EECS2011 Fundamentals of Data Structures (Winter 2022)

Q&A - Week 4 Lecture

Wednesday, February 9

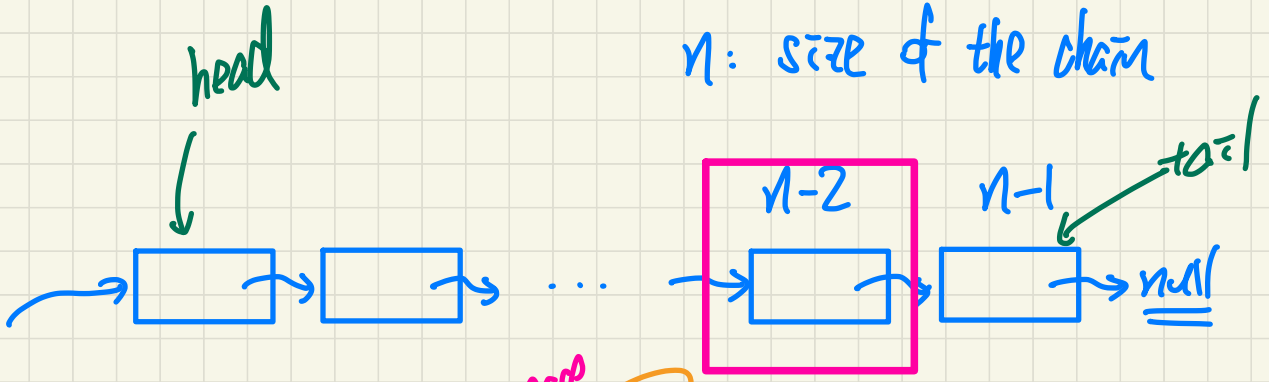
Announcements

eClass

- Written Test 1 due next Monday or Tuesday
- Revised start time of Written Test 1
- Example questions for Written Test 1 released

- Assignment 1 (on SLLs) due next Tuesday

✓
- Lecture W5 postponed until next Wednesday



worst case
i == n-2

Worst case:

Node getNodeAt(int i){

getNext invoked n-1 times

large.

...

}

When n increases, the worst-case value for i increases.

O(n)

10 M

For some of the methods we do 1 block of error checking like this:

```
method1 throws Exception
if () {
    throw exception
}
else{
    //rest of the code
    //
}
```

either is ok.

Sometimes I exclude the "else{}" to save 1 indentation level and
to minimize confusion when having multiple loose curly brackets at the end.
Since the code below the 'if block' can only be reached if there is no error anyways.

```
if (){
    throw exception
}
// rest of the code
```

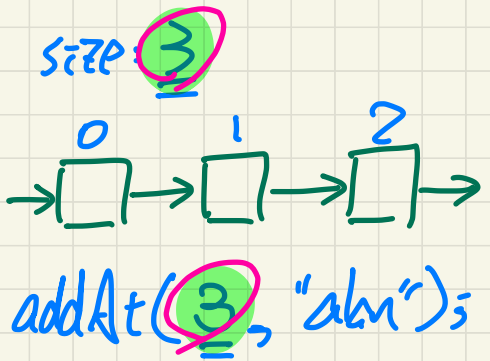
Question: Is this way of doing it bad programming practice ?

Hi professor,
In the "addAt" method, if we want to keep track of the tail, should we add a case in the algorithm for that?
My thinking is to have a special case
if the size of the list equals the index to be added.
In that case we simply use addLast. i.e.:

```
if(this.size == i-1) {
    this.addLast(new Node(e,null);
}
```

Else the algorithm runs as normal.
This allows us to keep track of the tail and makes one special case faster.

special case: addLast.




```
else if(size == i) {
    this.addLast(new Node(e, null));
}
```

updates tail ref. property.

```
1  void addAt(int i, String e) {
2    if (i < 0 || i > size) {
3      throw new IllegalArgumentException("Invalid Index.");
4    }
5    else {
6      if (i == 0) {
7        addFirst(e);
8      }
9      else {
10       Node nodeBefore = getNodeAt(i - 1);
11       Node newNode = new Node(e, nodeBefore.getNext());
12       nodeBefore.setNext(newNode);
13       size ++;
14     }
15   }
16 }
```

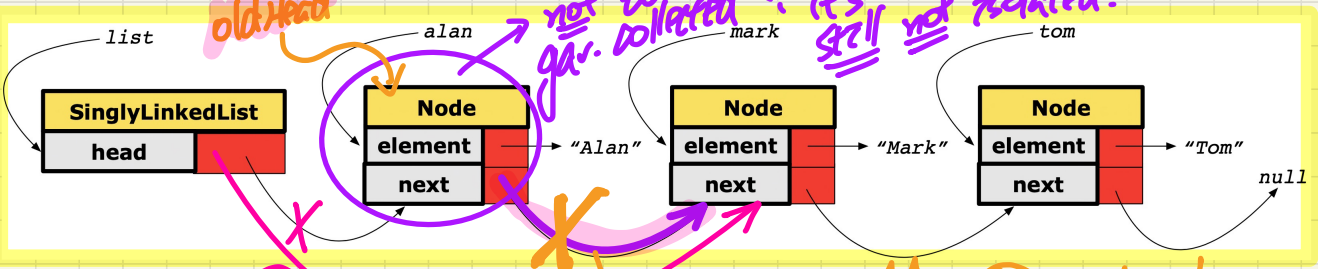
not updating tail property when 'e' is inserted as the last element.

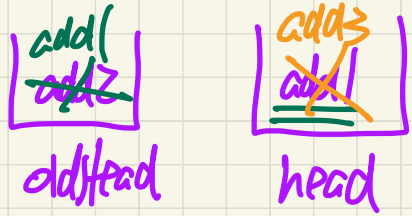
At 17:55 - in removeFirst(), we do

```
oldNode = head
head = head.getNext()
oldHead.setNext(null)
```

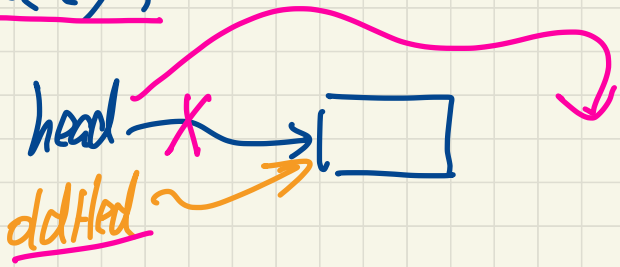
Why do we have to do line1 + line3 for the previous node0 to be Garbage Collected ?

Wouldn't it be garbage collected if we only wrote line2 ?
- since the variable head is no longer pointing to the previous node0,
and no other node or variable has a reference to the previous node0 either ?






```
Node ddHead = head;
head = head.getNext();
```




```
int i = 23;
int j = 46;
i = j;
j++; // i==46 j==47
```

Hi Professor,
When we remove a node from a singly linked list,
despite disconnecting it from the previous or the next nodes
whichever is applicable, I believe it will still occupy space in the memory.

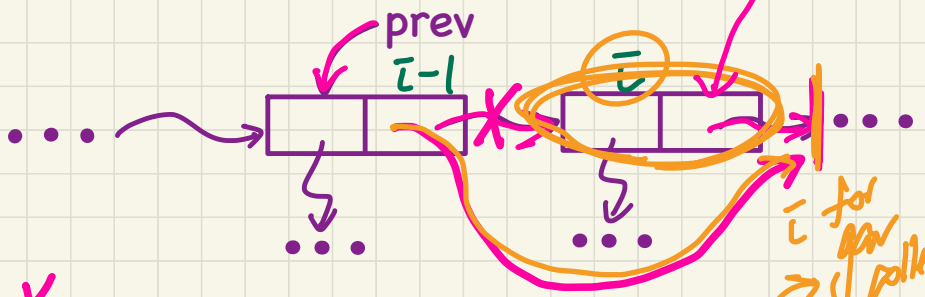
How can we completely remove it so it does not occupy any space in the memory?
I am thinking about a very large linked list and if we remove multiple nodes and they still occupy memory spaces, I think that would be a big problem.
Thank you.

void removeAt(int i)

addAt

```
public void removeAt(int i) {
  if(i < 0 || i >= size) { /* error */ }
  else {
    Node prev = getNodeAt(i - 1);
    ... ?
  }
}
```

O(n)




```
prev.setNext(prev.getNext().getNext());
prev.getNext().setNext(null);
prev.getNext().setElement(null);
```

Exercise: for this so as to get node at index

(not covered by WTI!)

As a developer, when writing a method we might
use the parameter to call some methods defined in that parameter's static type.

For example, if our method is m(Person p), when developing
we might need to call the method p.getBMI() because
we know that parameter p has this method defined.

But when we set the method's parameter type generic m(E p),
how can we call a method (e.g., p.getBMI()) given that we know that we have to call that method to be able to write the method that we are currently writing?

What about a case where an argument with static type Account is passed, how can we avoid this?
Thanks

Constrained Genericity

Node<String> ✗
Node<Integer> ✗

whatever class instantiating E must be a subclass of Person.

```
public class Node<E extends Person> {

    private E element;
    private Node<E> next;

    public double getBMI() {
        return this.element.getBMI();
    }

    public double getTuition() {
        double tuition;
        if(element instanceof Student) {
            tuition = ((Student) element).getTuition();
        }
        else {
            tuition = -1;
        }
        return tuition;
    }
}
```

ST: E

tuition = element.getTuition(); ✗
E (descendant of Person)

```
public class Person {
    public double getBMI() {
        return 0;
    }
}
```

```
public class Student extends Person {
    public double getTuition() {
        return 0;
    }
}
```