiene gli elementi della HCollection inserita, l'iteratore si trova alla fine del processo di iterazione.
- **Expected results**: verifica che il metodo restituisce gli oggetti coerentemente con la loro posizione nella lista.

32° ListIteratorNextIndexTest():

- Test del metodo public int nextIndex() di ListIter implements HListIterator
- **Summary / Description**: creato un iteratore su una lista vuota e verificato che il metodo ritorni zero. Viene poi aggiunta una HCollection alla lista e si verifica che ad ogni invocazione del metodo l'indice restituito corrisponde all'indice dell'elemento che verrebbe restituito da una chiamata a next(). Arrivati a fine della lista si verifica che il valore restituito corrisponde alla dimensione della lista.
- **Design**: verifica che nextIndex() mostra l'indice dell'oggetto che sarebbe restituito da una chiamata a next().
- **Pre-Condition**: un oggetto ListAdapter correttamente inizializzato.
- **Post-Condition**: la lista contiene gli elementi di una HCollection, l'iteratore si trova alla posizione finale, nextIndex() restituirà sempre un valore pari alla dimensione della lista.
- **Expected results**: l'indice restituito corrisponde all'indice dell'elemento che verrebbe restituito da una chiamata a next(). Arrivati a fine della lista si verifica che il valore restituito corrisponde alla dimensione della lista.

33° ListIteratorPreviusTest():

- Test del metodo public Object previus() di ListIter implements HListIterator
- **Summary / Description**: viene aggiunta una HCollection ad una lista vuota, si crea un iteratore inizializzato alla posizione finale e si verifica che il metodo previous() restituisce gli elementi della HCollection di riferimento con ordine inverso rispetto a come sono stati inseriti nella lista. Arrivati all'inizio della lista si verifica che un ulteriore chiamata al metodo genera l'eccezione NoSuchElementException.
- **Design**: verifica della corrispondenza tra valori restituiti e valori di riferimento mentre si effettuano chiamate ripetute al metodo finché si itera lungo la lista.
- **Pre-Condition**: una oggetto ListAdapter correttamente inizializzato.
- **Post-Condition**: la lista contiene gli elementi della HCollection inserita, l'iteratore si trova all'inizio della lista.
- **Expected results**: verifica che il metodo restituisce gli oggetti coerentemente con la loro posizione nella lista.

34° ListIteratorPreviousIndexTest():

- Test del metodo public int previusIndex() di ListIter implements HListIterator
- **Summary / Description**: creato un iteratore su una lista vuota e verificato che il metodo ritorni -1. Viene poi aggiunta una HCollection alla lista e si verifica che ad ogni invocazione del metodo l'indice restituito corrisponde all'indice dell'elemento che verrebbe restituito da una chiamata a previous(). Arrivati a fine della lista si verifica che il valore restituito corrisponde nuovamente a -1.
- **Design**: verifica che previousIndex() mostra l'indice dell'oggetto che sarebbe restituito da una chiamata a previous().
- **Pre-Condition**: un oggetto ListAdapter correttamente inizializzato.
- **Post-Condition**: la lista contiene gli elementi di una HCollection, l'iteratore si trova alla posizione iniziale, previousIndex() restituirà sempre un valore pari a -1.
- **Expected results**: l'indice restituito corrisponde all'indice dell'elemento che verrebbe restituito da una chiamata a previous(). Se ci si trova all'inizio della lista si verifica che il valore restituito corrisponde a -1.

35° ListIteratorRemoveTest():

- Test del metodo public void remove() di ListIter implements HListIterator
- **Summary / Description**: viene inserita una HCollection composta da elementi tutti distinti all'interno della lista e si crea un iteratore alla posizione centrale della lista. Si verifica che:
    - Avanzando verso la coda della lista alternando il metodo next() ed il metodo remove() la seconda metà della lista viene completamente cancellata.
    - Tornando ora indietro alternando il metodo previous() ed il metodo remove() si eliminano anche tutti i rimanenti elementi della lista.

  Viene ora riempita nuovamente la lista e si verifica che:

    - Invocare il metodo remove() dopo il metodo add() genera IllegalStateException.
    - Più invocazioni successive del metodo remove() generano IllegalStateException.
    - Invocare il metodo remove() in seguito ad una modifica concorrente alla lista su cui si sta iterando genera IllegalStateException.
- **Design**: verifica il corretto funzionamento del metodo remove() ed il corretto lancio dell'eccezione IllegalStateException da parte di questo.
- **Pre-Condition**: un oggetto ListAdapter correttamente inizializzato.
- **Post-Condition**: la Lista contiene 8 elementi (i primi 7 elementi di una HCollection ed un elemento ""), l'iteratore si trova in uno stato non valido in quanto la lista è stata modificata dal metodo ListAdapter.add() durante il processo di iterazione.
- **Expected results**: il metodo remove() elimina con successo un elemento dalla lista se chiamato dopo previous() o dopo next(). Le dimensioni e gli indici della lista vengono adattate di conseguenza.

36° ListIteratorSetTest():

- Test del metodo public void set(Object o) di ListIterator implements HListIterator
- **Summary / Description**: viene inserita una HCollection composta da elementi tutti distinti all'interno della lista e si crea un iteratore alla posizione centrale della lista. Si verifica che:
    - Avanzando verso la coda della lista alternando il metodo next() ed il metodo set(Object o) i valori della seconda metà della lista vengono tutti sovrascritti a "next".
    - Si torna a metà della lista ed alternando ora il metodo previous() ed il metodo set(Object o) si sovrascrivono anche tutti i rimanenti elementi della lista con il valore "prev".

  Viene ora aggiunta nuovamente la HCollection alla lista e si verifica che:

    - Invocare il metodo set(Object o) dopo il metodo add() genera IllegalStateException (anche se il parametro è null).
    - Invocare il metodo set(Object o) con parametro null genera NullPointerException.
    - Invocare il metodo set(Object o) in seguito ad una modifica concorrente alla lista su cui si sta iterando genera IllegalStateException.
- **Design**: verifica il corretto funzionamento del metodo set(Object o) ed il corretto lancio delle eccezioni IllegalStateException ed NullPointerException da parte di questo.
- **Pre-Condition**: un oggetto ListAdapter correttamente inizializzato.
- **Post-Condition**: la Lista contiene 17 elementi (4 "prev", 4 "next", gli elementi di una HCollection ed un elemento ""), l'iteratore si trova in uno stato non valido in quanto la lista è stata modificata dal metodo ListAdapter.add() durante il processo di iterazione.
- **Expected results**: il metodo set(Object o) sovrascrive con successo un elemento dalla lista se chiamato dopo previous() o dopo next() e con un parametro diverso da null.

37° RemoveIndexTest():

- Test del metodo public Object remove(int index)
- **Summary / Description**: aggiungo una HCollection ad una lista vuota, rimuovo un elemento e verifico che: l'elemento non sia effettivamente più presente nella lista, gli elementi successivi vengano spostati adeguatamente. Invoco poi più volte il metodo fino a quando la lista non risulta completamente vuota verificando che gli elementi vengano effettivamente cancellati e che indici e dimensioni della lista vengano modificati di conseguenza.
- **Design**: verifica il corretto comportamento del metodo applicandolo ripetutamente per rimuovere tutti gli elementi dalla lista.
- **Pre-Condition**: una oggetto ListAdapter correttamente inizializzato, un indice appartenente al range [0,list.size()-1].
- **Post-Condition**: la lista torna ad essere vuota, il metodo remove(int index) non può più essere utilizzato senza generare un'eccezione.
- **Expected results**: il metodo rimuove correttamente un oggetto dalla lista a partire dal suo indice. Il metodo ritorna l'oggetto che è stato rimosso.

38° RemoveIndexExceptionsTest():

- Test del metodo public Object remove(int index)
- **Summary / Description**: si verifica il lancio dell'eccezione effettuando:
    - un tentativo di rimozione su una lista vuota
    - un tentativo di rimozione ad un indice negativo
    - un tentativo di rimozione fuori range della lista
- **Design**: si genera IndexOutOfBoundException tentando di rimuovere un elemento ad indici non validi.
- **Pre-Condition**: un oggetto ListAdapter correttamente inizializzato.
- **Post-Condition**:il lancio delle eccezioni non modifica il contenuto della lista. Dimensione ed eventuali valori già presenti rimangono inalterati.
- **Expected results**: il lancio dell' eccezione IndexOutOfBoundException ed l'inserimento corretto di un elemento nella lista. Nessun altra eccezione deve essere generata.

39° RemoveObjcetTest():

- Test del metodo public boolean remove(Object o)
- **Summary / Description**: aggiungo una HCollection di elementi distinti ad una lista vuota, elimino poi tutti gli oggetti appena inseriti utilizzando il metodo remove(Object o) e verifico che, questo ritorni true, gli oggetti sia effettivamente eliminati dalla lista. Testo ora, sulla lista vuota, la rimozione di un elemento non appartenente alla lista e verifico che il metodo restituisca false. Aggiungo poi tre nuovi elementi alla lista di cui due uguali. Verifico che il metodo elimini dalla lista la prima istanza dell'elemento cercato e non consideri la seconda.
- **Design**: verifica il corretto comportamento del metodo applicandolo ripetutamente per rimuovere tutti gli elementi dalla lista.
- **Pre-Condition**: un oggetto ListAdapter correttamente inizializzato, un parametro "o" diverso da null.
- **Post-Condition**: la lista contiene due elementi, il metodo remove(Object o) può ancora essere invocato con successo.
- **Expected results**: il metodo restituisce true se e solo se ha cancellato con successo la prima istanza dell'elemento cercato all'interno della lista. Se questo non è presente restituisce false.

40° RemoveObjectExceptionsTest():

- Test del metodo public boolean remove(Object o)
- **Summary / Description**: si effettua prima un test su una lista vuota poi su una lista contenente altri elementi.
- **Design**: verifica che il metodo lanci NullPointerException qualora si passasse come parametro un oggetto “o” inizializzato al valore null.
- **Pre-Condition**: un oggetto ListAdapter vuoto e correttamente inizializzato.
- **Post-Condition**: il lancio delle eccezioni non modifica il contenuto della lista. Dimensione ed eventuali valori già presenti rimangono inalterati.
- **Expected results**: il lancio dell’ eccezione NullPointerException ed l’inserimento corretto di un elemento nella lista. Nessun altra eccezione deve essere generata.

41° RemoveAllTest():

- Test del metodo public boolean removeAll(HCollection c)
- **Summary / Description**: creo una HCollection da usare come parametro e verifico che il metodo restituisca false se applicato su una lista vuota. Aggiungo poi la stessa collection alla lista ed aggiungo inoltre altri tre elementi di cui uno presente anche nella collection. Verifico che invocando nuovamente il metodo vengono cancellati, una sola volta, tutti gli elementi della collection (l'elemento duplicato non viene cancellato due volte). Verifico dimensioni ed indici della lista. Aggiungo infine i tre elementi prima inseriti nella lista anche nella collection usata come parametro e verifico che invocando nuovamente il metodo la lista viene svuotata completamente.
- **Design**: verifica che il metodo rimuova dalla lista la sua intersezione con la collection usata come parametro non rimuovendo però eventuali elementi duplicati.
- **Pre-Condition**: un oggetto ListAdapter correttamente inizializzato, un paramtero HCollection c diverso da null.
- **Post-Condition**: la lista torna ad essere vuota, il metodo removeAll(HCollection c) restituirà sempre false.
- **Expected results**: il metodo rimuove gli elementi presenti nella HCollection dalla lista e restituisce true se e solo se la dimensione di questa è stata modificata.

42° RemoveAllExceptionsTest():

- Test del metodo public boolean removeAll(HCollection c)
- **Summary / Description**: si effettua prima un test su una lista vuota poi su una lista contenente altri elementi.
- **Design**: verifica che il metodo lanci NullPointerException qualora si passasse come parametro una HCollection c pari al valore null.
- **Pre-Condition**: un oggetto ListAdapter vuoto e correttamente inizializzato.
- **Post-Condition**: il lancio delle eccezioni non modifica il contenuto della lista. Dimensione ed eventuali valori già presenti rimangono inalterati.
- **Expected results**: il lancio dell' eccezione NullPointerException ed l'inserimento corretto di un elemento nella lista. Nessun altra eccezione deve essere generata.

43° RetainAllTest():

- Test del metodo public boolean retainAll(HCollection c)
- **Summary / Description**: creo una HCollection da usare come parametro e verifico che il metodo restituisca false se applicato su una lista vuota. Aggiungo poi la stessa collection alla lista ed aggiungo inoltre altri tre elementi di cui uno presente anche nella collection. Verifico che invocando nuovamente il metodo vengono cancellati i due dei tre elementi appena aggiunti: non viene cancellato l'elemento che era già presente anche nella collection che ora sarà presente in duplice copia nella lista. Verifico dimensioni ed indici della lista. Cancello infine tutti gli elementi della collection usata come parametro e verifico che invocando nuovamente il metodo la lista viene svuotata completamente.
- **Design**: verifica che il metodo rimuova dalla lista gli elementi che non fanno parte della sua intersezione con la collection usata come parametro.
- **Pre-Condition**: un oggetto ListAdapter correttamente inizializzato, un paramtero HCollection c diverso da null.
- **Post-Condition**: la lista torna ad essere vuota, il metodo retainAll(HCollection c) restituirà sempre false.
- **Expected results**: il metodo rimuove gli elementi della lista che non sono presenti nella HCollection e restituisce true se e solo se la dimensione della lista è stata modificata.

44° RetainAllExceptionsTest():

- Test del metodo public boolean retainAll(HCollection c)
- **Summary / Description**: si effettua prima un test su una lista vuota poi su una lista contenente altri elementi.
- **Design**: verifica che il metodo lanci NullPointerException qualora si passasse come parametro una HCollection c pari al valore null.
- **Pre-Condition**: un oggetto ListAdapter vuoto e correttamente inizializzato.
- **Post-Condition**: il lancio delle eccezioni non modifica il contenuto della lista. Dimensione ed eventuali valori già presenti rimangono inalterati.
- **Expected results**: il lancio dell' eccezione NullPointerException ed l'inserimento corretto di un elemento nella lista. Nessun altra eccezione deve essere generata.

45° SetTest():

- Test del metodo public Object set(int index, Object o)
- **Summary / Description**: si aggiunge una HCollection di elementi tutti distinti alla lista e si verifica che, invocando il metodo su più indici, il metodo sovrascriva correttamente un elemento e ne restituisca il valore senza mai modificare la dimensione della lista.
- **Design**: verifica che il metodo sovrascriva correttamente un valore della lista senza cambiarne la dimensione a prescindere dall'indice utilizzato (purché questo sia valido).
- **Pre-Condition**: un oggetto ListAdapter vuoto e correttamente inizializzato un indice valido ed un parametro "o" diverso da null.
- **Post-Condition**: la lista contenente 5 elementi della HCollection inserita e 3 elementi sovrascritti dal metodo.
- **Expected results**: il metodo sovrascrive correttamente un valore della lista e ritorna l'oggetto sostituito. Questo, se non è un doppione, non sarà più presente nella lista. La dimensione della lista non cambia.

46° SetExceptionsTest():

- Test del metodo public Object set(int index, Object o)
- **Summary / Description**: si effettua prima un test su una lista vuota poi su una lista contenente altri elementi e si verifica che:
    - Il metodo genera sempre NullPointerException se “o” è uguale a null (anche quando anche l’indice è invalido)
    - Il metodo genera IndexOutOfBoundsException se il parametro “o” è valido ma l’indice non appartiene al range [0,list.size()-1].
- **Design**: verifica che il metodo lanci NullPointerException qualora si passasse come parametro un valore null e verifica che lanci IndexOutOfBoundsException se si passa un indice non valido.
- **Pre-Condition**: un oggetto ListAdapter vuoto e correttamente inizializzato.
- **Post-Condition**: il lancio delle eccezioni non modifica il contenuto della lista. Dimensione ed eventuali valori già presenti rimangono inalterati.
- **Expected results**: il lancio dell’ eccezione NullPointerException e di IndexOutOfBoundsException ed l’inserimento corretto di un elemento nella lista. Nessun altra eccezione deve essere generata.

47° SizeTest():

- Test del metodo public int size()
- **Summary / Description**: verifica che il metodo restituisca sempre la dimensione della lista di riferimento, anche se vengono aggiunti o rimossi elementi.
- **Design**: verifica la correttezza del metodo aggiungendo e rimuovendo svariati valori.
- **Pre-Condition**: oggetto di tipo ListAdapter correttamente inizializzato.
- **Post-Condition**: la lista torna ad essere vuota, size() restituisce zero.
- **Expected results**: il metodo restituisce un valore che rispecchia fedelmente il numero di elementi contenuti nella lista.

48° SublistTest():

- Test del metodo public boolean add(Object o)
- **Summary / Description**: viene aggiunta una HCollection ad una lista inizialmente vuota, sia vanno a creare poi diverse sublist (con indici validi) a partire dalla lista e si verifica che queste abbiano la dimensione corretta.
- **Design**: verifica che venga creata correttamente una sublist con la giusta dimensione, senza modificare la lista originaria.
- **Pre-Condition**: un oggetto ListAdapter correttamente inizializzato ed indici fromIndex ed toIndex validi.
- **Post-Condition**: la lista di base non subisce modifiche.
- **Expected results**: ottenere per ogni chiamata una sublist di dimensione toIndex-fromIndex senza mai modificare la lista di base.

49° SublistExceptionsTest():

- Test del metodo public boolean add(Object o)
- **Summary / Description**: si verifica il lancio dell'eccezione quando:
    - fromIndex è minore di zero
    - toIndex è maggiore della dimensione della lista
    - fromIndex è maggiore di toIndex
- **Design**: verifica che si genera IndexOutOfBoundException tentando di creare una sublist ad indici non validi.
- **Pre-Condition**: un oggetto ListAdapter correttamente inizializzato.
- **Post-Condition**:il lancio delle eccezioni non modifica il contenuto della lista. Dimensione ed eventuali valori già presenti rimangono inalterati.
- **Expected results**: il lancio dell' eccezione IndexOutOfBoundException ed l'inserimento corretto di una HCollection nella lista. Nessun altra eccezione deve essere generata.

50° ToArrayTest():

- Test del metodo public Object[] toArray()
- **Summary / Description**: viene aggiunta una HCollection ad una lista inizialmente vuota. Si va poi ad invocare il metodo e si verifica che:
    - L'array e la lista hanno le stesse dimensioni
    - L'array contiene tutti gli elementi della lista, presenti nello stesso ordine.
    - La lista non viene modificata in nessun modo
- **Design**: verifica la corretta creazione di un array contenente tutti gli elementi presenti nella lista.
- **Pre-Condition**: un oggetto ListAdapter correttamente inizializzato.
- **Post-Condition**: la lista contiene tutti gli elementi della HCollection e non viene modificata in nessun modo.
- **Expected results**: il metodo restituisce un array ordinato contenente tutti gli elementi della lista.

51° ToArrayObjectTest():

- Test del metodo public Object[] toArray(Object[] a)
- **Summary / Description**: viene aggiunta una HCollection alla lista e viene testato il metodo sia su un array di dimensione minore rispetto alla lista sia su un array di dimensione maggiore uguale alla lista. Si verifica che:
    - Array troppo piccolo: l'array restituito ha ora dimensione uguale a quella della lista e contiene tutti i sui elementi nello stesso ordine.
    - Array sufficientemente grande: viene restituito lo stesso array dove i primi elementi sono stati sovrascritti con quelli della lista. La sua dimensione ed eventuali altri elementi non sono modificati.
- **Design**: verifica la corretta creazione di un array contenente tutti gli elementi presenti nella lista nel caso in cui l'array passato come parametro fosse troppo piccolo. Verifica altrimenti che l'array fornito venga riempito/sovrascritto adeguatamente.
- **Pre-Condition**: un oggetto ListAdapter correttamente inizializzato.
- **Post-Condition**: la lista contiene tutti gli elementi della HCollection e non viene modificata in nessun modo.
- **Expected results**: il metodo restituisce un array ordinato contenente tutti gli elementi della lista.

52° ToArrayObjectExceptionsTest():

- Test del metodo public Object[] toArray(Object[] a)
- **Summary / Description**: verifica che:
    - Il metodo genera sempre NullPointerException se “a” è uguale a null
    - Il metodo genera ArrayStoreException se l’array “a” passato come parametro è grande a sufficienza per contenere i valori della lista ma è di un tipo runtime non compatibile con tutti i valori contenuti nella lista.
- **Design**: verifica che il metodo lanci NullPointerException qualora si passasse come parametro un valore null e verifica che lanci ArrayStoreException se si passa un array di un tipo runtime non valido.
- **Pre-Condition**: un oggetto ListAdapter vuoto e correttamente inizializzato.
- **Post-Condition**: il lancio delle eccezioni non modifica il contenuto della lista. Dimensione ed eventuali valori già presenti rimangono inalterati.
- **Expected results**: il lancio dell’ eccezione NullPointerException e di ArrayStoreException ed l’inserimento corretto di un elemento nella lista. Nessun altra eccezione deve essere generata.

53° SubListAddIndexTest():

- Test del metodo public void add(int index, Object o) in “modalità sublist”
- **Summary / Description**: viene inizialmente inserita nella lista una HCollection e viene poi create, a partire da questa una sublist di tre elementi. Vengono inseriti tre elementi, diversi da null, in varie posizioni valide della sublist e si verifica che questi vengano inseriti anche nella lista. I vari inserimenti verificano che gli elementi della lista e dalla sublist vengano opportunamente spostati quando viene inserito un nuovo elemento prima di essi. Mentre i vari elementi vengono inseriti si verifica che la dimensione della lista e della sublist venga aumentata congruentemente e che gli oggetti inseriti occupino effettivamente la posizione richiesta
- **Design**:. verifica che l’inserimento di un elemento, diverso da null, all’interno della sublist avvenga correttamente modificando correttamente gli indici di eventuali elementi già presenti nella sia nella sublist che nella lista sottostante.
- **Pre-Condition**: un oggetto ListAdapter vuoto e correttamente inizializzato. Parametro “o” inizializzato con valore diverso da null. Parametro “index” appartenente al range [0, list.size()]
- **Post-Condition**: la sublist e la lista dovranno contenere gli elementi, inseriti correttamente, nessuna eccezione dovrà essere generata.
- **Expected results**: una sublist contenente tre elementi, nell’ordine desiderato e senza il lancio di nessuna eccezione.

54° SubListAddIndexExceptionsTest():

- Test del metodo public void add(int index, Object o) in “modalità sublist”
- **Summary / Description**: entrambe le eccezioni vengono inizialmente testate su una sublist vuota e si verifica che:
    - null su una sublist vuota ad un indirizzo valido genera NullPointerException.
    - un elemento valido su una sublist vuota ad un indice non valido genera IndexOutOfBoundException. (testato con più indici)
    - null su una sublist vuota ad un indirizzo invalido genera IndexOutOfBoundException.

  Si verifica che la dimensione della lista e della sublist non sono state modificate (non è stato effettivamente inserito nessun elemento).

  Viene poi aggiunto un elemento valido alla sublist e si ripetono i test in maniera analoga a prima, ma su una sublist non vuota, verificando che l’elemento inserito in precedenza non subisca modifiche.

- **Design**: verifica che:
    - il metodo lanci l’eccezione NullPointerException qualora si tentasse di inserire un oggetto inizializzato a null nella sublist, come specificato dall’interfaccia. Questo deve verificarsi sia quando la sublist è vuota sia quando questa contiene già altri elementi.
    - Il metodo lanci l’eccezione IndexOutOfBoundException qualora si tentasse di inserire un elemento al di fuori del range [0, list.size()].
- **Pre-Condition**: un oggetto ListAdapter vuoto e correttamente inizializzato.
- **Post-Condition**: il lancio delle eccezioni non modifica il contenuto della lista né della sublist. Dimensione ed eventuali valori già presenti rimangono inalterati.
- **Expected results**: il lancio delle eccezioni NullPointerException ed IndexOutOfBoundException ed l’inserimento corretto di un elemento nella sublist. Nessun altra eccezione deve essere generata.

55° SubListClearTest():

- Test del metodo public void clear() in “modalità sublist”
- **Summary / Description**: viene creata una sublist di tre elementi partendo da una lista contente un HCollection, si verifica che il metodo svuoti la sublist elimando gli elementi contenuti anche dalla lista sottostante senza però modificare gli altri elementi. La sublist viene poi riempita nuovamente ed il processo ripetuto.
- **Design**: verifica che tutti gli elementi della sublist vengano effettivamente eliminati e che la sua dimensione venga impostata a zero.
- **Pre-Condition**: un oggetto ListAdapter vuoto inizializzato correttamente.
- **Post-Condition**: la sublist dovrà tornare ad essere vuota, nessuna eccezione dovrà essere generata.
- **Expected results**: una sublist vuota con dimensione uguale a zero, gli elementi dovranno essere eliminati sia dalla sublist che dalla lista sottostante.

56° SubListContainsTest():

- Test del metodo public boolean contains(Object o) in “modalità sublist”
- **Summary / Description**: si crea una sublist contenente un elemento partendo da una lista. Si testa inizialmente il corretto funzionamento su un elemento non presente nella sublist, vengono poi aggiunti degli elementi alla sublist e si verifica che questi vengano effettivamente “individuati” dal metodo contains. Questi elementi vengono poi eliminati dalla lista e si verifica che ora non sia più visibili al metodo contains. Si verifica inoltre che un elemento presente nella lista ma non nella sublist non venga identificato dal metodo contains() di quest’ultima.
- **Design**: verifica che il metodo ritorni:
    - “true” quando si passa come parametro un oggetto effettivamente contenuto nella sublist. Questo deve essere vero a prescindere dalla posizione che questo occupa nella sublist.
    - “false” altrimenti.

  Il metodo deve lavorare correttamente anche su una sublist vuota (ritorna sempre false).

- **Pre-Condition**: un oggetto ListAdapter vuoto correttamente iniziallizato, un parametro “o” diverso da null.
- **Post-Condition**: la sublist torna ad essere vuota ed il metodo contains continuerà a ritornare false ad ogni invocazione. Nessun eccezione deve essere generata.
- **Expected results**: il metodo ritorna il valore true se e solo se l’oggetto è presente nella sublist.

57° SubListContainsExceptionsTest():

- Test del metodo public boolean contains(Object o) in “modalità sublist”
- **Summary / Description**: si effettua prima un test su una sublist vuota poi su una sublist contenente altri elementi.
- **Design**: verifica che il metodo lanci NullPointerException qualora si passasse come parametro il valore null.
- **Pre-Condition**: un oggetto ListAdapter vuoto e correttamente inizializzato.
- **Post-Condition**: il lancio delle eccezioni non modifica il contenuto della sublist né quello della lista. Dimensione ed eventuali valori già presenti rimangono inalterati.
- **Expected results**: il lancio dell’ eccezione NullPointerException ed l’inserimento corretto di un elemento nella sublist. Nessun altra eccezione deve essere generata.

58° SubListIndexOfTest():

- Test del metodo public int indexOf(Object o) in “modalità sublist”
- **Summary / Description**: aggiungo alla sublist una HCollection di cui conosco la posizione degli elementi, si verifica che gli indici degli elementi nel ListAdapter corrispondano a quelli della collection di riferimento. Si inserisce poi un elemento duplicato e si verifica che venga restituito l’indice minore corrispondete a tale elemento. Si testa infine il comportamento su elementi non presenti nella sublist e si verificano che non ci siano conflitti con elementi presenti nella lista di base ma non nella sublist.
- **Design**: verifica del funzionamento del metodo confrontando gli indici restituiti con quelli di una lista di riferimento, anche in caso di elementi duplicati o sublist vuota.
- **Pre-Condition**: un oggetto ListAdapter inizializzato correttamente, un parametro “o” diverso da null.
- **Post-Condition**: la sublist torna ad essere vuota, il metodo indexOf(Object o) restituirà sempre -1.
- **Expected results**: verifica che il metodo restituisca l’indice, all’interno della sublist, dell’oggetto richiesto. Ritorna -1 se tale oggetto non è presente.

59° SubListIndexOfExceptionsTest():

- Test del metodo public int indexOf(Object o) in “modalità sublist”
- **Summary / Description**: si effettua prima un test su una sublist vuota poi su una sublist contenente altri elementi.
- **Design**: verifica che il metodo lanci NullPointerException qualora si passasse come parametro un oggetto “o” inizializzato al valore null.
- **Pre-Condition**: un oggetto ListAdapter vuoto e correttamente inizializzato.
- **Post-Condition**: il lancio delle eccezioni non modifica il contenuto della sublist né della lista sottostante. Dimensione ed eventuali valori già presenti rimangono inalterati.
- **Expected results**: il lancio dell’ eccezione NullPointerException ed l’inserimento corretto di un elemento nella sublist. Nessun altra eccezione deve essere generata.

60° SubListRemoveIndexTest():

- Test del metodo public Object remove(int index) in “modalità sublist”
- **Summary / Description**: aggiungo una HCollection ad una lista vuota, e creo partendo da questa una sublist di 6 elementi. Rimuovo un elemento e verifico che: l’elemento non sia effettivamente più presente nella sublist né nella lista, gli elementi successivi vengano spostati adeguatamente. Invoco poi più volte il metodo fino a quando la sublist non risulta completamente vuota verificando che gli elementi vengano effettivamente cancellati e che indici e dimensioni della sublist e della lista sottostante vengano modificati di conseguenza.
- **Design**: verifica il corretto comportamento del metodo applicandolo ripetutamente per rimuovere tutti gli elementi dalla sublist.
- **Pre-Condition**: una oggetto ListAdapter correttamente inizializzato, un indice appartenente al range [0,list.size()-1].
- **Post-Condition**: la sublist torna ad essere vuota, il metodo remove(int index) non può più essere utilizzato senza generare un’eccezione.
- **Expected results**: il metodo rimuove correttamente un oggetto dalla sublist a partire dal suo indice. Il metodo ritorna l’oggetto che è stato rimosso.

61° SubListRemoveIndexExceptionsTest():

- Test del metodo public Object remove(int index) in “modalità sublist”
- **Summary / Description**: si verifica il lancio dell’eccezione effettuando:
  - un tentativo di rimozione su una sublist vuota
  - un tentativo di rimozione ad un indice negativo
  - un tentativo di rimozione fuori range della sublist
- **Design**: si genera IndexOutOfBoundException tentando di rimuovere un elemento ad indici non validi.
- **Pre-Condition**: un oggetto ListAdapter correttamente inizializzato.
- **Post-Condition**:il lancio delle eccezioni non modifica il contenuto della sublist. Dimensione ed eventuali valori già presenti rimangono inalterati.
- **Expected results**: il lancio dell’ eccezione IndexOutOfBoundException ed l’inserimento corretto di un elemento nella sublist. Nessun altra eccezione deve essere generata.

62° SubListRemoveObjectTest():

- Test del metodo public boolean remove(Object o) in “modalità sublist”
- **Summary / Description**: aggiungo una HCollection di elementi distinti ad una sublist vuota, elimino poi tutti gli oggetti appena inseriti utilizzando il metodo remove(Object o) e verifico che, questo ritorni true, gli oggetti sia effettivamente eliminati dalla sublist. Testo ora, sulla sublist vuota, la rimozione di un elemento non appartenente alla sublist e verifico che il metodo restituisca false. Aggiungo poi tre nuovi elementi alla sublist di cui due uguali. Verifico che il metodo elimini dalla lista la prima istanza dell’elemento cercato e non consideri la seconda. Verifico che gli indici della lista sottostante sia correttamente modificati.
- **Design**: verifica il corretto comportamento del metodo applicandolo ripetutamente per rimuovere tutti gli elementi dalla sublist.
- **Pre-Condition**: un oggetto ListAdapter correttamente inizializzato, un parametro “o” diverso da null.
- **Post-Condition**: la sublist contiene due elementi, il metodo remove(Object o) può ancora essere invocato con successo.
- **Expected results**: il metodo restituisce true se e solo se ha cancellato con successo la prima istanza dell’elemento cercato all’interno della sublist. Se questo non è presente restituisce false.

63° SubListRemoveObjectExceptionsTest():

- Test del metodo public boolean remove(Object o) in “modalità sublist”
- **Summary / Description**: si effettua prima un test su una sublist vuota poi su una sublist contenente altri elementi.
- **Design**: verifica che il metodo lanci NullPointerException qualora si passasse come parametro un oggetto “o” inizializzato al valore null.
- **Pre-Condition**: un oggetto ListAdapter vuoto e correttamente inizializzato.
- **Post-Condition**: il lancio delle eccezioni non modifica il contenuto della sublist né della lista sottostante. Dimensione ed eventuali valori già presenti rimangono inalterati.
- **Expected results**: il lancio dell’ eccezione NullPointerException ed l’inserimento corretto di un elemento nella sublist. Nessun altra eccezione deve essere generata.

64° SubListSizeTest():

- Test del metodo public int size() in “modalità sublist”
- **Summary / Description**: verifica che il metodo restituisca sempre la dimensione della sublist di riferimento, anche se vengono aggiunti o rimossi elementi.
- **Design**: verifica la correttezza del metodo aggiungendo e rimuovendo svariati valori.
- **Pre-Condition**: oggetto di tipo ListAdapter correttamente inizializzato.
- **Post-Condition**: la sublist torna ad essere vuota, size() restituisce zero.
- **Expected results**: il metodo restituisce un valore che rispecchia fedelmente il numero di elementi contenuti nella sublist senza mai modificare i valori della lista sottostante.