Due: Oct. 11 (10/11), 13:00

Overview

This assignment consists of two parts, implementing a doubly linked lists and the simulation of a waiting system.

General Notes

- *Read this homework guideline carefully.* If you do not follow the guidelines, you may receive a 0 regardless of whether your code works or not.
- Do not use any IDEs (Eclipse, IntelliJ IDEA, etc.)
  - We recommend Sublime Text (Linux/Mac/Windows), Notepad++ (Windows), or TextWrangler (Mac).
  - IDEs often create a "package" of your code, which breaks the autograder.
  - **If you know how to fix the package problem**, you can use any IDE you want. However, we will not answer any questions related to this problem since we have already recommended a solution.
- Do not change any method or class signatures. If your code changes any class or method names or signatures, you will receive an automatic 0.
- Make sure your code compiles. Non-compiling code will automatically receive a 0. If your code does not compile, it may be better to just comment out the incorrect code and return a dummy value (something like null or -1) so the rest can compile. We recommend using Java Developer Kit (JDK) version 21, as we will grade the submission in that environment.
- To ensure that your code will be accepted by the autograder, you should submit your code on LearnUS, download it again, recompile it and check the provided test suite. This way, you know the file you are submitting is correct.
- Do not use ChatGPT or AI tools. You should rely solely on your own skills and knowledge.

1 Linked List

There are many variations on the basic linked list that are used frequently in practice. Here you will implement a doubly linked list for generic data. You should use the Node class we provided.

You will implement several common operations related to doubly linked lists. This implementation has a `head` pointer, a `tail` pointer and a `size` counter as variables. You should implement the following methods as well as its constructor in `DoublyLinkedList.java`.

1. `public DoublyLinkedList()`: Constructor of `DoublyLinkedList` class.
2. `public int size()`: Return the number of elements in the linked list.
3. `public boolean isEmpty()`: Return `True` if there are no elements in the linked list. Otherwise, return `False`.
4. `public Node<T> getHead()`: Return the first node of the linked list. If the list is empty, then return `null`.
5. `public Node<T> getTail()`: Return the last node of the linked list. If the list is empty, then return `null`.
6. `public void addFirst(T newElement)`: Insert the given element at the front of the list.
7. `public void addLast(T newElement)`: Insert the given element at the back of the list.
8. `public T popFirst()`: Delete the node pointed by `head` and returns the element. If the list is empty, then return `null`.
9. `public T popLast()`: Delete the node pointed by `tail` and returns the element. If the list is empty, then return `null`.
10. `public Node<T> search(T target)`: Find the first node matched with the target element and return the node. Return null if the search failed. You should check whether two elements are equal using the .equals() method.

11. `public void insertAfter(T newElement, T target)`: Find the first node matched with the target element and insert the new element after the node. If such node is not found, then do not insert the new element.

12. `public void searchDelete(T target)`: Find the first node matched with the target element and delete the node. If such node is not found, then do nothing.

2 Table Catch

One common use of linked lists is a waiting system. Using linked lists, one can manage the groups of people who are waiting for a table. To simulate a waiting system, you can use previously implemented doubly linked list with a `Customer` class, which we have already implemented for you.

You will implement a waiting system using the doubly linked lists. This implementation has two variables, `waitingList` and `nextTurnNumber`. The variable `waitingList` is the previously implemented doubly linked list with the `Customer` class.

The variable `nextTurnNumber` is a counter starting from 1. When a group is added to the waiting list, `nextTurnNumber` will be assigned for the group. Once `nextTurnNumber` is assigned for a group, it increases by one. `nextTurnNumber` is strictly increasing, and thus unique for each group. For example, the turn numbers of the first, second, and third groups are 1, 2 and 3. Then, `nextTurnNumber` is 4, even if some groups stop waiting or sit down.

You should implement the following methods as well as its constructor in `TableCatch.java`.

1. `public TableCatch()`: Constructor of `TableCatch` class.

2. `public int numWaiting()`: Return the number of groups in the waiting list. `nextTurnNumber` should be increased after assigning.

3. `public void startWaiting(int numPeople)`: Insert a new group to the waiting list.

4. `public void stopWaiting(int turnNumber)`: Delete the group with the given turn number from the waiting list. If there is no such group, then do nothing.

5. `public int seatOpened(int personnel)`: Delete the group with the smallest turn number among those where the number of people is less than the given personnel. After that, return the turn number of the removed group. If there is no such group, then return -1.

6. `public int howManyGroupsAhead(int turnNumber)`: Return the number of groups whose turn numbers are less than the given one. If the given turn number is not in the waiting list, then return the total number of groups in the waiting list.

General Directions

- Write your name and student ID number in the comment at the top of the files in src/main directory.
- Implement all of the required methods.
- You should not import anything that is not already included in the file.
- Pay careful attention to the required return types and edge cases.
- All the codes we provided can be found in src/base directory. If you are unsure what a class/method exactly does, please refer to the code.

Submission Procedure

Submit the files in the src/main directory, excluding Main.java, as a zip file. You *must* make a zip file for submission using the Gradle build tool (refer to Compiling section).

For this assignment, you should submit only the following three files:

- DoublyLinkedList.java
- TableCatch.java
- your_student_id_number.txt

You must rename 2024xxxxxx.txt to your actual student ID number. Inside that text file, you must include the following text. Please be sure to write all the following text including the last period.

In completing this assignment, I pledge that I have not given nor received any unauthorized assistance.

If this file is missing, you will get a 0 on the assignment. It should be named *exactly* your student id, with no other text. For example, *2024123456.txt* is correct while something like *2024123456_pa1.txt* will receive 0.

Compiling

On Linux, Mac and Windows PowerShell, You can test your Java code using the following command:

```
% ./gradlew -q runTestRunner
```

You can also make a zip file for submission using the following command. The zip file named with your student id (the name of the .txt file) will lie in the "build" directory:

```
% ./gradlew -q zipSubmission
```

We provide an empty Main class for testing using standard input/output:

```
% ./gradlew -q --console=plain runMain
```

Since this file (src/main/Main.java) is not for submission, you can use any package in the file.

On Windows Command Prompt, use `gradlew.bat` instead of `./gradlew`.

Testing

We have provided a small test suite (src/test) for you to check your code. You can test your code by compiling and running the tester program.

Note that the test suite we will use to grade your code will be much more rigorous than the one provided here (and not necessarily a superset of the provided tests). You should consider making your own test cases to check your code more thoroughly.