REVISION FOR MIDTERM EXAMINATION

SUBJECT: DATA STRUCTURES AND ALGORITHMS

A. REVISION CONCEPTS

- Linked List: Implement a Linked List class, given a ListInterface, with methods for adding, removing, and retrieving items.
- Stack, Queue: Implement Stack, Queue classes with common methods (in lectures). Make use of Stack, Queue class in Java API to solve problems.
- Recursion: Implement recursive functions to solve problems, recursive formulas, recursive operations in arrays and linked lists.
- Sorting: Implement common sorting algorithms in lectures.

B. REVISION EXERCISES

STUDENTS MUST CAREFULLY READ THE INSTRUCTION BEFORE CONDUCTING TASKS:

1. Create a folder named **<Student ID>_<FullName>,** i.e., 52100000_NguyenVanAn.
2. Students receive 03 folders Question1, Question2, and Question3. Copy the three folders to folder **<Student ID>_<FullName>.**
3. After finishing, students compress folder **<Student ID>_<FullName>** and submit. A correct submission consists of a folder named **<Student ID>_<FullName>** which contains 03 folders named Cau1, Cau2, and Cau3 respectively with students' work.

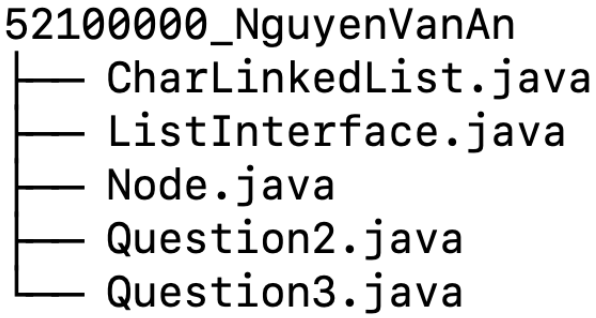
QUESTION

Students must ensure that the implementation of each task can be compiled, run successfully, and placed in the designated folder. **Misconducting**, **compile errors**, or **runtime errors** cause **0.0** point for the whole task.

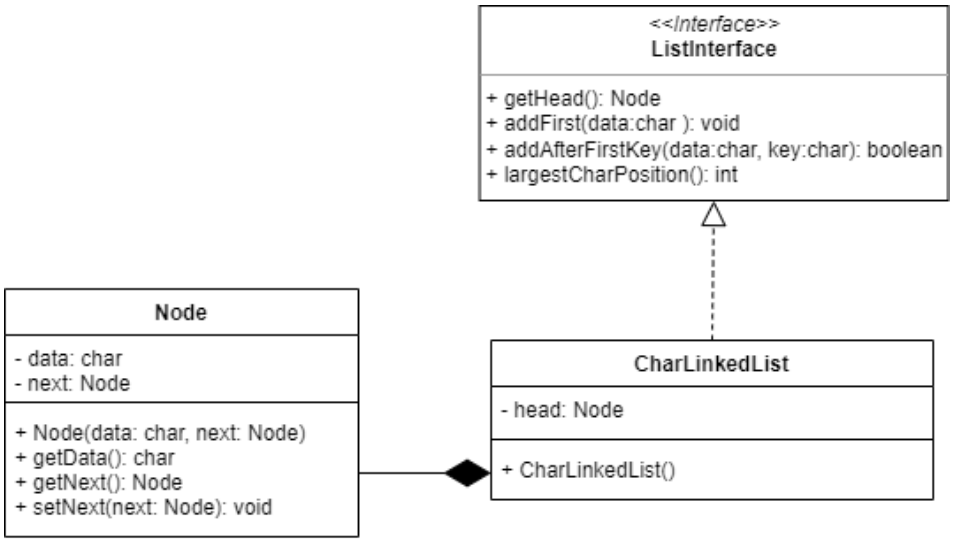
Given ***main*** functions are used to test students' implementation. There is no need to delete ***main*** functions, but students must ensure modified ***main*** functions do not cause any errors.

Students are not allowed to modify filenames, function names, order of parameters in given source files. Students conducts tasks in given files.

Below is an example of a folder with correctly completed tasks (52100000_NguyenVanAn is replaced by students' information correspondingly).



TASK 1:



In folder **Question1**, based on the given class diagram and two files **ListInterface.java** and **Node.java** (Students are not allowed to modify these two files), students implement **CharLinkedList** class to store characters.

Implement below methods of **CharLinkedList** class:

- **CharLinkedList():** default constructor, assign ***head* = *null***
- **getHead():** return ***head*** of the linked list.
- **addFirst(char data):** insert a new node with ***data*** to the linked list.
- **addAfterFirstKey(char data, char key):** insert a new node with ***data*** right after the first node consisting ***key*** (***head*** is the first node) and then return *true*. If there does not exist nodes with ***key***, then do not insert any new node and return *false*.

  Example: Given a list 'A', 'b', 'c' whose ***head*** is the node with 'A'. Invoke **addAfterFirstKey('b', 'E')** the updated linked list is 'A', 'b', 'E', 'c'.
- **largestCharPostition():** return the position of the first node consisting of the maximum ASCII-based value of its key. The position of ***head*** is 0. If the list is empty, then return -1.
  Example: Given a list 'A', 'b', 'c' whose ***head*** is the node with 'A'. Invoke **largestCharPostition()** then the returned value is 2 ('c' is the maximum value).

Students, by yourselves, implement a method to print the list down and a class with a **main** function used to test your work.

TASK 2: In folder **Question2**, there is a file named **Question2.java**. In class **Question2**, students implement the method

public static int recur(int n, int k)

to solve the problem below using recursion.

$$A(n,k) = \begin{cases} 1, & k = 0 \\ n * A(n, k-1), & k > 0 \end{cases} \quad where\ n\ and\ k\ are\ non-negative\ integers.$$

Students implement a ***main*** function to test your work. Note that if the ***main*** function causes any errors, then students get 0.0 point for the whole task.

TASK 3: In folder **Question3**, there is a file named **Question3.java**. In class **Question3**, students implement the method

public static int calculate(String[] expression)

which makes use of Stack to perform actions below.

In Task 3, students can use Stack class in Java API. Add the instruction below to import Stack class.

import java.util.Stack;

Given an array of Strings representing integers. For adding and minusing operations ((+) , (-)), students compute the result and finally return the value following the algorithm below.

1. Initialize a Stack<Integer>
2. For each item in the array of Strings
   2.1. If the item is number-like, then add it to the stack
   2.2. If the item is an operator, then pop two items, **o1** and **o2** respectively, from the stack, compute **o3** = **o2** <*operator*> **o1** and then add **o3** to the stack.
3. The finally remained item in the stack is the result of the algorithm.

Example: given an array of Strings as “3”, “4”, “+”, “2”, “1”, “+”, “-”

Iteration	Stack	Computation
3	3	
4	4 3	
+	7	3 + 4 = 7
2	2 7	
1	1 2 7	
+	3 7	2 + 1 = 3
-	4	7 – 3 = 4
The result is 4		

Students can implement the method

public static boolean isNumber(String str)

following the code block below to test whether an array item represents an integer.

```
private static boolean isNumber(String str) {
        return str.matches("0|([1-9][0-9]*)");
}
```

Students implement the ***main*** method to test your work. Note that if the ***main*** function causes any errors, then students get 0.0 point for the whole task.