/\*

Alexander Giang

masc2169

\*/

package data\_structures;

import java.util.Iterator;

import java.util.NoSuchElementException;

public class OrderedArrayList<E> implements OrderedListADT<E> {

private int maxSize, currentSize;

private E[] storage;

//constructor

public OrderedArrayList(int size) {

maxSize = size;

currentSize = 0;

storage = (E[]) new Object[maxSize];

}

//binary search for insert()

public int findInsertionPoint(E key, int lo, int hi) {

if( hi < lo )

return lo;

int mid = (lo+hi)/2;

if(((Comparable<E>)key).compareTo(storage[mid]) < 0)

return findInsertionPoint(key,lo,mid-1);

return findInsertionPoint(key,mid+1,hi);

}

// binary search for find()

public int findBinSearch(E key, int lo, int hi) {

if(hi == lo) {

if(((Comparable<E>)key).compareTo(storage[lo]) == 0)

return lo;

return -1;

}

int mid = (lo+hi)/2;

if(((Comparable<E>)key).compareTo(storage[mid]) <= 0)

return findBinSearch(key,lo,mid);

return findBinSearch(key,mid+1,hi);

}

// Adds the Object obj to the list in the correct position as determined by the Comparable interface.

public void insert(E obj) {

if(isFull())

throw new RuntimeException();

int where = findInsertionPoint(obj,0,currentSize-1);

for(int i = currentSize ; i > where ; i--) //shift

storage[i] = storage[i-1];

storage[where] = obj;

currentSize++;

}

// Removes and returns the object located at the parameter index position (zero based).

// Throws IndexOutOfBoundsException if the index does not map to a valid position within the list.

public E remove(int index) {

E temp = (E) new Object();

temp = storage[index];

for(int i = index ; i < currentSize-1 ; i++) //shift

storage[i] = storage[i+1];

if(index < 0 || index > currentSize)

throw new IndexOutOfBoundsException();

currentSize--;

return temp;

}

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

public E remove(E obj) {

int where = find(obj);

if(!contains(obj))

return null;

return remove(where);

}

// Removes and returns the smallest element in the list and null if the it is empty.

public E removeMin() {

if(isEmpty())

return null;

return remove(0);

}

// Removes and returns the largest element in the list and null if the it is empty.

public E removeMax() {

if(isEmpty())

return null;

return remove(currentSize-1);

}

// Returns the parameter object located at the parameter index position (zero based).

// Throws IndexOutOfBoundsException if the index does not map to a valid position within the underlying array

public E get(int index) {

if(index < 0 || index > currentSize)

throw new IndexOutOfBoundsException();

return storage[index];

}

// Returns the list object that matches the parameter, and null if the list is empty.

// Also returns null if the obj is NOT in the list.

// This method is stable, if obj matches more than one element, the element that

// has been in the list longest is returned.

public E get(E obj) {

if(isEmpty() )

return null;

if(!contains(obj))

return null;

int where = find(obj);

return storage[where];

}

// Returns the index of the first element that matches the parameter obj

// and -1 if the item is not in the list.

public int find(E obj) {

return findBinSearch(obj,0,currentSize-1);

}

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

public boolean contains(E obj) {

if(find(obj) == -1)

return false;

return true;

}

// The list is returned to an empty state.

public void clear() {

currentSize = 0;

}

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

public boolean isEmpty() {

return currentSize == 0;

}

// Returns true if the list is full, otherwise false

public boolean isFull() {

return currentSize == maxSize;

}

// 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 list.

public Iterator<E> iterator() {

return new IteratorHelper();

}

class IteratorHelper implements Iterator<E> {

private int iterIndex;

//constructor

public IteratorHelper() {

iterIndex = 0;

}

public boolean hasNext() {

return iterIndex < currentSize;

}

public E next() {

if(!hasNext())

throw new NoSuchElementException();

return storage[iterIndex++];

}

public void remove() {

throw new UnsupportedOperationException();

}

}

}