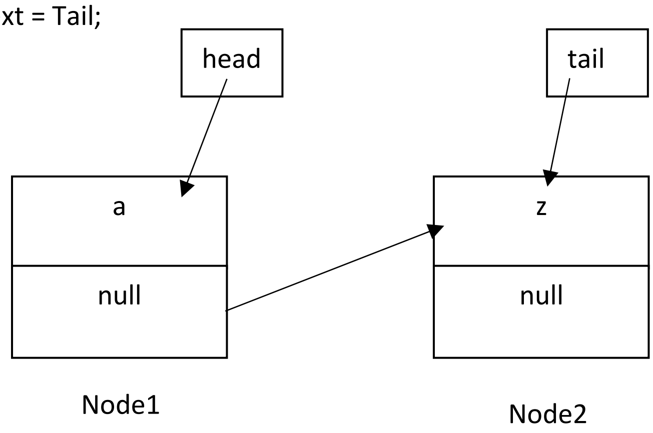
Darren Sow Zhu Jian U205252
Question 1

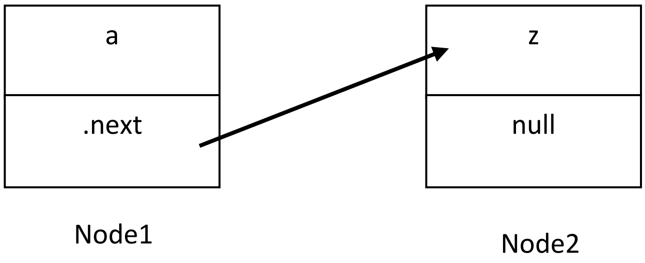
a) Assume that a node class called Node<E> exist. Create two nodes called node1 and node2. Node1 contains alphabet ‘a’ and node2 contains alphabet ‘z’.

```
Node<Character> Node1 = new Node<>(‘a’);
Node<Character> Node2 = new Node<>(‘z’);
```

b) Draw the nodes from (a).

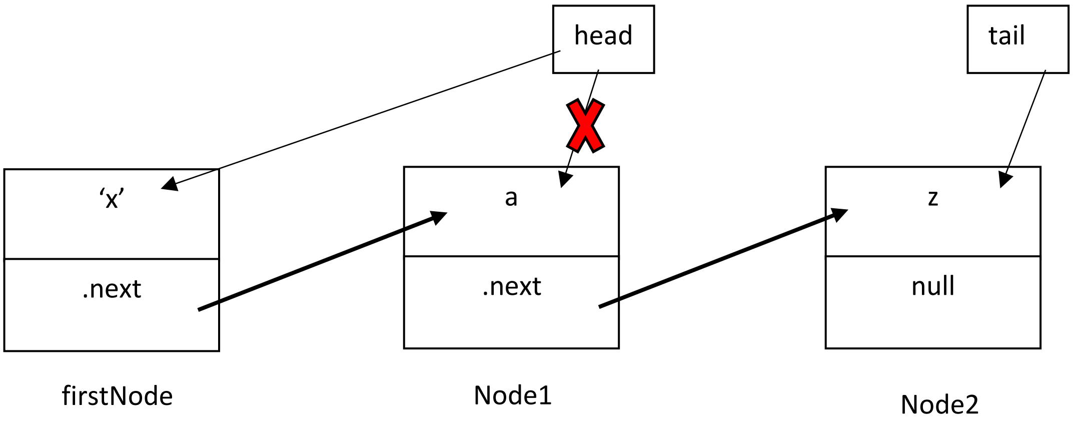
```
Head = Node1;
Tail = Head;

Tail = Node2;
Head.next = Tail;
```





c) Write a statement/code for node1 accessing the node2. Modify 1(b) to show this.
Current = head
Current = head.next;

d) Create a new node, firstNode. Add this new node at the first location of all existing nodes. Draw these nodes.



```
Public void addFirst(E e) {
        Node<E> newNode = new Node<>(e);
        newNode.next = head;
        head = newNode;
        size++;
        if(tail == null) {
                tail = head;
        }
}
```

e) What are the conditions for this operation?
firstNode will become the new head, Node1 becomes the second node in the list while the Node2 remains as the last node(tail).

(If the tail is null, then point the tail to head because this indicates the addition of the first node in the linked list)

f) Write a list of operations/steps/pseudocode needed to add the firstNode to the first location.

- Create the new node
- Point the new node to the current head
- Point the head to the new node
- Increase the size by 1
- If the tail is a null object, point the tail to the head as well as there are no existing nodes in the list.

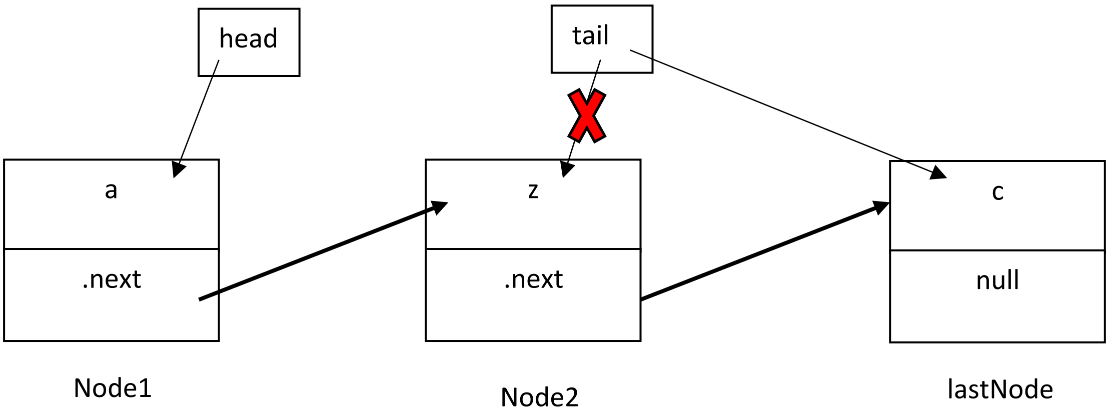
```
public void addFirst(E e){
        Node<E> newNode = new Node<>(e);
        newNode.next = head; // points the new node to the current head
        Head = newNode; // assign the head to the newly declared node, making it the
head
        size++;
        if (Tail == null)
                Tail = Head;
}
```

g) Write codes to assign the firstNode to the first location.

Head = firstNode;

h) Repeat (d) – (f), for the following operations :

a. addLast() – value of element, c

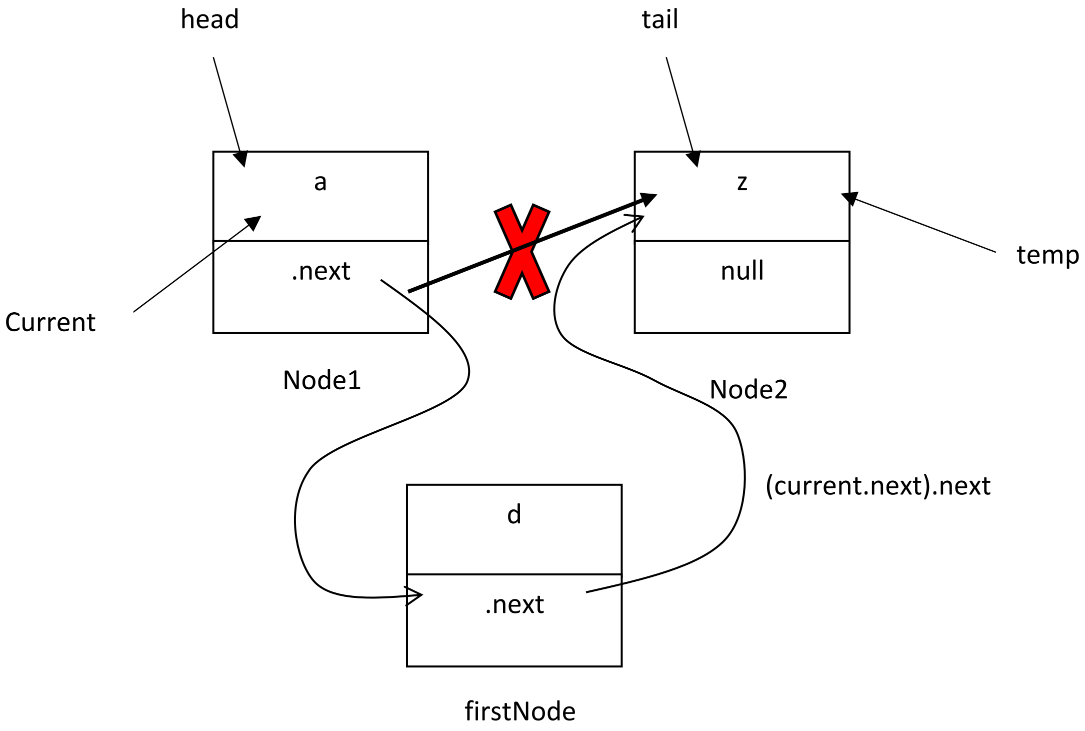


- Create a lastNode
- Point the next element of the tail to the newly created lastNode
- Assign the tail pointer to the tail's next element(lastNode)
- Increase the size of the linked list

lastNode will become the tail. Node2.next points to lastNode.

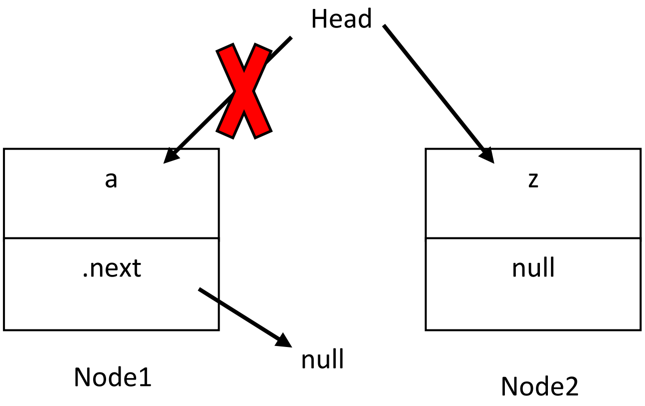
```
Node<E> lastNode = new Node<>('c');
Tail.next =lastNode;
Tail = Tail.next; //Tail = lastNode;
Size++;
```

b. add(int index, E e) – value of element, d



```
public void add(int index, E e) {
        if (index = 0) { addFirst(e); }
        else if (index >= size) { addLast(e); }
        else {
                Node<E> current = head;
                For (int i = 1; i < index; i++) {
                        Current = current.next;
                }
                Node<E> temp = current.next;
                Current.next = new Node<E>(e);
                (current.next).next = temp;
        }
}
```

c. removeFirst()

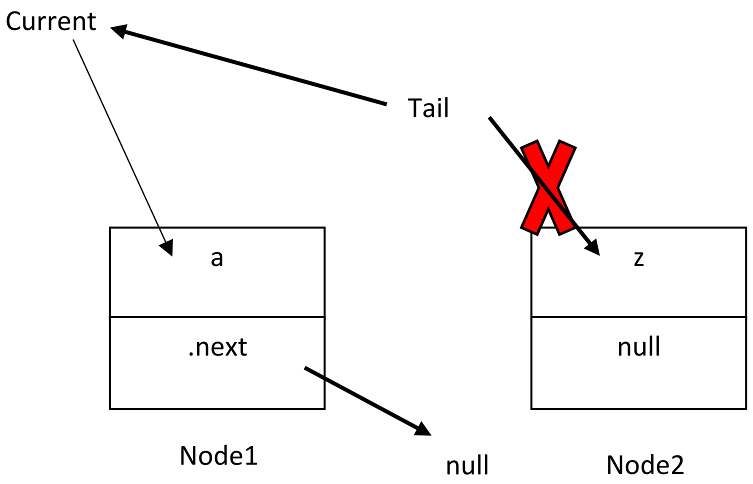


z becomes the head and also the tail.

- Locate the head
- Store the value of head in a variable ‘temp’
- Assign the head element to the second element in the list as the new Head.
- Decrease the size of the list
- Return the head removed ‘temp’

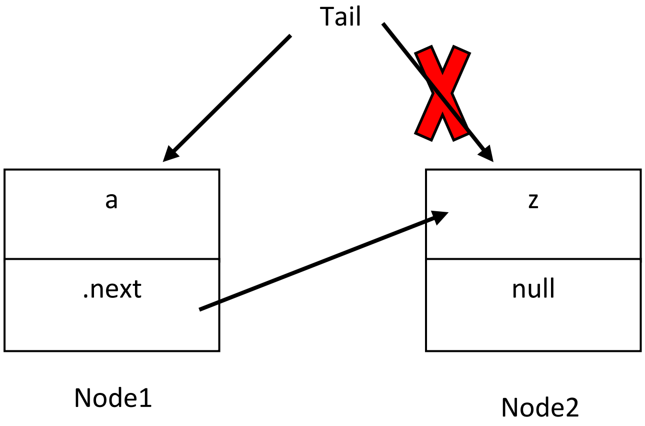
```
public E removeFirst(){
        if (size == 0) return null;
        else {
                Node<E> temp = head;
                head = head.next; // assigns the head reference variable to the
new head
                temp.next = null; //removes the link between the removedHead
and the new head
                size--;
                if (head == null) tail = null;
                return temp.element;
        }
}
```

d. removeLast()



```
public E removeLast() {
            if (size == 0) { return null };
            else if (size == 1) //only 1 node
            {
                        Node<E> temp = head;
                        head = tail = null; //reset to null
                        size = 0;
                        return temp.element; //to know what we delete
            }
            else
            {
                        Node<E> current = head;
                        for (int i = 0; i < size - 2; i++)
                                    current = current.next; //stop 1 node before tail
                        Node<E> temp = tail; //copy tail to temp b4 delete
                        tail = current; //current become tail
                        tail.next = null; //reset the next for tail to be null
                        size--;
                        return temp.element;
            }
}
```

e. remove(int index) – remove at index 1



***take note in some languages that remove index = -1 is removing the last element in the list**

```
public E remove(int index) { // index = 1;
        if (index < 0 || index >= size) return null; // to delete index of node not in range
        else if (index == 0) return removeFirst(); //call removeFirst
        else if (index == size – 1) return removeLast(); //call removeLast <- removes the last element
        else {
                Node<E> previous = head; //Set head to be previous
                for (int i = 1; i < index; i++) {
                        previous = previous.next; // stop before index that want to be deleted
                }
                Node<E> current = previous.next; //copy previous.next to current
                previous.next = current.next; //set new point to from previous.next to current.next
                size--; //reduce size
                return current.element;
        }
}
```

Question 2

a) The name of the methods is ‘contains’

b)

```
public boolean contains(E e) {
        Node<E> pointerB = head;
        for(int i = 0; i<size; int++) {
                if (pointerB.element == e) {
                        System.out.println(current.element);
                        return true;
                }
                pointerB = pointerB.next;
        }
        return false;
}
```

Question 3

a) Name: removeLast();

b)

```
public E removeLast() {
        if (size == 0) return null;
        else if (size == 1) { //only 1 node
                Node<E> temp = head;
                head = tail = null; //reset to null
                size = 0;
                return temp.element; //to know what we delete
        }
        else {
                Node<E> pointer1 = head;
                for (int i = 0; i < size - 2; i++)
                        pointer1 = pointer1.next; //stop 1 node before tail
                Node<E> temp = tail; //copy tail to temp b4 delete
                tail = pointer1; //current become tail
                tail.next = null; //reset the next for tail to be null
                size--;
                return temp.element;
        }
}
```