Homework 07 – LinkedList

Problem Description

Hello! Please make sure to read all parts of this document carefully.
In this homework assignment, you will be developing the Singly-Linked LinkedList data structure, which will implement the provided List interface. In doing so, you will write a variety of methods from each.

You will also be creating your own generic Node class that your LinkedList will use. The provided interface and the classes you create will be written using generics such that they could work for any class parameter - you will be facing many of the challenges that the developers who wrote the java.util.LinkedList class had to confront, as that class uses generics, too!

List of covered topics:

- Generics
- Singly Linked Lists & Operations
- Exceptions

Remember to test for Checkstyle Errors and include Javadoc Comments!

Solution Description

You will create two classes: `LinkedList.java` and `Node.java`. The LinkedList will consist of nodes linked to each other with pointers. `LinkedList.java` will implement the provided List interface. Using the abstract methods provided in the interface, you will have to implement these methods and adjust variables and pointers accordingly. To make these decisions, you should carefully follow the guidelines and logic as taught in lecture. Your program might function for base cases but not handle edge cases appropriately, ***so test your code extensively.***

`Node.java`:

- Represents the nodes which will make up the LinkedList. This is a generic class! Be sure to reflect the use of generics in the class declaration.

Variables
Do not create any other instance variables than specified below. Any extra instance variables will result in deductions. All variables must follow the rules of encapsulation.

A variable named `data`

- Variable of the generic type holding the data stored in the node.

A variable named `next`

- Variable of the type `Node` (with generic type attached on) that acts as a "next" pointer, representing the Node that is next in the LinkedList

Constructors

- A constructor that takes in two arguments **exactly in the given order**: `data` and the `next` node and assigns it to instance variables accordingly.
- A constructor that takes in one argument: `data`
    - The next pointer is set to `null`
    - *Should use constructor chaining*

Methods

Do not create any other methods than those specified. Any extra methods will result in point deductions. All methods must have the proper visibility to be used where it is specified they are used.

- Getters and setters for the instance variables

`LinkedList.java:`

- Represents a LinkedList comprised of `Nodes`. This is a generic class! Be sure to reflect the use of generics in the class declaration.

Interface:

- This class should implement the provided `List<T>` interface

Variables

Do not create any other instance variables than specified below. Any extra instance variables will result in deductions. All variables must follow the rules of encapsulation.

A variable named `head`

- Variable of type `Node` (with generic type attached)
- This variable represents the head of the `LinkedList`
- If the list is empty, set this to `null`
- **You must name this variable `head`**

A variable named `tail`

- Variable of type `Node` (with generic type attached)
- This variable represents the tail of the `LinkedList`
- If the list is empty, set this to `null`
- **You must name this variable `tail`**

A variable named `size`

- Variable of type integer
- This variable represents the current size of the `LinkedList`
- If the list is empty, set this to 0
- **You must name this variable `size`**

Constructors

- A no-args constructor setting `head` and `tail` to null

Methods

Helper methods are discouraged because a proper LinkedList should not require them. Any extra methods will result in a 5 point deduction for code style. All methods must have the proper visibility to be used where it is specified that they are used.

- A getter for the head instance variable named `getHead`
- A getter for the tail instance variable named `getTail`
- You must override all necessary methods to correctly implement `List<T>` interface

Provided:

`List.java:`

- An interface representing a List using generics. **This is provided to you.**

Methods:

The classes that implement List interface must implement the following methods:

- `addAtIndex(T data, int index)`
    - Adds a node to the position specified by `index`

    - If the index is negative or larger than the size of the list, throw `IllegalArgumentException` with a message "Your index is out of the list bounds"
    - If the passed data is `null`, throw `IllegalArgumentException` with a message "You cannot add null data to the list"
    - Adjust for head, tail, and size variables accordingly
- `getAtIndex(int index)`
    - Returns the data located at the specified index in the list
    - If the index is negative or larger than the size of the list minus 1, throw `IllegalArgumentException` with a message "Your index is out of the list bounds"
- `removeAtIndex(int index)`
    - Removes the data (and the node that stores it) from the specified index of the list and returns that data of the node that was removed
    - If the index is negative or larger than the size of the list minus 1, throw `IllegalArgumentException` with a message "Your index is out of bounds"
    - Adjust for head, tail, and size variables accordingly
- `remove(T data)`
    - Removes the first occurrence of the passed data from the list (and also remove the node that holds it) and returns the data from the removed node
    - If the passed data is not in the list, throw `NoSuchElementException` with a message "The data is not present in the list."
    - If the passed data is `null`, throw `IllegalArgumentException` with a message "You cannot remove null data from the list"
    - Adjust for head, tail, and size variables accordingly
- `clear()`
    - The method clears the LinkedList
    - There is a way to do this without iterating through the whole list (which would be O(n)). You won't get points off for having an inefficient solution, but keep in mind that java has Garbage Collecting so take full advantage of it when you can!
- `isEmpty()`
    - Returns a boolean value which represents whether the list is empty
- `size()`
    - Returns current size of the list

Rubric

[15] `Node.java`

- Correct implementation and instance variables

[85] `LinkedList.java`

- [5] Correct constructor and instance variables
- [80] Correct implementation of List<T> methods

We reserve the right to adjust the rubric, but this is typically only done for correcting mistakes.

Allowed Imports

To prevent trivialization of the assignment, you are only allowed to import `java.util.NoSuchElementException`

Feature Restrictions

There are a few features and methods in Java that overly simplify the concepts we are trying to teach or break our auto grader. For that reason, do not use any of the following in your final submission:

- `var` (the reserved keyword)
- `System.exit`

Javadoc

For this assignment, you will be commenting your code with Javadoc. Javadoc comments are a clean and useful way to document your code's functionality. For more information on what Javadoc comments are and why they are awesome, the online overview for them is extremely detailed and helpful.
You can generate the Javadoc overview for your code using the command below, which will put all the files into a folder named "javadoc". Note you should execute this after adding Javadoc comments to your code:

```
$ javadoc *.java -d javadoc
```

The relevant tags that you need to include are `@author`, `@version`, `@param`, and `@return`. Here is an example of a properly Javadoc'd class:

```
import java.util.Scanner;

/**
 *This class represents a Dog object.
 *@author George P.Burdell
 *@version 1.0
 */
 public class Dog {

    /**
     *Creates an awesome dog(NOT a dawg!)
     */
     public Dog() {
     ...
     }

     /**
      *This method takes in two ints and returns their sum
      *@param a first number
      *@param b second number
      *@return sum of a and b
      */
      public int add(int a, int b) {
      ...
      }
}
```

A more thorough tutorial for Javadoc Comments can be found here.
Take note of a few things:

1. Javadoc comments begin with `/**` and end with `*/`.
2. Every class you write must be Javadoc'd and the @author and @verion tag included. The comments for a class should start with a brief description of the role of the class in your program.
3. Every non-private method you write must be Javadoc'd and the `@param` tag included for every method parameter. The format for an @param tag is `@param <name of parameter as written in method header> <description of parameter>`. If the method has a non-void return type,

include the `@return` tag which should have a simple description of what the method returns, semantically.

Checkstyle can check for Javadoc comments using the -a flag, as described in the next section.

Checkstyle

For this assignment, we will be enforcing style guidelines with Checkstyle, a program we use to check Java style. Checkstyle can be downloaded here.
To run Checkstyle, put the jar file in the same folder as your homework files and run

```
java -jar checkstyle-6.2.2.jar -a *.java
```

The Checkstyle cap for this assignment is **35 points**. This means that up to 35 points can be lost from Checkstyle errors.

Collaboration

Collaboration Statement

To ensure that you acknowledge a collaboration and give credit where credit is due, **we require that you place a collaboration statement as a comment at the top of at least one .java file that you submit**. That collaboration statement should say either:
I worked on the homework assignment alone, using only course materials.
or
In order to help learn course concepts, I worked on the homework with [give the names of the people you worked with], discussed homework topics and issues with [provide names of people], and/or consulted related material that can be found at [cite any other materials not provided as course materials for CS 1331 that assisted your learning].

Allowed Collaboration

When completing homeworks for CS1331 you may talk with other students about:

- What general strategies or algorithms you used to solve problems in the homeworks
- Parts of the homework you are unsure of and need more explanation
- Online resources that helped you find a solution
- Key course concepts and Java language features used in your solution

You may **not** discuss, show, or share by other means the specifics of your code, including screenshots, file sharing, or showing someone else the code on your computer, or use code shared by others.
Examples of approved/disapproved collaboration:

- **approved**: "Hey, I'm really confused on how we are supposed to implement this part of the homework. What strategies/resources did you use to solve it?"
- **disapproved**: "Hey, it's 10:40 on Thursday... Can I see your code? I won't copy it directly I promise"

In addition to the above rules, note that it is not allowed to upload your code to any sort of public repository. This could be considered an Honor Code violation, even if it is after the homework is due.

Turn-In Procedure

Submission

To submit, upload the files listed below to the corresponding assignment on Gradescope:

- LinkedList.java
- Node.java

Make sure you see the message stating "HW07 submitted successfully". From this point, Gradescope will run a basic autograder on your submission as discussed in the next section.
You can submit as many times as you want before the deadline, so feel free to resubmit as you make substantial progress on the homework. We will only grade your last submission - be sure to **submit every file each time you resubmit**.

Gradescope Autograder

For each submission, you will be able to see the results of a few basic test cases on your code. Each test typically corresponds to a rubric item, and the score returned represents the performance of your code on those rubric items only. If you fail a test, you can look at the output to determine what went wrong and resubmit once you have fixed the issue.
The Gradescope tests serve two main purposes:

- Prevent upload mistakes (e.g. non-compiling code)
- Provide basic formatting and usage validation

In other words, the test cases on Gradescope are by no means comprehensive. Be sure to thoroughly test your code by considering edge cases and writing your own test files. You also should avoid using Gradescope to compile, run, or checkstyle your code; you can do that locally on your machine.
Other portions of your assignment can also be graded by a TA once the submission deadline has passed, so the output on Gradescope may not necessarily reflect your grade for the assignment.

Important Notes (Don't Skip)

- Non-compiling files will receive a 0 for all associated rubric items
- Do not submit `.class` files
- Test your code in addition to the basic checks on Gradescope
- Submit every file each time you resubmit
- Read the "Allowed Imports" and "Restricted Features" to avoid losing points
- Check on Piazza for a note containing all official clarifications