Generic Lists in Java

Due Date: Sunday, September 27th @11:59pm 2020

Description:

In this project you will implement your own versions of a linked list in Java. Although the Java API provides built-in support for them, you will write your own to practice the constructs and language details we have seen in class. That means you are NOT allowed to use any pre existing libraries or Collections for this assignment.

Essentially, you are writing your own data structure library that could be used by others in the same way that one can use ArrayList<>. Your library provides the data structure as a class, the method calls expected with that data structure (a subset of the actual Java LinkedList) and the definitions for several iterators so that the clients of your library can iterate through the list in the same way as most generic data structures in Java.

In this simplified version, your data structure is a doubly linked list. Your implementation must be generic, like ArrayList, as to allow for different types when each data structure object is instantiated.

You will also implement the Iterator design pattern; allowing users access to multiple custom Iterators for your data structure.

Implementation Details:

You will download and use the Maven project template GLLMaven_Project1_Fall2020 from Blackboard. You will find a file called GLLProject.java in the src folder. This class contains the main method.

You will create a separate, new file, inside of src/main/java, for each outer class and interface. In comments at the top of the file GLLProject.java, please include your name and netid and university email as well as a brief description of your project.

DO NOT add any folders or change the structure of the project in anyway. DO NOT alter the pom.xml file.

Create a generic class called GenericLinkedList<T>:

It should implement the Java **Iterable<T>** interface: this will allow clients of your LL to use a forEach loop to iterate.

- This class will contain only two data fields:
  Node<T> head (this is the head of the list and should be **private**).
  int length (the length of the list and should be **private**)

- This class will have one single arg constructor:

- **public GenericLinkedList(T val) //instantiates the head data member with val**

This class should also define a generic inner class Node<T>:

It will include three fields:

- **T data**
- **Node<T> next;**
- **Node<T> prev**

This inner class will also provide a single arg constructor:

- **public Node(T val) // sets data equal to val**

*****This inner class is to be used to create nodes, in your linked list class*****

Inside the GenericLinkedList class you must implement the following methods:

- **public void addFirst(T e) // inserts the specified element at the beginning of the list**
- **public void addLast(E e) // inserts the specified element at the end of the list**
- **public int size( ) // returns the number of elements in the list**
- **public boolean contains(T e) //returns true if the list contains the specified element**
- **public boolean remove(T e) // removes the first occurrence of the specified element and returns true or returns false if the element does not exist.**
- **pubic void clear( ) // removes all elements from the list**
- **public T get(int index) // returns the element at the specified index or null if the index is out of bounds.**
- **public T set(int index, T element) //replace the element at specified position in the list with the specified element and return the element previously at the specified position. Return null if index is out of bounds**
- **public T removeHead() // retrieves and removes the head of the list**

- **public T removeTail() //retrieves and removes the tail of the list**
- **public ListIterator<T> listIterator(int index) //returns a list-iterator of the elements of this list starting at the specified position.**
- **public Iterator<T> descendingIterator( ) //returns an iterator over the elements of the list in reverse order(tail to head)**

***** you will also have to implement any abstract methods from the Iterable<T> interface**

Implementing Iterator Design Pattern:

You must also create a class to contain logic for iterating through your data structure (head to tail). Call this class **GLLIterator**. GLLIterator should be a generic class since it provides the logic to iterate through a generic linked list.

It should implement the java interface **Iterator<E>** (java.util.Iterator). You will have to implement two inherited methods: **public boolean hasNext(),** checks to see if there is another value in the data structure and returns true or false, and **public I next(),** returns the current value in the data structure and advances to the next item. This is the class that will be returned when the **iterator()** method is called from the Iterable<T> interface.

You will create another class **ReverseGLLIterator** which will be identical to the GLLIterator class except that the hasNext() and next() methods will have logic to iterate from the list in reverse (tail to head). This is the class that will be returned when the **descendingIterator()** method is called in the GenericLinkedList class.

You will also need to create a class to contain the logic for the list-iterator, call it **GLListIterator**. It should implement the Java interface **ListIterator<E>** (java.util.ListIterator). This is the class that will be returned when the **listIterator(int index)** method is called in the GenericLinkedList class. You will need to implement all the abstract methods inherited from the interface.

You are expected to fully comment your code and use good coding practices.

Test Cases:

You must write and include test cases for your GenericLinkedList class as well as all three iterators. At a minimum, you must write 1 test case per method, test that you can implement a forEach loop and fully test the constructors for the GenericLinkedList and Node class. Also test that the descendingIterator performs as expected as well as the methods in the ListIterator. You should be writing these at the same time you write methods in your project.

Electronic Submission:

Zip the Maven project GLMavenFall19 and name it with your netid + Project1: for example, I would have a submission called mhalle5Project1.zip, and submit it to the link on Blackboard course website.

Assignment Details:
Late work **is accepted**. You may submit your code up to 24 hours late for a 10%

penalty. Anything later than 24 hours will not be graded and result in a zero.

We will test all projects on the command line using Maven 3.6.3. You may develop in any IDE you chose but make sure your project can be run on the command line using Maven commands. Any project that does not run will result in a zero. If you are unsure about using Maven, come see your TA or Professor.

Unless stated otherwise, all work submitted for grading *must* be done individually. While we encourage you to talk to your peers and learn from them, this interaction must be superficial with regards to all work submitted for grading. This means you *cannot* work in teams, you cannot work side-by-side, you cannot submit someone else's work (partial or complete) as your own. The University's policy is available here:

https://dos.uic.edu/conductforstudents.shtml.

In particular, note that you are guilty of academic dishonesty if you extend or receive any kind of unauthorized assistance. Absolutely no transfer of program code between students is permitted (paper or electronic), and you may not solicit code from family, friends, or online forums. Other examples of academic dishonesty include emailing

your program to another student, copying-pasting code from the internet, working in a group on a homework assignment, and allowing a tutor, TA, or another individual to write an answer for you. It is also considered academic dishonesty if you click someone else's iClicker with the intent of answering for that student, whether for a quiz, exam, or class participation. Academic dishonesty is unacceptable, and penalties range from a letter grade drop to expulsion from the university; cases are handled via the official student conduct process described at https://dos.uic.edu/conductforstudents.shtml.