HW07

Marcar esta página

Homework fecha límite Jul 26, 2023 17:46 -05 Vencidos

Problem Description

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

In this homework assignment, you will be developing the Singly-Linked LinkedList data structure, which will implement the provided List interface. In doing so, you will write a variety of methods from each.

You will also be creating your own generic Node class that your LinkedList will use. The provided interface and the classes you create will be written using generics such that they could work for any class parameter - you will be facing many of the challenges that the developers who wrote the java.util.LinkedList class had to confront, as that class uses generics, too!

List of covered topics:

- Generics
- Singly Linked Lists & Operations
- Exceptions

Solution Description

You will create two classes: `LinkedList.java` and `Node.java`. The LinkedList will consist of nodes linked to each other with pointers. `LinkedList.java` will implement the provided List interface. Using the abstract methods provided in the interface, you will have to implement these methods and adjust variables and pointers accordingly. To make these decisions, you should carefully follow the guidelines and logic as taught in lecture. Your program might function for base cases but not handle edge cases appropriately, **so test your code extensively**.

`Node.java`

Represents the nodes which will make up the LinkedList. This is a generic class! Be sure to reflect the use of generics in the class declaration.

Variables

Do not create any other instance variables than specified below. Any extra instance variables will result in deductions. All variables must follow the rules of encapsulation.

A variable named `data`

- Variable of the generic type holding the data stored in the node.

A variable named `next`

- Variable of the type `Node` (with generic type attached on) that acts as a "next" pointer, representing the Node that is next in the LinkedList

Constructors

- A constructor that takes in two arguments **exactly in the given order**: `data` and the `next` node and assigns it to instance variables accordingly.
- A constructor that takes in one argument: `data`
  - The next pointer is set to `null`
  - *Should use constructor chaining*

Methods

Do not create any other methods than those specified. Any extra methods will result in point deductions. All methods must have the proper visibility to be used where it is specified they are used.

- Getters and setters for the instance variables

`LinkedList.java`

Represents a LinkedList comprised of `Nodes`. This is a generic class! Be sure to reflect the use of generics in the class declaration.

Interface:

- This class should implement the provided `List<T>` interface

Variables

Do not create any other instance variables than specified below. Any extra instance variables will result in deductions. All variables must follow the rules of encapsulation.

A variable named `head`

- Variable of type `Node` (with generic type attached)
- This variable represents the head of the `LinkedList`
- If the list is empty, set this to `null`
- **You must name this variable `head`**

A variable named `tail`

- Variable of type `Node` (with generic type attached)
- This variable represents the tail of the `LinkedList`
- If the list is empty, set this to `null`
- **You must name this variable `tail`**

A variable named `size`

- Variable of type integer
- This variable represents the current size of the `LinkedList`
- If the list is empty, set this to 0
- **You must name this variable `size`**

Constructors

- A no-args constructor setting `head` and `tail` to null

Methods

Helper methods are discouraged because a proper LinkedList should not require them. Any extra methods will result in a 5 point deduction for code style. All methods must have the proper visibility to be used where it is specified that they are used.

- A getter for the `head` instance variable named `getHead`

- A getter for the `tail` instance variable named `getTail`
- You must override all necessary methods to correctly implement `List<T>` interface

Provided

`List.java`

An interface representing a List using generics. **This is provided to you.**

Methods:

The classes that implement List interface must implement the following methods:

- `addAtIndex(T data, int index)`
  - Adds a node to the position specified by `index`
  - If the index is negative or larger than the size of the list, throw `IllegalArgumentException` with a message "Your index is out of the list bounds"
  - If the passed data is `null`, throw `IllegalArgumentException` with a message "You cannot add null data to the list"
  - Adjust for head, tail, and size variables accordingly
- `getAtIndex(int index)`
  - Returns the data located at the specified index in the list
  - If the index is negative or larger than the size of the list minus 1, throw `IllegalArgumentException` with a message "Your index is out of the list bounds"
- `removeAtIndex(int index)`
  - Removes the data (and the node that stores it) from the specified index of the list and returns that data of the node that was removed
  - If the index is negative or larger than the size of the list minus 1, throw `IllegalArgumentException` with a message "Your index is out of bounds"
  - Adjust for head, tail, and size variables accordingly
- `remove(T data)`
  - Removes the first occurrence of the passed data from the list (and also remove the node that holds it) and returns the data from the removed node
  - If the passed data is not in the list, throw `NoSuchElementException` with a message "The data is not present in the list."
  - If the passed data is `null`, throw `IllegalArgumentException` with a message "You cannot remove null data from the list"
  - Adjust for head, tail, and size variables accordingly
- `clear()`
  - The method clears the LinkedList
  - There is a way to do this without iterating through the whole list (which would be O(n)). You won't get points off for having an inefficient solution, but keep in mind that java has Garbage Collecting so take full advantage of it when you can!
- `isEmpty()`
  - Returns a boolean value which represents whether the list is empty
- `size()`
  - Returns current size of the list

Allowed Imports

To prevent trivialization of the assignment, you are only allowed to import `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`

Grading

Homeworks are graded in an "all or nothing" manner. If your code is correct, you receive a 100 for the assignment; if it isn't, you receive a 0.

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.

The Vocareum (code editor) interface has six main components:

- The **Drop-Down** in the top left. This lets you choose from multiple available files. Note that this drop-down will only be visible in assignments that require multiple files.
- The **Build / Run** button. For all assignments in this course, the build and run button will perform the same action: compile your code and run a file scan. Building and running your code will not count towards your total allowed submission attempts, therefore you are free to build / run as many times as needed.
- The **Submit** button. This will compile your code, run a file scan, grade your assignment, and output results to console. Note that for most assignments in this class, you will only be allowed a limited number of submissions. A submission is counted when the submit button is clicked, regardless of whether or not your code is able to compile or if there are any file issues. Therefore, **we highly recommend that you build or run your code before submitting to ensure that there are no issues that will prevent your code from being graded and that every submission attempt will generate meaningful results**.
- The **Reset** button. This will revert all your changes and reset your code to the default code template.
- The **Code Window**. This is where you will write your code. Again, We highly recommend copying the starter code and working in your preferred IDE.
- The **Output Window**. This window will appear whenever you build, run, or submit your code and will display the results for you to view.

Vocareum Troubleshooting

We acknowledge that the Vocareum integration has some issues when submitting programs with multiple files. That is, the Vocareum interface may hide both the current file name and the combo box you need to select a file to edit. Unfortunately, this is beyond our control. Recall that the instructor recommends that you code and debug on your local JVM and submit in Vocareum mainly for grading. However, if you do need to edit in Vocareum, we have found the following workaround that you might try if you experience the stated problem.

1. Close any other homework from this course or another that uses the Vocareum integration.
2. Reload the webpage.
3. Open Vocareum in a new window. Keep the edX homework open: Vocareum won't visualize the homework description with the same formatting.
4. Select the course CS1331 and follow the instructions to start or continue working on the homework.
5. Look for where these four checkboxes appear: Files, README, Terminal, Source. Then, click the checkbox "Files." The README checkbox should uncheck automatically; if it doesn't, uncheck it.
6. You can now edit the files in the folder "work." Do not edit any files the extension "class" in the folder. Do not edit any file in any other folder or subfolder (such as "resource," "Submissions," or "workspace"). Do not create any additional files or folders.

For additional help, please visit the Vocareum information page located in the course information module!

For learners unable to access the Vocareum environment, this is the provided List.java file:

List.java

< Anterior | Siguiente >