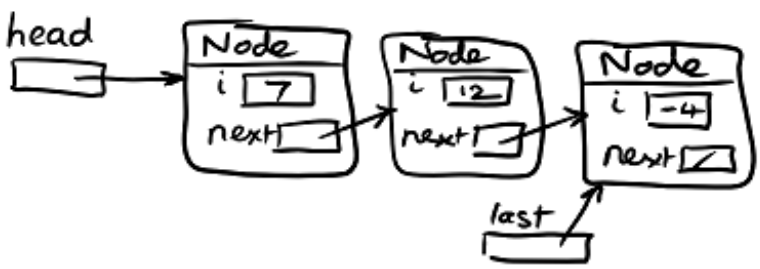
From: **Alan Liang** alan.5100@yahoo.com
Subject: assignment08
Date: November 7, 2016 at 10:30 PM
To: Winnie Liang winnieliang27@yahoo.com

Assignment 8

Due Friday Nov 11 at 11:59pm

Linked lists. The code goes in package `assignment08`

First we are going to build linked lists that look like this:



Make a class `IntLinkedList`. The fields are `head` and `last` of type `Node`. `Node` is a class inside the class `IntLinkedList`. `Node` has fields `i` (an `int`) and `next` (a `Node`). Make all the fields private and the `Node` class private. It may be a surprise but the private fields inside `Node` are visible in all the code of `IntLinkedList`.

Make a method `public void startList(int i)` which makes `head` equal to a new `Node`, sets the `i` field of `head` to the parameter `i`, and sets `last` equal to `head`. The method throws a `RuntimeException` with the message "List already created" if `head` was not *null* at the beginning of the method.

Make a method public void appendList(int i) that throws a `RuntimeException` with the message "Must create list" if `head` is *null* at the beginning of the method. it sets `last.next` to a new `Node`and changes `last` to `last.next` and finally sets the `i` field of `last` to the parameter `i`.

For the next method, use the basic way to move though the linked list:

```
Node temp = head;
while(temp.next != null && <some other problem-specific test>) {
   //optional processing of temp here
   temp = temp.next;
}
```

To remove the `Node` referenced by `temp.next` from the list, we write `temp.next = temp.next.next;`

Write a method `public void remove(int i)` that removes the first `Node` from the list pointed

Write a method `public void remove(int i)`, that removes the first `Node` from the list pointed at by `head` that contains the int `i`. If `head` is *null* throw a `RuntimeException` with the message "List is empty". If the `head` `Node` contains `i`, update `head` to hold `head.next`. Otherwise, use the above code to search through the list and remove the appropriate node, if found. If there is no `Node` in the list that contains `i`, make no changes to the list.

(Draw diagrams to see how this works)

Override `public String toString()` to print the elements of the list, separated by commas and a space and surrounded by brackets. We will describe how Java does this in `AbstractCollection`, making the appropriate changes.

If `head` is *null*, return `"[]"`. Make a `StringBuffer sb` instantiated to contain `"["`. Set the `Node temp` to `head`. Append `head.i` to `sb` and change `temp` to `temp.next`. In an infinite loop, either `while(true)` or `for(;;)`, do the following: if `temp` is *null*, append `"]"` to `sb` and return `sb.toString()`, else append a *comma* and a *space* to `sb`, then append `temp.i` and change `temp` to `temp.next`.

Write JUnit tests that check the correct String is returned if you build a list and call toString and then separately if you get the correct String after an element is removed from the beginning, the end and somewhere in the middle. We will discuss the standard way of checking that Exceptions are working in JUnits later on in the course.

Next write a main method that implements the Sieve of Eratosthenes for finding prime numbers.

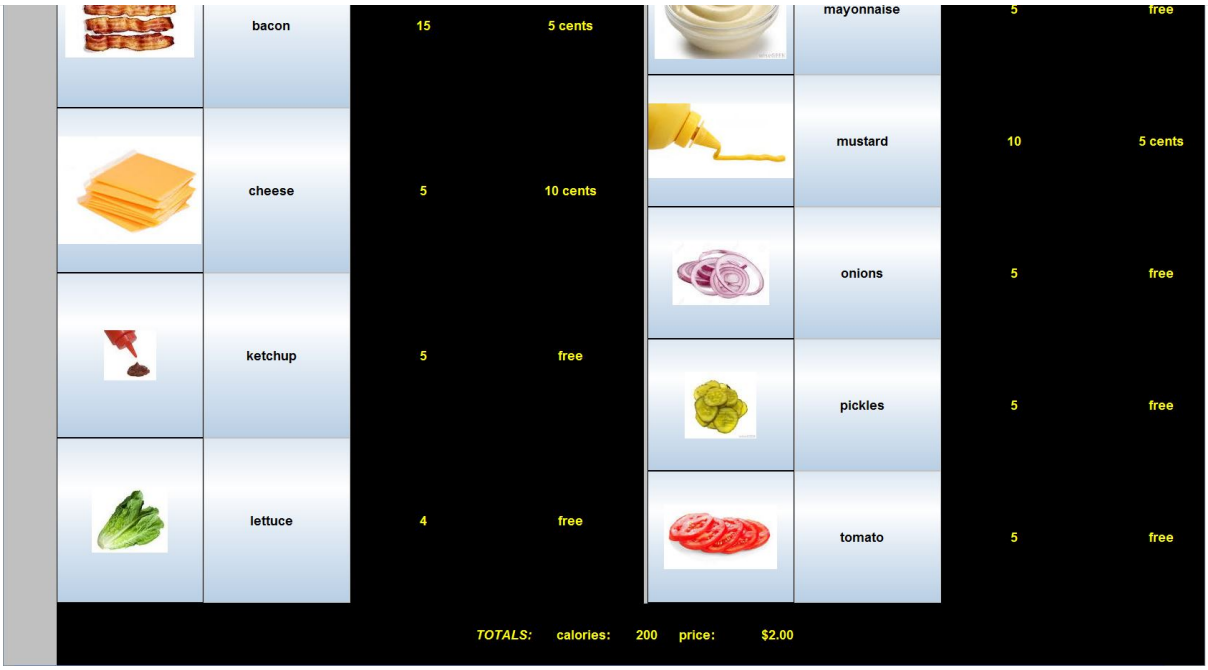
Make an instance `primes` of `IntLinkedList`. Start `primes` with the value 2. In a for loop from `i` = 3 up to something big (start with up to 1000 but later get more ambitious) append `i`. Set the `Node nextPrime` to `primes.head` and print `"["` (use System.out.print, not System.out.println). In a *try/catch* block that catches a `RuntimeException` and print-lines `"]"`. Inside the try, write an infinite loop that does the following: print `nextPrime.i`. Make a `Node temp` equal to `nextPrime` and while `temp` and `temp.next` are not *null*, if `temp.next.i` is an exact multiple of `nextPrime.i`, remove `temp.next` (no need to use the remove method, just use `temp.next = temp.next.next;`). Set `temp` to `temp.next` inside the inner loop.
After the inner loop but inside the infinite loop, set `nextPrime` to `nextPrime.next`.
If `nextPrime` is not *null*, print `", "`.

Make a second version of your code: a class `IntLinkedList2`. All the fields are the same. This time `Node` has a constructor that sets the value of `i`. In `Node` put a method `Node remove(int j)`
If `j` is equal to `i`, the return value is `next`. If not, set `next` to `next.remove(j)` and return `this`.
Change the `startList` method to use `head = new Node(i)`, instead of directly assigning `head.i`. Change `appendList` similarly. Change remove to throw the exception for an empty list but otherwise set `head` to `head.remove(i)`.

Do the same unit tests as before for `IntLinkedList2`.

This second version of the code for the linked list is closer to the way I implemented the following Burger topping ordering system. The code IS overly complex but it explores several design patterns (strategy, singleton, decorator, model-view-controller, ...). The purpose is to maintain a linked list of toppings on a basic burger, which can be added or removed by clicking on the GUI. As the toppings are changed the running total of calories and total cost are updated. Maybe someone will help me complete the Chinese names and if anyone wants to add a language, please share it.

Go to Blackboard->Content->Code->Assignment 9 Burger Decorator to get the code and graphics.



assignment8

Updated 1 day ago by Matthew Hems and Leslie Lander

followup discussions *for lingering questions and comments*

Resolved
Unresolved

Anonymous 2 days ago
At the end, do you mean after the try/catch block? Otherwise the code is inaccessible directly after the infinite loop

Leslie Lander 2 days ago I think this is straightened out now

Reply to this followup discussion

Resolved
Unresolved

Anonymous 1 day ago
The remove method in IntLinkedList2 doesn't remove the first node in the list when the first node is to be removed, but it

works for the rest.

Matthew Hems 1 day ago Correct, the directions have been updated.

Reply to this followup discussion

Resolved

Unresolved

Anonymous 23 hours ago
Does anyone else for the Sieve just get [2, 3] printed a bunch of times? I can't see what I'm doing wrong

Anonymous 22 hours ago I did at first. The for loop is used to populate the linked list from i=3 to 1000 or more by calling append on the linked list. After the for loop, proceed with instantiating a Node nextPrime and so forth.

Anonymous 13 hours ago Thanks

assignment08.zip