5.3 Double-Ended Queues

Consider now a queue-like data structure that supports insertion and deletion at both the front and the rear of the queue. Such an extension of a queue is called a ***double-ended queue***, or ***deque***, which is usually pronounced "deck" to avoid confusion with the dequeue method of the regular queue ADT, which is pronounced like the abbreviation "D.Q."

5.3.1 The Deque Abstract Data Type

The deque abstract data type is richer than both the stack and the queue ADTs. The fundamental methods of the deque ADT are as follows:

addFirst(e): Insert a new element e at the head of the deque.

addLast(e): Insert a new element e at the tail of the deque.

removeFirst(): Remove and return the first element of the deque; an error occurs if the deque is empty.

removeLast(): Remove and return the last element of the deque; an error occurs if the deque is empty.

Additionally, the deque ADT may also include the following support methods:

getFirst(): Return the first element of the deque; an error occurs if the deque is empty.

getLast(): Return the last element of the deque; an error occurs if the deque is empty.

size(): Return the number of elements of the deque.

isEmpty(): Determine if the deque is empty.

Example 5.5: *The following table shows a series of operations and their effects on an initially empty deque D of integer objects. For simplicity, we use integers instead of integer objects as arguments of the operations.*

Operation	***Output***	D
addFirst(3)	–	(3)
addFirst(5)	–	(5,3)
removeFirst()	5	(3)
addLast(7)	–	(3,7)
removeFirst()	3	(7)
removeLast()	7	()
removeFirst()	*"error"*	()
isEmpty()	*true*	()

5.3.2 Implementing a Deque

Since the deque requires insertion and removal at both ends of a list, using a singly linked list to implement a deque would be inefficient. We can use a doubly linked list, however, to implement a deque efficiently.

As discussed in Section 3.3, inserting or removing elements at either end of a doubly linked list is straightforward to do in $O(1)$ time, if we use sentinel nodes for the header and trailer, which is an implementation we suppo~~rt.~~

For an insertion of a new element e, we can have access to the node p before the place e should go and the node q after the place e should go. To insert a new element between the two nodes p and q (either or both of which could be sentinels), we create a new node t, have t's prev and next links respectively refer to p and q, and then have p's next link refer to t, and have q's prev link refer to t.

Likewise, to remove an element stored at a node t, we can access the nodes p and q on either side of t (and these nodes must exist, since we are using sentinels). To remove node t between nodes p and q, we simply have p and q point to each other instead of t. We need not change any of the fields in t, for now t can be reclaimed by the garbage collector, since no one is pointing to t.

Performance and Linked List Implementation Details

Table 5.4 shows the running times of methods for a deque implemented with a doubly linked list. Note that every method runs in $O(1)$ time.

Method	Time
size, isEmpty	$O(1)$
getFirst, getLast	$O(1)$
addFirst, addLast	$O(1)$
removeFirst, removeLast	$O(1)$

Table 5.4: Performance of a deque realized by a doubly linked list.

Thus, a doubly linked list can be used to implement each method of the deque ADT in constant time. We leave the complete details of implementing the deque ADT efficiently in Java as an exercise (see P-5.3). Nevertheless, we show a Deque interface in Code Fragment 5.17 and a partial implementation of this interface in Code Fragment 5.18.

```
/**
 * Interface for a deque: a collection of objects that are inserted
 * and removed at both ends; a subset of java.util.LinkedList methods.
 *
 * @author Roberto Tamassia
 * @author Michael Goodrich
 */

public interface Deque<E> {
 /**
  * Returns the number of elements in the deque.
  */
  public int size();
 /**
  * Returns whether the deque is empty.
  */
  public boolean isEmpty();
 /**
  * Returns the first element; an exception is thrown if deque is empty.
  */
  public E getFirst() throws EmptyDequeException;
 /**
  * Returns the last element; an exception is thrown if deque is empty.
  */
  public E getLast() throws EmptyDequeException;
 /**
  * Inserts an element to be the first in the deque.
  */
  public void addFirst (E element);
 /**
  * Inserts an element to be the last in the deque.
  */
  public void addLast (E element);
 /**
  * Removes the first element; an exception is thrown if deque is empty.
  */
  public E removeFirst() throws EmptyDequeException;
 /**
  * Removes the last element; an exception is thrown if deque is empty.
  */
  public E removeLast() throws EmptyDequeException;
}
```

Code Fragment 5.17: Interface Deque documented with comments in Javadoc style (Section 1.9.3). Note also the use of the generic parameterized type, E, which implies that a deque can contain elements of any specified class.

```
public class NodeDeque<E> implements Deque<E> {
  protected DLNode<E> header, trailer; // sentinels
  protected int size;    // number of elements
  public NodeDeque() { // initialize an empty deque
    header = new DLNode<E>();
    trailer = new DLNode<E>();
    header.setNext(trailer);   // make header point to trailer
    trailer.setPrev(header);   // make trailer point to header
    size = 0;
  }
  public int size() {
    return size;
  }
  public boolean isEmpty() {
    if (size == 0)
      return true;
    return false;
  }
  public E getFirst() throws EmptyDequeException {
    if (isEmpty())
      throw new EmptyDequeException("Deque is empty.");
    return header.getNext().getElement();
  }
  public void addFirst(E o) {
    DLNode<E> second = header.getNext();
    DLNode<E> first = new DLNode<E>(o, header, second);
    second.setPrev(first);
    header.setNext(first);
    size++;
  }
  public E removeLast() throws EmptyDequeException {
    if (isEmpty())
      throw new EmptyDequeException("Deque is empty.");
    DLNode<E> last = trailer.getPrev();
    E o = last.getElement();
    DLNode<E> secondtolast = last.getPrev();
    trailer.setPrev(secondtolast);
    secondtolast.setNext(trailer);
    size--;
    return o;
  }
}
```

Code Fragment 5.18: Class NodeDeque implementing the Deque interface, except that we have not shown the class DLNode, which is a generic doubly linked list node, nor have we shown methods getLast, addLast, and removeFirst.