Homework 08 – Linked List

Authors: Eric, Srujal, Joshua, Vinayak, David
Topics: generics, ADTs, Iterator, Iterable

Problem Description

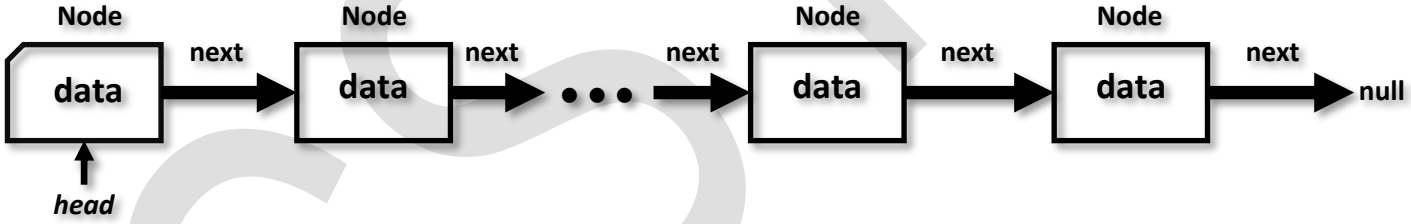
You will create an implementation of a linked list, one of the fundamental data structures in computer science! Under the hood, your operating system, and countless everyday applications rely upon these.

Solution Description

We highly recommend that you **read this entire document in its entirety** and familiarize yourself with the requirements of this assignment **before writing any code**!

Important: All classes created in this assignment should be **generic classes**. All instances of generic classes should specify a concrete type and should never use raw types or Object. If a method returns a reference type, it must use generic types, and should never use raw types or Object.

A linked list is an ordered sequence of nodes. Each node stores data and a reference to the next node in the sequence. The first node is called the head and a reference to it should be kept. This is the "entry point" to the linked list. When there are no nodes in the list, the head should be *null.*



We have provided you with a file named `List.java`. This file is an interface named List that you will implement in LinkedList.java. It represents the List ADT and extends from the Iterable interface. You will create three classes: `Node.java`, `LinkedListIterator.java`, and `LinkedList.java`.

Note:

1. All variables should be inaccessible from other classes and must require an instance to be accessed through, unless specified otherwise.
2. All methods should be accessible from everywhere and must require an instance to be accessed through, unless specified otherwise.
3. Reuse code when possible and private helper methods are optional.
4. When throwing exceptions, a descriptive and specific message should be provided.
5. Add Javadoc comments to all public methods and classes, including in provided files where Javadoc comments are missing.

THIS GRADED ASSESSMENT IS NOT FOR DISTRIBUTION.
ANY DUPLICATION OUTSIDE OF GEORGIA TECH'S LMS IS UNAUTHORIZED.

Tip from your TAs: There are a lot of edge cases to consider when completing this assignment. Make sure to perform extensive testing in a Driver file! We encourage sharing your tests and outputs in the Discussion Thread on Ed Discussion.

List.java (Provided File)

This generic interface defines the List ADT and describes the methods that the LinkedList class must implement. The Javadocs in this file will contain more information about each of the methods you must implement, including what the method does, what should be returned, and when to throw exceptions.

This interface extends from the `Iterable` interface, so the `iterator` method must also be implemented. Refer to the Java documentation on **Iterable** for more information about this interface.

Methods: ***In addition to*** the methods required by the Iterable interface, the List interface requires the following methods:

- `void add(E element)`
- `void add(int index, E element)`
- `E remove()`
- `E remove(int index)`
- `E remove(E element)`
- `E set(int index, E element)`
- `E get(int index)`
- `boolean contains(E element)`
- `void clear()`
- `boolean isEmpty()`
- `int size()`

Note: You do not need to implement the `forEach` and `spliterator` methods of the Iterable interface for this assignment.

Node.java

This generic class represents an individual node. It should be able to store data of any type.
Note: You **may not** create any instance fields nor instance methods that are not required.

Type Parameter:

- `T` – the type of data stored in this node

Variables:

- `data` – the data stored within this node
  - If we ever attempt to set its value to null, throw an `IllegalArgumentException`.
- `next` – a reference to the next node, which should also store the same type of data

Constructors:

- A constructor that takes in `data` and `next`.
- A constructor that takes in `data` and sets `next` to null.

Methods:

- Getters and setters for each of the fields.

THIS GRADED ASSESSMENT IS NOT FOR DISTRIBUTION.
ANY DUPLICATION OUTSIDE OF GEORGIA TECH'S LMS IS UNAUTHORIZED.

LinkedListIterator.java

This is a generic class that represents an iterator to iterate over a LinkedList. It implements the Iterator interface using generics. Refer to the Java documentation on **Iterator** for more about this interface.

For LinkedList to implement the Iterable interface (which the provided List interface extends from), it needs to provide an implementation for the `iterator` method in the `LinkedList` class. The `iterator` method in the `LinkedList` class should return an instance of a `LinkedListIterator` that knows how to iterate over the linked list to retrieve its elements.

To successfully implement the `iterator` method, we will create our own iterator, LinkedListIterator. An iterator is an object that enables iteration over objects in a collection that you create, such as our linked list. A LinkedListIterator should iterate over all the nodes in the linked list, and in the same order as the elements.
Note: You **may not** create any instance fields nor instance methods that are not required.

Type Parameter:

- `T` – the type of data stored in the node that this iterator iterates over

Variables:

- `next` – the next node to iterator over
  - ***Hint*:** When a `LinkedListIterator` is instantiated, `next` should store a reference to the first node car in the linked list (the head of the linked list).

Constructor:

- A constructor that takes in the `LinkedList` that this iterator will iterate over.
  - If the linked list passed in is null, throw an `IllegalArgumentException`.

Methods:

- `hasNext`
  - This method overrides the `hasNext` method required by `Iterator`.
  - Returns a boolean specifying whether there are more nodes to iterate over.
    - ***Hint***: What value of `next` indicates the existence of a node?
- `next`
  - This method overrides the `next` method required by `Iterator`.
  - Returns the next `data` in the iteration and advances the iterator.
    - **Note**: The return type of this method should **not** be `Node`, but instead the type of `data` in the node.
  - ***Hint:*** The first time the `next` method is called, it should return the data in the first node of the linked list. The second time `next` is called, it should return the data in the second node of the linked list, and so on.
  - If there are no more nodes to iterate over, throw a `NoSuchElementException`.

Functionality:

- After properly implementing `LinkedListIterator`, you should be able to use the iterator to print out all the data in a `LinkedList` using this block of code. Assume that `list` is an instance of `LinkedList`, and that `T` is replaced with a concrete type.

```
Iterator<T> itr = list.iterator();
while (itr.hasNext()) {
    System.out.println(itr.next());
}
```

THIS GRADED ASSESSMENT IS NOT FOR DISTRIBUTION.
ANY DUPLICATION OUTSIDE OF GEORGIA TECH'S LMS IS UNAUTHORIZED.

LinkedList.java

This is a generic class that represents a linked list. A `LinkedList` is comprised of nodes that are *linked* together. This class implements all the methods required by the provided `List` interface (don't forget about the methods required by the `Iterable` interface). This class should utilize the `Node` and `LinkedListIterator` classes to satisfy the requirements.

Type Parameter:

- `T` – the type of data stored in this linked list

Variables:

- `head` – a reference to the first node in the linked list
  - ***Hint:*** Think about which part of a linked list this refers to.
- `size` – the number of data in the linked list
  - Don't forget to update this value when a node is added or removed from the linked list!

Constructors:

- A constructor that takes in an array of `T`.
  - The order of the data in this linked list must be in the same order as they were in the array.
  - If the input array is null or any elements of the array is null, throw an `IllegalArgumentException`.
- A no-argument constructor that constructs an empty linked list.

Methods:
In addition to the methods required by the List interface, implement the following method:

- `toArray`
  - Return the data in the linked list as an array.
  - You must implement this method by explicitly using an iterator (i.e., by explicitly utilizing `Iterator`'s `hasNext` and `next` methods). Do not use a for-each loop.
  - The order of the data in the array must be the same order as the data in this linked list.
  - **Note:** The intuitive approach of creating a generic array in Java by writing `T[] arr = new T[size];` causes a compiling error. Instead, create an Object array and cast that to a generic array: `T[] arr = (T[]) new Object[size];`
    - Note that this cast is not checked by the compiler, so it will display a warning (not an error) to notify you of an unchecked or unsafe operation. It is acceptable if the only warnings you receive are related to performing this unchecked cast.
    - You can recompile and use the `-Xlint` flag before the file name to see more specific details about the warnings.

Checkstyle

You must run Checkstyle on your submission (to learn more about Checkstyle, check out cs1331-style-guide.pdf under the Checkstyle Resources module on Canvas). **The Checkstyle cap for this assignment is 45 points.** This means there is a maximum point deduction of 45. If you don't have Checkstyle yet, download it from Canvas → Modules → Checkstyle Resources → checkstyle-8.28.jar. Place it in the same folder as the files you want to run Checkstyle on. Run Checkstyle on your code like so:

```
$ java -jar checkstyle-8.28.jar YourFileName.java
Starting audit...
Audit done.
```

The message above means there were no Checkstyle errors. If you had any errors, they would show up above this message, and the number at the end would be the number of points we would take off (limited by the Checkstyle cap). In future assignments we will be increasing this cap, so get into the habit of fixing these style errors early!

Additionally, you must Javadoc your code.

Run the following to only check your Javadocs:

```
$ java -jar checkstyle-8.28.jar -j yourFileName.java
```

Run the following to check both Javadocs and Checkstyle:

```
$ java -jar checkstyle-8.28.jar -a yourFileName.java
```

For additional help with Checkstyle see the CS 1331 Style Guide.

Turn-In Procedure

Submission

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

- `Node.java`
- `LinkedListIterator.java`
- `LinkedList.java`

Make sure you see the message stating the assignment was submitted successfully. From this point, Gradescope will run a basic autograder on your submission as discussed in the next section. **Any autograder tests are provided as a courtesy to help "sanity check" your work and you may not see all the test cases used to grade your work.** You are responsible for thoroughly testing your submission on your own to ensure you have fulfilled the requirements of this assignment. If you have questions about the requirements given, reach out to a TA or Professor via the class forum for clarification.

You can submit as many times as you want before the deadline, so feel free to resubmit as you make substantial progress on the assignment. We will only grade your latest submission. **Be sure to submit every file each time you resubmit**.

Gradescope Autograder

If an autograder is enabled for this assignment, you may be able to see the results of a few basic test cases on your code. Typically, tests will correspond 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. **We reserve the right to hide any or all test cases, so you should make sure to test your code thoroughly against the assignment's requirements.**

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.

Burden of Testing

You are responsible for thoroughly testing your submission against the written requirements to ensure you have fulfilled the requirements of this assignment.

Be **very careful** to note the way in which text output is formatted and spelled. Minor discrepancies could result in failed autograder cases.

If you have questions about the requirements given, reach out to a TA or Professor via the class forum for clarification.

Allowed Imports

- `java.util.Iterator`
- `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 autograder. For that reason, do not use any of the following in your final submission:

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

Collaboration

Only discussion of the assignment at a conceptual high level is allowed. You can discuss course concepts and assignments broadly; that is, at a conceptual level to increase your understanding. If you find yourself dropping to a level where specific Java code is being discussed, that is going too far. Those discussions should be reserved for the instructor and TAs. To be clear, you should never exchange code related to an assignment with anyone other than the instructor and TAs.

The only code you may share are test cases written in a Driver class. If you choose to share your Driver class, they should be posted to the assignment discussion thread on the course discussion forum. We encourage you to write test cases and share them with your classmates, but we will not verify their correctness (i.e., use them at your own risk).

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 Ed Discussion for a note containing all official clarifications and sample outputs.**

It is expected that everyone will follow the Student-Faculty Expectations document and the Student Code of Conduct. The professor expects a **positive, respectful, and engaged academic environment** inside the classroom, outside the classroom, in all electronic communications, on all file submissions, and on any document submitted throughout the duration of the course. **No inappropriate language is to be used, and any assignment deemed by the professor to contain inappropriate offensive language or threats will get a zero.** You are to use professionalism in your work. Violations of this conduct policy will be turned over to the Office of Student Integrity for misconduct.

THIS GRADED ASSESSMENT IS NOT FOR DISTRIBUTION.
ANY DUPLICATION OUTSIDE OF GEORGIA TECH'S LMS IS UNAUTHORIZED.