COM498 Algorithms & Data Structures

3.2 Linked Implementation of the Bag

Specifying a Bag (summary)

- A reminder list of our method signatures for the Bag ADT:

  - `int getCurrentSize()`
  - `boolean isEmpty`
  - `boolean addNewEntry(T newEntry)`
  - `T remove()`
  - `boolean remove(T anEntry)`
  - `void clear()`
  - `int getFrequencyOf(T anEntry)`
  - `boolean contains(T anEntry)`
  - `T[] toArray`

- We have implemented these with the Bag organized as an array – let's do the same for a linked list implementation

Partial Outline of Class `LinkedBag`

- The implementation of Bag will be as a chain of linked nodes (each node contains an entry in the bag)
- Implementation must have a data value (known as a head reference) to record the address of the first node in the chain
- Implementation must also contain a data value to track the number of entries stored in the bag (number of nodes in the chain)

Partial Outline of Class `LinkedBag`

- In Java…

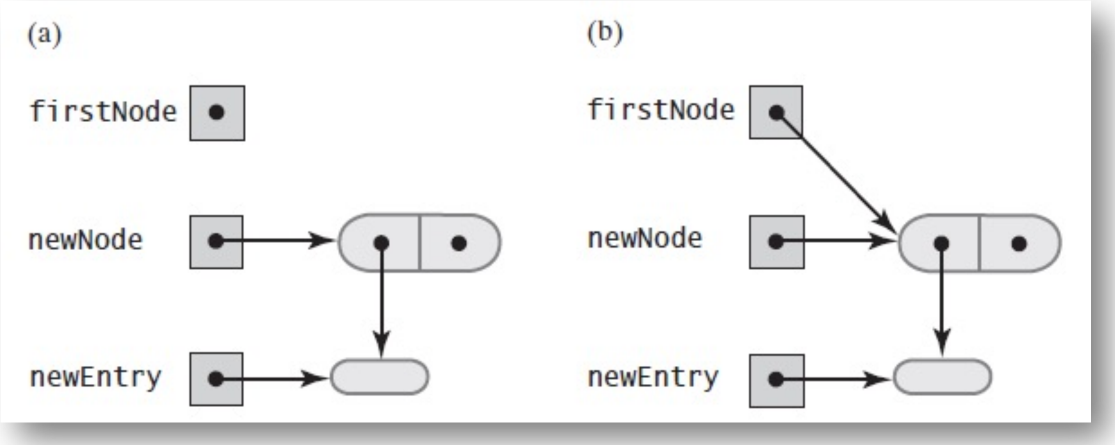
```
public final class LinkedBag<T> implements BagInterface<T> {
   private MyNode<T> firstNode;
   private int numberOfEntries;

   public LinkedBag() {
      firstNode = null;
      numberOfEntries = 0;
   }
}
```

Beginning a Chain of Nodes

- The `addNewEntry()` method is one of our (previously established) core methods and must add the first entry to an empty **Bag**:

```
MyNode newNode = new MyNode(newEntry);
firstNode = newNode;
```



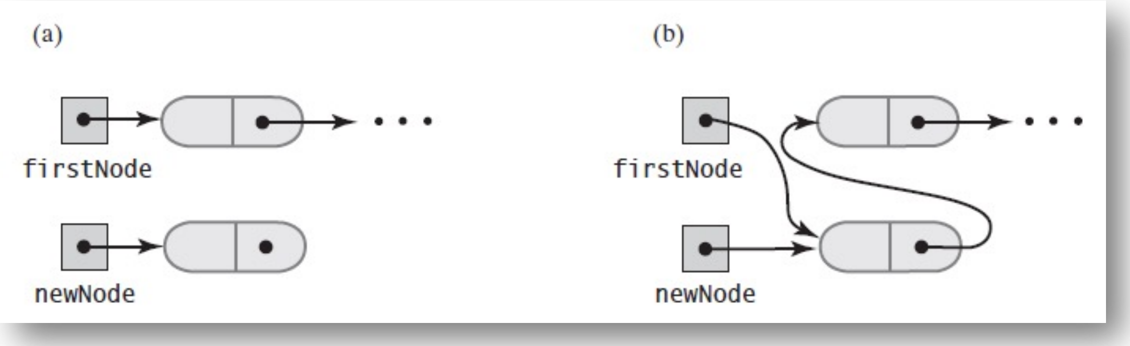
a) An empty chain and a new Node

b) After adding the new Node to the empty chain

Adding to a Chain of Nodes

- Method addNewEntry() will add new nodes to the beginning of the chain
- The new node becomes the first node in the chain

```
MyNode newNode = new MyNode(newEntry);
newNode.next = firstNode;
firstNode = newNode;
```



a) Prior to adding newNode at the beginning of the list

b) After adding newNode to the beginning of the list

LinkedBag `addNewEntry()` Method

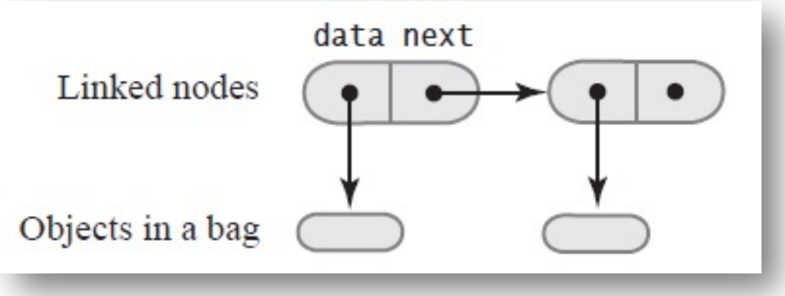
- Adding a node to an empty chain is actually the same as adding a node to the beginning of the chain (the new node becomes the first node

```
public boolean addNewNode(T newEntry) {
   MyNode<T> newNode = new MyNode<T>(newEntry);
   newNode.setNext(firstNode);
   firstNode = newNode;
   numberOfEntries++;
   return true;
}
```

- Any time a new node is added, the operation is successful
- If you use all of the computer’s memory, you will receive an OutOfMemoryError

Traversal of a Linked Chain

- Another core method `toArrary()` lets us test that `addNewEntry()` works
- To access a bag's entries we need to access each node in the chain beginning with the first node, a process known as traversal
- Each node contains a reference to the next node in the linked chain
- In method `toArrary()` a temporary local variable `currentNode` is needed to reference each node in turn



- Initially `currentNode` will refence the first node so it is set to `firstNode`
- After accessing the data by `currentNode.getData()` the next node is obtained using `currentNode = currentNode.getNext();`
- This process continues until `currentNode` becomes `null` (last node in chain)

LinkedBag `toArray()` Method

- Traversing the linked chain to generate the array to return an array

```
public T[] toArray() {
   T[] result = (T[]) new Object[numberOfEntries];
   int index = 0;
   MyNode<T> currentNode = firstNode;
   while (currentNode != null) {
      result[index++] = currentNode.getData();
      currentNode = currentNode.getNext();
   }
   return result;
}
```

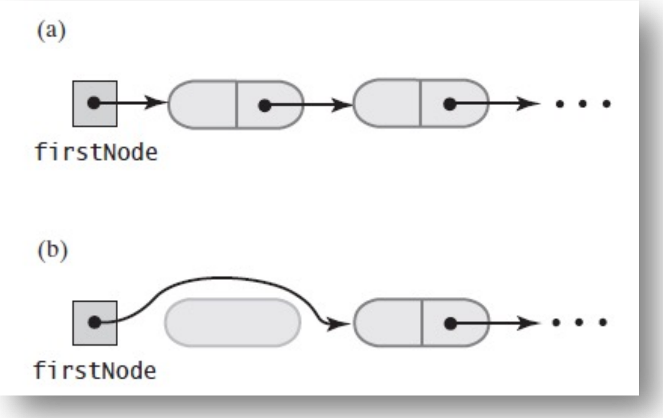
Retrieve the data from currentNode

Move to next node in the chain

- Need to ensure the `currentNode` reference is <u>not</u> null before using it to access the data or getting the next node, otherwise a **NullPointerException** occurs!

Removing an Item from a Linked Chain

- Recall a bag doesn’t order its items in any particular way
- One of the `remove()` methods is to remove an unspecified entry - since the first node is the easiest to remove from a linked chain we will use this approach in this case



a) Prior to removing the first node

b) Just after removing the first node

```
Algorithm removeFirstElement ()
// Remove and return the first element from a
// linked chain

if firstNode is not null
   set result = data field of firstNode
   set firstNode to the next field of firstNode
   decrement number of entries and return result
else return null
```

What if the first node is the only node in the chain?

Removing a Specified Item from a Linked Chain

- Second `remove()` method removes a specified entry, so we need to first traverse the chain to return a pointer to that node
- Suppose we find the desired entry in node *N*, we will have one of 2 possible situations:

A. Node N is the first node in the chain:

A. Remove the first node from the chain

B. Node N is not the first node in the chain:

1) Replace the entry in Node *N* with the entry in the first node
2) Remove the first node from the chain

- Its easier to apply B (above) to all situations than to add logic to determine if *N* is the first node in the chain

Finding an Element in a Chain

- Traversing the linked chain to return a pointer to a specific entry

```
private MyNode findEntry(T entry) {
   MyNode currentNode = firstNode;
   boolean found = false;
   while (!found && currentNode != null) {
      if (currentNode.getData().equals(entry))
         found = true;
      else currentNode = currentNode.getNext();
   }
   if (found) return currentNode;
   else return null;
}
```

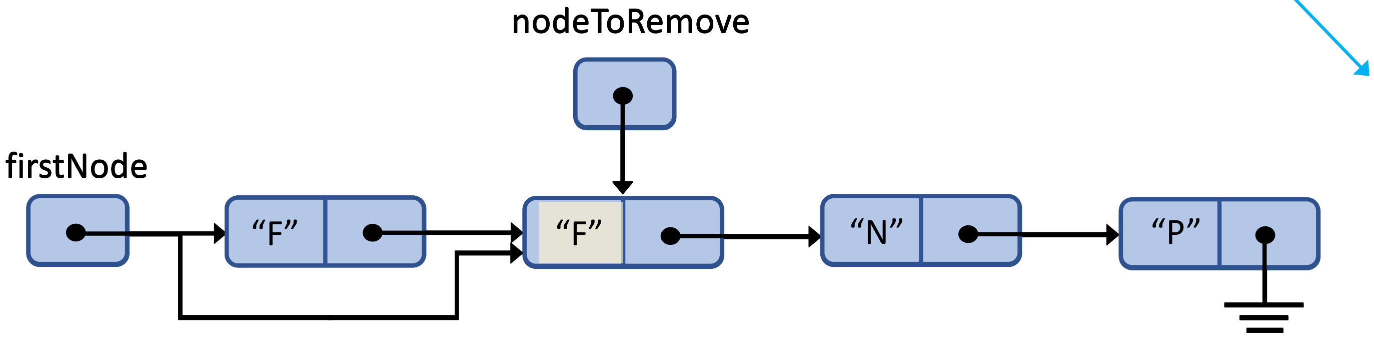
Removing a Specific Element

```
public boolean remove(T entry) {
    MyNode nodeToRemove = findEntry(entry);
    if (nodeToRemove == null) return false;
    nodeToRemove.setData(firstNode.getData());
    firstNode = firstNode.getNext();
    numberOfEntries--;
    return true;
}
```

Find the entry to remove

Replace the data field of the node to remove with the data field from the first element

Eliminate the first element



Specifying a Bag (summary)

- Our methods for the Bag ADT:

```
• int getCurrentSize()
• boolean isEmpty
• boolean addNewEntry(T newEntry)
• T remove()
• boolean remove(T anEntry)
• void clear()
• int getFrequencyOf(T anEntry)
• boolean contains(T anEntry)
• T[] toArray
```

- The shaded methods have now been implemented as a linked chain

- Those remaining either do not need to change i.e. `getCurrentSize()`, `isEmpty()`, `clear()` - or are very easily implemented using the list traversal technique i.e. `getFrequencyOf()`,`contains()`

Scenario

- In your **Bag** project, create the file *LinkedBag.java* and implement the class `LinkedBag` to implement the `BagInterface` class, providing all public methods that have been previously provided by `ArrayBag`.
- Update the `BagTest` class to perform the same tests on an instance of `LinkedBag` and trace through the diagnostic messages returned to check the success of your implementation.

Pros and Cons of Using a Chain

- Pros:
  - Bag can grow and shrink in size as necessary
  - Remove and recycle nodes that are no longer needed
  - Adding to the beginning of the chain is equally as simple as adding to the end of an array
  - Removing from the beginning of the chain is equally as simple as removing from the end of an array

- Cons:
  - Chain requires more memory than array of same length
  - Removing specific entry requires search of the chain (similar to array)