Iterators and Sequences

Iterators

- An iterator abstracts the process of scanning through a collection of elements.
- It maintains a cursor that sits between elements in the list, or before the first or after the last element.
- Methods of the Iterator ADT:
  - `hasNext()`: returns true so long as the list is not empty and the cursor is not after the last element.
  - `next()`: returns the next element.
- Extends the concept of position by adding a traversal capability.
- Implementation with an array or singly linked list.

Iterable Classes

- An iterator is typically associated with another data structure, which can implement the Iterable ADT.
- We can augment the Stack, Queue, Vector, List and Sequence ADTs with method:
  - `Iterator<E> iterator()`: returns an iterator over the elements.
- In Java, classes with this method extend `Iterable<E>`.
- Two notions of iterator:
  - `snapshot`: freezes the contents of the data structure at a given time.
  - `dynamic`: follows changes to the data structure.
- In Java: an iterator will fail (and throw an exception) if the underlying collection changes unexpectedly.

The For-Each Loop

- Java provides a simple way of looping through the elements of an Iterable class:
  - `for (type name: expression) loop_body`
- For example:
  ```java
  List<Integer> values;
  int sum=0
  for (Integer i : values)
  sum += i;   // boxing/unboxing allows this
  ```
Implementing Iterators

- Array based
  - array A of the elements
  - index i that keeps track of the cursor
- Linked list based
  - doubly-linked list L storing the elements, with sentinels for header and trailer
  - pointer p to node containing the last element returned (or the header if this is a new iterator).
- We can add methods to our ADTs that return iterable objects, so that we can use the for-each loop on their contents

List Iterators in Java

- Java uses a the ListIterator ADT for node-based lists.
- This iterator includes the following methods:
  - `add(e)`: add e at the current cursor position
  - `hasNext()`: true if there is an element after the cursor
  - `hasPrevious`: true if there is an element before the cursor
  - `previous()`: return the element e before the cursor and move cursor to before e
  - `next()`: return the element e after the cursor and move cursor to after e
  - `set(e)`: replace the element returned by last next or previous operation with e
  - `remove():` remove the element returned by the last next or previous method

Sequence ADT

- The Sequence ADT is the union of the Array List and Node List ADTs
- Elements accessed by
  - Index, or
  - Position
- Generic methods:
  - `size()`, `isEmpty()`
- ArrayList-based methods:
  - `get(i)`, `set(i, o)`, `add(i, o)`, `remove(i)`
- List-based methods:
  - `first()`, `last()`, `prev(p)`, `next(p)`, `replace(p, o)`, `addBefore(p, o)`, `addAfter(p, o)`, `addFirst(o)`, `addLast(o)`, `remove(p)`
- Bridge methods:
  - `atIndex(i)`, `indexOf(p)`

Applications of Sequences

- The Sequence ADT is a basic, general-purpose, data structure for storing an ordered collection of elements
- Direct applications:
  - Generic replacement for stack, queue, vector, or list
  - Small database (e.g., address book)
- Indirect applications:
  - Building block of more complex data structures
Linked List Implementation

- A doubly linked list provides a reasonable implementation of the Sequence ADT.
- Nodes implement Position and store:
  - element
  - link to the previous node
  - link to the next node
- Special trailer and header nodes

Array-based Implementation

- We use a circular array storing positions.
- A position object stores:
  - Element
  - Index
- Indices $f$ and $l$ keep track of first and last positions.

Comparing Sequence Implementations

<table>
<thead>
<tr>
<th>Operation</th>
<th>Array</th>
<th>List</th>
</tr>
</thead>
<tbody>
<tr>
<td>size, isEmpty</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>atIndex, indexOf, get</td>
<td>1</td>
<td>$n$</td>
</tr>
<tr>
<td>first, last, prev, next</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>set(p,e)</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>set(i,e)</td>
<td>1</td>
<td>$n$</td>
</tr>
<tr>
<td>add, remove(i)</td>
<td>$n$</td>
<td>$n$</td>
</tr>
<tr>
<td>addFirst, addLast</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>addAfter, addBefore</td>
<td>$n$</td>
<td>1</td>
</tr>
<tr>
<td>remove(p)</td>
<td>$n$</td>
<td>1</td>
</tr>
</tbody>
</table>