Documentazione SetAdapterTest

Questa suite ha l'obiettivo di testare tutti i metodi di SetAdapter, alla ricerca di errori per verificarne l'assenza. I metodi vengono testati con variabili di diverso tipo e parametri validi e non, allo scopo di controllare il funzionamento delle eccezioni
Versione JUnit utilizzata per eseguire tutti i test: junit-4.13
Componenti del frame work utilizzati: assertArrayEquals, assertEquals, assertNotEquals, assertFalse, assertNull, assertTrue.

Pre condizioni generali:

Prima di ogni test, viene creato un set vuoto set1 con il costruttore di default, a cui sono stati aggiunti elementi di vario tipo, nell'ordine {null, (double) 3.3333333333333335, (int) -5, (char) L, (string) Test, (int) 7, (boolean) true}.

0. b4()

- Riassunto: @Before tramite il quale viene riempita il set set1, per essere poi utilizzata.
- Design: vengono aggiunti elementi alil set elementi di vario tipo.
- Descrizione: vengono aggiunti a set1 elementi di vario tipo: nell'ordine {null, (double) 3.3333333333333335, (int) -5, (char) L, (string) Test, (int) 7, (boolean) true}.
- Pre condizioni: set vuoto set1.
- Post condizioni: set set1 pieno.
- Risultato atteso: set set1 pieno.

1. constructorNotNull()

- Riassunto: test del costruttore setAdapter() per verificare che l'oggetto setAdapter creato non sia nullo.
- Design: viene creato il set e verificato che non sia nullo.
- Descrizione: viene creato un nuovo set con il costruttore di default e viene verificato che non sia nullo.
- Pre condizioni: set pieno set1.
- Post condizioni: set set1 torna alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: set2 non nullo.

2. constructorEquals()

- Riassunto: test del costruttore setAdapter() per verificare due nuovi set vuoti siano uguali.
- Design: vengono create due set vuote e viene verificato che siano uguali.
- Descrizione: dopo aver svuotato set 1, viene creato un set 2 e viene verificato che siano uguali
- Pre condizioni: set pieno set1.
- Post condizioni: set set1 torna alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: set2 è uguale a set3.

3. constructor2IndexOutOfBounds()

- Riassunto: test dell'eccezione IndexOutOfBounds() del costruttore setAdapter(int initialCapacity).
- Design: viene creato uno nuovo set con parametro capacità non accettabile.
- Descrizione: viene creato un nuovo set con parametro capacità non accettabile casuale compreso tra -100 e -1.
- Pre condizioni: set pieno set1.
- Post condizioni: set set1 torna alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: eccezione IndexOutOfBounds().

4. constructor2NotNull()

- Riassunto: test del costruttore setAdapter(int initialCapacity) per verificare che l'oggetto setAdapter creato non sia nullo.
- Design: viene creato il set e verificato che non sia nullo.
- Descrizione: viene creato un nuovo set con il costruttore setAdapter(int initialCapacity) e un parametro accettabile di capacità e viene verificato che non sia nullo.
- Pre condizioni: set pieno set1.
- Post condizioni: set set1 torna alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: set2 non nullo.

5. constructor3IndexOutOfBounds()

- Riassunto: test dell'eccezione IndexOutOfBounds() del costruttore setAdapter(int initialCapacity, int capacityIncrement).
- Design: viene creato un nuovo set con parametro capacità accettabile e incremento capacità non accettabile.
- Descrizione: viene creato un nuovo set con parametro capacità casuale accettabile compreso tra 0 e 100, e incremento capacità non accettabile casuale compreso tra -100 e -1.
- Pre condizioni: set pieno set1.
- Post condizioni: set set1 torna alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: eccezione IndexOutOfBounds().

6. constructor3NotNull()

- Riassunto: test del costruttore setAdapter(int initialCapacity, int capacityIncrement) per verificare che l'oggetto setAdapter creato non sia nullo.
- Design: viene creato il set e verificato che non sia nullo.
- Descrizione: viene creato un nuovo set2 con il costruttore setAdapter(int initialCapacity, int capacityIncrement) e un parametro di capacità e incremento capacità accettabili e viene verificato che non sia nullo.
- Pre condizioni: set pieno set1.
- Post condizioni: set set1 torna alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: set2 non nullo.

7. add()

- Riassunto: test del metodo add().
- Design: vengono aggiunti elementi di vario tipo ad un set vuoto.

- Descrizione: dopo aver svuotato set1, gli vengono ri aggiunti i medesimi elementi e ne viene controllata la posizione.
- Pre condizioni: set pieno set1.
- Post condizioni: set set1 torna alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: elementi aggiunti nelle posizioni previste.

8. add2()

- Riassunto: test del metodo add ().
- Design: vengono aggiunti elementi di vario tipo ad un set vuoto.
- Descrizione: dopo aver svuotato set1, gli vengono ri aggiunti i medesimi elementi e ne viene controllata la posizione.
- Pre condizioni: set pieno set1.
- Post condizioni: set set1 torna alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: elementi aggiunti nelle posizioni previste.

9. add3()

- Riassunto: test del metodo add ().
- Design: vengono aggiunti dei duplicati ad un set.
- Descrizione: vengono aggiunti a set1 degli elementi già presente nel set, e viene verificato che non vengano aggiunti e che la dimensione del set sia rimasta invariata.
- Pre condizioni: set pieno set1.
- Post condizioni: set set1 torna alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: i duplicati non sono stati aggiunti al set.

10. addAllNullPointer()

- Riassunto: test dell'eccezione NullPointer del metodo addAll(HCollection c).
- Design: viene aggiunto un elemento nullo ad un set con il metodo addAll(HCollection c).
- Descrizione: viene aggiunto un elemento nullo a set1 con il metodo addAll(HCollection c).
- Pre condizioni: set pieno set1.
- Post condizioni: set set1 torna alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: eccezione NullPointer.

11. addAll()

- Riassunto: test del metodo addAll(HCollection c).
- Design: viene aggiunta una HCollection ad un set e viene verificata la corretta aggiunta degli elementi. Successivamente viene eseguita la stesso operazione su un altro set e i due set vengono confrontati.
- Descrizione: vengono create due set vuote set2 e set3. Vengono poi aggiunti a set2 degli elementi e l'HCollection set1. Viene verificata la corretta posizione degli eventuali elementi aggiunti. Successivamente vengono aggiunti gli stessi elementi e la stesso HCollection a set3. Gli elementi di set2 e set3 vengono poi confrontati per verificare che il metodo funzioni allo stesso modo su set diversi.
- Pre condizioni: set pieno set1.
- Post condizioni: set set1 torna alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: i due set (set3 e set2) contengono gli stessi elementi nelle stesse posizioni.

12. clear()

- Riassunto: test del metodo clear().
- Design: viene svuotato il set set1 e viene verificato che la sua dimensione sia 0.
- Descrizione: viene svuotato il set set1 e viene verificato che la sua dimensione sia 0.
- Pre condizioni: set pieno set1.
- Post condizioni: set set1 torna alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: set set1 vuoto, quindi con dimensione uguale a 0.

13. contains()

- Riassunto: test del metodo contains().
- Design: viene verificata la presenza di elementi in un set tramite il metodo contains(). Viene inoltre verificata l'efficienza del metodo controllando la presenza di elementi non facenti parti del set.
- Descrizione: viene verificata la presenza degli elementi in set1 presenti dalle pre condizioni col metodo contains(). Viene inoltre verificata l'efficienza del metodo controllando la presenza di elementi non facenti parti del set.
- Pre condizioni: set pieno set1.
- Post condizioni: set set1 torna alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: il metodo contains() ritorna true con gli elementi effettivamente presenti e false con gli elementi non facenti parte del set.

14. containsAllNullPointer()

- Riassunto: test dell'eccezione NullPointer del metodo containsAll(HCollection c).
- Design: viene passato un null come parametro al metodo addAll(HCollection c).
- Descrizione: viene passato un null come parametro al metodo addAll(HCollection c).
- Pre condizioni: set pieno set1.
- Post condizioni: set set1 torna alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: eccezione NullPointer.

15. containsAll()

- Riassunto: test del metodo containsAll(HCollection c).
- Design: viene verificata la presenza degli elementi di una HCollection in un set con il metodo containsAll(HCollection c).
- Descrizione: viene creato il set vuoto set2, a cui vengono aggiunti elementi non presenti in set1, e successivamente tutti gli elementi di set1. Viene poi verificato tramite il metodo containsAll(HCollection c) la presenza di tutti gli elementi di set1 in set2 e viceversa.
- Pre condizioni: set pieno set1.
- Post condizioni: set set1 torna alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: il metodo riconosce che set2 contiene tutti gli elementi di set1 ma che set1 non contiene tutti gli elementi di set1.

16. equals()

- Riassunto: test del metodo equals().
- Design: viene testato il metodo equals() confrontando il set set1 con set uguali, diverse ed elementi di tipo diverso.
- Descrizione: viene creato il set vuoto set2, a cui vengono aggiunti tutti gli elementi di set1. Viene verificata l'uguaglianza tra set1 e set2 con il metodo equals(). Successivamente viene rimosso un elemento da set2 e viene verificato tramite equals() che le due set non sono più uguali. Poi viene creato un altro set vuoto set3, che viene riempito con tutti gli elementi di set1, ma in un ordine diverso. Viene verificato con equals() che set1 non è uguale a set3, né ad altri elementi di tipo diverso da SetAdapter.
- Pre condizioni: set pieno set1.
- Post condizioni: set set1 torna alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: il metodo restituisce true solo quando le set confrontate sono effettivamente identiche.

17. hashCodeTest()

- Riassunto: test del metodo hashCode().
- Design: vengono confrontati i valori dell'hashcode restituito dall'omonimo metodo con dei valori calcolati esternamente.
- Descrizione: viene creato il set vuoto set2. Vengono aggiunti degli elementi e man mano che vengono aggiunti, vengono confrontati i valori dell'hashcode restituito dall'omonimo metodo con dei valori calcolati esternamente.
- Pre condizioni: set pieno set1.
- Post condizioni: set set1 torna alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: i valori restituiti dal metodo coincidono con quelli previsti.

18. isEmpty()

- Riassunto: test del metodo isEmpty().
- Design: viene verificato con il metodo isEmpty() che il set sia vuoto dopo averlo svuotato e dopo aver aggiunto elementi di diverso tipo.
- Descrizione: viene creato il set vuoto set2. Vengono aggiunti e rimossi elementi e viene verificato che il set sia vuoto o meno dopo ogni inserimento/rimozione.
- Pre condizioni: set pieno set1.
- Post condizioni: set set1 torna alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: il metodo restituisce true se il set è effettivamente vuoto, e false altrimenti.

19. iteratorNotNull()

- Riassunto: test del metodo iterator() per verificare che l'oggetto HIterator restituito non sia nullo.
- Design: viene creato l'iteratore e verificato che non sia nullo.
- Descrizione: viene creato un iteratore di set1 con il metodo iterator() e viene verificato che non sia nullo.
- Pre condizioni: set pieno set1.
- Post condizioni: set set1 torna alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: iteratore non nullo.

20. iteratorHasNext()

- Riassunto: test del metodo hasNext() di HIterator.
- Design: viene creato l'iteratore e verificato che il metodo hasNext() funzioni correttamente.
- Descrizione: viene creato un iteratore di set1 con il metodo iterator() e viene verificato che hasNext() funzioni correttamente anche dopo aver aggiunto elementi o aver chiamato il metodo next().
- Pre condizioni: set pieno set1.
- Post condizioni: set set1 torna alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: hasNext() restituisce true solo se ci sono elementi successivi.

21. iteratorNextNoSuchElementException()

- Riassunto: test dell'eccezione NoSuchElementException() del metodo next() di HIterator.
- Design: viene creato l'iteratore su un set vuoto e verificato che chiamando next() venga lanciata l'eccezione NoSuchElementException().
- Descrizione: dopo aver svuotato set1, viene creato l'iteratore su di esso e chiamato next().
- Pre condizioni: set pieno set1.
- Post condizioni: set set1 torna alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: eccezione NoSuchElementException().

22. iteratorNextNoSuchElementException2()

- Riassunto: test dell'eccezione NoSuchElementException() del metodo next() di HIterator.
- Design: viene creato l'iteratore su un set pieno e verificato che chiamando next() alla fine del set, venga lanciata l'eccezione NoSuchElementException().
- Descrizione: viene creato l'iteratore su set1 e chiamato next() finché l'iteratore non arriva alla fine del set. Dopodiché viene chiamato nuovamente next.
- Pre condizioni: set pieno set1.
- Post condizioni: set set1 torna alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: eccezione NoSuchElementException().

23. iteratorNext()

- Riassunto: test del metodo next() di HIterator.
- Design: viene creato l'iteratore e verificato che il metodo next() funzioni correttamente.
- Descrizione: viene creato l'iteratore sul set piena set1 e verificato che il metodo next() restituisca gli elementi corretti nel giusto ordine.
- Pre condizioni: set pieno set1.
- Post condizioni: set set1 torna alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: next() restituisce gli elementi corretti nel giusto ordine.

24. iteratorRemoveIllegalStateException()

- Riassunto: test dell'eccezione IllegalState() del metodo remove() di HIterator.

- Design: viene creato l'iteratore su un set pieno e verificato che chiamando remove() subito dopo la creazione dell'iteratore, venga lanciata l'IllegalStateException().
- Descrizione: viene creato l'iteratore sul set piena set1 chiamato remove() subito dopo.
- Pre condizioni: set pieno set1.
- Post condizioni: set set1 torna alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: eccezione IllegalStateException().

25. iteratorRemoveIllegalStateException2()

- Riassunto: test dell'eccezione IllegalState() del metodo remove() di HIterator.
- Design: viene creato l'iteratore su un set pieno e verificato che chiamando remove() due volte di fila, venga lanciata l'IllegalStateException().
- Descrizione: viene creato l'iteratore sul set piena set1, aggiunto un elemento all'iteratore e chiamato remove() due volte consecutive.
- Pre condizioni: set pieno set1.
- Post condizioni: set set1 torna alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: eccezione IllegalStateException().

26. iteratorRemove()

- Riassunto: test del metodo remove() di HIterator.
- Design: viene creato l'iteratore e verificato che il metodo remove() funzioni correttamente.
- Descrizione: viene creato l'iteratore sul set piena set1, viene chiamato più volte il metodo next() e verificato che il metodo remove() rimuova gli elementi corretti. Viene inoltre verificata la dimensione di set1.
- Pre condizioni: set pieno set1.
- Post condizioni: set set1 torna alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: remove() rimuove gli elementi corretti.

27. removeObject(Object o).

- Riassunto: test del metodo remove(Object o).
- Design: viene testato il metodo remove(Object o) provando a rimuovere alcuni fra gli elementi di set1 tramite remove(Object o ex).
- Descrizione: viene testato il metodo remove(Object o) provando a rimuovere alcuni fra gli elementi di set1 tramite remove(Object o) e viene verificato che il metodo rimuova gli elementi corretti dagli indici corretti. Viene inoltre controllata la dimensione di set1.
- Pre condizioni: set pieno set1.
- Post condizioni: set set1 torna alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: il metodo rimuove gli elementi corretti dagli indici corretti.

28. removeAllNullPointer()

- Riassunto: test dell'eccezione NullPointer del metodo removeAll(HCollection c).
- Design: viene rimosso un elemento nullo ad un set con il metodo removeAll(HCollection c).
- Descrizione: viene rimosso un elemento nullo da set1 con il metodo removeAll(HCollection c).
- Pre condizioni: set pieno set1.
- Post condizioni: set set1 torna alla situazione iniziale (vedasi pre condizioni).

- Risultato atteso: eccezione NullPointer.

29. removeAllTrue()

- Riassunto: test del metodo removeAll(HCollection c).
- Design: vengono aggiunti elementi di diverso tipo ad un set. Vengono poi rimossi tutti gli elementi di quest'ultimo da un altro set e viene verificato che siano stati rimossi gli elementi corretti.
- Descrizione: dopo aver creato un set vuoto set2, gli vengono aggiunti elementi di diverso tipo. Vengono poi rimossi tutti gli elementi di quest'ultimo da set1 e viene verificato che siano stati rimossi gli elementi corretti.
- Pre condizioni: set pieno set1.
- Post condizioni: set set1 torna alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: removeAll(HCollection c) ha rimosso tutti e soli gli elementi di set2 da set1.

30. removeAllFalse()

- Riassunto: test del metodo removeAll(HCollection c).
- Design: vengono rimossi gli elementi di un set da un altro tramite removeAll(HCollection c). I set non hanno elementi in comune, perciò, ci si aspetta che non ne vengano tolti.
- Descrizione: viene creato un nuovo set vuoto set2 a cui vengono aggiunti alcuni elementi non contenuti in set1. A quest'ultimo viene rimosso il contenuto di set2 tramite removeAll(HCollection c). Viene poi verificato che il set set1 è rimasto immutato.
- Pre condizioni: set pieno set1.
- Post condizioni: set set1 torna alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: removeAll(HCollection c) non ha rimosso alcun elemento da set1.

31. retainAllNullPointer()

- Riassunto: test dell'eccezione NullPointer del metodo retainAll(HCollection c).
- Design: viene passato un elemento nullo al metodo retainAll(HCollection c).
- Descrizione: viene passato un elemento nullo al metodo retainAll(HCollection c) su set1.
- Pre condizioni: set pieno set1.
- Post condizioni: set set1 torna alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: eccezione NullPointer.

32. retainAllTrue()

- Riassunto: test del metodo retainAll(HCollection c).
- Design: vengono aggiunti elementi di diverso tipo ad un set. Vengono poi conservati solo gli elementi di quest'ultimo in un altro set e viene verificato che siano stati rimossi gli elementi corretti.
- Descrizione: dopo aver creato un set vuoto set2, gli vengono aggiunti elementi di diverso tipo. Vengono poi conservati solo gli elementi di quest'ultimo in set1 e viene verificato che siano stati rimossi gli elementi corretti.
- Pre condizioni: set pieno set1.
- Post condizioni: set set1 torna alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: retainAll(HCollection c) ha conservato solo gli elementi di set2 presenti in set1.

32. retainAllTrue2()

- Riassunto: test del metodo retainAll(HCollection c).
- Design: viene verificato che applicando retainAll(HCollection c) a un set passando come parametro un'HCollection senza elementi in comune, il set viene svuotato.
- Descrizione: dopo aver creato un set vuoto set2, gli vengono aggiunti elementi di diverso tipo non appartenenti a set1. Vengono poi conservati solo gli elementi di set2 in set1 e viene verificato che siano stati rimossi tutti gli elementi.
- Pre condizioni: set pieno set1.
- Post condizioni: set set1 torna alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: retainAll(HCollection c) ha conservato solo gli elementi di set2 presenti in set1, quindi ha rimosso tutti gli elementi da set1.

33. retainAllFalse()

- Riassunto: test del metodo retainAll(HCollection c).
- Design: vengono conservati gli elementi di un set presenti in un'altra tramite retainAll(HCollection c). I set hanno tutti gli elementi in comune perciò ci si aspetta che non ne vengano tolti.
- Descrizione: viene creato un nuovo set vuoto set2 a cui vengono aggiunti tutti gli elementi contenuti in set1. A quest'ultimo viene applicato retainAll(HCollection c) con set2 come parametro. Viene poi verificato che il set set1 è rimasto immutato.
- Pre condizioni: set pieno set1.
- Post condizioni: set set1 torna alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: retainAll(HCollection c) non ha rimosso alcun elemento da set1.

34. size()

- Riassunto: test del metodo size().
- Design: viene testato il metodo size().
- Descrizione: viene testato size() su set1 con degli elementi al suo interno e dopo effettuato un clear().
- Pre condizioni: set pieno set1.
- Post condizioni: set set1 torna alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: il restituisce la dimensione corretta.

35. toArray()

- Riassunto: test del metodo toArray().
- Design: viene creato un array con gli elementi di un set con toArray() e viene verificato che contenga gli stessi elementi del set nel medesimo ordine.
- Descrizione: viene creato un array con gli elementi di set1 con toArray() e viene verificato che contenga gli stessi elementi di set1 nel medesimo ordine.
- Pre condizioni: set pieno set1.
- Post condizioni: set set1 torna alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: l'array ottenuto contiene gli stessi elementi di set1 nel medesimo ordine.

36. toArrayNullPointer()

- Riassunto: test dell'eccezione NullPointer del metodo toArray(Object[] a).
- Design: viene passato un null come parametro al metodo toArray(Object[] a).
- Descrizione: viene passato un null come parametro al metodo toArray(Object[] a).

- Pre condizioni: set pieno set1.
- Post condizioni: set set1 torna alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: eccezione NullPointer.

37. toArrayObjInt()

- Riassunto: test del metodo toArray(Object[] a).
- Design: viene passato un array di Int al metodo toArray(Object[] a) che agisce su un set di interi. Viene verificato che contenga gli stessi elementi del set nel medesimo ordine e che l'array sia compatibile col tipo passato.
- Descrizione: viene svuotato set1 e viene riempita di interi. Viene inoltre creato un array arrInt di interi di dimensione maggiore del set, che viene riempito di interi. Viene quindi verificato che l'array restituito dal metodo toArray(Object[] a) su set1 con arrInt come parametro, sia dello stesso tipo di quello passato (arrInt) e che esso contenga gli stessi elementi di list1 nelle medesime posizioni.
- Pre condizioni: set pieno set1.
- Post condizioni: set set1 torna alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: l'array ottenuto contiene gli stessi elementi di set1 nelle medesime posizioni ed è dello stesso tipo di quello passato.

38. toArrayObj()

- Riassunto: test del metodo toArray(Object[] a).
- Design: vengono confrontati un array creato con il metodo toArray() e uno creato con il metodo toArray(Object[] a).
- Descrizione: viene inoltre creato un array arrInt di interi di dimensione minore del set, che viene riempito di interi. Viene quindi verificato che l'array restituito dal metodo toArray(Object[] a) su set1 con arrInt come parametro, restituisca un array di tipo Object[] contenente gli stessi elementi di set1.
- Pre condizioni: set pieno set1.
- Post condizioni: set set1 torna alla situazione iniziale (vedasi pre condizioni).
- Post condizioni: set set1 torna alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: l'array ottenuto contiene gli stessi elementi di set1 nelle medesime posizioni.

39. getIndexOutOfBounds()

- Riassunto: test dell'eccezione IndexOutOfBounds() del metodo get(int index).
- Design: viene utilizzato il metodo get(int index) con un indice non accettabile.
- Descrizione: si prova a prendere un elemento da set1 in un indice non accettabile casuale compreso tra -100 e -1.
- Pre condizioni: set pieno set1.
- Post condizioni: set set1 torna alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: eccezione IndexOutOfBounds().

40. getIndexOutOfBounds2()

- Riassunto: test dell'eccezione IndexOutOfBounds() del metodo get(int index).
- Design: viene utilizzato il metodo get(int index) con un indice non accettabile.
- Descrizione: si prova a prendere un elemento da set1 in un indice non accettabile casuale compreso tra 8 e 100 (valori maggiori della dimensione del set).

- Pre condizioni: set pieno set1.
- Post condizioni: set set1 torna alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: eccezione IndexOutOfBounds().

41. get(int index)

- Riassunto: test del metodo get(int index)
- Design: viene testato il metodo get(int index) provando a prendere tutti gli elementi di set1 tramite get(int index).
- Descrizione: viene testato il metodo get(int index) provando a prendere tutti gli elementi di set1 tramite get(int index) e viene verificato che il metodo restituisca gli elementi corretti agli indici corretti.
- Pre condizioni: set pieno set1.
- Post condizioni: set set1 torna alla situazione iniziale (vedasi pre condizioni).
- Risultato atteso: il metodo restituisce gli elementi corretti agli indici corretti.