  
Name : Azaria Cindy Sahasika

Number Id : 2341760169 / 06

Class : 1G – Business Information System

Lesson : Algorithm and Data Structure

Material : Material 10 – Double Linked Lists

Github Link : <https://github.com/azariacindy/algorithm-ds>

JOBSHEET 12

Double Linked Lists

* 1. **Learning Objective**

After learning this lab activity, students will be able to:

1. Understand Double Linked List algorithm
2. Create and declare double linked list algorithm
3. Implement double linked list algorithm in various case studies
   1. **Lab Activities 1**

In this lab activity, we will create Node class and DoubleLinkedList class that has operations to insert data in multiple way. (from the beginning or the tail of the list)

* + 1. **Steps**
       1. Take this class diagram as your reference for creating the **DoubleLinkedList clas**

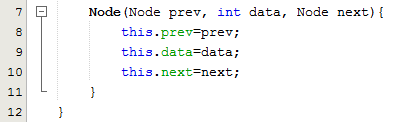
|  |
| --- |
| Node |
| data: int prev: Node  next: Node |
| Node(prev: Node, data:int, next:Node) |

|  |
| --- |
| DoubleLinkedLists |
| head: Node  size : int |
| DoubleLinkedLists() isEmpty(): boolean addFirst (): void addLast(): void  add(item: int, index:int): void size(): int  clear(): void  print(): void |

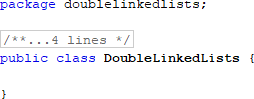
* + - 1. Create a new package named **DoubleLinkedList**
      2. Create a new class in that package named **Node**
      3. In that class, declare the attributes as described in the class diagram



* + - 1. Next, add the default constructor in Node class



* + - 1. Create a new class named **DoubleLinkedList** in the same package with the node as following image:



* + - 1. Next, we add the attributes



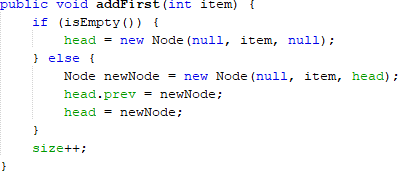
* + - 1. Then, add the constructor in class **DoubleLinkedList**

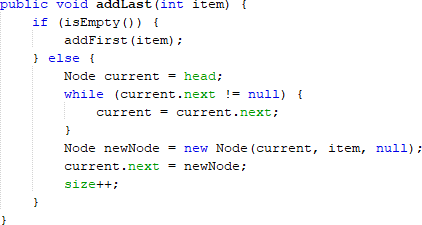


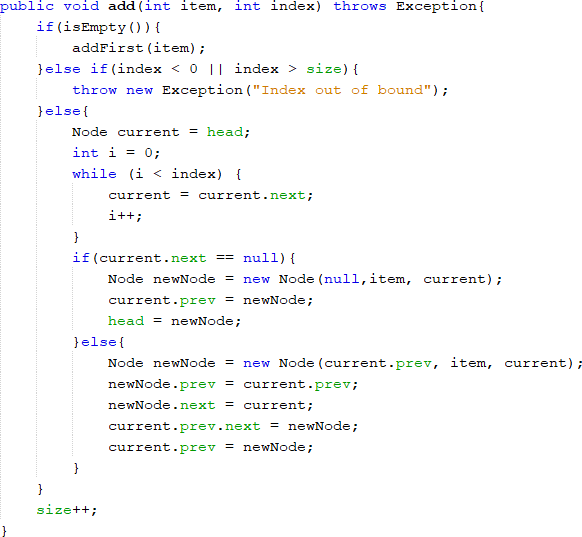
* + - 1. Create method isEmpty(), this method will be used to check whether the linked list is empty or not



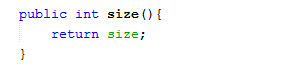
* + - 1. Then add method **addFirst().** This method will be executed when we want to add data in the beginning of the list



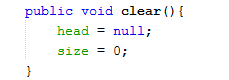
* + - 1. Let’s not forget about adding the data in the end of the list. We can do it after adding these lines of code in **addLast()** method
      2. If we want to add a data that specified by a certain index, we will need to provide additional method to do so. It can be done by creating the **add()** method



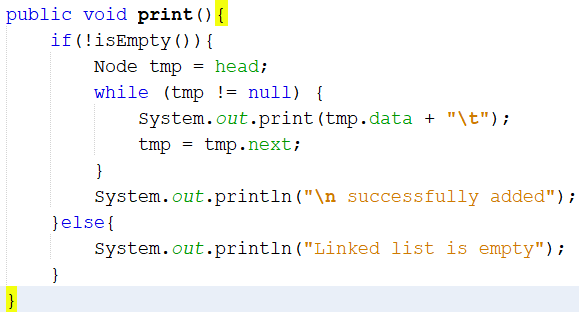
* + - 1. We want to make our list has an easy access to retrieve the length of the list. That’s why we create method **size()**



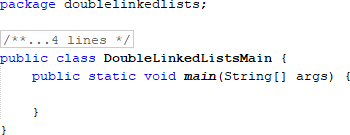
* + - 1. We create a method **clear()** to remove all the data that are exist in linked lists



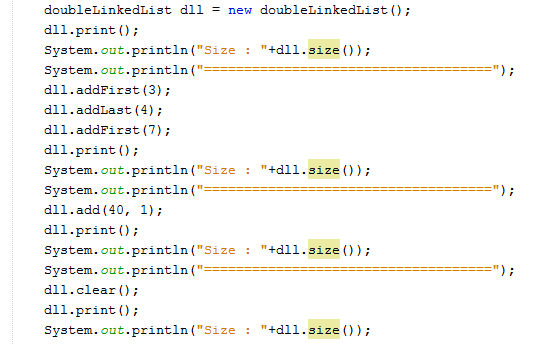
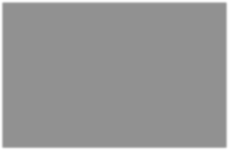
* + - 1. Next up, to print the whole data in the list, we need to create a method print().



* + - 1. After creating the blueprint classes, we will need one main class so that all of that can be included in the program. Create **DoubleLinkedListMain class** to do so

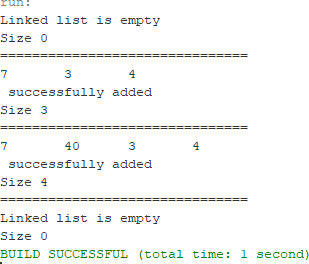


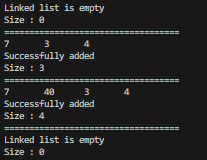
* + - 1. Instantiate an object from **DoubleLinkedList** class in the main method. Then apply these program code



* + 1. **Result**

Compile the program and see if the result matches with following image





* + 1. **Questions**
       1. What’s the difference between single linked list and double linked list?
* Single linked list
* Each node in a link list contains data and references to the next node in the sequence.
* Navigation is unidirectional, meaning that it can only browse the list in one direction (from end to end).
* Memory usage is lower as each node only stores one reference (to the next node).
* Double linked lists
* Each node in a double-linked list contains data, a reference to the next node, and a reference to the previous node.
* Navigation is bidirectional, meaning that you can traverse the list in both directions (from head to tail and tail to head).
* Memory usage is higher as each node stores two references (to the next and previous node).
  + - 1. In **Node class**, what is the usage of attribute next and prev?
* next: This attribute stores a reference to the next node in the list. This attribute is used to browse the list in the forward direction.
* prev: This attribute stores a reference to the previous node in the list. This attribute is used to traverse the list in the backward direction.
* These attributes are essential to maintain the structure of a doubly chained list and allow efficient insertion and deletion of nodes at both ends of the list.
  + - 1. In constructor of **DoubleLinkedList class.** What’s the purpose of head and size attribute in this following code?



* head: This attribute serves as a reference to the first node in the linked list. Initializing it to null indicates that the list is initially empty.
* size: This attribute keeps track of the number of elements (nodes) in the linked list. Initializing it to 0 indicates that the list starts with zero elements.
* These attributes are fundamental for managing and manipulating the linked list. The head allows access to the nodes, and the size helps in keeping track of the number of nodes in the list.
  + - 1. In method **addFirst(),** why do we initialize the value of Node object to be null at first?

Node newNode = new Node(**null**, item, head);

* In the addFirst() method, initialize the Node object with a null value for subsequent references because the new node will be the first node in the list, so there are no subsequent nodes. The head reference of the list will be updated to point to the new node, and the previous reference of the new node will be null because it is the first node.
  + - 1. In method **addLast(),** what’s the purpose of creating a node object by passing the **prev** parameter with **current** and **next** with **null** ?

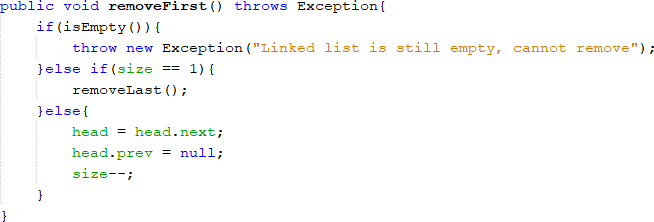
Node newNode = new Node(**current**, item, **null**);

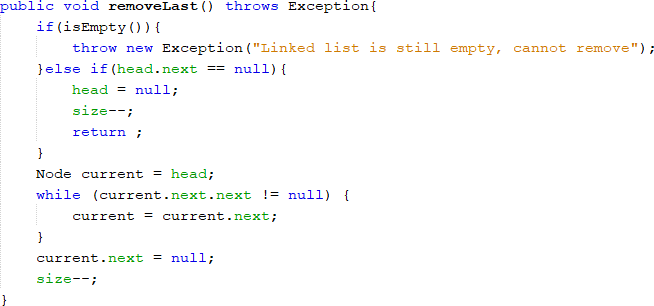
* In the addLast() method, it creates a new Node object with the previous reference set to the current node, and subsequent references set to zero. This is because the new node will be the last node in the list, so there are no subsequent nodes. The next reference of the current node will be updated to point to the new node, and the previous reference of the new node will be set to current. In this way, the new node is added to the end of the list, and all existing nodes are linked together in the correct order.
  1. **Lab Activities 2**

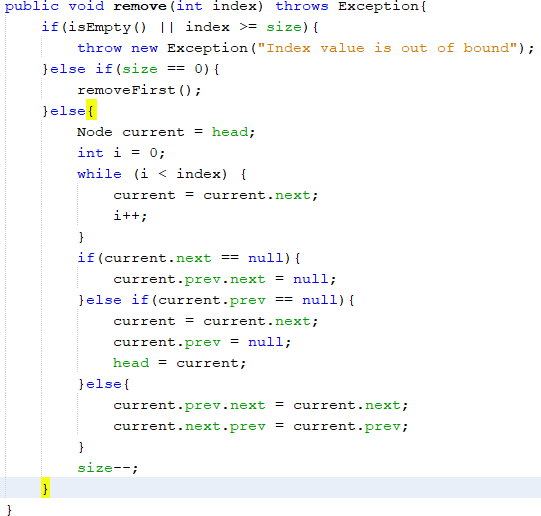
In this lab activity, we have added some methods from our 1st lab activity. Now, we added some ways for the users to remove a data in the beginning of the list, the tail, or with specified index. For more details, pay attention to this class diagram:

|  |
| --- |
| DoubleLinkedLists |
| head: Node  size : int |
| DoubleLinkedLists() isEmpty(): boolean addFirst (): void addLast(): void  add(item: int, index:int): void size(): int  clear(): void print(): void  **removeFirst(): void removeLast(): void**  **remove(index:int):void** |

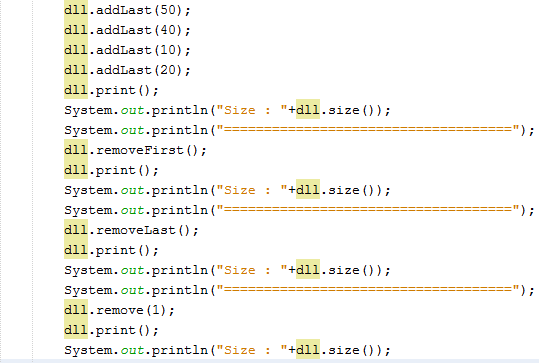
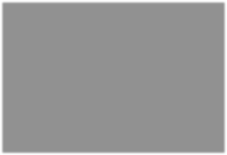
* + 1. **Steps**
       1. Create method **removeFirst()** in **class DoubleLinkedList**



* + - 1. Create method **removeLast()** in **class DoubleLinkedList**
      2. Create method **remove()** in **class DoubleLinkedList,** alongside with its parameter

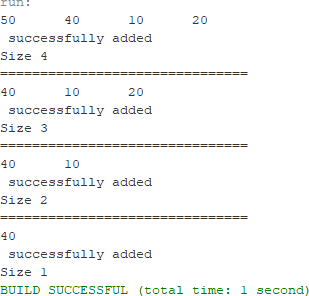


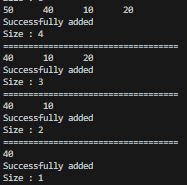
* + - 1. To execute additional codes we’ve just added, also make addition in the main class as well



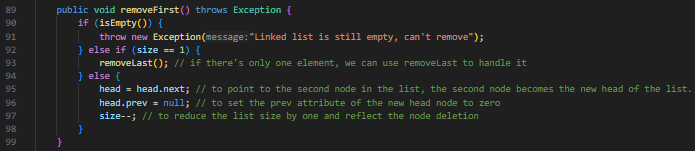
* + 1. **Result**

Compile the program and see if the result matches with following image





* + 1. **Questions**
       1. What’s the meaning of these statements in **removeFirst()** method?



* + - 1. How do we detect the position of the data that are in the last index in method

**removeLast()**?

* After the loop, current.next is the last node in the list. Setting current.next = null will remove the reference to the last node, effectively removing it from the list.
  + - 1. Explain why this program code is not suitable if we include it in **remove** command!
* because this command specifically deletes the first element (head.next). The general delete method needs to take the value to be deleted as input and then browse the list to find a matching node. Once found, it can use logic similar to removeFirst() to remove that specific node.
  + - 1. Explain what’s the function of this program code in method **remove**!

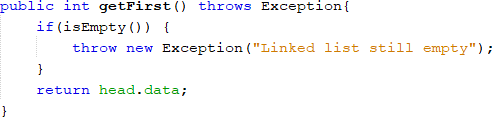


* 'current.prev.next = current.next;': the function updates the next pointer of the node before the deleted node (current.prev) to point to the node after the deleted node (current.next). This effectively bypasses the deleted node in list traversal.
* 'current.next.prev = current.prev;': the function updates the prev pointer of the node after the deleted node (current.next) to point to the node before the deleted node (current.prev). This maintains the bidirectional connection between nodes in the doubly connected list.
  1. **Lab Activities 3**

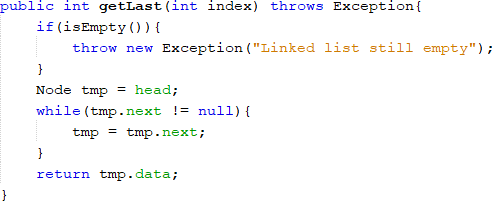
In this 3rd lab activity, we will test if we can retrieve a data in linked list in various needs. The first is we can get a data in the beginning of the list, at the end of the list, or in specified index of the list. We will create 3 methods to realize the idea. For more details, feel free to check this class diagram

|  |
| --- |
| DoubleLinkedLists |
| head: Node  size : int |
| DoubleLinkedLists() isEmpty(): boolean addFirst (): void addLast(): void  add(item: int, index:int): void size(): int  clear(): void print(): void removeFirst(): void removeLast(): void  remove(index:int):void  **getFirst(): int getLast() : int get(index:int): int** |

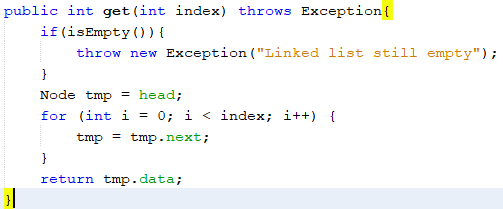
* + 1. **Steps**
       1. Create a method **getFirst()** in class **DoubleLinkedList** to retrieve the first data in the list



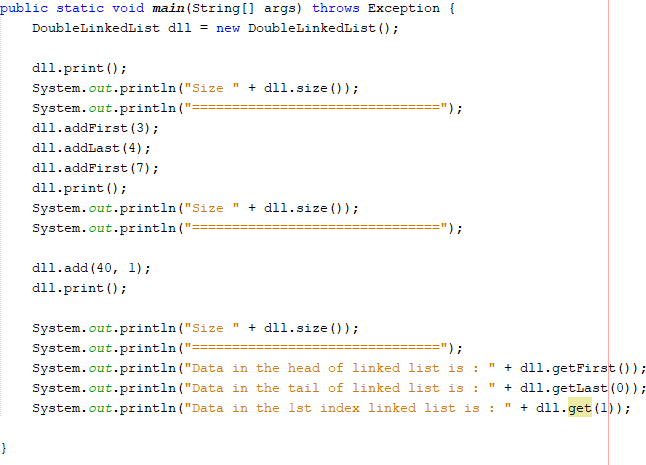
* + - 1. Create a method **getLast()** in class **DoubleLinkedList** to retrieve the data in the list



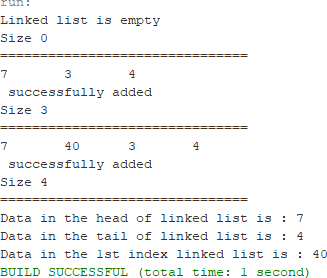
* + - 1. Create a method **get(int index)** in class **DoubleLinkedList** to retrieve the data in specified index of the list

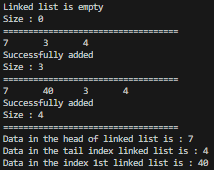


* + - 1. In the main class, add the program code as follows and see the result

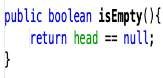
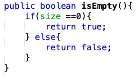


* + 1. **Result**

Compile the program and see if the result matches with following image

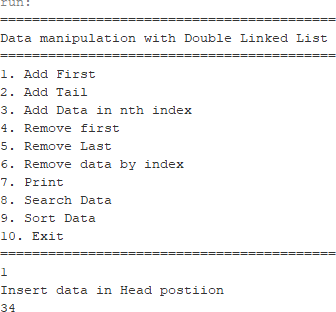


* + 1. **Questions**
       1. What is the function of method **size()** in **DoubleLinkedList** class ?
* The size() method in the DoubleLinkedList class is used to determine the number of elements currently in the list. It loops through the list, starting from the head node, and increments the counter for each node encountered. Finally, it returns the counter value, which represents the total number of elements.
  + - 1. How do we set the index in double linked list so that it starts from 1st index instead of 0th index?
* Double linked lists, like most data structures, usually use 0-based indexing. This means that the first element has index 0, the second element has index 1, and so on.
  + - 1. Please explain the difference between method **Add()** in double linked list and single linked list !
* Single linked list
* A linked list is created, then a new node is added to nodeNew (null if the list is empty). Then nodeNew is updated to become the new first element.
* Double-linked lists
* Creates a linked list, then adds a new node to the first node (null if list is empty). Then the predecessor of nodeNew is set to zero (because it becomes the new first element).
  + - 1. What’s the logic difference of these 2 following codes?

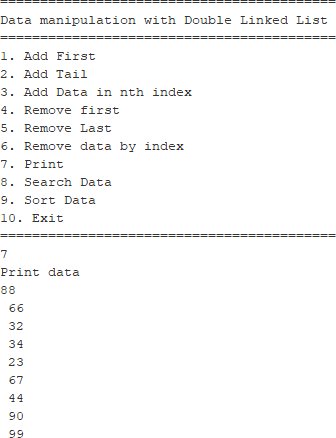
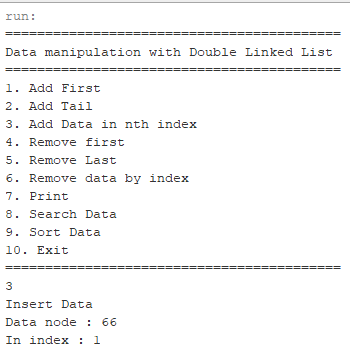


1. (b)
2. If size is 0, the list is empty, and true is returned. Otherwise, the list is not empty, and false is returned.
3. If head is null, the list is empty, and true is returned immediately. If head is not null, the list is not empty, and false is implicitly returned (since the method doesn't have an explicit else statement).
   1. **Assignment**
4. Create a program with double linked list implementation that allows user to choose a menu as following image! The searching uses sequential search approach and the program should be able to sort the data in descending order. You may any choose sorting approach you prefer (bubble sort, selection sort, insertion sort, or merge sort)

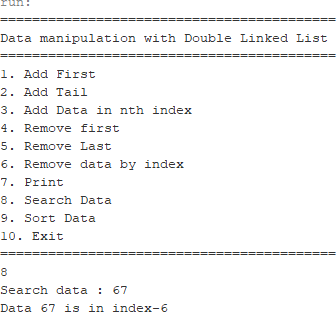
**Adding a data**



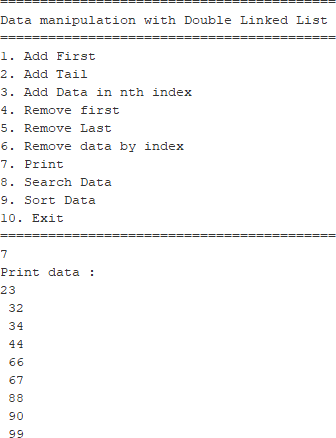
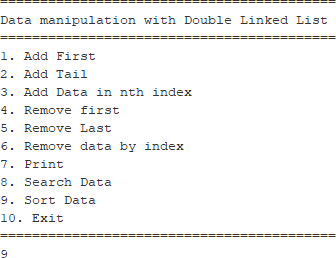
**Add data in specified index and display the result**



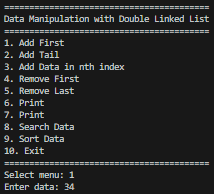
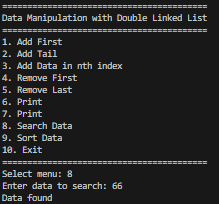
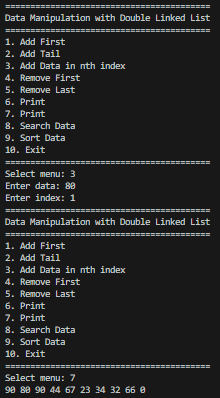
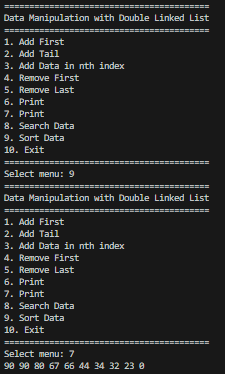
**Search Data**



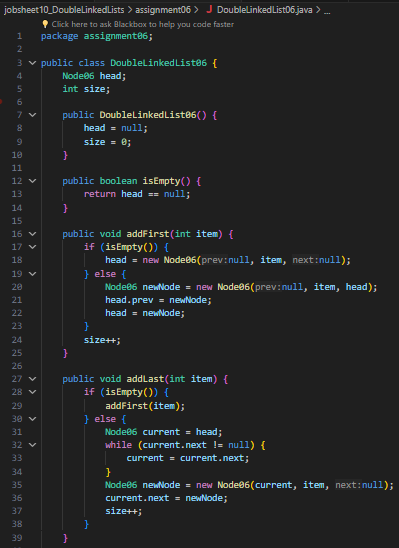
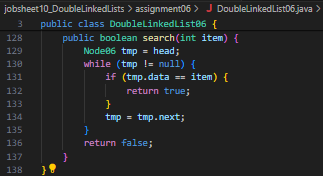
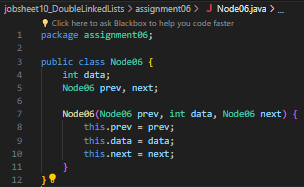
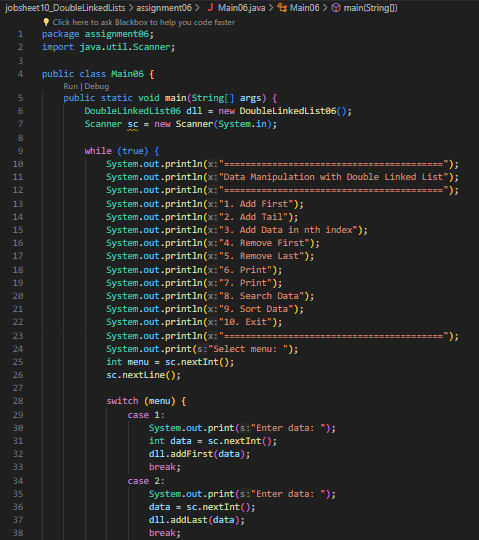
**Sorting Data**



**Output :**

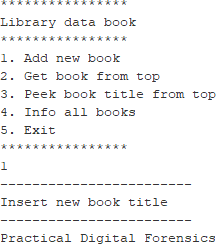
  

**Code :**

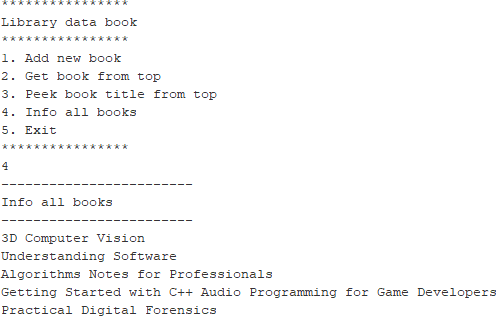
    

1. We are required to create a program which Implement Stack using double linked list. The features are described in following illustrations:

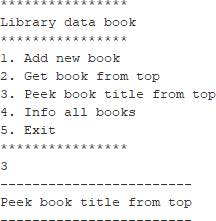
**Initial menu and add Data (push)**



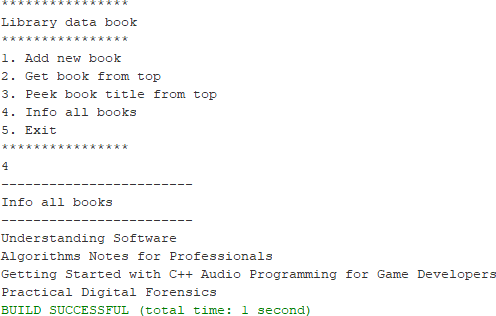
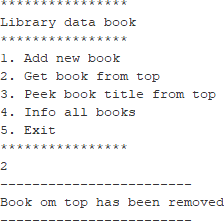
**Print All Data**



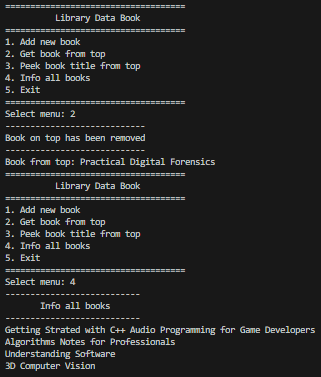
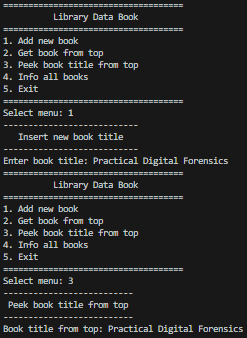
**See the data on top of the stack**



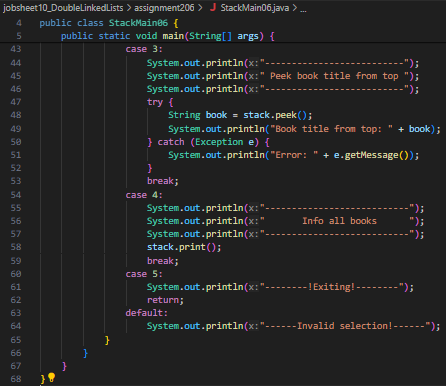
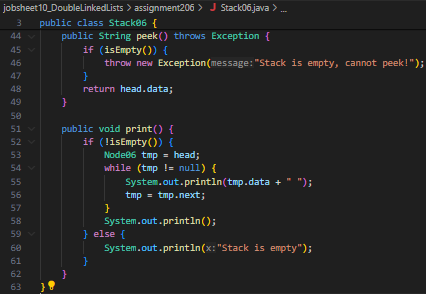
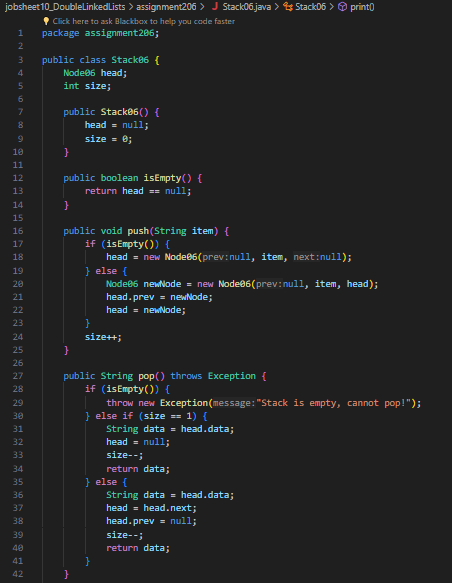
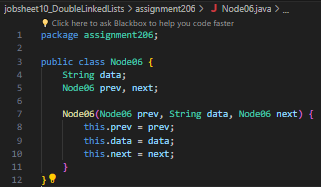
**Pop the data from the top of the stack**



**Output :**

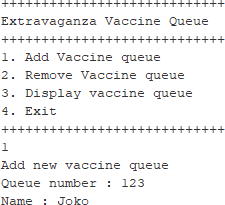


**Code :**

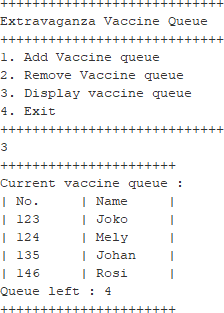


1. Create a program that helps vaccination process by having a queue algorithm alongside with double linked list as follows (**the amount left of queue length in menu print(3) and recent vaccinated person in menu Remove data (2) should be displayed**)

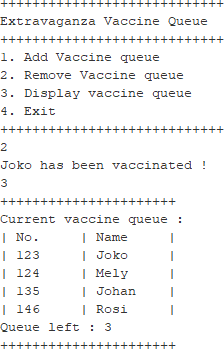
**Initial menu and adding a data**



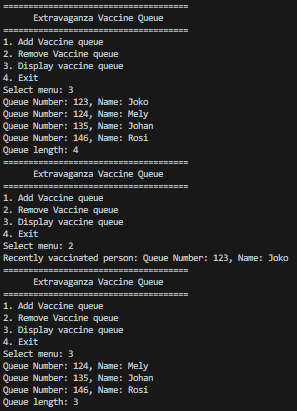
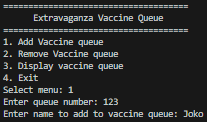
**Print data (notice the highlighted red in the result)**



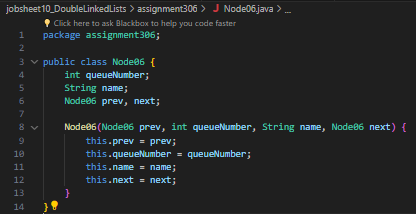
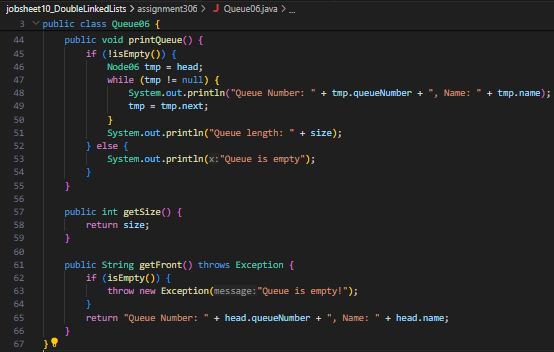
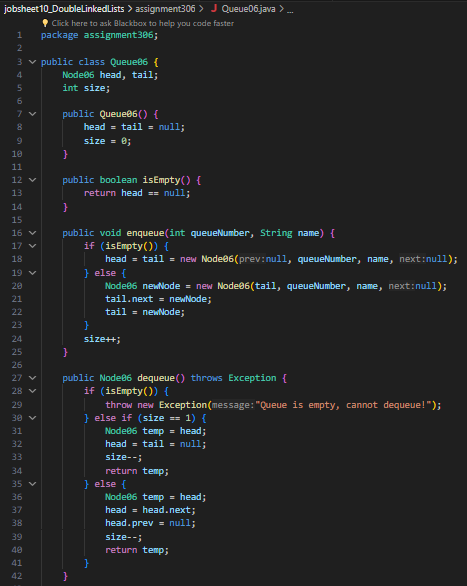
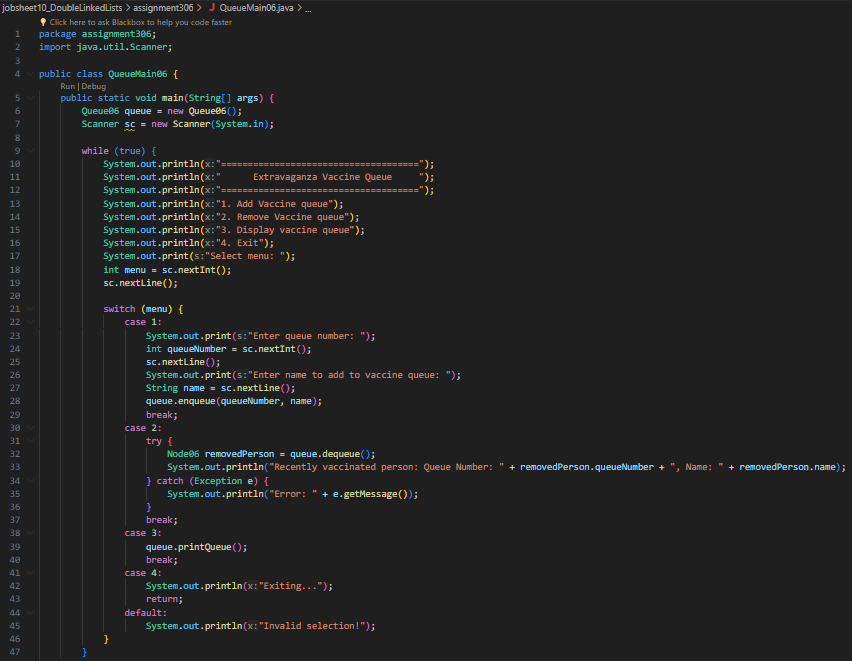
**Remove Data (the highlighted red must displayed in the console too)**



**Output :**

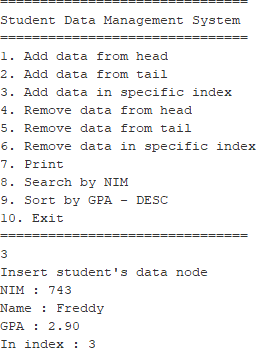
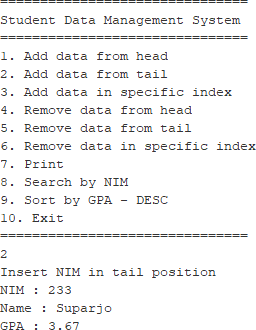
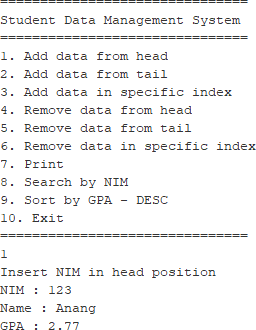


**Code :**

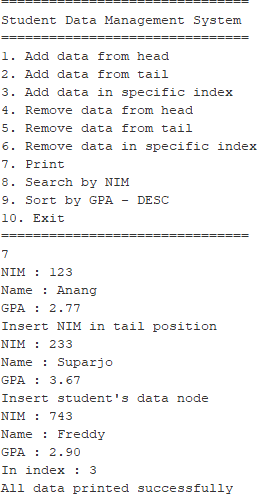
  

1. Create a program implementation that list students score. Each student’s data consist of their nim, name, and gpa. The program should implement double linked list and should be able to search based on NIM and sort the GPA in descending order. **Students class must be implemented in this program**

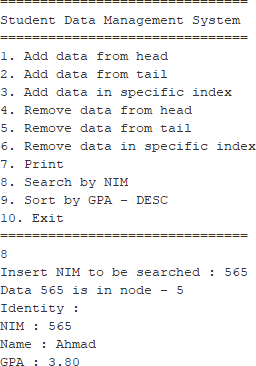
**Initial menu and adding data**



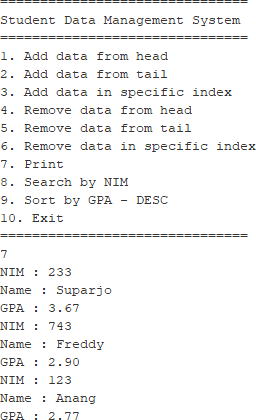
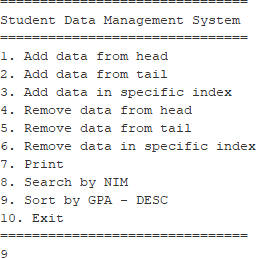
**Printing data**



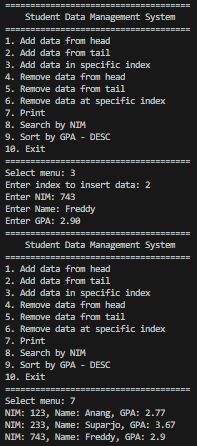
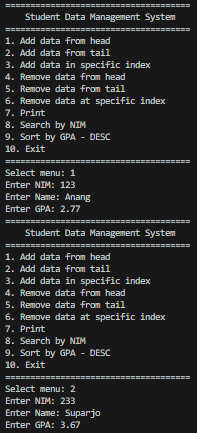
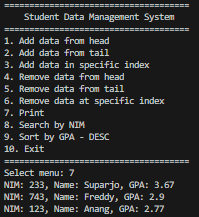
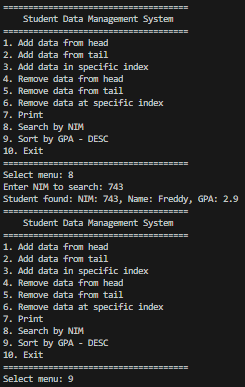
**Searching data**



**Sorting data**



**Output :**

**Code :**

