LISTADAPTER

AddTest	
SUMMARY	Il test verifica il funzionamento del metodo add(Object)
TEST CASE DESIGN	Il test verifica la funzionalità del metodo sia in presenza di valori validi che in presenza di valori non validi
TEST DESCRIPTION	Creo una lista, applico il applico il metodo add a valori numerici e stringhe e poi verifico, usando un blocco try catch, che non puo' essere inserito null
PRE-CONDITION	/
POST-CONDITION	/
EXPECTED RESULT	verifico con get() che siano stati inseriti i valori e che aggiungendo null venga lanciato NullPointerException

Add1Test	
SUMMARY	Il test verifica il funzionamento del metodo add(int,Object)
TEST CASE DESIGN	Il test verifica la funzionalità del metodo sia in presenza di valori validi che in presenza di valori non validi. Controlla che gli elementi vengano inseriti nelle giuste posizioni
TEST DESCRIPTION	Creo una lista, applico il applico il metodo add a valori numerici e stringhe e poi verifico, usando un blocco try catch, che non puo' essere inserito null
PRE-CONDITION	/
POST-CONDITION	/
EXPECTED RESULT	verifico con get(int) che siano stati inseriti i valori (e che siano nelle corrette posizioni)e che aggiungendo null venga lanciato NullPointerException

AddAllTest	
SUMMARY	Il test verifica il funzionamento del metodo addAll(HClollection)
TEST CASE DESIGN	Il test verifica il corretto inserimento di una collezione di Object in coda alla list
TEST DESCRIPTION	Creo una lista, applico il metodo add ripetutamente, poi creo un'altra lista di supporto aggiungendo valori anche ad essa, applico il metodo addAll trattato alla seconda lista, passando come parametro la prima list
PRE-CONDITION	/
POST-CONDITION	/
EXPECTED RESULT	verifico con get(int) che siano stati inseriti i valori nella seconda lista e che siano nella corretta posizione (alla fine della lista). Controllo che prima dell'indice siano presenti i valori originari della seconda lista

AddAll2Test	
SUMMARY	Il test verifica il funzionamento del metodo addAll(int,HClollection)
TEST CASE DESIGN	Il test verifica il corretto inserimento di una collezione di Object ponendo attenzione all'inserimento dei vari elementi nella corretta posizione data dall'index passato come parametro
TEST DESCRIPTION	Creo una lista, applico il metodo add ripetutamente, poi creo un'altra lista di supporto aggiungendo valori anche ad essa, applico il metodo addAll trattato alla seconda lista, passando come parametro la prima list e un indice tale per cui i nuovi elementi verranno aggiunti in sequenza dallo stesso
PRE-CONDITION	/
POST-CONDITION	/
EXPECTED RESULT	verifico con get(int) che siano stati inseriti i valori nella seconda lista e che siano nella corretta posizione (in sequenza a partire dall'indice passato). Controllo che, prima dell'indice passato e dopo aver inserito gli elementi della collection, siano presenti i valori originari della seconda lista

ContainsAllTest	
SUMMARY	Il test verifica il funzionamento del metodo contains(Object) addAll(int,HClollection)
TEST CASE DESIGN	Il test verifica il corretto inserimento di una collezione di Object ponendo attenzione all'inserimento dei vari elementi nella corretta posizione data dall'index passato come parametro
TEST DESCRIPTION	Creo una lista, applico il metodo add ripetutamente, poi creo un'altra lista di supporto aggiungendo valori anche ad essa, applico il metodo containsAll trattato alla prima lista, passando come parametro la seconda lista. Verifico inoltre che un singolo elemento scelto sia presente nella lista con il metodo contains
PRE-CONDITION	/
POST-CONDITION	/
EXPECTED RESULT	Verifico nel primo assert che sia contenuto un elemento scelto presente nella lista, con il secondo verifico che la seconda lista sia interamente contenuta nella prima, con il terzo verifico che, dopo aver modificato la seconda lista, essa non sia piu' interamente contenuta nella prima lista.

HashCodeTest	
SUMMARY	Il test verifica il funzionamento del metodo hashCode() tramite confronto fra liste
TEST CASE DESIGN	Il test verifica che a hashCode uguali corrispondano liste uguali (sia la lunghezza che i singoli elementi che il loro ordine devono essere uguali)
TEST DESCRIPTION	Creo tre liste, applico il metodo add ripetutamente in modo che la prima e terza lista abbiano stessi elementi nello stesso ordine, la seconda lista presenta stessi elementi ma in ordine diverso. Infine viene aggiunto un valore alla terza lista in modo che abbia un valore diverso rispetto alla prima.
PRE-CONDITION	/
POST-CONDITION	/
EXPECTED RESULT	Verifico nel primo assert che la prima e seconda lista abbiano hashCode diversi (ordine diverso degli elementi), con il secondo che la prima e terza lista siano abbiano stesso hashCode, con il terzo che, aggiungendo alla terza lista un altro valore, le due presentino hashCode diversi

EqualsTest	
SUMMARY	Il test verifica il funzionamento del metodo Equals(Object) tramite confronto fra liste
TEST CASE DESIGN	Il test verifica che due liste sono uguali se sia la lunghezza che i singoli elementi che il loro ordine essere uguali
TEST DESCRIPTION	Creo tre liste, applico il metodo add ripetutamente in modo che la prima e terza lista abbiano stessi elementi nello stesso ordine, la seconda lista presenta stessi elementi ma in ordine diverso. Infine viene aggiunto un valore alla terza lista in modo che abbia un valore diverso rispetto alla prima.
PRE-CONDITION	/
POST-CONDITION	/
EXPECTED RESULT	Verifico nel primo assert che la prima e seconda lista siano diverse (ordine diverso degli elementi), con il secondo che la prima e terza lista siano uguali, con il terzo che, aggiungendo alla terza lista un altro valore, le due diventino diverse

IteratorTest	
SUMMARY	Il test verifica il funzionamento del metodo iterator().
TEST CASE DESIGN	Verifico che il metodo remove() eliminera' l'elemento successiovo a quello identificato con un if (iter.next()==”a”) (di conseguenza verifica anche next() e hasNext())
TEST DESCRIPTION	Creo una lista, applico il metodo add ripetutamente. Eseguo un ciclo while procedendo in avanti con l'iteratore finche' non raggiungo un valore deciso della lista. A quel punto eseguo remove (eliminando l'elemento successivo a quello identificato) .Quindi verifico che la dimensione sia decrementata, che l'elemento eliminato non sia piu' presente, che quelli non eliminati siano ancora presenti. Infine verifico che eseguendo remove due volte di fila, venga lanciato IllegalStateException()
PRE-CONDITION	/
POST-CONDITION	/
EXPECTED RESULT	Verifico nel primo assert che, a seguito del remove, la dimensione sia stata decrementata, con i seguenti tre che non sia piu' presente solo l'elemento successivo a quello identificato dall' if (vedere commento), con l'ultimo che eseguendo remove due volte di fila, venga lanciato IllegalStateException()

RemoveTest	
SUMMARY	Il test verifica il funzionamento del metodo Remove(Object)
TEST CASE DESIGN	Il test verifica che, rimuovendo da una lista un valore presente viene aggiornata la dimensione e viene ritornato il giusto booleano
TEST DESCRIPTION	Creo una lista, applico il metodo add ripetutamente. Verifico l'output di remove eliminando un valore presente o assente nella lista. Controllo la dimensione della lista dopo la rimozione
PRE-CONDITION	/
POST-CONDITION	/
EXPECTED RESULT	Verifico nel primo assert che, eliminando un elemento presente nella mappa venga restituito true, eliminando un valore assente nella mappa venga restituito false. Infine verifico che rimuovendo tutti gli elementi, la dimensione della lista diventi 0

Remove2Test	
SUMMARY	Il test verifica il funzionamento del metodo Remove(int)
TEST CASE DESIGN	Il test verifica che, rimuovendo da una lista un valore presente viene aggiornata la dimensione e viene ritornato il giusto oggetto
TEST DESCRIPTION	Creo una lista, applico il metodo add ripetutamente. Verifico l'output di remove eliminando un valore presente nella lista. Controllo la dimensione della lista dopo la rimozione
PRE-CONDITION	/
POST-CONDITION	/
EXPECTED RESULT	Verifico nel primo assert che, eliminando un elemento presente nella mappa venga restituito l'oggetto rimosso. Infine verifico che rimuovendo tutti gli elementi, la dimensione della lista diventi 0

RetainAllTest	
SUMMARY	Il test verifica il funzionamento del metodo RetainAll(HCollection)
TEST CASE DESIGN	Eseguo retainAll verificando la previsione dell'operazione e controllando dimensione finale e che siano contenuti valori opportuni.
TEST DESCRIPTION	Creo due liste e passo una delle due come parametro per retainAll applicato sull'altra. Verifico che siano presenti i valori previsti.
PRE-CONDITION	/
POST-CONDITION	/
EXPECTED RESULT	Verifico nei primi due assert che gli elementi previsti siano presenti nella lista, nel terzo che la dimensione della lista sia diminuita secondo previsione.

SetTest	
SUMMARY	Il test verifica il funzionamento del metodo set(int,Object)
TEST CASE DESIGN	Verifico che il metodo set sostituisca il valore opportuno presente nella lista senza modificare la dimensione
TEST DESCRIPTION	Creo una lista, aggiungo degli elementi. Poi sostituisco un elemento con set. Poi verifico che l'elemento precedentemente presente, ora e' assente
PRE-CONDITION	/
POST-CONDITION	/
EXPECTED RESULT	Verifico nel primo assert che il valore sia stato sostituito, nel secondo che la dimensione sia rimasta invariata a seguito del set (sostituzione in questo caso), nel terzo che l'elemento precedentemente presente non lo e' piu', nell'ultimo che applicando set con un indice pari a size() viene lanciato IndexOutOfBoundException()

ToArray2Test	
SUMMARY	Il test verifica il funzionamento del metodo toArray(Object[])
TEST CASE DESIGN	Verifico che il metodo restituisca un array di Object della giusta dimensione
TEST DESCRIPTION	Creo una lista, aggiungo degli elementi. Poi applico il metodo toArray in questione (e stampo i diversi elementi)
PRE-CONDITION	/
POST-CONDITION	/
EXPECTED RESULT	Verifico che la lunghezza dell'array corrisponda alla lunghezza della lista passata () in quanto essa e' di dimensione maggiore rispetto alla lista

SublistTest	
SUMMARY	Il test verifica il funzionamento del metodo sublist(int,int)
TEST CASE DESIGN	Verifico che passando fromIndex e toIndex uguali, non vi e' effetto sulla lista, passando invece il primo indice minore del secondo, e poi invocando clear sulla sublist, vengono eliminati dei valori anche dalla lista originaria
TEST DESCRIPTION	Creo una lista, aggiungo degli elementi. Creo due sottoliste, una che non contiene elementi (dato chedue indici passati sono uguali). Poi applico il metodo clear ad entrambe, prima sulla lista con nessun elemento
PRE-CONDITION	/
POST-CONDITION	/
EXPECTED RESULT	Verifico per entrambe le sublist che la dimensione e' zero a seguito dell'operazione clear, che la dimensione della lista non e' variata per la seconda sublist (e' variata per la prima).

SublistAddTest	
SUMMARY	Il test verifica il funzionamento del metodo add(Object) e add(int,Object) della sottoclasse Sublist
TEST CASE DESIGN	Verifico che applicando add(Object) e add(int,Object) la dimensione della sublist aumenta.
TEST DESCRIPTION	Creo una lista, aggiungo degli elementi. Creo una sublistapplico i due metodi add.
PRE-CONDITION	/
POST-CONDITION	/
EXPECTED RESULT	Verifico la dimensione della sublist prima e dopo aver inserito nuovi elementi. Verifico inoltre che un oggetto aggiunto sia effettivamente presente nella sublist.

SublistAddAllTest	
SUMMARY	Il test verifica il funzionamento del metodo addAll(HCollection) della sottoclasse Sublist
TEST CASE DESIGN	Verifico che applicando AddAll considerato alla sublist la dimensione della stessa aumenta e che gli elementi vengono inseriti in sequenza in coda .Verifico anche il booleano di ritorno.
TEST DESCRIPTION	Creo una lista, aggiungo degli elementi. Creo una sublist, applico il metodo addAll alla prima, passando come parametro la seconda lista.
PRE-CONDITION	/
POST-CONDITION	/
EXPECTED RESULT	Verifico nel primo assert che la dimensione della sublist sia aumentata dopo addAll, nel secondo provo che un elmento scelto di sublist e' stato inserito in coda alla sublist originaria.

SublistAddAll2Test	
SUMMARY	Il test verifica il funzionamento del metodo addAll(int,HCollection) della sottoclasse Sublist
TEST CASE DESIGN	Verifico che applicando AddAll considerato alla sublist la dimensione della stessa aumenta e che gli elementi vengono inseriti in sequenza a partire dall'indice passato. Verifico anche il booleano di ritorno.
TEST DESCRIPTION	Creo due liste, aggiungo degli elementi. Creo una sublist, applico il metodo addAll ad essa, passando come parametro la seconda lista
PRE-CONDITION	/
POST-CONDITION	/
EXPECTED RESULT	Verifico nel primo assert che applicando addAll passando la seconda lista vuota, il booleano di ritorno e' false, nel secondo che la dimensione della sublist e' aumentata, nel terzo che in che un determinato elemento della seconda lista e' stato inserito nella corretta posizione della sublist.

SublistClearTest	
SUMMARY	Il test verifica il funzionamento del metodo clear() della sottoclasse Sublist
TEST CASE DESIGN	Verifico che applicandoclear la dimensione della lista diminuisce e quella della sublist si azzera. Controllo che uno specifico elemento della lista, non presente in sublist, sia ancora presente nella lista.
TEST DESCRIPTION	Creo due liste, aggiungo degli elementi. Creo una sublist, aggiungo ad essa un elemento, applico il metodo clear.
PRE-CONDITION	/
POST-CONDITION	/
EXPECTED RESULT	Verifico nel primo assert che applicando clear la dimensione della sublist e' 0, nel secondo che la dimensione della lista e' diminuita della dimensione di sublist, nel terzo che un elemento contenuto non presente nella sublist, ma presente nella lista, sia ancora presente in essa

SublistContainsAllTest	
SUMMARY	Il test verifica il funzionamento del metodo contains(Object) e soprattutto del metodo containsAll(HCollection) della sottoclasse Sublist.
TEST CASE DESIGN	Verifico che, applicando il metodo contains e containsAll (passando come parametro la seconda lista) controllo l'effettiva presenza di un valore o di una collezione di valori nella sublist, ponendo attenzione al booleano restituito.
TEST DESCRIPTION	Creo due liste, aggiungo degli elementi. Creo una sublist, aggiungo ad essa un elemento, applico il metodo clear. Applico piu' verifiche.
PRE-CONDITION	/
POST-CONDITION	/
EXPECTED RESULT	Verifico nel primo assert che un elemento specifico della sublist sia presente, nel secondo che, aggiungendo un elemento gia' presente nella sublist, l'esito di contains non vari, nel terzo che vari invece se aggiungo alla seconda lista un valore non presente nella sublist. Infine con un blocco try e catch verifico che viene lanciato NullPointerException se viene passato come parametro null.

SublistEqualsTest	
SUMMARY	Il test verifica il funzionamento del metodo equals(Object) della sottoclasse Sublist.
TEST CASE DESIGN	Verifico che due liste (o sottoliste) sono uguali solo se hanno stessa dimensione, gli stessi elementi con ugual ordine
TEST DESCRIPTION	Creo quattro liste, aggiungo degli elementi. Confronto poi le liste con la sublist.
PRE-CONDITION	/
POST-CONDITION	/
EXPECTED RESULT	Verifico nel primo assert che sublist e la seconda lista non sono uguali perche hanno stessi elementi ma con ordine diverso, nel secondo che invece la sublist e la terza lista sono uguali, nel terzo che la sublist e la quarta lista sono diverse (un elemento diverso)

SublistHashCodeTest	
SUMMARY	Il test verifica il funzionamento del metodo hashCode() della sottoclasse Sublist.
TEST CASE DESIGN	Verifico che due liste hanno hashcode uguale se le due liste sono uguali.
TEST DESCRIPTION	Creo tre liste, aggiungo degli elementi. Confronto poi l'hashcode delle liste con quello della sublist.
PRE-CONDITION	/
POST-CONDITION	/
EXPECTED RESULT	Verifico nel primo assert che sublist e la seconda lista presentano hashcode diversi (ordine degli elementi diverso), nel secondo che l'hashcode di sublist e la terza lista e' uguale.

SublistLastIndexOfTest	
SUMMARY	Il test verifica il funzionamento del metodo lastIndexOf() della sottoclasse Sublist.
TEST CASE DESIGN	Verifico l'indice che viene restituito per due occorrenze dello stesso elemento nella sublist
TEST DESCRIPTION	Creo una lista, aggiungo degli elementi. Verifico che venga restituito l'indice piu' vicono alla coda per una certa occorrenza nella sublist.
PRE-CONDITION	/
POST-CONDITION	/
EXPECTED RESULT	Verifico nell'assert che, per due occorrenze nella sublist, venga restituito l'indice dell'elemento piu' vicino alla coda della stessa

SublistIndexOfTest	
SUMMARY	Il test verifica il funzionamento del metodo indexOf() della sottoclasse Sublist.
TEST CASE DESIGN	Verifico l'indice che viene restituito per due occorrenze dello stesso elemento nella sublist
TEST DESCRIPTION	Creo una lista, aggiungo degli elementi. Verifico che venga restituito l'indice piu' vicono alla testa per una certa occorrenza nella sublist.
PRE-CONDITION	/
POST-CONDITION	/
EXPECTED RESULT	Verifico nell'assert che, per due occorrenze nella sublist, venga restituito l'indice dell'elemento piu' vicino alla testa della stessa.

SublistRemoveTest	
SUMMARY	Il test verifica il funzionamento dei metodi remove(Object) e romove(int) della sottoclasse Sublist.
TEST CASE DESIGN	Verifico il funzionamento della rimozione, controllando sia la diminuzone della dimensione della sublist, sia l'assenza di un elemento scelto (eliminato dalla sublist).
TEST DESCRIPTION	Creo una lista, aggiungo degli elementi. Verifico che venga rimosso l'elemento giusto e che la dimensione della sublist diminuisca.
PRE-CONDITION	/
POST-CONDITION	/
EXPECTED RESULT	Verifico nel primo assert che con il metodo remove(Object) venga eliminato l'elemento scelto, con il secondo che la dimensione della sublist sia diminuita, con il terzo l'assenza di un elemento scelto (appena eliminato dalla sublist).

SublistRemoveAllTest	
SUMMARY	Il test verifica il funzionamento del metodo RemoveAll(HCollection) della sottoclasse Sublist.
TEST CASE DESIGN	Verifico che vengano rimossi tutti gli elementi della collection passata come parametro e che venga diminuita la dimensione della sublist
TEST DESCRIPTION	Creo una lista, aggiungo degli elementi. Applico il metodo removeAll passando come parametro un'altra lista.
PRE-CONDITION	/
POST-CONDITION	/
EXPECTED RESULT	Verifico nel primo assert che la dimensione della sublist sia diminuita della dimensione della seconda lista, nel secondo l'assenza di un elemento scelto (appena eliminato dalla sublist).

SublistRetainAllTest	
SUMMARY	Il test verifica il funzionamento del metodo RetainAll(HCollection) della sottoclasse Sublist.
TEST CASE DESIGN	Verifico che vengano rimossi dalla sublist tutti gli elementi tranne quelli della collection passata come parametro e che venga diminuita la dimensione della sublist
TEST DESCRIPTION	Creo una lista, aggiungo degli elementi. Applico il metodo retainAll passando come parametro un'altra lista.
PRE-CONDITION	/
POST-CONDITION	/
EXPECTED RESULT	Verifico nel primo assert che la dimensione della sublist sia diminuita come da previsione, nel secondo la presenza di un elemento scelto (che deve essere mantenuto nella sublist secondo la logica di retainAll).

SublistSetTest	
SUMMARY	Il test verifica il funzionamento del metodo Set(int,Object) della sottoclasse Sublist.
TEST CASE DESIGN	Verifico che venga sostituito l'elemento scelto dall'indice con quello passato come parametro.
TEST DESCRIPTION	Creo una lista, aggiungo degli elementi. Applico il metodo set modificando un elemento della sublist
PRE-CONDITION	/
POST-CONDITION	/
EXPECTED RESULT	Verifico nel secondo assert che la dimensione della sublist sia diminuita come da previsione, nel primo la presenza di un elemento scelto (che e' stato introdotto nella sublist con il metodo set).

SublistToArrayTest	
SUMMARY	Il test verifica il funzionamento dei metodi toArray(Object[]) e toArray() della sottoclasse Sublist.
TEST CASE DESIGN	Verifico che venga creato un array della giusta dimensione e con gli elementi corretti, per entrambi i toArray
TEST DESCRIPTION	Creo una lista, aggiungo degli elementi. Applico uno alla volta entrambi i metodi toArray ed eseguo delle verifiche.
PRE-CONDITION	/
POST-CONDITION	/
EXPECTED RESULT	Verifico nel primo assert che la sub-sublist abbia la dimensione corretta, nel secondo che la sub-sublist presenti un valore corretto in corrispondenza di un indice valido, nel terzo che, a seguito del metodo clear, la sub-sublist presenti la dimensione nulla.

Sublist2Test	
SUMMARY	Il test verifica il funzionamento dei metodi Sublist(Object[]) della sottoclasse Sublist.
TEST CASE DESIGN	Verifico il corretto funzionamento di sublist annidate, testando i metodi piu' significativi come clear(), remove(Object) e add(Object)
TEST DESCRIPTION	Creo una lista, aggiungo degli elementi. Creo su di essa una sublist e poi creo suna sub-sublist. Applico e testo il metodo add, clear e remove.
PRE-CONDITION	/
POST-CONDITION	/
EXPECTED RESULT	Verifico nel primo assert che la sub-sublist abbia la dimensione corretta, nel secondo che la sub-sublist presenti un valore corretto in corrispondenza di un indice valido, nel terzo che, a seguito del metodo clear, la sub-sublist presenti la dimensione nulla.

SublistIteratorPreviousTest	
SUMMARY	Il test verifica il funzionamento dei metodi iterator() della sottoclasse Sublist.
TEST CASE DESIGN	Verifico che venga creato l'iteratore e che funzioni soprattutto nella rimozione.
TEST DESCRIPTION	Creo una lista, aggiungo degli elementi. Creo un iteratore sulla sublist. Elimino un elemento specifico dopo aver avanzato l'iteratore. Inoltre testo la rimozione consecutiva di due elementi.
PRE-CONDITION	/
POST-CONDITION	/
EXPECTED RESULT	Verifico nel primo assert che la dimensione della sublist sia diminuita a seguito della rimozione, nei seguenti tre che siano contenuti nella sublist solo gli elementi non rimossi, infine nell'ultimo che per due rimozioni consecutive viene lanciato IllegalStateException().

SublistListIteratorPreviousTest	
SUMMARY	Il test verifica il funzionamento dei metodi listIterator(int) della sottoclasse Sublist.
TEST CASE DESIGN	Verifico che venga testata la rimozione di un elemento quando si e' appena eseguito il metodo previous().
TEST DESCRIPTION	Creo una lista, aggiungo degli elementi. Creo un iteratore sulla sublist. Elimino un elemento specifico dopo aver decrementato l'iteratore fino ad un valore scelto.
PRE-CONDITION	/
POST-CONDITION	/
EXPECTED RESULT	Verifico nel primo assert che la dimensione della sublist sia diminuita a seguito della rimozione, nel secondo che il valore rimosso non sia piu' presente nella lista.

SublistListIteratorNextPreviousTest	
SUMMARY	Il test verifica il funzionamento dei metodi listIterator() della sottoclasse Sublist.
TEST CASE DESIGN	Verifico che venga testato l'avanzamento e il decremento dell'iteratore (deve essere restituito lo stesso elemento)
TEST DESCRIPTION	Creo una lista, aggiungo degli elementi. Creo un iteratore sulla sublist. Applico in sequenza next e previous controllando che il valore di ritorno sia il medesimo.
PRE-CONDITION	/
POST-CONDITION	/
EXPECTED RESULT	Verifico nel primo e nel secondo assert che il valore di ritorno sia contenuto nella sublist e il medesimo

SublistListIteratorSetTest	
SUMMARY	Il test verifica il funzionamento dei metodi listIterator() della sottoclasse Sublist.
TEST CASE DESIGN	Verifico che venga testato il metodo set , in particolare che se esso viene invocato senza alcuna chiamata a next o previous, lancia un'eccezione. Controllo inoltre che l'elemento settato sia poi veramente presente nella sublist
TEST DESCRIPTION	Creo una lista, aggiungo degli elementi. Creo un iteratore sulla sublist. Applico set prima senza e poi a seguito di una chiamata a next.
PRE-CONDITION	/
POST-CONDITION	/
EXPECTED RESULT	Verifico nel primo assert che set non puo' essere chiamato prima di una chiamata a next o previous, nel secondo e nel terzo che viene restituito il valore appena settato a seguito di una chiamata a next e previous

ListIteratorRemoveTest	
SUMMARY	Il test verifica il funzionamento dei metodi listIterator().
TEST CASE DESIGN	Verifico che venga testato il metodo set , in particolare che se esso viene invocato senza alcuna chiamata a next o previous, lancia un'eccezione. Controllo inoltre che l'elemento settato sia poi veramente presente nella sublist
TEST DESCRIPTION	Creo una lista, aggiungo degli elementi. Creo un iteratore sulla sublist. Applico set prima senza e poi a seguito di una chiamata a next.
PRE-CONDITION	/
POST-CONDITION	/
EXPECTED RESULT	Verifico nel primo assert che la dimensione della sublist e' diminuita a seguiti della rimozione, nei successivi tre che siano presenti nella sublist solo gli elementi non rimossi, nell'ultimo che invocando remove due volte di fila viene lanciato IllegalStateException().

ListIteratorPreviousTest	
SUMMARY	Il test verifica il funzionamento del metodo listIterator(int).
TEST CASE DESIGN	Verifico che venga testata la rimozione di un elemento quando si e' appena eseguito il metodo previous().
TEST DESCRIPTION	Creo una lista, aggiungo degli elementi. Creo un iteratore sulla sublist. Elimino un elemento specifico dopo aver decrementato l'iteratore fino ad un valore scelto.
PRE-CONDITION	/
POST-CONDITION	/
EXPECTED RESULT	Verifico nel primo assert che la dimensione della sublist sia diminuita a seguito della rimozione, nel secondo che il valore rimosso non sia piu' presente nella lista.

ListIteratorNextPreviousTest	
SUMMARY	Il test verifica il funzionamento del metodo listIterator().
TEST CASE DESIGN	Verifico che venga testato l'avanzamento e il decremento dell'iteratore (deve essere restituito lo stesso elemento). Verifico il funzionamento di hasPrevious() all'inizio
TEST DESCRIPTION	Creo una lista, aggiungo degli elementi. Creo un iteratore sulla sublist. Applico in sequenza next e previous controllando che il valore di ritorno sia il medesimo.
PRE-CONDITION	/
POST-CONDITION	/
EXPECTED RESULT	Verifico nel primo e nel secondo assert che il valore di ritorno sia contenuto nella sublist e il medesimo

ListIteratorSetTest	
SUMMARY	Il test verifica il funzionamento dei metodi listIterator().
TEST CASE DESIGN	Verifico che venga testato il metodo set , in particolare che se esso viene invocato senza alcuna chiamata a next o previous, lancia un'eccezione. Controllo inoltre che l'elemento settato sia poi veramente presente nella sublist
TEST DESCRIPTION	Creo una lista, aggiungo degli elementi. Creo un listIterator, applico set prima senza e poi a seguito di una chiamata a next.
PRE-CONDITION	/
POST-CONDITION	/
EXPECTED RESULT	Verifico nel primo assert che set non puo' essere chiamato prima di una chiamata a next o previous (viene lanciato IllegalStateException), nel secondo e nel terzo che viene restituito il valore appena settato a seguito di una chiamata a next.

MAPADAPTER

ContainsTest	
SUMMARY	Il test verifica il funzionamento del metodo values()
TEST CASE DESIGN	Verifico che il metodo contains(Object) di MyValuesCollection funzioni correttamente.
TEST DESCRIPTION	Creo una mappa e aggiungo degli elementi con il metodo put. Effettuo delle verifiche sui v contenuti, prima e dopo aver applicato il metodo clear().
PRE-CONDITION	/
POST-CONDITION	/
EXPECTED RESULT	Verifico nei primi due assert che i v contenuti nella collection siano presenti, nei secondi due che, a seguito dell'applicazione del metodo clear, siano asssenti.

RemoveTest	
SUMMARY	Il test verifica il funzionamento del metodo values()
TEST CASE DESIGN	Verifico che il metodo remove(Object) di MyValuesCollection funzioni correttamente.
TEST DESCRIPTION	Creo una mappa e aggiungo degli elementi con il metodo put. Applico il metodo remove ed effettuo dei controlli
PRE-CONDITION	/
POST-CONDITION	/
EXPECTED RESULT	Nel primo e secondo assert verifico che si sia ridotta sia la dimensione della mappa sia quella della collection, negli ultimi due che il valore rimosso non sia piu' presente sia nella mappa che nella collection.

RetainAll2Test	
SUMMARY	Il test verifica il funzionamento del metodo values()
TEST CASE DESIGN	Verifico che il metodo retainAll(HCollection) di MyValuesCollection funzioni correttamente. (questo test non e' fondamentale in quanto il metodo retainAll non viene sovrascritto in MyValueCollection, quindi viene richiamata la funzione di MyKeySet gia' testata e documentata).
TEST DESCRIPTION	Creo due mappe e aggiungo degli elementi con il metodo put. Applico il metodo retainAll ed effettuo dei controlli
PRE-CONDITION	/
POST-CONDITION	/
EXPECTED RESULT	Nel primo assert verifico che la dimensione della collection si sia opportunamente ridotta, nel secondo e terzo che siano contenuti solo i valori previsti.

Iterator1Test	
SUMMARY	Il test verifica il funzionamento del metodo values().
TEST CASE DESIGN	Verifico che il metodo iterator() di MyValuesCollection funzioni correttamente: controllo che siano presenti i diversi valori iterati e che venga lanciata un'eccezione se si esegue remove senza aver eseguito next prima.
TEST DESCRIPTION	Creo due mappe e aggiungo degli elementi con il metodo put. Creo un iteratore sulla collection di v . Controllo che remove non si possa applicare senza che prima sia stato chiamato next (lancia un'eccezione)
PRE-CONDITION	/
POST-CONDITION	/
EXPECTED RESULT	Nel primo assert verifico che invocando remove senza prima next viene lanciata IllegalStateException(), negli assert del ciclo while verifico che per ogni next ottengo un v presente nella mappa

Iterator2Test	
SUMMARY	Il test verifica il funzionamento del metodo values().
TEST CASE DESIGN	Verifico che il metodo iterator() di MyValuesCollection funzioni correttamente: controllo che venga lanciata un'eccezione
TEST DESCRIPTION	Creo due mappe e aggiungo degli elementi con il metodo put. Creo un iteratore sulla collection di v. Controllo che remove non si possa applicare due volte consecutivamente.
PRE-CONDITION	/
POST-CONDITION	/
EXPECTED RESULT	Nell'assert verifico che venga lanciata IllegalStateException() se remove viene invocato due volte consecutivamente (senza eseguire un next dopo il primo remove).

RemoveTest	
SUMMARY	Il test verifica il funzionamento del metodo remove(Object) (tuttavia viene di fatto testato simultaneamente anche il metodo contains(Object)) (all'interno di MyEntrySet)
TEST CASE DESIGN	Verifico che il metodo rimuova l'elemento scelto riducendo la dimensione della mappa
TEST DESCRIPTION	Creo una mappa e aggiungo degli elementi con il metodo put, creo un set di elementi e poi creo un iteratore su di esso. Ora itero fino ad individuare la chiave scelta, infine rimuovo l'elemento che presenta tale chiave
PRE-CONDITION	/
POST-CONDITION	/
EXPECTED RESULT	Verifico nel primo assert che, a seguito della rimozione, la dimensione rimane invariata, nel secondo che l'elemento eliminato non e' di fatto piu' presente nella lista.

EntryClassTest	
SUMMARY	Il test verifica il funzionamento della classe EntryClass (in particolare del metodo setValue(Object) (all'interno di MyEntrySet).
TEST CASE DESIGN	Verifico che il metodo sostituisca il v scelto con quello passato per parametro.
TEST DESCRIPTION	Creo una mappa e aggiungo degli elementi con il metodo put, creo un set di entries e creo su di esso un iteratore. Apllico il metodo set dopo un next,
PRE-CONDITION	/
POST-CONDITION	/
EXPECTED RESULT	Verifico nel primo assert che il metodo set ritorna il v precedentemente presente e poi sostituito, nel secondo che sia stato inserito il nuovo v e sia quindi presente nella mappa.

Iterator2Test	
SUMMARY	Il test verifica il funzionamento del metodo Iterator(Object) (all'interno di MyEntrySet)
TEST CASE DESIGN	Verifico che il metodo remove lanci un'eccezione se viene invocato due volte consecutivamente
TEST DESCRIPTION	Creo una mappa e aggiungo degli elementi con il metodo put, creo un set di elementi e poi creo un iteratore su di esso.
PRE-CONDITION	/
POST-CONDITION	/
EXPECTED RESULT	Verifico tramite blocco try e catch che l'esecuzione consecutiva di due remove lancia IllegalStateException()

Iterator1Test	Il test verifica il funzionamento del metodo Iterator(Object) (all'interno di MyEntrySet)
SUMMARY	Verifico che il metodo remove puo' lanciare un'eccezione e che il metodo next restituisce l'opportuno elemento di tipo Object
TEST CASE DESIGN	Creo una mappa e aggiungo degli elementi con il metodo put, creo un set di elementi e poi creo due iteratori su di esso. Ora itero il secondo finche' ho elementi
TEST DESCRIPTION	/
PRE-CONDITION	/
POST-CONDITION	Verifico nel primo assert che il metodo remove non puo' essere invocato prima che sia mai stato chiamato next (lancia IllegalStateException()). Negli assert all'interno del ciclo while, verifico che ogni elemento restituito da next() e' contenuto nella mappa
EXPECTED RESULT	Il test verifica il funzionamento del metodo Iterator(Object)

MapAdapterHashCode	
SUMMARY	Il test verifica il funzionamento del metodo HashCode(Object)
TEST CASE DESIGN	Verifico che il metodo restituisca true se le due mappe confrontate hanno stessa dimensione e stessi elementi (sia k che v uguali)
TEST DESCRIPTION	Creo due mappe e aggiungo degli elementi con il metodo put, verifico l'uguaglianza fra le due prima e dopo una modifica.
PRE-CONDITION	/
POST-CONDITION	/
EXPECTED RESULT	Il primo assert verifica che le due mappe abbiano stesso hashcode, il secondo che le due mappe presentino hashcode diversi dopo una modifica di una v di un elemento, il terzo che una mapppa non null ha hashCode diverso dall'hashcode di null

Equals	
SUMMARY	Il test verifica il funzionamento del metodo Equals(Object)
TEST CASE DESIGN	Verifico che il metodo restituisca true se le due mappe confrontate hanno stessa dimensione, stessi elementi (sia k che v uguali)
TEST DESCRIPTION	Creo due mappe e aggiungo degli elementi con il metodo put, verifico l'uguaglianza fra le due prima e dopo una modifica
PRE-CONDITION	/
POST-CONDITION	/
EXPECTED RESULT	Il primo assert verifica che le due mappe siano uguali, il secondo che le due mappe non siano piu' uguali dopo una modifica di una v di un elemento, il terzo che una mapppa non null non e' uguale a null

MapAdapterRemoveElementTest	
SUMMARY	Il test verifica il funzionamento del metodo remove(int) e remove(Object)
TEST CASE DESIGN	Verifico che il metodo elimini l'entry se presente, altrimenti non faccia niente se la chiave non e' presente.
TEST DESCRIPTION	Creo una mappa e aggiungo degli elementi con il metodo put, rimuovo un elemento presente a un indice valido, elimino una entry non presente (non fa nulla il metodo)
PRE-CONDITION	/
POST-CONDITION	/
EXPECTED RESULT	Verifico che la dimensione della mappa si sia ridotta (solo quando ho eliminato una entry presente)

MapAdapterPutElementTest	
SUMMARY	Il test verifica il funzionamento del metodo put(Object,Object)
TEST CASE DESIGN	Verifico che il metodo inserisca l'elemento nella mappa.
TEST DESCRIPTION	Creo una mappa e aggiungo un elemento con il metodo put
PRE-CONDITION	/
POST-CONDITION	/
EXPECTED RESULT	Verifico che la chiave inserita (insieme a v) sia presente nella mappa

MapAdapterPutRemoveTest	
SUMMARY	Il test verifica il funzionamento del metodo put(Object,Object) e remove(int) insieme
TEST CASE DESIGN	Verifico che il metodo put inserisca l'elemento nella mappa e che poi esso venga rimosso con remove.
TEST DESCRIPTION	Creo una mappa e aggiungo degli elementi con il metodo put. Poi li rimuovo e verifico che la mappa sia vuota.
PRE-CONDITION	/
POST-CONDITION	/
EXPECTED RESULT	Verifico che la mappa sia vuota a seguito della rimozione delle entries dalla mappa.

MapAdapterPutSubstituteTest	
SUMMARY	Il test verifica il funzionamento del metodo put(Object,Object)
TEST CASE DESIGN	Verifico che il metodo put sostituisca una entry senza aumentare la dimensione della mappa nel caso la chiave da inserire sia gia' presente
TEST DESCRIPTION	Creo una mappa e aggiungo degli elementi con il metodo put (molti presentano pero' stessa chiave).
PRE-CONDITION	/
POST-CONDITION	/
EXPECTED RESULT	Verifico nel primo assert che il v associato alla chiave inserita piu' volte sia quello dell'ultimo inserimento, nel secondo che la dimensione della mappa non sia variata inserendo piu' volte la stessa chiave.

MapAdapterPutAllGetTest	
SUMMARY	Il test verifica il funzionamento del metodo putAll(HMap)
TEST CASE DESIGN	Verifico che il metodo putAll
TEST DESCRIPTION	Creo due mappe e aggiungo degli elementi con il metodo put. Applico il metodo putAll a una mappa passando come parametro l'altra.
PRE-CONDITION	/
POST-CONDITION	/
EXPECTED RESULT	Verifico nel primo assert che sia presente un nuovo v aggiunto (in questo caso sostituito perche' la chiave era gia' presente), nel secondo che la dimensione della mappa sia opportunamente aumentata.

keySetModifyMapAdapterTest	
SUMMARY	Il test verifica il funzionamento del metodo keySet: modificando la mappa deve essere modificata contestualmente anche il set di chiavi.
TEST CASE DESIGN	Verifico che modificando la mappa, vari anche la dimensione anche del set.
TEST DESCRIPTION	Creo una mappa, aggiungo degli elementi con put. Verifico che si modifichi anche il set contestualmente.
PRE-CONDITION	/
POST-CONDITION	/
EXPECTED RESULT	Verifico nel primo assert che il set di chiavi sia vuoto, nel secondo che la dimensione del set di chiavi sia aumentato a seguito dell'aggiunta di entries nella mappa.

keySetModifyKeySetTest	
SUMMARY	Il test verifica il funzionamento del metodo keySet: modificando l'oggetto MyKeySet deve essere modificata contestualmente anche la mappa
TEST CASE DESIGN	Verifico che modificando il set di chiavi, vari anche la mappa.
TEST DESCRIPTION	Creo una mappa, aggiungo degli elementi con put. Rimuovo un elemento dal set di chiavi e verifico che, per la chiave rimossa, non sia presente piu' una entry neanche nella mappa.
PRE-CONDITION	/
POST-CONDITION	/
EXPECTED RESULT	Verifico nell'assert che, per la chiave rimossa, eseguendo il metodo get, ottengo null.

keySetClearTest	
SUMMARY	Il test verifica il funzionamento del metodo keySet: modificando l'oggetto MyKeySet deve essere modificata contestualmente anche la mappa
TEST CASE DESIGN	Verifico che modificando il set di chiavi, con il metodo clear(), viene modificata anche la mappa contestualmente.
TEST DESCRIPTION	Creo una mappa, aggiungo degli elementi con put. Eseguo il metodo clear sul set di chiavi.
PRE-CONDITION	/
POST-CONDITION	/
EXPECTED RESULT	Verifico nel primo assert che la dimensione del set di chiavi sia nullo, nel secondo che la dimensione della mappa sia nulla, nel terzo che non sia presente alcun valore in corrispondenza di una chiave eliminata

keySetModifyKeyContainsAllTest	
SUMMARY	Il test verifica il funzionamento del metodo keySet: viene testato il metodo contains(Object) e containsAll(HCollection)
TEST CASE DESIGN	Verifico che modificando i metodi contains e containsAll venga ritornato il corretto booleano.
TEST DESCRIPTION	Creo due mappe, aggiungo degli elementi con put. Creo due set di key (associati alle due diverse mappe). Applico il metodo containsAll ad un set di chiavi passando come parametro l'altro.
PRE-CONDITION	/
POST-CONDITION	/
EXPECTED RESULT	Verifico nel primo assert che, dopo la rimozione, non sia piu' presente una entry scelta, nel secondo che il metodo containsAll ritorni il corretto booleano.

keySetHashCodeTest	
SUMMARY	Il test verifica il funzionamento del metodo keySet: viene testato il metodo hashCode()
TEST CASE DESIGN	Verifico che il numero di hashCode sia ugualeper due mappe uguali (stessa lunghezza, stesse entries)
TEST DESCRIPTION	Creo due mappe, aggiungo degli elementi con put(aggiungo gli stessi elementi nelle due mappe). Confronto i due hashCode.
PRE-CONDITION	/
POST-CONDITION	/
EXPECTED RESULT	Verifico nell'assert che l'hashcode di due mappe che presentano stesse entry (uguali sia k che v) e stesse dimensioni sia uguale.

keySetRemoveAllTest	
SUMMARY	Il test verifica il funzionamento del metodo keySet: viene testato il metodo removeAll(HCollection)
TEST CASE DESIGN	Verifico che il metodo removeAll elimini dalla mappa gli elementi corretti.
TEST DESCRIPTION	Creo due mappe, aggiungo degli elementi con put. Creo due set di key (associati alle due diverse mappe). Applico il metodo removeAll.
PRE-CONDITION	/
POST-CONDITION	/
EXPECTED RESULT	Verifico nell'assert che la dimensione e' variata come previsto, nel secondo verifico che sia rimasto il corretto elemento.

keySetRetainAllTest	
SUMMARY	Il test verifica il funzionamento del metodo keySet: viene testato il metodo retainAll(HCollection)
TEST CASE DESIGN	Verifico che il metodo retainAll elimini dalla mappa gli elementi corretti.
TEST DESCRIPTION	Creo due mappe, aggiungo degli elementi con put. Creo due set di key (associati alle due diverse mappe). Applico il metodo retainAll.
PRE-CONDITION	/
POST-CONDITION	/
EXPECTED RESULT	Verifico nei due assert che la dimensione della mappa su cui si applica il retainAll cambi opportunamente.

keySetIteratorTest	
SUMMARY	Il test verifica il funzionamento del metodo keySet: viene testato il metodo iterator che ritorna un iterator. Verifico le funzionalita' dello stesso (next(), hasNext(), remove()).
TEST CASE DESIGN	Verifico che il metodo iterator ritorni un iteratore opportuno, ponendo attenzione soprattutto sul metodo remove dell'iteratore stesso.
TEST DESCRIPTION	Creo una mappa, aggiungo degli elementi con put. Creo un set di key, su di esse creo un iteratore. Rimuovo un elemento scelto.
PRE-CONDITION	/
POST-CONDITION	/
EXPECTED RESULT	Verifico nel primo assert che la dimensione sia diminuita, nel secondo che l'elemento rimosso non e' piu' presente.

keySetToArrayTest	
SUMMARY	Il test verifica il funzionamento del metodo keySet: viene testato il metodo toArray() e il metodo toArray(Object[])
TEST CASE DESIGN	Verifico il corretto funzionamento dei metodi toArray, ponendo attenzione alla dimensione della mappa.
TEST DESCRIPTION	Creo una mappa, aggiungo degli elementi con put. Creo un set di key. Applico i due metodi toArray.
PRE-CONDITION	/
POST-CONDITION	/
EXPECTED RESULT	Verifico nei due assert che la dimensione sia corretta (nel secondo che la dimensione dell'array sia pari a quella dell'array precedentemente creato)

DOCUMENTAZIONE TEST SUITE

Le classi ListAdapter e MyAdapter non accettano valori nulli, in altre parole viene lanciato NullPointerException() in fase di inserimento in caso si cerchi di inserire null (questo vale anche per le dIverse sottoclassi che sono state implementate per entrambe le classi). ListAdapter e' implementato con l'ausilio di un vector, MapAdapter con l'ausilio di una hashTable.

Per realizzare i metodi backed per entrambe le classi e' stata utilizzata una medesima logica: sono state create delle sottoclassi (Sublist per ListAdapter che reimplementa tutti i metodi di List, MykeySet,MyValueCollection e MyEntrySet per MapAdapter che implementano HSet o HCollection) che permettono di agire, tramite vector o hashtable, direttamente anche sul'oggetto ListAdapter o MapAdapter.

SUMMARY	La suite contiene tutti i test riguardanti ListAdapter e MyAdapter. Si e' cercato di testare ogni metodo di queste due classi in classi di test distinti (tuttavia c'erano metodi che richiedevano implementazioni di sottoclassi, ad esempio per risolvere il problema del backing: i test dei metodi di tali sottoclassi sono stati implementati nella stessa classe di test).
TEST SUITE DESIGN	I diversi test sono stati implementati in modo da testare un metodo in particolare (o a volte anche due se i metodi erano simili e facili da testare simultaneamente. Pertanto per ogni test case e' presente generalmente piu' di un assert). Nella classe MyAdapter, ho implementato tutti i metodi della sottoclasse MyKeySet per realizzare il backing, successivamente ho implementato le sottoclassi MyValueCollection e MyEntrySet estendendo MyKeySet. Non ho testato i metodi che non sono stati sovrascritti perche' gia' testati fra i metodi di MyKeySet. Non ho testato il metodo equals di MyValueCollection perche' presenta il medesimo algoritmo del corrispondente nella classe MyKeySet a eccezione di una piccola variazione nella prima riga.
PRE-CONDITION	Non ho precondizioni: a ogni test generalmente creo un nuovo oggetto ListAdapter o una MapAdapter e aggiungo degli elementi per creare liste e mappe non vuote che possano testare i vari metodi (ovviamente cio' dipende da caso a caso)
POST-CONDITION	Non ho post condizioni
TEST CASES	**ListAdapter**: AddTest Add1Test AddAllTest AddAll2Test ContainsAllTest HashCodeTest EqualsTest IteratorTest RemoveTest Remove2Test RetainAllTest SetTest ToArray2Test SublistTest SublistAddTest SublistAddAllTest SublistAddAll2Test

	SublistClearTest SublistContainsAllTest SublistEqualsTest SublistHashCodeTest SublistLastIndexOfTest SublistIndexOfTest SublistRemoveTest SublistRemoveAllTest SublistRetainAllTest SublistSetTest SublistToArrayTest Sublist2Test SublistIteratorPreviousTest SublistListIteratorPreviousTest SublistListIteratorNextPreviousTest SublistListIteratorSetTest ListIteratorRemoveTest ListIteratorPreviousTest ListIteratorNextPreviousTest ListIteratorSetTest **MapAdapter**: ContainsTest RemoveTest RetainAll2Test Iterator1Test Iterator2Test RemoveTest EntryClassTest Iterator2Test Iterator1Test MapAdapterHashCode Equals MapAdapterRemoveElementTest MapAdapterPutElementTest MapAdapterPutRemoveTest MapAdapterPutSubstituteTest MapAdapterPutAllGetTest keySetModifyMapAdapterTest keySetClearTest keySetModifyKeyContainsAllTest keySetHashCodeTest keySetRemoveAllTest keySetRetainAllTest keySetIteratorTest keySetToArrayTest
TEST SUITE EXECUTION RECORDS	Sono stati eseguiti 63 test e sono stati eseguiti tutti correttamente.
EXECUTION VARIABLES	Non ho execution variables