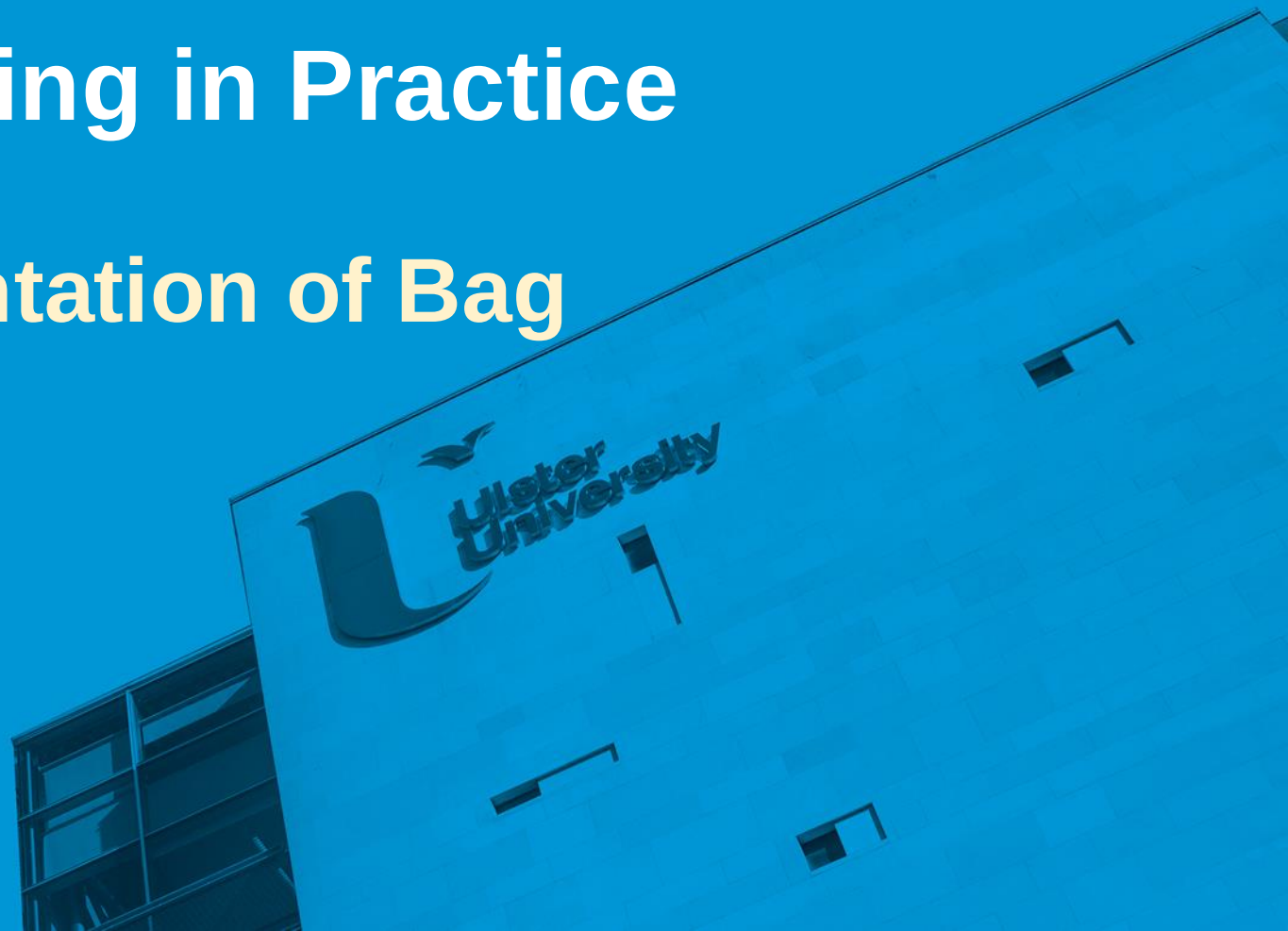
Ulster University

COM410 Programming in Practice

A4.2 Linked Implementation of Bag



Specifying a Bag (summary)

- A reminder list of our method signatures for the Bag ADT (using the Generic data type `T`):

```
• int getCurrentSize()
• boolean isEmpty()
• boolean addNewEntry(T newEntry)
• T remove()
• boolean remove(T anEntry)
• void clear()
• int getFrequencyOf(T anEntry)
• boolean contains(T anEntry)
• String toString()
```

- 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 record the address of the first node in the chain (known as a head reference)
- Implementation should also maintain an instance variable to track the number of entries stored in the bag (number of nodes in the chain)

Partial Outline of Class `LinkedBag`

```
public class LinkedBag implements BagInterface {
    private Node firstNode;
    private int numberOfEntries;

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

    // plus method skeletons methods for the
    // BagInterface
}
```

- Initial state of the linked list

Scenario

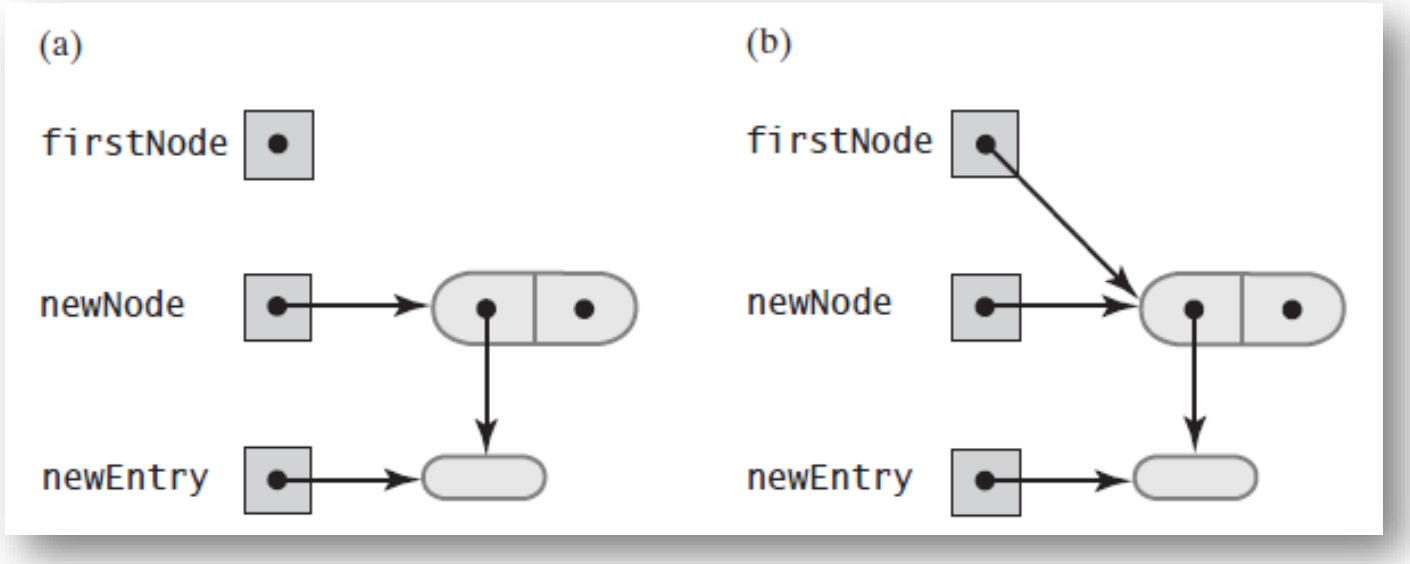
- Implement a skeleton for the class `LinkedBag` in the new file ***LinkedBag.java*** within your **Anytown** project.

    - Provide the `LinkedBag` class header to implement the `BagInterface` previously defined and populate it with the definition of the class and instance variables, the constructor and empty methods for each of the public methods defined in the interface class

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**:

```
Node newNode = new Node(newEntry);
this.firstNode = newNode;
```



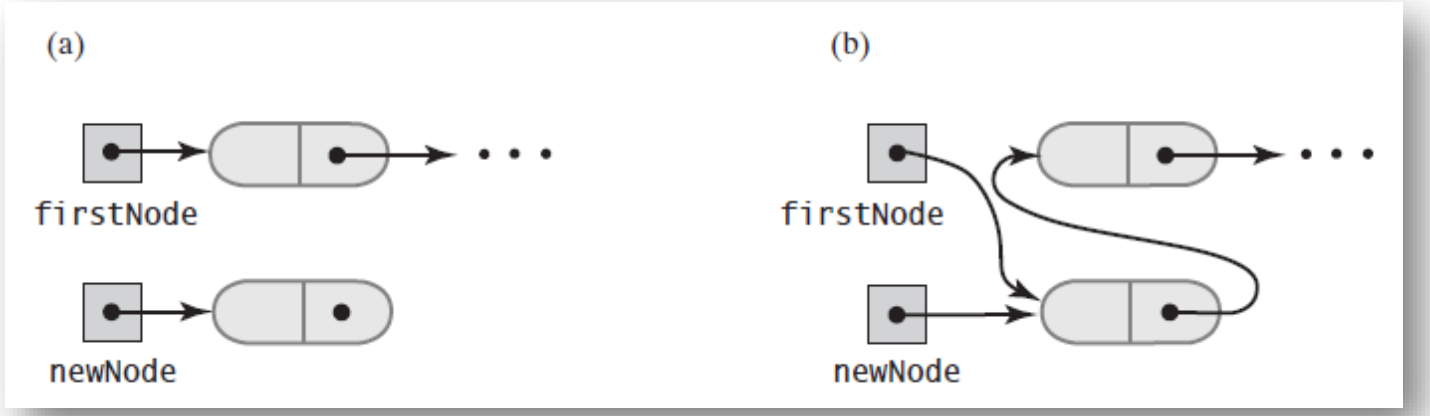
a) An empty chain and a new Node

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

```
Node newNode = new Node(newEntry);
newNode.next = this.firstNode;
this.firstNode = newNode;
```

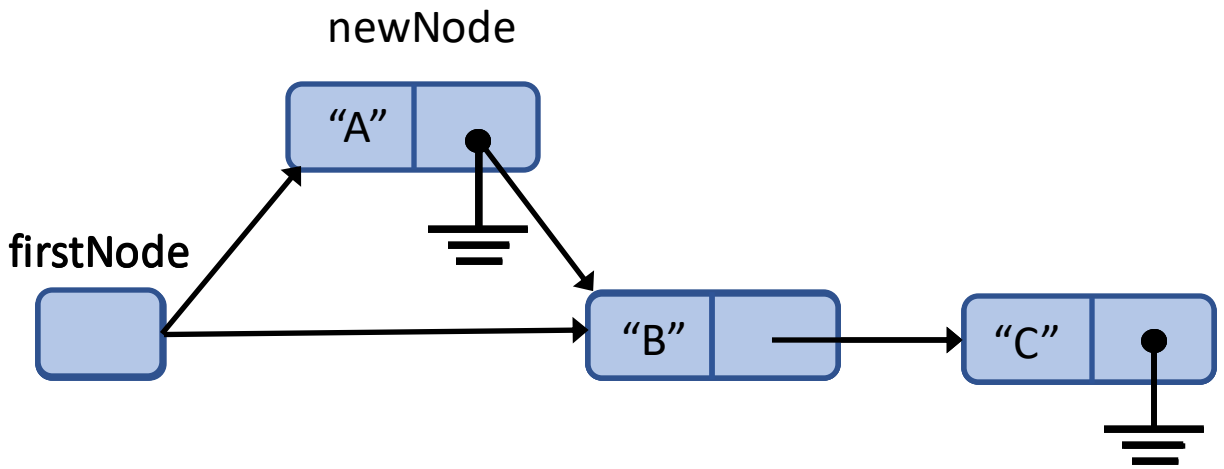


a) Prior to adding newNode at the beginning of the list

b) After adding newNode to the beginning of the list

Adding to a Chain of Nodes

```
Node newNode = new Node(newEntry);
newNode.setNext(this.firstNode);
this.firstNode = newNode;
```



LinkedBag addNewEntry() Method

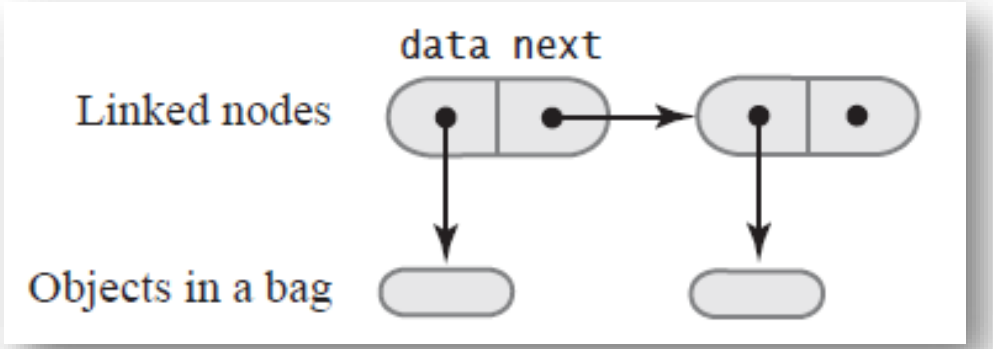
- Adding a node to an empty chain is actually the same as adding a node to the beginning of the chain (the new node becomes the first node)
- Assuming a collection of `Building` objects

```
public boolean addNewEntry(Building newEntry) {
    Node newNode = new Node(newEntry);
    newNode.setNext(this.firstNode);
    this.firstNode = newNode;
    this.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

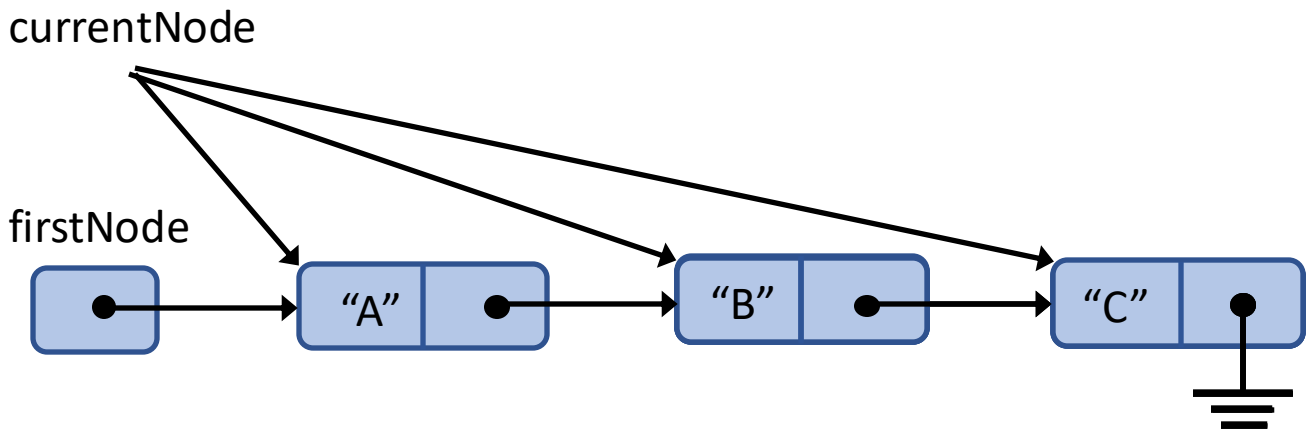
- Another core method `toString()` lets us test that `addNewEntry()` works
- To access a bag's entries we need to access each node in the chain beginning with the first node, a process known as traversal
- Each node contains a reference to the next node in the linked chain
- In method `toString()` a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

Traversing a Chain of Nodes

```
Node currentNode = this.firstNode;
while (currentNode != null) {
      data = currentNode.getData();
      currentNode = currentNode.getNext();
}
```



LinkedBag `toString()` Method

- Traversing the linked chain to return a string representation

```
public String toString() {
    String resultStr = "Bag[ \n";
    Node currentNode = this.firstNode;
    while (currentNode != null) {
        resultStr += currentNode.getData() + "\n";
        currentNode = currentNode.getNext();
    }
    resultStr += "] \n";
    return resultStr;
}
```

Retrieve the data from currentNode

Move to next node in the chain

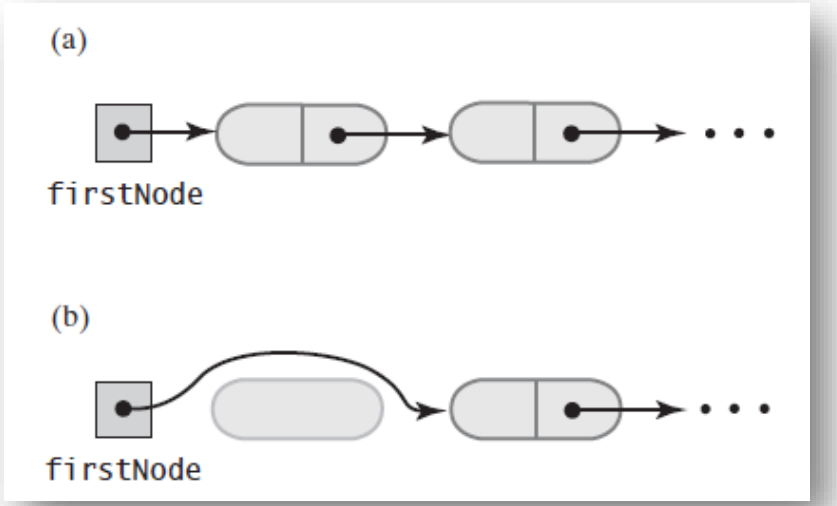
- Need to ensure the **`currentNode`** reference is not **`null`** before using it to access the data or getting the next node, otherwise a **NullPointerException** occurs!

Scenario

- Add the core methods `addNewEntry()` and `toString()` to your `LinkedBag` class.
- Create the class `LinkedBagTest` and copy the code from the `main()` method in your existing `ArrayBagTest`
    i. In the `main()` method, replace the existing `ArrayBag` collection with a `LinkedBag`.
    ii. Run the `main()` method of `LinkedBagTest` and verify that all elements are successfully added to the bag. Note that the bag never becomes full and that elements are presented in the reverse order from entry – i.e. new elements are entered at the front of the list and the list is reported from front to back.

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 <u>is</u> the first node in the chain:

A. Remove the first node from the chain

B. Node N <u>is not</u> 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 (more efficient) 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 Node findEntry(Building entry) {
    Node currentNode = this.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;
}
```

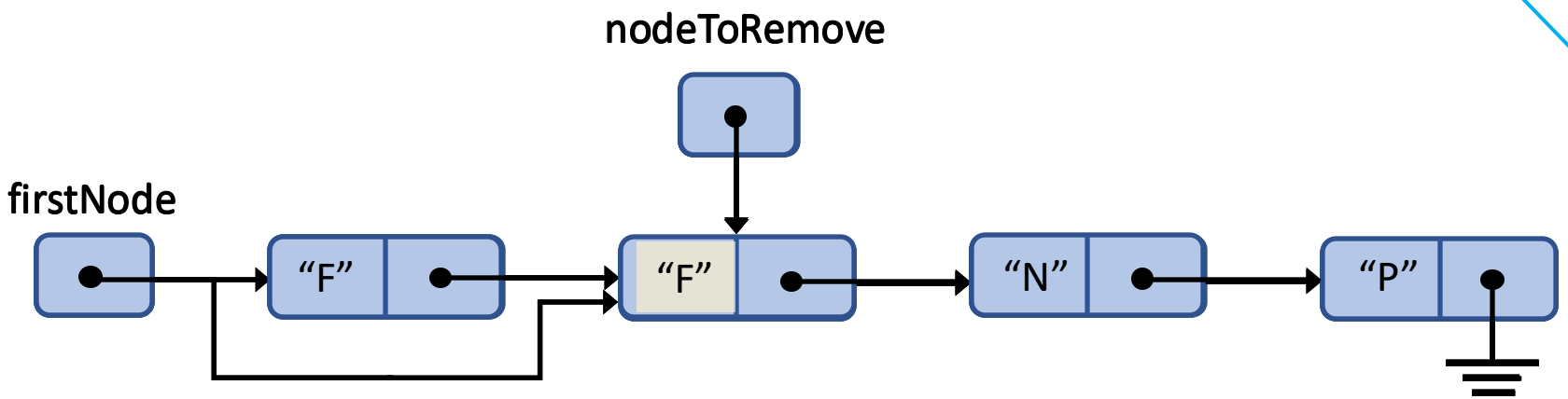
Removing a Specific Element

```
public boolean remove(Building entry) {
    Node nodeToRemove = findEntry(entry);
    if (nodeToRemove == null) return false;
    nodeToRemove.setData(this.firstNode.getData());
    firstNode = this.firstNode.getNext();
    this.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 (assuming the Generic type `T`):

```
• int getCurrentSize()
• boolean isEmpty()
• boolean addNewEntry(T newEntry)
• T remove()
• boolean remove(T anEntry)
• void clear()
• int getFrequencyOf(T anEntry)
• boolean contains(T anEntry)
• String toString()
```

- 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

- Add the remaining methods to your `LinkedBag` class. All methods outlined in the interface class should now be implemented. Now verify the operation of the `LinkedBag` class by the following
    - Modify the file ***BagOfBuildingsTest.java*** to operate on an instance of a `LinkedBag`
    - Run the `main()` method in ***BagOfBuildingsTest.java*** and trace the diagnostic comments provided to ensure that your `LinkedBag` implementation is working as expected

Pros and Cons of Using a Chain

- Pros:
  - Bag can grow and shrink in size as necessary
  - Ability to 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)