```
1      /**
2       * Make the queue logically empty.
3       */
4      public void makeEmpty( )
5      {
6          currentSize = 0;
7          front = 0; back = theArray.length - 1;
8      }
```

Figure 15.15 `makeEmpty` routine for array-based `Queue` class

15.2 Linked-list Implementations

The advantage of a linked-list implementation is that the excess memory is only one reference per item. The disadvantage is that the memory allocation is time consuming.

An alternative to the contiguous array implementation is to use a linked list. Recall from Section 6.4 that in a linked list, we store each item in a separate object that also contains a reference to the next object in the list.

The advantage of the linked list is that the excess memory is only one reference per item, whereas a contiguous array implementation uses excess space equal to the number of vacant array items (plus some additional memory during the doubling phase). This advantage is immaterial in Java because the vacant array items represent null references and thus consume little space. In other languages, however, this advantage is significant because the vacant array items tend to store uninitialized instances of objects that can consume significant space. Even so, the linked-list implementations are discussed for several reasons:

1. It is important to understand implementations that might be useful in other languages.
2. Implementations that use linked lists are shorter than the comparable array versions, especially for the queue.
3. These implementations illustrate the principles behind the more general linked list operations given in Chapter 16.

For the implementation to be competitive with contiguous array implementations, we must be able to perform the basic linked-list operations in constant time. This is easy to do because the changes in the linked list are restricted to the elements at the two ends (front and back) of the list.

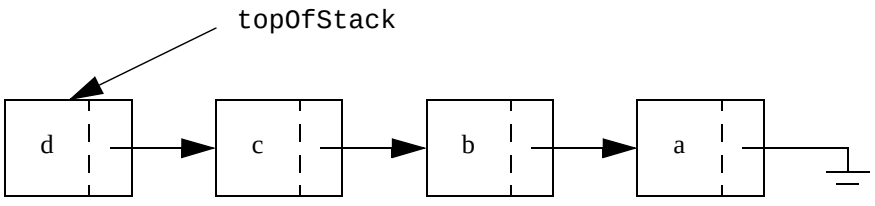

Figure 15.16 Linked-list implementation of the stack