**The LinkedList Class and the Iterator, ListIterator, and Iterable Interfaces**

Iterators are what makes the linked list better than array list.

The LinkedList Class:

Table

Description automatically generated

Drawings are running time for double linked list.

**THE ITERATOR**

An iterator can be viewed as a moving place marker that keeps track of the current position in a particular linked list

All containers should have an iterator

An Iterator object for a list starts at the first node

The programmer can move the Iterator by calling its next method

The Iterator stays on its current list item until it is needed

An Iterator traverses in O(n) while a list traversal using get() calls in a linked list is O(n^2)

for (int i = 0; i < size; ++i) get(i); 🡪 get is O(n), for loop is O(n), so total is O(n^2)

To be able to access elements faster, we use Iterator since iterators keep the current position and return the next one will be constant time

**Iterator Interface**

The Iterator interface is defined in java.util

The List interface declares the method iterator which returns an Iterator object that iterates over the elements of that list

Text

Description automatically generated with low confidence

next 🡪 returns the next value and iterator is advanced from that position

If next is never used before the remove operation, we will get an exception.

An Iterator is conceptually between elements; it does not refer to a particular object at any given time

Diagram

Description automatically generated

If you use next method, “Returned element” will be returned and iterator will be advanced to new position.

If we use remove after this next method, returned element will be removed.

In the following loop, we process all items in List<Integer> through an Iterator:

Iterator<Integer> iter = aList.iterator();

while (iter.hasNext()) {

int value = iter.next();

//Do something with value

…

}

So overall running time is linear time θ(n) since the loop will be repeated n times.

Iterators and Removing Elements

You can use the Iterator remove() method to remove items from a list as you access them

remove() deletes the most recent element returned

You must call next() before each remove(); otherwise, and IllegalStateException will be thrown

LinkedList.remove vs Iterator.remove:

* LinkedList.remove must walk down the list each time, then remove, so in general it is O(n)
* Iterator.remove removes items without starting over at the beginning, so in general it is O(1)

What about the remove method of the Iterator of the ArrayList? 🡪 O(n) bc after you remove, you have to shift all the elements. So O(1) is correct for only linked list.

If you use remove method of the List, even if it is a LinkedList or ArrayList, it takes O(n) time.

To remove all elements from a list of type Integer that are divisible by a particular value:

public static void removeDivisibleBy(LinkedList<Integer> aList, int

div){

Iterator<Integer> iter = aList.iterator();

while (iter.hasNext()) {

int nextInt = iter.next();

if (nextInt % div == 0) {

iter.remove();

}

}

}

If you do this with LinkedList.remove, it takes quadratic time.

ListIterator Interface

Iterator limitations:

* Traverses List only in the forward direction
* Provides a remove method, but no add method
* You must advance the Iterator using your own loop if you don’t start from beginning of the list
  + Iterators always start from the beginning

ListIterator extends Iterator, overcoming these limitations.

As with Iterator, ListIterator is conceptually positioned between elements of the list

ListIterator positions are assigned an index from 0 to size

Diagram

Description automatically generated with medium confidence

METHODS OF THE LISTITERATOR:

Table

Description automatically generated

Table

Description automatically generated with medium confidence

Comparison of Iterator and ListIterator

ListIterator is a subinterface of Iterator

* Classes that implement ListIterator must provide the features of both
* Can be used for LinkedList, ArrayList, or Vector

Iterator:

* Requires fewer methods
* Can iterate over more general data structures (all data structures)

Iterator is required by the Collection interface

* ListIterator is required only by the List interface

Conversion Between ListIterator and an Index

ListIterator:

* nextIndex() returns the index of item to be returned by next()
* previousIndex() returns the index of item to be returned by previous()

LinkedList has method listIterator(int index)

* Returns a ListIterator positioned so next() will return the item at position index

The listIterator(int index) method creates a new ListIterator that starts at the beginning, and walks down the list to the desired position -- generally an O(n) operation

Enhanced for Statement

Java 5.0 introduced an enhanced for statement

The enhanced for statement creates an Iterator object and implicitly calls its hasNext and next methods

Other Iterator methods, such as remove, are not available

The following code counts the number of times target occurs in myList (Type LinkedList<String>)

count = 0;

for (String nextStr : myList) {

if (target.equals(nextStr)) {

count++;

}

}

In list myList of type LinkedList<Integer>, each Integer object is automatically unboxed:

sum = 0;

for (int nextInt : myList) {

sum += nextInt;

}

The enhanced for statement also can be used with arrays, in this case, chars or type char[]

for (char nextCh : chars) {

System.out.println(nextCh);

}

Iterable Interface

Each class that implements the List interface must provide an iterator method

The Collection interface extends the Iterable interface

All classes that implement the List interface (a subinterface of Collection) must provide an iterator method

Allows use of the Java 5.0 for-each loop

public interface Iterable<E> {

/\*\* returns an iterator over the elements in this collection. \*/

Iterator<E> iterator();

}