Homework 09 – Linked List

Authors: Varun, Calvin, Matthew, Melanie, Allan
Topics: Generics, ADTs, Iterator, Iterable

Problem Description

Please make sure to read the document fully before starting!
This HW is optional and will replace the lowest from the HW category. You have been tasked with creating an implementation for a common abstract data type (ADT), a List. Your implementation should be able to handle any data type (hint: generics). Using your advanced knowledge of data structures, you have decided to use a Linked List to solve this problem.

Solution Description

You will be given an interface `List` representing the List ADT, which extends `Iterable`. Your `LinkedList` class will have to implement this interface. To create the LinkedList, you will also need to create a `Node` class and a custom iterator class, `LinkedListIterator`.

Important: Do NOT return objects of type `Object` or use raw types instead of generic types.
An example of a raw type is `ArrayList` as opposed to `ArrayList<String>`.

`List.java` (provided)

This interface defines the List ADT and will contain methods that your LinkedList class must implement. The JavaDocs in this file will contain more information on the following methods you must implement:

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

Make sure you read the JavaDocs to see information on when to throw exceptions and what to return for edge cases.
This interface also extends the `Iterable` interface, so you must also implement the `iterator()` method. Refer to the Java API documentation for more information on this interface.
Note: You do **NOT** need to implement the `forEach()` and `spliterator()` methods of the `Iterable` interface for this assignment.

`Node.java`

This class will represent an individual node in your LinkedList. Most implementations of a LinkedList node only have two instance variables:

- `data`

    - This holds the actual data stored within a node
    - Since your LinkedList needs to be able to be created for any data type, your Node class needs to be **generic**, and the type of `data` should be the generic type `T`
    - **Note**: You can technically use any letter to denote your generic type. As a convention, we use `T` for "type" in most contexts and `E` for "element" when we are storing objects of a generic type in some sort of collection (such as an array, an ArrayList, or even your LinkedList)
- `next`
    - This will hold a reference to the next node in the LinkedList
    - These references are how the nodes in a LinkedList are connected, each node is connected to only one other node.
    - Since we are implementing a singly linked LinkedList, we don't need a reference to the previous node.

Ensure that you follow the principle of encapsulation and write getters and setters where necessary. You may write constructors to make your code more concise.

`LinkedListIterator.java`

In order to successfully implement the Iterable interface (which the provided List interface extends from), your LinkedList class must provide an implementation for the `iterator()` method. This method returns an instance of an `Iterator` for your data type. To successfully do this, we will create our own Iterator, `LinkedListIterator`, which implements the `Iterator` interface. An iterator is an object that allows you to loop over objects in a custom collection you create, such as our LinkedList. Note that this also needs to be a generic class, since it should iterate over a LinkedList of any type. This class will need to implement the following methods:

- `hasNext()`
    - Returns a `boolean` value specifying whether there are more elements to iterate over
- `next()`
    - Returns the next element in the LinkedList
    - This should return the data in the next node of the LinkedList, not the node itself
    - This method should return an object of the generic type
    - The first time you call next(), it should return the data of the first node of the LinkedList
    - Needs to throw a NoSuchElementException if there is no next element (already iterated through all of the elements).

Hint: What instance variables are needed? What arguments may the constructor take in to be able to iterate over a LinkedList?

`LinkedList.java`

This is the main file for your LinkedList and will utilize the previous two classes to satisfy the requirements of the List ADT. This class will implement all the methods from the List interface (including the iterator() method inherited from the Iterable interface). This class also needs to be generic.

Instance variables:

- `size`
- The number of nodes in your LinkedList=`head`
    - A reference to the head node of the LinkedList

    - Recall that the `Node` class is generic. What should the type of this instance variable be?

Methods:

In addition to the methods from the List interface, implement the following methods:

- Constructors:
    - A no-arguments constructor that initializes an empty LinkedList
    - A constructor that takes in an array of elements of the generic type and creates a LinkedList from those elements
- `removeDuplicates()`
    - This method should go through the list and remove any duplicate nodes (nodes with the same data).
    - After this method is called, each node in the list should have unique data.
    - This method will not return anything
- `toArray()`
    - Return the elements of the LinkedList as an array.
    - **Note**: The head of your LinkedList should be the first element of the array (index 0 of the array) that is returned.
    - **Note**: You must be careful when trying to create a generic array in Java. The intuitive approach of `E[] arr = new E[size]` causes a compile error (the details of why exactly that happens are out of the scope of this homework). Instead, to create a generic array you must create an array of objects and cast that to a generic array, like `E[] arr = (E[]) new Object[size]`

Checkstyle

You must run checkstyle on your submission (To learn more about Checkstyle, check out cs1331-style-guide.pdf under CheckStyle Resources in the Modules section of Canvas.) **The Checkstyle cap for this assignment is 40 points.** This means there is a maximum point deduction of 40. 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 a bove this message, and the number at the end would be the points we would take off (limited by the ch eckstyle cap mentioned in the Rubric section). In future homeworks 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 test 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 Piazza 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 homework. 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.
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.

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 auto grader. 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 Homework (HW) at a conceptual high level is allowed. You can discuss course concepts and HW 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.

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.