**WIA 1002**

**TUTORIAL 4 LinkedList**

**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’. Also, create 2

references, head and tail. Let head points to node 1 and tail points to node 2.

b) Draw the nodes from (a).

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

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

nodes. Draw these nodes.

e) If we have no information about the status of a linked-list, what are the conditions we

need to consider to perform the operation in (d)?

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

location.

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

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

i. addLast() – value of element, c

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

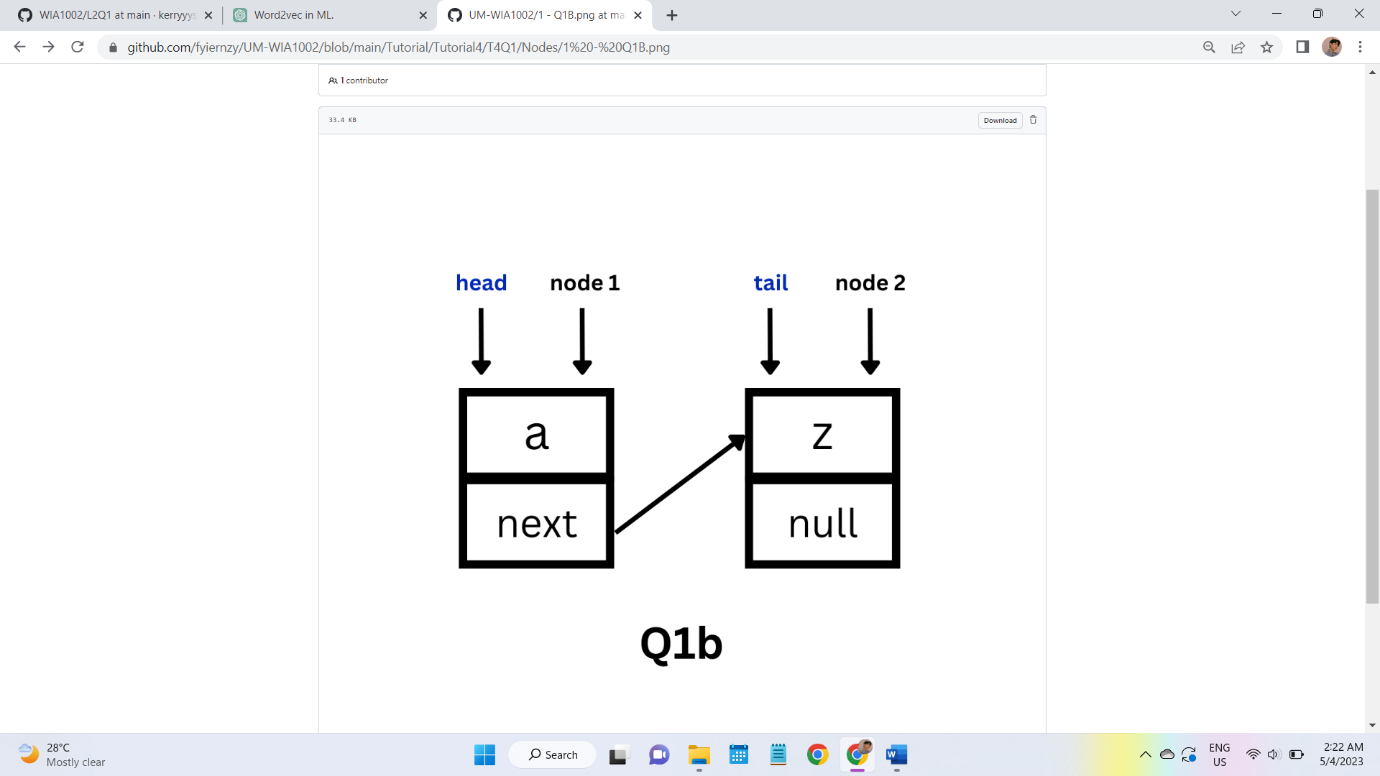
iii. removeFirst()

iv. removeLast()

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

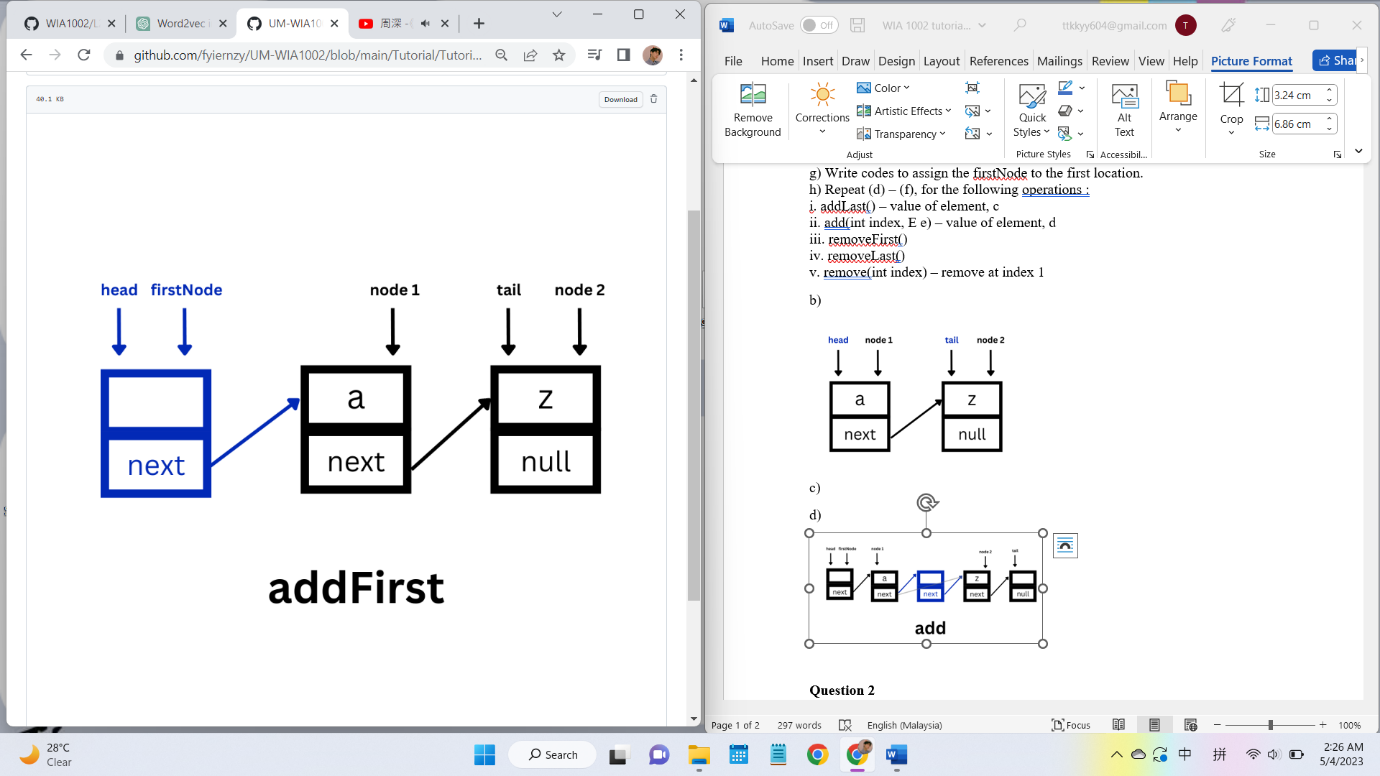
a)

b)



c)node1.next=node2; // head.next=tail;

d)



e)Condition to consider:

whether the list contain any node

if it doesn’t, the tail should be set to this new node

f) Set the next of this firstNode to the current head

Set the head to this firstNode.

*g) public class LinkedList<E>{*

|  |
| --- |
|  |
|  | *Node<E> head;* |
|  | *Node<E> tail;* |
|  | *int size;* |
|  |  |

public void addFirst(**E** item){  
 Node<**E**> newNode=new Node<**E**>(item**,**head)**; //head=next, =set item as head, so head will move to become item.next** newNode.next=head**;** if(tail == null)  
 tail = head**;** head = newNode**;** size++**;**}

*h)*

*public class Main{*

*public static void main(String[] args){*

*LinkedList<Character> list = new LinkedList<>();*

*Node<Character> node1 = new Node<>(‘a’);*

*Node<Character> node2 = new Node<>(‘z’);*

*list.head=node1;*

*list.tail=node2;*

*list.addLast(‘c’);*

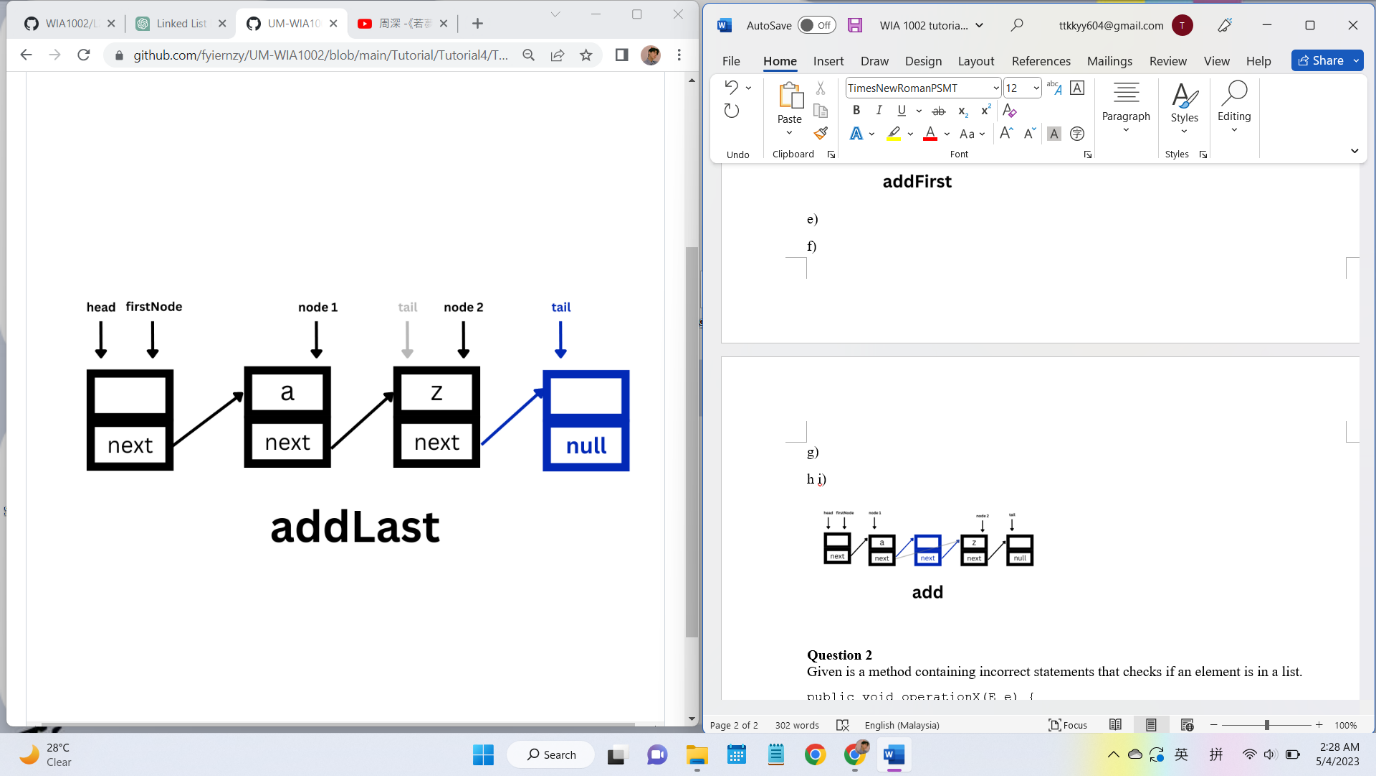
*list.add(2,‘d’);*

*list.remove(1);*

*}*

*}*

*h i)*

**

*Condition to consider:*

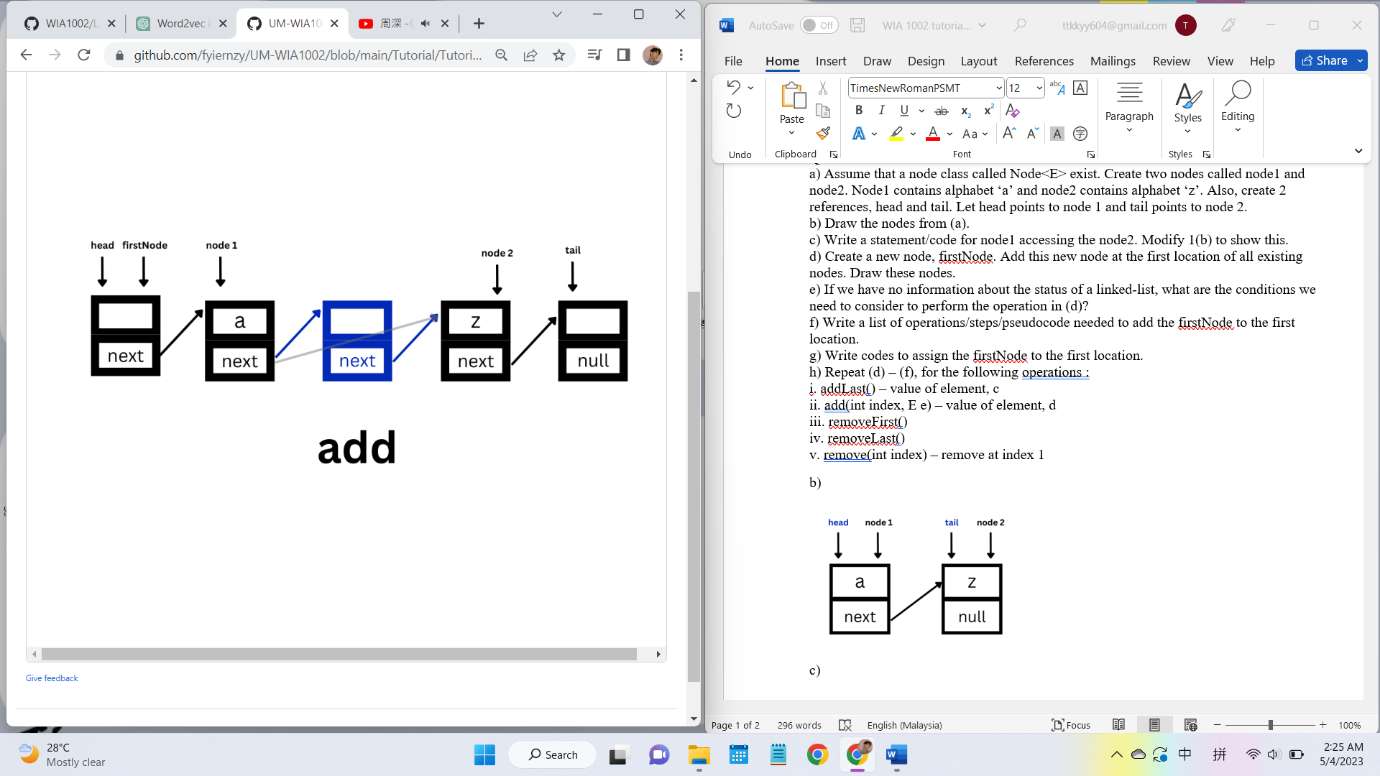
*Whether the list contain any node.*

*If it doesn’t, the head should be set to this new node.*

*f) set the next of tail to the newNode.*

public void addLast(**E** item){  
 Node<**E**> newNode=new Node<**E**>(item**,**null)**;** if(tail == null){  
 head=tail=newNode**;** }  
 else  
 tail.next=newNode**;** tail=tail.next**;** //tail=newNode;  
 size++**;**}

*h ii)*

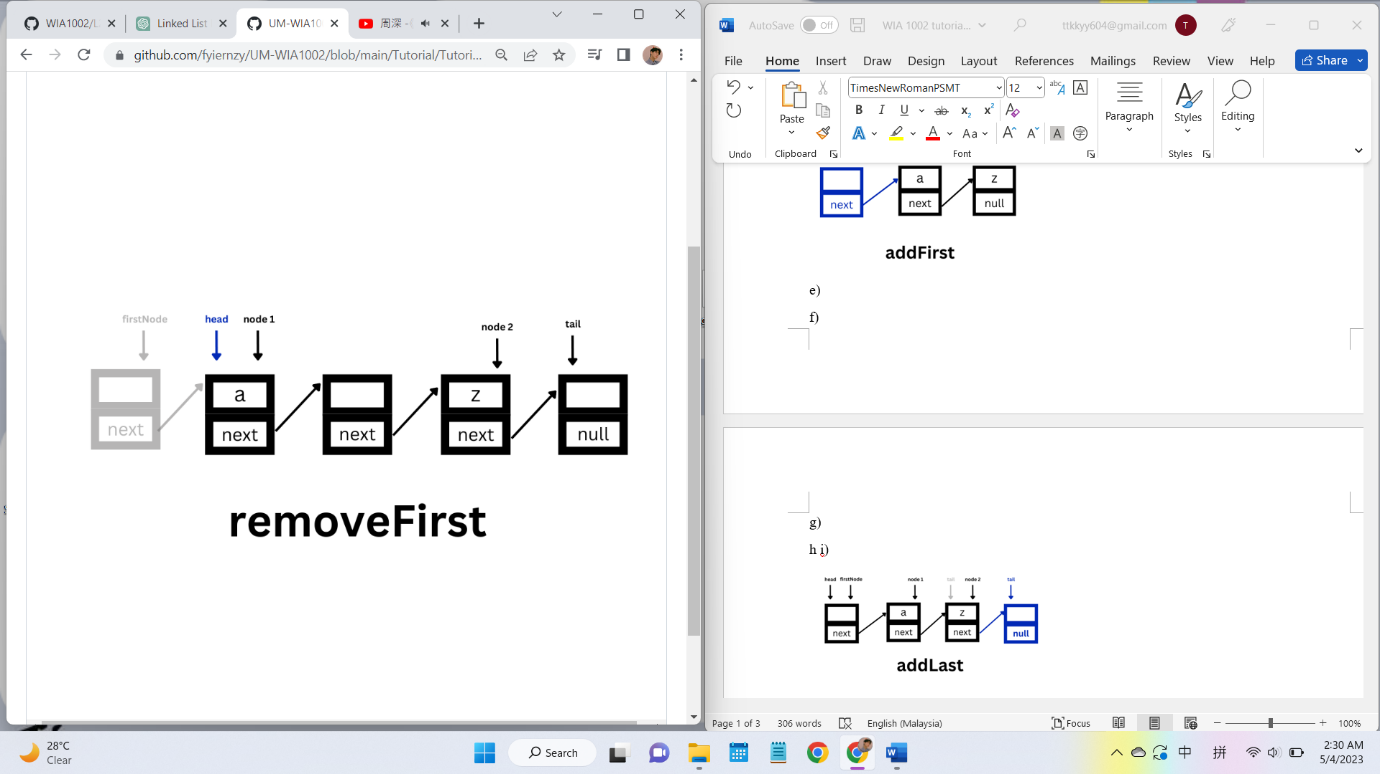
**

*Condition to consider:*

|  |  |
| --- | --- |
|  | *Throw an exception when index < 0 || index > size* |
|  | *Add the node otherwise* |
| *f)* |
|  | *If index equals 0, invoke addFirst() method* |
|  | *If index equals the size of the list, invoke the addLast() method* |
|  | *Otherwise, search for the (index - 1) the node assign it to the prev* |
|  | *Add the newNode next to the prev* |
|  | *Set the next of the newNode to the node after the prev* |
|  | *Increment the size of the list by 1* |

public void add(int index**, E** e){  
 if(index<=**0**){  
 addFirst(e)**;** return**;** }  
 try{  
 Node<**E**>beforeToAdd=head**;** for(int i=**1;** i<index**;**i++)  
 beforeToAdd = beforeToAdd.next**;** Node<**E**> toAdd = new Node<>(e)**;** toAdd.next = beforeToAdd.next**;** beforeToAdd.next=toAdd**;** if(toAdd.next == null)  
 tail = toAdd**;** }catch (NullPointerException ex){ // an exception that occurs when you try to access a reference variable that is null. In other words, you are trying to use an object that has not been initialized.  
 addLast(e)**;** }  
}

*h iii)*

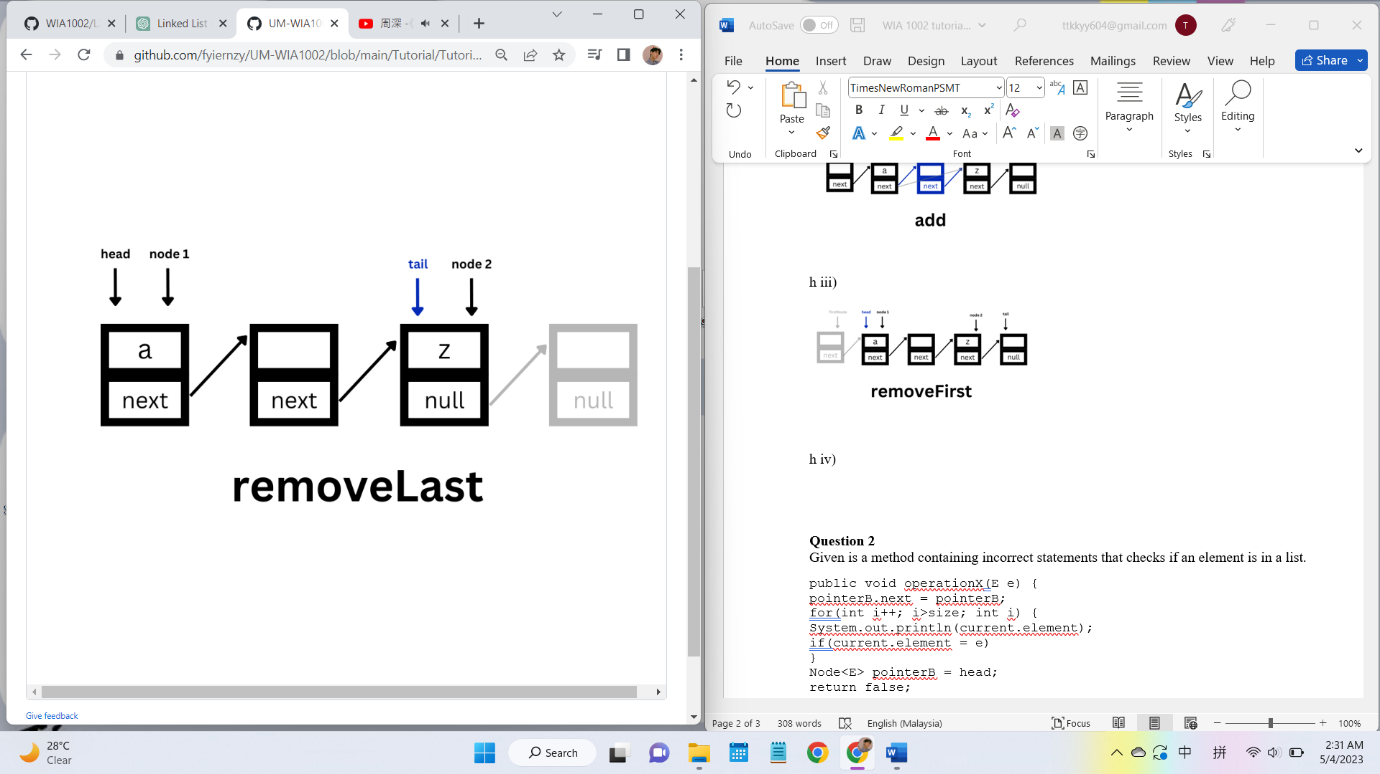
**

|  |
| --- |
|  |
|  | *e)* |
|  | *Condition to consider:* |
|  | *If the list has no node, throw NoSuchElementException* |
|  | *If the list has only one node, set both head and tail to null.* |
|  |  |
|  | *f)* |
|  | *Set the head to the node after the current head* |
|  | *Return the item of the initial head* |
|  |  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

public **E** removeFirst(){  
 if (head == null) return null**;**

**E** item = head.element**;** head = head.next**;** if (head == null) tail = null**;** return item**;**}

*h iv)*

**

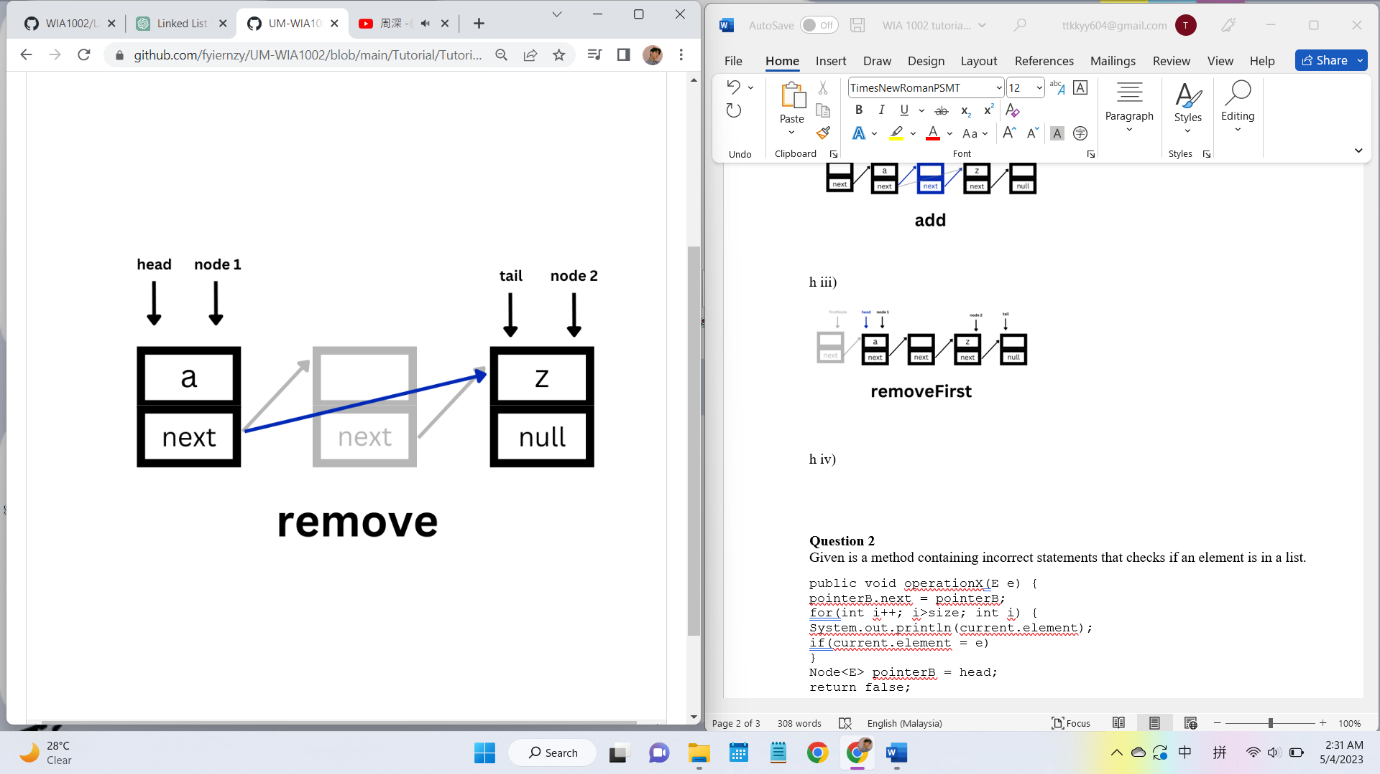
|  |
| --- |
|  |
| e) Condition to consider: If the list has no node**,** throwNoSuchElementException If the list has only one node**,** set both head and tail to null.  f) Assign the node before the tail to prev. Set the next of the prev to null. Return the item of the initial tail. | |
|  | |
|  | |
|  | |
|  | |
|  | |
|  | |
|  | |
|  | |
|  | |  |

*g)*

public **E** removeLast(){  
 if (tail == null) return null**;**

**E** item = tail.element**;** if (head == tail) {  
 head = tail = null**;** return item**;** }  
 Node<**E**> beforeLast = head**;** while (beforeLast.next.next != null)  
 beforeLast = beforeLast.next**;** beforeLast.next = null**;** tail = beforeLast**;** return item**;**}

*h v)*

**

|  |
| --- |
|  |
|  | *e)* |
|  | *Condition to consider:* |
|  | *Throw IndexOutOfBoundsException when index < 0 || index > size - 1* |
|  |  |
|  | *f)* |
|  | *If the index is 0, invoke the removeFirst method.* |
|  | *Otherwise, search for (index-1)th node and assign it to current.* |
|  | *Set the next of the current to the node after the next of the current.* |
|  | *Return the item of the node after the initial current.* |
|  |  |

public **E** remove(int index){  
 if (index < **0**) return null**;** if (index == **0**) return removeFirst()**;** try {  
 Node<**E**> beforeToRemove = head**;** for (int i = **1;** i < index**;** i++)  
 beforeToRemove = beforeToRemove.next**;  
 E** item = beforeToRemove.next.element**;** beforeToRemove.next = beforeToRemove.next.next**;** if (beforeToRemove.next == null) tail = beforeToRemove**;** return item**;** } catch (NullPointerException ex) {  
 return null**;** }  
}

**Question 2**

Given is a method containing incorrect statements that checks if an element is in a list.

public void operationX(E e) {

pointerB.next = pointerB;

for(int i++; i>size; int i) {

System.out.println(current.element);

if(current.element = e)

}

Node<E> pointerB = head;

return false;

}

a) What is the name of the method for operationX?

b) Correct the statements by rewriting them in the correct order and syntax. Write the

correct/right method name to replace operationX.

1. contains

public class Main {

}

class LinkedList<E> {

static class Node<E> {

E element;

Node<E> next;

}

Node<E> head;

Node<E> tail;

int size;

public boolean contains(E e) {

// for(Node<E> node = head; node != null; node = node.next)

// if(item.equals(node.item))

// return true;

// return false;

Node<E> current = head;

for (int i = 0; i < size; i++) {

if (e.equals(current))

return true;

current = current.next;

}

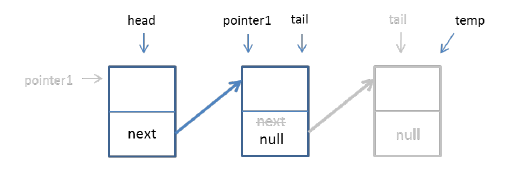
return false;

}

}

**Question 3**

Given the following nodes. Answer the following:



a) Based on the above figure, what is the name of the method for above operation?

b) Write codes to represent the above figure. (Kindly use the variables stated in the

figure)

1. romoveLast

import java.util.NoSuchElementException;

public class Main {

}

class LinkedList<E> {

static class Node<E> {

E item;

Node<E> next;

}

Node<E> head;

Node<E> tail;

public E removeLast() {

if (tail == null)

throw new NoSuchElementException();

Node<E> prev = null;

Node<E> node = head;

while (node.next != null) {

prev = node;

node = node.next;

}

if (prev == null)

head = null;

else

prev.next = null;

tail = prev;

return node.item;

}

}