L17-L18 Homework: Linked List

Problem Description

Hello! Please make sure to read all parts of this document carefully.

In this assignment, you will be applying your knowledge of generics and linked lists to write your own generic LinkedList class.

Solution Description

You will write a generic LinkedList class with a private inner generic Node class, both in LinkedList.java.

- Type parameters: The type parameter for both classes will be E
- Visibility for required elements:
  - Node and all its elements must be private
  - All variables of LinkedList must be private
  - LinkedList and all its methods must be public
- The throws declaration in method headers should include checked exceptions only (in this HW, that means no method should use it)
- You may add to aid you (if you want)
  - Methods, if they are private
  - Variables, if they are private static final, and their type is a primitive type, a wrapper class, or String.
- You may not add any other classes or constructors
- For all methods that throw an IndexOutOfBoundsException exception, they should do it with the following message: "Index out of bounds: [index]" (without square brackets)

Note: your LinkedList should NOT implement List<E>. While that is what you'd need to do to properly make your class a List in Java, that would require you to code too many methods, including some for which we don't cover the required topics.

Note: to ensure you're properly using generics, your code must compile without warnings (mainly with no `rawtypes` and `unchecked` warnings, but we will check for all warnings except `overrides` warnings) to receive credit. Ensure that the code compiles with the following command:

- `javac -Xlint:all,-overrides -Werror LinkedList.java`

LinkedList<E>

- This is a class to manage data in a linked list (values may be `null`). It will have:
  - A private generic inner `Node<E>` class, with `data` and `next` variables and a 2-arg constructor to create the nodes.
  - The following instance variables `head`, `size (int)`. The size should always reflect the number of values (which is the same as the number of nodes) in the linked list, so any method that changes how many nodes there are should also update `size`.
  - A 0-arg constructor setting `head` to `null` and `size` to `0`. You can also omit the constructor code and use the default constructor.
  - A `int size()` method to return the size of the linked list.
  - A `boolean isEmpty()` method that returns whether the linked list is empty or not.
  - A `void clear()` method that removes all data from the linked list.
    - Tip: the implementation body should be 2 lines of code, regardless of how many nodes there are present. Remember that Java is a garbage-collected language, and you don't need to deallocate nodes manually
  - A `void add(int, E)` to add a value at a specific index of the linked list. The method should throw an `IndexOutOfBoundsException` if the value cannot be added to the requested index.
  - A `void add(E)` to add a value at the end of the linked list.
    - Tip: the implementation body should be a single line of code. Think how you can use `add(int, E)`
  - `boolean contains(Object)` method that returns whether the value is in the linked list.
    - Note: Be careful with `null` values, and compare values by equality
  - `E get(int)` method that returns the value at the index. The method should throw an `IndexOutOfBoundsException` if the requested index does not exist in the linked list.
  - `int indexOf(Object)` method that returns the index of the first occurrence of the specified element in this linked list, or -1 if it is not present.
    - Note: Be careful with `null` values, and compare values by equality
  - `E remove(int)` method to remove and return a value at the index. The method should throw an `IndexOutOfBoundsException` if the requested index does not exist in the linked list.
  - `boolean remove(Object)` method that removes the first occurrence of a value in the linked list. Returns true if the value was present at least once (and was removed), and false otherwise.
    - Note: Be careful with `null` values, and compare values by equality
  - `E set(int, E)` method that updates the value at an index and returns the old one. The method should throw an `IndexOutOfBoundsException` if the requested index does not exist in the linked list.
  - A `toString` method that returns a representation of the list. The representation will include the String representation of each element (or the text "null" without double quotes for `null` values – see `String.valueOf` method), in a comma-separated separated (with a space after each comma) and surrounded with square brackets. For example (without the double quotes)
    - No elements: "[]"
    - One element "[1]"
    - Two elements: "[1, 2]"
  - An `equals` method:

- IMPORTANT: As this method requires some concepts of generics we don't cover, the implementation is OPTIONAL, and its tests will not be worth any points. However, the method must still be present in your class (feel free to always return `true` to skip implementing the method)
- Unlike other `equals` methods, coding an `equals` method for a data structure usually involves iteration. In this case, we need to ensure that we perform a comparison between each node and its counterpart in the parameter LinkedList (provided the parameter is a LinkedList)
- Because of **type erasure**, one cannot decide at runtime if the passed parameter is a `LinkedList<E>`, only if it's a `LinkedList`. Because of that:
    - Your `instanceof` should check against `LinkedList`, not `LinkedList<E>` (you can try adding the generics, you'll notice you will get a compiler error)
    - If it is a `LinkedList`, you should downcast and store the value with the type `LinkedList<?>`, which is a wildcard type used to indicate "LinkedList with unknown type parameter"
    - To store a reference of nodes of the parameter linked list, you can use the type `LinkedList<?>.Node<?>`, which is to represents nodes of unknown types, which themselves belong to a linked list of unknown type.
    - Check that the size of the two linked lists matches, and if so, check that for each Node<E>, the data is equal to the one in its counterpart Node<?> (be careful with `null` values)

Testing Your Solution

You will notice that none of the classes have a `main` method. That is because not all classes have to be Java programs, and in this case, none are. To help test your implementations, you can create more classes in which you can create instances of your concrete classes, invoke methods to have them interact with each other, and print to console information that helps verify the correctness of your solution.

Rubric

[25] Type Review (it is very important for your submission that these tests pass. All of these are visible in every submission)

- [5] All types compile, all type headers are correct, all static variables are correct
- [10x2] Check variables, methods, constructors in each student-written class

[3] Node constructor implementation
[72] LinkedList implementations

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 not allowed to import any packages.

Feature Restriction

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

- **var** (the reserved keyword)
- **SuppressWarnings** (annotation)

Checkstyle

The Checkstyle deduction limit for this assignment is 35 points, counting Javadoc errors. Refer to the course guide for the use of Checkstyle and review the autograder Checkstyle report on your submissions.

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].

If the autograder encounters issues processing your collaboration statement, begin it with "Collaboration Statement: ".

Allowed Collaboration

When completing homeworks for CS 1331 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

Upload the files listed below to the corresponding assignment on Gradescope:

- LinkedList.java

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.

<u>Important Notes</u>

- Non-compiling files will receive a 0 for all associated rubric items
- Do not submit .class files or .jar 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