Progetto PR2
ElasticSet<E>

Stefan Daniel Motoc

Novembre 2015

L'implementazione dell'interfaccia ElasticSet prevede la creazione di una collezione di oggetti omogenei, generici, confrontabili, in cui non sono presenti elementi duplicati.

Descrizione principali tecniche utilizzate

Per poter applicare i metodi richiesti, ho scelto di implementare *ElasticSet<E>* come un ArrayList formato da elementi che non si possono ripetere, tra loro ordinati, diminuendo così la complessità del programma. Avendo un insieme ordinato di elementi, infatti, sarà più facile trovare il minimo (rappresentato dall'elemento in posizione 0 nell'array), il massimo (rappresentato dall'elemento in posizione size-1 nell'array), oppure un elemento generico. Quest'ultimo sarà trovato utilizzando la tecnica della *Ricerca Binaria* che restituirà l'indice dell'elemento cercato, se presente nella collezione, (-1-*pos*) altrimenti), dove *pos* rappresenta l'indice che sarebbe assegnato all'elemento se questo fosse presente all'interno della collezione. Usando questa versione modificata della *Ricerca Binaria*, ad ogni inserimento è possibile determinare la posizione del nuovo elemento all'interno dell'array, senza dover quindi usare metodi aggiuntivi per riordinare gli elementi della collezione. Ho inoltre scelto di implementare un'eccezione, chiamata *EmptyCollectionException*, per gestire meglio il caso in cui la collezione è vuota.
I metodi che ritornano dei valori (*glb, lub, min, max, exist, view*) sfruttano l'utilizzo della tecnica di *copia shallow*, più efficiente, ma meno sicura. Si assume quindi che l'utente non vada a modificare la struttura degli oggetti restituiti, perché ciò potrebbe invalidare l'invariante di rappresentazione.

Funzione di Astrazione

$AF(collection) = \{collection.get(0), collection.get(1), ..., collection.get(size-1)\}$

Invariante di Rappresentazione

L'invariante di rappresentazione è dato da:
- $collection \neq null$
- $\forall i. \quad 0 \leq i < collection.size() \quad \Rightarrow \quad collection.get(i) \neq null$
- $\forall i, j. \; 0 \leq i, j < collection.size() \;\; \&\& \;\; i \neq j \quad \Rightarrow \quad collection.get(i) \neq collection.get(j)$
- $\forall i, j. \; 0 \leq i, j < collection.size() \;\; \&\& \;\; j > i \quad \Rightarrow \quad collection.get(j) > collection.get(i)$
- *gli elementi inseriti devono essere dello stesso tipo*

Dimostro quindi che i metodi implementati preservano l'Invariante di Rappresentazione, suddividendoli in diversi gruppi:

```
public boolean contains(E e);
public boolean isEmpty();
public E glb(E e);
public E lub(E e);
public E max();
public E min();
public E exist(E f, E t) throws EmptyCollectionException;
public void print();
public int ricercaBin(E key);
```

Questi metodi, in quando non modificano la collezione, preserveranno sicuramente l'Invariante di Rappresentazione.

```
public boolean remove(E e) throws EmptyCollectionException;
public boolean rmFirst() throws EmptyCollectionException;
public boolean rmLast() throws EmptyCollectionException;
```

I metodi che prevedono la rimozione di un elemento dalla collezione, preservano l'Invariante di Rappresentazione perché:
- se $collection = null$, lancerebbero l'eccezione checked *EmptyCollectionException*
- ogni elemento e della collezione continuerebbe ad essere $e \neq null$ anche in seguito ad una rimozione
- se prima della rimozione tutti gli elementi erano diversi, continueranno ad esserlo anche dopo aver eliminato uno di essi dalla collezione
- se prima della rimozione la collezione era ordianta, lo sarà anche dopo aver eliminato un elemento
- gli elementi della collezione rimasti in seguito alla rimozione continueranno ad essere dello stesso tipo

```
public boolean add(E e);
```

L'aggiunta di un elemento alla collezione preserva l'Invariante di Rappresentazione perché:
- aggiungendo un elemento alla collezione, quest'ultima sarà sicuramente non vuota;
- se l'elemento e aggiunto fosse tale che $e = null$, il metodo lancerebbe un'eccezione;
- prima che l'elemento e venga aggiunto alla collezione, il metodo controlla che esso non sia già presente al suo interno. In tal caso l'elemento non verrà aggiunto, mantenendo quindi la collezione formata da elementi tutti diversi tra loro;
- il metodo *add* utilizza la *Ricerca Binaria* per trovare la corretta collocazione, rappresentata dall'indice i, dell'elemento e all'interno della collezione, in base al suo valore. L'elemento e viene quindi inserito in modo tale che l'elemento precedente sia minore o uguale mantenere la collezione ordinata.
- il tipo dell'elemento e inserito sarà necessariamente uguale a quello della collezione a cui viene aggiunto. Se così non fosse, verrebbe generato un errore durante la fase di compilazione.

Descrizione Test

Per controllare la corretta implementazione dei metodi richiesti, ho realizzato due batterie di test:

TestSet1: Il primo test considera una collezione di stringhe, applicando i metodi dell'interfaccia ElasticSet;

TestSet2: Il secondo test considera una collezione di interi, applicando i metodi dell'interfaccia ElasticSet;

Compilazione ed Esecuzione

Per eseguire il programma da terminale, entrare nella cartella del progetto e digitare:

make build
per compilare il codice

make test1
per eseguire il primo test sulle stringhe

make test2
per eseguire il primo test sugli interi

make run
per eseguire il codice e le due batterie di test

make buildandrun
per compilare ed eseguire il codice e le due batterie di test

make clean
per eliminare i file gi compilati