Linked Lists

Last updated: November 4th 2018, at 3.47pm

1 This Week's Code

This week's code bundle contains two packages, each of which, in turn, contains two more packages.

The `arrayGenerator` package contains the array generator code from previous weeks, though arranged slightly differently. The array generators themselves are in the `generator` packages, while the various `Scope` classes are defined in the `scope` package. It is not necessary to understand the implementation of these packages in order to undertake this week's exercises. They are provided solely as a tool for generating arrays of (constrained) random values that can be used as test data.

It is the `linkedList` package that is relevant to the exercises. This package implements some common linked list structures, and contains two further packages. The `node` package defines and implements nodes that can be used to construct linked lists, while the `list` package defines and implements some linked list data structures.

`node` Implements single and double lined nodes.

- **interface** `ListNode` A list node is a node used to construct a list. A list node will contain, at a minimum, a value, and a link to the next node in the list.
- **class** `SingleLinkNode` Implements the minimal functionality defined in `ListNode`, plus a method for changing the next node pointed to.
- **class** `DoubleLinkNode` Allows for an additional link in the node, pointing to the previous node in the list.

`list` Implements various linked list data structures.

- **class** `ListAccessError` An exception class for reporting errors in accessing values and nodes in lists.
- **interface** `LinkedList` The basic interface for all linked list classes. The only method the interface requires is a test for "emptiness".

`keywordclass BasicList` A basic implementation of a list. No methods are provided for adding or removing values/nodes, but `isEmpty` *is* implemented, as are methods for setting and accessing the root node (the first node in the list). A `toString` method is also implemented.

interface `Stack` Specifies the methods required of a stach data structure — `push` and `pop`.

class `SingleLinkStack` Implements stacks using single linked nodes.

interface `Queue` Specifies the methods required of a queue data structure — `enqueue` and `dequeue`.

class `DoubleLinkQueue` Implements queues using double link nodes.

interface `List` Specifies the methods required of a list data structure — `add`, `remove`, and `get`. No implementation of this interface is provided.

2 This Week's Exercises

1. Logbook exercise Implement the `List` ⟨`T`⟩ interface, using singly linked lists.

   Note: an implementation of the `get(int index)` metthod will be provided as a model exercise answer, but do not wait for this before attempting your own solution.

2. Then write some test code that uses array generators to create (large) random arrays. Use the values in these arrays to populate instances of your implementation of linked lists. Now attempt multiple accesses of the data both in the arrays and in the lists — e.g.

   ```
   value = array[index];
   ```

   and

   ```
   value = list.get(index);
   ```

   and time the time taken for these accesses. Make sure that you try this for large values of `index`. Compare the times taken for arrays and lists.