package lab6;

public class Lab6 {

public static void main(String[] args) {

// Create a list for strings

MyLinkedList<String> list = new MyLinkedList<String>();

// Add elements to the list

list.add("America"); // Add it to the list

System.out.println("(1) " + list);

list.add(0, "Canada"); // Add it to the beginning of the list

System.out.println("(2) " + list);

list.add("Russia"); // Add it to the end of the list

System.out.println("(3) " + list);

list.addLast("France"); // Add it to the end of the list

System.out.println("(4) " + list);

list.add(2, "Germany"); // Add it to the list at index 2

System.out.println("(5) " + list);

list.add(5, "Norway"); // Add it to the list at index 5

System.out.println("(6) " + list);

list.add(0, "Poland"); // Same as list.addFirst("Poland")

System.out.println("(7) " + list);

// Remove elements from the list

list.remove(0); // Same as list.remove("Australia") in this case

System.out.println("(8) " + list);

list.remove(4); // Remove the element at index 2

System.out.println("(9) " + list);

list.remove(list.size() - 1); // Remove the last element

System.out.println("(10) " + list);

/\* Below is the code to test methods that should be implemented in Lab 8\*/

String s = "Germany";

if (list.contains(s))

System.out.println("(11) The list contains " + s);

else

System.out.println("(11) The list does not contain " + s);

int pos = 4;

s = list.get(pos);

if ( s != null)

System.out.println("(12) The list element at position " +pos+ " is " + s);

else

System.out.println("(12) Invalid position");

s = "France";

pos = list.indexOf(s);

if (pos != -1)

System.out.println("(13) The list element " + s + "is at position " + pos);

else

System.out.println("(13) The list does not contain " + s);

list.addFirst("India"); // Add it to the end of the list

System.out.println("(14) " + list);

list.add(6,"America"); // Add it to the end of the list

System.out.println("(15) " + list);

s = "America";

pos = list.lastIndexOf(s);

if (pos != -1)

System.out.println("(16) The list element " + s + " occurs last at " + pos);

else

System.out.println("(16) The list does not contain " + s);

list.set(5,"China");

System.out.println("(17) " + list);

}//main

}//class

Output:

run:

(1) [America]

(2) [Canada, America]

(3) [Canada, America, Russia]

(4) [Canada, America, Russia, France]

(5) [Canada, America, Germany, Russia, France]

(6) [Canada, America, Germany, Russia, France, Norway]

(7) [Poland, Canada, America, Germany, Russia, France, Norway]

(8) [Canada, America, Germany, Russia, France, Norway]

(9) [Canada, America, Germany, Russia, Norway]

(10) [Canada, America, Germany, Russia]

(11) The list contains Germany

(12) Invalid position

(13) The list does not contain France

(14) [India, Canada, America, Germany, Russia]

(15) [India, Canada, America, Germany, Russia, America]

(16) The list element America occurs last at 5

(17) [India, Canada, America, Germany, Russia, China]

BUILD SUCCESSFUL (total time: 0 seconds)

Interface Class:

package lab6;

/\* DO NOT CHANGE THIS CODE \*/

public interface MyList<E> {

/\*\* Add a new element at the end of this list \*/

public void add(E e);

/\*\* Add a new element at the specified index in this list \*/

public void add(int index, E e);

/\*\* Clear the list \*/

public void clear();

/\*\* Return true if this list contains the element \*/

public boolean contains(E e);

/\*\* Return the element from this list at the specified index \*/

public E get(int index);

/\*\* Return the index of the first matching element in this list.

\* Return -1 if no match. \*/

public int indexOf(E e);

/\*\* Return true if this list contains no elements \*/

public boolean isEmpty();

/\*\* Return the index of the last matching element in this list

\* Return -1 if no match. \*/

public int lastIndexOf(E e);

/\*\* Remove the first occurrence of the element o from this list.

\* Shift any subsequent elements to the left.

\* Return true if the element is removed. \*/

public boolean remove(E e);

/\*\* Remove the element at the specified position in this list

\* Shift any subsequent elements to the left.

\* Return the element that was removed from the list. \*/

public E remove(int index);

/\*\* Replace the element at the specified position in this list

\* with the specified element and returns the new set. \*/

public E set(int index, E e);

/\*\* Return the number of elements in this list \*/

public int size();

}//interface

MyAbstractList:

package lab6;

/\* DO NOT CHANGE THIS CODE \*/

public abstract class MyAbstractList<E> implements MyList<E> {

protected int size = 0; // The size of the list

/\*\* Create a default list \*/

protected MyAbstractList() {

}

/\*\* Create a list from an array of objects \*/

protected MyAbstractList(E[] objects) {

for (int i = 0; i < objects.length; i++)

add(objects[i]);

}

/\*\* Add a new element at the end of this list \*/

public void add(E e) {

add(size, e);

}

/\*\* Return true if this list contains no elements \*/

public boolean isEmpty() {

return size == 0;

}

/\*\* Return the number of elements in this list \*/

public int size() {

return size;

}

/\*\* Remove the first occurrence of the element o from this list.

\* Shift any subsequent elements to the left.

\* Return true if the element is removed. \*/

public boolean remove(E e) {

if (indexOf(e) >= 0) {

remove(indexOf(e));

return true;

}

else

return false;

}// remove

}// abstract class

package lab6;

public class MyLinkedList<E> extends MyAbstractList<E> {

private Node<E> head, tail;

/\*\* Create a default list \*/

public MyLinkedList() {

}

/\*\* Create a list from an array of objects \*/

public MyLinkedList(E[] objects) {

super(objects);

}

/\*\* Return the head element in the list \*/

public E getFirst() {

if (size == 0) {

return null;

} else {

return head.element;

}

}

/\*\* Return the last element in the list \*/

public E getLast() {

if (size == 0) {

return null;

} else {

return tail.element;

}

}

/\*\* Add an element to the beginning of the list \*/

public void addFirst(E e) {

Node<E> newNode = new Node<E>(e); // Create a new node

newNode.next = head; // link the new node with the head

head = newNode; // head points to the new node

size++; // Increase list size

if (tail == null) // the new node is the only node in list

{

tail = head;

}

}

/\*\* Add an element to the end of the list \*/

public void addLast(E e) {

Node<E> newNode = new Node<E>(e); // Create a new for element e

if (tail == null) {

head = tail = newNode; // The new node is the only node in list

} else {

tail.next = newNode; // Link the new with the last node

tail = tail.next; // tail now points to the last node

}

size++; // Increase size

}

/\*\* Add a new element at the specified index in this list

\* The index of the head element is 0 \*/

public void add(int index, E e) {

if (index == 0) {

addFirst(e);

} else if (index >= size) {

addLast(e);

} else {

Node<E> current = head;

for (int i = 1; i < index; i++) {

current = current.next;

}

Node<E> temp = current.next;

current.next = new Node<E>(e);

(current.next).next = temp;

size++;

}

}

/\*\* Remove the head node and

\* return the object that is contained in the removed node. \*/

public E removeFirst() {

if (size == 0) {

return null;

} else {

Node<E> temp = head;

head = head.next;

size--;

if (head == null) {

tail = null;

}

return temp.element;

}

}

/\*\* Remove the last node and

\* return the object that is contained in the removed node. \*/

public E removeLast() {

if (size == 0) {

return null;

} else if (size == 1) {

Node<E> temp = head;

head = tail = null;

size = 0;

return temp.element;

} else {

Node<E> current = head;

for (int i = 0; i < size - 2; i++) {

current = current.next;

}

Node<E> temp = tail;

tail = current;

tail.next = null;

size--;

return temp.element;

}

}

/\*\* Remove the element at the specified position in this list.

\* Return the element that was removed from the list. \*/

public E remove(int index) {

if (index < 0 || index >= size) {

return null;

} else if (index == 0) {

return removeFirst();

} else if (index == size - 1) {

return removeLast();

} else {

Node<E> previous = head;

for (int i = 1; i < index; i++) {

previous = previous.next;

}

Node<E> current = previous.next;

previous.next = current.next;

size--;

return current.element;

}

}

/\*\* Override toString() to return elements in the list \*/

public String toString() {

StringBuilder result = new StringBuilder("[");

Node<E> current = head;

for (int i = 0; i < size; i++) {

result.append(current.element);

current = current.next;

if (current != null) {

result.append(", "); // Separate two elements with a comma

} else {

result.append("]"); // Insert the closing ] in the string

}

}

return result.toString();

}

/\*\* Clear the list \*/

public void clear() {

head = tail = null;

}

@Override

public boolean contains(E e){

Node <E> current = head;

for (int i = 0; i < size; i++) {

if(current.element.equals(e)){

return true;

}//if

else{

current = current.next;

}//else

}//for

return false;

}//contains

@Override

public E get(int index) {

if(index < 0 || index > size -1){

return null;

}//if

Node <E> current = head;

for (int i = 0; i < index; i++) {

current = current.next;

}//for

return current.element;

}//get

@Override

public int indexOf(E e) {

Node <E> current = head;

for (int i = 0; i < size; i++) {

if(current.element.equals(e))

return i;

}

current = current.next;

return -1;

}

@Override

public int lastIndexOf(E e) {

int lastIndex = -1;

Node <E> current = head;

for (int i = 0; i < size; i++) {

if(current.element.equals(e)){

lastIndex = i;

}

current = current.next;

}

return lastIndex;

}//lastIndexOf

@Override

public E set(int index, E e) {

if(index < 0 || index > size - 1){

return null;

}//if

Node <E> current = head;

for (int i = 0; i < index; i++) {

current = current.next;

}

E temp = current.element;

current.element = e;

return temp;

}//set

/\*\*

methods for lab should be implemented here

\*/

private static class Node<E> {

E element;

Node<E> next;

public Node(E element) {

this.element = element;

}

}

}