Homework 9

CS 1331 - Summer 2018

Written by Joshua Dierberger

Problem Description

For this homework, you will be implementing the List interface provided in the Java API. *Do not panic*! You don't have to implement all the methods there, just some.

Your list implementation, which will be called "RecursiveLinkedList", will be a doubly-linked LinkedList which makes use of recursion. Any methods which utilize iteration will receive no credit. **Your list must also support null elements**; that is to say that adding, removing, etc. with nulls should be permitted and should not throw any exceptions.

LinkedLists

ArrayLists are expensive data structures. Resizing is expensive and takes a long time, adding at the front also takes a long time, and they are memory-inefficient. To solve this problem, three really smart guys created the LinkedList. LinkedLists store their data in *nodes*, each node references the node before it and the node after it. If no node exists, the reference is null. By keeping track of the 1st node (a.k.a the *head*), we can repeatedly get the next or previous node to traverse the list. For example, going next from the head 3 times gets the 3rd element.

Recursion

Loops are pretty cool, but they can be messy to use at times. Certain problems rely on solving smaller versions of themselves; for example, to compute the nth Fibonacci number, you need to compute the n-1th and n-2th Fibonacci number and add them together. These problems are known as *recursive* problems. All recursive problems can be solved using while loops, and all while loops can be made into recursive programs.

In Java, recursion involves a method calling itself. In order to prevent it from calling itself infinitely and throwing a StackOverflowError, we also need a *base case*. The base case in Fibonacci numbers is when you get to the 0th or 1st Fibonacci number, which are defined to be 1, so we don't need to recurse any farther.

Solution Description

You have been provided with two files for your solution. The first, “RecursiveLinkedList.java”, is a skeleton file for your RecursiveLinkedList. You must use the head variable provided as the head for your linked list and you must keep the no-args constructor given, although you may change the constructor body. The second file, “LinkedListNode.java”, is the linked-list node which you will use for this assignment. It stores an element as a linked list node. Its hashCode, toString and equals just use the respective methods from the element it is storing. You should use LinkedListNode for your LinkedList.

In this assignment you will have to **“stub”** certain methods in your LinkedList. This just means you won’t be implementing them. The API has a specific way you must do this, and that is to throw an UnsupportedOperationException. **You must do this for any methods you choose not to do!** If you stub a method, you receive no credit for it.

For this assignment, you are to defer to the API. If the API is confusing, contradicts the assignment, or is ambiguous, feel free to ask, but questions which are answered in the API will be redirected to the API.

Methods Rubric

You will be implementing the following methods, each worth the given amount of points. **Boldface** methods should be implemented recursively. This PDF only details specific instructions not given in the API for one method; you should read the API for List before doing this assignment. We will tell you to stub some methods which the API requires; if the PDF contradicts the API follow the PDF. Specific subcategories are given for items. (For example, add is scored based on two five-point items for a total of ten points):

- **[15] add**
  - [10] Adds non-null elements
  - [5] Adds null elements
- [5] clear
  - [5] Implemented without recursion or iteration
- [10] contains
  - [10] Successfully checks for the given element (null or non-null)
- [5] equals
  - [1] Can check equality against other RecursiveLinkedLists.
  - [4] Can check equality against any other Lists (recursive.equals(list), not other way around)
- [5] toString

    - Should look like “[get(0), get(1), …,get(size() - 1)]”. Basically all the elements should appear sequentially, separated by commas, inside square braces with no commas at the end.
    - Try printing an actual LinkedList to see what we mean.
- [5] hashCode
    - [5] Makes use of the formula the API gives for hashCode
- [10] indexOf
    - [5] Correctly gets the indexOf the given element
    - [5] Returns -1 otherwise
- [10] iterator
    - [5] Correct (includes throwing NoSuchElementException when end is reached)
    - [5] Efficient (no recursion in iterator methods)
    - Iterator methods do not need to be recursive
- [10] get
    - [5] Correctly gets normally
    - [5] Throws IndexOutOfBoundsException when the index is out of bounds
- [5] size
    - [5] Implemented without recursion or iteration
- **[10] remove(Object)**
    - [5] Correctly removes if the list contains the element
    - [5] Returns false otherwise
- [5] toArray()
    - [5] Correctly returns the list as an array
- [15] toArray(T[])
    - [5] Correctly fills arrays of equal size AND makes new array if input array is smaller
    - [5] Only sets the first unused index to null if the provided array is larger
    - [5] Throws ArrayStoreException if the input array is incompatible AND throws NullPointerException if the array is null
- **[10] retainAll**
    - [5] Correctly retains all elements
    - [5] Throws NullPointerException if the argument is null
- **[10] lastIndexOf**
    - [5] Correctly gets the last index of the given element
    - [5] Returns -1 if the element is not present
- [10] addAll(int, Collection)
    - [5] Correctly adds all elements in the sequence at the provided index
    - [5] Throws IndexOutOfBoundsException when the index is out of bounds AND NullPointerException if the given Collection is null
- [10] Implement every method recursively (when not specified otherwise)
    - All or nothing

Additional Rubric Items

- You may have noticed there is a lot to do. That's because there are **50 points of extra credit** (1% of your final grade) incorporated into this assignment!
- [-25] Flat penalty for any RuntimeExceptions not detailed in the API. This includes unexpected NullPointerExceptions and StackOverflowErrors
    - [-15] Flat penalty for broad catch statements or catching StackOverflowErrors
- [-10] Flat penalty for incorrectly stubbing any methods
- [-20] Flat penalty for modifying head, or deleting the no-args constructor
- [-5] Nonsensical/Bad Exception messages
    - Good message: index + " is out of bounds in list of size " + size()
    - Bad message: "Index out of bounds"; "gjiasf"; "null"
    - For UnsupportedOperationException you do not need a message
- You will receive a 0 if your submission does not compile.
- [-25] Using block control statements (break, continue, goto, labels, etc.) You may use break in switch statements only.
- [-10] Additional non-private fields or methods.
- You will not receive any credit if your recursive methods use iteration.
- You may not use anything in java.lang.reflect, nor any imports which trivialize this assignment. Doing so can incur hefty penalties. If you are unsure, ask a TA.

Testing Your Assignment

A tester has been provided to you which checks the following items:

- Illegal modifications (should only have head, must have no args constructor, file compiles & implements list)
- Incorrect method stubbing for methods which are required to be stubbed
- Expected exceptions (ClassCast, IndexOutOfBounds, etc.) are thrown when expected
    - This does not test to ensure they are not thrown abnormally

We've also provided a tester template which you can use to write testers without having to add additional getters and setters for the head of the linked list. Feel free to write and share your own on Piazza!

Hints and Tips

- Although you can choose which methods you wish to implement, these four methods were selected with the idea that they would be harder: toArray(T[]), addAll(int, Collection), lastIndexOf and retainAll.

- Start on this assignment sooner rather than later! Some of the non-extra credit methods are tricky when you have to do them recursively, and if you're not comfortable with recursion you should give yourself ample time to think.
- If you're finding a method conceptually hard, it may help to write it as a while-loop first and then to de-loop it.
- You can add as many private variables as you like as long as they don't violate other rules given.
- If you're struggling with the concept of LinkedLists or recursion, stop by the TA Lab! We're always happy to help (ﾉ◕ヮ◕)ﾉ*:･ﾟ✧

Javadocs

- You will need to write Javadoc comments and watch for checkstyle errors with your submission.
- Every class should have a class level Javadoc that includes `@author <GT Username>` and `@version <version number>`.
- Every public method should have a Javadoc explaining what the method does and includes any of the following tags if applicable:
    - @param <parameter name> <brief description of parameter>
    - @return <brief description of what is returned>

See the CS 1331 Style Guide on Canvas for details.

Checkstyle

For each of your homework assignments we will run checkstyle and deduct one point for every checkstyle error, with the points deducted being capped.

For this homework the checkstyle cap is 100, meaning you can lose up to 100 points on this assignment due to style errors. This limit obviously cannot increase further.

- If you encounter trouble running checkstyle, check Piazza for a solution and/or ask a TA as soon as you can!
- You can run checkstyle on your code by using the jar file found on Canvas that includes xml configuration file specifying our checks. To check the style of your code run java -jar checkstyle-6.2.2.jar *.java.
- To check your Javadocs run java -jar checkstyle-6.2.2.jar -j *.java.
- Note that the command for checking code and the command for checking Javadocs are different.
- You will have to run both commands to fully test for style errors.
    - Instead of "-j" you can say "-a" to automatically run them both.
- Javadoc errors are the same as checkstyle errors, as in each one is worth a single point and they are counted towards the checkstyle cap.

- You will be responsible for running checkstyle on ALL of your code.
- Depending on your editor, you might be able to change some settings to make it easier to write style-compliant code. See the Customization Tips on Canvas for more information.

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 homework
- Parts of the homework specification 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

For this homework as well, you may upload tester files to Piazza.

<u>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.</u>

<u>Examples of approved/disapproved collaboration:</u>

OKAY: "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?"

BY NO MEANS OKAY: "Hey... the homework is due in like 20 minutes... Can I see your code? I promise won't copy it directly!"

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.

Submission

- Submit RecursiveLinkedList.java for this homework (you do not need to submit the Tester or LinkedListNode) as attachments to the Homework 9 assignment on Canvas. You can submit as many times as you want, so feel free to submit as you make substantial progress on the homework. We only grade your last submission, meaning we will ignore any previous submissions.
- Submissions are due at 10:55 but **will not be penalized up to 11:55 PM**. After 11:55 PM no submissions will be accepted.
- If you submit multiple times Canvas will append a number to your zip file (RecursiveLinkedList.java becomes RecursiveLinkedList-1.java). **Do not worry about this, we will fix the file name before compiling and running your code.**

- Non-compiling code will be given a score of 0. For this reason, we recommend submitting early and then confirming that you submitted ALL of the necessary files by re-downloading your file(s) and compiling/running them.
- Files that contain inappropriate language, profanity, or otherwise inflammatory or unprofessional statements will receive a 0.

Here’s to your last CS1331 homework. Good Luck! \(°□°)/