|  |
| --- |
| @@ -0,0 +1,86 @@ |
|  |  | package airen; |
|  |  |  |
|  |  | public class SinglyLinkedList<E> { |
|  |  | // ---------------- nested Node class ---------------- |
|  |  | private static class Node<E> { |
|  |  | private E element; // reference to the element stored at this node |
|  |  | private Node<E> next; // reference to the subsequent node in the list |
|  |  |  |
|  |  | public Node(E e, Node<E> n) { |
|  |  | element = e; |
|  |  | next = n; |
|  |  | } |
|  |  |  |
|  |  | public E getElement() { |
|  |  | return element; |
|  |  | } |
|  |  |  |
|  |  | public Node<E> getNext() { |
|  |  | return next; |
|  |  | } |
|  |  |  |
|  |  | public void setNext(Node<E> n) { |
|  |  | next = n; |
|  |  | } |
|  |  | } // ----------- end of nested Node class ----------- |
|  |  |  |
|  |  | // instance variables of the SinglyLinkedList |
|  |  | private Node<E> head = null; // head node of the list (or null if empty) |
|  |  | private Node<E> tail = null; // last node of the list (or null if empty) |
|  |  | private int size = 0; // number of nodes in the list |
|  |  |  |
|  |  | public SinglyLinkedList() { } // constructs an initially empty list |
|  |  |  |
|  |  | // access methods |
|  |  | public int size(){ |
|  |  | int len = 0; |
|  |  | Node<E> node; |
|  |  | for(node = head; node != null ;node=node.getNext()){ |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  | len ++; |
|  |  | } |
|  |  | return len; |
|  |  | } |
|  |  | public boolean isEmpty() { |
|  |  | return size == 0; |
|  |  | } |
|  |  | public E first() { // returns (but does not remove) the first element |
|  |  | if (isEmpty()) |
|  |  | return null; |
|  |  | return head.getElement(); |
|  |  | } |
|  |  | public E last() { // returns (but does not remove) the last element |
|  |  | if (isEmpty()) |
|  |  | return null; |
|  |  | return tail.getElement(); |
|  |  | } |
|  |  |  |
|  |  | // update methods |
|  |  | public void addFirst(E e) { // adds element e to the front of the list |
|  |  | head = new Node<>(e, head); // create and link a new node |
|  |  | if (size == 0) |
|  |  | tail = head; // special case: new node becomes tail also |
|  |  | size++; |
|  |  | } |
|  |  | public void addLast(E e) { // adds element e to the end of the list |
|  |  | Node<E> newest = new Node<>(e, null); // node will eventually be the tail |
|  |  | if (isEmpty()) |
|  |  | head = newest; // special case: previously empty list |
|  |  | else |
|  |  | tail.setNext(newest); // new node after existing tail |
|  |  | tail = newest; // new node becomes the tail |
|  |  | size++; |
|  |  | } |
|  |  | public E removeFirst() { // removes and returns the first element |
|  |  | if (isEmpty()) |
|  |  | return null; // nothing to remove |
|  |  | E answer = head.getElement(); |
|  |  | head = head.getNext(); // will become null if list had only one node |
|  |  | size--; |
|  |  | if (size == 0) |
|  |  | tail = null; // special case as list is now empty |
|  |  | return answer; |
|  |  | } |
|  |  | } |