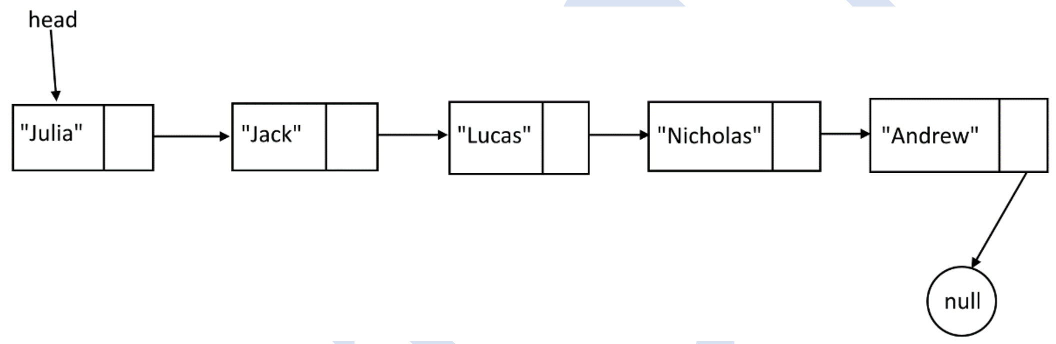
Homework 08 – Singly LinkedList

Authors: Julia Zhu, Nicholas Shi, Lucas Morais, Andrew Chafos, Jack Kelly

Problem Description

In HW08 you will be developing the Singly-LinkedList data structure and will implement methods from the provided `List` and `Stack` interface. A `LinkedList` is a collection of elements often known as nodes, with each node holding its own information and a reference to the next node in the structure. A `Stack` is a collection of elements which follows the LIFO (last in, first out) principle. Elements are added one by one, and you can only access the element at the top of the stack. (Think about how pringles are stacked in a pringle can. The first pringle you eat was the last pringle placed in the can). You will also be writing a generic `Node` class that your `LinkedList` will use. Below is a visual representation of a Singly-LinkedList with nodes of type `String`.



Solution Description

You will write two classes: `Node.java` and `LinkedList.java`. The LinkedList class must implement the provided `List` and `Stack` interfaces, and both classes must utilize generics.

`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 is a generic class, so be sure to reflect that in your class declaration and methods. **Make sure that this is a separate file from your LinkedList class.**

- Variables (all fields below should follow the rules of encapsulation):
  - `data`
    - Represents data stored in the node
    - Should be of generic type
  - `Node next`
    - Represents the node that comes next in the list
  - Do not add any other instance variables
- Constructors:
  - A constructor that takes values in the following order-`data`, `next`. The constructor should initialize the fields of the object with the corresponding values passed in.
  - A constructor that takes in `data` and assigns it to the corresponding instance variable. The constructor should set the next variable to `null`.
  - A constructor's parameter's types should match the types of the fields they are assigning.

- Methods:
    - Getter and setters for each instance variable.

`LinkedList.java`

A Singly-LinkedList of Nodes. This is a generic class, so be sure to reflect that in your class declaration. This class should implement the provided `List<E>` interface and `Stack<E>`.

- Variables (field below should follow the rules of encapsulation):
    - `Node head`
        - Represents the head of your LinkedList
        - If list is empty, head should be `null`
    - Do not add any additional instance variables
- Constructors:
    - A no-args constructor that sets your head to `null`.
- Methods:
    - A getter for the head instance variable.
    - Override all methods in the `List` interface.
    - Override all methods in the `Stack` interface.
    - Stack and List methods are supposed to be separate from each other. For example, we do not call both `push()` and `add()`; we will pick one. We treat the LinkedList as a `List` or a `Stack`, not both. Refer to the example outputs for code reference.

`List.java`

Provided List interface that contains all the methods your LinkedList should implement.

- Methods:
    - `void add(int index, E element)`
        - Adds element to the specified index
        - If the index is negative or index is greater than the size of the list, throw an `IllegalArgumentException` with the message “Your index is invalid”
        - If the passed in element is null, throw a `NullPointerException` with the message “Cannot add null data to list”
    - `boolean remove( element)`
        - Removes the first equal element from the list
        - If the list does not contain the element, return false, otherwise return true
    - `E remove(int index)`
        - Removes the element at the specified index and returns it
        - If the index is negative or index is greater than or equal the size of the list, throw an `IllegalArgumentException` with the message “Your index is invalid”
    - `E get(int index)`
        - Retrieves the element at the specified index and returns it
        - If the index is negative or index is greater than or equal the size of the list, throw an `IllegalArgumentException` with the message “Your index is invalid”
    - `E set(int index, E element)`
        - Replaces the element at the specified index with the passed in data, and then returns the data that was previously at the index
        - If the index is negative or index is greater than or equal the size of the list, throw an `IllegalArgumentException` with the message “Your index is invalid”
        - If the passed in element is null, throw a `NullPointerException` with the message “Cannot add null data to list”

- `boolean contains(E element)`
  - Checks if a list contains the passed in element
  - If the passed in element is null, throw a `NullPointerException` with the message "Null data cannot be in list"
- `void clear()`
  - Removes all elements from the list
- `boolean isEmpty()`
  - Check whether the list is empty or not
- `int size()`
  - Returns the size of the list
  - Size of list must be calculated by iterating through all the nodes; you cannot have a private size variable and return that

`Stack.java`

Provided Stack interface that contains all the methods your LinkedList should implement. The head of the `LinkedList` should be the top of the stack.

- Methods:
  - `E push(E element)`
    - Adds element to the top of the stack
    - If the passed in element is null, throw a `NullPointerException` with the message "Cannot add null data to stack"
  - `E pop()`
    - Removes the element from the top of the stack and returns it
    - If the stack is empty, throw an `IllegalArgumentException` with the message "Empty stack"
  - `E peek()`
    - Looks at the element at the top of the stack but does not remove it
    - If the stack is empty, throw an `IllegalArgumentException` with the message "Empty stack"

Example Outputs:

```
List<String> linkedlist = new LinkedList<>();
linkedlist.add(0, "Jack");
linkedlist.add(1, "Julia");
linkedlist.add(2, "Chafos");
linkedlist.add(3, "Lucas");
linkedlist.add(4, "Nicholas");
//list is: [Jack] → [Julia] → [Chafos] → [Lucas] → [Nicholas]

System.out.println(linkedlist.remove(3)); //should print "Lucas"
//list is: [Jack] → [Julia] → [Chafos] → [Nicholas]

System.out.println(linkedlist.remove("Julia")); //should print "Julia"
System.out.println(linkedlist.remove(0)); //should print "Jack"
//list is: [Chafos] → [Nicholas]

int size = linkedlist.size(); //size should equal 2
linkedlist.add(2, "Suzy");
linkedlist.add(2, "Landry");
//list is: [Chafos] → [Nicholas] → [Landry] → [Suzy]
```

```
linkedlist.clear();
System.out.println(linkedlist.contains("Landry")); //should print false
//list is: []


Stack<String> stack = new LinkedList<>();
stack.push("Nicholas");
stack.push("Chafos");
stack.push("Lucas");
stack.push("Jack");
//Stack is: [Jack] → [Lucas] → [Chafos] → [Nicholas]

System.out.println(stack.peek()); //should print "Jack"
//Stack is: [Jack] → [Lucas] → [Chafos] → [Nicholas]

System.out.println(stack.pop()); //should print "Jack"
System.out.println(stack.pop()); //should print "Lucas"
//Stack is: [Chafos] → [Nicholas]
```

Rubric

The Checkstyle cap for this homework assignment is 25. Up to 25 points can be lost from Checkstyle errors.

We reserve the right to adjust the rubric, but this is typically only done for correcting mistakes.

Allowed Imports

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`

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 <u>top</u> 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].

Allowed 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 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

Submission

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

- **`Node.java`**
- **`LinkedList.java`**

Make sure you see the message stating "HW08 submitted successfully".

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, so be sure to **submit every file each time you resubmit**.

Gradescope Autograder

The Gradescope tests serve two main purposes:

- Prevent upload mistakes (e.g. non-compiling code)
- Provide basic formatting and usage validation

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.

This assignment will 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 "Restricted Features" to avoid losing points
- Check on Piazza for a note containing all official clarifications