**StinglyLinkedList Class**

/\*\*

\*

\* @author Rich

\* @version 03/16/2017

\* This class consist of method for creations of nodes, and methods on how to manipulate nodes

\* data structure.

\*/

public class StinglyLinkedList<E> {

private static class Node<E>{

private E element;

private Node<E> next;

public Node(E e, Node<E> n)

{

element = e;

next = n;

}

/\*\*

\*

\* @return the element in the node.

\*/

public E getElement()

{

return element;

}

/\*\*

\*

\* @return the pointer to the next node of the list.

\*/

public Node<E> getNext()

{

return next;

}

// setters

/\*\*

\*

\* @param newNext set the pointer to point to the next item

\*/

public void setNext(Node<E> newNext)

{

next = newNext;

}

}

private Node<E> head = null; // point to 1st node of the list

private Node<E> tail = null; // pointer to the last node of the list

private int size = 0;

public StinglyLinkedList(){} // construcst an initially empty list.

/\*\*

\*

\* @return how many nodes in the list

\*/

public int size()

{

return size;

}

/\*\*

\*

\* @return true if list is empty.

\*/

public boolean isEmpty()

{

return size==0;

}

/\*\*

\*

\* @return the first element in the first node

\*/

public E first()

{

if (isEmpty())

return null;

return head.getElement();

}

/\*\*

\*

\* @return the last element in the list

\*/

public E last()

{

if(isEmpty()) return null;

return tail.getElement();

}

/\*\*

\*

\* @param e generic element to be place as first element

\*/

public void addFirst(E e)

{

head = new Node(e,head);

if (size == 0)

tail = head;

size++;

}

/\*\*

\*

\* @param e generic type element to set as last

\*/

public void addLast(E e)

{

Node<E> newest = new Node(e, null);

if(isEmpty())

{

head = newest;

}

else

tail.setNext(newest);

tail = newest;

size++;

}

/\*\*

\*

\* @return the element in the 1st node removed

\*/

public E removeFirst()

{

if (isEmpty())

return null;

E answer = head.getElement();

head = head.getNext();

size--;

if(size == 0)

tail = null;

return answer;

}

// my removeLast

/\*\*

\*

\* @return the removed item

\*/

public E removeLast()

{

if(isEmpty()) return null;

E answer = tail.getElement();

Node<E> current = head, previous = head;

while(current.getNext() != null)

{

previous = current; // the one before to last node/tail.

current = current.getNext();

}

// after exiting while loop current holds the memRef of tail, it is pointing to tail. current =tail

previous.setNext(null); // break the chain btw the one- before last and last node.

tail = previous;

return answer;

}

/\*\*

\*

\* @return String format of the object

\*/

@Override

public String toString()

{

String listElements = "";

Node<E> current = head;

while(current != null)

{

listElements += current.getElement() +"-->";

current = current.getNext(); // update current to point to next node in the listt.

}

return listElements;

}

/\*\*

\*

\* @param o object ref

\* @return true if two linked list are equal

\*/

public boolean equals(Object o)

{

if(!(o instanceof StinglyLinkedList))

return false;

StinglyLinkedList l = (StinglyLinkedList) o;

if (size != l.size())

return false;

Node<E> sourceCurrentNodePtr = head; // current node pointer/Refvar for the "blueprint" list.

Node<E> targetCurrentNodePtr = l.head; // identifiers that points to current head of the list we're testing for equality.

while(sourceCurrentNodePtr != null)

{

if(!sourceCurrentNodePtr.getElement().equals(targetCurrentNodePtr.getElement()))

return false;

sourceCurrentNodePtr = sourceCurrentNodePtr.getNext(); // updtate memory pointer.; advancing current to next Node.

targetCurrentNodePtr = targetCurrentNodePtr.getNext();

}

return true;

}

}

**Client**

/\*\*

\*

\* @author Rich

\*/

import java.util.Random;

public class StinglyListTest {

public static void main(String[] args) {

StinglyLinkedList<Integer> linkedList = new StinglyLinkedList<>();

Random rand = new Random();

for (int i = 0; i < 10; ++i)

{

int randNum = rand.nextInt(101);

linkedList.addFirst(randNum);

}

System.out.println("Printing the contents of the List\n" + linkedList.toString());

System.out.println("The first Element in the list: " + linkedList.first());

System.out.println("The last Element in the list " + linkedList.last() );

StinglyLinkedList<Integer> linkedList2 = new StinglyLinkedList<>();

linkedList2.addLast(1);

linkedList2.addLast(2);

linkedList2.addLast(3);

System.out.println("The contents of List # 2\n" + linkedList2.toString());

StinglyLinkedList<Integer> linkedList3 = new StinglyLinkedList<>();

linkedList3.addFirst(3);

linkedList3.addFirst(2);

linkedList3.addFirst(1);

System.out.println("The contents of List # 3\n" + linkedList3.toString());

System.out.println("Are List 2 and List 3 equal? " + linkedList2.equals(linkedList3));

linkedList2.removeFirst();

System.out.println("The contents of List # 2 after removal of first node\n" + linkedList2.toString());

linkedList3.removeLast();

System.out.println("The contents of List # 3 after remove of last element\n" + linkedList3.toString());

System.out.println("Are List 3 and List 2 equal?" + linkedList2.equals(linkedList3));

}

}

**Output**

run:

Printing the contents of the List

4-->92-->14-->41-->64-->13-->54-->84-->32-->46-->

The first Element in the list: 4

The last Element in the list 46

The contents of List # 2

1-->2-->3-->

The contents of List # 3

1-->2-->3-->

Are List 2 and List 3 equal? true

The contents of List # 2 after removal of first node

2-->3-->

The contents of List # 3 after remove of last element

1-->2-->

Are List 3 and List 2 equal?false

BUILD SUCCESSFUL (total time: 0 seconds)