Lab 2

CSE 274

The focus of this lab is on Doubly linked lists. Accordingly, in this document a link list refers to a doubly linked list. Create a new project in Eclipse, import the `Application.java` file as a source file, and run the Application. Make sure that you receive no error.

I. The displayBackward method

Look into the `displayForward` method and see how it works. This method displays the `iData` variables of the links in a forward way, meaning that the method starts from the first link and moves to the right ones. Next, implement the following method for the `DoublyLinkedList` class:

`public void displayBackward()`

The above method displays the `iData` variables of the links in a backward way, meaning that the method starts from the last link and moves to the left. Erase the body of the `main` method, copy the following lines of code in the body of the main method, run the Application, and make sure that the correct output is produced without an error:

```
DoublyLinkedList myLinkedList = new DoublyLinkedList();
myLinkedList.displayBackward();
myLinkedList.insertFirst(11);
myLinkedList.displayBackward();
myLinkedList.insertFirst(22);
myLinkedList.displayBackward();
myLinkedList.insertFirst(33);
myLinkedList.displayBackward();
```

The correct output is printed below:

```
List (last-->first):
List (last-->first): 11
List (last-->first): 11 22
List (last-->first): 11 22 33
```

II. The deleteFirst method

Implement the following method for the `DoublyLinkedList` class:

`public void deleteFirst()`

The above method deletes the first link of the linked list. Please do not use the `public void delete(int key)` method that is provided in the `Application.java` file. Erase the body of the `main` method, copy the following lines of code in the body of the main method, run the Application, and make sure that the correct output is produced without an error:

```
DoublyLinkedList myLinkedList = new DoublyLinkedList();
myLinkedList.deleteFirst();
myLinkedList.insertFirst(11);
myLinkedList.deleteFirst();
myLinkedList.insertFirst(22);
myLinkedList.insertFirst(33);
myLinkedList.deleteFirst();
myLinkedList.insertFirst(44);
myLinkedList.insertFirst(55);
myLinkedList.insertFirst(66);
myLinkedList.displayForward();
```

The correct output is printed below:

```
List (first-->last): 66 55 44 22
```

III. The `deleteLast` method

Implement the following method for the `DoublyLinkedList` class:

`public void deleteLast()`

The above method deletes the last link of the linked list. Please do not use the `public void delete(int key)` method that is provided in the `Application.java` file. Erase the body of the `main` method, copy the following lines of code in the body of the main method, run the Application, and make sure that the correct output is produced without an error:

```
DoublyLinkedList myLinkedList = new DoublyLinkedList();
myLinkedList.deleteLast();
myLinkedList.insertFirst(11);
myLinkedList.deleteLast();
myLinkedList.insertFirst(22);
myLinkedList.insertFirst(33);
myLinkedList.deleteLast();
myLinkedList.insertFirst(44);
myLinkedList.insertFirst(55);
myLinkedList.insertFirst(66);
myLinkedList.displayForward();
```

The correct output is printed below:

```
List (first-->last): 66 55 44 33
```

IV. The `delete` method

The following method is implemented for the `DoublyLinkedList` class:

`public void delete(int key)`

Let say that we have a linked list where for three links we have `iData==key`. Look into the implementation of the above method and specify which one of the following notes is correct?

1) The `delete` method deletes all the links in which `iData==key`.
2) The `delete` method deletes the first link in which `iData==key`.
3) The `delete` method deletes the last link in which `iData==key`.

Please note that you don't need to submit your answer, but you need the correct answer when working on Question VI. Once you selected one of the above options, run a test to make sure that your answer is correct.

V. The insertAfter method

In this section, we implement the following method for the `DoublyLinkedList` class:

`public void insertAfter(int key, int id)`

The above method receives two inputs as arguments. If there is no link with `iData==key`, the method `return`. But if there is a link with `iData==key`, the above method inserts a new link right after the link with `iData==key`. The method sets `iData` variable of the new link to be `id`. Implement the method using pseudocode/logic that is provided below:

```
If the Linked List is empty return.

Let "current" be a reference to the first link.

while "iData" variable of "current" link is not equal to "key":

      Shift "current" one link forward.

      If "current" is equal to "null" return.

At this point there is a link with "iData==key" which "current" is
   referring to.

Check if "current" is a reference to the last link. If it was,
   insert the new link after the last link and return.

At the point, "current" is not a reference to the last link. Insert
    the new link after the "current" link.
```

Erase the body of the `main` method, copy the following lines of code in the body of the main method, run the Application, and make sure that the correct output is produced without an error:

```
DoublyLinkedList myLinkedList = new DoublyLinkedList();
myLinkedList.insertAfter(11,15);
myLinkedList.insertFirst(11);
myLinkedList.insertAfter(11,16);
myLinkedList.insertLast(22);
myLinkedList.insertLast(33);
myLinkedList.insertLast(44);
myLinkedList.insertAfter(44,17);
myLinkedList.insertAfter(33,18);
myLinkedList.displayForward();
```

The correct output is printed below:

```
List (first-->last): 11 16 22 33 18 44 17
```

VI. The cleanup method

In this section, we implement the following method for the `DoublyLinkedList` class:

`public void cleanup()`

This method deletes the links with duplicate `iData` so that the links of the linked list have unique `iData`. More precisely, if there are links with same `iData`, the method keeps only one of such links and deletes the rest of them. When there are links with same `iData`, only the last one that is closer to the ending part of the link list is kept and the others are deleted. Implement the `cleanup` method using the below pseudocode/logic:

```
If the link list is empty return.

If the link list has only one link return.

Let "current" be a reference to the second link.

Do the below operations, then shift "current" one link to the right
    and keep repeating until "current" reaches to the end of the
   link list:

            Let "back" be a reference to the link before "current".

            Do the below operations, then shift "back" one link to
               the left, and keep repeating until "back" reaches to
                the beginning part of the linked list:

                        If back.iData==current.iData:

                              delete(back.iData);
                              Set "back" to be equal to "current"
```

Please note that `delete` method in the above lines of code is the `delete` method that is available in `Application.java` file. To test the developed method, erase the body of the `main` method, copy the following lines of code in the body of the main method, run the Application, and make sure that the correct output is produced without an error:

```
myLinkedList.cleanup();
myLinkedList.displayForward();
myLinkedList.insertLast(22);
myLinkedList.cleanup();
myLinkedList.displayForward();
myLinkedList.insertLast(33);
myLinkedList.insertLast(44);
myLinkedList.insertLast(44);
myLinkedList.insertLast(33);
myLinkedList.insertLast(44);
myLinkedList.cleanup();
myLinkedList.displayForward();
```

The correct output is printed below:

```
List (first-->last):
List (first-->last): 22
List (first-->last): 22 33 44
```

VII. Submitting the Lab Assignment

Erase the body of the `main` method, copy the content of the `Test.txt` file to the body of the main method, run the Application, and make sure that the correct output is produced without an error. The correct output is printed below:

```
List (last-->first):
List (last-->first): 11
List (last-->first): 11 22
List (last-->first): 11 22 33
List (last-->first):
List (first-->last): 66 55 44 22
List (last-->first):
List (first-->last): 66 55 44 33
List (last-->first):
List (first-->last): 11 16 22 33 18 44 17
List (last-->first):
List (first-->last):
List (first-->last): 22
List (first-->last): 22 33 44
```