Progetto#2 LinkedList – Corso di Programmazione Orientata agli Oggetti

È assegnata la seguente interfaccia List<T> appartenente al package poo.util, che si ispira liberamente al collection framework di Java (package java.util), mantenendo per i metodi omonimi lo stesso significato, tipi di eccezioni etc.

```
package poo.util;
public interface List<T> extends Iterable<T>{
    int size();
    void clear();
    boolean contains( T e );
    boolean isEmpty();
    void add( T e );
    void remove( T e );
    ListIterator<T> listIterator();
    ListIterator<T> listIterator( int pos );
    void addFirst( T e );
    void addLast( T e );
    T removeFirst();
    T removeLast();
    T getFirst();
    T getLast();
    void sort( Comparator<T> c );
}//List
```

Prima parte

Sfruttando la possibilità dei metodi default e metodi statici (consentiti da Java 8 e versioni successive) nelle interfacce, concretizzare, con l'aiuto degli iteratori, quanti più metodi è possibile direttamente nell'interfaccia List<T>. Metodi che non possono essere realizzati in List<T>, concretizzarli, ove possibile, come parte dello sviluppo di una classe astratta AbstractList<T> che implementa List<T> ed espone certamente una realizzazione dei metodi toString(), equals() e hashCode(). Sviluppare quindi la classe concreta LinkedList<T> che estende AbstractList<T> e memorizza gli elementi su una lista concatenata a doppio puntatore.

Nel progetto di LinkedList<T> rilevante è la realizzazione delle strutture di iterazione (iteratore semplice e list iterator). Si ricorda che un iteratore, quando non è al di fuori della lista, si trova logicamente *tra* due elementi consecutivi e non *su* un elemento. Al fini di identificare correttamente, quando esiste, l'elemento corrente (definito da previous() o next()), può essere opportuno introdurre nella classe dell'iteratore una coppia di puntatori <previous, next> in mezzo ai quali si colloca logicamente l'iteratore. Inoltre una enumerazione può essere utilmente introdotta per esprimere i possibili movimenti in avanti (FORWARD), indietro (BACKWARD) o l'assenza di movimento (UNKNOWN), e mantenere in una variabile di istanza l'ultimo movimento effettuato con l'iteratore. Poiché Java non consente di dichiarare un tipo enum in una inner class, collocare l'enumerazione direttamente nell'outer class LinkedList. Si nota che dopo un movimento in avanti (next()), l'elemento corrente è riferito dal puntatore previous; dopo un movimento indietro (previous()) l'elemento corrente è riferito dal puntatore next.

Siccome un ListIterator *è* un Iterator, limitarsi ad implementare una sola inner class per il ListIterator<T>. Un oggetto di tale classe, a questo punto, è restituibile sia dal metodo iterator() (di Iterable<T>) che dai metodi listIterator([pos]). Quando si restituisce un oggetto list iterator mediante iterator(), esso verrà utilizzato con i soli tre metodi hasNext(), next() e remove() come prescritto dal tipo statico Iterator<T> etc.

Il metodo sort() riceve un oggetto Comparator<T> e ordina la lista this in accordo all'algoritmo bubble sort.

Il codice Java ottenuto va testato accuratamente. Si suggerisce di predisporre una classe Main col metodo main che verifica le "varie" condizioni di funzionamento. Nei casi di dubbio, confrontarsi con l'uso di java.util.LinkedList<T>. Come altro test, provare a rimpiazzare nella classe PolinomioLL l'uso della LinkedList di Java, con le classi/interfacce parte di questo progetto.

Parte facoltativa

Dopo aver messo a punto le classi di cui alla prima parte del progetto, provare a realizzare una GUI in grado di esporre una struttura a menù che permetta di evocare tutte le operazioni disponibili su una lista. Inizialmente la GUI deve consentire di scegliere il tipo degli elementi della lista, tra Integer o String. Successivamente, subito dopo un'operazione sulla lista, occorre mostrare su una text area lo stato corrente della lista, es. come [12, 4, 5, 15], o un'eccezione sollevata. Quando si accende un iteratore, si deve visualizzare la posizione (^) dove si trova l'iteratore. Dopo una next si deve mostrare la nuova posizione mentre in un campo di testo Corrente si deve mostrare l'elemento corrente o ? se esso non è definito. Si nota che scegliendo il comando iterator, la freccia deve trovarsi inizialmente prima del primo elemento, e risultano abilitati i comandi hasNext e next. Dopo una next() che ha successo, si abilita la remove() etc. Se invece si accende un ListIterator, si può specificare la posizione iniziale del list iterator come quella di default (identica a quella di iterator) o si può fornire un indice intero (compreso tra 0 e size()) che individua l'elemento della lista prima del quale va piazzata la freccia dell'iteratore (specificando size(), la freccia va posta dopo l'ultimo elemento, specificando size()-1, la freccia va posta tra penultimo ed ultimo etc.). Con ListIterator devono poter essere accessibili tutti i metodi di ListIterator. Naturalmente, occorre garantire che di momento in momento risultano abilitati tutti e soli i menu item che rappresentano operazioni ammissibili nello stato attuale della lista.