/\*

Alexander Giang

masc 2169

\*/

package data\_structures;

import java.util.Iterator;

import java.util.NoSuchElementException;

import java.util.ConcurrentModificationException;

public class UnorderedList<E> implements Iterable<E> {

private long changeCounter;

private int currentSize;

Node<E> head, tail;

public UnorderedList() {

head = null;

tail = null;

currentSize = 0;

changeCounter = 0;

}

// Insert at the head

public void insertFirst(E data) {

Node<E> newNode = new Node<E>(data);

if (head == null)

head = tail = newNode;

else {

newNode.next = head;

head = newNode;

}

currentSize++;

changeCounter++;

}

// Insert at the tail

public void insertLast(E data) {

Node<E> newNode = new Node<E>(data);

if (tail == null)

tail = head = newNode;

else {

tail.next = newNode;

tail = newNode;

}

currentSize++;

changeCounter++;

}

// Removes and returns the head, otherwise null

public E removeFirst() {

if (isEmpty())

return null;

E temp = head.data;

head = head.next;

currentSize--;

changeCounter++;

if (head == null)

tail = null;

return temp;

}

// Removes and returns the tail, otherwise null

public E removeLast() {

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

while (current != tail) {

previous = current;

current = current.next;

}

if (current == null) // if empty

return null;

E temp = tail.data;

if (previous == null) { // if one element

head = tail = null;

}

else {

previous.next = null;

tail = previous;

}

currentSize--;

changeCounter++;

return temp;

}

// Removes and returns the first instance of an object, otherwise null

public E removeFirstInstance(E obj) {

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

if (isEmpty())

return null;

while (current != null &&

((Comparable<E>)obj).compareTo(current.data) != 0) {

previous = current;

current = current.next;

}

if (current == null) // empty

return null;

if (previous == null) // one element

head = head.next;

else if (current == tail) { // last element

previous.next = null;

tail = previous;

}

else

previous.next = current.next;

if(head == null)

tail = null;

currentSize--;

changeCounter++;

return current.data;

}

// Removes and returns last instance of an object, otherwise null

public E removeLastInstance(E obj) {

Node<E> previous = null, current = head, before = null,

target = current;

while (current != null) {

if (((Comparable<E>)obj).compareTo(current.data) == 0) {

before = previous;

target = current;

}

previous = current;

current = current.next;

}

if (target == null) // object not here or is empty

return null;

if (before == null) // if it's in 1st position

head = head.next;

else if (target == tail) { // if it's in last position

before.next = null;

tail = before;

}

else

before.next = target.next;

currentSize--;

changeCounter++;

return target.data;

}

// Returns true if list contains an object, otherwise false

public boolean contains(E obj) {

Node<E> current = head;

while(current != null) { //traverse thru list

if (((Comparable<E>)obj).compareTo(current.data) == 0)

return true;

current = current.next;

}

return false;

}

// Returns the object if found, otherwise null

public E find(E obj) {

if(contains(obj))

return obj;

return null;

}

public int size() {

return currentSize;

}

public void clear() {

currentSize = 0;

}

public boolean isFull() {

return false;

}

public boolean isEmpty() {

return currentSize == 0;

}

public Iterator<E> iterator() {

return new IteratorHelper();

}

class IteratorHelper implements Iterator<E> {

private Node<E> iterPointer;

private long modCounter;

public IteratorHelper() {

iterPointer = head;

modCounter = changeCounter;

}

public boolean hasNext() {

if (changeCounter != modCounter)

throw new ConcurrentModificationException();

return iterPointer != null;

}

public E next() {

if (!hasNext())

throw new NoSuchElementException();

E temp = iterPointer.data;

iterPointer = iterPointer.next;

return temp;

}

public void remove() {

throw new UnsupportedOperationException();

}

}

//////////////////////////

private class Node<E> {

E data;

Node<E> next;

public Node(E d) {

data = d;

}

}

/////////////////////////

}