Advanced Programming

List Implementations
That Link Data

(Part II)
Completing the Linked Implementation of a List
Contents

- Removing an item from a linked list
- Completing the linked implementation of the ADT list
  - Method `remove`
  - Method `replace`
  - Method `getEntry`
  - Using a Class Node with Set and Get Methods
- Tail References
  - Revision of List
- Pros and Cons of Using Chain
- Java Class Library: LinkedList
Removing an Item from a Linked List

Possible cases:
1. Remove from beginning of the list
2. Remove from somewhere else
The Method \textit{remove}

```java
firstNode = firstNode.next
```

Fig. 6-22 A chain of nodes (a) prior to removing first node; (b) after removing the first node
The Method remove

Fig. 6-23 A chain of nodes (a) prior to removing interior node; (b) after removing interior node
Node nodeBefore = getNodeAt(givenPosition - 1);
Node nodeToRemove = nodeBefore.next;
Node nodeAfter = nodeToRemove.next;
nodeBefore.next = nodeAfter;
nodeToRemove = null;
Further Methods of **LList**

- **Method** `remove`
  - Note how cases are handled
- **Method** `replace`
- **Method** `getEntry`
- **Method** `contains`
- **Method** `isFull`
  - Always returns `false` in this context
Using Class **Node** that Has **Set** and **Get** Methods

- **View new version** of class **Node**
  - An inner class
  - The class **LList** can access private data fields directly
  - Stylistically better to use the **Set** and **Get** methods of the class **Node**

- Thus better to use statements such as:
  ```java
  currentNode.getData();  // or
  desiredNode.setData(newEntry);
  ```
Tail References

- Consider a set of data where we repeatedly add data to the end of the list
- Each time the `getNodeAt` method must traverse the whole list
  - This is inefficient
- Solution: maintain a pointer that always keeps track of the end of the chain
  - The tail reference
Tail References

Fig. 6-24 A linked chain with a head and tail reference.
public class LList<T> implements ListInterface<T> {
    private Node firstNode; // reference to first node
    private Node lastNode; // reference to last node
    private int length; // number of list entries

    ...
}

Needed changes to:
    both add methods
    remove
    clear
Tail References

- Must change the clear method
  - Constructor calls it

```java
public final void clear ()
{
    firstNode = null;
    lastNode = null;
    length = 0;
} // end clear
```
Tail References

- **When adding to an empty list**
  - Both head and tail references must point to the new solitary node

- **When adding to a non empty list**
  - No more need to traverse to the end of the list
  - `lastNode` points to it
  - Adjust `lastNode.next` in new node and `lastNode`
Tail References

(a)

Fig. 6-25 Adding a node to the end of a nonempty chain that has a tail reference

(b)
Tail References

Fig. 6-26 Removing a node from a chain that has both head and tail references when the chain contains (a) one node; (b) more than one node
Pros and Cons of a Chain for an ADT List

- The chain (list) can grow as large as necessary
- Can add and remove nodes without shifting existing entries

But …

- Must traverse a chain to determine where to make addition/deletion
- Retrieving an entry requires traversal
  - As opposed to direct access in an array
Java Class Library: The Class LinkedList

- The standard java package java.util contains the class LinkedList
- This class implements the interface List
- Contains additional methods
  - addFirst()
  - addLast()
  - removeFirst()
  - removeLast()
  - getFirst()
  - getLast()
Literature

• Chapter 7
  • Data Structures and Abstractions with Java, by Carrano, Second Edition.