Implementation
LinkedList

COMP128 Data Structures

Linked Structures: Node

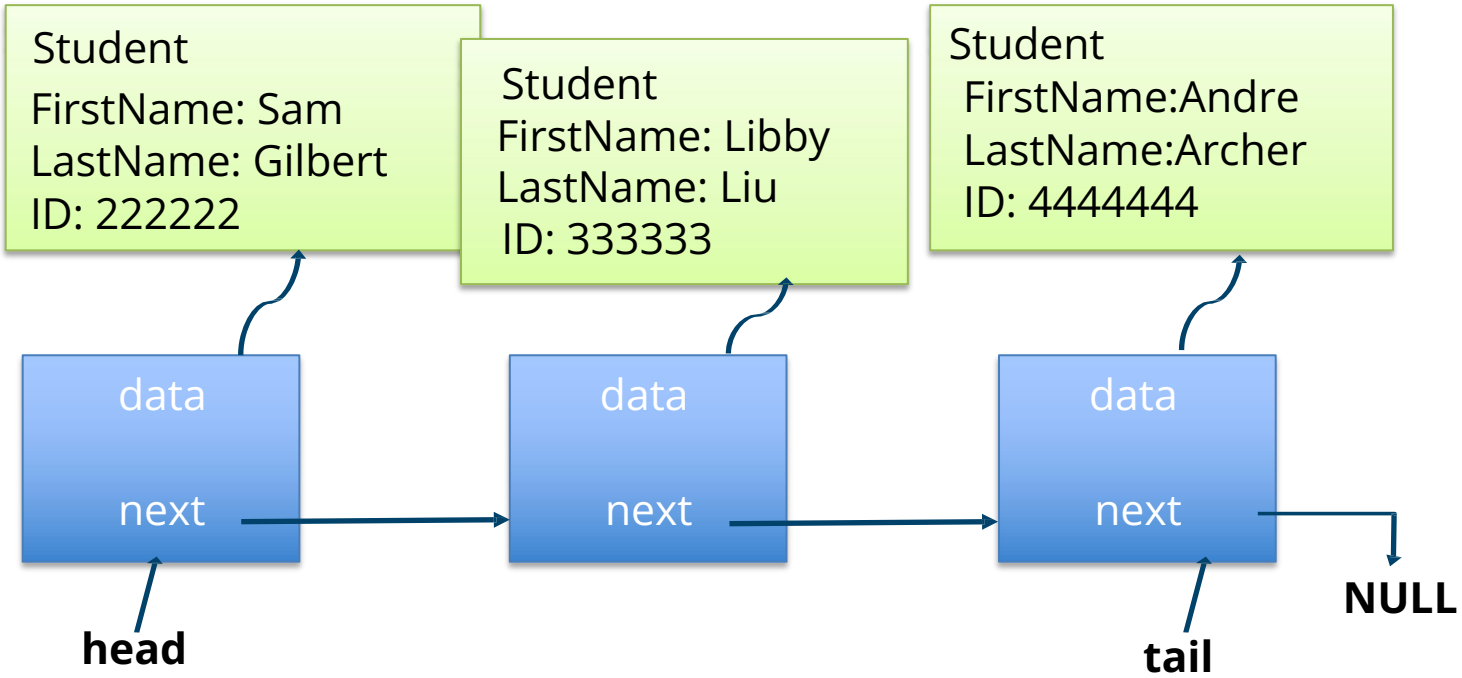
```
private static class Node<E>   {  private E data;
private Node<E> next;  // 'link' to next one

public Node(E data, Node<E> next)
{
this.data = data;  this.next = next;
}

}
```

LinkedList Implementation

The bare minimum LinkedList class contains a declaration of the Node class and private instance variables of type Node for the head and the tail, along with methods for all of the operations.



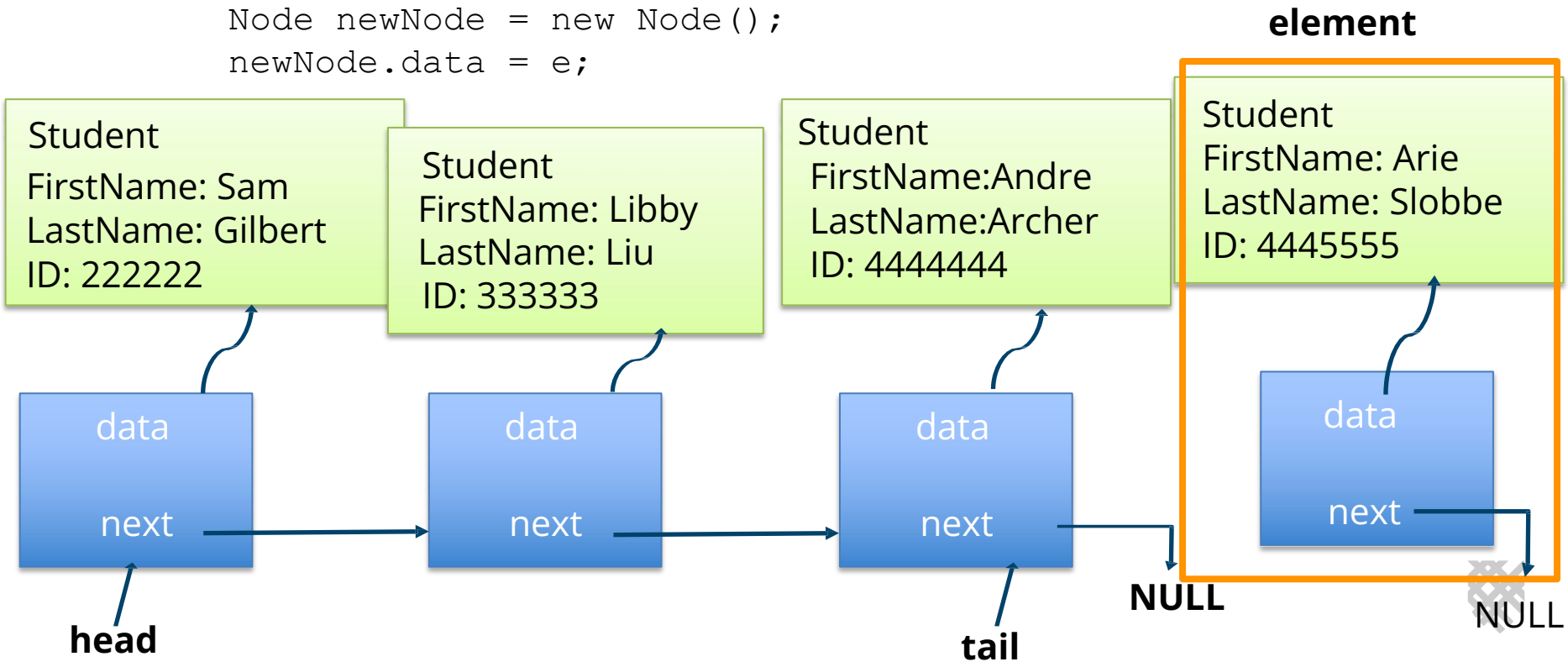
Typical Operations

`boolean add(E e)`	Appends the specified element to the end of this list.
`void add(int index, E element)`	Inserts the specified element at the specified position in this list.
`void addFirst(E e)`	Inserts the specified element at the beginning of this list.
`Void addLast(E e)`	Appends the specified element to the end of the list.
`E get(int index)`	Returns the element at the specified position in this list.
`Int indexOf(Object o)`	Returns the index of the first occurrence of the specified element in this list, or -1 if this list does not contain the element.
`E remove(int index)`	Removes the element at the specified position in this list.
`boolean remove(Object o)`	Removes the first occurrence of the specified element from this list, if it is present.

void addLast(E e) / boolean add(E e)

Step 1: Create new node for the element

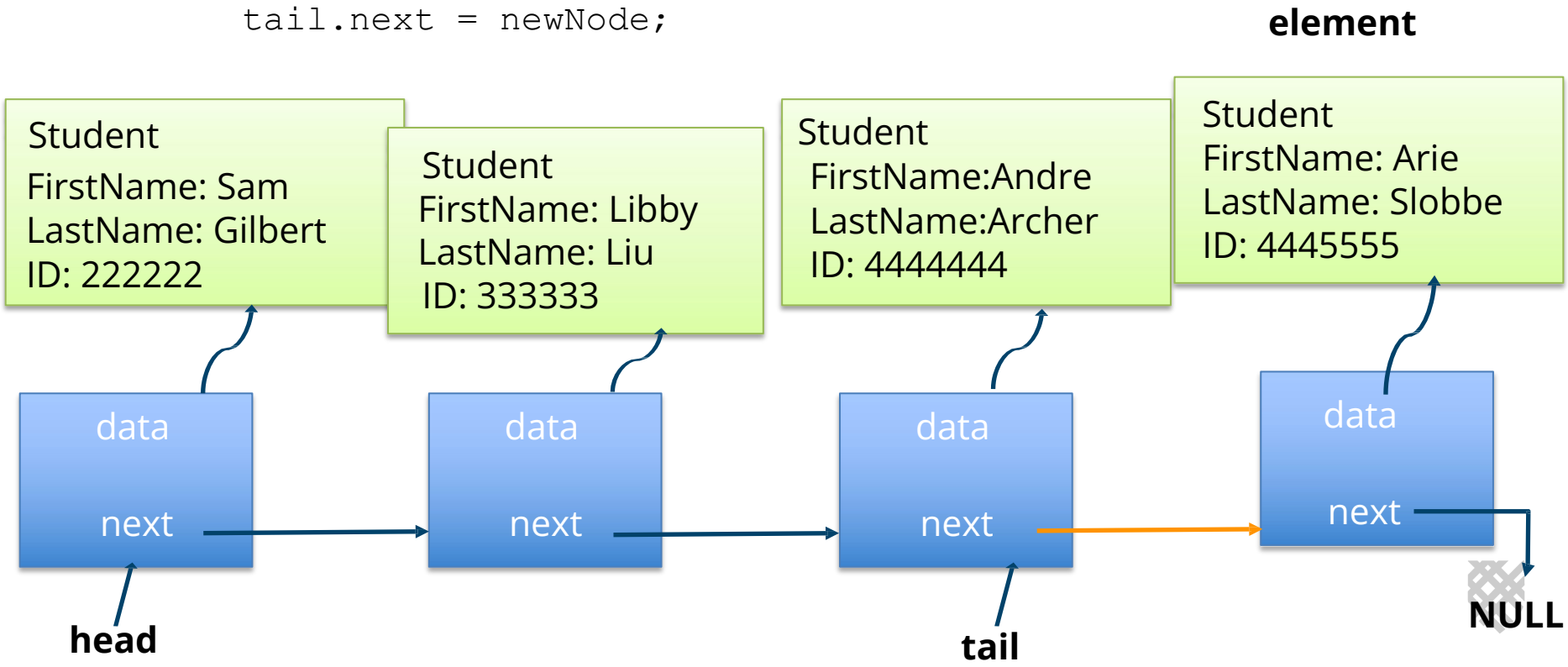
```
Node newNode = new Node();
newNode.data = e;
```



void addLast(E e) / boolean add(E e)

Step 2: Set next of the current tail to be the new element.

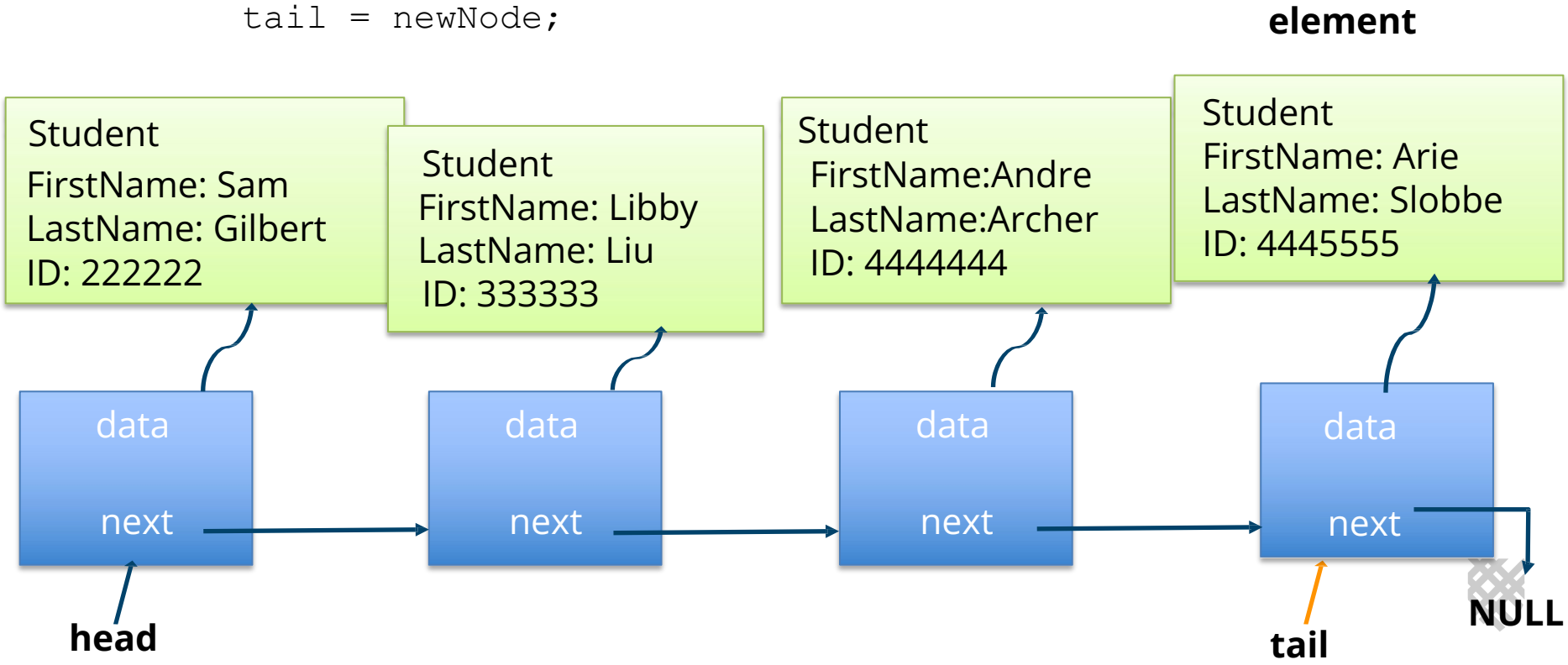
```
tail.next = newNode;
```



void addLast(E e) / boolean add(E e)

Step 3: Update tail.

```
tail = newNode;
```



At worst, how long do these take?

Operation	Runtime
`boolean add(E e)`	O(1)
`void add(int index, E element)`	O(n)
`void addFirst(E e)`	O(1)
`void addLast(E e)`	O(1)
`E get(int index)`	O(n)
`Int indexOf(Object o)`	O(n)
`E remove(int index)`	O(n)
`boolean remove(Object o)`	O(n)

Iterating with Basic For Loop

```
for (int i = 0; i < studentList.size(); i++) {
    System.out.println(studentList.get(i));
}
```

Using an Iterator Instead

```
Iterator iter = studentList.iterator();
while (iter.hasNext()) {
    Student stu = (Student) iter.next();
    System.out.println(stu);
    // Calling iter.remove() while iterating takes O(1)
}
```

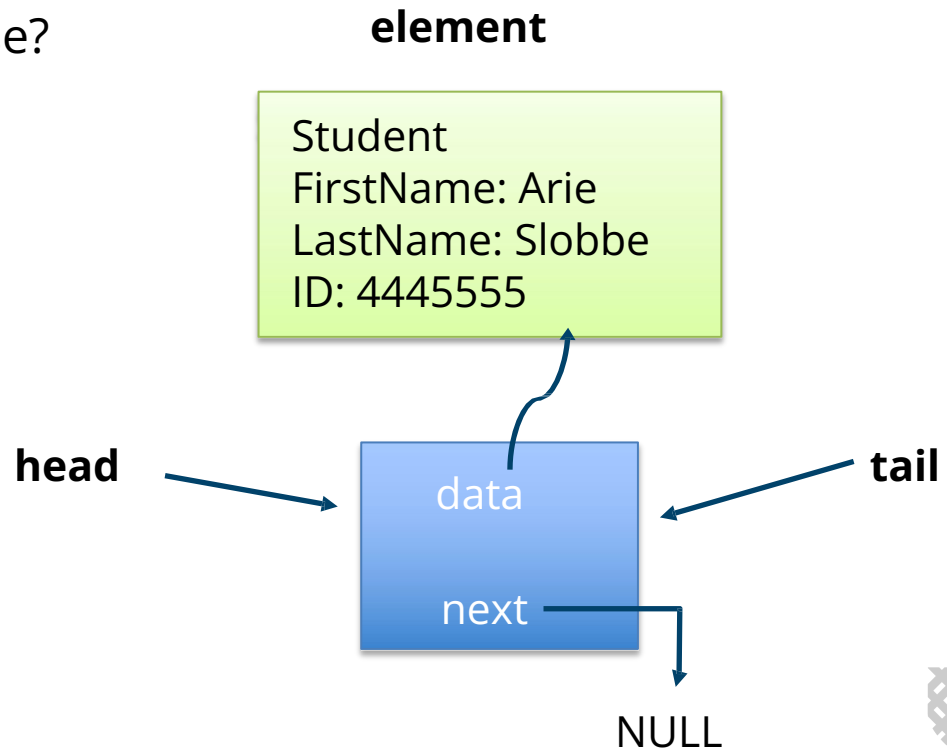
Edge Case: Empty List

How does add() work in this case?

Edge Case: Empty List

How does add() work in this case?

Algorithm:
Node newNode = new Node();
newNode.data = e;
head = newNode;
tail = newNode;



Edge Case: Empty List

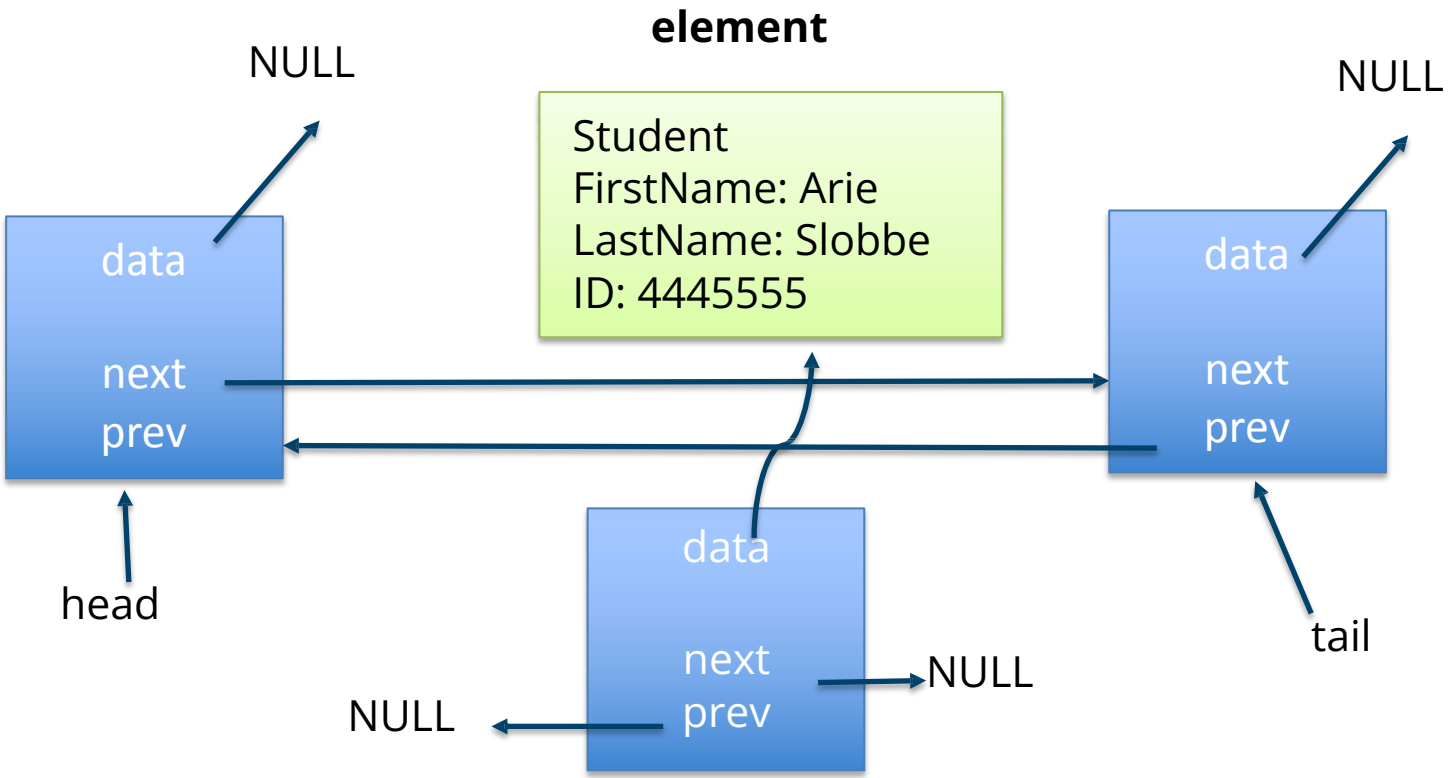
How does add() work in this case?

Algorithm:
Node newNode = new Node();
newNode.data = e;
head = newNode;
tail = newNode;

A non-empty list worked like this:

Algorithm:
Node newNode = new Node();
newNode.data = e;
tail.next = newNode;
tail = newNode;

Using Sentinel/Dummy Nodes



Using Sentinel/Dummy Nodes

Fill in the rest of these methods:

add at front Algorithm:

Node newNode = new Node();
newNode.data = e;

add at end Algorithm:

Node newNode = new Node();
newNode.data = e;

Using Sentinel/Dummy Nodes

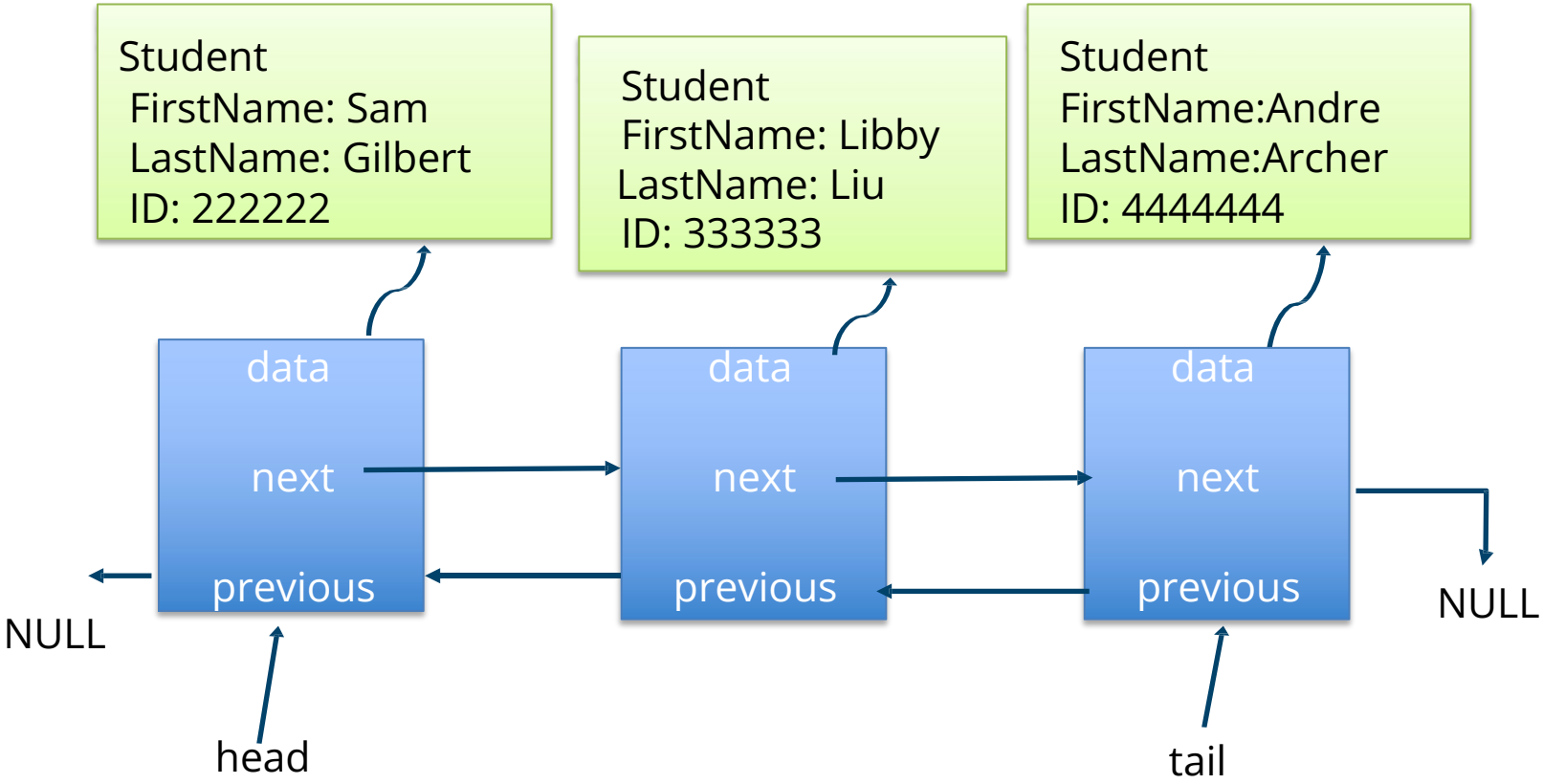
Fill in the rest of these methods:

add at front Algorithm:

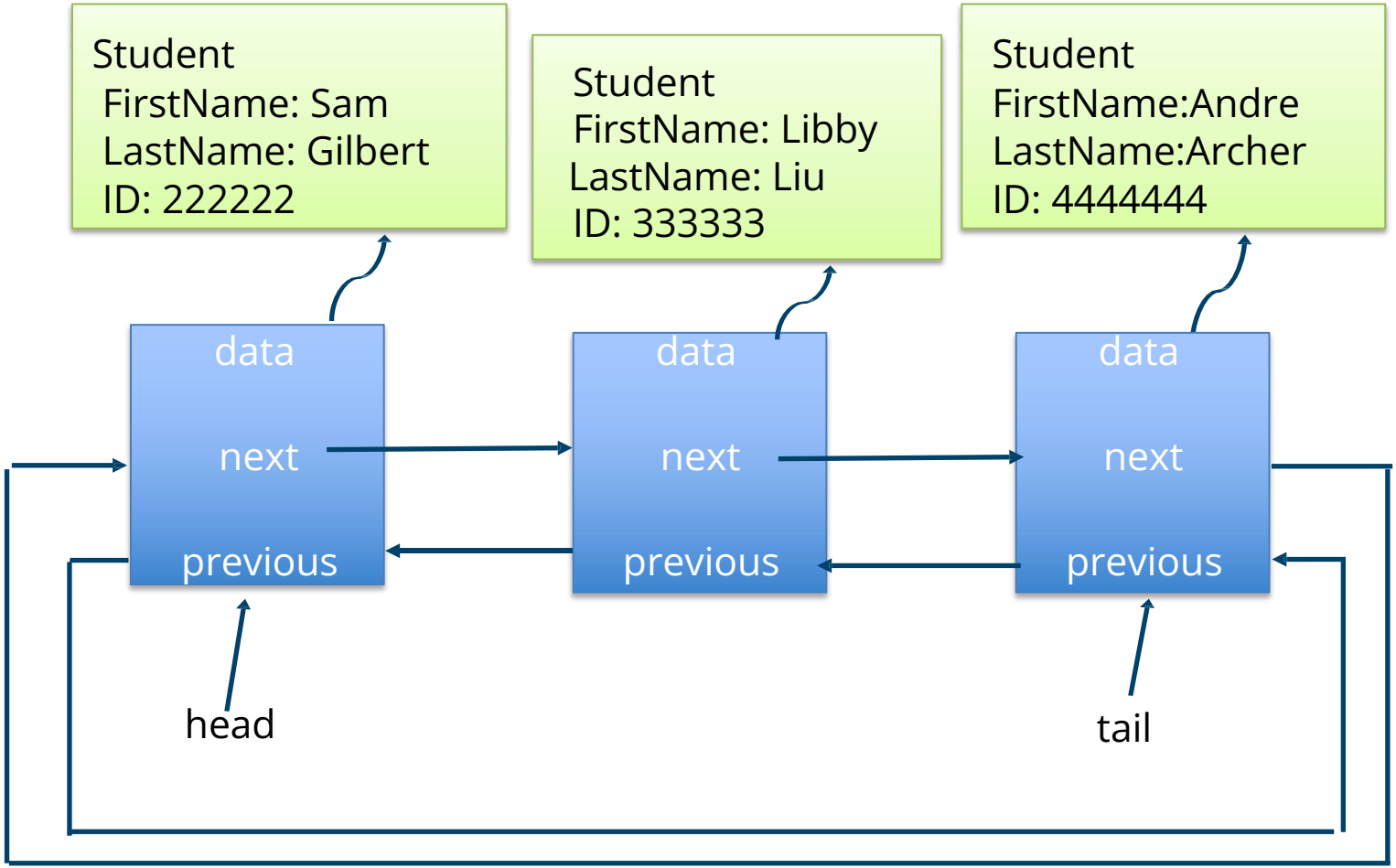
```
Node newNode = new Node();
newNode.data = e;
head.next.prev = newNode;
newNode.next = head.next;
head.next = newNode;
newNode.prev = head;
```

add at end Algorithm:

```
Node newNode = new Node();
newNode.data = e;
newNode.prev = tail.prev;
tail.prev.next = newNode;
newNode.next = tail;
tail.prev = newNode;
```



Linked Lists can be single-linked (only a next pointer) or double-linked (next and previous) which allows you to iterate in both directions.



They can also be circular.

In-class Activity

LinkedList Implementation Activity