Ordered Vector

Writeup by **Vincent Chan**

Student Account: **masc0264**

RedID: **815909699**

OrderedVector.java

```java
 1  /* Vincent Chan
 2   * masc0264
 3   */
 4
 5  package data_structures;
 6
 7  import java.util.Iterator;
 8  import java.util.NoSuchElementException;
 9
10  public class OrderedVector<E> implements OrderedListADT<E> {
11     /*Functions Included
12      * ===PUBLIC===
13      * Constructor                Ln. 34
14      * insert(Object)             Ln. 40
15      * remove(index)              Ln. 56
16      * remove(Object)             Ln. 77
17      * get(index)                 Ln. 85
18      * get(Object)                Ln. 91
19      * contains(Object)           Ln. 95
20      * clear()                    Ln. 100
21      * isEmpty()                  Ln. 107
22      * size()                     Ln. 112
23      * iterator()                 Ln. 117
24      *
25      * ===PRIVATE===
26      * binSearch(Object, low, hi) Ln. 123
27      * find(Object, low, hi)      Ln.138
28     */
29
30     //Variable Declarations
31     private int size, maxSize;
32     private E[] vectorArray = (E[])new Object[DEFAULT_MAX_CAPACITY];
33
34     //Constructor
35     public OrderedVector() {
36       size = 0;
37       maxSize = DEFAULT_MAX_CAPACITY;
38     } //End constructor
39
40     //This will insert the object and organize the array.
41     public void insert(E obj) {
42       //If the array grows too large, this will grow the array.
43       if(size+1>maxSize) {
44         maxSize *= 2;
45         E[] temp = (E[])new Object[maxSize];
46         for(int i=0; i<size; i++) temp[i] = vectorArray[i];
47         vectorArray = temp;
48       }
49
50       int insertLoc = find(obj, 0, size-1);
51       for(int i=size; i>insertLoc; i--) vectorArray[i] = vectorArray[i-1];
52       vectorArray[insertLoc] = obj;
53       size++;
54     } //End insert()
55
56     //Pops the element from the array and adjusts the array accordingly
```

```java
  public E remove(int index) {
    //This will throw an exception if an out of bounds operation is attempted
    if(index<0 || index>=size) throw new IndexOutOfBoundsException();

    //This will put the object in a temp array and update the size.
    E tempObj = vectorArray[index];
    size--;

    //If the array is less than 25% populated, shrink array.
    if(maxSize/4 > size) {
      maxSize /= 2;
      E[] temp = (E[])new Object[maxSize];
      for(int i=0; i<size; i++) temp[i] = vectorArray[i];
      vectorArray = temp;
    }

    for(; index<size; index++) vectorArray[index] = vectorArray[index+1];
    return tempObj;
  } //End remove()

  //Removes and returns the object and null on failure.
  public E remove(E obj) {
    try {return remove(binSearch(obj, 0, size-1));}
    catch(Exception e){return null;}
  } //End remove()

  //Returns the parameter object located at the parameter
  //Throws OutOfBoundsException if the index provided is out of bounds.
  public E get(int index) {
    if(index<0 || index>=size) throw new IndexOutOfBoundsException();
    return vectorArray[index];
  } //End get()

  //Returns the object if it exists inside the array, null if not.
  public E get(E obj) {
    return contains(obj)?obj : null;
  } //End get()

  //Returns true if the parameter object is in the list, false otherwise.
  public boolean contains(E obj) {
    return binSearch(obj, 0, size-1) != -1;
  } //End contains()

  //The list is returned to an empty state.
  public void clear() {
    vectorArray = (E[])new Object[DEFAULT_MAX_CAPACITY];
    size = 0;
    maxSize = DEFAULT_MAX_CAPACITY;
  } //End clear()

  //Returns true if the array is empty
  public boolean isEmpty() {
    return size==0;
  } //End isEmpty()

  //Returns the number of objects currently in the array.
  public int size() {
              return size;
  } //End size()

  //Returns an iterator of the values in the list,
  //presented in the same order as the list
  public Iterator<E> iterator() {
    return new IteratorHelper();
  }

  //This function will return the index of where the element is located
  //returns -1 if not found
  private int binSearch(E obj,int low, int hi) {
    //Termination condition: checked the array and could not find it
    if(hi<low) return -1;

    //Compare the middle of the array to the sought object
    int mid = (low+hi)/2;
    int comp = ((Comparable<E>)obj).compareTo(vectorArray[mid]);
    if(comp==0) return mid;

    //If not found, recursively call the function with a refined search area.
    return (comp<0)?binSearch(obj, low, mid-1) : binSearch(obj, mid+1, hi);
  } //End binSearch()

  //This will return the index of where an element should be inserted
  private int find(E obj, int low, int hi) {
    //Termination condition: Found the insertion point.
    if(hi<low) return low;

    //Compare the middle of the array to the sought object
    int mid = (low+hi)/2;
    int comp = ((Comparable<E>)obj).compareTo(vectorArray[mid]);

    //If not found, recursively call the function with a refined search area.
    return (comp<0)?find(obj, low, mid-1) : find(obj, mid+1, hi);
  } //End find()

  //This class will help with iteration, and provide the iterator function.
  class IteratorHelper implements Iterator<E> {
    int iterIndex;

    public IteratorHelper() {
      iterIndex = 0;
    }

    public boolean hasNext() {
      return iterIndex < size;
    }

    public E next() {
      if(!hasNext()) throw new NoSuchElementException();
      return vectorArray[iterIndex++];
    }

    public void remove() {
      throw new UnsupportedOperationException();
    }
  }
} //End OrderVector()
```