Single Circular Link List

Used Node and Class/Interface in single link list

---------------------------------------------------------------

public interface Lap<K, V> {

public K getHead();

public V getTail();

public void put(K k, V v);

}

===========================

public class SLHashLap implements Lap<SLNode<Integer>, SLNode<Integer>> {

public SLNode<Integer> head = null;

private SLNode<Integer> tail = null;

@Override

public SLNode<Integer> getHead() {

return head;

}

public void setHead(SLNode<Integer> head) {

this.head = head;

}

@Override

public SLNode<Integer> getTail() {

return tail;

}

public void setTail(SLNode<Integer> tail) {

this.tail = tail;

}

@Override

public void put(SLNode<Integer> k, SLNode<Integer> v) {

setHead(k);

setTail(v);

}

}

public class SLNode<K> {

public K k;

public SLNode<K> next;

public SLNode(K k) {

this.k = k;

this.next = null;

}

}

Used interface

----------------------------------------------------

/\*\* 2. Circular Linked List Traversal \*\*/

/\* insert collection of element \*/

**public** Lap<SLNode<K>, SLNode<K>> insertArrLast(Lap<SLNode<K>, SLNode<K>> lap, K k[]);

/\* insert single element \*/

**public** Lap<SLNode<K>, SLNode<K>> insertElementLast(Lap<SLNode<K>, SLNode<K>> lap, K k);

**public** **void** printList(SLNode<K> k);

/\*\* 3. Split a Circular Linked List into two halves \*\*/

**public** Lap<SLNode<K>, SLNode<K>> splitTwoHalves(SLNode<K> k);

/\*\* 4. Sorted insert for circular linked list \*\*/

**public** SLNode<K> insertArrSorted(SLNode<K> head, K k[]);

/\*\* 8. Deletion from a Circular Linked List \*\*/

**public** SLNode<Integer> deleteAllX(SLNode<K> head, K x);

=====================================================================================Implementation of interface

-------------------------------------------------------------------------------------

**public** **class** SCLinkListImpl **implements** ISCLinkList<Integer> {

**public** SLNode<Integer> head = **null**;

**public** SLNode<Integer> tail = **null**;

/\*\* 3. Linked List Insertion \*\*/

@Override

**public** Lap<SLNode<Integer>, SLNode<Integer>> insertArrLast(Lap<SLNode<Integer>, SLNode<Integer>> lap, Integer a[]) {

**if** (a == **null** || a.length == 0)

**return** lap;

**if** (lap == **null**)

lap = **new** SLHashLap();

head = lap.getHead();

tail = lap.getTail();

**int** i = 0;

**if** (head == **null**) {

head = tail = **new** SLNode<Integer>(a[0]);

i++;

}

**for** (; i < a.length; i++) {

tail.next = **new** SLNode<Integer>(a[i]);

tail = tail.next;

}

tail.next = head;// this is makes list circular

lap.put(head, tail);

**return** lap;

}

/\* insert single element \*/

@Override

**public** Lap<SLNode<Integer>, SLNode<Integer>> insertElementLast(Lap<SLNode<Integer>, SLNode<Integer>> lap,

Integer x) {

SLNode<Integer> temp = **new** SLNode<Integer>(x);

**if** (lap == **null**) {

lap = **new** SLHashLap();

head = tail = temp;

} **else** {

tail = lap.getTail();

tail.next = temp;

tail = tail.next;

}

lap.put(head, tail);

**return** lap;

}

/\* print the link list nodes \*/

**public** **void** printList(SLNode<Integer> node) {

SLNode<Integer> head = node;

**for** (; node.next != head; System.***out***.print(node.k + "->"), node = node.next)

;

System.***out***.print(node.k + "->");

}

/\*\* 3. Split a Circular Linked List into two halves \*\*/

@Override

**public** Lap<SLNode<Integer>, SLNode<Integer>> splitTwoHalves(SLNode<Integer> head) {

SLHashLap lap = **new** SLHashLap();

SLNode<Integer> p, q, h2;

p = q = head;

**while** (p.next != head && p.next.next != head) {

q = q.next;

p = p.next.next;

}

h2 = q.next;

q.next = head;

p.next = h2;

lap.setHead(head);

lap.setTail(h2);

**return** lap;

}

/\*\* 4. Sorted insert for circular linked list \*\*/

@Override

**public** SLNode<Integer> insertArrSorted(SLNode<Integer> head, Integer a[]) {

SLNode<Integer> p = **null**;

**for** (**int** i = 0; i < a.length; i++) {

SLNode<Integer> temp = **new** SLNode<Integer>(a[i]);

**if** (head == **null**) {

head = tail = temp;

} **else** **if** (a[i] <= head.k) { // = applied later

temp.next = head;

head = temp;

} **else** {

p = head;

**while** (p.next != **null**) {

**if** (a[i] <= p.next.k) { // = applied later

temp.next = p.next;

p.next = temp;

**break**;

}

p = p.next;

}

**if** (p.next == **null**)

p.next = temp;

}

}

p = head;

**while** (p.next != **null**)

p = p.next;

tail = p;

tail.next = head;

**return** head;

}

@Override

**public** SLNode<Integer> deleteAllX(SLNode<Integer> head, Integer x) {

SLNode<Integer> tail = head;

**while** (tail.next != head)

tail = tail.next;

tail.next = **null**;// make it non circular list

**boolean** isFound = **false**;

**if** (head == **null**)

**return** **null**;

SLNode<Integer> q = head;

**while** (head != **null** && head.k == x) {

head = head.next;

q.next = **null**;

q = head;

isFound = **true**;

}

SLNode<Integer> node = head.next;

SLNode<Integer> prev = head;

**for** (; node != **null**; prev = node, node = node.next) {

**if** (node.k == x) {

q = node;

prev.next = q.next;

q.next = **null**;

q = **null**;

node = prev;

isFound = **true**;

}

}

// connecect tail to head

tail = head;

**while** (tail.next != **null**)

tail = tail.next;

tail.next = head;

**if** (isFound)

System.***out***.println("\nall: " + x + ": is deleted");

**else**

System.***out***.println(x + ": is not found");

**return** head;

}

}

=====================================================================================

Test case:

-----------------------------------------------------------------

**public** **class** ISCLinkListTest {

**public** ISCLinkList<Integer> iscl = **null**;

@Before

**public** **void** init() {

iscl = **new** SCLinkListImpl();

}

/\*\* 3. Linked List Insertion \*\*/

@Test

**public** **void** insertArrTest() {

Integer a[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9 };

Lap<SLNode<Integer>, SLNode<Integer>> lap = iscl.insertArrLast(**null**, a);

SLNode<Integer> head = lap.getHead();

iscl.printList(head);

// 1->2->3->4->5->6->7->8->9->

Assert.*assertTrue*(head.k == 1);

Assert.*assertTrue*(head.next.next.k == 3);

Assert.*assertTrue*(head.next.next.next.next.k == 5);

Integer b[] = { 10, 11, 12 };

lap = iscl.insertArrLast(lap, b);

// iscl.printList(head);

// 1->2->3->4->5->6->7->8->9->10->11->12->

}

/\* insert single element \*/

@Test

**public** **void** insertElementLastTest() {

Integer a[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9 };

Lap<SLNode<Integer>, SLNode<Integer>> lap = iscl.insertArrLast(**null**, a);

SLNode<Integer> head = lap.getHead();

// isl.printList(head);

Assert.*assertTrue*(head.k == 1);

Assert.*assertTrue*(head.next.next.k == 3);

Assert.*assertTrue*(head.next.next.next.next.k == 5);

Integer b[] = { 10, 11, 12 };

lap = iscl.insertArrLast(lap, b);

System.***out***.println();

iscl.printList(head);

}

/\*\* 3. Split a Circular Linked List into two halves \*\*/

@Test

**public** **void** splitTwoHalvesTest() {

Integer a[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9 };

SLNode<Integer> head = iscl.insertArrLast(**null**, a).getHead();

Lap<SLNode<Integer>, SLNode<Integer>> lap = iscl.splitTwoHalves(head);

SLNode<Integer> h1 = lap.getHead();

SLNode<Integer> h2 = lap.getTail();

iscl.printList(h1);

System.***out***.println();

iscl.printList(h2);

}

/\*\* 4. Sorted insert for circular linked list \*\*/

/\* insert collection of element in sorted order \*/

@Test

**public** **void** insertArrSortedTest() {

Integer a[] = { 8, 7, 9, 2, 3, 4, 0, 1, 5, 6 };

SLNode<Integer> head = iscl.insertArrSorted(**null**, a);

System.***out***.println("\nprintSorted order list");

iscl.printList(head);

Assert.*assertTrue*(head.k == 0);

Assert.*assertTrue*(head.next.next.k == 2);

Assert.*assertTrue*(head.next.next.next.next.k == 4);

}

@Test

**public** **void** deleteAllXTest() {

SLNode<Integer> head = **null**;

Integer a[] = { 1, 1, 0, 1, 0, 2, 3, 1 };

Integer b[] = { 1, 2, 2, 3, 2, 4, 2, 5, 2 };

Lap<SLNode<Integer>, SLNode<Integer>> lap = iscl.insertArrLast(**null**, a);

head = lap.getHead();

System.***out***.println();

iscl.printList(head);

head = iscl.deleteAllX(head, 1);

System.***out***.println();

iscl.printList(head);

Assert.*assertTrue*(head.k == 0);

Assert.*assertTrue*(head.next.k == 0);

Assert.*assertTrue*(head.next.next.k == 2);

head = **null**;

System.***out***.println();

head = iscl.insertArrLast(**null**, b).getHead();

iscl.printList(head);

head = iscl.deleteAllX(head, 2);

System.***out***.println();

iscl.printList(head);

Assert.*assertTrue*(head.k == 1);

Assert.*assertTrue*(head.next.k == 3);

Assert.*assertTrue*(head.next.next.k == 4);

}

}