Lab 4: 30 pts

The main new components in this lab are discussed in Chapter 3's section on singly linked lists and were reviewed in class.

IMPORTANT: This lab builds on Lab 3, so if you didn't do it, then complete Lab 3 as a part of this lab, but see next comment on GRADING to understand how that works.

GRADING: I won't be deducting points if you didn't do parts of the Lab 3 piece correctly.

**I will only be grading your implementation of the Linked List and how you use it and the player move count.**

**HOWEVER: Be sure to include the screenshot (the last step of the lab) if you get everything working.**

1. **SUMMARY**
   1. Add to Lab 3 by tracking the player's moves in a linked list. Then, before printing out the maze at the end, cycle through all the nodes of the linked list and change the cell value of each grid spot where the user has been to a number like 8 or 5 or 3 or any integer other than 0 or 1. This way the grid print out at the end will show the player's path.
   2. Secondly, track the number of moves the player makes and print that out at the end as well.
   3. Lastly, you will insert some test code that I use and take a screenshot of the print out after running it.
   4. This lab will involve the following new features:
      1. Singly Linked List
2. **DETAILS**
   1. There won't be as much detail on the singly linked list since we went over an implementation in class.
   2. Classes and Adjustments to Lab 3.
      1. **Node** - This is a new class that needs to be added to Lab 3.
         1. Instance variables:
            1. xPosition - int type.
            2. yPosition - int type.
            3. nextNode - Node type.
      2. **LinkedList -** This class will hold the linked list.
         1. Instance variables:
            1. headNode - Node type.
         2. Methods:
            1. addNode

Takes in two parameters, x and y, which are integer types.

Returns nothing.

With the x and y params, it creates a new node object and sets the nextNode on the new node object to the existing headNode object.

It then assigns this new node object to the headNode variable.

* + - * 1. removeNode

Takes in no parameters.

This is tricky so you need to carefully think through how to do this, but the following is what it needs to accomplish:

It needs to change the current head node to the node that is in the head node's nextNode variable; and it needs to return that previous head node to the method caller.

Essentially, you're taking the head node out of the linked list and returning it and setting the next node in line as the new head node.

* + - * 1. getHeadNode

Takes no parameters.

Simply return head node.

* + 1. **Updating run method in GameGrid class.**
       1. You need to instantiate a LinkedList instance.
       2. In the while loop, use the iUserRow and iUserCol values to call the addNode method on the LinkedList instance to add a node for every user move.
       3. Create second while loop: AFTER the existing while loop exits but BEFORE the grid print out.
          1. This loop will run until all LinkedList nodes have been removed & processed.
          2. In the loop, **call the removeNode method on the LinkedList instance** and use the returned node's xPosition and yPosition to update the grid at that position with any number you choose besides 0 and 1. **DO NOT use the nextNode prop** of each node to traverse the linked list, because the whole point of using the remove node method is to take it out of the list once it’s been processed.
          3. You can also use this loop to **count the number of player moves**.
       4. Print out the total number of player moves at the end **AFTER** printing the grid.
       5. Right after the above total player move print out…

**Add ALL the following lines of code** (this is just for me and helps me in code review)

**- Insert this code** at the end of the game just after you print out player move total above.

**- Adjust**, only if needed, the **oLinkedList** variable name below in the first line of the code to whatever you named your LinkedList variable if something different than oLinkedList.

LinkedList oTestLL = **oLinkedList**; // change oLinkedList if you called it something else in your code.

try{ oTestLL.addNode(124,0);System.out.println("\*\*\*test-add-" + oTestLL.getHeadNode().xPosition);}

catch(Exception ex){System.out.println("###test-add");}

try{if(oTestLL.removeNode().xPosition == 124){System.out.println("\*\*\*test-remove");}

else{throw new Exception();}}catch(Exception ex){System.out.println("###test-remove");}

try{int i=oTestLL.removeNode().xPosition;System.out.println("###test-empty");}

catch(Exception ex){System.out.println("\*\*\*test-empty");}

* + 1. **Submitting a SCREENSHOT.**
       1. Lastly, **TAKE A SCREENSHOT** after you run it so I can see the print out of the last few lines in the console at the bottom.
          1. NOTE: It doesn’t have to show all the print out from the game, just most importantly the print out of the player move count and the lines I print out above in the that code of mine that you’re inserting.
          2. NOTE 2: **IF YOU CAN’T RUN your app, then no need to do the screenshot**.
          3. NOTE 3: In Windows, you can use the “**snipping tool**” (included with Windows)… easiest way to get a screenshot.

1. **TURNING IN LAB:** 
   1. Turn your lab into Canvas.
   2. IMPORTANT: DO NOT ZIP your folders/files please.
   3. Turn in **ALL** the java files for this lab (this also includes all the Lab 3 files you used in this lab),

**AND DON’T FORGET to turn in the screenshot file** if you did that.