Homework 08 – LinkedList

Authors: Brittney, Ariel, Ian, Vince, Tejas

Problem Description

Welcome to your final challenge, er- I mean, final homework! For this homework, you will be writing a concrete implementation of a List ADT--a Singly-Linked List. `LinkedLists` are comprised of `Nodes`, which are objects that store data and have a pointer to the next `Node` in the list, which has its own data and next `Node`, and so on.

Solution Description

We have provided `List.java` (different than the List interface from `java.util` package), an abstract data type (ADT) that contains abstract methods for all of the functionality your `LinkedList` needs to implement. Using your knowledge of **Collections, Generics, and Iterable,** you will write a `Node` class that represents a singular element in a `LinkedList` and a `LinkedList` class that represents a Singly-LinkedList and implements all of the functionality defined in the provided `List<T>` interface. As always, you will need to adhere to proper Checkstyle including Javadocs.

`Node.java`

Your `LinkedList` will consist of `Nodes`. Each `Node` will contain its own data and point to the next `Node` in the list. **This class takes in a generic type parameter** for the type of data the `Node` will hold, so be sure to reflect that in your class declaration, instance data, and methods. Make sure that this is a separate file from your `LinkedList` class.

- Variables
  - `data`
    - This represents the data stored in the `Node`
    - Should be of generic type
  - `Node next`
    - This represents the `Node` that comes next in the list
    - This `Node`'s data type will be of the same generic type as `data`
- Constructors
  - A constructor that takes values in the following order: `data` and `next` and assigns them to the corresponding instance variables.
  - A constructor that takes in a `data` value and sets `next` to `null`.
- Methods
  - Getters and setters for all instance variables.

`LinkedList.java`

This file contains your implementation of a Singly-LinkedList. **This class takes in a generic type parameter** for the type of data each `Node` in the `LinkedList` will hold, so be sure to reflect that in your class declaration, instance data, and methods. Your Singly-LinkedList should store `Nodes` and implement the provided `List<T>` interface

- Variables
  - `Node head`
    - Represents the head of your `LinkedList`
    - If the `LinkedList` is empty, should be `null`

    - `int size`
        - Represents the number of `Nodes` in your `LinkedList`
        - Remember to update `size` accordingly as you add to and remove from the `LinkedList`
- Constructors
    - A no-argument constructor that initializes `head` to `null` and `size` to 0
- Methods
    - Override all methods in the provided `List<T>` interface
        - For all methods that take in parameters you should throw the following Exceptions if the parameters are invalid:
            - ◊ Throw an `IllegalArgumentException` if any parameters are `null`
            - ◊ Throw an `IndexOutOfBoundsException` if an index parameter is invalid
        - Make sure to read the JavaDocs in the `List<T>` interface so you know what functionality to implement.
    - To implement the `iterator()` method you should write an inner class named `LinkedListIterator` and return an instance of it. The `LinkedListIterator` class should contain the following.
        - **This class should not be generic**, but should utilize the generic type from `LinkedList` as needed.
        - This class should implement `Iterator<T>` and override all **non-default** abstract methods in the `Iterator<T>` interface
            - ◊ Make sure to throw a `NoSuchElementException` from the abstract method stated in the `Iterator<T>` API to throw one when an error occurs
        - This class should have one instance variable named `nextNode` of type `Node` to keep track of the next element in the `LinkedList`.
        - This class should have one no-argument constructor that initializes `nextNode` to `head`.
    - Getters for the instance variables.
    - You may also want to create a `toString()` method for testing but we will NOT be grading for this.

Clarifications and Example Output

Please refer to the pinned Piazza thread for HW08 Clarifications for any necessary clarifications about the requirements outlined in this pdf. We may also post some example output that you can use to check your code against on the same Piazza post.

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`

Checkstyle and Javadocs

You must run Checkstyle on your submission. The Checkstyle cap for this assignment is **15 points**. If you don't have Checkstyle yet, download it from Canvas -> Modules -> Checkstyle Resources (all sections). Place it in the same folder as the files you want Checkstyled. 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 points we would take off (limited by the Checkstyle cap mentioned above). The Java source files we provide contain no Checkstyle errors. 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.

Collaboration

No collaboration is allowed on this assignment. See syllabus for more details.

In addition, note that it is not allowed to upload your code to any sort of public repository or website. 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:

- Node.java
- LinkedList.java

Please only submit the individual files—do not put them into a package or you will risk getting a zero on the assignment. Make sure you see the message stating "HW08 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
- Run Checkstyle on your code to avoid losing points
- Submit every file each time you resubmit
- Read the "Allowed Imports" and "Feature Restrictions" to avoid losing points
- Check on Piazza for a note containing all official clarifications