COL-106 Assignment 5

(Assignment objective after FAQs)

FAQS:

1. **Can I remove extends Comparator in LLMergeSort:**
   Answer: Yes

2. **Refer to piazza post https://piazza.com/class/k4mqiafjbtx79j?cid=401 for further FAQs.**

3. **In the addItem method in StockMgmt, should we consider the possibility of an item being added more than once?**

   E.g.

   addItem 3 3 item33 500

   addItem 3 3 item33 2678

   Answer: No you can assume that item will be added only once for a particular item

4. **So, we have to decrease the stock of an item, if we do a transaction for that item? and stock = stock - numItemsPurchased?**

   Answer : Yes

5. **What is the purpose of attribute stock in itemnode class? What should we do if while adding a transaction, numItemPurchased>stock?**

   Answer: There won't be such case when number of items being purchased is exceeding available stock. But you need to adjust available stock after adding any transaction.

6. **Can we add another parameter to the mergesort function of llmergesort class?**

   Answer: You can make any changes in you implementation of the method but you just need to make sure that you implement the interfaces correctly and don't make any changes in interface as at the time of evaluation we will replace your submitted interface files with the original one.

7. **Can we add new constructor to the Node Class?**

Answer: Yes you can as far as everything works with no changes in the interface!

8. **In both query 3 & 4, we have to simply sort all ItemNodes of a particular category (according to dec. order of stock_left and according to last date of purchase respectively)?**

   I mean to confirm -

   1. that we do not have to create a new linked list of ItemNodes (like in Query 1 & 2) and can simply sort the linked list of the said category in place only?

   2. If we do the above, then it is guaranteed that while evaluating (checking/printing) our mergesort, this original linkedlist will not be modified

   Also "Select a particular category and sort..." and "Filter items of a category and return all items sorted by ..." both phrases mean the same only (as explained above)? Since these are used differently in query 3 & 4.

   Answer: yes in both queries 3 you need to first choose itemNodes of given category id and then sort them according to the rules mentioned! There is no issue of original list gets modified as in each test case there will be only one query! Just make sure to return correct sorted list, and take nodes while making list of ItemNodes that you'll pass to MergeSort function as specified in the assignment as it will ensure order of mid nodes during sort.

9. The globallinkedlist is just for the checkmergesort function right? We can implement mergesort without using that attribute? Right?

   Answer: follow the instructions given in the comments in the driver code! globalList is just for us to verify the correct implementation of merge sort and for that you need to add whatever middle node's data you find in findSplit method using adjustGlobalPointer.

10. We can add helper functions for different classes without changing the interface right? For eg merge function in merge sort

    Answer: Yes

- ## <u>Overall Objective</u>

In this assignment, you have to work with sorting.

Primarily, this assignment will test your understanding of Merge sort as you will implement Merge sort on Linked List.

- **Scenario**:

Stock management:
Consider a warehouse that consists of several types of products. Such as Samsung galaxy ace, iPhone 6, Denim jeans, etc.

There are a fixed set of **categories** namely:
1. Mobile
2. Utensils
3. Sanitary
4. Medical equipment
5. Clothes

Each product belongs to exactly one category.

We have a Linked list(Class provided in skeleton code you must use this provided class) of **Categories**: **LinkedList<CategoryNode>**.

Each item in Categories has a category Id, category name and pointer to a linked list of Items of that category

Pointer to Mobile linked list	Pointer to Utensils Linked list	Pointer to Sanitary Linked list	Pointer to Medical equipment Linked list	Pointer to Clothes Linked list

Fig.1

Category interface is of the form:

```
interface CategoryInterface{
    public int getCategoryId();
    public String getCategoryName();
    public LinkedList<ItemNode> getLinkedListOfCategory();
}
```

Example:

Mobile linked list: Nokia6600 -> blackberryE70-> samsung galaxy Ace-> IPhone5.
Cloth Linked list: Denim jeans -> Bombay Dyeing bedsheet-> Kanchipuram Silk Saree.

Each **ItemNode** is of a class which implements the below **ItemInterface**:

```
interface ItemInterface{

   int getItemId();
   String getItemName();
   int getStockLeft();
   Node<PurchaseNode> getPurchaseHead();

}
```

Each item has a **purchase history of transactions** maintained via a linked list:
LinkedList<**PurchaseNode**> purchaseTransactions=new LinkedList<>();

getPurchaseHead() function returns the first element of that list. Further, each **purchase Node** is an object of a class that implements PurchaseInterface.

```
interface PurchaseInterface{

   DateNode getDate();
   int numItemsPurchased();

}
```

- **Note that Item Id is unique for a category but different categories might have same Item Id. This is not a problem for the queries that are given below.**

- **Queries** :

Note: After 'addItemTransaction' is executed for an item, the 'stock' of that item should be reduced by the 'numItemPurchased'.

The following queries will be executed after insertions of items and purchase histories while we evaluate your code and you must use LLMergeSort(Merge Sort for Linked List) to execute these queries.

1. Sort all items across categories by their last date of purchase(ascending order oldest to newest). If the date of purchase is the same for 2 items, then sort in descending order (lexicographic ordering) by their name. Return linked list of all items of type **ItemNode.** Items for which there is no purchase transaction done, consider their last date of purchase as 1st August 1970 with the number of items purchased as 0.

   To do this task, create a new LinkedList: **LinkedList<ItemNode>**. First, add ItemNode 's of category 1 and then of category 2 then of category 3 ….. so on. Perform sorting on the above newly created Linked list(Ref Fig. 2).

Category 1 items	**Category 2 items**	**Category 3 items**	**Category 4 items**	**Category 5 items**

LinkedList<Item Node>	**LinkedList<**Item Node>	**LinkedList<**Item Node>	**LinkedList<**Item Node>	**LinkedList<**Item Node>

Fig. 2

2. Sort items in ascending order of **“number of items purchased in specified period/((Absolute value of difference between year of first and last purchase in specified period)+1)”** as key in a certain period(Purchase date in the given range both start and end inclusive). If **“number of items purchased in specified period/((Absolute value of difference between year of first and last purchase in specified period)+1)”** are the same then sort them in lexicographic descending order by item name. Return linked list of **ItemNode**. Note that you should treat the term : **“number of items purchased in specified period/((Absolute value of difference between year of first and last purchase in specified period)+1)”** as a float. Note: **If there is only one transaction in that period then difference between year of first and last purchase is to be taken as 0 i.e denomanitor is 1.**

   To do this task, create a new LinkedList **LinkedList<ItemNode>** as done in query 1 above, filter the list according to the time period, and perform sorting on that. Note that**, “number of items purchased”** is the number of purchases made in the provided time interval. If there are no purchases in that interval for an item, then consider the **“ number of items purchased”** as 0 for that item. Items for which there is no purchase transaction done, consider their last date of purchase as 1st August 1970 with the number of items purchased as 0.
3. Select a particular category and sort items in descending order according to stock_left. Example: medical equipment category. If stock left is the same for the two items then sort them in lexicographic descending order by name. Return linked list of **ItemNode**.

4. Filter items of a category (let’s say mobiles). Now return all items sorted by last_date of purchase(descending order-newest to oldest). If the date of purchase is the same then use lexicographic descending order by their names. Items for which there is no purchase transaction done, consider their last date of purchase as 1st August 1970 with the number of items purchased as 0. Return linked list of **ItemNode**.

Sorting technique to use: Merge sort:

How to perform split operation:
For a linked list with n elements:
1. If n is even: the middle element is n/2th element.
2. If n is odd: middle element is [n/2] +1 th element, where [.] is the Greatest integer function.

Kindly Read LLMergeSort class (in the skeleton code) for further details.

- **<u>Instructions</u>**:

1. We have provided you the skeleton code containing interfaces and class files you need to add your code in the respective functions. **You have to adhere to the interfaces provided.**

2. We have also provided a checker script with mandatory test cases to check your implementation.

3. Please read the comments in the skeleton code carefully and don't call or modify the mentioned methods in your code as this may break things while we will evaluate your submissions and you may get 0 grade due to that.

4. Do not put any print statements while submitting the code. Make sure to remove all print statements in your implementation before submitting to avoid side effects.

5. In each test case, there will be a series of insert Item and purchase transactions followed by one of the four queries and checkMergeSort(Which will verify your implementation of merge sort).

6. Do not use * import anywhere in your code. You are not allowed to import anything from the standard java library except these functions.

    a. java.util.Comparator

7. **The driver file, checker files and interface files will be replaced with the original ones.**

8. **Any sort of plagiarism will be dealt strictly as per course policy.**

9. **Assignment will be evaluated on jdk11 so check your jdk version before submitting.**

Submission Format:

1. Compress the skeleton code folder given to you in a zip file named **'ENTRYNUMBER**.zip' and upload on Moodle where **ENTRYNUMBER** is your entry number. THIS IS VERY IMPORTANT.