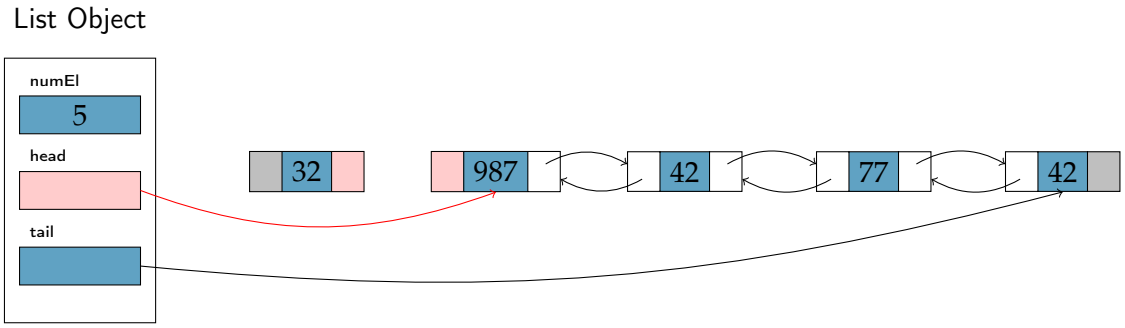
Lecture 05 Exercise Solutions

Mark Eramian

Exercise 1

After deleting the first node, the data structure will look like this:

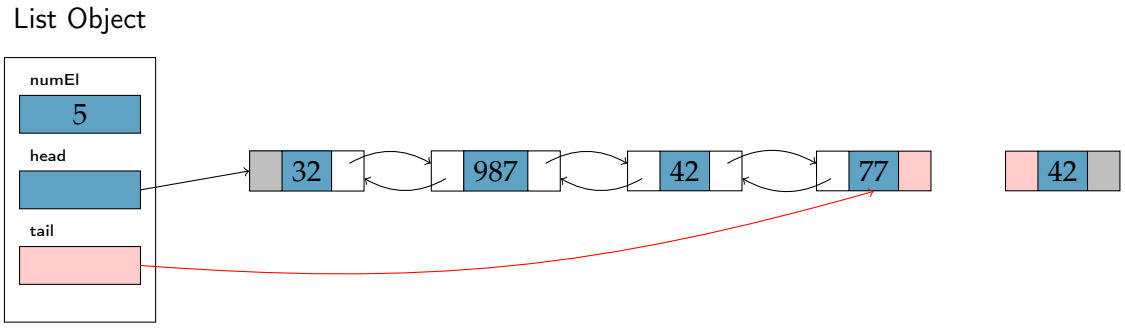


Red shaded references and red arrows indicate references that were modified in the deletion process.

The cursor must be updated if the cursor is on the first or second element. If the cursor is on the second element, the `prevPosition` location must be updated to `null`. If the cursor is on the first element, the cursor must either be moved to the second element or placed in the "before" position.

Exercise 2

After deleting the last node, the data structure will look like this:

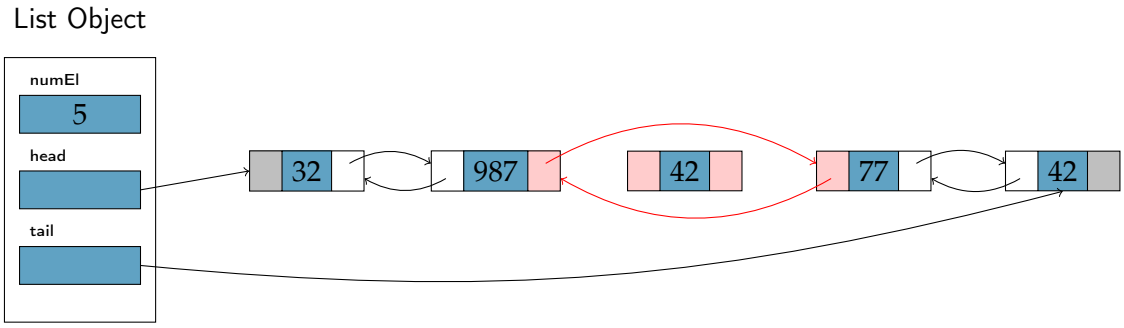


Red shaded references and red arrows indicate references that were modified in the deletion process.

The cursor must be updated if it is in the "after" position or on the last element. If it is in the "after" position, `prevPosition` must be updated to refer to the new tail. If it is positioned on the last element, it must be either moved to the previous element or placed in the "after" position.

Exercise 3

After deleting the middle node (42), the data structure will look like this:



Red shaded references and red arrows indicate references that were modified in the deletion process.

The cursor must be updated if it is positioned on the item being deleted or its successor. If it is positioned on the successor, the `prevPosition` must be updated to refer to the deleted node's predecessor. If the cursor is on the node being deleted, it must be moved to either the node's successor or predecessor (both must exist in this scenario or the node being deleted would be either the first node or the last node).

Excercises 4-6

The solutions to exercises 4 through 6 can be found in the accompanying Java files.

Exercise 7

We should extend the `Cursor` interface to a `BilinkedCursor` interface in which we define the methods that can only be used if we have a bi-linked node chain (e.g. `goBack()`). In addition we must adjust the header of `BilinkedLIst` by declaring that it implements the `BilinkedCursor` interface.

Why would we do this just for the sake of adding one method? It is consistent with the idea that we should have a common interface for a cursors on linear data structures that we can re-use for other linear data structures, for example, our `ArrayedList`. We could then go back and have `ArrayedList` implement the `BilinkedCursor` interface instead of the `Cursor` interface, thus extending and improving its functionality with a minimum of effort.