Test Suite & Test Cases di ListAdapter.java e MapAdapter.java

In questo documento andrò a spiegare brevemente il funzionamento dei test dei due adattatori che sono stati realizzati per l’Homework. Tutti i test verranno eseguiti direttamente dalla classe testRunner.

**ListTest.java** – verranno istanziati 3 oggetti ed un MistAdapter che potremmo usare in ogni test case

Di default il ListAdapter viene inizializzato vuoto

1. Test metodo list\_add()

* Test: public boolean add(Object o), boolean add(int index, Object o)
* Design: corretto inserimento degli elementi all’interno della lista nell’indice corretto se specificato altrimenti aggiunge in coda
* Pre-condizioni: metodo get() funzionante
* Post-condizioni: la lista deve tornare ad essere vuota

Risultato atteso: la lista conterrà un numero di elementi maggiore ogni volta che viene invocato un metodo add() e andrà ad incrementare la dimensione.

1. Test metodo list\_addAll()

* Test: public boolean addAll(HCollection c), public boolean addAll(int index, HCollection c)
* Design: corretto inserimento degli elementi della collection nella lista, nell’indice corretto se specificato altrimenti aggiunge in coda
* Pre-condizioni: metodo get() funzionante
* Post-condizioni: la lista deve tornare ad essere vuota
* Risultato atteso: la lista conterrà un numero di elementi maggiore ogni volta che viene eseguito un metodo addAll() ed andrà ad aumentare la dimensione della lista.

1. Test metodo list\_clear()

* Test: public void clear()
* Design: il metodo deve rimuovere tutti gli elementi contenuti nella lista, lasciandola vuota
* Pre-condizioni: metodo size() funzionante
* Post-condizioni: la lista deve tornare ad essere vuota
* Risultato atteso: la lista non conterrà più alcun elemento

1. Test metodo list\_contains()

* Test: public boolean contains(Object o)
* Design: il metodo deve verificare che nella lista sia contenuto l’oggetto passato come paramentro
* Pre-condizioni: metodo add() funzionante
* Post-condizioni: la lista deve tornare ad essere vuota
* Risultato atteso: il metodo ritornerà il valore true se l’oggetto è presente all’interno della lista

1. Test metodo list\_containsAll()

* Test: public boolean containsAll(HCollection c)
* Design: il metodo deve verificare che all’interno della lista siano presenti tutti gli elementi che sono presenti nella HCollection passata come paramentro
* Pre-condizioni: Il metodo add() deve essere funzionante (sia per list che per HCollection)
* Post-condizioni: la lista deve tornare ad essere vuota
* Risultato atteso: il metodo deve ritornare true se gli elementi della HCollection sono tutti presenti nella lista

1. Test metodo list\_equals()

* Test: public boolean equals(Object o)
* Design: Il metodo deve verificare che i due oggetti siano uguali
* Pre-condizioni: Il metodo add() deve essere funzionante
* Post-condizioni: la lista deve tornare ad essere vuota
* Risultato atteso: il metodo deve ritornare true se i due oggetti sono uguali

1. Test metodo list\_get()

* Test: public Object get(int index)
* Design: il metodo deve verificare che l’elemento inserito sia nella posizione corretta
* Pre-condizioni: il metodo add() deve essere funzionante per poter aggiungere gli elementi nella lista
* Post-condizioni: la lista deve tornare ad essere vuota
* Risultato atteso: Il metodo deve ritornare l’oggetto se presente all’indice della lista

1. Test metodo list\_indexOf()

* Test: public int indexOf(Object o)
* Design: il metodo deve verificare che all’indice passato come parametro corrisponda l’oggetto precedentemente aggiunto
* Pre-condizioni: il metodo add() deve essere funzionante per poter aggiungere gli elementi nella lista
* Post-condizioni: la lista deve tornare ad essere vuota
* Risultato atteso: Il metodo deve ritornare l’indice dell’oggetto passato se presente

1. Test metodo list\_isEmpty()

* Test: public boolean isEmpty()
* Design: Il metodo deve verificare che la dimensione della lista sia vuota
* Pre-condizioni:
* Post-condizioni: la lista deve tornare ad essere vuota
* Risultato atteso: Il metodo restituisce true se la lista risulta vuota

1. Test metodo list\_Iterator()

* Test: public HIterator iterator(), hasNext(), next() e remove() dell’iteratore creato
* Design: Il metodo deve creare un iteratore semplice sulla lista
* Pre-condizioni:
* Post-condizioni: la lista deve tornare ad essere vuota
* Risultato atteso: l’iteratore deve scorrere tanti elementi quanti sono quelli della lista, nell’ordine corretto. hasNext ritorna true solamente se una chiamata a next ritorna un elemento valido. Il metodo ritorna dunque un iteratore valido

1. Test metodo list\_lastIndexOf()

* Test: public int lastIndexOf(Object o)
* Design: il metodo deve ritornare l’indice dell’ultimo elemento o passato
* Pre-condizioni: metodo add() funzionante
* Post-condizioni: la lista deve tornare ad essere vuota
* Risultato atteso: il metodo ritorna l’indice dell’ultimo elemento presente nella lista passato come parametro, se non è presente viene ritornato -1

1. Test metodo list\_listIterator()

* Test: public HListIterator listIterator(int index), next(), hasNext(), previous(), hasPrevious() e remove
* Design: utilizzo del listIterator per scorrere la lista
* Pre-condizioni: il metodo add() deve essere funzionante per poter aggiungere gli elementi nella lista
* Post-condizioni: la lista deve tornare ad essere vuota
* Risultato atteso: l’iteratore deve scorrere tanti elementi quanti sono quelli della lista, nell’ordine corretto. hasNext ritorna true solamente se una chiamata a next ritorna un elemento valido. Dopo aver iterato la lista hasNext ritorna false e next() lancia eccezione (uguale per hasPrevious() e previous())

1. Test metodo list\_remove()

* Test: public Object remove(int index), public boolean remove(Object o)
* Design: entrambi i metodi devono rimuovere l’elemento posto all’indice passato oppure il primo oggetto uguale a quello passato, se è presente nella lista
* Pre-condizioni: il metodo add() deve essere funzionante per poter aggiungere gli elementi nella lista
* Post-condizioni: la lista deve tornare ad essere vuota
* Risultato atteso: La lista deve essere vuota o con elementi ad esclusione da quello che è stato rimosso dalla chiamata del remove

1. Test metodo list\_removeAll()

* Test: public boolean removeAll(HCollection c)
* Design: Il metodo deve essere in grado di rimuovere la prima istanza degli elementi presenti nella HCollection passata come parametro
* Pre-condizioni: Una nuova HList in cui aggiungo elementi con add() e l’aggiunta di elementi nella lista
* Post-condizioni: la lista deve tornare ad essere vuota
* Risultato atteso: La lista vuota o con elementi ad esclusione da quelli che sono stati rimossi dalla chiamata del removeAll tranne le copie, dato che removeAll rimuove solamente la prima istanza degli elementi uguali a quelli presenti nella HCollection

1. Test metodo list\_retainAll()

* Test: public boolean retainAll(HCollection c)
* Design: rimozione di tutti gli elementi che sono presenti nella lista ma non nella HCollection
* Pre-condizioni: La HCollection creata deve avere elementi validi aggiunti correttamente con add()
* Post-condizioni: la lista deve tornare ad essere vuota
* Risultato atteso: il metodo ritornerà true solamente se la dimensione della lista subisce cambiamenti di dimensione dopo l’invocazione del metodo. In quel caso la lista dovrà avere meno oggetti e mantenere quelli che sono presenti anche nella HCollection

1. Test metodo list\_set()

* Test: public Object set(int index, Object element)
* Design: Rimpiazzo di un elemento contenuto nella lista con un altro elemento passato come parametro della funzione
* Pre-condizioni: L’indice deve essere contenuto nella lista e l’elemento non deve essere null
* Post-condizioni: la lista deve tornare ad essere vuota
* Risultato atteso: L’elemento di indice index deve essere rimpiazzato con quello passato come parametro, il metodo ritorna l’oggetto che è stato rimpiazzato

1. Test metodo list\_size()

* Test: public int size()
* Design: Controllo della dimensione della lista
* Pre-condizioni: il metodo add() deve essere funzionante per poter aggiungere gli elementi nella lista
* Post-condizioni: la lista deve tornare ad essere vuota
* Risultato atteso: Il metodo deve ritornare la dimensione della lista

1. Test metodo list\_sublist()

* Test: In questo test viene controllata tutta la sublist
* Design: Questo test verifica tutta la sublist, dalla sua creazione al toArray, i test avvengono nello stesso modo di quelli fatti per il resto del listAdapter, in questo modo si è potuto risparmiare codice copiando i test e cambiando solamente l’oggetto in cui viene invocato il metodo. L’unico test diverso è quello della creazione della subList (costruttore) in cui si verifica la corretta creazione della struttura dagli elementi presenti nella lista
* Pre-condizioni: il metodo add() deve essere funzionante per poter aggiungere gli elementi nella lista, questi elementi verranno poi copiati nella sublist
* Post-condizioni: la lista deve tornare ad essere vuota
* Risultato atteso: La creazione di una sublist con estremi (from, to) rispettivamente (incluso, escluso) in cui posso effettuare le stesse operazioni e chiamare gli stessi metodi della lista madre, in particolare saranno presenti gli stessi metodi e gli stessi iteratori solamente “adattati” per rispettare i limiti degli indici della sublist. Dopo ogni test case della sublist la sublist stessa viene ripulita tramite il suo metodo clear che la rende così utilizzabile per il test case successivo senza il pericolo di fare operazioni pericolose con gli indici degli elementi ancora presenti nella sublist. Alla fine quindi avremo una sublist vuota.

1. Test metodo list\_toArray()

* Test: public Object[] toArray(), public Object[] toArray(Object[] a)
* Design: Il test verifica che i metodi copino gli elementi della lista dentro un array creato o passato come parametro al metodo
* Pre-condizioni: il metodo add() deve essere funzionante per poter aggiungere gli elementi nella lista
* Post-condizioni: la lista deve tornare ad essere vuota
* Risultato atteso: un array contenente gli elementi della lista o dell’array passato, nel caso in cui la dimensione dell’array come parametro sia minore di quella della lista, si crea un nuovo array la cui dimensione è come quella della lista per poi copiarci la lista

**MapTest.java** – verranno istanziate 5 coppie chiave valore e un MapAdapter che potremo usare in ogni test case

Di default il MapAdapter viene inizializzato vuoto