Francesco Visentin matricola 1216419

Test suite della classe MapAdapter.java

La seguente test suite è stata ideata per testare i metodi della classe MapAdapter.java e si riferisce ai test presenti all'interno delle test classes MapTester.java ed EntryTester.java, SetTester.java, SetIteratorTester.java.

Ogni test class è focalizzata nel testing di una delle classi/sottoclassi contenute all'interno di MapAdapter.java, nello specifico:

- La classe MapTester comprende i test dal numero 1 a 20 e testa i metodi della classe di base di MapAdapter relativi all'interfaccia HMap. I test di questa classe hanno a disposizione un oggetto MapAdapter inizialmente vuoto e correttamente inizializzato.
- La classe EntryTester comprende i test dal numero 21 a 24 e testa i metodi della sottoclasse HEntry di MapAdapter relativi all'interfaccia HMap.Entry. I test di questa classe hanno a disposizione un oggetto HMap.Entry non ancora inizializzato.
- La classe SetTester comprende i test dal numero 25 a 52 e testa i metodi della sottoclassi EntrySet, KeySet e ValueSet di MapAdapter relativi all'interfaccia HSet. Le sottoclassi KeySet ed ValueSet estendono la classe EntrySet e ne sovrascrivano solo tre metodi. I metodi sovrascritti sono stati testati indipendentemente per ogni classe. I metodi derivati dalla classe di base sono stati anch'essi testati per ogni classe utilizzando però lo stesso test, differenziando unicamente il tipo di set su cui questo è stato applicato. La documentazione di questi test, in quanto tra loro analoghi, è stata raggruppata nelle pagine sottostanti come se questi fossero un unico test (Es: i test entrySetClearTest, keySetClearTest ed valueSetClearTest sono stati documentati in unisono sotto il nome generico setClearTest).
  I test di questa classe hanno a disposizione un oggetto MapAdapter ed un oggetto HSet (specifico per ogni test) entrambi correttamente inizializzati a dei valori di riferimento.
- La classe SetIteratorTester comprende i test dal numero 53 a 57 e testa i metodi delle sottoclassi EntryIterator e KeyValIterator di MapAdapter relativi all'interfaccia HIterator. La classe KeyValIterator estende la classe EntryIterator e ne sovrascrive il solo metodo next() testato individualmente per entrambe le classi. I test di questa classe hanno a disposizione un oggetto MapAdapter ed un oggetto HSet (di tipo EntrySet) entrambi inizialmente vuoti e correttamente inizializzati.

La suite è formata da un totale di 57 test descritti nelle pagine a seguire, ed riportati secondo l'ordine sopra citato corrispondente alla loro posizione all'interno del file MapAdapter.java

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

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

Oltre agli oggetti forniti in fase di inizializzazione come precondizione, tutti gli altri oggetti necessari per effettuare i test vengono creati all'interno di questi. Per favorire questo processo all'interno delle classi sono stati messi a disposizione il metodo getEntry(Object k, Object v) che restituisce un entry correttamente inizializzata ed il metodo fill(MapAdapter m) che inserisce all'interno della mappa otto coppie, tutte diversi da null, di cui: le prime quattro aventi chiavi di tipo "String" e valori di tipo "int", le ultime quattro chiavi "int" e valori "String".

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

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

1° ClearTest():

- Test del metodo public void clear()
- **Summary / Description**: verifica inizialmente il corretto funzionamento su una mappa vuota, vengono poi inseriti due entry nella mappa e invocato nuovamente il metodo. Si verifica che le due entry siano state effettivamente eliminate.
- **Design**: verifica che il metodo svuoti correttamente la mappa.
- **Pre-Condition**: un oggetto MapAdapter vuoto e correttamente inizializzato.
- **Post-Condition**: la mappa torna ad essere vuota, nessuna eccezione è stata generata.
- **Expected results**: il metodo elimina correttamente tutte le coppie contenute nella mappa.

2° ContainsKeyTest():

- Test del metodo public boolean containsKey(Object key)
- **Summary / Description**: viene prima testato il metodo su una mappa vuota. Questa viene poi riempita con delle coppie di riferimento e si verifica il corretto funzionamento del metodo su queste entry. Alcune coppie vengono rimosse e si ritesta il metodo su questi valori aspettandosi che restituisca false. La mappa viene infine svuotata del tutto ed il metodo testato ancora.
- **Design**: verifica il corretto comportamento del metodo applicandolo in relazione ad una lista di valori di riferimento.
- **Pre-Condition**: un oggetto MapAdapter vuoto e correttamente inizializzato, un parametro “key” diverso da “null”.
- **Post-Condition**: la mappa torna ad essere vuota, il metodo restituirà sempre false.
- **Expected results**: il metodo ritorna true se e solo se la chiave cercata è presente nella mappa.

3° ContainsKeyExceptionsTest():

- Test del metodo public boolean containsKey(Object key)
- **Summary / Description**: il lancio dell'eccezione viene prima verificato su una mappa vuota, viene poi inserita una coppia e si verifica nuovamente il corretto funzionamento del metodo. Viene poi verificato che il valore inserito in precedenza non abbia subito modifiche.
- **Design**: verifica il corretto lancio dell'eccezione NullPointerException sia quando la mappa è vuota sia quando contiene degli elementi.
- **Pre-Condition**: un oggetto MapAdapter vuoto e correttamente inizializzato.
- **Post-Condition**: la mappa contiene un elemento che non è stato modificato dal lancio delle eccezioni. Nessun altra eccezione è stata generata.
- **Expected results**: il corretto lancio di due NullPointerException quando il parametro passato è "null".

4° ContainsValueTest():

- Test del metodo public boolean containsKey(Object key)
- **Summary / Description**: viene prima testato il metodo su una mappa vuota. Questa viene poi riempita con delle coppie di riferimento e si verifica il corretto funzionamento del metodo su queste entry. Alcune coppie vengono rimosse e si ritesta il metodo su questi valori aspettandosi che restituisca false. La mappa viene infine svuotata del tutto ed il metodo testato ancora.
- **Design**: verifica il corretto comportamento del metodo applicandolo in relazione ad una lista di coppie di riferimento.
- **Pre-Condition**: un oggetto MapAdapter vuoto e correttamente inizializzato, un parametro “value” diverso da “null”.
- **Post-Condition**: la mappa torna ad essere vuota, il metodo restituirà sempre false.
- **Expected results**: il metodo ritorna true se e solo se la chiave cercata è presente nella mappa.

5° ContainsValueExceptionsTest():

- Test del metodo public boolean containsValue(Object value)
- **Summary / Description**: il lancio dell'eccezione viene prima verificato su una mappa vuota, viene poi inserita una coppia e si verifica nuovamente il corretto funzionamento del metodo. Viene poi verificato che il valore inserito in precedenza non abbia subito modifiche.
- **Design**: verifica il corretto lancio dell'eccezione NullPointerException sia quando la mappa è vuota sia quando contiene degli elementi.
- **Pre-Condition**: un oggetto MapAdapter vuoto e correttamente inizializzato.
- **Post-Condition**: la mappa contiene un elemento che non è stato modificato dal lancio delle eccezioni. Nessun altra eccezione è stata generata.
- **Expected results**: il corretto lancio di due NullPointerException quando il parametro passato è "null".

6° EntrySetTest():

- Test del metodo public HSet entrySet()
- **Summary / Description**: verifica inizialmente il funzionamento del metodo su una mappa vuota. La mappa viene poi riempita e il metodo invocato nuovamente. Si verifica che il set abbia la stessa dimensione della mappa e che i valori al suo interno siano effettivamente delle istanze di HMap.Entry come richiesto dall'interfaccia.
- **Design**: verifica che il metodo funzioni correttamente a prescindere dalle dimensioni della mappa su cui viene invocato.
- **Pre-Condition**: un oggetto MapAdapter vuoto e correttamente inizializzato.
- **Post-Condition**: la mappa ed il set vengono inizializzati con successo.
- **Expected results**: il metodo restituisce sempre un set contenente le entry presenti nella mappa.

7° EqualsTest():

- Test del metodo public boolean equals(Object o)
- **Summary / Description**: viene verificato che, date due mappe:
    - Il metodo restituisce false quando vengono confrontati oggetti di classi diverse.
    - Il metodo restituisce false quando il parametro “o” è uguale a “null”.
    - Dopo aver riempito due mappe con gli stessi elementi il metodo restituisce true.
    - Dopo aver estratto un elemento dalla mappa il metodo restituisce false.
    - Dopo aver inserito nuovamente l’elemento precedentemente eliminato dalla mappa il metodo torna a restituire true.
    - Due mappe vuote sono considerate uguali.
- **Design**: verifica che il metodo si comporti adeguatamente in tutte le condizioni esaminando due mappa contenenti degli elementi di riferimento.
- **Pre-Condition**: un oggetto MapAdapter vuoto e correttamente inizializzato.
- **Post-Condition**: le due mappa tornano ad essere vuote, un confronto tra esse ritornerà true.
- **Expected results**: il metodo soddisfa la condizione m1.equals(m2) se e solo se m1.entrySet().equals(m2.entrySet()).

8° GetTest():

- Test del metodo public Object get(Object key)
- **Summary / Description**: verifica inizialmente il corretto funzionamento su una lista vuota. Viene poi riempita la mappa ed il metodo viene testato su dei valori di riferimento controllando che il valore ritornata sia effettivamente presente nella mappa. La mappa viene poi nuovamente svuotata e viene verificato che il metodo torni a restituite sempre il valore null.
- **Design**: verifica il corretto fu
- **Pre-Condition**: un oggetto MapAdapter vuoto e correttamente inizializzato. Un parametro “key” con valore diverso da “null”.
- **Post-Condition**: la mappa torna ad essere vuota, nuove chiamate al metodo restituiranno sempre null.
- **Expected results**: il metodo restituisce un valore diverso da null se e solo se la chiave cercata è effettivamente presente nella mappa.

9° GetExceptionsTest():

- Test del metodo public Object get(Object key)
- **Summary / Description**: il lancio dell'eccezione viene prima verificato su una mappa vuota, viene poi inserita una coppia e si verifica nuovamente il corretto funzionamento del metodo. Viene poi verificato che il valore inserito in precedenza non abbia subito modifiche.
- **Design**: verifica il corretto lancio dell'eccezione NullPointerException sia quando la mappa è vuota sia quando contiene degli elementi.
- **Pre-Condition**: un oggetto MapAdapter vuoto e correttamente inizializzato.
- **Post-Condition**: la mappa contiene un elemento che non è stato modificato dal lancio delle eccezioni. Nessun altra eccezione è stata generata.
- **Expected results**: il corretto lancio di due NullPointerException quando il parametro passato è "null".

10° HashCodeTest():

- Test del metodo public int hashCode()
- **Summary / Description**: viene verificato che, date due mappe:
    - Il metodo sia sempre applicabile ad una mappa correttamente inzializzata.
    - Dopo aver riempito due mappe con gli stessi elementi queste hanno lo stesso hashCode.
    - Dopo aver estratto un elemento dalla mappa i due hashCode sono distinti.
    - Dopo aver inserito nuovamente l'elemento precedentemente eliminato le due mappe tornano ad avere lo stesso hashCode.
    - Mappe vuote hanno un hashCode pari a zero.
- **Design**: verifica che il metodo si comporti adeguatamente in tutte le condizioni esaminando due mappa contenenti degli elementi di riferimento.
- **Pre-Condition**: un oggetto MapAdapter vuoto e correttamente inizializzato.
- **Post-Condition**: le due mappa tornano ad essere vuote, il loro hashCode è uguale a zero
- **Expected results**: il metodo soddisfa la condizione: se m1.equals(m2) allora m1.hashCode() == m2.hashCode().

11° IsEmptyTest():

- Test del metodo public boolean isEmpty()
- **Summary / Description**: viene inizialmente verificato che per una mappa appena inizializzata il metodo restituisca true, la mappa viene poi riempita e si verifica che il metodo restituisca false. La mappa viene nuovamente svuotata e si verifica che il metodo torni a restituire true.
- **Design**: verifica il corretto comportamento del metodo sia quando una mappa è vuota sia quando questa contiene degli elementi.
- **Pre-Condition**: un oggetto MapAdapter vuoto e correttamente inizializzato.
- **Post-Condition**: la mappa torna ad essere vuota, nuove invocazioni del metodo restituiranno sempre true.
- **Expected results**: il metodo restituisce true se e solo se la mappa è effettivamente vuota.

12° KeySetTest():

- Test del metodo public HSet keySet()
- **Summary / Description**: verifica inizialmente il funzionamento del metodo su una mappa vuota. La mappa viene poi riempita e il metodo invocato nuovamente. Si verifica che il set abbia la stessa dimensione della mappa e che i valori al suo interno siano effettivamente delle chiavi della mappa come richiesto dall'interfaccia.
- **Design**: verifica che il metodo funzioni correttamente a prescindere dalle dimensioni della mappa su cui viene invocato.
- **Pre-Condition**: un oggetto MapAdapter vuoto e correttamente inizializzato.
- **Post-Condition**: la mappa ed il set vengono inizializzati con successo.
- **Expected results**: il metodo restituisce sempre un set contenente le chiavi presenti nella mappa.

13° PutTest():

- Test del metodo public Object put(Object key, Object value)
- **Summary / Description**: partendo da una mappa inizialmente vuota viene inizialmente inserita una nuova coppia nella mappa e si verifica che: la dimensione di questa sia stata aggiornata, “key” ed “value” siano stati memorizzati correttamente, il metodo restituisca true. Vengono poi inseriti altri valori, alcuni dei quali aventi stessa chiave, e si verifica che il metodo operi correttamente.
- **Design**: si verifica la correttezza del metodo effettuando aggiunte ripetute alla lista.
- **Pre-Condition**: un oggetto MapAdapter vuoto e correttamente inizializzato. Parametri “key” e “value” con valori diversi da “null”.
- **Post-Condition**: alla mappa sono stati aggiunte tre coppie. La sua dimensione è stata modificata correttamente.
- **Expected results**: il metodo inserisce correttamente una nuova coppia all’interno della mappa, eventualmente sovrascrivendo un valore, e aggiorna la dimensione della mappa.

14° PutExceptionsTest():

- Test del metodo public Object put(Object key, Object value)
- **Summary / Description**: il lancio dell'eccezione viene prima verificato su una mappa vuota. Viene poi inserita una coppia e si verifica nuovamente il corretto funzionamento del metodo. Viene poi verificato che il valore inserito in precedenza non abbia subito modifiche.
- **Design**: verifica il corretto lancio dell'eccezione NullPointerException sia quando la mappa è vuota sia quando contiene degli elementi.
- **Pre-Condition**: un oggetto MapAdapter vuoto e correttamente inizializzato.
- **Post-Condition**: la mappa contiene un elemento che non è stato modificato dal lancio delle eccezioni. Nessun altra eccezione è stata generata.
- **Expected results**: il corretto lancio di due NullPointerException quando almeno uno dei parametri passati è "null".

15° PutAllTest():

- Test del metodo public void putAll(HMap t)
- **Summary / Description**: alla mappa, inizialmente vuota, si verifica che il metodo funzioni correttamente anche usando una mappa vuota come parametro. Vengono poi aggiunti alcuni valori alla mappa di base. Viene poi invocato il metodo passando come parametro una mappa appositamente create contenente dei valori di riferimento. Si verifica che le coppie della mappa "t" siano state aggiunte alla mappa di base e si verifica che i valori aventi chiave comune siano stati sovrascritti.
- **Design**: verifica che gli elementi della mappa passata come parametro vengano correttamente inseriti all'interno della mappa eventualmente sovrascrivendo il valore degli elementi che hanno la stessa chiave.
- **Pre-Condition**: un oggetto MapAdapter vuoto e correttamente inizializzato. Un parametro "t" diverso da "null".
- **Post-Condition**: la mappa è formata da 9 coppie. Nessuna eccezione è stata generata.
- **Expected results**: la mappa contiene tutte le coppie della mappa t.

16° PutAllExceptionsTest():

- Test del metodo public void putAll(HMap t)
- **Summary / Description**: il lancio dell'eccezione viene prima verificato su una mappa vuota, viene poi inserita una coppia e si verifica nuovamente il corretto funzionamento del metodo. Viene poi verificato che il valore inserito in precedenza non abbia subito modifiche.
- **Design**: verifica il corretto lancio dell'eccezione NullPointerException sia quando la mappa è vuota sia quando contiene degli elementi.
- **Pre-Condition**: un oggetto MapAdapter vuoto e correttamente inizializzato.
- **Post-Condition**: la mappa contiene un elemento che non è stato modificato dal lancio delle eccezioni. Nessun altra eccezione è stata generata.
- **Expected results**: il corretto lancio di due NullPointerException quando il parametro passato è "null".

17° RemoveTest():

- Test del metodo public Object remove(Object key)
- **Summary / Description**: la mappa è inizialmente vuota e si verifica che il metodo applicato ad un mappa vuota restituisca null. La mappa viene poi riempita con dei valori di riferimento e si verifica il corretto comportamento del metodo invocandolo sia con chiavi presenti nella mappa sia con chiavi non presenti. Il metodo viene poi invocato su tutte le chiavi rimanenti della mappa fino a svuotarla completamente verificando che i valori restituiti non siano effettivamente più presenti. Si verifica che la dimensione della mappa venga aggiornata correttamente.
- **Design**: applicare più volte il metodo su una mappa riempita con valori di riferimento per verificarne la correttezza.
- **Pre-Condition**: un oggetto MapAdapter vuoto e correttamente inizializzato. Un parametro “key” diverso da “null”.
- **Post-Condition**: la mappa torna ad essere vuota. Ulteriori invocazioni del metodo restituiranno il valore null.
- **Expected results**: il metodo ritorna un valore diverso da “null” se e solo se è stato un rimosso una coppia dalla mappa. La dimensione della mappa viene aggiornata correttamente.

18° RemoveExceptionsTest():

- Test del metodo public Object remove(Object key)
- **Summary / Description**: il lancio dell'eccezione viene prima verificato su una mappa vuota, viene poi inserita una coppia e si verifica nuovamente il corretto funzionamento del metodo. Viene poi verificato che il valore inserito in precedenza non abbia subito modifiche.
- **Design**: verifica il corretto lancio dell'eccezione NullPointerException sia quando la mappa è vuota sia quando contiene degli elementi.
- **Pre-Condition**: un oggetto MapAdapter vuoto e correttamente inizializzato.
- **Post-Condition**: la mappa contiene un elemento che non è stato modificato dal lancio delle eccezioni. Nessun altra eccezione è stata generata.
- **Expected results**: il corretto lancio di due NullPointerException quando il parametro passato è "null".

19° SizeTest():

- Test del metodo public int size()
- **Summary / Description**: verifica che il metodo ritorni zero per una mappa vuota. Alla mappa vengono poi aggiunti e tolti degli elementi e si verifica il corretto comportamento del metodo. La mappa viene infine svuotata e si verifica che il metodo restituisca zero.
- **Design**: aggiunta e rimozione di elementi dalla mappa per verificare che il metodo descrive fedelmente il numero di elementi contenuti all'interno della mappa.
- **Pre-Condition**: un oggetto MapAdapter vuoto e correttamente inizializzato.
- **Post-Condition**: la mappa torna ad essere vuota. Ulteriori chiamate al metodo restituiscono sempre il valore zero.
- **Expected results**: il metodo restituisce un valore che rispecchia fedelmente il numero di elementi della mappa.

20° ValuesTest():

- Test del metodo public HCollection values()
- **Summary / Description**: verifica inizialmente il funzionamento del metodo su una mappa vuota. La mappa viene poi riempita e il metodo invocato nuovamente. Si verifica che la HCollection restituita abbia la stessa dimensione della mappa e che gli elementi al suo interno siano effettivamente dei valori della mappa come richiesto dall'interfaccia.
- **Design**: verifica che il metodo funzioni correttamente a prescindere dalle dimensioni della mappa su cui viene invocato.
- **Pre-Condition**: un oggetto MapAdapter vuoto e correttamente inizializzato.
- **Post-Condition**: la mappa e la collection vengono inizializzati con successo.
- **Expected results**: il metodo restituisce sempre una collection contenente i valori presenti nella mappa.

21° EntryEqualsTest():

- Test del metodo public boolean equals(Object o) di HEntry implements HMap.Entry
- **Summary / Description**: vengono inizializzate due entry con dei valori di riferimento uguali. Si verifica che il metodo restituisca true e che sia commutativo. Vengono poi invertiti i campi chiave-valore di una entry e si verifica ora che il confronto tra le due restituisca false. I valori dell'entry vengono modificati ulteriormente e si verifica nuovamente il corretto funzionamento del metodo. Si verifica inoltre il corretto funzionamento del metodo quando il parametro è "null" o quando questo non è un oggetto HMap.Entry.
- **Design**: verifica la correttezza del metodo confrontando due entry inizializzate con dei valori di riferimento.
- **Pre-Condition**: un oggetto HMap.Entry da inizializzare.
- **Post-Condition**: le due entry hanno diversi campi chiave-valore. Il loro confronto restituirà sempre false.
- **Expected results**: il metodo restituisce true se e solo se: e1.getKey() == e2.getKey() && e1.getValue() == e2.getValue()

22° EntryGettersTest():

- Test dei metodi public Object getKey(Object o) e public Object getValue(Object o) di HEntry implements HMap.Entry
- **Summary / Description**: dopo aver inizializzato un entry con dei valori di riferimento si verifica che i valori restituiti dai due getters siano coerenti con quelli precedentemente inseriti.
- **Design**: verifica la correttezza del metodo tramite confronto dei valori restituiti con dei valori di riferimento.
- **Pre-Condition**: un oggetto HMap.Entry da inizializzare.
- **Post-Condition**: l'oggetto entry contiene la coppia "uno"-1.
- **Expected results**: i getters restituiscono il corretto campo dell'entry sottostante senza modificare il valore dell'altro campo.

23° EntryHashCodeTest():

- Test del metodo public int hashCode() di HEntry implements HMap.Entry
- **Summary / Description**: vengono inizializzate due entry agli stessi valori e si verifica che queste siano effettivamente uguali. Si verifica poi che l'hashcode delle due entry risulta essere uguale. Viene poi modificato, allo stesso valore, il campo value entrambe entry e si verifica che il metodo funzioni correttamente anche in questo caso.
- **Design**: verifica che il metodo rispecchi il comportamento definito dall'interfaccia quando si confrontano due entry uguali tra loro.
- **Pre-Condition**: un oggetto HMap.Entry da inizializzare.
- **Post-Condition**: le due entry sono uguali, i due hashcode restituiti sono uguali.
- **Expected results**: date due entri tale che e1.equals(e2) allora necessariamente e1.hashCode() == e2.hashCode()

24° EntrySetValueTest():

- Test del metodo public Object setValue(Object o) di HEntry implements HMap.Entry
- **Summary / Description**: viene inizializzata una nuova entry e se ne modifica il valore. Si verifica che il nuovo valore sia stato aggiornato sia nella entry sia nella mappa sottostante, si verifica inoltre che il valore restituito dal metodo coincida con il precedente campo value dell'entry e che questo valore non sia effettivamente più presente all'interno della mappa sottostante. Si verifica che l'operazione non abbia influito sul campo key dell'entry.
- **Design**: verifica la corretta modifica del campo value sia nell'entry sia nella mappa sottostante.
- **Pre-Condition**: un oggetto HMap.Entry da inizializzare.
- **Post-Condition**: l'entry e la mappa contengono il nuovo valore. Il vecchio valore è stato sovrascritto.
- **Expected results**: sia nella mappa sia nell'entry il campo value è stato correttamente aggiornato senza modificare nessun altro valore. Il metodo restituisce correttamente il valore sovrascritto.

25° SetAddTest():

- Test del metodo public void add(Object o) ed public void addAll(HCollection c) di EntrySet, KeySet, ValueSet implements HSet
- **Summary / Description**: verifica che l'invocazione dei metodi add(Object o) ed addAll(HCollection c) su un oggetto di tipo HSet generi l'eccezione UnsupporteOperationsException come specificato dall'interfaccia. Verifica che il lancio dell'eccezione non modifica gli altri valori. Il test viene ripetuto dopo aver svuotato il set.
- **Design**: verifica il corretto lancio dell'eccezione sia quando il set è vuoto sia quando questo contiene degli elementi.
- **Pre-Condition**: un oggetto HSet correttamente inizializzato e relativo ad una mappa di 8 elementi.
- **Post-Condition**: il set e la mappa tornano ad essere vuoti, Nessun altra eccezione è stata generata.
- **Expected results**: i due metodi non sono supportati dagli oggetti di tipo HSet.

26° SetClearTest():

- Test del metodo public void clear() di EntrySet, KeySet, ValueSet implements HSet
- **Summary / Description**: verifica che il metodo elimini tutti gli elementi sia del set sia della mappa sottostante. Verifica che la dimensione del set e della mappa vengano modificate correttamente e che tutti gli elementi siano effettivamente rimossi.
- **Design**: verifica la correttezza del metodo sia su un set vuoto sia su un set contenente degli elementi.
- **Pre-Condition**: un oggetto HSet correttamente inizializzato e relativo ad una mappa di 8 elementi.
- **Post-Condition**: il set e la mappa sottostante tornano ad essere vuoti.
- **Expected results**: rimuove correttamente tutti gli elementi sia dal set che dalla mappa sottostante.

27° EntrySetContainsTest():

- Test del metodo public boolean contains(Object o) di EntrySet implements HSet
- **Summary / Description**: partendo da un set contenente valori noti verifica inizialmente che, data una entry di riferimento come parametro al metodo, restituisca true. Si verifica poi che la entry in questione sia effettivamente presente nella mappa sottostante. Verifica che il metodo restituisca false quando usato su oggetti di classi diverse ha HMap.Entry. Si verifica poi il corretto comportamento rispetto ad una entry non presente sulla mappa e il corretto comportamento su un set/mappa vuoto. Verifica che il metodo non modifichi le dimensioni del set od altri valori presenti.
- **Design**: verifica il corretto comportamento del metodo confrontando i risultati ottenuti usando come parametri valori di riferimento.
- **Pre-Condition**: un oggetto EntrySet correttamente inizializzato e relativo ad una mappa di 8 elementi. Un parametro “o” diverso da “null”.
- **Post-Condition**: il set torna ad essere vuoto, il metodo restituirà sempre false.
- **Expected results**: il metodo restituisce true se e solo se il parametro passato è effettivamente presente all’interno del set (e quindi della mappa sottostante).

28° EntrySetContainsExceptionsTest():

- Test del metodo public boolean contains(Object o) di EntrySet implements HSet
- **Summary / Description**: verifica che il metodo generi l'eccezione NullPointerException quando viene invocato con un parametro pari a "null". Questo deve essere vero sia quando il set è vuoto sia quando questo contiene altri elementi. Eventuali elementi presenti all'interno del set e della mappa sottostante non devono essere modificati.
- **Design**: verifica lancio dell'eccezione NullPointerExceptions come specificato dall'interfaccia.
- **Pre-Condition**: un oggetto EntrySet correttamente inizializzato e relativo ad una mappa di 8 elementi.
- **Post-Condition**: il set e la mappa tornano ad essere vuote, nessun altra eccezione è stata generata.
- **Expected results**: il metodo genere NullPointerExceptions quando invocato con parametro "null".

29° KeySetContainsTest():

- Test del metodo public boolean contains(Object o) di KeySet implements HSet
- **Summary / Description**: partendo da un set contenente valori noti verifica inizialmente che, data una chiave di riferimento come parametro al metodo, restituisca true. Si verifica poi che la chiave in questione sia effettivamente presente nella mappa sottostante. Si verifica poi il corretto comportamento rispetto ad una chiave non presente sulla mappa e il corretto comportamento su un set/mappa vuoto. Verifica che il metodo non modifichi le dimensioni del set od altri valori presenti.
- **Design**: verifica il corretto comportamento del metodo confrontando i risultati ottenuti usando come parametri valori di riferimento.
- **Pre-Condition**: un oggetto KeySet correttamente inizializzato e relativo ad una mappa di 8 elementi. Un parametro “o” diverso da “null”.
- **Post-Condition**: il set torna ad essere vuoto, il metodo restituirà sempre false.
- **Expected results**: il metodo restituisce true se e solo se il parametro passato è effettivamente presente all’interno del set (e quindi della mappa sottostante).

30° KeySetContainsExceptionsTest():

- Test del metodo public boolean contains(Object o) di KeySet implements HSet
- **Summary / Description**: verifica che il metodo generi l'eccezione NullPointerException quando viene invocato con un parametro pari a "null". Questo deve essere vero sia quando il set è vuoto sia quando questo contiene altri elementi. Eventuali elementi presenti all'interno del set e della mappa sottostante non devono essere modificati.
- **Design**: verifica lancio dell'eccezione NullPointerExceptions come specificato dall'interfaccia.
- **Pre-Condition**: un oggetto KeySet correttamente inizializzato e relativo ad una mappa di 8 elementi.
- **Post-Condition**: il set e la mappa tornano ad essere vuote, nessun altra eccezione è stata generata.
- **Expected results**: il metodo genere NullPointerExceptions quando invocato con parametro "null".

31° ValueSetContainsTest():

- Test del metodo public boolean contains(Object o) di ValueSet implements HSet
- **Summary / Description**: partendo da un set contenente valori noti verifica inizialmente che, dato un valore di riferimento come parametro al metodo, restituisca true. Si verifica poi che il valore in questione sia effettivamente presente nella mappa sottostante. Si verifica poi il corretto comportamento rispetto ad un valore non presente sulla mappa e il corretto comportamento su un set/mappa vuoto. Verifica che il metodo non modifichi le dimensioni del set od altri valori presenti.
- **Design**: verifica il corretto comportamento del metodo confrontando i risultati ottenuti usando come parametri valori di riferimento.
- **Pre-Condition**: un oggetto ValueSet correttamente inizializzato e relativo ad una mappa di 8 elementi. Un parametro “o” diverso da “null”.
- **Post-Condition**: il set torna ad essere vuoto, il metodo restituirà sempre false.
- **Expected results**: il metodo restituisce true se e solo se il parametro passato è effettivamente presente all’interno del set (e quindi della mappa sottostante).

32° ValueSetContainsExceptionsTest():

- Test del metodo public boolean contains(Object o) di ValueSet implements HSet
- **Summary / Description**: verifica che il metodo generi l'eccezione NullPointerException quando viene invocato con un parametro pari a "null". Questo deve essere vero sia quando il set è vuoto sia quando questo contiene altri elementi. Eventuali elementi presenti all'interno del set e della mappa sottostante non devono essere modificati.
- **Design**: verifica lancio dell'eccezione NullPointerExceptions come specificato dall'interfaccia.
- **Pre-Condition**: un oggetto ValueSet correttamente inizializzato e relativo ad una mappa di 8 elementi.
- **Post-Condition**: il set e la mappa tornano ad essere vuote, nessun altra eccezione è stata generata.
- **Expected results**: il metodo genere NullPointerExceptions quando invocato con parametro "null".

33° SetContainsAllTest():

- Test del metodo public boolean containsAll(HCollection c) di EntrySet, KeySet, ValueSet implements HSet
- **Summary / Description**: oltre al set viene creata una seconda HCollection, inizialmente vuota, che verrà usata come parametro per il metodo. Si verifica che passando una HCollection vuota come parametro il metodo restituisce true. Viene poi riempita la collection con gli stessi valori contenuti nel set e si verifica che il metodo restituisca true. Viene poi rimosso un elemento dal set verificato che il metodo restituisca false. Vengono poi rimossi dalla collection lo stesso elemento più un secondo elemento e si verifica che il metodo torni a restituire true. Vengono svuotati sia la collection che il set e si verifica che il metodo operi correttamente anche su oggetti vuoti
- **Design**: si verifica il corretto funzionamento del metodo invocandolo su una HCollection di valori di riferimento.
- **Pre-Condition**: un oggetto HSet correttamente inizializzato e contenente 8 elementi.
- **Post-Condition**: il set torna ad essere vuoto, nuove invocazioni del metodo restituiranno true solo quando viene passato come parametro una HCollection vuota.
- **Expected results**: il metodo restituisce true se e solo se il set contiene tutti gli elementi della HCollection o se questa è vuota.

34° SetContainsAllExceptionsTest():

- Test del metodo public boolean containsAll(HCollection c) di EntrySet, KeySet, ValueSet implements HSet
- **Summary / Description**: verifica che il metodo generi l'eccezione NullPointerException quando viene invocato con un parametro pari a "null". Questo deve essere vero sia quando il set è vuoto sia quando questo contiene altri elementi. Eventuali elementi presenti all'interno del set e della mappa sottostante non devono essere modificati.
- **Design**: verifica lancio dell'eccezione NullPointerExceptions come specificato dall'interfaccia.
- **Pre-Condition**: un oggetto HSet correttamente inizializzato e relativo ad una mappa di 8 elementi.
- **Post-Condition**: il set e la mappa tornano ad essere vuote, nessun altra eccezione è stata generata.
- **Expected results**: il metodo genere NullPointerExceptions quando invocato con parametro "null".

35° SetEqualsTest():

- Test del metodo public boolean equals(Object o) di di EntrySet, KeySet, ValueSet implements HSet
- **Summary / Description**: vengono inizializzate due set con dei valori di riferimento uguali. Si verifica che il metodo restituisca:
    - False quando vengono confrontati set aventi un diverso numero di elementi
    - False quando vengono confrontati set aventi elementi diversi al loro interno
    - False quando il parametro non è un istanza di HSet od è pari a null
    - True quando si confrontano due set vuoti
- **Design**: verifica la correttezza del metodo confrontando due set inizializzati con dei valori di riferimento.
- **Pre-Condition**: un oggetto HSet già inizializzato e contenente 8 valori.
- **Post-Condition**: il set torna ad essere vuoto.
- **Expected results**: il metodo restituisce true se e solo se il parametro è anch'esso un set ed entrambi contengono gli stessi valori.

36° SetHashCodeTest():

- Test del metodo public int hashCode() di EntrySet, KeySet, ValueSet implements HSet
- **Summary / Description**: viene verificato che, date due set:
    - Il metodo sia sempre applicabile ad un set correttamente inzializzato.
    - Dopo aver riempito due set con gli stessi elementi questi hanno lo stesso hashCode.
    - Dopo aver estratto un elemento da un set i due hashCode sono distinti.
    - Dopo aver rimosso anche dall'altro set lo stesso elemento questi tornano ad avere lo stesso hashCode.
    - Mappe vuote hanno un hashCode pari a zero.
- **Design**: verifica che il metodo si comporti adeguatamente in tutte le condizioni esaminando due set contenenti degli elementi di riferimento.
- **Pre-Condition**: un oggetto HSet correttamente inizializzato e contenente 8 elementi.
- **Post-Condition**: i due set tornano ad essere vuoti, il loro hashCode è uguale a zero
- **Expected results**: il metodo soddisfa la condizione: se s1.equals(s2) allora s1.hashCode() == s2.hashCode().

37° SetIsEmptyTest():

- Test del metodo public boolean isEmpty() di EntrySet, KeySet, ValueSet implements HSet
- **Summary / Description**: viene inizialmente verificato che per un set contentenente degli elementi il metodo restituisca false. Il set viene poi svuotato e si verifica che il metodo torni a restituire true.
- **Design**: verifica il corretto comportamento del metodo sia quando il set è vuoto sia quando questo contiene degli elementi.
- **Pre-Condition**: un oggetto HSet correttamente inizializzato e contenente 8 elementi.
- **Post-Condition**: il set torna ad essere vuoto, nuove invocazioni del metodo restituiranno sempre true.
- **Expected results**: il metodo restituisce true se e solo se il set e la mappa sottostante sono effettivamente vuoti.

38° SetIteratorTest():

- Test del metodo public HIterator iterator() di EntrySet, KeySet, ValueSet implements HSet
- **Summary / Description**: verifica che il metodo funzioni sempre e che l'iteratore restituito sia congruente al tipo di set che effettua la chiamata. In particolare verifica che:
    - L'iteratore ottenuto dalla chiamata da parte di un EntrySet restituisce istanze di HMap.Entry contenuti all'interno del set.
    - L'iteratore ottenuto dalla chiamata da parte di un KeySet restituisce chiavi contenute all'interno del set e della mappa sottostante.
    - L'iteratore ottenuto dalla chiamata da parte di un ValueSet restituisce valori contenuti all'interno del set e della mappa sottostante.
    - Per tutti e tre i tipi di set il metodo funzioni anche quando la mappa sottostante è vuota.
- **Design**: verifica che il metodo funzioni correttamente a prescindere dalle dimensioni della mappa su cui viene invocato.
- **Pre-Condition**: un oggetto HSet correttamente inizializzato e contenente 8 elementi.
- **Post-Condition**: l'iteratore viene sempre istanziato con successo.
- **Expected results**: il metodo restituisce sempre un iteratore sugli elementi del set che lo invoca.

39° EntrySetRemoveTest():

- Test del metodo public boolean remove(Object o) di EntrySet implements HSet
- **Summary / Description**: partendo da un set contenente dei valori di riferimento viene inizialmente rimosso un primo valore da questo. Si verifica che il valore non sia più presente né nel set né nella mappa sottostante. Si verifica poi che, tentando di rimuovere un elemento non presente nella mappa (lo stesso valore appena rimosso nello specifico) il metodo restituisce false e mappa e set rimangono invariati. Viene poi invocato il metodo su tutti gli elementi del set fino a svuotarlo del tutto.
- **Design**: verifica la correttezza del metodo andando a rimuovere dei valori di riferimento dal set verificando che questi siano effettivamente rimossi anche dalla mappa sottostante.
- **Pre-Condition**: un oggetto EntrySet correttamente inizializzato e relativo ad una mappa di 8 elementi, un parametro “o” diverso da “null”.
- **Post-Condition**: il set e la mappa tornano ad essere vuote, ulteriori invocazioni del metodo ritorneranno sempre false.
- **Expected results**: il metodo restituisce true se e solo se ha eliminato correttamente un valore dal set e dalla mappa sottostante. Modifica congruentemente la dimensione del set e della mappa.

40° EntrySetRemoveExceptionsTest():

- Test del metodo public boolean remove(Object o) di EntrySet implements HSet
- **Summary / Description**: verifica che il metodo generi l'eccezione NullPointerException quando viene invocato con un parametro pari a "null". Questo deve essere vero sia quando il set è vuoto sia quando questo contiene altri elementi. Eventuali elementi presenti all'interno del set e della mappa sottostante non devono essere modificati.
- **Design**: verifica lancio dell'eccezione NullPointerExceptions come specificato dall'interfaccia.
- **Pre-Condition**: un oggetto EntrySet correttamente inizializzato e relativo ad una mappa di 8 elementi.
- **Post-Condition**: il set e la mappa tornano ad essere vuote, nessun altra eccezione è stata generata.
- **Expected results**: il metodo genere NullPointerExceptions quando invocato con parametro "null".

41° KeySetRemoveTest():

- Test del metodo public boolean remove(Object o) di KeySet implements HSet
- **Summary / Description**: partendo da un set contenente dei valori di riferimento viene inizialmente rimosso un primo valore da questo. Si verifica che il valore non sia più presente né nel set né nella mappa sottostante. Si verifica poi che, tentando di rimuovere un elemento non presente nella mappa (lo stesso valore appena rimosso nello specifico) il metodo restituisce false e mappa e set rimangono invariati. Viene poi invocato il metodo su tutti gli elementi del set fino a svuotarlo del tutto.
- **Design**: verifica la correttezza del metodo andando a rimuovere dei valori di riferimento dal set verificando che questi siano effettivamente rimossi anche dalla mappa sottostante.
- **Pre-Condition**: un oggetto KeySet correttamente inizializzato e relativo ad una mappa di 8 elementi, un parametro “o” diverso da “null”.
- **Post-Condition**: il set e la mappa tornano ad essere vuote, ulteriori invocazioni del metodo ritorneranno sempre false.
- **Expected results**: il metodo restituisce true se e solo se ha eliminato correttamente un valore dal set e dalla mappa sottostante. Modifica congruentemente la dimensione del set e della mappa.

42° KeySetRemoveExceptionsTest():

- Test del metodo public boolean remove(Object o) di KeySet implements HSet
- **Summary / Description**: verifica che il metodo generi l'eccezione NullPointerException quando viene invocato con un parametro pari a "null". Questo deve essere vero sia quando il set è vuoto sia quando questo contiene altri elementi. Eventuali elementi presenti all'interno del set e della mappa sottostante non devono essere modificati.
- **Design**: verifica lancio dell'eccezione NullPointerExceptions come specificato dall'interfaccia.
- **Pre-Condition**: un oggetto KeySet correttamente inizializzato e relativo ad una mappa di 8 elementi.
- **Post-Condition**: il set e la mappa tornano ad essere vuote, nessun altra eccezione è stata generata.
- **Expected results**: il metodo genere NullPointerExceptions quando invocato con parametro "null".

43° ValueSetRemoveTest():

- Test del metodo public boolean remove(Object o) di ValueSet implements HSet
- **Summary / Description**: partendo da un set contenente dei valori di riferimento viene inizialmente rimosso un primo valore da questo. Si verifica che il valore non sia più presente né nel set né nella mappa sottostante. Si verifica poi che, tentando di rimuovere un elemento non presente nella mappa (lo stesso valore appena rimosso nello specifico) il metodo restituisce false e mappa e set rimangono invariati. Viene poi invocato il metodo su tutti gli elementi del set fino a svuotarlo del tutto.
- **Design**: verifica la correttezza del metodo andando a rimuovere dei valori di riferimento dal set verificando che questi siano effettivamente rimossi anche dalla mappa sottostante.
- **Pre-Condition**: un oggetto ValueSet correttamente inizializzato e relativo ad una mappa di 8 elementi, un parametro “o” diverso da “null”.
- **Post-Condition**: il set e la mappa tornano ad essere vuote, ulteriori invocazioni del metodo ritorneranno sempre false.
- **Expected results**: il metodo restituisce true se e solo se ha eliminato correttamente un valore dal set e dalla mappa sottostante. Modifica congruentemente la dimensione del set e della mappa.

44° ValueSetRemoveExceptionsTest():

- Test del metodo public boolean remove(Object o) di ValueSet implements HSet
- **Summary / Description**: verifica che il metodo generi l'eccezione NullPointerException quando viene invocato con un parametro pari a "null". Questo deve essere vero sia quando il set è vuoto sia quando questo contiene altri elementi. Eventuali elementi presenti all'interno del set e della mappa sottostante non devono essere modificati.
- **Design**: verifica lancio dell'eccezione NullPointerExceptions come specificato dall'interfaccia.
- **Pre-Condition**: un oggetto ValueSet correttamente inizializzato e relativo ad una mappa di 8 elementi.
- **Post-Condition**: il set e la mappa tornano ad essere vuote, nessun altra eccezione è stata generata.
- **Expected results**: il metodo genere NullPointerExceptions quando invocato con parametro "null".

45° SetRemoveAllTest():

- Test del metodo public boolean removeAll(HCollection c) di EntrySet, KeySet, ValueSet implements HSet
- **Summary / Description**: viene creata una HCollection da usare come parametro per il metodo. Sul set, inizialmente contenente otto elementi, viene invocato il metodo passando una collection vuota e si verifica che il metodo restituisca false e che la dimensione del set non sia cambiata. Vengono poi aggiunti alcuni elementi alla collection, di cui alcuni presenti anche nel set, e si verifica che, invocando nuovamente il metodo, questi vengano rimossi sia dal set che dalla mappa sottostante. Viene infine riempita la collection con tutti gli elementi presenti originariamente nel set e si verifica che, l'invocazione del metodo svuoti completamente set e mappa sottostante.
- **Design**: viene invocato il metodo sfruttando una HCollection contenente dei valori di riferimento per verificarne il corretto comportamento.
- **Pre-Condition**: un oggetto HSet correttamente inizializzato e contenente 8 elementi. Un parametro HCollection diverso da "null".
- **Post-Condition**: il set torna ad essere vuoto, ulteriori chiamate al metodo restituiranno sempre false.
- **Expected results**: il metodo rimuove dal set e dalla mappa sottostante tutti i valori contenuti all'interno della HCollection passata come parametro, le dimensioni vengono adattate coerentemente.

46° SetRemoveAllExceptionsTest():

- Test del metodo public boolean removeAll(HCollection c) di EntrySet, KeySet, ValueSet implements HSet
- **Summary / Description**: verifica che il metodo generi l'eccezione NullPointerException quando viene invocato con un parametro pari a "null". Questo deve essere vero sia quando il set è vuoto sia quando questo contiene altri elementi. Eventuali elementi presenti all'interno del set e della mappa sottostante non devono essere modificati.
- **Design**: verifica lancio dell'eccezione NullPointerExceptions come specificato dall'interfaccia.
- **Pre-Condition**: un oggetto HSet correttamente inizializzato e relativo ad una mappa di 8 elementi.
- **Post-Condition**: il set e la mappa tornano ad essere vuote, nessun altra eccezione è stata generata.
- **Expected results**: il metodo genere NullPointerExceptions quando invocato con parametro "null".

47° SetRetainAllTest():

- Test del metodo public boolean retainAll(HCollection c) di EntrySet, KeySet, ValueSet implements HSet
- **Summary / Description**: viene creata una HCollection da usare come parametro e questa viene riempita con gli stessi elementi che sono stati inseriti nel set in fase di inizializzazione. Viene testato il metodo usando la collection appena descritta come parametro e si verifica che nessun elemento viene eliminato. Viene poi svuotata la collection e vengono inseriti al suo interno dei nuovi elementi, alcuni dei quali presenti anche nel set. Si invoca nuovamente il metodo e si verifica che vengano eliminati tutti gli elementi tranne quelli presenti sia nel set che nella collection. Viene infine svuotata la collection e si verifica che, una nuova invocazione del metodo svuoti completamente la lista.
- **Design**: verifica il corretto comportamento del metodo invocandolo con una HCollection contenente dei valori di riferimento.
- **Pre-Condition**: un oggetto HSet correttamente inizializzato e contenente 8 elementi. Un parametro HCollection diverso da “null”
- **Post-Condition**: il set torna ad essere vuoto, ulteriori invocazioni del metodo restituiranno sempre false.
- **Expected results**: il metodo mantiene all’interno del set i soli elementi appartenenti all’intersezione tra questo e la HCollection passata come parametro. Il metodo restituisce tre se e solo se la dimensione del set viene modificata.

48° SetRetainAllExceptionsTest():

- Test del metodo public boolean retainAll(HCollection c) di EntrySet, KeySet, ValueSet implements HSet
- **Summary / Description**: verifica che il metodo generi l'eccezione NullPointerException quando viene invocato con un parametro pari a "null". Questo deve essere vero sia quando il set è vuoto sia quando questo contiene altri elementi. Eventuali elementi presenti all'interno del set e della mappa sottostante non devono essere modificati.
- **Design**: verifica lancio dell'eccezione NullPointerExceptions come specificato dall'interfaccia.
- **Pre-Condition**: un oggetto HSet correttamente inizializzato e relativo ad una mappa di 8 elementi.
- **Post-Condition**: il set e la mappa tornano ad essere vuote, nessun altra eccezione è stata generata.
- **Expected results**: il metodo genere NullPointerExceptions quando invocato con parametro "null".

49° SetSizeTest():

- Test del metodo public int size() di EntrySet, KeySet, ValueSet implements HSet
- **Summary / Description**: verifica che applicato ad un set contentente 8 elementi il metodo ritorni 8. Al set vengono poi rimossi ed aggiunti degli elementi e si verifica il corretto comportamento del metodo e che questo sia consistente con il metodo size() della mappa sottostante. Il set viene infine svuotato e si verifica che il metodo restituisca zero.
- **Design**: aggiunta e rimozione di elementi dal set per verificare che il metodo descrive fedelmente il numero di elementi contenuti all'interno del set e della mappa sottostante.
- **Pre-Condition**: un oggetto HSet correttamente inizializzato contentente 8 elementi.
- **Post-Condition**: la mappa torna ad essere vuota. Ulteriori chiamate al metodo restituiscono sempre il valore zero.
- **Expected results**: il metodo restituisce un valore che rispecchia fedelmente il numero di elementi del set e della mappa sottostante.

50° SetToArrayTest():

- Test del metodo public Object[] toArray() di EntrySet, KeySet, ValueSet implements HSet
- **Summary / Description**: viene invocato il metodo per creare un array contentente tutti gli elementi del set. Si verifica che l'array restituito abbia dimensione pari a quella del set e che tutti gli elementi al suo interno siano effettivamente contenuti nel set. Verifica che se applicato su un set vuoto restituisce un set vuoto.
- **Design**: verifica che il metodo restituisca un array contenente tutti gli elementi contenuti all'interno della lista senza modificare gli elementi al suo interno.
- **Pre-Condition**: un oggetto HSet correttamente inizializzato contenente 8 elementi.
- **Post-Condition**: il set torna ad essere vuoto. Invocare nuovamente il metodo restituisce una array vuoto.
- **Expected results**: l'array restituito contiene tutti gli elementi presenti all'interno del set. Gli elementi all'interno del set e della mappa sottostante non subiscono modifiche.

51° SetToArrayObjectTest():

- Test del metodo public Object[] toArray(Object[] a) di EntrySet, KeySet, ValueSet implements HSet
- **Summary / Description**: viene testato il metodo sia su un array di dimensione minore rispetto al set sia su un array di dimensione maggiore uguale al set. Si verifica che:
    - Array troppo piccolo: l'array restituito ha ora dimensione uguale a quella del set e contiene tutti i sui elementi.
    - Array sufficientemente grande: viene restituito lo stesso array dove i primi elementi sono stati sovrascritti con quelli del set. La sua dimensione ed eventuali altri elementi non sono modificati.
    - Verifica che in entrambi i casi gli elementi contenuti nel set non sono stati modificati.
- **Design**: verifica la corretta creazione di un array contenente tutti gli elementi presenti nel set nel caso in cui l'array passato come parametro fosse troppo piccolo. Verifica altrimenti che l'array fornito venga riempito/sovrascritto adeguatamente.
- **Pre-Condition**: un oggetto HSet correttamente inizializzato contenente 8 elementi.
- **Post-Condition**: il set contiene tutti gli elementi originari e non viene modificato in nessun modo.
- **Expected results**: il metodo restituisce un array ordinato contenente tutti gli elementi contenuti nel set. Il set non viene modificato in nessun modo dall'invocazione del metodo.

52° SetToArrayObjectExceptionsTest():

- Test del metodo public Object[] toArray(Object[] a) di EntrySet, KeySet, ValueSet implements HSet
- **Summary / Description**: verifica che il metodo generi l'eccezione NullPointerException quando viene invocato con un parametro pari a "null". Questo deve essere vero sia quando il set è vuoto sia quando questo contiene altri elementi. Eventuali elementi presenti all'interno del set e della mappa sottostante non devono essere modificati. Verifica inoltre che, qualora l'array passato come parametro avesse dimensione superiore a quella del set, il metodo genera ArrayStoreException se il tipo run time dell'array non è compatibile con tutti i valori contenuti nel set.
- **Design**: verifica lancio dell'eccezione NullPointerExceptions ed ArrayStoreException come specificato dall'interfaccia.
- **Pre-Condition**: un oggetto HSet correttamente inizializzato e relativo ad una mappa di 8 elementi.
- **Post-Condition**: il set e la mappa tornano ad essere vuote, nessun altra eccezione è stata generata.
- **Expected results**: il metodo genere NullPointerExceptions quando invocato con parametro "null" e genera ArrayStoreException se il tipo run time dell'array passato come parametro non è compatibile con tutti gli oggetti contenuti nel set.

53° SetIteratorHasNextTest():

- Test del metodo public boolean hasNext() di EntryIterator, KeyValIterator implements HIteraor
- **Summary / Description**: su una set vuota, ricavo un iteratore tramite il metodo iterator(), si verifica che il metodo hasNext() restituisca false. Vengono poi inseriti degli elementi nella mappa sottostante al set e viene ricreato nuovamente l'iteratore. Avanzando sulla sut si verifica che il metodo hasNext() restituisce true esattamente tante volte quante sono gli elementi della mappa.
- **Design**: verifica il corretto funzionamento chiamando il metodo su un iteratore che scorre sugli elementi del set.
- **Pre-Condition**: una oggetto HSet correttamente inizializzato.
- **Post-Condition**: il set contiene gli elementi inseriti, nessuno di questi è stato modificato, l'iteratore si trova alla fine del processo di iterazione.
- **Expected results**: verifica che il metodo resistituisce true se e solo se l'iteratore deve ancora visitare alcuni elementi del set.

54° SetIteratorRemoveTest():

- Test del metodo public Object remove() di EntryIterator, KeyValIterator implements HIterator
- **Summary / Description**: vengono inseriti degli elementi nella mappa sottostante al set e viene creato un iteratore. Viene alternata una chiamata ad next con una chiamata ad remove e si verifica che l'oggetto rimosso dalla lista corrisponde all'ultimo oggetto restituito dal metodo next. Si verifica che alla fine del processo sia il set, sia la mappa sottostante siano vuoti.
- **Design**: chiamate ripetute al metodo (alternate da chiamate a next) per verificarne il giusto comportamento eliminando tutti gli elementi del set.
- **Pre-Condition**: una oggetto HSet correttamente inizializzato ed inizialmente vuoto.
- **Post-Condition**: il set è tornata ad essere vuota, l'iteratore si trova alla fine del processo di iterazione.
- **Expected results**: verifica che il metodo elimina dal set il valore restituito dalla precedente chiamata al metodo next.

55° SetIteratorRemoveExceptionsTest():

- Test del metodo public Object remove() di EntryIterato, KeyValIterator implements HIterator
- **Summary / Description**: si verifica che il metodo generi l'eccezione quando:
    - viene invocato su un set vuoto
    - quando viene invocato prima del metodo next
    - quando sono invocate due chiamate a remove di fila.
- **Design**: verifica che il metodo lancia eccezioni quando non viene intervallato da chiamate a next, viene chiamato su set vuoti.
- **Pre-Condition**: un oggetto HSet vuoto e correttamente inizializzato.
- **Post-Condition**: il lancio delle eccezioni non modifica il contenuto del set. Dimensione ed eventuali valori già presenti rimangono inalterati.
- **Expected results**: il lancio dell' eccezione IllegalStateException quando si tenta di rimuovere più di un elemento per volta.

56° SetEntryIteratorNextTest():

- Test del metodo public Object next() di EntryIterator implements HIterator
- **Summary / Description**: vengono inseriti degli elementi nella mappa sottostante al set e viene creato un iteratore. Si verifica che il metodo next() restituisce delle istanze di HMap.Entry che corrispondono ai valori contenuti all'interno del set (e della mappa sottostante). Arrivati alla fine della processo di iterazione si verifica che un ulteriore chiamata al metodo genera l'eccezione NoSuchElementException.
- **Design**: verifica della corrispondenza tra valori restituiti e valori di riferimento mentre si effettuano chiamate ripetute al metodo finché si itera lungo il set.
- **Pre-Condition**: una oggetto HSet correttamente inizializzato.
- **Post-Condition**: il set contiene gli elementi nessuno di questi è stato modificato, l'iteratore si trova alla fine del processo di iterazione.
- **Expected results**: verifica che il metodo restituisce gli elementi contenuti all'interno del set senza modificarne il contenuto.

57° SetKeyValIteratorNextTest():

- Test del metodo public Object next() di KeyValIterator implements HIterator
- **Summary / Description**: vengono inseriti degli elementi nella mappa sottostante al set e viene creato un iteratore. Si verifica che il metodo next() restituisce:
    - Le chiavi della mappa sottostante al set quando viene usato in “modalità chiave”. (ovvero quando si ottiene l’iteratore da un KeySet)
    - I valori della mappa sottostane al set quando viene usato in “modalità value”. (ovvero quando si ottiene l’iteratore da un ValueSet)

    Arrivati alla fine della processo di iterazione si verifica che un ulteriore chiamata al metodo genera l’eccezione NoSuchElementException (in entrambe le modalità)

- **Design**: verifica della corrispondenza tra valori restituiti e valori di riferimento mentre si effettuano chiamate ripetute al metodo finché si itera lungo il set.
- **Pre-Condition**: una oggetto HSet correttamente inizializzato.
- **Post-Condition**: il set contiene gli elementi nessuno di questi è stato modificato, l’iteratore si trova alla fine del processo di iterazione.
- **Expected results**: verifica che il metodo restituisce gli elementi contenuti all’interno del set senza modificarne il contenuto.