Luca Paulon Matricola: 1187028

Documentazione test

Test dell'adapter: ListObjectAdapter

La suite TestListObjectAdapter contiene i test relativi all'adapter dell'interfaccia target List. Si crea un ListObjectAdapter come istanza di classe che inizializza internamente un vector di dimensione 10. Prima di ogni test si porta l'adapter ad uno stato consistente per poi eseguire i test.

Variabili di ambiente:
JUNIT_HOME=/usr/share/java
CLASSPATH=$CLASSPATH:$JUNIT_HOME/junit4.jar

1. Test addIndex(): verifica che gli elementi vengano inseriti nella lista in corrispondenza degli indici specificati.
   Pre-condition: l'adapter è vuoto.
   Test cases:
   - Si verifica che inserendo 10 valori denominati "stringai", dove i è un numero incrementale appartenente all'intervallo [0,9], questi vengano inseriti partendo dall'indice 0. Al termine dell'inserimento si testa che gli elementi siano nella posizione assegnata.
     Post-condition: l'adapter contiene 10 elementi.
     Execution record: test superato se gli elementi sono nelle posizioni specificate.
   - Si verifica che tentando l'inserimento di un elemento all'indice 100, quindi non appartenente agli indici validi della lista, venga lanciata l'eccezione IndexOutOfBoundsException.
     Post-condition: l'adapter è vuoto.
     Execution record: test superato se viene lanciata l'eccezione IndexOutOfBoundsException.
   - Si verifica che tentando l'inserimento di un elemento di riferimento null venga lanciata l'eccezione NullPointerException.
     Post-condition: l'adapter è vuoto.
     Execution record: test superato se viene lanciata l'eccezione NullPointerException.

2. Test add() e size(): verifica che l'inserimento degli elementi all'interno della lista vada a buon fine e che la dimensione della lista sia corretta.
   Pre-condition: l'adapter viene inizializzato con 3 elementi denominati "stringa0", "stringa1", "stringa2".
   Test cases:
   - Si verifica che a seguito dell'inizializzazione la dimensione della lista sia 3.
     Post-condition: la lista rimane invariata.
     Execution record: test superato se la dimensione della lista è quella attesa ovvero 3.
   - Si verifica che ad ogni inserimento il valore ritornato dal metodo add() sia true.
     Post-condition: la lista rimane invariata.
     Execution record: test superato se ad ogni inserimento il valore ritornato dal metodo è true.
   - Si verifica che tentando l'inserimento di un elemento di riferimento null venga lanciata l'eccezione NullPointerException.
     Post-condition: la lista rimane invariata.
     Execution record: test superato se viene lanciata l'eccezione NullPointerException.

3. Test clear(): verifica che a seguito della cancellazione di tutti gli elementi della lista la dimensione di questa sia 0.

Pre-condition: l'adapter viene inizializzato con 10 elementi denominati "stringa0", "stringa1"…"stringa9".
Test cases:

- Si invoca il metodo clear() e si testa che la dimensione della lista sia nulla.
  Post-condition: la lista non contiene elementi.
  Execution record: test superato se tutti gli elementi sono stati rimossi.

4. Test indexOf(): verifica che l'elemento specificato sia nella posizione attesa.
   Pre-condition: l'adapter viene inizializzato con 10 elementi denominati "stringa0", "stringa1"…"stringa9".
   Test cases:
   - Si verifica che l'elemento "stringa2" corrisponda alla posizione 2 della lista.
     Post-condirion: la lista rimane invariata.
     Execution record: test superato se l'elemento "stringa2" corrisponde alla posizione attesa.
   - Si verifica che l'elemento "stringa100" corrisponda alla posizione -1, poiché non è presente nella lista.
     Post-condirion: la lista rimane invariata.
     Execution record: test superato se l'elemento "stringa100" corrisponde alla posizione attesa.
   - Si verifica che tentando d'ottenere la posizione corrispondente ad un elemento di riferimento null venga lanciata l'eccezione NullPointerException.
     Post-condition: la lista rimane invariata.
     Execution record: test superato se viene lanciata l'eccezione NullPointerException.

5. Test isEmpty(): verifica che a seguito della rimozione di tutti gli elementi della lista questa sia vuota.
   Pre-condition: l'adapter viene inizializzato con 10 elementi denominati "stringa0", "stringa1"…"stringa9".
   Test cases:
   - Si verifica che a seguito dell'invocazione del metodo clear() il valore ritornato dal metodo isEmpty() sia true.
     Post condition: la lista è vuota.
     Execution record: test superato se la lista risulta vuota e quindi il valore di ritorno è true.
   - Si verifica che dopo l'inizializzazione, senza l'invocazione del metodo clear(), il valore ritornato dal metodo isEmpty() sia false e la dimenzione della lista sia quindi quella iniziale.
     Post-condition: la lista rimane invariata.
     Execution record: test superato se il valore di ritorno del metodo è false e se la dimensione della lista rimane invariata.

6. Test contains(): verifica che l'elemento specificato sia contenuto nella lista.
   Pre-condition: l'adapter viene inizializzato con 10 elementi denominati "stringa0", "stringa1"…"stringa9".
   Test cases:
   - Si verifica che l'elemento "stringa7" sia contenuto nella lista, quindi il valore di ritorno del metodo deve essere true.
     Post-condition: la lista rimane invariata.
     Execution record: test superato se l'elemento è contenuto nella lista ed il valore ritornato è true.
     Si verifica che l'elemento "stringa100" non sia contenuto nella lista e che quindi il valore ritornato dal metodo sia false.

Execution record: test superato se il valore non è presente ed il valore ritornato è false.

- Si verifica che tentando la ricerca di un elemento di riferimento null venga lanciata l'eccezione NullPointerException.
  Post-condition: la lista rimane invariata.
  Execution record: test superato se viene lanciata l'eccezione NullPointerException.

7. Test lastIndexOf(): verifica che l'indice corrispondente all'ultima occorrenza dell'elemento specificato sia quella attesa.
   Pre-condition: l'adapter viene inizializzato con 5 elementi denominati "stringa0", "stringa1", "stringa2", "striga1", "stringa2".
   Test cases:
   - Si verifica che l'utimo indice corrispondente al valore "stringa2" sia 4.
     Post-condition: la lista rimane invariata.
     Execution record: test superato se l'indice ritornato dal metodo corrisponde a quello atteso.
   - Si verifica che l'utimo indice corrispondente ell'elemento "stringa100", che non appartiene alla lista, sia -1.
     Post-condition: la lista rimane invariata.
     Execution record: test superato se la posizione ritornata è -1.
   - Si verifica che tentando d'ottenere l'ultimo indice di un elemento di riferimento null venga lanciata l'eccezione NullPointerException.
     Post-condition: la lista rimane invariata.
     Execution record: test superato se viene lanciata l'eccezione NullPointerException.

8. Test remove() con indice: verifica che la rimozione di un elemento della lista, alla posizione specificata, vada a buon fine.
   Pre-condition: l'adapter viene inizializzato con 10 elementi denominati "stringa0", "stringa1"…"stringa9".
   Test cases:
   - Si verifica che la rimozione dell'elemento in posizione 5 ritorni il valore che si trova in quella posizione, ovvero "stringa5". Inoltre la dimensione della lista deve diminuire di una unità.
     Post-condition: l'elemento in posizione 5 viene rimosso. La lista viene compattata.
     Execution record: test superato se l'elemento in posizione 5 viene rimosso.
   - Si verifica che tentando la rimozione di un elemento in posizione 5, ma all'interno di una lista vuota, venga lanciata l'eccezione IndexOutOfBoundsException.
     Post-condition: la lista rimane invariata.
     Execution record: test superato se viene lanciata l'eccezione IndexOutOfBoundsException.
   - Si verifica che tentando la rimozione di un elemento in posizione -1, cioè fuori dagli indici validi della lista, venga lanciata l'eccezione IndexOutOfBoundsException.
     Post-condition: la lista rimane invariata.
     Execution record: test superato se viene lanciata l'eccezione IndexOutOfBoundsException.

9. Test remove() di un oggetto: verifica che la rimozione di un elemento della lista vada a buon fine.
   Pre-condition: l'adapter viene inizializzato con 10 elementi denominati "stringa0", "stringa1"…"stringa9".
   Test cases:
   - Si verifica che la rimozione dell'elemento "stringa6" della lista vada a buon fine, quindi che il metodo ritorni il valore true. Inoltre la dimensione della lista deve diminuire di una unità.

Post-condition: l'elemento specificato come parametro viene rimosso. La lista viene compattata.
Execution record: test superato se l'elemento "stringa6" viene rimosso.

- Si verifica che tentando la rimozione di un elemento non appartenete alla lista, "stringa20", il valore di ritorno del metodo sia false.
  Post-condition: la lista rimane invariata.
  Execution record: test superato se viene lanciata l'eccezione IndexOutOfBoundsException.
- Si verifica che tentando la rimozione di un elemento di riferimento null venga lanciata l'eccezione NullPointerException.
  Post-condition: la lista rimane invariata.
  Execution record: test superato se viene lanciata l'eccezione NullPointerException.

10. Test set() : verifica che modificando un elemento in una posizione specificata le modifiche siano andate a buon fine.
    Pre-condition: l'adapter viene inizializzato con 10 elementi denominati "stringa0", "stringa1"…"stringa9".
    Test cases:
    - Si verifica che dopo aver modificato il valore dell'elemento in posizione 8, che precedentemente era "stringa8" e che ora è "nuova stringa", il valore ritornato dal metodo corrisponda al vecchio valore. Si verifica poi che il nuovo valore in posizione 8 sia quello specificato.
      Post-condition: l'elemento in posizione 8 viene modificato.
      Execution record: test superato se l'elemento in posizione 8 risulta modificato.
    - Si verifica che tentando la modifica di un elemento in posizione 100, quindi non appartenente alla lista, venga lanciata l'eccezione IndexOutOfBoundsException.
      Post-condition: la lista rimane invariata.
      Execution record: test superato se viene lanciata l'eccezione IndexOutOfBoundsException.
    - Si verifica che tentando di modificare un elemento in una posizione valida della lista, ma con un valore null, venga lanciata l'eccezione IndexOutOfBoundsException.
      Post-condition: la lista rimane invariata.
      Execution record: test superato se viene lanciata l'eccezione IndexOutOfBoundsException.

11. Test toArray() : verifica che l'array ritornato dall'adapter contenga gli elementi nello stesso ordine in cui sono memorizzati nella lista.
    Pre-condition: l'adapter viene inizializzato con 10 elementi denominati "stringa0", "stringa1"…"stringa9".
    Test cases:
    - Si verifica che gli elementi dell'array siano disposti nello stesso ordine posizionale con cui sono memorizzati nella lista.
      Post-condition: la lista rimane invariata.
      Execution record: test superato se le posizioni degli elementi dell'array corrispondono alle posizioni degli elementi della lista.
    - Si cancella il contenuto della lista. Si verifica che l'array corrispondente ad una lista vuota sia vuoto.
      Post-condition: la lista risulta priva di elementi.
      Execution record: test superato se l'array ritornato risulta vuoto.

12. Test iterator() : verifica che l'iteratore ritornato dall'adapter ritorni gli elementi nello stesso ordine con sui sono stati memorizzati nella lista. Si testano i metodi della sua interfaccia quindi: next() e hashNext().
    Pre-condition: l'adapter viene inizializzato con 10 elementi denominati "stringa0", "stringa1"…"stringa9".
    Test cases:
    - si verifica che gli elementi che vengono ritornati dall'iteratore siano ritornati nello stesso ordine posizionale corrispondente a quello con cui sono stati memorizzati nella lista.
      Post-condition: la lista rimane invariata.
      Execution record: test superato se le posizioni degli elementi ritornati dall'iteratore corrispondono alle posizioni degli elementi della lista.

13. Test sublist() : verifica i range degli indici validi necessari alla creazione di una sottolista.
    Pre-condition: l'adapter viene inizializzato con 10 elementi denominati "stringa0", "stringa1"…"stringa9".
    Test-cases:
    - Si verifica che definendo un indice di partenza negativo per la sottolista venga lanciata l'eccezione IndexOutOfBoundsException.
      Post-condition: la lista rimane invariata.
      Execution record: test superato se viene lanciata l'eccezione IndexOutOfBoundsException.
    - Si verifica che definendo un indice di arrivo troppo grande per la sottolista venga lanciata l'eccezione IndexOutOfBoundsException.
      Post-condition: la lista rimane invariata.
      Execution record: test superato se viene lanciata l'eccezione IndexOutOfBoundsException.
    - Si verifica che definendo un indice di partenza maggiore dell'indice di arrivo per la sottolista venga lanciata l'eccezione IndexOutOfBoundsException.
      Post-condition: la lista rimane invariata.
      Execution record: test superato se viene lanciata l'eccezione IndexOutOfBoundsException.

Per la verifica del funzionamento di Sublist si rimanda alla suite di test corrispondente.

14. Test ListIterator() : verifica il corretto funzionamento del ListIterator della lista. Si testano tutti i metodi dell'interfaccia: hasNext(), next(), hashPrevious(), previous(), nextIndex(), previousIndex(), add(), remove() e set().
    Pre-condition: l'adapter viene inizializzato con 10 elementi denominati "stringa0", "stringa1"…"stringa9".
    Test cases:
    - Si verifica che facendo scorrere l'iteratore da sinistra a destra, poi da destra a sinistra e nuovamente da sinistra a destra, gli elementi che vengono ritornati nell'ordine posizionale dall'iteratore corrispondano agli elementi della lista nello stesso ordine posizionale.
      Post-condition: la lista rimane invariata.
      Execution record: test superato se gli elementi ritornali dal ListIterator corrispondono agli elementi della lista.
    - Si fa scorrere il ListIterator all'estremo destro della lista. S'inserisce un elemento in coda, "stringa10", si verifica che la dimensione della lista sia aumentata di una unità
      Post-condition: viene aggiunto un elemento alla lista.
      Execution record: test superato se la dimensione risulta aumentata di una unità.

- Dopo il precedente inserimento l'iteratore punta alla coda della lista. Si rimuove l'elemento in coda, "stringa10", si verifica che la dimensione della lista sia diminuita di una unità
  Post-condition: viene rimosso un elemento dalla lista.
  Execution record: test superato la dimensione risulta diminuita di una unità.
- Si fa scorrere il ListIterator all'estremo sinistro della lista. S'inserisce un elemento in testa, "newStringa0", si verifica che la dimensione della lista sia aumentata di una unità
  Post-condition: viene aggiunto un elemento alla lista.
  Execution record: test superato se la dimensione risulta aumentata di una unità.
- Dopo il precedente inserimento l'iteratore punta alla testa della lista. Si rimuove l'elemento in testa, "newStringa0", si verifica che la dimensione della lista sia diminuita di una unità
  Post-condition: viene rimosso un elemento alla lista.
  Execution record: test superato la dimensione risulta diminuita di una unità.
- Si fa scorrere il ListIterator fino alla posizione 5 della lista e si verifica che l'indice precedente sia 4 e che il successivo sia 6. Si utilizzano i metodi nextIndex() e previousIndex(). Infine si modifica l'elemento in posizione 5 con un nuovo valore, usado set() del ListIterator, e si verifica l'avvenuta modifica.
  Post-condition: viene modificato l'elemento in posizione 5 della lista.
  Execution record: test superato se gli indici ritornati sono quelli attesi e se l'elemento è stato modificato correttamente.
- Si testa la creazione di un ListIterator il cui puntatore corrente è settato alla posizione iniziale specificata come parametro. Si verifica che gli elementi ritornati da previous() e da next() siano quelli previsti
  Post-condition: la lista rimane invariata.
  Execution record: test superato se i valori ritornati prima e dopo la posizione iniziale sono quelli attesi.
- Si verifica che creando un ListIterator il cui indice di partenza è inferiore a 0 venga lanciata l'eccezione IndexOutOfBoundsException.
  Post-condition: la lista rimane invariata.
  Execution record: test superato se viene lanciata l'eccezione IndexOutOfBoundsException.
- Si verifica che creando un ListIterator il cui indice di partenza è maggiore o uguale alla dimensione della lista venga lanciata l'eccezione IndexOutOfBoundsException.
  Post-condition: la lista rimane invariata.
  Execution record: test superato se viene lanciata l'eccezione IndexOutOfBoundsException.

15. Test equals() : verifica che due liste sono uguali se e solo se contengono gli stessi elementi ed hanno la stessa dimensione.
    Pre-condition: l'adapter viene inizializzato con 10 elementi denominati "stringa0", "stringa1"…"stringa9".
    Test cases:
    - Si verifica che dopo aver creato una lista contenete gli stessi elementi di quella iniziale, a seguito dell'invocazione di equals, il valore ritornato sia true. Si testa poi che le due liste abbiano lo stesso hashcode, che è la somma degli hashcode dei vari elementi presenti nella lista.
      Post-condition: la lista rimane invariata.
      Execution record: test superato le due liste sono uguali.
    - Si verifica che dopo aver creato una lista contenete elementi differenti da quella iniziale, a seguito dell'invocazione di equals, il valore ritornato sia false. Si testa poi che le due liste abbiano hashcode differenti. Il test viene eseguito anche su un sott'insieme degli elementi

della lista iniziale, ma il risultato è comunque false poiché le liste non hanno la stessa lunghezza.
Execution record: test superato se le due liste sono diverse.

- Si verifica che tentando di confrontare la lista iniziale con un qualsiasi altro tipo di oggetto venga lanciata l'eccezione ClassCastException.
  Post-condition: la lista rimane invariata.
  Execution record: test superato se viene lanciata l'eccezione ClassCastException.
- Si verifica che tentando di confrontare la lista iniziale con un oggetto di riferimento null venga lanciata l'eccezione NullPointerException.
  Post-condition: la lista rimane invariata.
  Execution record: test superato se viene lanciata l'eccezione NullPointerException.

16. Test removeAll() : verifica che a seguito dell'invocazione di removeAll() la dimensione della lista sia diminuita di almeno un'unità.
    Pre-condition: l'adapter viene inizializzato con 10 numeri interi 0, 1…9.
    Test cases:
    - Si verifica che passata una collezione contenente i numeri da 5 a 9, questi vengano rimossi dalla lista poiché contenuti. Si testa che la dimensione della lista sia dimezzata e che siano rimasti solo gli elementi da 0 a 4 e che il metodo ritorni true.
      Post condition: la lista contiene i soli elementi da 0 a 4.
      Execution record: test superato se la lista contiene i soli elementi da 0 a 4 e se la dimensione della lista è dimezzata.
    - Si verifica che passata una collezione contenente i numeri pari da 10 a 18 questi non vengano rimossi dalla lista poiché non contenuti.
      Post-condition: la lista rimane invariata.
      Execution record: test superato se la lista ha la stessa dimensione iniziale.
    - Si verifica che tentando di passare al metodo una collezione con riferimento null venga lanciata l'eccezione NullPointerException.
      Post-condition: la lista rimane invariata.
      Execution record: test superato se viene lanciata l'eccezione NullPointerException.
    - Pre condition: si inizializza l'adapter con i valori 0, 0, 1, 1, 2, 2.
      Si verifica che passata una collezione contenente i numeri da 0 a 9 vengano rimosse dalla lista le prime occorrenze dei valori 0, 1, 2. Si verifica che la dimensione risultante sia 3.
      Post-condition: la lista contiene i soli elementi 0, 1, 2.
      Execution record: test superato se la lista contiene i soli elementi 0, 1, 2.

17. Test containsAll() : verifica che la lista contenga tutti gli elementi di una collezione passata come parametro.
    Pre-condition: l'adapter viene inizializzato con 10 numeri interi da 0 a 9.
    Test cases:
    - Si verifica che passata una collezione contenente i numeri da 5 a 9, questi siano contenuti nella lista. Si verifica che il valore di ritorno del metodo sia true.
      Post-condition: la lista rimane invariata.
      Execution record: test superato se la lista contiene tutti gli elementi della collezione passata.
    - Si verifica che passata una collezione contenente i numeri pari da 10 a 18 questi non siano contenuti. Inoltre si verifica che il valore di ritorno del metodo sia false.
      Post-condition: la lista rimane invariata.
      Execution record: test superato se la lista ha la stessa dimensione iniziale.

- Si verifica che tentando di passare al metodo una collezione con riferimento null venga lanciata l'eccezione NullPointerException.
  Post-condition: la lista rimane invariata.
  Execution record: test superato se viene lanciata l'eccezione NullPointerException.

18. Test retainAll() : verifica che la lista mantenga al suo interno tutti gli elementi di una collezione passata come parametro.
    Pre-condition: l'adapter viene inizializzato con 10 elementi denominati "stringa0", "stringa1"…"stringa9".
    Test cases:
    - Si verifica che passata una collezione contenente i numeri da 5 a 9, questi siano gli unici valori mantenuti nella lista. Inoltre si verifica che il valore di ritorno del metodo sia true e che gli elementi mantenuti siano soltanto quelli presenti anche nella collezione passata.
      Post-condition: la lista mantiene al suo interno solo gli elementi contenuti anche nella collezione specificata.
      Execution record: test superato se la lista contiene tutti e soli gli elementi della collezione passata come parametro. Il metodo ritorna true se e solo se la dimensione della lista risulta cambiata.
    - Si verifica che, passata una collezione contenente gli stessi interi contenuti anche nella collezione, il valore di ritorno del metodo sia false. Poiché la dimensione della lista non cambia.
      Post-condition: la lista rimane invariata.
      Execution record: test superato se il valore di ritorno è false e la lista rimane invariata.
    - Si verifica che tentando di passare al metodo una collezione con riferimento null venga lanciata l'eccezione NullPointerException.
      Post-condition: la lista rimane invariata.
      Execution record: test superato se viene lanciata l'eccezione NullPointerException.

19. Test addAll() : verifica che tutti i valori di una nuova collezione siano inseriti nella lista iniziale.
    Pre-condition: l'adapter viene inizializzato con 10 elementi denominati "stringa0", "stringa1"…"stringa9".
    Test cases:
    - Si verifica che passata una collezione contenente i numeri da 10 a 19, questi siano aggiunti nella lista. Inoltre si verifica che il valore di ritorno del metodo sia true e che gli elementi inseriti siano soltanto quelli presenti nella collezione passata.
      Post-condition: oltre alle 10 stringhe inizialmente presenti si aggiungono 10 interi da 10 a 19.
      Execution record: test superato se la lista contiene tutti gli elementi che conteneva prima, con eventuali duplicazioni, più tutti gli elementi che sono contenuti nella collezione passata.
    - Si verifica che tentando di passare al metodo una collezione con riferimento null venga lanciata l'eccezione NullPointerException.
      Post-condition: la lista rimane invariata.
      Execution record: test superato se viene lanciata l'eccezione NullPointerException.

20. Test addAll() con indice : verifica che tutti i valori di una nuova collezione siano inseriti nella lista iniziale a partire dall'indice specificato.
    Pre-condition: l'adapter viene inizializzato con 10 elementi denominati "stringa0", "stringa1"…"stringa9".

Test cases:

- Si verifica che passata una collezione contenente i numeri da 10 a 14 ed un indice pari a 5, questi siano aggiunti a partire dalla metà della lista. Si verifica, poi, che il valore di ritorno del metodo sia true e che gli elementi inseriti a partire dalla posizione 5, per le successive 5 posizioni, siano soltanto quelli presenti nella collezione passata.
  Post-condition: oltre alle 10 stringhe inizialmente presenti si aggiungono altri 5 interi da 10 a 14 nella metà della lista.
  Execution record: test superato se la lista contiene tutti gli elementi che conteneva prima, con eventuali duplicazioni, più tutti gli elementi che sono contenuti nella collezione passata.
- Si verifica che tentando di passare al metodo una collezione con riferimento null, anche se l'indice è valido, venga lanciata l'eccezione NullPointerException.
  Post-condition: la lista rimane invariata.
  Execution record: test superato se viene lanciata l'eccezione NullPointerException.
- Si verifica che tentando di passare al metodo una collezione valida, ma un indice di partenza troppo grande, venga lanciata l'eccezione IndexOutOfBoundsException.
  Post-condition: la lista rimane invariata.
  Execution record: test superato se viene lanciata l'eccezione IndexOutOfBoundsException.

21. Test sort() : verifica gli elementi della lista siano stati ordinati secondo un ordinamento posizionale crescente.
    Pre-condition: l'adapter viene inizializzato con 10 numeri interi da 0 a 9, non ordinati.
    Test cases:
    - Si crea una seconda lista contenente gli stessi elementi di quella iniziale, ma ordinati. S'invoca il metodo sort() sulla lista iniziale. Si verifica, quindi, che la lista iniziale risulti ordinata allo stesso modo in cui è ordinata la seconda lista.
      Post-condition: la lista iniziale risulta ordinata.
      Execution record: test superato se la lista iniziale è ordinata come la lista secondaria usata come criterio di confronto.

Test dell'adapter: Sublist
La suite TestSublist contiene i test relativi all'object adapter implementato per adattare l'interfaccia List alle funzionalità di una sottolista. Si crea un ListObjectAdapter come istanza di classe che contiene i numeri interi da 0 a 19 a cui si richiede poi una sottolista.

Variabili di ambiente:
JUNIT_HOME=/usr/share/java
CLASSPATH=$CLASSPATH:$JUNIT_HOME/junit4.jar

Nei seguenti test non vengono testate molte delle combinazioni testate sulla lista, poiché sublist utilizza al suo interno i metodi della lista che le fa da backing storage. Di conseguenza si preferisce testare le ripercussioni sulla lista di modifiche fatte sulla sottolista.

Pre-condition: per ognuno dei test successivi si parte da una lista contenente i numeri interi da 0 a 19 ed una sottolista, che si appoggia sulla lista come storage, i cui indici di partenza e arrivo sono 5 (compreso) e 15 (non compreso), sempre tenendo come riferimento la lista di classe.

1. Test size(): verifica che la dimensione della sottolista sia quella attesa.
   Test cases:
   - Si verifica che dopo l'inizializzazione la sublist abbia dimensione 10.
     Post-condition: la sublist rimane invariata.
     Execution record: test superato se la dimensione della lista è quella attesa.

2. Test clear() e isEmpty(): verifica che la sottolista sia vuota.
   Test cases:
   - Si testa che subito dopo l'inizializzazione la sottolista non sia vuota, cioè il metodo isEmpty() ritorna il valore false.
     Post condition: la sottolista rimane invariata.
     Execution record: test superato se la sottolista non è vuota.
   - Si verifica che dopo l'inizializzazione, invocando il metodo clear(), la sottolista sia vuota ma la lista di riferimento contiene ancora 10 elementi. Il valore di ritorno di isEmpty() è true.
     Post-condition: la sublist risulta vuota, la lista di riferimento contiene 10 elementi.
     Execution record: test superato se la sottolista è vuota e la lista di riferimento contiene ancora degli elementi.

3. Test add(): verifica che aggiungendo tre elementi alla sottolista questi siano aggiunti anche alla lista di riferimento.
   Test cases:
   - Si testa che subito dopo l'inizializzazione, aggiungendo 3 elementi, questi siano aggiunti sia nella sottolista e sia nella lista di riferimento. Si verifica quindi che le dimensioni di lista e sottolista siano aumentate di tre unità.
     Post-condition: la sottolista e la lista aumentano di dimensione.
     Execution record: test superato se sia la lista che la sottolista aumentano la loro dimensione di tre unità.

4. Test addAll(): verifica che aggiungendo alla sottolista tutti gli elementi che sono contenuti nella collezione specificata questi siano aggiunti anche alla lista di riferimento.
   Test cases:

- Si testa che passando una collezione contenente 4 interi: 20, 30, 40, 50 questi siano aggiunti sia nella sottolista e sia nella lista di riferimento. Si verifica quindi che le dimensioni di lista e sottolista siano aumentate di quattro unità.
  Post-condition: la sottolista e la lista aumentano di dimensione.
  Execution record: test superato se sia la lista che la sottolista aumentano la loro dimensione di quattro unità.

5. Test get(): verifica che l'elemento corrispondente all'indice specificato sia quello atteso.
   Test cases:
   - Si testa che l'elemento in posizione 2 della sottolista sia 7, quello in posizione 9 sia 14 e quello in posizione 5 sia 10.
     Post-condition: la lista rimane invariata.
     Execution record: test superato se nelle posizioni specificate risiedono gli elementi attesi.
   - Si verifica che tentando d'accedere ad un elemento della sottolista con un indice troppo grande, che non appartiene al range di indici validi, venga lanciata l'eccezione IndexOutOfBoundsException.
     Post-condition: sia la sottolista che la lista rimangono invariate.
     Execution record: test superato se viene lanciata l'eccezione IndexOutOfBoundsException.
   - Si verifica che tentando d'accedere ad un elemento della sottolista con un indice troppo piccolo, che non appartiene al range di indici validi, venga lanciata l'eccezione IndexOutOfBoundsException.
     Post-condition: sia la sottolista che la lista rimangono invariate.
     Execution record: test superato se viene lanciata l'eccezione IndexOutOfBoundsException.

6. Test toArray(): si crea un classe IntHolder per verificare che modificando un elemento dell'array, che è contenuto anche nella sottolista, le modifiche vengano viste sia dalla lista che dalla sottolista. Quindi si ha un array di shallow copies, in cui si sono memorizzati i riferimenti egli oggetti e non i loro valori.
   Pre condition: creazione di una lista di IntHolder inizializzati con i numeri interi da 0 a 19, creazione di una sottolista di IntHolder contenente una vista degli elementi tra gli indici 5 e 15 (non compreso).
   Test cases:
   - Si testa che modificando l'IntHolder in posizione 0 dell'array, cioè quello in posizione 0 della sottolista e quindi quello in posizione 5 della lista, moltiplicando l'intero per 100, si verifica che sia la lista che la sottolista contengano il numero 500 in corrispondenza della posizione modificata.
     Post-condition: la lista e la sottolista hanno l'elemento rispettivamente nelle posizioni: 5 e 0 che risulta modificato.
     Execution record: test superato se sia la lista che la sottolista vedono la modifica dell'elemento.
   - Si testa che modifiche strutturali alla sottolista e quindi alla lista non abbiano alcun effetto sull'array. Si rimuove l'elemento in posizione 5 della lista, quindi quello in posizione 0 della sottolista, e si verifica che l'elemento in posizione 0 dell'array sia quello che invece nelle due collezioni è stato rimosso.
     Post-condition: sia la sottolista che la lista diminuiscono la loro dimensione di una unità, l'array rimane invariato.
     Execution record: test superato se solo lista e la sottolista subiscono modifiche strutturali e queste non si ripercuotono sull'array.

- Si testa che modifiche strutturali all'array non abbiano alcun effetto sulla lista e sulla sottolista. Si sostituisce l'elemento in posizione 0 dell'array, quindi quello in posizione 0 della sottolista ed il rispettivo in posizione 5 della lista, con un nuovo elemento. Si verifica che il nuovo IntHolder è presente solo nell'array e non anche nella lista oppure anche nella sottolista.
  Post-condition: sia la sottolista che la lista rimangono invariate, l'elemento in posizione 0 dell'array cambia.
  Execution record: test superato se solo l'array viene modificato e quindi lista e sottolista non subiscono alcuna modifica.

7. Test indexOf(): verifica che l'elemento corrispondente all'indice specificato sia quello atteso.
   Test cases:
   - Si testa che l'elemento 5 sia nella posizione 0 della sottolista, l'elemento 10 sia in posizione 5 della sottolista e l'elemento 14 sia in posizione 9 della sottolista.
     Post-condition: la sottolista rimane invariata.
     Execution record: test superato se gli elementi risiedono nelle posizioni specificate.
   - Si testa che: il valore 4 della sottolista abbia indice -1, poiché non appartiene alla sottolista ma solo alla lista, lo stesso vale per il valore 15 che appartiene alla lista ma non alla sottolista.
     Post-condition: la lista rimane invariata.
     Execution record: test superato se i due elementi specificati non appartengono alla sottolista ed il metodo indexOf() ritorna -1.

8. Test contains(): verifica che gli elementi specificati siano contenuti nella sottolista.
   Test-cases:
   - Si verifica che i valori 6, 12, 13 siano contenuti nella sottolista e che il metodo contains() ritorni il valore true.
     Post-condition: la sottolista rimane invariata.
     Execution record: test superato se gli elementi sono contenuti nella sottolista ed il valore ritornato è true.
   - Si verifica che i valori 4, 15, 20 non siano contenuti nella sottolista e che il metodo contains() ritorni il valore false.
     Post-condition: la sottolista rimane invariata.
     Execution record: test superato se gli elementi non sono contenuti nella sottolista ed il valore ritornato è false.
   - Si verifica che tentando la ricerca di un elemento di riferimento null venga lanciata l'eccezione NullPointerException.
     Post-condition: la sottolista rimane invariata.
     Execution record: test superato se viene lanciata l'eccezione NullPointerException.

9. Test lastIndexOf(): verifica che l'indice corrispondente all'ultima occorrenza dell'elemento specificato sia quella attesa.
   Test-cases:
   - Si inseriscono in coda alla sottolista quattro nuovi valori: 5, 5, 10, 10. Si verifica che l'ultimo indice corrispondente al valore 5 sia 11 e che l'ultimo indice corrispondente al valore 10 sia 13.
     Post-condition: si inseriscono i 4 elementi specificati in coda alla sottolista e quindi a partire dalla posizione 15 della lista.

Execution record: test superato se gli indici ritornati dal metodo corrispondono a quelli attesi.

- Si verifica che l'ultimo indice corrispondente all'elemento 15, che non appartiene alla sottolista, sia -1.
  Post-condition: la lista rimane invariata.
  Execution record: test superato se la posizione ritornata è quella attesa.
- Si verifica che tentando d'ottenere l'ultimo indice di un elemento di riferimento null venga lanciata l'eccezione NullPointerException.
  Post-condition: la sottolista rimane invariata.
  Execution record: test superato se viene lanciata l'eccezione NullPointerException.

10. Test set() : verifica che modificando un elemento della sottolista in una posizione specificata le modifiche abbiano effetto anche sulla lista.
    Test cases:
    - Si verifica che dopo aver modificato il valore dell'elemento in posizione 0, che precedentemente era 5 e che ora è 10, il valore ritornato dal metodo corrisponda al vecchio valore. Si verifica poi che il nuovo valore in posizione 0 sia quello specificato. Si ripete il test modificando il valore in posizione 5, che prima conteneva il valore 10, con il valore 30. Si verifica poi che l'elemento in posizione 5 ora sia cambiato. In entrambi i casi si testa che le modifiche apportate sulla sottolista siano andate a buon fine anche sulla lista.
      Post-condition: gli elementi in posizione 0 e 5 vengono modificati.
      Execution record: test superato i due elementi risultano modificati.
    - Si verifica che tentando la modifica di un elemento in posizione 15, quindi non appartenente alla sottolista, venga lanciata l'eccezione IndexOutOfBoundsException.
      Post-condition: la sottolista rimane invariata.
      Execution record: test superato se viene lanciata l'eccezione IndexOutOfBoundsException.
    - Si verifica che tentando di modificare un elemento in una posizione valida della sottolista, ma con un valore null, venga lanciata l'eccezione IndexOutOfBoundsException.
      Post-condition: la sottolista rimane invariata.
      Execution record: test superato se viene lanciata l'eccezione IndexOutOfBoundsException.

11. Test remove() di un oggetto: verifica che la rimozione di un elemento della sottolista vada a buon fine.
    Test cases:
    - Si verifica che la rimozione degli elementi 10 e 14 della sottolista vada a buon fine, quindi che il metodo ritorni il valore true e che la dimensione di lista e sottolista sia diminuita di due unità.
      Post condition: gli elementi specificati come parametri vengono rimossi. La sottolista restringe la sua vista, ovvero si decrementa l'indice limite superiore.
      Execution record: test superato se gli elementi 10 e 14 vengono rimossi.
    - Si verifica che tentando la rimozione di un elemento non appartenete alla lista, il valore 1, il valore di ritorno del metodo sia false.
      Post-condition: la sottolista rimane invariata.
      Execution record: test superato se il metodo ritorna il valore false.

12. Test remove() con indice: verifica che la rimozione di un elemento della sottolista, alla posizione specificata, vada a buon fine.
    Test cases:

- Si verifica che la rimozione dell'elemento in posizione 5 ritorni il valore che si trova in quella posizione, ovvero valore 10. Si riesegue il test rimuovendo l'elemento in posizione 8 il cui valore corrispondente deve essere 13. Inoltre la dimensione della sottolista deve diminuire di una unità ed anche quella della lista.
  Post-condition: l'elemento in posizione 5 e quello in posizione 8 vengono rimossi.
  Execution record: test superato se l'elemento in posizione 5 e quello in posizione 8 vengono rimossi.
- Si verifica che tentando la rimozione di un elemento di indice troppo grande venga lanciata l'eccezione IndexOutOfBoundsException.
  Post-condition: la sottolista rimane invariata.
  Execution record: test superato se viene lanciata l'eccezione IndexOutOfBoundsException.
- Si verifica che tentando la rimozione di un elemento di indice troppo piccolo, cioè fuori dagli indici validi della sottolista, venga lanciata l'eccezione IndexOutOfBoundsException.
  Post-condition: la sottolista rimane invariata.
  Execution record: test superato se viene lanciata l'eccezione IndexOutOfBoundsException.

13. Test removeAll() : verifica che a seguito dell'invocazione di removeAll() le dimensioni della sottolista e della lista siano diminuite di almeno un'unità.
    Test cases:
    - Si verifica che passata una collezione contenente gli interi 6, 8, 9, 12, 16, 18 questi vengano, solo in parte, rimossi dalla sottolista poiché non tutti sono contenuti. Si testa che le dimensioni della lista e della sottolista siano diminuite di 4 unità e che il metodo ritorni true.
      Post-condition: la sottolista e la lista perdono gli interi 6, 8, 9 e 12.
      Execution record: test superato se il metodo ritorna true e le dimensioni diminuiscono di 4 unità.
    - Si verifica che tentando di passare al metodo una collezione con riferimento null venga lanciata l'eccezione NullPointerException.
      Post-condition: la sottolista rimane invariata.
      Execution record: test superato se viene lanciata l'eccezione NullPointerException.

14. Test retainAll() : verifica che la sottolista mantenga al suo interno tutti gli elementi di una collezione passata come parametro.
    Test cases:
    - Si verifica che, passata una collezione contenente i numeri 5, 6, 7, 10, 12 e 13, questi siano gli unici valori mantenuti nella sottolista e nella lista. Si verifica che il valore di ritorno del metodo sia true e che gli elementi mantenuti siano soltanto quelli presenti anche nella collezione passata.
      Post-condition: la sottolista mantiene al suo interno solo gli elementi contenuti anche nella collezione specificata.
      Execution record: test superato se e solo se la dimensione della lista risulta cambiata ed il metodo ritorna true.
    - Si verifica che, passata una collezione contenente elementi che non sono contenuti nella sottolista, la dimensione della sottolista sia nulla.
      Post-condition: la sottolista risulta vuota.
      Execution record: test superato se il valore di ritorno è false.
    - Si verifica che passando una sottolista che contiene gli stessi elementi della lista iniziale la dimensione della sottolista di riferimento rimanga invariata e che il metodo ritorni false.
      Post-condition: la sottolista rimane invariata.

Execution record: test superato se viene ritornato false e la sottolista rimane invariata.

15. Test ListIterator() : verifica il corretto funzionamento del ListIterator della sottolista. Si testano tutti i metodi dell'interfaccia: hasNext(), next(), hashPrevious(), previous(), nextIndex(), previousIndex(), add(), remove(), set().
    Test cases:
    - Si verifica che facendo scorrere l'iteratore da sinistra a destra (usando hasNext() e next()) in posizione toIndex, l'indice precedente alla posizione corrente è l'ultima posizione valida nella sottolista (si testa cosi previousIndex().
      Post-condition: la lista rimane invariata, cursore dell'iteratore in ultima posizione.
      Execution record: test superato se l'iteratore è in posizione toIndex e se l'indice ritornato è corretto.
    - Si fa scorrere il ListIterator da destra a sinistra nella sottolista fino alla posizione fromIndex (si usano i metodi hasPrevious() e previous()), si verifica che l'indice successivo (testando nextIndex()) sia fromIndex+1.
      Post-condition: la sottolista rimane invariata, cursore dell'iteratore in posizione fromIndex.
      Execution record: test superato se viene ritornato l'indice corretto.
    - S'inserisce con il metodo add() dell'iteratore un elemento in testa alla sottolista, si verifica tramite il metodo get() l'avvenuta modifica.
      Post-condition: si aggiunge un elemento nella sottolista.
      Execution record: test superato se è stato aggiunto un elemento alla sottolista e la dimensione è aumentata di una unità.
    - Si fa scorrere l'iteratore nuovamente a destra della sottolista e si rimuove l'ultimo elemento attraverso il remove() dell'iteratore.
      Post-condition: viene rimosso l'ultimo elemento della sottolista.
      Execution record: test superato se la dimensione risulta diminuita di una unità e l'elemento rimosso è quello atteso.
    - Si modifica poi l'elemento in ultima posizione con un nuovo valore (100), usando il set() del ListIterator, e si verifica l'avvenuta modifica usando il metodo get() della sottolista.
      Post-condition: viene modificato l'elemento in ultima posizione della sottolista.
      Execution record: test superato se l'elemento è stato modificato correttamente.

16. Test sort() : verifica gli elementi della sottolista siano stati ordinati secondo un ordinamento crescente.
    Pre-condition: si crea una lista di interi non ordinati (6, 3, 5, 1, 2, 4) e si richiede a questa una sottolista tra gli indici 1 e 5, quindi la sottolista comprende gli elementi (3, 5, 1, 2).
    Test cases:
    - S'invoca il metodo sort() sulla sottolista creata. Si verifica che la sottolista risulti ordinata in ordine crescente e che la lista usata come backing storage sia ordinata solo nel range di indici che comprendono la sottolista.
      Post-condition: la sottolista risulta ordinata, la lista è ordinata solo nella parte che comprende gli indici della sottolista.
      Execution record: test superato se la sottolista è ordinata.

17. Si testa infine che cosa accade se viene fatta un'aggiunta oppure una rimozione di un elemento nella sottolista prima e dopo la posizione corrente mentre si sta eseguendo un'iterazione sugli elementi della lista.
    Test cases:

- Si aggiunge un elemento prima della posizione corrente. Poiché la sottolista si basa su una lista a sua volta memorizzata su un vector, gli elementi subiscono uno shift verso destra quindi ci sarà un elemento che viene ritornato due volte dall'iteratore. La dimensione della sottolista viene aggiornata, aumenta di una unità.
  Post-condition: si aggiunge un elemento alla sottolista.
  Execution record: test superato se viene aggiornata la dimensione della sottolista a seguito dell'inserimento.
- Si aggiunge un elemento dopo la posizione corrente. Come prima, gli elementi della sottolista subiscono uno shift verso destra, ma in questo caso anche il nuovo elemento viene ritornato dall'iteratore. Anche in questo caso la dimensione della sottolista viene aggiornata.
  Post-condition: si aggiunge un elemento alla sottolista.
  Execution record: test superato se viene aggiornata la dimensione della sottolista a seguito dell'inserimento.
- Si rimuove un elemento prima della posizione corrente. Per le stesse osservazioni fatte precedentemente, gli elementi subiscono questa volta uno shift verso sinistra quindi ci sarà un elemento che non viene ritornato dall'iteratore, in particolare quello nella posizione corrente del cursore prima di fare la rimozione. La dimensione della sottolista viene aggiornata, diminuisce di una unità.
  Post-condition: si rimuove un elemento dalla sottolista.
  Execution record: test superato se viene aggiornata la dimensione della sottolista a seguito della rimozione.
- Si rimuove un elemento dopo la posizione corrente. Come prima gli elementi della sottolista subiscono uno shift verso sinistra, ma in questo caso l'elemento che è stato rimosso non viene ritornato dall'iteratore. Anche in questo caso la dimensione della sottolista viene aggiornata.
  Post-condition: si rimuove un elemento dalla sottolista.
  Execution record: test superato se viene aggiornata la dimensione della sottolista a seguito dell'inserimento.

Test dell'adapter: CollectionObjectAdapter

La suite TestCollectionObjectAdapter contiene i test relativi all'object adapter implementato per adattare l'interfaccia Collection. Si crea un CollectionObjectAdapter come istanza di classe che verrà portato ad uno stato opportuno prima dell'esecuzione di qualsiasi test.

Variabili di ambiente:
JUNIT_HOME=/usr/share/java
CLASSPATH=$CLASSPATH:$JUNIT_HOME/junit4.jar

Pre-condition: per ognuno dei test successivi si parte da una collezione contenente i numeri interi da 0 a 9.

1. Test add() e size(): verifica che l'inserimento degli elementi all'interno della collezione vada a buon fine e che la dimensione della collezione sia corretta.
   Test cases:
   - Oltre ai valori con cui si è inizializzata la collezione si inseriscono altri 10 valori interi multipli di 10 (0, 10, 20…90). Si testa quindi che la dimensione della collezione sia raddoppiata.
     Post-condition: vengono aggiunti altri 10 elementi alla collezione.
     Execution record: test superato se la dimensione della collezione è raddoppiata.
   - Si verifica che ad ogni inserimento in valore ritornato dal metodo add() sia true.
     Post-condition: la collezione rimane invariata.
     Execution record: test superato se ad ogni inserimento il valore ritornato dal metodo è true.
   - Si verifica che tentando l'inserimento di un elemento di riferimento null venga lanciata l'eccezione NullPointerException.
     Post-condition: la collezione rimane invariata.
     Execution record: test superato se viene lanciata l'eccezione NullPointerException.

2. Test iterator() : verifica che l'iteratore ritornato dall'adapter ritorni gli elementi nello stesso ordine con sui sono stati memorizzati nella collezione. Si testano i metodi della sua interfaccia quindi: next() e hashNext().
   Test cases:
   - Si verifica che gli elementi che vengono ritornati dall'iteratore siano ritornati nello stesso ordine posizionale corrispondente a quello con cui sono stati memorizzati nella collezione.
     Post-condition: la collezione rimane invariata.
     Execution record: test superato se le posizioni degli elementi ritornati dall'iteratore corrispondono alle posizioni degli elementi della lista.

3. Test addAll() : verifica che tutti i valori di una nuova collezione siano inseriti nella collezione iniziale.
   Test cases:
   - Si verifica che passata una collezione, una lista in questo caso, contenente i numeri da 10 a 19 questi siano aggiunti nella collezione. Si testa che il valore di ritorno del metodo sia true e che gli elementi inseriti siano quelli presenti nella lista passata. Si eseguono confronti diretti tra i numeri generati iterativamente tramite ciclo for ed i valori ritornati dall'iteratore.
     Post-condition: oltre ai 10 interi inizialmente presenti si aggiungono altri 10 interi da 10 a 19.
     Execution record: test superato se la collezione contiene tutti gli elementi che conteneva prima più tutti gli elementi che sono contenuti nella collezione passata.

- Si verifica che tentando di passare al metodo una collezione con riferimento null venga lanciata l'eccezione NullPointerException.
  Post-condition: la collezione rimane invariata.
  Execution record: test superato se viene lanciata l'eccezione NullPointerException.

4. Test clear(): verifica che a seguito della cancellazione di tutti gli elementi della collezione la dimensione di questa sia 0.
   Test cases:
   - Si invoca il metodo clear() e si testa che la dimensione della collezione sia nulla.
     Post-condition: la collezione non contiene elementi.
     Execution record: test superato se tutti gli elementi sono stati rimossi.

5. Test contains(): verifica che l'elemento specificato sia contenuto nella lista.
   Test cases:
   - Si verifica che la collezione contenga i numeri pari da 0 a 8, quindi il valore di ritorno ad ogni verifica deve essere true.
     Post-condition: la collezione rimane invariata.
     Execution record: test superato se gli elementi sono contenuti nella collezione.
   - Si verifica che la collezione non contenga gli interi multipli di 10 (10, 20…90) quindi il valore ritornato dal metodo è false ad ogni verifica.
     Execution record: test superato i valori non sono presenti nella collezione.
   - Si verifica che tentando la ricerca di un elemento di riferimento null venga lanciata l'eccezione NullPointerException.
     Post-condition: la collezione rimane invariata.
     Execution record: test superato se viene lanciata l'eccezione NullPointerException.

6. Test containsAll() : verifica che la collezione contenga tutti gli elementi di una collezione passata come parametro.
   Test cases:
   - Si verifica che passato un set contenente i numeri da 5 a 9 questi siano contenuti nella collezione. Si verifica che il valore di ritorno del metodo sia true.
     Post-condition: la collezione rimane invariata.
     Execution record: test superato se la collezione contiene tutti gli elementi della collezione passata.
   - Si testa che passato un set contenente i numeri pari da 10 a 18 ed i valori 0, 1, 2 questi non siano contenuti tutti nella collezione. Si controlla che il valore di ritorno del metodo sia false.
     Post-condition: la collezione rimane invariata.
     Execution record: test superato se la collezione ha la stessa dimensione iniziale ed il metodo ritorna false.
   - Si verifica che tentando di passare al metodo una collezione con riferimento null venga lanciata l'eccezione NullPointerException.
     Post-condition: la collezione rimane invariata.
     Execution record: test superato se viene lanciata l'eccezione NullPointerException.

7. Test toArray() : verifica che l'array ritornato dall'adapter contenga gli elementi nello stesso ordine in cui sono memorizzati nella collezione.
   Test cases:

- Si testa che gli elementi dell'array siano disposti nello stesso ordine posizionale con cui sono memorizzati nella collezione.
  Post-condition: la collezione rimane invariata.
  Execution record: test superato se le posizioni degli elementi dell'array corrispondono alle posizioni degli elementi della collezione.
- Si cancella il contenuto della collezione. Si verifica che l'array corrispondente ad una collezione vuota sia vuoto.
  Post-condition: la collezione risulta priva di elementi.
  Execution record: test superato se l'array ritornato risulta vuoto.

8. Test isEmpty(): verifica che a seguito della cancellazione di tutti gli elementi della collezione questa sia vuota.
   Test cases:
   - Si verifica che a seguito dell'invocazione del metodo clear() il valore ritornato dal metodo isEmpty() sia true.
     Post-condition: la collezione è vuota.
     Execution record: test superato se la collezione risulta vuota e quindi il valore di ritorno è true.
   - Si testa che dopo l'inizializzazione, senza l'invocazione del metodo clear(), il valore ritornato dal metodo isEmpty() sia false e la dimensione della lista sia quindi quella iniziale.
     Post-condition: la collezione rimane invariata.
     Execution record: test superato se il valore di ritorno del metodo è false e se la dimensione della lista rimane invariata.

9. Test remove() di un oggetto: verifica che la rimozione di un elemento dalla collezione vada a buon fine.
   Test cases:
   - Si verifica che la rimozione numero 6 della collezione venga eseguita correttamente, quindi che il metodo ritorni il valore true. Inoltre la dimensione della lista deve diminuire di una unità.
     Post-condition: l'elemento specificato come parametro viene rimosso.
     Execution record: test superato l'intero 6 viene rimosso.
   - Si verifica che tentando la rimozione di un elemento non appartenete alla lista, il numero 12, il valore di ritorno del metodo sia false.
     Post-condition: la collezione rimane invariata.
     Execution record: test superato se il metodo ritorna il valore false e la dimensione della collezione rimane invariata.
   - Si verifica che tentando la rimozione di un elemento di riferimento null venga lanciata l'eccezione NullPointerException.
     Post-condition: la collezione rimane invariata.
     Execution record: test superato se viene lanciata l'eccezione NullPointerException.
   - Si cancella il contenuto della collezione, si testa che cercando di rimuovere un qualsiasi elemento, poiché la collezione è vuota, venga ritornato il valore false.
     Post-condition: si eliminano tutti gli elementi della collezione.
     Execution record: test superato se viene ritornato il valore false.

10. Test removeAll() : verifica che a seguito dell'invocazione di removeAll() la dimensione della lista sia diminuita di almeno un'unità.

Test cases:

- Si testa che la collezione risulti modificata, cioè che siano stati rimossi dalla collezione solo gli elementi contenuti nella collezione passata come parametro. La collezione specificata è inizializzata con gli interi da 5 a 9.
  Post-condition: la collezione contiene i soli elementi da 0 a 4.
  Execution record: test superato se la collezione contiene i soli elementi da 0 a 4, se la dimensione della collezione è dimezzata e se il metodo ritorna true.
- Si verifica che passata una collezione contenente i numeri pari da 10 a 18 questi non vengano rimossi dalla collezione poiché non contenuti.
  Post-condition: la collezione rimane invariata.
  Execution record: test superato se la collezione ha la stessa dimensione iniziale e se il metodo ritorna false.
- Si verifica che tentando di passare al metodo una collezione con riferimento null venga lanciata l'eccezione NullPointerException.
  Post-condition: la lista rimane invariata.
  Execution record: test superato se viene lanciata l'eccezione NullPointerException.

11. Test equals() : verifica che due collezioni sono uguali se e solo se contengono gli stessi elementi ed hanno la stessa dimensione.
    Test cases:
    - Si testa che dopo aver creato una collezione che ha lo stesso riferimento di quella di classe le due collezioni siano uguali e cha abbiano lo stesso hashcode.
      Post-condition: le collezioni rimangono invariate.
      Execution record: test superato se le collezioni sono uguali ed hanno lo stesso hashcode.
    - Si verifica che dopo aver creato una collezione contenete gli stessi elementi di quella iniziale a seguito dell'invocazione di equals il valore ritornato sia true. Si testa poi che le due collezioni abbiano lo stesso hashcode, che è la somma degli hashcode dei vari elementi presenti.
      Post-condition: la collezione iniziale rimane invariata.
      Execution record: test superato se le due collezioni sono uguali.
    - Si testa che dopo aver creato un set contenete gli stessi elementi della collezione iniziale a seguito dell'invocazione di equals il valore ritornato sia true. Si testa poi che set e la collezione iniziale abbiano lo stesso hashcode, che è la somma degli hashcode dei vari elementi presenti.
      Post-condition: la collezione ed il set rimangono invariati.
      Execution record: test superato se il set e la collezione sono uguali.
    - Si verifica che dopo aver creato una lista contenete gli stessi elementi della collezione iniziale a seguito dell'invocazione di equals il valore ritornato sia true. Si testa poi che la lista e la collezione iniziale abbiano lo stesso hashcode.
      Post-condition: la collezione e la lista rimangono invariati.
      Execution record: test superato se la lista e la collezione sono uguali.
    - Si testa che dopo aver creato una lista contenete elementi differenti da quella iniziale a seguito dell'invocazione di equals il valore ritornato sia false. Si testa poi che la lista e la collezione iniziale abbiano hashcode differenti. Il test viene eseguito anche su un sott'insieme degli elementi della lista iniziale, ma il risultato è comunque false poiché le collezioni non hanno la stessa lunghezza.
      Post-condition: la collezione e la lista rimangono invariati.
      Execution record: test superato se le collezioni sono diverse.

- Si verifica che tentando di confrontare la collezione iniziale con un qualsiasi altro tipo di oggetto, non Collection, venga lanciata l'eccezione ClassCastException.
  Post-condition: la collezione rimane invariata.
  Execution record: test superato se viene lanciata l'eccezione ClassCastException.
- Si verifica che tentando di confrontare la collezione iniziale con un oggetto di riferimento null venga lanciata l'eccezione NullPointerException.
  Post-condition: la collezione rimane invariata.
  Execution record: test superato se viene lanciata l'eccezione NullPointerException.

12. Test retainAll() : verifica che la collezione mantenga al suo interno tutti gli elementi di una collezione passata come parametro.
    Test cases:
    - Si verifica che passata una collezione contenente i numeri da 5 a 9, questi siano gli unici valori mantenuti nella collezione iniziale. Si verifica che il valore di ritorno del metodo sia true e che gli elementi mantenuti siano soltanto quelli presenti anche nella collezione passata.
      Post-condition: la collezione mantiene al suo interno solo gli elementi contenuti anche in quella specificata.
      Execution record: test superato se la collezione contiene tutti e soli gli elementi di quella passata. Il metodo ritorna true se e solo se la dimensione della collezione risulta cambiata.
    - Si verifica che passata una collezione contenente gli stessi numeri contenuti anche nella collezione iniziale, il valore di ritorno del metodo sia false.
      Post-condition: la collezione rimane invariata.
      Execution record: test superato se il valore di ritorno è false.
    - Si verifica che tentando di passare al metodo una collezione con riferimento null venga lanciata l'eccezione NullPointerException.
      Post-condition: la collezione rimane invariata.
      Execution record: test superato se viene lanciata l'eccezione NullPointerException.

Test dell'adapter: MapObjectAdapter
La suite TestMapObjectAdapter contiene i test relativi all'object adapter implementato per adattare l'interfaccia Map. Si crea un MapObjectAdapter come istanza di classe che verrà portato ad uno stato opportuno prima dell'esecuzione di qualsiasi test.

Variabili di ambiente:
JUNIT_HOME=/usr/share/java
CLASSPATH=$CLASSPATH:$JUNIT_HOME/junit4.jar

Pre-condition: prima di ogni test si inizializza la mappa con 10 coppie chiave-valore. Le chiavi sono numeri interi appartenenti all'intervallo [0,9], i valori invece sono stringhe denominate valorei, dove i è un numero intero appartenente all'intervallo [0,9].

1. Test clear(): verifica che a seguito dell'invocazione del metodo la dimensione della mappa sia nulla.
   Test cases:
   - Si verifica che dopo aver invocato il metodo clear() la dimensione della mappa è 0.
     Post-condition: la mappa non contiene coppie chiave valore.
     Execution record: test superato se la dimensione della mappa è 0.

2. Test isEmpty(): verifica che la mappa sia vuota, cioè che non contenga coppie chiave-valore.
   Test cases:
   - Si verifica che dopo aver invocato il metodo clear(), il metodo isEmpty() ritorni il valore true.
     Post-condition: la mappa è vuota.
     Execution record: test superato se la mappa risulta vuota, quindi il metodo ritorna il valore true.
   - Si testa che invocando il metodo isEmpty(), subito dopo l'inizializzazione, il valore ritornato sia false, poiché la mappa contiene coppie chiave-valore.
     Post-condition: la mappa contiene le entry con cui è stata inizializzata.
     Execution record: test superato se la mappa non risulta vuota, quindi il metodo ritorna il valore false.

3. Test containsKey(): si verifica la mappa contenga le chiavi che vengono specificate come parametro.
   Test cases:
   - Si verifica che dopo aver invocato il metodo containsKey() a cui si passano in sequenza le chiavi: 0, 3, 9 questo ritorni il valore true ad ogni invocazione, poiché le chiavi hanno un mapping nella mappa.
     Post-condition: la mappa rimane inalterata.
     Execution record: test superato se ad ogni verifica di una chiave il metodo ritorna true.
   - Si testa che dopo aver invocato il metodo containsKey() a cui si passano in sequenza le chiavi: 10, 20, 30 questo ritorni il valore false ad ogni invocazione, poiché le chiavi non hanno un mapping nella mappa.
     Post-condition: la mappa rimane inalterata.
     Execution record: test superato se ad ogni verifica di una chiave il metodo ritorna false.
   - Si verifica che tentando di passare al metodo una chiave con riferimento null venga lanciata l'eccezione NullPointerException.
     Post-condition: la mappa rimane invariata.

Execution record: test superato se viene lanciata l'eccezione NullPointerException.

4. Test containsValue(): verifica se la mappa contiene i valori che vengono specificate come parametro.
   Test cases:
   - Si verifica che dopo aver invocato il metodo containsValue(), a cui si passano in sequenza i valori: "valore2", "valore5" e "valore7", questo ritorni il valore true ad ogni invocazione, poiché i valori hanno una chiave ad essi associata nella mappa.
     Post-condition: la mappa rimane inalterata.
     Execution record: test superato se ad ogni verifica di un valore il metodo ritorna true.
   - Si testa che dopo aver invocato il metodo containsValues() a cui si passano in sequenza i valori: "valore50", "valore60" e "valore70", questo ritorni false ad ogni invocazione, poiché i valori non hanno una chiave ad essi associata nella mappa.
     Post-condition: la mappa rimane inalterata.
     Execution record: test superato se ad ogni verifica di un valore il metodo ritorna false.
   - Si verifica che tentando di passare al metodo un valore con riferimento null venga lanciata l'eccezione NullPointerException.
     Post-condition: la mappa rimane invariata.
     Execution record: test superato se viene lanciata l'eccezione NullPointerException.

5. Test get(): verifica che le chiavi passate come parametro corrispondano al valore atteso.
   Test cases:
   - Si verifica che dopo aver invocato il metodo get(), a cui si passano in sequenza le chiavi: 1, 2 e 11, questo ritorni il valore corrispondente alle chiavi ovvero "valore1", "valore2" e null, poiché l'ultima chiave non ha alcun mapping nella mappa.
     Post-condition: la mappa rimane inalterata.
     Execution record: test superato se ad ogni verifica di una chiave il metodo ritorna il valore corretto corrispondente alla chiave.
   - Si verifica che tentando di passare al metodo una chiave con riferimento null venga lanciata l'eccezione NullPointerException.
     Post-condition: la mappa rimane invariata.
     Execution record: test superato se viene lanciata l'eccezione NullPointerException.

6. Test put(): verifica che l'inserimento di un insieme di coppie chiave-valore nella mappa vada a buon fine. In realtà tale metodo si potrebbe considerare testato quando si è testato isEmpty(), poiché se quest'ultimo torna false significa che devono essere state inserite delle entry nella mappa.
   Test cases:
   - Si verifica che inserendo tre coppie chiave-valore nella mappa questa incrementi la sua dimensione di tre unità.
     Post-condition: vengono inserite nella mappa tre nuove coppie chiave valore non presenti.
     Execution record: test superato se la dimensione della mappa aumenta di tre unità.
   - Si verifica che tentando di passare al metodo una chiave con riferimento null ed un valore valido oppure una chiave valida ed un valore con riferimento null oppure sia una chiave che un valore con riferimento null venga lanciata l'eccezione NullPointerException.
     Post-condition: la mappa rimane invariata.
     Execution record: test superato se viene lanciata l'eccezione NullPointerException.

7. Test getOrDefault(): verifica che se viene richiesta una chiave che ha una corrispondenza nella mappa allora viene ritornato il valore corrispondente, altrimenti viene ritornato il valore di default passato come parametro.
   Test cases:
   - Si testa che invocando il metodo getOrDefault() con una chiave valida, 9 ad esempio, ed un valore di default valido allora il valore ritornato corrisponde a "valore 9". Se invece viene passata una chiave valida ma che non ha un mapping il valore ritornato è quello passato come parametro di default.
     Post-condition: la mappa rimane invariata.
     Execution record: test superato o il valore ritornato è quello corrispondente alla chiave oppure se corrisponde al valore di default.
   - Si verifica che tentando di passare al metodo una chiave con riferimento null ed un valore di default valido oppure una chiave valida ed un valore di default con riferimento null oppure sia una chiave che un valore di default con riferimento null venga lanciata l'eccezione NullPointerException.
     Post-condition: la mappa rimane invariata.
     Execution record: test superato se viene lanciata l'eccezione NullPointerException in uno dei tre casi.

8. Test remove() di una chiave: verifica che rimuovendo una chiave appartenente alla mappa venga restitutito il valore ad essa corrispondente.
   Test cases:
   - Si testa che tentando di rimuovere la chiave 8, il valore corrispondente ritornato, poiché appartenente alla mappa, sia "valore8".
     Post-condition: viene rimosso dalla mappa l'entry di chiave 8.
     Execution record: test superato se viene rimossa l'entry corrispondente alla chiave specificata e se la dimensione della mappa diminuisce di una unità.
   - Si testa che tentando di rimuovere la chiave 12, il valore corrispondente ritornato, poiché non appartenente alla mappa, sia null.
     Post-condition: la mappa rimane inalterata.
     Execution record: test superato se viene ritornato il valore null e se la dimensione della mappa risulta inalterata.
   - Si verifica che tentando di passare al metodo una chiave con riferimento null venga lanciata l'eccezione NullPointerException.
     Post-condition: la mappa rimane invariata.
     Execution record: test superato se viene lanciata l'eccezione NullPointerException.

9. Test remove() di una entry: verifica che l'entry passata come parametro sia contenuta nella mappa e se ciò accade deve essere rimossa.
   Test cases:
   - Si testa che invocando la rimozione dell'entry <1, valore1> questa restituisca valore true, poiché esiste nella mappa un'entry di chiave 1 e valore "valore1".
     Post-condition: viene rimossa l'entry <1, valore1> e la dimensione della mappa diminuisce di una unità.
     Execution record: test superato se il valore ritornato dal metodo è true e se la dimensione diminuisce di una unità.
   - Si testa che invocando la rimozione dell'entry <1, valore10> questa restituisca valore false, poiché non esiste nella mappa un'entry di chiave 1 e valore "valore10".

Post-condition: la mappa rimane inalterata.
Execution record: test superato se il valore ritornato dal metodo è false.

- Si verifica che tentando di passare al metodo una chiave con riferimento null ed un valore valido oppure una chiave valida ed un valore con riferimento null oppure sia una chiave che un valore con riferimento null venga lanciata l'eccezione NullPointerException.
  Post-condition: la mappa rimane invariata.
  Execution record: test superato se viene lanciata l'eccezione NullPointerException in uno dei tre casi.

10. Test replace() con 2 parametri (chiave e valore): verifica che le modifiche apportate ed un'entry specificata siano andate a buon fine.
    Test cases:
    - Si modificano le entry di chiave: 5, 7 e 9 assegnando a queste tre nuovi valori: "nuovo5", "nuovo7" e "nuovo9". Si verifica che dopo aver apportato le modifiche affettivamente i valori vecchi non siano più presenti.
      Post-condition: le entry di chiave 5, 7 e 9 vengono modificate.
      Execution record: test superato se le modifiche sono andate a buon fine.
    - Si cerca di modificate le entry di chiave: 10 e -1 ma poiché le chiavi non hanno alcuna corrispondenza nella mappa il valore ritornato è null.
      Post-condition: la mappa rimane invariata.
      Execution record: test superato se non viene apportata nessuna modifica e viene ritornato in ambo i casi il valore null.
    - Si verifica che tentando di passare al metodo una chiave con riferimento null ed un nuovo valore valido oppure una chiave valida ed un nuovo valore con riferimento null oppure sia una chiave che un nuovo valore con riferimento null venga lanciata l'eccezione NullPointerException.
      Post-condition: la mappa rimane invariata.
      Execution record: test superato se viene lanciata l'eccezione NullPointerException in uno dei tre casi.

11. Test replace() con 3 parametri (chiave, vecchio valore e nuovo valore): verifica che le modifiche apportate ed un'entry siano andate a buon fine.
    Test cases:
    - Si modificano le entry <1, valore1>, <3, valore3> e <6, valore6> assegnando a queste tre nuovi valori: "new1", "new3" e "new6". Si verifica che dopo aver apportato le modifiche affettivamente i valori vecchi non siano più presenti.
      Post-condition: le entry di chiave 1, 3 e 6 vengono modificate.
      Execution record: test superato se ad ogni inserimento il metodo ritorna il valore true e se le modifiche sono andate a buon fine.
    - Si cerca di modificate le entry <3, valore8>, <1, valore9>, <6, valore2> ma poiché le entry specificate non sono contenute nella mappa le modifiche non vengono apportate quindi si verifica che i valori corrispondenti alle chiavi specificate siano ancora quelli vecchi.
      Post-condition: la mappa rimane invariata.
      Execution record: test superato se non viene apportata nessuna modifica ed il valore ritornato nei tre casi è false.
    - Si verifica che tentando di passare al metodo:
      - una chiave con riferimento null, un vecchio valore valido ed un nuovo valore valido;

- una chiave valida, un vecchio valore con riferimento null ed un nuovo valore valido;
- una chiave valida, un vecchio valore valido ed un nuovo valore con riferimento null;

venga lanciata l'eccezione NullPointerException.
Post-condition: la mappa rimane inalterata.
Execution record: test superato se viene lanciata l'eccezione NullPointerException.
Non sono state testate tutte le possibili combinazioni dei parametri ma soltanto alcune. L'importante era testare che venisse lanciata l'eccezione in alcuni casi particolari.

12. Test putIfAbsent(): verifica che l'inserimento di coppie chiave-valore non presenti nella mappa vada a buon fine.
    Test cases:
    - Si inseriscono nella mappa tre entry non presenti: <10, new10>, <11, new11> e <12, new12>, si verifica poi che i valori corrispondenti alle chiavi: 10, 11 e 12 siano i valori "new10", "new11" e "new12" e che il valore di ritorno ad ogni inserimento sia null.
      Post-condition: si aggiungono tre entry alla mappa.
      Execution record: test superato se le entry vengono inserite.
    - Si tenta l'inserimento nella mappa di tre entry, la cui chiave ha già un mapping su un valore: <0, new0>, <1, new1> e <2, new2>, si verifica poi che i valori corrispondenti alle chiavi: 0, 1 e 2 siano i valori "valore0", "valore1" e "valore2" che sono anche i valori di ritorno di ogni inserimento.
      Post-condition: la mappa rimane invariata.
      Execution record: test superato se la mappa non modifica la sua dimensione.
    - Si verifica che tentando di passare al metodo una chiave con riferimento null ed un valore valido oppure una chiave valida ed un valore con riferimento null oppure sia una chiave che un valore con riferimento null venga lanciata l'eccezione NullPointerException.
      Post-condition: la mappa rimane invariata.
      Execution record: test superato se viene lanciata l'eccezione NullPointerException in uno dei tre casi.

13. Test putAll(): verifica che l'inserimento nella mappa corrente di tutte le entry appartenenti ad una mappa passata come parametro vada a buon fine.
    Test cases:
    - Si crea una mappa in cui s'inseriscono le quattro entry: <100, valore100>, <200, valore200>, <300, valore300>, <400, valore400> poi si invoca il putAll() sulla mappa iniziale e si passa come parametro la seconda mappa. Si verifica che la dimensione della mappa sia aumentata di 4 unità.
      Post-condition: vengono aggiunte alla mappa tutte le entry della mappa specificata.
      Execution record: test superato se la mappa aumenta la sua dimensione.
    - Si verifica che tentando di passare al metodo una mappa di riferimento null venga lanciata l'eccezione NullPointerException.
      Post-condition: la mappa rimane invariata.
      Execution record: test superato se viene lanciata l'eccezione NullPointerException in uno dei tre casi.

14. Test equals() : verifica che due mappe sono uguali se e solo se contengono le stesse entry ed hanno la stessa dimensione.

Test cases:

- Si verifica che dopo aver creato una mappa contenete le stesse coppie chiave-valore di quella iniziale, a seguito dell'invocazione di equals, il valore ritornato sia true. Si testa poi che le due mappe abbiano lo stesso hashcode, che è la somma degli hashcode delle entry presenti nella mappa.
  Post-condition: la mappa rimane inalterata.
  Execution record: test superato se le due mappe sono uguali.
- Si verifica che dopo aver creato una mappa contenete coppie chiave-valore differenti da quella iniziale, a seguito dell'invocazione di equals, il valore ritornato sia false. Si testa poi che le due mappe abbiano hashcode differenti. Il test viene eseguito anche su un sottinsieme delle entry della lista iniziale, ma il risultato è comunque false poiché le mappe non hanno la stessa lunghezza.
  Post-condition: la mappa rimane inalterata.
  Execution record: test superato se le due mappe sono diverse.
- Si verifica che tentando di confrontare la mappa iniziale con un qualsiasi altro tipo di oggetto venga lanciata l'eccezione ClassCastException.
  Post-condition: la mappa rimane invariata.
  Execution record: test superato se viene lanciata l'eccezione ClassCastException.
- Si verifica che tentando di confrontare la mappa iniziale con un oggetto di riferimento null venga lanciata l'eccezione NullPointerException.
  Post-condition: la mappa rimane invariata.
  Execution record: test superato se viene lanciata l'eccezione NullPointerException.

Per i test dei metodi keySet(), entrySet() e values() si rimanda alla documentazione, nelle successive pagine del documento, sviluppata esclusivamente per gli oggetti che sono ritornati da tali metodi.

Test dell'adapter: SetObjectAdapter
La suite TestObjectAdapter contiene i test relativi all'object adapter implementato per adattere l'interfaccia terget Set. Si crea un SetObjectAdapter come istanza di classe che verrà poi portato in uno stato consistente prima dell'esecuzione di qualsiasi test.

Variabili di ambiente:
JUNIT_HOME=/usr/share/java
CLASSPATH=$CLASSPATH:$JUNIT_HOME/junit4.jar

Pre-condition: prima di eseguire qualsiasi test si inizializza il set con 10 nomi di colori: "Giallo", "Rosso", "Arancione", "Viola", "Verde", "Nero", "Marrone", "Grigio", "Blu" e "Bianco".

1. Test size(): verifica la corretta dimensione del set.
   Test cases:
   - Si verifica che a seguito dell'inizializzazione la dimensione del set sia 10.
     Post-condition: il set non contiene coppie chiave valore.
     Execution record: test superato se la dimensione del set è 10.

2. Test isEmpty(): verifica che il set sia vuoto, cioè che non contenga elementi.
   Test cases:
   - Si verifica che dopo aver invocato il metodo clear(), il metodo isEmpty() ritorni il valore true.
     Post-condition: il set è vuoto.
     Execution record: test superato se il set è vuoto, quindi il metodo ritorna il valore true.
   - Si testa che invocando il metodo isEmpty(), subito dopo l'inizializzazione, il valore ritornato sia false, poiché il set contiene coppie degli elementi.
     Post-condition: il set contiene gli elementi con cui è stato inizializzato.
     Execution record: test superato se il set non risulta vuoto, quindi il metodo ritorna il valore false.

3. Test add(): verifica che l'inserimento di alcuni elementi nel set vada a buon fine.
   Test cases:
   - Si verifica che dopo aver inserito tre elementi, che non erano presenti: "Azzurro", "Lilla" e "Magenta" la dimensione del set sia aumentata di tre unità.
     Post-condition: vengono inseriti tre nuovi elementi nel set.
     Execution record: test superato se la dimensione del set aumenta di tre unità.
   - Si verifica che tentando d'inserire tre elementi, che erano già presenti nel set, questi non siano stati inseriti, perché non sono accettati i duplicati e quindi il set ha la stessa dimensione iniziale.
     Post-condition: il set rimane invariato.
     Execution record: test superato se la dimensione del set non cambia.
   - Si verifica che tentando d'aggiungere al set iniziale un elemento di riferimento null venga lanciata l'eccezione NullPointerException.
     Post-condition: il set rimane invariato.
     Execution record: test superato se viene lanciata l'eccezione NullPointerException.

4. Test addAll(): verifica che l'aggiunta degli elementi contenuti in una collezione specificata al set vada a buon fine solo il set risulta cambiato a seguito della chiamata al metodo.
   Test cases:

- Si verifica che passando come parametro una collezione contenente quattro elementi, di cui due già presenti (“Verde” e “Blu”) e due non presenti (“Azzurro”, ”Argento”), la dimensione del set deve aumentare di due unità. Il metodo deve ritornare il valore true.
  Post-condition: vengono inseriti solo due dei quattro elementi contenuti nella collezione.
  Execution record: test superato se la dimensione del set aumenta di due unità.
- Si testa che passando come parametro una collezione contenente quattro elementi, tutti già presenti (“Verde”, “Blu”, “Rosso” e “Nero”), la dimensione del set deve rimanere invariata. Il metodo deve ritornare il valore false.
  Post-condition: il set rimane inalterato.
  Execution record: test superato se viene ritornato il valore false ed il set non aumenta di dimensione.
- Si testa che tentando di passare al metodo una collezione di riferimento null venga lanciata l’eccezione NullPointerException.
  Post-condition: il set rimane invariato.
  Execution record: test superato se viene lanciata l’eccezione NullPointerException.

5. Test contains(): verifica che il set contenga gli elementi specificati.
   Test cases:
   - Si verifica che gli elementi passati al metodo contains(): “Giallo”, “Nero” e “Arancione” siano contenuti nel set, quindi il metodo ritorna il valore true per ognuno di essi.
     Post-condition: il set rimane inalterato.
     Execution record: test superato se gli elementi risultano contenuti nel set.
   - Si verifica che gli elementi passati al metodo contains(): “Lilla”, “Magenta” e “Azzurro” non siano contenuti nel set, quindi il metodo ritorna il valore false per ognuno di essi.
     Post-condition: il set rimane inalterato.
     Execution record: test superato se gli elementi non risultano contenuti nel set.
   - Si testa che tentando di passare al metodo un elemento di riferimento null venga lanciata l’eccezione NullPointerException.
     Post-condition: il set rimane invariato.
     Execution record: test superato se viene lanciata l’eccezione NullPointerException.

6. Test containsAll(): verifica che il set contenga gli elementi che sono presenti nella collezione specificata.
   Test cases:
   - Si verifica che gli elementi contenuti nella collezione passata al metodo conainsAll(): “Verde”, “Viola”, “Bianco” e “Blu” siano contenuti nel set, quindi il metodo ritorna il valore true poiché tutti gli elementi risultano presenti nella collezione.
     Post-condition: il set rimane inalterato.
     Execution record: test superato se tutti gli elementi della collezione risultano contenuti nel set.
   - Si verifica che gli elementi contenuti nella collezione passata al metodo conainsAll(): “Verde”, “Grigio”, “Lilla” e “Magenta” non siano tutti contenuti nel set, quindi il metodo ritorna il valore false poiché non tutti gli elementi risultano presenti nella collezione.
     Post-condition: il set rimane inalterato.
     Execution record: test superato se alcuni elementi della collezione non risultano contenuti nel set
   - Si testa che tentando di passare al metodo una collezione di riferimento null venga lanciata l’eccezione NullPointerException.

Post-condition: il set rimane invariato.
Execution record: test superato se viene lanciata l'eccezione NullPointerException.

7. Test remove(): verifica che la rimozione degli elementi specificati vada a buon fine.
   Test cases:
   - Si verifica che rimuovendo tre elementi contenuti nel set ("Viola", "Rosso" e "Grigio") il valore ritornato ad ogni rimozione sia true. Si controlla poi, che la dimensione del set sia diminuita di 3 unità.
     Post-condition: vengono rimossi tre elementi dal set.
     Execution record: test superato se ad ogni rimozione corrisponde il valore true e se la dimensione del set è diminuita.
   - Si verifica rimuovendo tre elementi non contenuti nel set ("Lilla", "Magenta" e "Argento") che il valore ritornato ad ogni rimozione sia false. Si controlla poi, che la dimensione del set sia rimasta invariata.
     Post-condition: il set rimane invariato.
     Execution record: test superato se ad ogni rimozione corrisponde il valore false e se la dimensione del set rimane invariata.
   - Si testa che tentando di passare al metodo un elemento di riferimento null venga lanciata l'eccezione NullPointerException.
     Post-condition: il set rimane invariato.
     Execution record: test superato se viene lanciata l'eccezione NullPointerException.

8. Test removeAll(): verifica che la rimozione degli elementi contenuti nella collezione specificata vada a buon fine.
   Test cases:
   - Si verifica che rimuovendo dal set gli elementi appartenenti alla collezione specificata, di cui tre contenuti anche nel set ("Verde", "Viola" e "Bianco") ed uno non appartenente ("Lilla"), la dimensione del set diminuisca di tre unità. Il valore ritornato dal metodo, poiché la dimensione è variata, è true.
     Post-condition: vengono rimossi tre elementi dal set.
     Execution record: test superato se la dimensione del set risulta diminuita rispetto alla dimensione iniziale e viene ritornato il valore true.
   - Si verifica che tentando di rimuovere dal set gli elementi appartenenti alla collezione specificata: "Magenta", "Lilla" e "Argento" ma tutti e tre non appartenenti al set, la dimensione del set deve rimanere invariata. Il metodo ritorna il valore false.
     Post-condition: il set rimane invariato.
     Execution record: test superato se la dimensione del set risulta uguale a quella iniziale ed il valore ritornato è false.
   - Si testa che tentando di passare al metodo una collezione di riferimento null venga lanciata l'eccezione NullPointerException.
     Post-condition: il set rimane invariato.
     Execution record: test superato se viene lanciata l'eccezione NullPointerException.

9. Test retainAll(): verifica che vengano mantenuti all'interno del set tutti gli elementi che sono contenuti all'interno della collezione passata.
   Test cases:
   - Si testa che passando una collezione contenente gli elementi: "Verde", "Viola", "Bianco", "Blu" e "Lilla" poiché quattro di questi sono contenuti nel set allora devono essere gli unici

ad essere mantenuti nel set. Si controlla quindi che la dimensione del set sia pari a 4 e che il valore ritornato sia true.
Post-condition: vengono mantenuti solo 4 elementi all'interno della collezione.
Execution record: test superato se il set contiene i soli elementi "Verde", "Viola", "Bianco" e "Blu" ed il valore ritornato è true.

- Si verifica che passando una collezione contenente gli stessi elementi che sono contenuti nel set, poiché a seguito dell'invocazione del metodo la dimensione non cambia, si controlla che la dimensione del set sia rimasta invariata e quindi che il valore di ritorno sia false.
  Post-condition: la collezione rimane invariata.
  Execution record: test superato se la collezione rimane invariata ed il valore ritornato è false.
- Si testa che tentando di passare al metodo una collezione di riferimento null venga lanciata l'eccezione NullPointerException.
  Post-condition: il set rimane invariato.
  Execution record: test superato se viene lanciata l'eccezione NullPointerException.

10. Test toArray() : verifica che l'array ritornato dall'adapter contenga gli elementi che sono memorizzati nella collezione.
    Test cases:
    - Si crea un set in cui s'inseriscono tutti gli elementi che sono contenuti nell'array, s'invoca poi il metodo containsAll() sul set iniziale a cui si passa la collezione appena creata. Si testa che il valore di ritorno sia true. In questo caso non è possibile fare riferimento all'ordine degli elementi poiché non si sa come vengono ritornati dalla tabella hash.
      Post-condition: la collezione rimane invariata.
      Execution record: test superato se il set iniziale contiene tutti gli elementi che sono contenuti nel secondo set e quindi il valore ritornato dal metodo è true.

11. Test iterator() : verifica che l'iteratore ritorni gli elementi nello stesso ordine con cui vengono ritornati dal metodo toArray().
    Test cases:
    - Si richiede un iteratore al set, si controlla poi che ogni elemento ritornato nell'ordine imposto dall'iteratore, corrisponda ad uno stesso elemento nell'array.
      Post-condition: sia il set che l'array rimangono invariati.
      Execution record: test superato se gli elementi dell'array sono nello stesso ordine di quelli ritornati dall'iteratore.

15. Test equals() : verifica che due set sono uguali se e solo se contengono gli stessi elementi ed hanno la stessa dimensione.
    Test cases:
    - Si verifica che dopo aver creato un set contenete gli stessi elementi di quello iniziale, a seguito dell'invocazione di equals, il valore ritornato sia true. Si testa poi che i due set abbiano lo stesso hashcode, che è la somma degli hashcode degli elementi presenti nel set.
      Post-condition: il set rimane invariato.
      Execution record: test superato i due set sono uguali.
    - Si verifica che dopo aver creato un set contenete elementi differenti da quello iniziale, a seguito dell'invocazione di equals, il valore ritornato sia false. Si testa poi che i due set abbiano hashcode differenti. Il test viene eseguito anche su un subset degli elementi del set iniziale, ma il risultato è comunque false poiché i set non hanno la stessa lunghezza.
    - Post-condition: il set rimane invariato.

Execution record: test superato se i due set sono diversi.

- Si verifica che tentando di confrontare il set iniziale con un qualsiasi altro tipo di oggetto venga lanciata l'eccezione ClassCastException.
  Post-condition: il set rimane invariato.
  Execution record: test superato se viene lanciata l'eccezione ClassCastException.
- Si verifica che tentando di confrontare il set iniziale con un oggetto di riferimento null venga lanciata l'eccezione NullPointerException.
  Post-condition: il set rimane invariato.
  Execution record: test superato se viene lanciata l'eccezione NullPointerException.

Test dell'adapter: KeySet

La suite TestKeySet contiene i test relativi al class adapter implementato per adattare la gestione delle chiavi di una mappa attraverso alcuni metodi dell'interfaccia Set. Si crea un MapObjectAdapter come istanza di classe e si invoca su di esso il metodo keySet() per ottenere il set delle chiavi su cui poter eseguire i test.

Variabili di ambiente:
JUNIT_HOME=/usr/share/java
CLASSPATH=$CLASSPATH:$JUNIT_HOME/junit4.jar

Pre-condition: prima di eseguire qualsiasi test si inizializza la mappa con le coppie chiave-valore <i, valorei> dove i è un numero intero appartenente all'intevallo [0, 9]. Poi si richiede un KeySet contenente le sole chiavi della mappa. Quindi in questo caso il set contiene i soli numeri interi da 0 a 9.

1. Test add() : verifica che il metodo add() non possa aggiungere elementi a KeySet.
   Test cases:
   - Si tenta l'aggiunta all'interno del set dell'intero 0, ma questo non viene inserito poiché il metodo non è stato implementato e lancia l'eccezione UnsupportedOperationException.
     Post-condition: sia KeySet che la mappa rimangono invariati.
     Execution record: test superato se viene lanciata l'eccezione UnsupportedOperationException.

2. Test addAll() : verifica che il metodo addAll() non possa aggiungere elementi KeySet.
   Test cases:
   - Si tenta l'aggiunta all'interno del set degli elementi che sono contenuti nella collezione specificata, ma questi non vengono inseriti poiché il metodo non è stato implementato e lancia l'eccezione UnsupportedOperationException.
     Post-condition: sia KeySet che la mappa rimangono invariati.
     Execution record: test superato se viene lanciata l'eccezione UnsupportedOperationException.

3. Test clear() : verifica che se vengono cancellati tutti gli elementi da KeySet questi devono essere cancellati anche dalla mappa e vice-versa.
   Test cases:
   - Si verifica che invocando il metodo clear() sul set, sia la mappa che KeySet devono risultare vuoti.
     Post-condition: sia KeySet che la mappa devono essere vuoti.
     Execution record: test superato se il set e la mappa non contengono elementi.
   - Si testa che invocando il metodo clear() sulla mappa, sia il set che la mappa devono risultare vuoti.
     Post-condition: sia KeySet che la mappa devono essere vuoti.
     Execution record: test superato se il set e la mappa non contengono elementi.

4. Test remove() : verifica che cancellando alcuni elementi dal set questi vengono eliminati anche dalla mappa e vice-versa.
   Test-cases:

- Si verifica che rimuovendo da KeySet le chiavi: 1, 2 e 3 vengono rimosse le corrispondenti entry nella mappa, ovvero si controlla che le rispettive dimensioni di KeySet e della mappa siano diminuite di tre unità.
  Post-condition: sia KeySet che la mappa hanno diminuito la dimensione di tre unità.
  Execution record: test superato se vengono eliminati gli elementi dal set e dalla mappa.
- Si testa che rimuovendo dalla mappa le entry di chiave: 1, 2 e 3 vengono rimosse le chiavi 1, 2, 3 da KeySet, ovvero si controlla che le rispettive dimensioni di KeySet e della mappa siano diminuite di tre unità.
  Post-condition: sia KeySet che la mappa hanno diminuito la dimensione di tre unità.
  Execution record: test superato se vengono eliminati gli elementi dal set e dalla mappa.

5. Test put() : verifica che inserendo delle nuove entry nella mappa, le chiavi corrispondenti siano contenute anche in KeySet.
   Test cases:
   - Si aggiungono nella mappa le entry: <100, valore100>, <200, valore200> e <300, valore300> si controlla quindi che le chiavi: 100, 200 e 300 siano visibili in KeySet. Quindi il set deve contenere le tre nuove chiavi.
     Post-condition: a seguito dell'inserimento delle nuove coppie chiave valore anche KeySet risente delle modifiche.
     Execution record: test superato se KeySet contiene le tre nuove chiavi inserite.

6. Test removeAll() : verifica che rimuovendo dal set un insieme di chiavi che hanno un mapping nella mappa, contenute in una collezione specificata, queste siano rimosse anche da essa oltre che da KeySet.
   Test cases:
   - Si rimuovono da KeySet tutte le chiavi che sono contenute nella collezione passata come parametro. La collezione contiene: 1, 2, 5 e 12. Poiché solo tre di esse sono contenute in KeySet e quindi hanno una corrispondenza nella mappa, sono le uniche ad essere rimosse.
     Si controlla quindi che dopo l'invocazione del metodo sia KeySet sia la mappa abbiano diminuito la loro dimensione di tre unità e che il metodo ritorni il valore true.
     Post-condition: sia KeySet che la mappa diminuiscono la loro dimensione di tre unità ed il metodo ritorna true.
     Execution record: test superato se gli elementi vengono rimossi sia dal set che dalla mappa ed il valore di ritornato dal metodo è true.
   - Si tenta la rimozione da KeySet di tutte le chiavi che sono contenute nella collezione passata come parametro. La collezione contiene: 10, 20, 30 e 40. Poiché nessuna delle chiavi è contenuta nel set e quindi nessuna di esse ha una corrispondenza nella mappa, allora la rimozione non va a buon fine. Le rispettive dimensioni rimangono invariate e quindi il metodo ritorna il valore false.
     Post-condition: sia KeySet che la mappa rimangono invariati
     Execution record: test superato se il set e la mappa rimangono invariati.

7. Test retainAll() : verifica che passando a KeySet una collezione di chiavi, che hanno una corrispondenza nella mappa, queste siano le uniche ad essere mantenute in KeySet se contenute.
   Test cases:
   - Si passa a KeySet una collezione contenente le chiavi: 1, 2, 5, 10 e 11. Solo le prime quattro di queste sono contenute nel set quindi a seguito dell'invocazione del metodo retainAll() le

uniche chiavi ad essere mantenute sono solo: 1, 2, 5 e 10. Le modifiche si ripercuotono anche sulla mappa infatti le uniche entry che rimangono sono quelle corrispondenti alle suddette chiavi. Poiché KeySet cambia la sua dimensione il metodo ritorna true.
Post-condition: sia KeySet che la mappa contengono quattro elementi corrispondenti alle chiavi passate nella collezione.
Execution record: test superato se sia la mappa che KeySet hanno dimensione pari a 4.

- Si passa a KeySet una collezione contenente le stesse identiche chiavi che sono contenute in KeySet. Poiché a seguito dell'invocazione del metodo la dimensione del set e della mappa non cambiano il valore ritornato dal metodo è false.
  Post-condition: sia KeySet che la mappa rimangono invariati.
  Execution record: test superato se il metodo ritorna il valore false.

8. Si testa infine che cosa accade se viene modificato il set, e quindi anche la mappa, mentre si sta iterando sugli elementi di KeySet.
   - Poiché l'iteratore del set non è lazy, ma opera su una copia delle chiavi degli elementi che sono presenti nella mappa, se durante l'iterazione vengono inseriti elementi prima o dopo la posizione corrente questi non vengono ritornati dall'iteratore poiché i dati che esso elabora non sono aggiornati.
   - Per lo stesso motivo sopra descritto, se vengono rimossi degli elementi prima o dopo la posizione corrente questi vengono comunque ritornati dall'iteratore perché i dati sono delle copie e non sono aggiornati.

Test dell'adapter: Values

La suite TestValues contiene i test relativi al class adapter implementato per adattare la gestione dei valori di una mappa attraverso alcuni metodi dell'interfaccia Collection. Si crea un MapObjectAdapter come istanza di classe e si invoca su di esso il metodo values() per ottenere una collezione dei valori su cui poter eseguire i test.

Variabili di ambiente:
JUNIT_HOME=/usr/share/java
CLASSPATH=$CLASSPATH:$JUNIT_HOME/junit4.jar

Pre-condition: prima di eseguire qualsiasi test si inizializza la mappa con le coppie chiave-valore <i,valorei> dove i è un numero intero appartenente all'intevallo [0, 9]. Poi si richiede un una collezione di valori invocando il metodo values(). Quindi in questo caso la collezione contiene le stringhe denominate "valorei".

1. Test add() : verifica che il metodo add() non possa aggiungere elementi alla collezione di valori.
   Test cases:
   - Si tenta l'aggiunta all'interno della collezione di valori della stringa "valore0", ma questa non viene inserita poiché il metodo non è stato implementato e lancia l'eccezione UnsupportedOperationException.
     Post-condition: sia la collezione di valori che la mappa rimangono invariati.
     Execution record: test superato se viene lanciata l'eccezione UnsupportedOperationException.

2. Test addAll() : verifica che il metodo addAll() non possa aggiungere elementi alla collezione di valori.
   Test-cases:
   - Si tenta l'aggiunta all'interno della collezione di valori degli elementi che sono contenuti nella collezione specificata, ma questi non vengono inseriti poiché il metodo non è stato implementato e lancia l'eccezione UnsupportedOperationException.
     Post-condition: sia la collezione che la mappa rimangono invariati.
     Execution record: test superato se viene lanciata l'eccezione UnsupportedOperationException.

3. Test clear() : verifica che se vengono cancellati tutti gli elementi dalla collezione di valori devono essere cancellate anche le corrispondenti entry dalla mappa e vice-versa.
   Test cases:
   - Si verifica che invocando il metodo clear() sulla collezione, sia la mappa che l'oggetto Values devono risultare vuoti.
     Post-condition: sia Values che la mappa devono essere vuoti.
     Execution record: test superato se la collezione e la mappa non contengono elementi.
   - Si testa che invocando il metodo clear() sulla mappa, sia Values che la mappa devono risultare vuoti.
     Post-condition: sia Values che la mappa sono vuoti.
     Execution record: test superato se la collezione e la mappa non contengono elementi.

4. Test remove() : verifica che cancellando alcuni elementi dall'oggetto Values vengono eliminate anche le corrispondenti entry dalla mappa e vice-versa.

Test cases:

- Si verifica che rimuovendo da Values i valori: “valore1”, “valore2” e “valore3” vengono rimosse le corrispondenti entry nella mappa, ovvero si controlla che le rispettive dimensioni di Values e della mappa siano diminuite di tre unità.
  Post-condition: sia la collezione di valori che la mappa hanno diminuito la dimensione di tre unità.
  Execution record: test superato se vengono eliminati gli elementi dalla collezione e dalla mappa.
- Si testa che rimuovendo dalla mappa le entry di chiave: 1, 2 e 3 vengono rimossi i valori ad esse corrispondenti dalla collezione, ovvero si controlla che le rispettive dimensioni di Values e della mappa siano diminuite di tre unità.
  Post-condition: sia Values che la mappa hanno diminuito la dimensione di tre unità.
  Execution record: test superato se vengono eliminati gli elementi dalla collezione di valori e dalla mappa.

5. Test put() : verifica che inserendo delle nuove entry nella mappa, i valori corrispondenti siano contenuti anche nell’oggetto values.
   Test cases:
   - Si aggiungono nella mappa le entry: <100, valore100>, <200, valore200> e <300, valore300> si controlla quindi che i valori: “valore100”, “valore200” e “valore300” siano visibili in Values.
     Post-condition: a seguito dell’inserimento delle nuove coppie chiave valore anche la collezione di valori risente delle modifiche.
     Execution record: test superato se Values contiene i tre nuovi valori inseriti.

6. Test removeAll() : verifica che rimuovendo da Values un insieme di valori che hanno un mapping nella mappa, contenuti in una collezione specificata, questi siano rimossi anche da essa oltre che da Values.
   Test cases:
   - Si rimuovono da Values tutti i valori che sono contenuti nella collezione passata come parametro. La collezione contiene: “valore1”, “valore2”, “valore5” e “valore12”. Poiché solo tre di essi sono contenute in Values e quindi hanno una corrispondenza nella mappa, sono gli unici ad essere rimossi.
     Si controlla quindi che dopo l’invocazione del metodo sia la collezione di valori sia la mappa abbiano diminuito la loro dimensione di tre unità e che il metodo ritorni il valore true.
     Post-condition: sia Values che la mappa diminuiscono la loro dimensione di tre unità ed il metodo ritorna true.
     Execution record: test superato se gli elementi vengono rimossi sia da Values che dalla mappa ed il valore di ritorno del metodo è true.
   - Si tentano di rimuovere da Values tutti i valori che sono contenute nella collezione passata come parametro. La collezione contiene: “valore10”, “valore20”, “valore30” e “valore40”. Poiché nessuno dei valori è contenuto in values e quindi nessuno di essi ha una corrispondenza nella mappa, allora la rimozione non va a buon fine. Le rispettive dimensioni rimangono invariate e quindi il metodo ritorna il valore false.
     Post-condition: sia Values che la mappa rimangono invariati
     Execution record: test superato se la collezione e la mappa rimangono invariati.

7. Test retainAll() : verifica che passando all’oggetto Values una collezione di valori, che hanno una corrispondenza nella mappa, questi siano gli unici ad essere mantenuti in Values se contenuti.

Test cases:

- Si passa a Values una collezione contenente i valori: “valore1”, “valore2”, “valore5”, “valore10” e “valore11”. Solo i primi quattro di questi sono contenute in Values quindi a seguito dell’invocazione del metodo retainAll() gli unici ad essere mantenuti sono solo: “valore1”, “valore2”, “valore5” e “valore10”. Le modifiche si ripercuotono anche sulla mappa infatti le uniche entry che rimangono sono quelle corrispondenti ai suddetti valori. Poiché Values cambia la sua dimensione il metodo ritorna true.
  Post-condition: sia Values che la mappa contengono quattro elementi corrispondenti ai valori passati nella collezione.
  Execution record: test superato se sia la mappa che la collezione di valori hanno dimensione pari a 4.
- Si passa alla collezione di valori una collezione contenente gli stessi identici valori che sono contenuti in Values. Poiché a seguito dell’invocazione del metodo la dimensione della collezione e della mappa non cambiano il valore ritornato dal metodo è false.
  Post-condition: sia Values che la mappa rimangono invariati.
  Execution record: test superato se il metodo ritorna il valore false.

8. Si testa infine che cosa accade se viene modificata la collezione di valori, e quindi anche la mappa, mentre si sta iterando sugli elementi di Values.
   - Poiché l’iteratore dei Values non è lazy, ma opera su una copia dei valori degli elementi che sono presenti nella mappa, se durante l’iterazione vengono inseriti elementi prima o dopo la posizione corrente questi non vengono ritornati dall’iteratore poiché i dati che esso elabora non sono aggiornati.
   - Per lo stesso motivo sopra descritto, se vengono rimossi dei valori da Values prima o dopo la posizione corrente questi vengono comunque ritornati dall’iteratore perché i dati sono delle copie e non sono aggiornati.

Test dell'adapter: EntrySet

La suite TestEntrySet contiene i test relativi al class adapter implementato per adattare la gestione di coppie chiave-valore, che implementato l'intefaccia Map.Entry, attraverso alcuni metodi dell'interfaccia Set. Si crea un MapObjectAdapter come istanza di classe e si invoca su di esso il metodo entrySet() per ottenere un set di Map.Entry su cui poi poter eseguire i vari test.

Variabili di ambiente:
JUNIT_HOME=/usr/share/java
CLASSPATH=$CLASSPATH:$JUNIT_HOME/junit4.jar

Pre-condition: prima di eseguire qualsiasi test si inizializza la mappa con le coppie chiave-valore <i,valorei> dove i è un numero intero appartenente all'intevallo [0, 9]. Poi si richiede un set di Map.Entry invocando il metodo entrySet().

1. Test add() : verifica che il metodo add() non possa aggiungere elementi al Set di entry.
   Test cases:
   - Si tenta l'aggiunta all'interno della collezione di entry della coppia chiave-valore <0, valore0>, ma questa non viene inserita poiché il metodo non è stato implementato e lancia l'eccezione UnsupportedOperationException.
     Post-condition: sia il set che la mappa rimangono invariati.
     Execution record: test superato se viene lanciata l'eccezione UnsupportedOperationException.

2. Test addAll() : verifica che il metodo addAll() non possa aggiungere elementi al set di entry.
   Test cases:
   - Si tenta d'aggiungere all'interno del set di entry di coppie chiave-valore che sono contenute nella collezione specificata, ma queste non vengono inserite poiché il metodo non è stato implementato e lancia l'eccezione UnsupportedOperationException.
     Post-condition: sia il set che la mappa rimangono invariati.
     Execution record: test superato se viene lanciata l'eccezione UnsupportedOperationException.

3. Test clear() : verifica che se vengono cancellati tutti gli elementi dall'EntrySet questi devono essere cancellati anche dalla mappa e vice-versa.
   Test cases:
   - Si verifica che invocando il metodo clear() sul set, sia la mappa che il l'EntrySet devono risultare vuoti.
     Post-condition: sia EntrySet che la mappa devono essere vuoti.
     Execution record: test superato se il set e la mappa non contengono elementi.
   - Si testa che invocando il metodo clear() sulla mappa, sia il set che la mappa devono risultare vuoti.
     Post-condition: sia EntrySet che la mappa devono essere vuoti.
     Execution record: test superato se il set e la mappa non contengono elementi.

4. Test remove() : verifica che cancellando alcune entry dal set queste vengono eliminate anche dalla mappa e vice-versa.
   Test-cases:

- Si verifica che rimuovendo da EntrySet le entry: <0, valore0>, <1, valore1> e <2, valore2> vengono rimosse le corrispondenti entry nella mappa, ovvero si controlla che le rispettive dimensioni di EntrySet e della mappa siano diminuite di tre unità.
  Post-condition: sia EntrySet che la mappa hanno diminuito la dimensione di tre unità, ad ogni rimozione il metodo ritorna true.
  Execution record: test superato se vengono eliminati le entry dal set e dalla mappa.
- Si testa che cercando di rimuovere dal set le entry: <0, valore10>, <1, valore11> e <10, valore3> queste non vengono rimosse poiché non c'è nessuna corrispondenza nella mappa e nel set. Quindi le dimensioni di EntrySet e della mappa sono identiche a quelle iniziali.
  Post-condition: sia EntrySet che la mappa rimangono invariati.
  Execution record: test superato se ad ogni rimozione il metodo ritorna false.
- Si verifica che tentando d'invocare il metodo remove() con un parametro di un tipo diverso da Map.Entry venga lanciata l'eccezione ClassCastException.
  Post-condition: il set rimane invariato.
  Execution record: test superato se viene lanciata l'eccezione ClassCastException.
- Si verifica che rimuovendo dalla mappa le entry di chiave: 1, 2, e 3 vengano rimosse le corrispondenti entry nella mappa, ovvero si controlla che le rispettive dimensioni di EntrySet e della mappa siano diminuite di tre unità.
  Post-condition: sia EntrySet che la mappa hanno diminuito la dimensione di tre unità.
  Execution record: test superato se vengono eliminati le entry dal set e dalla mappa.

5. Test contains() : verifica se le entry specificate sono contenute o meno in EntrySet.
   Test cases:
   - Si verifica che le entry: <0, valore0>, <1, valore1> e <2, valore2> sono contenute in EntrySet, quindi ad ogni invocazione di contains() viene ritornato il valore true.
     Post-condition: sia EntrySet che la mappa rimangono invariati.
     Execution record: test superato se il metodo ritorna sempre il valore true.
   - Si verifica che le entry: <10, valore10>, <1, valore2> e <9, valore8> non sono contenute in EntrySet, quindi ad ogni invocazione di contains() viene ritornato il valore false.
     Post-condition: sia EntrySet che la mappa rimangono invariati.
     Execution record: test superato se il metodo ritorna sempre il valore false.

6. Test containsAll(): verifica che EntrySet contenga tutte le entry che sono presenti nella collezione specificata.
   Test cases:
   - Si verifica che le entry contenute nella collezione passata al metodo conainsAll():<5, valore5>, <6, valore6> e <7, valorre7> siano contenute nel set, quindi il metodo ritorna il valore true poiché tutte le entry risultano presenti nel set.
     Post-condition: il set e la mappa rimangono inalterati.
     Execution record: test superato se tutte le entry della collezione risultano contenute nel set.
   - Si verifica che le entry contenute nella collezione passata al metodo conainsAll(): <10, valore10>, <5, valore6>, <4, valore7> e <9, valore0> non siano tutte contenute in EntrySet, quindi il metodo ritorna il valore false poiché non tutte le entry risultano presenti nella collezione.
     Post-condition: il set e la mappa rimangono inalterati.
     Execution record: test superato se alcuni elementi della collezione non risultano contenuti in EntrySet.

7. Test removeAll() : verifica che rimuovendo da EntrySet un insieme di entry, contenute in una collezione specificata, queste siano rimosse dalla mappa oltre che dal KeySet.
   Test cases:
   - Si cercano di rimuovere da EntrySet tutte le entry che sono contenute nella collezione passata come parametro. La collezione contiene: <1, valore10>, <5, valore5>, <6, valor6> e <9, valore0>. Poiché solo due di esse sono contenute in EntrySet e quindi hanno una corrispondenza nella mappa, sono le uniche ad essere rimosse.
     Si controlla quindi che dopo l'invocazione del metodo sia EntrySet sia la mappa abbiano diminuito la loro dimensione di due unità e che il metodo ritorni il valore true.
     Post-condition: sia il KeySet che la mappa diminuiscono la loro dimensione di tre unità ed il metodo ritorna true.
     Execution record: test superato se le entry vengono rimosse sia dal set che dalla mappa ed il valore di ritorno del metodo è true.
   - Si tentano di rimuovere da EntrySet tutte le entry che sono contenute nella collezione passata come parametro. La collezione contiene: <10 valore1>, <20, valore2>, <30, valore30> e <2, valore20>. Anche se alcune entry hanno la chiave che in realtà hanno una corrispondenza nella mappa, il valore però ad esse associato non è quello corretto. Quindi nessuna delle entry viene rimossa dal set ed il metodo ritorna il valore false.
     Post-condition: sia EntrySet che la mappa rimangono invariati
     Execution record: test superato se il set e la mappa rimangono invariati ed il metodo ritorna false.

8. Test retainAll() : verifica che passando ad EntrySet una collezione di entry, queste siano le uniche ad essere mantenute in EntrySet, se contenute.
   Test cases:
   - Si passa ad EntrySet una collezione contenente le entry: <1, valore1>, <2, valore2>, <3, valore3> e <4, valore20>. Solo le prime tre di queste sono contenute nel set quindi a seguito dell'invocazione del metodo retainAll() le uniche entry ad essere mantenute sono solo: <1, valore1>, <2, valore2> e <3, valore3>. Le modifiche si ripercuotono anche sulla mappa infatti le entry suddette sono le uniche che rimangono anche all'interno della mappa. Poiché EntrySet cambia la sua dimensione il metodo ritorna true.
     Post-condition: sia EntrySet che la mappa contengono le sole tre entry <1, valore1>, <2, valore2> e <3, valore3>.
     Execution record: test superato sia la mappa che EntrySet hanno dimensione pari a tre unità.
   - Si passa ad EntrySet una collezione contenente le stesse identiche entry che sono contenute nel set. Poiché a seguito dell'invocazione del metodo la dimensione del set e della mappa non cambiano il valore ritornato dal metodo è false.
     Post-condition: sia EntrySet che la mappa rimangono invariati.
     Execution record: test superato se il metodo ritorna il valore false.

9. Se si modifica EntrySet mentre si sta iterando su di esso valgono le stesse osservazioni fatte per KeySet e Values, ovvero, poiché l'iteratore elabora delle copie dei dati, quando si esegue una rimozione oppure un inserimento tali operazioni non vengono viste perché non aggiornano i dati dell'iteratore. Quindi per iterare i dati con le avvenute modifiche è necessario richiedere un nuovo iteratore ad EntrySet.