Chapter 12 The Deque

A *deque* (pronounced "deck") is a double-ended queue. In this chapter we will implement a deque container by extending class `Queue`, and we will discuss common deque applications.

12.1 Our Current Model

Our current class diagram appears below as Figure 12.1. Our doubly linked implementation of class `Queue` lends itself to the specialization shown in the diagram.

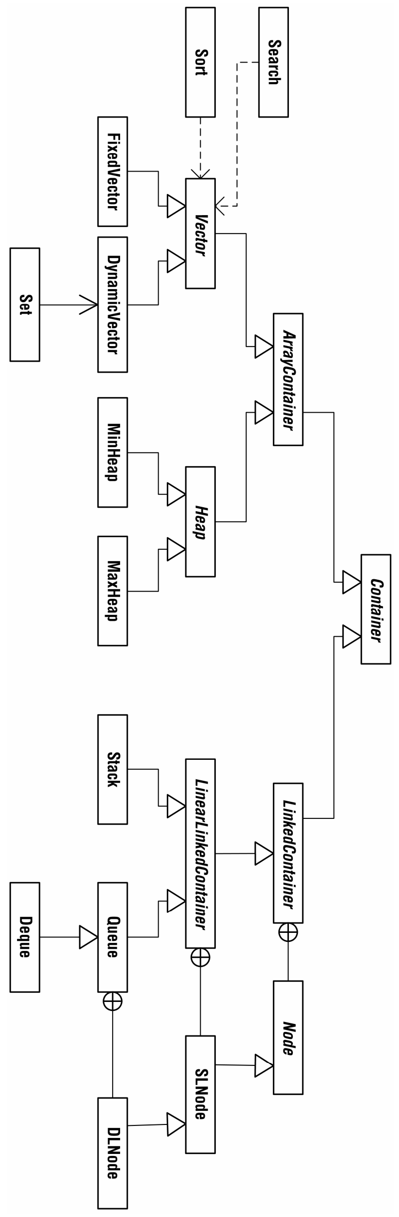

Fig. 12.1. Our current software model, in which class `Deque` specializes class `Queue`

12.2 The Deque ADT

The deque ADT is shown below. Observe that a deque is a queue that also supports `insertFront`, `removeBack`, and `back` operations.

```
deque: a double-ended queue

operations:

                back() – Get the back element of the collection.
               clear() - Make the collection empty.
               front() - Get the front element of the collection.
   insertBack(element) - Make the given element the back element.
  insertFront(element) – Make the given element the front element.
             isEmpty() - Is the collection empty?
          removeBack() – Remove the back element.
         removeFront() - Remove the front element.
                size() - How many elements are in the collection?
```

12.3 Deque Implementation

We implement the `Deque` container by extending the `Queue` container. Class `Queue`'s doubly-linked list offers significant advantages over a singly-linked implementation.

```
public class Deque extends Queue
{
    public Object back()
    {
        if (isEmpty())
            return null;
        return tail.data;
    }

    public void insertFront(Object element)
    {
        // If the container is empty, inserting at the front is
        // equivalent to inserting at the back.

        if (isEmpty())
        {
            insertBack(element);
            numItems++;
            return;
        }
        head = new DLNode(element, null, (DLNode)head);
        if (head.next != null)
           ((DLNode)head.next).prev = (DLNode)head;
        numItems++;
    }
    public Object removeBack()
    {
        Object temp = back();
        if (temp == null)
            return null;
        tail = tail.prev;
        if (tail == null)
            head = null;
        else
        {
            ((DLNode)tail.next).prev = null;
            tail.next = null;
        }
        numItems--;
        return temp;
    }
}
```

12.4 Deque Applications

A nice application of the deque is storing a web browser's history. Recently visited URLs are added to the front of the deque, and the URL at the back of the deque is removed after some specified number of insertions at the front. Another common application of the deque is storing a software application's list of undo operations.

Exercises

1. Add attributes and operations to the class diagram from Figure 12.1.
2. Perform a time analysis of the deque operations as we implemented them and also for a singly-linked implementation. Conclude that the doubly-linked implementation is more efficient with respect to running time. How could a singly-linked deque be implemented so that its running times were the same as for the doubly-linked implementation? How much space would be saved by a singly-linked implementation?