CS 1083

Module 9 Assignment

By Ngoc Phuong Anh Nguyen - 3712361

July 31st, 2021

Part A – Testing the LinkedList Class:

Source codes:

ListTest.java:

```
/**
 * This is a driver program.
 * @author Ngoc Phuong Anh Nguyen - 3712361
 */
public class ListTest
{
     public static void main(String[] args)
     {
          LinkedList nameList = new LinkedList();

//************************** Display an empty list **************************
          System.out.println("\nDisplay an empty list:");
          nameList.display();

//********************** Remove a name from the empty list **********************
          System.out.println("\nRemove a name from the empty list:");
          if(nameList.remove("Amy"))
          {
               System.out.println("Failed - improper result returned by remove()");
          }
          else
          {
```

```
            System.out.println("Correct - remove() returned false");
        }

//*********** Call getHead() for empty list and display the result ************
        System.out.println("\nCall getHead() on the empty list and display the result: ");
        if(nameList.getHead() == null)
        {
            System.out.println("Correct - getHead() returned null");
        }
        else
        {
            System.out.println("Incorrect - getHead() did not returned null");
        }

//************************ Check if the list is empty ************************
        System.out.println("\nCheck if the list is empty:");
        if(nameList.isEmpty())
        {
            System.out.println("This list is empty");
        }
        else
        {
            System.out.println("This list is not empty");
        }

//************************ Add "Tom" to the empty list ************************
```

```
nameList.insertInOrder("Tom");

//************************ Display the list ************************
System.out.println("\nDisplay a list after add \"Tom\":");
nameList.display();

//************************** Use getHead() ***************************
System.out.println("\nCall getHead() on the list and display the result:");
if(nameList.getHead() == null)
{
     System.out.println("getHead() returned null");
}
else
{
     System.out.println("getHead() did not returned null");
}

//************************** Use isEmpty() ***************************
System.out.println("\nCheck if the list is empty:");
if(nameList.isEmpty())
{
     System.out.println("This list is empty");
}
else
{
     System.out.println("This list is not empty");
```

```
        }

        //************************ Use remove("Tom") ************************
        System.out.println("\nRemove \"Tom\":");
        nameList.remove("Tom");
        System.out.println("The list after removing \"Tom\":");
        nameList.display();

        //*************** Use isEmpty() after removing "Tom" ******************
        System.out.println("\nCheck if the list is empty:");
        if(nameList.isEmpty())
        {
             System.out.println("This list is empty");
        }
        else
        {
             System.out.println("This list is not empty");
        }

//********************* Add and display a list with 5 names *******************
        String[] names = {"Tom", "May", "Darcy", "Ben", "Jeff"};
        for(int i = 0; i < names.length; i++)
        {
             nameList.insertInOrder(names[i]);
        }
        System.out.println("\nDisplay a list with five names:");
```

```
            nameList.display();

//*************** Attempt to remove the last node of the list ***************
            int count = 0;
            Node temp = nameList.getHead();
            while(temp!=null)
            {
                 count++;
                 temp = temp.getNext();
            }
            int count2 = 0;
            temp = nameList.getHead();
            while(nameList.getHead().getNext()!=null)
            {
                 count2++;
                 temp = temp.getNext();
                 if(count2 == (count/2))
                 {
                      nameList.remove(temp.getName());
                      System.out.println("\nDisplay a list after remove the node from the " +
                                "middle of the list:");
                      nameList.display();
                      break;
                 }
            }
```

```
//*************** Attempt to remove the first node of the list ***************
          nameList.remove(nameList.getHead().getName());
          System.out.println("\nDisplay a list after remove the first node from the list:");
          nameList.display();

//*************** Attempt to remove the last node of the list ***************
          temp = nameList.getHead();
          while(nameList.getHead().getNext()!=null)
          {
               temp = temp.getNext();
               if(temp.getNext() == null)
               {
                    nameList.remove(temp.getName());
                    System.out.println("\nDisplay a list after remove the last node from " +
                              "the list:");
                    nameList.display();
                    break;
               }
          }
     }
}
```

Output:

```
Display an empty list:
*** Start of list ***
```

```
*** End of list ***

Remove a name from the empty list:
Correct – remove() returned false

Call getHead() on the empty list and display the result:
Correct - getHead() returned null

Check if the list is empty:
This list is empty

Display a list after add "Tom":
*** Start of list ***
Tom
*** End of list ***

Call getHead() on the list and display the result:
getHead() did not returned null

Check if the list is empty:
This list is not empty

Remove "Tom":
The list after removing "Tom":
*** Start of list ***
*** End of list ***
```

```
Check if the list is empty:
This list is empty

Display a list with five names:
*** Start of list ***
Ben
Darcy
Jeff
May
Tom
*** End of list ***

Display a list after removing the node from the middle of the list:
*** Start of list ***
Ben
Darcy
May
Tom
*** End of list ***

Display a list after removing the first node from the list:
*** Start of list ***
Darcy
May
Tom
```

```
*** End of list ***

Display a list after removing the last node from the list:
*** Start of list ***
Darcy
May
*** End of list ***
```

Part B – Enhancing the Linked List

Source codes:

Node.java:

```
/** Enables names to be stored within a linked list
     @author Ngoc Phuong Anh Nguyen - 3712361
*/
public class Node
{
     /**  The data stored by this node - a name
     */
     private String name;

     /** Points to the next node in the list
     */
     private Node next;

     /**
      * Points to the previous node in the list.
      */
     private Node previous;

     /**
      * Constructs an empty node
     */
     public Node()
```

```
{
     name = null;
     next = null;
     previous = null;
}

/** Constructs a node containing the given name
     @param nameIn The name to be inserted into
                   the new node
*/
public Node(String nameIn)
{
     name = nameIn;
     next = null;
     previous = null;
}

// Accessor and mutator methods

public void setName(String nameIn)
{
     name = nameIn;
}

public void setNext(Node nextIn)
{
```

```
        next = nextIn;
    }

    public String getName()
    {
        return name;
    }

    public Node getNext()
    {
        return next;
    }

    public Node getPrevious()
    {
        return previous;
    }

    public void setPrevious(Node previousIn)
    {
        previous = previousIn;
    }
}
```

LinkedList.java:

```
/**
 * A simple linked list class.
 * @author Ngoc Phuong Anh Nguyen - 3712361
*/
public class LinkedList
{
     /**
      * Points to the first node in the list.
           Is null if and only if the list is empty.
     */
     private Node head;

     /**
      * Points to the last node in the list.
      */
     private Node tail;
     /**
      * Number of Nodes in the list.
      */
     private int size = 0;
     /** Constructs an empty list
     */
     public LinkedList()
     {
```

```
     head = null;
}

/**  Accessor for head
     @return The head of the list
*/
public Node getHead()
{
     return head;
}

/**  Indicates if the list is empty
     @return true if the list is empty, false
             otherwise
*/
public boolean isEmpty()
{
     return head == null;
}

/**
 * Displays the names in the list, one name per line
*/
public void display()
{
     System.out.println("*** Start of list ***");
```

```
     Node current = head;
     while (current != null)
     {    System.out.println(current.getName());
          current = current.getNext();
     }
     System.out.println("*** End of list ***");
}

/** Creates a new Node with the given name.
     Inserts this new Node into the list so the list
     is maintained in ascending order alphabetically
     by name. Assumes the list is sorted prior to
     this operation. Duplicate names are
     permitted in the list.
     @param nameIn The name to be inserted into
                   the list
*/
public void insertInOrder(String nameIn)
{
     Node newNode = new Node(nameIn);

     if(head == null)
     {
          head = tail = newNode;
          head.setPrevious(null);
          tail.setNext(null);
```

```
        size++;
        return;
    }

    boolean found = false;
    Node current = head;
    Node previous = null;

    while (!found)
    {
        if(nameIn.compareTo(current.getName()) <= 0)
      {
          found = true;
      }
        else
        {
            previous = current;
            current = current.getNext();
            found = current == null;
            if (found) tail = previous;
        }
    } // end while

if (previous == null)
{       // Insert ahead of the first node
    newNode.setNext(head);
```

```
        head.setPrevious(newNode);
        head = newNode;
      }
   else
   {
          newNode.setNext(previous.getNext());
          previous.setNext(newNode);
     newNode.setPrevious(previous);

     }
     size++;
}

/**  Searches the list for a node containing nameIn. If
     found, removes that node from the list and returns
     true. Otherwise, makes no change to the list and
     returns false.
     @param nameIn The name to remove from the list
     @return true if a node is removed, false otherwise
*/
public boolean remove(String nameIn)
{
if (head == null)
          return false;

// The list has at least one Node.
```

```
// Find the given name in the list.

Node current  = head;

boolean found = false;

while (!found)
{
     if(nameIn.equals(current.getName()))
               found = true;
          else
          {
          current = current.getNext();
          found = current == null;
          }
} // end while

// Remove only if the name was found

if (current != null)
{    // The name was found.

     if (head == current)
          head = current.getNext();

          if(current.getNext() != null)
```

```
            {
                current.getNext().setPrevious(current.getPrevious());
            }

            if(current.getPrevious() != null)
            {
                current.getPrevious().setNext(current.getNext());
            }
            size--;
            return true;
}
     return false;
} // end remove

public Node getTail()
{
     return tail;
}

public int getSize()
{
     return size;
}

/**
 * Displays the list in the same fashion as the current
```

```
 * display() method, but in reverse order
 */
public void displayInReverse()
{
     System.out.print("*** Start of list ***");
     Node current = head;
     String output ="";
     while (current != null)
     {
          output = "\n" + current.getName() + output;
          current = current.getNext();
     }
     System.out.println(output);
     System.out.println("*** End of list ***");
}

/**
 * Inserts a new Node with the given name at the
 * beginning of the list.
 * @param nameIn The input name
 */
public void insertAtHead(String nameIn)
{
     Node newNode = new Node(nameIn);
     if(head == null)
     {
```

```
            head = tail = newNode;
            head.setPrevious(null);
            tail.setNext(null);
        }
        else
        {
            newNode.setNext(head);
            head.setPrevious(newNode);
            head = newNode;
        }
        size++;
    }

    /**
     * Inserts a new Node with the given name at the
     * end of the list.
     * @param nameIn The input name.
     */
    public void insertAtTail(String nameIn)
    {
        Node newNode = new Node(nameIn);

        if(head == null)
        {
            head = tail = newNode;
            head.setPrevious(null);
```

```
        }
        else
        {
            tail.setNext(newNode);
            newNode.setPrevious(tail);
            tail = newNode;
        }
        tail.setNext(null);
        size++;

    }

    /**
     * Removes the first Node in the list.
     * @return True if successful, false otherwise.
     */
    public boolean removeAtHead()
    {
        Node current = head;
        if(head == null)
        {
            return false;
        }
        head = current.getNext();
        size--;
        return true;
```

```
    }

    /**
     * Removes the last Node in the list.
     * @return True if successful, false otherwise.
     */
    public boolean removeAtTail()
    {
        if(head == null)
        {
            return false;
        }
        if (tail.getPrevious()!=null)
        {
            tail = tail.getPrevious();
            tail.setNext(null);
            size--;
            return true;
        }
        else
        {
            return false;
        }

    }
} // end LinkedList class
```

ListTest.java:

```
/**
 * This is a driver program.
 * @author Ngoc Phuong Anh Nguyen - 3712361
 */

public class ListTest
{
     public static void main(String[] args)
     {
          LinkedList nameList = new LinkedList();

//******************************** Empty list ********************************
          System.out.println("\n********** Empty list **********");
          System.out.println("\nDisplay a empty list:");
          nameList.display();
          System.out.println("\nDisplay the list in reverse:");
          nameList.displayInReverse();

          System.out.println("\nSize of the list: " + nameList.getSize());

          if(nameList.getSize() == 0)
          {
               System.out.println("Correct - size() returns 0");
          }
```

```
        else
        {
             System.out.println("incorrect – size() should returns 0");
        }

        nameList.insertAtTail("Kelvin");
        System.out.println("\nDisplay a list after input at the end:");
        nameList.display();

        nameList.insertAtHead("Anna");
        System.out.println("\nDisplay a list after input at the beginning:");
        nameList.display();

        nameList.removeAtTail();
        System.out.println("\nDisplay a list after remove at the end:");
        nameList.display();

        nameList.removeAtHead();
        System.out.println("\nDisplay a list after remove at the beginning:");
        nameList.display();

//*************************** List with one name ****************************
        System.out.println("\n********** List with one name **********");
        nameList.insertInOrder("Tom");
        System.out.println("\nDisplay a list after input \"Tom\":");
        nameList.display();
```

```
System.out.println("\nDisplay the list in reverse:");
nameList.displayInReverse();

System.out.println("\nSize of the list: " + nameList.getSize());

if(nameList.getSize() == 1)
{
     System.out.println("Correct - size() returns 1");
}
else
{
     System.out.println("incorrect - size() should returns 1");
}
nameList.insertAtTail("Johan");
System.out.println("\nDisplay a list after input at the end:");
nameList.display();

nameList.insertAtHead("Andrew");
System.out.println("\nDisplay a list after input at the beginning:");
nameList.display();

nameList.removeAtTail();
System.out.println("\nDisplay a list after remove at the end:");
nameList.display();

nameList.removeAtHead();
```

```
        System.out.println("\nDisplay a list after remove at the beginning:");
        nameList.display();

        nameList.remove("Tom");

//************************** List with five names ***************************
        System.out.println("\n********** List with five names **********");
        String[] names = {"Tom", "May", "Darcy", "Ben", "Jeff"};

        for(int i = 0; i < names.length; i++)
        {
             nameList.insertInOrder(names[i]);
        }

        System.out.println("\nDisplay a list after input 5 names:");
        nameList.display();
        System.out.println("\nDisplay the list in reverse:");
        nameList.displayInReverse();

        System.out.println("\nSize of the list: " + nameList.getSize());

        if(nameList.getSize() == 5)
        {
             System.out.println("Correct - size() returns 5");
        }
        else
```

```
        {
             System.out.println("incorrect - size() should returns 5");
        }

        nameList.insertAtTail("Emily");
        System.out.println("\nDisplay a list after input at the end:");
        nameList.display();

        nameList.insertAtHead("James");
        System.out.println("\nDisplay a list after input at the beginning:");
        nameList.display();

        nameList.removeAtTail();
        System.out.println("\nDisplay a list after remove at the end:");
        nameList.display();

        nameList.removeAtHead();
        System.out.println("\nDisplay a list after remove at the beginning:");
        nameList.display();
    }
}
```

Output:

```
********** Empty list **********
```

```
Display a empty list:
*** Start of list ***
*** End of list ***

Display the list in reverse:
*** Start of list ***
*** End of list ***

Size of the list: 0
Correct - size() returns 0

Display a list after input at the end:
*** Start of list ***
Kelvin
*** End of list ***

Display a list after input at the beginning:
*** Start of list ***
Anna
Kelvin
*** End of list ***

Display a list after remove at the end:
*** Start of list ***
Anna
```

```
*** End of list ***

Display a list after remove at the beginning:
*** Start of list ***
*** End of list ***

********** List with one name **********

Display a list after input "Tom":
*** Start of list ***
Tom
*** End of list ***

Display the list in reverse:
*** Start of list ***
Tom
*** End of list ***

Size of the list: 1
Correct - size() returns 1

Display a list after input at the end:
*** Start of list ***
Tom
Johan
*** End of list ***
```

```
Display a list after input at the beginning:
*** Start of list ***
Andrew
Tom
Johan
*** End of list ***

Display a list after remove at the end:
*** Start of list ***
Andrew
Tom
*** End of list ***

Display a list after remove at the beginning:
*** Start of list ***
Tom
*** End of list ***

********** List with five names **********

Display a list after input 5 names:
*** Start of list ***
Ben
Darcy
Jeff
```

```
May
Tom
*** End of list ***

Display the list in reverse:
*** Start of list ***
Tom
May
Jeff
Darcy
Ben
*** End of list ***

Size of the list: 5
Correct - size() returns 5

Display a list after input at the end:
*** Start of list ***
Ben
Darcy
Jeff
May
Tom
Emily
*** End of list ***
```

```
Display a list after input at the beginning:
*** Start of list ***
James
Ben
Darcy
Jeff
May
Tom
Emily
*** End of list ***

Display a list after remove at the end:
*** Start of list ***
James
Ben
Darcy
Jeff
May
Tom
*** End of list ***

Display a list after remove at the beginning:
*** Start of list ***
Ben
Darcy
Jeff
```

May

Tom

*** End of list ***