/\* Arthur Flores

masc0809

\*/

package data\_structures;

import java.util.Iterator;

import java.util.NoSuchElementException;

public class ArrayLinearList<E> implements LinearListADT<E> {

private E[] list;

private int currentSize, maxSize;

public ArrayLinearList(){

currentSize=0;

maxSize=DEFAULT\_MAX\_CAPACITY;

list= (E[]) new Object[maxSize+1];

}

// Adds the Object obj to the end of list.???

public void addLast(E obj){

insert(obj, currentSize+1);

}

private void resizeArray(int type){

if(type== 1)

maxSize <<=1;

if(type == 2)

maxSize >>=1;

E[] tmp=(E[])new Object[maxSize+1];

for(int i=1; i<=currentSize; i++)

tmp[i]=list[i];

list=tmp;

}

// Adds the Object obj to the beginning of list.???

public void addFirst(E obj){

insert(obj, 1);

}

// Inserts the Object obj at the position indicated. If there is an element at

// that location, all elements from that location to the end of the list are

// shifted down to make room for the new insertion. The location is one based.

// If the location > size()+1 then a RuntimeException is thrown. List elements

// must be contiguous.?

public void insert(E obj, int location){

if(location>currentSize+1)

throw new RuntimeException("Attempt to insert is invalid at location"+location);

if(currentSize == maxSize)

resizeArray(1);

for(int i=currentSize; i >= location; i--)

list[i+1] = list[i];

list[location]=obj;

currentSize++;

}

// Removes the object located at the parameter location (one based).

// Throws a RuntimeException if the location does not map to a valid //position within the list.?

public E remove(int location){

if(location>currentSize)

throw new RuntimeException("No such location found");

if(currentSize-1 < maxSize/4)

resizeArray(2);

E object= list[location];

for(int i=location+1;i<=currentSize;i++)

list[i-1]=list[i];

list[currentSize--]=null;

return object;

}

// Removes and returns the parameter object obj from the list if the list //contains it, null otherwise.

// The ordering of the list is preserved. The list may contain duplicate //elements. This method

// removes and returns the first matching element found when traversing the //list from first position.???

public E remove(E obj){

int i= locate(obj);

if(i==-1)

return null;

remove(i);

return obj;

}

// Removes and returns the parameter object obj in first position in list if the list is not empty,

// null if the list is empty. The ordering of the list is preserved.???

public E removeFirst(){

E removed= remove(1);

return removed;

}

// Removes and returns the parameter object obj in last position in list if the list is not empty,

// null if the list is empty. The ordering of the list is preserved.???

public E removeLast(){

E removed= remove(currentSize);

return removed;

}

// Returns the parameter object located at the parameter location position (one based).!!!

// Throws a RuntimeException if the location does not map to a valid position within the list.

public E get(int location){

return list[location];

}

// Returns true if the parameter object obj is in the list, false otherwise.???

public boolean contains(E obj){

int i= locate(obj);

return i!=-1;

}

// Returns the one based location of the parameter object obj if it is in the list, -1 otherwise.

// In the case of duplicates, this method returns the element closest to position #1.???

public int locate(E obj){

for(int i=1;i<=currentSize;i++)

if(((Comparable<E>)obj).compareTo(list[i]) == 0)

return i;

return -1;

}

// The list is returned to an empty state.???

public void clear(){

list= (E[]) new Object[maxSize+1];

currentSize = 0;

}

// Returns true if the list is empty, otherwise false.???

public boolean isEmpty(){

return currentSize==0;

}

// Returns the number of Objects currently in the list.???

public int size(){

return currentSize;

}

// Returns an Iterator of the values in the list, presented in

// the same order as the underlying order of the list. (position #1 first)

public Iterator<E> iterator(){

return new IteratorHelper();

}

class IteratorHelper implements Iterator<E> {

int iterIndex;

public IteratorHelper() {

iterIndex = 1;

}

public boolean hasNext() {

return iterIndex <= currentSize;

}

public E next() {

if(!hasNext()) throw new NoSuchElementException();

return list[iterIndex++];

}

public void remove() {

throw new UnsupportedOperationException();

}

}

}