Descizione Safe Test suite

Sviluppatore: **Cappellotto Lorenzo, matricola 1188257**, corso ingegneria informatica generale.

Ambiente di sviluppo:

Tutte le classi adapter e le classi di test sono state sviluppate usando l'ide Intellj IDEA 2020

La versione di java usata "openjdk version 1.8.0_252". Nonostante cio' mi sono attenuto alle limitazioni imposte nella cosegna e le classi sono state costruite in modo da usare solamente le funzionalita' proposte dalla versione javaME cldc 1.1, proponendo le funzionalita' della versione Java2 Collections Framework 1.4.2.

Per svolgere i test ho usato la versione 4.13 di Junit. I test sono strutturati attraverso un testRunner e 5 classi di test da eseguire per le classi implementate.

Gli assert usati per testare i metodi delle classi sono:

- assertEquals(), assertNotEquals(), assertNull(), assertTrue(), assertFalse(), assertArrayEquals()
  Per testare i casi in cui un valore (o un vettore) e' ritornato dai metodi rispetto a un valore atteso.
- assertThrows()
  Per testare i casi in cui i metodi lanciano eccezioni.

I pattern usati per implementare le classi sono due:

- Adapter Pattern, usato per le classi richieste nella consegna.
- Iterator Pattern, usato per le classi di supporto che implementano le interfacce Iterator e ListIterator.

Entrambi i pattern e il loro specifico uso nel sofware e' spiegato all'interno dei javadoc delle specifiche classi.

In seguito vi e' una spiegazione nel dettaglio della struttura dei test e della loro modalita' di esecuzione.

CollectionAdapterTest

Sommario:

Questa classe serve solamente a testare i metodi di CollectionAdapter sovrascritti rispetto a ListAdapter, dato che il funzionamento degli altri e' stato testato in ListAdapterTest.

I due test svolti sono:

- metodoHashCode()
- metodoEquals()

Design della suite di test:

I test dei due metodi sovrascritti equals() e hashcode() sono unici data la semplicita' degli stessi.

Variabili di esecuzione:
Vi sono delle variabili comuni ai metodi di test:

- HCollection miaCollezione, altraCollezione: due istanza di tipo CollectionAdapter su cui richiamare i metodi da testare
- Vector vettoreObj: un vettore di supporto, popolato con oggetti di tipo diverso tra di loro, e' usato per inserire valori nelle collezioni all'interno dei test

Metodi:

Nome metodo	MetodoHashCode()
Descrizione	Questo metodo esegue il test di CollectionAdapter.hashCode()
Variabili di esecuzione	Non vi sono variabili aggiuntive per questo metodo
Pre-condizioni	Entrambe le collezioni devono essere vuote
Casi di test	Viene richiamato il metodo hashCode() su due collezioni diverse in diverse situazioni: 1. entrambe le collezioni sono vuote 2. entrambe le collezioni vengono popolate con gli stessi oggetti contenuti il vettoreObj 3. viene eliminato e reinserito un oggetto in una delle due collezioni
Record di esecuzione	Si deve ottenere la seguente situazione: 1. le collezioni hanno hashcode 0 2. le collezioni hanno lo stesso hashcode 3. le collezioni continuano ad avere lo stesso hashcode (l'ordine non e' importante)
Post-condizioni	Non vi sono post-condizioni particolari

Nome metodo	MetodoEquals()
Descrizione	Questo metodo esegue il test di CollectionAdapter.equals(Object)
Variabili di esecuzione	Non vi sono variabili aggiuntive per questo metodo
Pre-condizioni	Entrambe le collezioni devono essere vuote
Casi di test	Viene richiamato il metodo equals() confrontando la collezione con: 1. il parametro null 2. un elemento di tipo diverso da HCollection Viene in seguito richiamato il metodo equals tra due collezioni nelle seguenti situazioni: 3. entrambe le collezioni sono vuote 4. una sola collezione viene popolata con gli oggetti contenuti il vettoreObj 5. entrambe le collezioni popolate con gli oggetti contenuti il vettoreObj 6. dopo aver eliminato e reinserito un oggetto in una delle due collezioni
Record di esecuzione	Il metodo deve ritornare: 1. false 2. false 3. true 4. false 5. true 6. true (l'ordine non e' importante)

Post-condizioni	Non vi sono post-condizioni particolari

SetAdapterTest

Sommario:

Questa classe serve solamente a testare i metodi di SetAdapter sovrascritti rispetto a CollectionAdapter, dato che il funzionamento degli altri e' stato testato in CollectionAdapterTest.

I test svolti sono:

- metodoAdd()
- metodoAddAllCasoLimite()
- metodoAddAllCasoNormale()
- metodoEquals()

Design della suite di test:

I test dei due metodi sovrascritti equals() e add() sono unici data la semplicita' degli stessi.

Il test di addAll() e' diviso per verificare separatamente il caso limite in cui viene lanciata l'eccezione dal caso d'uso normale.

Variabili di esecuzione:
Vi sono delle variabili comuni ai metodi di test:

- HSet mioSet: un istanza di tipo SetAdapter su cui richiamare i metodi da testare
- Vector vettoreObj: un vettore di supporto, popolato con oggetti di tipo diverso tra di loro, e' usato per inserire valori nei set all'interno dei test

Metodi:

Nome metodo	MetodoEquals()
Descrizione	Questo metodo esegue il test di SetAdapter.equals(Object)
Variabili di esecuzione	Vi e' una variabile aggiuntiva HSet altroSet, usata come controparte a mioSet per il test del metodo
Pre-condizioni	Entrambi i set devono essere vuoti
Casi di test	Viene richiamato il metodo equals() confrontando il set con: 1. il parametro null 2. un elemento di tipo diverso da HSet Viene in seguito richiamato il metodo equals tra due set diversi nelle seguenti situazioni: 3. entrambe i set sono vuoti 4. un solo set viene popolato con gli oggetti contenuti il vettoreObj 5. entrambe i set popolati con gli oggetti contenuti il vettoreObj 6. dopo aver eliminato e reinserito un oggetto in uno dei due set
Record di esecuzione	Il metodo deve ritornare: 1. false 2. false 3. true 4. false

	5. true 6. true (l’ordine non e’ importante)
Post-condizioni	Non vi sono post-condizioni particolari

Nome metodo	MetodoAdd()
Descrizione	Questo metodo esegue il test di SetAdapter.add(Object)
Variabili di esecuzione	Non vi sono variabili aggiuntive per questo metodo
Pre-condizioni	Il set deve essere vuoto
Casi di test	Viene richiamato il metodo add() su di un set cercando di inserire: 1. elementi unici 2. elementi duplicati
Record di esecuzione	Si deve ottenere la seguente situazione: 1. il metodo ritorna true e l’elemento viene inserito 2. il metodo ritorna false e l’elemento non viene inserito
Post-condizioni	Il set deve contenere tutti gli elementi iseriti, a meno che non siano duplicati

Nome metodo	MetodoAddAllCasoLimite()
Descrizione	Questo metodo esegue il test di SetAdapter.addAll(HCollection) in un caso limite
Variabili di esecuzione	Non vi sono variabili aggiuntive per questo metodo
Pre-condizioni	Il set deve essere vuoto
Casi di test	Viene richiamato il metodo addAll() su di un set passando come parametro null
Record di esecuzione	Il metodo lancia l’eccezione NullPointerException e il set non viene modificato
Post-condizioni	Non vi sono post-condizioni particolari

Nome metodo	MetodoAddAllCasoNormale()
Descrizione	Questo metodo esegue il test di SetAdapter.addAll(HCollection) in un caso d’uso normale
Variabili di esecuzione	Vi e’ una variabile aggiuntiva Hcollection coll, usata come collezione da passare al metodo
Pre-condizioni	Il set deve essere vuoto
Casi di test	Viene richiamato il metodo addAll() passando come parametro: 1. una collezione vuota 2. una collezione contenente anche elementi duplicati
Record di esecuzione	Il metodo deve ritornare:

	1. false e il set deve rimanere invariato 2. true e il set deve contenere tutti gli oggetti contenuti nella collezione a parte i duplicati
Post-condizioni	Il set deve contenere tutti gli elementi iseriti, a meno che non siano duplicati

MapAdapterTest

Sommario:

Questa classe serve a testare MapAdapter e tutte le sottoclassi di supporto che essa usa per funzionare correttamente.

I test svolti sono:

- metodoHashCode()
- metodoEquals()
- metodoPutCasiLimite()
- metodoPutCasoNormale()
- metodoPutAllCasoLimite()
- metodoPutAllCasoNormale()
- metodoContainsKeyCasoLimite()
- metodoContainsKeyCasoNormale()
- metodoContainsValueCasoLimite()
- metodoContainsValueCasoNormale()
- metodoGetCasoLimite()
- metodoGetCasoNormale()
- metodoRemoveCasoLimite()
- metodoRemoveCasoNormale()
- metodoEntrySet()
- metodoKeySet()
- metodoValues()
- i metodi size, clear e isEmpty vengono usati e dunque testati durante tutto il processo, non vi sono dei test appositi per questi tre metodi

Design della suite di test:

I test di alcuni metodi sono unici perche' i metodi non sono complessi o perche' si basano su classi inerne a MapAdapter e di cui ho testato il funzionamento solo in un unico metodo, metre i test di altri metodi sono divisi in casi limite e casi normale.

Variabili di esecuzione:
Vi sono delle variabili comuni ai metodi di test:

- HMap miaMappa: un istanza di tipo MapAdapter su cui richiamare i metodi da testare

Metodi:

Nome metodo	MetodoHashCode()

Descrizione	Questo metodo esegue il test di MapAdapter.hashCode()
Variabili di esecuzione	Non vi sono variabili aggiuntive per questo metodo
Pre-condizioni	La mappa deve essere vuota
Casi di test	Viene richiamato il metodo hashCode() sulla mappa in diverse situazioni: 1. la mappa e' vuota 2. la mappa e' popolata con delle entry 3. dopo aver eliminato e reinserito una entry
Record di esecuzione	Si deve ottenere la seguente situazione: 1. la mappa ha hashcode 0 2. la mappa ha hashcode diverso da 0 3. le mappa continua ad avere lo stesso hashcode del punto 2. (l'ordine non e' importante)
Post-condizioni	Non vi sono post-condizioni particolari

Nome metodo	MetodoEquals()
Descrizione	Questo metodo esegue il test di MapAdapter.equals(Object)
Variabili di esecuzione	Vi e' una variabile aggiuntiva HMap altraMappa, usata come controparte a miaMappa per il test del metodo
Pre-condizioni	Entrambi le mappe devono essere vuote
Casi di test	Viene richiamato il metodo equals() confrontando la mappa con: 1. il parametro null 2. un elemento di tipo diverso da HMap Viene in seguito richiamato il metodo equals tra due mappe diverse nelle seguenti situazioni: 3. entrambe sono vuote 4. una sola mappa viene popolata con delle entry 5. entrambe le mappe popolati con le stesse entry 6. dopo aver eliminato e reinserito una entry in una delle due mappe
Record di esecuzione	Il metodo deve ritornare: 1. false 2. false 3. true 4. false 5. true 6. true (l'ordine non e' importante)
Post-condizioni	Non vi sono post-condizioni particolari

Nome metodo	MetodoPutCasiLimite()
Descrizione	Questo metodo esegue il test di MapAdapter.put(Object,Object) in dei casi limite
Variabili di esecuzione	Non vi sono variabili aggiuntive per questo metodo

Pre-condizioni	La mappa deve essere vuota
Casi di test	Viene richiamato il metodo put() passando come parametri: 1. chivare e valore null 2. valore null 3. chiave null
Record di esecuzione	Il metodo in tutti e tre i casi deve lanciare l'eccezione NullPointerException
Post-condizioni	La mappa non deve essere modificata

Nome metodo	MetodoPutCasoNormale()
Descrizione	Questo metodo esegue il test di MapAdapter.put(Object,Object) in un caso d'uso normale
Variabili di esecuzione	Non vi sono variabili aggiuntive per questo metodo
Pre-condizioni	La mappa deve essere vuota
Casi di test	Viene richiamato il metodo put() passando come parametri: 1. chiavi mai inserite e valori qualsiasi 2. una chiave gia' inserita e valore qualsiasi
Record di esecuzione	Il metodo deve ritornare: 1. null e la mappa deve contenere le entry attese 2. il valore precedentemente associato alla chiave e la mappa deve contenere la entry con valore aggiornato
Post-condizioni	Non vi sono post-condizioni particolari

Nome metodo	MetodoPutAllCasoLimite()
Descrizione	Questo metodo esegue il test di MapAdapter.putAll(HCollection) in un caso limite
Variabili di esecuzione	Non vi sono variabili aggiuntive per questo metodo
Pre-condizioni	La mappa deve essere vuota
Casi di test	Viene richiamato il metodo putAll() passando come parametro null
Record di esecuzione	Il metodo deve lanciare l'eccezione NullPointerException
Post-condizioni	La mappa non deve essere modificata

Nome metodo	MetodoPutAllCasoNormale()
Descrizione	Questo metodo esegue il test di MapAdapter.putAll(HCollection) in un caso d'uso normale
Variabili di esecuzione	Vi e' una variabile aggiuntiva HMap secondaMappa usata come parametro
Pre-condizioni	La mappa deve essere vuota
Casi di test	Viene richiamato il metodo putAll() passando come parametro:

	1. una mappa vuota 2. una mappa contenente delle entry 3. una mappa contenente entry gia' inserite
Record di esecuzione	Si deve ottenere la seguente situazione: 1. la mappa non viene modificata 2. la mappa deve contenere le entry presenti nella mappa di partenza 3. la mappa conterra' le entry nuove e in caso di entry gia' inserite queste verranno aggiornate al nuovo valore
Post-condizioni	Non vi sono post-condizioni particolari

Nome metodo	MetodoContainsKeyCasoLimite()
Descrizione	Questo metodo esegue il test di MapAdapter.containsKey(Object) in un caso limite
Variabili di esecuzione	Non vi sono variabili aggiuntive per questo metodo
Pre-condizioni	La mappa deve essere vuota
Casi di test	Viene richiamato il metodo containsKey() passando come parametro null
Record di esecuzione	Il metodo deve lanciare l'eccezione NullPointerException
Post-condizioni	La mappa non deve essere modificata

Nome metodo	MetodoContainsKeyCasoNormale()
Descrizione	Questo metodo esegue il test di MapAdapter.containsKey(Object) in un caso d'uso normale
Variabili di esecuzione	Non vi sono variabili aggiuntive per questo metodo
Pre-condizioni	La mappa deve essere popolata con delle entry a scelta
Casi di test	Viene richiamato il metodo containsKey() passando come parametro: 1. una chiave presente nella mappa 2. una chiave non presente nella mappa
Record di esecuzione	Il metodo deve ritornare: 1. true 2. false
Post-condizioni	La mappa non deve essere modificata

Nome metodo	MetodoContainsValueCasoLimite()
Descrizione	Questo metodo esegue il test di MapAdapter.containsValue(Object) in un caso limite
Variabili di esecuzione	Non vi sono variabili aggiuntive per questo metodo
Pre-condizioni	La mappa deve essere vuota
Casi di test	Viene richiamato il metodo containsValue() passando come parametro null

Record di esecuzione	Il metodo deve lanciare l’eccezione NullPointerException
Post-condizioni	La mappa non deve essere modificata

Nome metodo	MetodoContainsValueCasoNormale()
Descrizione	Questo metodo esegue il test di MapAdapter.containsValue(Object) in un caso d’uso normale
Variabili di esecuzione	Non vi sono variabili aggiuntive per questo metodo
Pre-condizioni	La mappa deve essere popolata con delle entry a scelta
Casi di test	Viene richiamato il metodo containsValue() passando come parametro: 1. un valore presente nella mappa 2. un valore non presente nella mappa
Record di esecuzione	Il metodo deve ritornare: 1. true 2. false
Post-condizioni	La mappa non deve essere modificata

Nome metodo	MetodoGetCasoLimite()
Descrizione	Questo metodo esegue il test di MapAdapter.get(Object) in un caso limite
Variabili di esecuzione	Non vi sono variabili aggiuntive per questo metodo
Pre-condizioni	La mappa deve essere vuota
Casi di test	Viene richiamato il metodo get() passando come parametro null
Record di esecuzione	Il metodo deve lanciare l’eccezione NullPointerException
Post-condizioni	La mappa non deve essere modificata

Nome metodo	MetodoGetCasoNormale()
Descrizione	Questo metodo esegue il test di MapAdapter.get(Object) in un caso d’uso normale
Variabili di esecuzione	Non vi sono variabili aggiuntive per questo metodo
Pre-condizioni	La mappa deve essere popolata con delle entry a scelta
Casi di test	Viene richiamato il metodo get() passando come parametro: 1. una chiave presente nella mappa 2. una chiave non presente nella mappa
Record di esecuzione	Il metodo deve ritornare: 1. il valore corretto associato alla chiave 2. null
Post-condizioni	La mappa non deve essere modificata

Nome metodo	MetodoRemoveCasoLimite()
Descrizione	Questo metodo esegue il test di MapAdapter.remove(Object) in un caso limite
Variabili di esecuzione	Non vi sono variabili aggiuntive per questo metodo
Pre-condizioni	La mappa deve essere vuota
Casi di test	Viene richiamato il metodo remove() passando come parametro null
Record di esecuzione	Il metodo deve lanciare l'eccezione NullPointerException
Post-condizioni	La mappa non deve essere modificata

Nome metodo	MetodoRemoveCasoNormale()
Descrizione	Questo metodo esegue il test di MapAdapter.remove(Object) in un caso d'uso normale
Variabili di esecuzione	Non vi sono variabili aggiuntive per questo metodo
Pre-condizioni	La mappa deve essere popolata con delle entry a scelta
Casi di test	Viene richiamato il metodo remove() passando come parametro: 1. una chiave presente nella mappa 2. una chiave non presente nella mappa
Record di esecuzione	Il metodo deve ritornare: 1. il valore corretto associato alla chiave e l'entry deve essere eliminata dalla mappa 2. null e la mappa non deve essere modificata
Post-condizioni	Non vi sono post-condizioni particolari

Nome metodo	MetodoEntrySet()
Descrizione	Questo metodo esegue il test di MapAdapter.entrySet() in modo piu' completo possibile, pur non scrivendo un metodo di test per ogni funzionalita proposta dalla classe EntrySetView
Variabili di esecuzione	Vi e' una variabile aggiuntiva HSet setDiEntry usata per memorizzare il valore ritornato dal metodo entrySet, Hcollection collezione usata come parametro da passare ad alcuni metodi del set e un iteratore HIterator itth.
Pre-condizioni	La mappa deve essere vuota
Casi di test	Viene richiamato il metodo entrySet() e memorizzato il valore nella variabile setDiEntry, in seguito viene verificato il funzionamento dei metodi del set: 1. contains() con parametro null 2. containsAll() con parametro null 3. remove() con parametro null 4. removeAll() con parametro null 5. retainAll() con parametro null 6. iterator.hasNext() con mappa vuota 7. iterator.next() con mappa vuota 8. iterator.remove() senza aver chiamato prima iterator.next()

	A questo punto viene popolata la mappa e viene verificato: 9. add() con parametro qualsiasi 10. addAll() con parametro qualsiasi 11. contains() con entry valida A questo punto viene popolata la variabile collezione con delle entry contenute nella mappa e viene verificato: 12. retainAll() 13. remove() 14. toArray() 15. toArray(Object[]) 16. removeAll() i metodi size, clear e isEmpty vengono usati e dunque testati durante tutto il processo
Record di esecuzione	Il metodo deve: 1. ritornare false 2. lanciare l'eccezione NullPointerException 3. ritornare false 4. lanciare l'eccezione NullPointerException 5. lanciare l'eccezione NullPointerException 6. ritornare false 7. lanciare l'eccezione NoSuchElementException 8. lanciare l'eccezione IllegalStateException 9. ritornare false 10. ritornare false 11. ritornare true 12. true se viene rimosso almeno un elemento, false altrimenti 13. true se viene passata una chiave contenuta nella mappa, false altrimenti 14. deve ritornare un array delle entry 15. deve ritornare un array delle entry 16. true se viene rimosso almeno un elemento, false altrimenti
Post-condizioni	La mappa deve essere modificata di conseguenza al set in ogni chiamata a metodo di questa classe

Nome metodo	MetodoKeySet()
Descrizione	Questo metodo esegue il test di MapAdapter.keySet() in modo piu' completo possibile, pur non scrivendo un metodo di test per ogni funzionalita proposta dalla classe KeySetView
Variabili di esecuzione	Vi e' una variabile aggiuntiva HSet setDiKey usata per memorizzare il valore ritornato dal metodo keySet, Hcollection collezione usata come parametro da passare ad alcuni metodi del set e un iteratore HIterator itth.
Pre-condizioni	La mappa deve essere vuota
Casi di test	Viene richiamato il metodo keySet() e memorizzato il valore nella variabile setDiKey, in seguito viene verificato il funzionamento dei metodi del set: 1. contains() con parametro null 2. containsAll() con parametro null

	3. remove() con parametro null 4. removeAll() con parametro null 5. retainAll() con parametro null 6. iterator.hasNext() con mappa vuota 7. iterator.next() con mappa vuota 8. iterator.remove() senza aver chiamato prima iterator.next() A questo punto viene popolata la mappa e viene verificato: 9. add() con parametro qualsiasi 10. addAll() con parametro qualsiasi 11. contains() con chiave valida A questo punto viene popolata la variabile collezione con delle chiavi contenute nella mappa e viene verificato: 12. retainAll() 13. remove() 14. toArray() 15. toArray(Object[]) 16. removeAll() i metodi size, clear e isEmpty vengono usati e dunque testati durante tutto il processo
Record di esecuzione	Il metodo deve: 1. ritornare false 2. lanciare l'eccezione NullPointerException 3. ritornare false 4. lanciare l'eccezione NullPointerException 5. lanciare l'eccezione NullPointerException 6. ritornare false 7. lanciare l'eccezione NoSuchElementException 8. lanciare l'eccezione IllegalStateException 9. ritornare false 10. ritornare false 11. ritornare true 12. true se viene rimosso almeno un elemento, false altrimenti 13. true se viene passata una chiave contenuta nella mappa, false altrimenti 14. deve ritornare un array delle chiavi 15. deve ritornare un array delle chiavi 16. true se viene rimosso almeno un elemento, false altrimenti
Post-condizioni	La mappa deve essere modificata di conseguenza al set in ogni chiamata a metodo di questa classe

Nome metodo	MetodoValues()
Descrizione	Questo metodo esegue il test di MapAdapter.values() in modo piu' completo possibile, pur non scrivendo un metodo di test per ogni funzionalita proposta dalla classe ValueCollectionView
Variabili di esecuzione	Vi e' una variabile aggiuntiva HCollection collDiValues usata per memorizzare il valore ritornato dal metodo values, Hcollection collezione usata come parametro da passare ad alcuni metodi del set e un iteratore

	HIterator itth.
Pre-condizioni	La mappa deve essere vuota
Casi di test	Viene richiamato il metodo values() e memorizzato il valore nella variabile collDiValues, in seguito viene verificato il funzionamento dei metodi della collection: 1. contains() con parametro null 2. containsAll() con parametro null 3. remove() con parametro null 4. removeAll() con parametro null 5. retainAll() con parametro null 6. iterator.hasNext() con mappa vuota 7. iterator.next() con mappa vuota 8. iterator.remove() senza aver chiamato prima iterator.next() A questo punto viene popolata la mappa e viene verificato: 9. add() con parametro qualsiasi 10. addAll() con parametro qualsiasi 11. contains() con valore valido A questo punto viene popolata la variabile collezione con dei valori contenuti nella mappa e viene verificato: 12. retainAll() 13. remove() 14. toArray() 15. toArray(Object[]) 16. removeAll() i metodi size, clear e isEmpty vengono usati e dunque testati durante tutto il processo
Record di esecuzione	Il metodo deve: 1. ritornare false 2. lanciare l’eccezione NullPointerException 3. ritornare false 4. lanciare l’eccezione NullPointerException 5. lanciare l’eccezione NullPointerException 6. ritornare false 7. lanciare l’eccezione NoSuchElementException 8. lanciare l’eccezione IllegalStateException 9. ritornare false 10. ritornare false 11. ritornare true 12. true se viene rimosso almeno un elemento, false altrimenti 13. true se viene passata una chiave contenuta nella mappa, false altrimenti 14. deve ritornare un array di entry 15. deve ritornare un array di entry 16. true se viene rimosso almeno un elemento, false altrimenti
Post-condizioni	La mappa deve essere modificata di conseguenza al set in ogni chiamata a metodo di questa classe

ListAdapterTest

Sommario:

Questa classe serve a testare ListAdapter e tutte le sottoclassi di supporto che essa usa per funzionare correttamente, ad eccezione di SubListView per la quale ho creato una classe di test apposita.

I test svolti sono:

- metodoHashCode()
- metodoEquals()
- metodoToArray1()
- metodoToArray2CasiLimite()
- metodoToArray2CasoNormale()
- metodoContains()
- metodoContainsAllCasiLimite()
- metodoContainsAllCasoNormale()
- metodoAdd()
- metodoAddAtCasiLimite()
- metodoAddAtCasoNormale()
- metodoAddAllCasiLimite()
- metodoAddAllCasoNormale()
- metodoAddAllAtCasiLimite()
- metodoAddAllAtCasoNormale()
- metodoIndexOf()
- metodoLastIndexOf()
- metodoGetCasiLimite()
- metodoGetCasoNormale()
- metodoSetCasiLimite()
- metodoSetCasoNormale()
- metodoRemove()
- metodoRemoveAtCasiLimite()
- metodoRemoveAtCasoNormale()
- metodoRemoveAllCasiLimite()
- metodoRemoveAllCasoNormale()
- metodoRetainAllCasoLimite()
- metodoRetainAllCasoNormale()
- metodoIterator()
- metodoListIterator()
- metodoListIteratorAt()
- metodoSubListCasiLimite()
- i metodi size, clear e isEmpty vengono usati e dunque testati durante tutto il processo, non vi sono dei test appositi per questi tre metodi

Design della suite di test:
I test per ogni metodo, eccetto per alcuni, sono divisi in casi limite e casi normali. In questo modo e' stato possibile testare separatamente il comportamento dei metodi nelle due situazioni, questo si rivela molto utile in fase di sviluppo.

Variabili di esecuzione:
Vi sono delle variabili comuni ai metodi di test:
- Hlist miaLista: un istanza di tipo ListAdapter su cui richiamare i metodi da test
- Vector vettoreObj: un vettore di supporto, popolato con oggetti di tipo diverso tra di loro, e' usato per inserire valori nelle liste all'interno dei test

Metodi:

Nome metodo	MetodoHashCode()
Descrizione	Questo metodo esegue il test di ListAdapter.hashCode()
Variabili di esecuzione	Vi e' una variabile aggiuntiva HList altraLista
Pre-condizioni	Entrambe le liste devono essere vuote
Casi di test	Viene richiamato il metodo hashCode() su due liste diverse in diverse situazioni: 1. entrambe le liste sono vuote 2. entrambe le liste vengono popolate con gli stessi oggetti contenuti il vettoreObj 3. dopo aver eliminato e reinserito un oggetto in una delle due liste
Record di esecuzione	Si deve ottenere la seguente situazione: 1. le liste hanno hashcode 0 2. le liste hanno lo stesso hashcode 3. le liste non hanno piu' lo stesso hashcode (l'ordine e' importante)
Post-condizioni	Non vi sono post-condizioni particolari

Nome metodo	MetodoEquals()
Descrizione	Questo metodo esegue il test di ListAdapter.equals(Object)
Variabili di esecuzione	Vi e' una variabile aggiuntiva HList altraLista
Pre-condizioni	Entrambe le liste devono essere vuote
Casi di test	Viene richiamato il metodo equals() confrontando la lista con: 1. il parametro null 2. un elemento di tipo diverso da HList Viene in seguito richiamato il metodo equals tra due liste diverse nelle seguenti situazioni: 3. entrambe le liste sono vuote 4. una sola lista viene popolata con gli oggetti contenuti il vettoreObj 5. entrambe le liste popolate con gli oggetti contenuti il vettoreObj 6. dopo aver eliminato e reinserito un oggetto in una delle due liste
Record di esecuzione	Il metodo deve ritornare: 1. false

	2. false 3. true 4. false 5. true 6. false (l’ordine e’ importante)
Post-condizioni	Non vi sono post-condizioni particolari

Nome metodo	MetodoToArray1()
Descrizione	Questo metodo esegue il test di ListAdapter.toArray()
Variabili di esecuzione	Non vi sono variabili aggiuntive per questo metodo
Pre-condizioni	La lista deve essere vuota
Casi di test	Viene richiamato il metodo toArray() sulla lista in diverse situazioni: 1. la lista e’ vuota 2. la lista e’ popolata con degli oggetto
Record di esecuzione	Il metodo deve ritornare: 1. un array vuoto 2. un array della stessa dimensione della lista e che l'array contenga tutti gli elementi contenuti nella lista
Post-condizioni	La lista non deve essere modificata

Nome metodo	MetodoToArray2CasiLimite()
Descrizione	Questo metodo esegue il test di ListAdapter.toArray(Object[]) in casi limite
Variabili di esecuzione	Non vi sono variabili aggiuntive per questo metodo
Pre-condizioni	La lista deve essere vuota
Casi di test	Viene richiamato il metodo toArray() passando come parametro: 1. null 2. un’array di tipo non congruente con gli oggetti all’interno della lista
Record di esecuzione	Il metodo deve lanciare l’eccezione: 1. NullPointerException 2. ArrayStoreException
Post-condizioni	La lista non deve essere modificata

Nome metodo	MetodoToArray2CasoNormale()
Descrizione	Questo metodo esegue il test di ListAdapter.toArray(Object[]) in un caso d’uso normale
Variabili di esecuzione	Non vi sono variabili aggiuntive per questo metodo
Pre-condizioni	La lista deve essere popolata con degli oggetti a scelta
Casi di test	Viene richiamato il metodo toArray() passando come parametro:

	1. un’array di dimensione uguale alla lista 2. un’array di dimensione maggiore alla lista 3. un’array di dimensione minore alla lista
Record di esecuzione	Il metodo deve : 1. inserire gli elementi nel vettore passato come parametro e ritornarlo 1. inserire gli elementi nel vettore passato come parametro, impostare l’elemento in posizione [dimensioneLista] a null e ritornarlo 1. creare un nuovo array, inserendo al suo interno gli elementi e ritornarlo
Post-condizioni	Non vi sono post-condizioni particolari

Nome metodo	MetodoContains()
Descrizione	Questo metodo esegue il test di ListAdapter.contains(Object)
Variabili di esecuzione	Non vi sono variabili aggiuntive per questo metodo
Pre-condizioni	La lista deve essere popolata con degli oggetti a scelta
Casi di test	Viene richiamato il metodo contains() passando come parametro: 1. un valore presente nella mappa 2. un valore non presente nella mappa
Record di esecuzione	Il metodo deve ritornare: 1. true 2. false
Post-condizioni	La lista non deve essere modificata

Nome metodo	MetodoContainsAllCasoLimite()
Descrizione	Questo metodo esegue il test di ListAdapter.containsAll(HCollection) in un caso limite
Variabili di esecuzione	Non vi sono variabili aggiuntive per questo metodo
Pre-condizioni	La lista deve essere vuota
Casi di test	Viene richiamato il metodo containsAll() passando come parametro null
Record di esecuzione	Il metodo deve lanciare l’eccezione NullPointerException
Post-condizioni	La lista non deve essere modificata

Nome metodo	MetodoContainsAllCasoNormale()
Descrizione	Questo metodo esegue il test di ListAdapter.containsAll(HCollection) in un caso d’uso normale
Variabili di esecuzione	Vi e’ una variabile aggiuntiva Hcollection coll, usata come collezione da passare al metodo
Pre-condizioni	La lista deve essere popolata con degli oggetti a scelta
Casi di test	Viene richiamato il metodo containsAll() passando come parametro:

	1. una collezione vuota 2. una collezione contenente oggetti inseriti nella lista 3. una collezione contenente oggetti non inseriti
Record di esecuzione	Il metodo deve ritornare: 1. true 2. false
Post-condizioni	La lista non deve essere modificata

Nome metodo	MetodoAdd()
Descrizione	Questo metodo esegue il test di ListAdapter.add(Object)
Variabili di esecuzione	Non vi sono variabili aggiuntive per questo metodo
Pre-condizioni	La lista deve essere vuota
Casi di test	Viene richiamato diverse volte il metodo add() passando come parametro un oggetto qualsiasi
Record di esecuzione	Il metodo deve ritornare true
Post-condizioni	La lista deve essere popolata in ordine con gli oggetti inseriti

Nome metodo	MetodoAddAllCasoLimite()
Descrizione	Questo metodo esegue il test di ListAdapter.addAll(HCollection) in un caso limite
Variabili di esecuzione	Non vi sono variabili aggiuntive per questo metodo
Pre-condizioni	La lista deve essere vuota
Casi di test	Viene richiamato il metodo addAll() passando come parametro null
Record di esecuzione	Il metodo deve lanciare l’eccezione NullPointerException
Post-condizioni	La lista non deve essere modificata

Nome metodo	MetodoAddAllCasoNormale()
Descrizione	Questo metodo esegue il test di ListAdapter.addAll(HCollection) in un caso d’uso normale
Variabili di esecuzione	Vi e’ una variabile aggiuntiva Hcollection coll, usata come collezione da passare al metodo
Pre-condizioni	La lista deve essere vuota
Casi di test	Viene richiamato il metodo addAll() passando come parametro: 1. una collezione vuota 2. una collezione contenente oggetti qualsiasi
Record di esecuzione	Il metodo deve ritornare: 1. false e la lista non deve essere modificata

	2. true e la lista deve contenere gli oggetti inseriti nell'ordine corretto
Post-condizioni	Non vi sono post-condizioni particolari

Nome metodo	MetodoAddAtCasoLimite()
Descrizione	Questo metodo esegue il test di ListAdapter.add(int,Object) in un caso limite
Variabili di esecuzione	Non vi sono variabili aggiuntive per questo metodo
Pre-condizioni	La lista deve essere vuota
Casi di test	Viene richiamato il metodo add() passando come parametro; 1. un indice minore di 0 2. un indice maggiore della dimensione della lista
Record di esecuzione	Il metodo deve lanciare in entrambi i casi l'eccezione NullPointerException
Post-condizioni	La lista non deve essere modificata

Nome metodo	MetodoAddAtCasoNormale()
Descrizione	Questo metodo esegue il test di ListAdapter.add(int,Object) in un caso d'uso normale
Variabili di esecuzione	Non vi sono variabili aggiuntive per questo metodo
Pre-condizioni	La lista deve essere vuota
Casi di test	Viene richiamato il metodo add() passando come parametro: 1. un indice 0 2. un indice compreso tra 0 e la dimensione della lista 2. un indice pari alla dimensione della lista
Record di esecuzione	Si deve ottenere la seguente situazione: 1. l'oggetto viene inserito all'inizio della lista 2. l'oggetto viene inserito nel punto richiesto 3. l'oggetto viene inserito alla fine della lista
Post-condizioni	Non vi sono post-condizioni particolari

Nome metodo	MetodoAddAllAtCasoLimite()
Descrizione	Questo metodo esegue il test di ListAdapter.addAll(int,HCollection) in un caso limite
Variabili di esecuzione	Non vi sono variabili aggiuntive per questo metodo
Pre-condizioni	La lista deve essere vuota
Casi di test	Viene richiamato il metodo addAll() passando come parametro: 1. una collezione null 2. un indice minore di 0 3. un indice maggiore della dimensione della lista
Record di esecuzione	Il metodo deve lanciare l'eccezione:

	1. NullPointerException 2. IndexOutOfBoundException 3. IndexOutOfBoundException
Post-condizioni	La lista non deve essere modificata

Nome metodo	MetodoAddAllAtCasoNormale()
Descrizione	Questo metodo esegue il test di ListAdapter.addAll(int,HCollection) in un caso d'uso normale
Variabili di esecuzione	Vi e' una variabile aggiuntiva Hcollection coll, usata come collezione da passare al metodo
Pre-condizioni	La lista deve essere popolata con degli oggetti a scelta
Casi di test	Viene richiamato il metodo addAll() passando come parametro: 1. una collezione vuota 2. una collezione contenente oggetti qualsiasi e un indice compreso tra 0 e la dimensione della lista
Record di esecuzione	Il metodo deve ritornare: 1. false e la lista non deve essere modificata 2. true e la lista deve contenere gli oggetti inseriti nell'ordine e nella posizione corretta
Post-condizioni	Non vi sono post-condizioni particolari

Nome metodo	MetodoIndexOf()
Descrizione	Questo metodo esegue il test di ListAdapter.indexOf(Object)
Variabili di esecuzione	Non vi sono variabili aggiuntive per questo metodo
Pre-condizioni	La lista deve essere popolata con degli oggetti a scelta
Casi di test	Viene richiamato diverse volte il metodo indexOf() passando come parametro: 1. un oggetto presente e duplicato nella lista 2. un oggetto non presente nella lista
Record di esecuzione	Il metodo deve ritornare: 1. l'indice del primo oggetto contenuto nella lista avente quel valore (primo nell'ordine della lista) 2. -1
Post-condizioni	La lista non deve essere modificata

Nome metodo	MetodoLastIndexOf()
Descrizione	Questo metodo esegue il test di ListAdapter.lastIndexOf(Object)
Variabili di esecuzione	Non vi sono variabili aggiuntive per questo metodo
Pre-condizioni	La lista deve essere popolata con degli oggetti a scelta

Casi di test	Viene richiamato diverse volte il metodo lastIndexOf() passando come parametro: 1. un oggetto presente e duplicato nella lista 2. un oggetto non presente nella lista
Record di esecuzione	Il metodo deve ritornare: 1. l'indice dell'ultimo oggetto contenuto nella lista avente quel valore (primo nell'ordine della lista) 2. -1
Post-condizioni	La lista non deve essere modificata

Nome metodo	MetodoGetCasiLimite()
Descrizione	Questo metodo esegue il test di ListAdapter.get(Object) in dei casi limite
Variabili di esecuzione	Non vi sono variabili aggiuntive per questo metodo
Pre-condizioni	La lista deve essere vuota
Casi di test	Viene richiamato il metodo get() passando come parametro: 1. un indice minore di 0 2. un indice maggiore o uguali alla dimensione della lista
Record di esecuzione	Il metodo in entrambi i casi deve lanciare l'eccezione IndexOutOfBoundException
Post-condizioni	La lista non deve essere modificata

Nome metodo	MetodoGetCasoNormale()
Descrizione	Questo metodo esegue il test di ListAdapter.get(Object) in un caso d'uso normale
Variabili di esecuzione	Non vi sono variabili aggiuntive per questo metodo
Pre-condizioni	La lista deve essere popolata con degli oggetti a scelta
Casi di test	Viene richiamato il metodo get() passando come parametro indici compresi tra 0 e la dimensione della lista
Record di esecuzione	Il metodo deve ritornare gli oggetti che sono memorizzati in quella posizione all'interno della lista
Post-condizioni	La lista non deve essere modificata

Nome metodo	MetodoSetCasiLimite()
Descrizione	Questo metodo esegue il test di ListAdapter.set(int,Object) in dei casi limite
Variabili di esecuzione	Non vi sono variabili aggiuntive per questo metodo
Pre-condizioni	La lista deve essere vuota
Casi di test	Viene richiamato il metodo set() passando come parametro: 1. un indice minore di 0

	2. un indice maggiore o uguali alla dimensione della lista
Record di esecuzione	Il metodo in entrambi i casi deve lanciare l'eccezione IndexOutOfBoundException
Post-condizioni	La lista non deve essere modificata

Nome metodo	MetodoSetCasoNormale()
Descrizione	Questo metodo esegue il test di ListAdapter.set(int,Object) in un caso d'uso normale
Variabili di esecuzione	Non vi sono variabili aggiuntive per questo metodo
Pre-condizioni	La lista deve essere popolata con degli oggetti a scelta
Casi di test	Viene richiamato il metodo set() passando come parametro indici compresi tra 0 e la dimensione della lista
Record di esecuzione	Il metodo deve ritornare gli oggetti che erano memorizzati in quella posizione all'interno della lista e sostituirli con quelli passati come parametro
Post-condizioni	Non vi sono post-condizioni particolari

Nome metodo	MetodoRemove()
Descrizione	Questo metodo esegue il test di ListAdapter.remove(Object)
Variabili di esecuzione	Non vi sono variabili aggiuntive per questo metodo
Pre-condizioni	La lista deve essere popolata con degli oggetti a scelta
Casi di test	Viene richiamato il metodo remove() passando come parametro: 1. un oggetto presente nella mappa 2. un oggetto non presente nella mappa
Record di esecuzione	Il metodo deve ritornare: 1. true e l'oggetto deve essere eliminato dalla lista 2. false e la lista non deve essere modificata
Post-condizioni	Non vi sono post-condizioni particolari

Nome metodo	MetodoRemoveAtCasiLimite()
Descrizione	Questo metodo esegue il test di ListAdapter.remove(int) in dei casi limite
Variabili di esecuzione	Non vi sono variabili aggiuntive per questo metodo
Pre-condizioni	La lista deve essere vuota
Casi di test	Viene richiamato il metodo remove() passando come parametro: 1. un indice minore di 0 2. un indice maggiore o uguali alla dimensione della lista
Record di esecuzione	Il metodo deve lanciare in entrambi i casi l'eccezione IndexOutOfBoundException

Post-condizioni	La lista non deve essere modificata

Nome metodo	MetodoRemoveAtCasoNormale()
Descrizione	Questo metodo esegue il test di ListAdapter.remove(int) in un caso d'uso normale
Variabili di esecuzione	Non vi sono variabili aggiuntive per questo metodo
Pre-condizioni	La lista deve essere popolata con degli oggetti a scelta
Casi di test	Viene richiamato il metodo remove() passando come parametro indici compresi tra 0 e la dimensione della lista
Record di esecuzione	Il metodo deve ritornare gli oggetti che erano memorizzati in quella posizione all'interno della lista e rimuovere gli oggetti dalla lista
Post-condizioni	Non vi sono post-condizioni particolari

Nome metodo	MetodoRemoveAllCasoLimite()
Descrizione	Questo metodo esegue il test di ListAdapter.removeAll(HCollection) in un caso limite
Variabili di esecuzione	Non vi sono variabili aggiuntive per questo metodo
Pre-condizioni	La lista deve essere vuota
Casi di test	Viene richiamato il metodo removeAll() passando come parametro null
Record di esecuzione	Il metodo deve lanciare l'eccezione NullPointerException
Post-condizioni	La lista non deve essere modificata

Nome metodo	MetodoRemoveAllCasoNormale()
Descrizione	Questo metodo esegue il test di ListAdapter.removeAll(HCollection) in un caso d'uso normale
Variabili di esecuzione	Vi e' una variabile aggiuntiva Hcollection coll, usata come collezione da passare al metodo
Pre-condizioni	La lista deve essere popolata con degli oggetti a scelta
Casi di test	Viene richiamato il metodo removeAll() passando come parametro: 1. una collezione vuota 2. una collezione contenente oggetti inseriti nella lista 3. una collezione contenente oggetti non inseriti
Record di esecuzione	Il metodo deve ritornare: 1. true e la lista non deve essere modificata 2. false e devono essere rimossi dalla lista gli oggetti contenuti in coll 3. false se in coll non ci sono oggetti contenuti nella lista, true altrimenti, devono essere rimossi dalla lista solo gli oggetti giusti
Post-condizioni	Non vi sono post-condizioni particolari

Nome metodo	MetodoRetainAllCasoLimite()
Descrizione	Questo metodo esegue il test di ListAdapter.retainAll(HCollection) in un caso limite
Variabili di esecuzione	Non vi sono variabili aggiuntive per questo metodo
Pre-condizioni	La lista deve essere vuota
Casi di test	Viene richiamato il metodo retainAll() passando come parametro null
Record di esecuzione	Il metodo deve lanciare l'eccezione NullPointerException
Post-condizioni	La lista non deve essere modificata

Nome metodo	MetodoRetainAllCasoNormale()
Descrizione	Questo metodo esegue il test di ListAdapter.retainAll(HCollection) in un caso d'uso normale
Variabili di esecuzione	Vi e' una variabile aggiuntiva Hcollection coll, usata come collezione da passare al metodo
Pre-condizioni	La lista deve essere popolata con degli oggetti a scelta
Casi di test	Viene richiamato il metodo retainAll() passando come parametro: 1. una collezione vuota 2. una collezione contenente oggetti sia inseriti che non inseriti nella lista
Record di esecuzione	Il metodo deve ritornare: 1. true e la lista deve diventare vuota 2. true e devono essere rimossi dalla lista gli oggetti non contenuti in coll
Post-condizioni	Non vi sono post-condizioni particolari

Nome metodo	MetodoIterator()
Descrizione	Questo metodo esegue il test di ListAdapter.iterator() in diversi casi d'uso
Variabili di esecuzione	Vi e' un array aggiuntivo Object[] mioVettore, usato come supporto nel test della classe di tipo iteratore ritornata dal metodo
Pre-condizioni	La lista deve essere vuota
Casi di test	Viene richiamato il metodo iterator() e memorizzato in una variabile di tipo HIterator, in seguito va testato il funzionamento dei metodi della classe ritornata: 1. iterator.next() con lista vuota 2. iterator.remove() con lista vuota, senza chiamare prima iterator.next() A questo punto la lista deve essere popolata con degli oggetti a scelta e continua il test di altri metodi: 3. iterator.remove() dopo aver chiamato iterator.remove()

	4. iterator.hasNext() e iterator.next() con lista popolata 5. iterator.remove() dopo iterator.next() con lista popolata
Record di esecuzione	Il metodo deve: 1. lanciare l'eccezione NoSuchElementException e la lista non deve essere modificata 2. lanciare l'eccezione IllegalStateException e la lista non deve essere modificata 3. lanciare l'eccezione IllegalStateException e la lista non deve essere modificata 4. ritornare true(hasNext) e ritornare l'elemento successivo su cui iterare(next) 2. rimuovere l'elemento appena iterato dalla lista
Post-condizioni	La lista deve essere modificata in accordo con l'iteratore (remove)

Nome metodo	MetodoListIterator()
Descrizione	Questo metodo esegue il test di ListAdapter.listIterator() in diversi casi d'uso
Variabili di esecuzione	Non vi sono variabili aggiuntive per questo metodo
Pre-condizioni	La lista deve essere vuota
Casi di test	Viene richiamato il metodo listIterator() e memorizzato in una variabile di tipo HListIterator, in seguito va testato il funzionamento dei metodi della classe ritornata: 1. listiterator.next() e listiterator.previous() con lista vuota 2. listiterator.remove() e listiterator.set() con lista vuota, senza chiamare prima iterator.next() 3. listiterator.add() per popolare la lista 4. listiterator.remove() dopo aver chiamato iterator.remove() 5. listiterator.hasNext(), listiterator.next() e listiterator.nextindex() con lista popolata 6. listiterator.hasPrevious(), listiterator.previous() e listiterator.previousindex() con lista popolata 7. listiterator.set () dopo listiterator.next() e dopo listiterator.previous() con lista popolata 8. listiterator.remove() dopo listiterator.next() e dopo listiterator.previous() con lista popolata
Record di esecuzione	Il metodo deve: 1. in entrambi i casi lanciare l'eccezione NoSuchElementException e la lista non deve essere modificata 2. in entrambi i casi lanciare l'eccezione IllegalStateException e la lista non deve essere modificata 3. aggiungere l'oggetto passato come parametro alla lista 4. lanciare l'eccezione IllegalStateException e la lista non deve essere modificata 5. ritornare true(hasNext), ritornare l'elemento successivo su cui iterare(next) e ritornare l'indice dell'elemento successivo(nextindex)

	6. ritornare true(hasPrevious), ritornare l'elemento precedente su cui iterare(previous) e ritornare l'indice dell'elemento precedente(previous) 7. modificare l'ultimo elemento appena iterato 8. rimuovere l'ultimo elemento appena iterato
Post-condizioni	La lista deve essere modificata in accordo con l'iteratore (remove e set)

Nome metodo	MetodoListIteratorAt()
Descrizione	Questo metodo esegue il test di ListAdapter.listIterator(int) in diversi casi d'uso
Variabili di esecuzione	Non vi sono variabili aggiuntive per questo metodo
Pre-condizioni	La lista deve essere vuota
Casi di test	Viene richiamato il metodo listIterator() passando come parametro: 1. un indice minore di 0 2. un indice maggiore della dimensione della lista Viene in seguito popolata la lista e richiamato il metodo listIterator passando come parametro un indice compreso tra 0 e la dimensione della lista Viene quindi testato il funzionamento dei metodi della classe ritornata: 3. listiterator.hasNext(), listiterator.next() e listiterator.nextindex() con lista popolata 4. listiterator.hasPrevious(), listiterator.previous() e listiterator.previousindex() Il resto e' gia' testato nel metodo di test precedente
Record di esecuzione	Il metodo deve: 1. 2. in entrambi i casi lanciare l'eccezione IndexOutOfBoundException e la lista non deve essere modificata 3. ritornare true(hasNext), ritornare l'elemento successivo su cui iterare(next) e ritornare l'indice dell'elemento successivo(nextindex) 4. ritornare true(hasPrevious), ritornare l'elemento precedente su cui iterare(previous) e ritornare l'indice dell'elemento precedente(previous)
Post-condizioni	Non vi sono post-condizioni particolari

Nome metodo	MetodoSubListCasiLimite()
Descrizione	Questo metodo esegue il test di ListAdapter.subList(int,int) in diversi casi limite
Variabili di esecuzione	Non vi sono variabili aggiuntive per questo metodo
Pre-condizioni	La lista deve essere vuota
Casi di test	Viene richiamato il metodo subList() passando come parametri: 1. fromIndex minore di 0 2. toIndex maggiore della dimensione della lista 3. fromIndex e toIndex compresi tra 0 e la dimensione della lista, ma fromIndex maggiore di toIndex
Record di esecuzione	Il metodo deve in tutti i casi lanciare l'eccezione IndexOutOfBoundException

Post-condizioni	La lista non deve essere modificata

SubListViewTest

Sommario:

Questa classe serve a testare SubListView e la classe ListAdapterIterator di supporto che essa usa per funzionare correttamente

I test svolti sono gli stessi di ListAdapter, vengono elencati in seguito solo con il dettaglio delle differenze e non le intere tabelle

Design della suite di test:
I test per ogni metodo sono costruiti in modo molto simile a quelli svolti per ListAdapter tenendo pero' anche conto che i metodi richiamati su subList devono avere un effetto anche sulla lista di partenza (post-condizione).
Ho quindi preferito non inserire una tabella per ogni metodo di test, ma piuttosto descrivere solo le piccole differenze di svolgimento e di scopo tra i metodi di test svolti su ListAdapter e su SubListView. Questo mi permette di mantenere il documento piu' contenuto e risparmiare piu' di una decina di pagine contenenti tabelle molto simili a quelle gia' descritte senza sorvolare sui dettagli implementativi importanti.

Variabili di esecuzione:
Vi sono delle variabili comuni ai metodi di test:
- Hlist miaLista: un istanza di tipo ListAdapter, popolata con degli oggetti, su cui richiamare il metodo sublist()
- Hlist miaSubList: un istanza di tipo SublistView, ottenuta a partire da miaLista, su cui richiamare i metodi di test
- Vector vettoreObj: un vettore di supporto, popolato con degli oggetti, usato per inserire valori nelle liste all'interno dei test

Metodi:

I seguenti usano semplicement miaSubList (una sottolista vuota di miaLista) al posto di miaLista per eseguire il test
- metodoHashCode()
- metodoEquals()
- metodoToArray1()
- metodoToArray2CasiLimite()
- metodoToArray2CasoNormale()
- metodoIterator()
- metodoListIterator()
- metodoListIteratorAt()

All'interno dei seguenti metodi di test e' necessario verificare che, se miaSubList e' una sottolista di miaLista e inserisco elementi all'interno di miaSubList, questi siano visibili sia all'interno di miaSubList che di miaLista
- metodoContains()

- metodoContainsAllCasiLimite()
- metodoContainsAllCasoNormale()
- metodoAdd()

  Inoltre devo verificare che l'elemento aggiunto in una precisa posizione nella sottolista sia stato aggiunto nella posizione corretta anche nella lista di partenza
- metodoAddAtCasiLimite()
- metodoAddAtCasoNormale()

  Inoltre devo verificare che l'elemento aggiunto in una precisa posizione nella sottolista sia stato aggiunto nella posizione corretta anche nella lista di partenza
- metodoAddAllCasiLimite()
- metodoAddAllCasoNormale()

  Inoltre devo verificare che l'elemento aggiunto in una precisa posizione nella sottolista sia stato aggiunto nella posizione corretta anche nella lista di partenza
- metodoAddAllAtCasiLimite()
- metodoAddAllAtCasoNormale()

  Inoltre devo verificare che l'elemento aggiunto in una precisa posizione nella sottolista sia stato aggiunto nella posizione corretta anche nella lista di partenza

All'interno dei seguenti metodi di test e' necessario verificare che gli indici degli elementi di miaSubList corrispondano ai corretti indici all'interno di miaLista

- metodoIndexOf()
- metodoLastIndexOf()
- metodoGetCasiLimite()
- metodoGetCasoNormale()
- metodoSetCasiLimite()
- metodoSetCasoNormale()
- metodoRemoveAtCasiLimite()
- metodoRemoveAtCasoNormale()

All'interno dei seguenti metodi di test e' necessario verificare che sia possibile rimuovere solamente oggetti contenuti in miaSubList ma non in miaLista

- metodoRemove()
- metodoRemoveAllCasiLimite()
- metodoRemoveAllCasoNormale()
- metodoRetainAllCasoLimite()
- metodoRetainAllCasoNormale()

I metodi size, clear e isEmpty vengono usati e dunque testati durante tutto il processo, non vi sono dei test appositi per questi tre metodi.

Nome metodo	metodoSubList()
Descrizione	Questo metodo esegue il test di SubListView.subList(int,int)
Variabili di esecuzione	Non vi sono variabili aggiuntive per questo metodo
Pre-condizioni	La sottolista deve essere vuota
Casi di test	Viene richiamato il metodo subList() passando due parametri qualsiasi
Record di esecuzione	Il metodo deve in ogni caso ritornare null
Post-condizioni	Non vi sono post-condizioni particolari