LAB #8 - ARRAY BASED LISTS

The next exercise is based on this implemetation for an `UnorderedArrayList` of integers:

```
//Interface: ArrayListADT
//works for int
public interface ArrayListADT {
    public boolean isEmpty(); //Method to determine whether the list is empty.
    public boolean isFull();  //Method to determine whether the list is full.
    public int listSize();    //Method to return the number of elements in the list.
    public int maxListSize(); //Method to return the maximum size of the list.
    public void print();      //Method to output the elements of the list.
    public boolean isItemAtEqual(int location, int item); //Method to determine
whether item is the same as the item in the list at location.
    public void insertAt(int location, int insertItem);   //Method to insert
insertItem in the list at the position
    public void insertEnd(int insertItem); //Method to insert insertItem at the end
of the list.
    public void removeAt(int location);  //Method to remove the item from the list
at location.
    public int retrieveAt(int location);   //Method to retrieve the element from the
list at location.
    public void replaceAt(int location, int repItem); //Method to replace the
element in the list at location with repItem.
    public void clearList();  //Method to remove all the elements from the list.
    public int search(int searchItem);    //Method to determine whether searchItem
is in the list.
    public void remove(int removeItem);   //Method to remove an item from the list.
}

//Class: ArrayListClass implements
//Interface: ArrayListADT
public abstract class ArrayListClass implements ArrayListADT {
    protected int length;    //to store the length of the list
    protected int maxSize;   //to store the maximum size of the list
    protected int[] list;    //array to hold the list elements

    //Default constructor
    public ArrayListClass() {
        maxSize = 100;
        length = 0;
        list = new int[maxSize];
    }

    //Alternate Constructor
    public ArrayListClass(int size) {
        if(size <= 0) {
            System.err.println("The array size must be positive. Creating an array
of size 100.");
            maxSize = 100;
        }
        else
```

```
            maxSize = size;
        length = 0;
        list = new int[maxSize];
    }

    public boolean isEmpty() {
        return (length == 0);
    }

    public boolean isFull() {
        return (length == maxSize);
    }

    public int listSize() {
        return length;
    }

    public int maxListSize() {
        return maxSize;
    }

    public void print() {
        for (int i = 0; i < length; i++)
            System.out.print(list[i] + "  ");
        System.out.println();
    }

    public boolean isItemAtEqual(int location, int item) {
        if (location < 0 || location >= length) {
            System.err.println("The location of the item to be compared is out of
range.");
            return false;
        }
        return list[location]== item;
    }

    public void clearList() {
        for (int i = 0; i < length; i++)
            list[i] = 0;
        length = 0;
        System.gc();   //invoke the Java garbage collector
    }

    public void removeAt(int location) {
        if (location < 0 || location >= length)
            System.err.println("The location of the item to be removed is out of
range.");
        else {
            for(int i = location; i < length - 1; i++)
                 list[i] = list[i + 1];
            length--;
        }
    }
```

```
    public int retrieveAt(int location) {
        if (location < 0 || location >= length) {
            System.err.println("The location of the item to be retrieved is out of
range.");
            return 0;
        }
        else
            return list[location];
    }

    public abstract void insertAt(int location, int insertItem);
    public abstract void insertEnd(int insertItem);
    public abstract void replaceAt(int location, int repItem);
    public abstract int search(int searchItem);
    public abstract void remove(int removeItem);
}

//Class: UnorderedArrayList extends
//Super class: ArrayListClass
public class UnorderedArrayList extends ArrayListClass {

    public UnorderedArrayList() {
        super();
    }

    public UnorderedArrayList(int size) {
        super(size);
    }

    //Bubble Sort
    public void bubbleSort() {
     for (int pass = 0; pass < length - 1; pass++) {
         for (int i = 0; i < length - 1; i++) {
             if (list[i] > list[i + 1]) {
                 int temp = list[i];
                 list[i] = list[i + 1];
                 list[i + 1] = temp;
             }
         }
     }
    }

    //implementation for abstract methods defined in ArrayListClass
    //unordered list --> linear search
    public int search(int searchItem) {
        for(int i = 0; i < length; i++)
            if(list[i] == searchItem)
                return i;
        return -1;
    }

    public void insertAt(int location, int insertItem) {
```

```
        if (location < 0 || location >= maxSize)
            System.err.println("The position of the item to be inserted is out of
range.");
        else if (length >= maxSize)
            System.err.println("Cannot insert in a full list.");
        else {
            for (int i = length; i > location; i--)
                list[i] = list[i - 1];  //shift right
            list[location] = insertItem;
            length++;
        }
    }

    public void insertEnd(int insertItem) {
        if (length >= maxSize)
            System.err.println("Cannot insert in a full list.");
        else {
            list[length] = insertItem;
            length++;
        }
    }

    public void replaceAt(int location, int repItem)  {
        if (location < 0 || location >= length)
            System.err.println("The location of the item to be replaced is out of
range.");
        else
            list[location] = repItem;
    }

    public void remove(int removeItem) {
        int i;
        if (length == 0)
            System.err.println("Cannot delete from an empty list.");
        else {
            i = search(removeItem);
            if (i != -1)
                removeAt(i);
            else
                System.out.println("Cannot delete! The item to be deleted is not in
the list.");
        }
    }
}
```

1.1 Add to class `UnorderedArrayList` a new method called `scaleByK` that should replace every integer of value `k` with `k` copies of itself. For example, if the list is: [2, 4, -2, 5, 3, 0, 7] before the method is invoked, it should be [2, 2, 4, 4, 4, 4, 5, 5, 5, 5, 5, 3, 3, 3, 7, 7, 7, 7, 7, 7, 7] after the method executes. Note that the method should remove from the list all 0s and negative values. Test the method using this client:

```
//Testing the method scaleByK added to the user created UnorderedArrayList class
public class Lab8_1 {
    public static final int SIZE = 100;
    public static void main(String[] args)  {
        UnorderedArrayList list = new UnorderedArrayList(SIZE);
        list.insertEnd(2);
        list.insertEnd(4);
        list.insertEnd(-2);
        list.insertEnd(5);
        list.insertEnd(3);
        list.insertEnd(0);
        list.insertEnd(7);
        System.out.println("The original list is: ");
        list.print();
        System.out.println("The list after method call is: ");
        list.scaleByK();
        list.print();
    }
}
```

1.2. Same problem. This time use the `ArrayList` class in Java. Write `scaleByK` as a client method and use the `print` method provided.

```
//Testing the method scaleByK using the Java ArrayList class
import java.util.ArrayList;
public class Lab8_2 {
    public static void main(String[] args)  {
        ArrayList <Integer> list = new ArrayList <Integer>();
        list.add(2);
        list.add(4);
        list.add(-2);
        list.add(5);
        list.add(3);
        list.add(0);
        list.add(7);
        System.out.println("The original list is: ");
        print(list);
        System.out.println("The list after method call is: ");
        scaleByK(list);
        print(list);
    }

    public static void scaleByK(ArrayList<Integer> list) {
        ...

    }

    public static void print(ArrayList <Integer> someList) {
        for(Integer i:someList)
            System.out.print(i + "  ");
```

```
            System.out.println();
        }
    }
```

The next exercise is based on this implemetation for an `OrderedArrayList` of integers:

```
//Interface: ArrayListADT
public interface ArrayListADT {
    //same as above, you already have it!
}

//Class: ArrayListClass implements
//Interface: ArrayListADT
public abstract class ArrayListClass implements ArrayListADT {
   //same as above, you already have it!
}

//Class: OrderedArrayList extends
//Super class: ArrayListClass
public class OrderedArrayList extends ArrayListClass{

    public OrderedArrayList() {
        super();
    }

    public OrderedArrayList(int size) {
        super(size);
    }


    //implementation for abstract methods defined in ArrayListClass

    //ordered list --> binary search
    public int search(int item) {
        int first = 0;
        int last = length - 1;
        int middle = -1;

        while (first <= last)  {
            middle = (first + last) / 2;
            if (list[middle] == item)
                return middle;
            else
                if (list[middle] > item)
                    last = middle - 1;
                else
                    first = middle + 1;
        }
        return -1;
    }

    public void insert(int item) {
        int loc;
```

```
        boolean found = false;
        if (length == 0)            //list is empty
            list[length++] = item;  //insert item and increment length
        else if (length == maxSize) //list is full
            System.err.println("Cannot insert in a full list.");
        else {
            for (loc = 0; loc < length; loc++) {
                if (list[loc] >= item) {
                    found = true;
                    break;
                }
            }
            //starting at the end, shift right
            for (int i = length; i > loc; i--)
                list[i] = list[i - 1];
            list[loc] = item; //insert in place
            length++;
        }
    }

    /* Another version for insert:
    public void insert(int item) {
        int loc;
        boolean found = false;
        if (length == 0)            //list is empty
            list[length++] = item;  //insert item and increment length
        else if (length == maxSize) //list is full
            System.err.println("Cannot insert in a full list.");
        else {
            int i = length - 1;
            while (i >= 0 && list[i] > item) {
                list[i + 1] = list[i];
                i--;
            }
            list[i + 1] = item;  // Insert item
            length++;
       }
    } */

    public void insertAt(int location, int item)  {
        if (location < 0 || location >= maxSize)
            System.err.println("The position of the item to be inserted is out of
range.");
        else if (length == maxSize)  //the list is full
            System.err.println("Cannot insert in a full list.");
        else {
            System.out.println("Cannot do it, this is a sorted list. Doing insert in
place (call to insert).");
            insert(item);
        }
    }
```

```
    public void insertEnd(int item) {
        if (length == maxSize)  //the list is full
            System.err.println("Cannot insert in a full list.");
        else {
            System.out.println("Cannot do it, this is a sorted list. Doing insert in
place (call to insert).");
            insert(item);
        }
    }

    public void replaceAt(int location, int item) {
        //the list is sorted!
        //is actually removing the element at location and inserting item in place
        if (location < 0 || location >= length)
             System.err.println("The position of the item to be replaced is out of
range.");
        else {
            removeAt(location);//method in ArrayListClass
            insert(item);
        }
    }

     public void remove(int item) {
        int loc;
        if (length == 0)
            System.err.println("Cannot delete from an empty list.");
        else  {
            loc = search(item);
            if (loc != -1)
                removeAt(loc);//method in ArrayListClass
            else
                System.out.println("The item to be deleted is not in the list.");
        }
    }

    /*Another version for remove:
    public void remove(T item) {
        int loc;
        if (length == 0)
            System.err.println("Cannot delete from an empty list.");
        else  {
            loc = search(item);
            if (loc != -1) {
                for(int i = loc; i < length - 1; i++)
                    list[i] = list[i + 1]; //shift left
                length--;
            }
            else
                System.out.println("The item to be deleted is not in the list.");
        }
    } */
```

```
}
```

2.1. Add to class `OrderedArrayList` a new method called `removeDuplicates` that should eliminate any duplicates from a sorted list. For example, if the list is: [2, 2, 2, 2, 5, 5, 8, 9, 9, 9] before the method is invoked, it should be [2, 5, 8, 9] after the method executes. Test the method using this client:

```
//Testing the method removeDuplicates added to the user created
OrderedArrayList class
public class Lab8_3 {
    public static void main(String[] args)  {
        OrderedArrayList list = new OrderedArrayList();
        list.insert(8);
        list.insert(2);
        list.insert(2);
        list.insert(9);
        list.insert(5);
        list.insert(9);
        list.insert(2);
        list.insert(9);
        list.insert(2);
        list.insert(5);
        System.out.println("The original list is: ");
        list.print();
        System.out.println("The list after method call is: ");
        list.removeDuplicates();
        list.print();
    }
}
```

2.2. Same problem. This time use the `ArrayList` class in Java. Write `removeDuplicates` as a client method and use the print method provided.

```
//Testing the method removeDuplicates using the Java ArrayList class
import java.util.ArrayList;
public class Lab8_4 {
    public static void main(String[] args)  {
        ArrayList <Integer> list = new ArrayList <Integer>();
        list.add(2);
        list.add(2);
        list.add(2);
        list.add(5);
        list.add(5);
        list.add(8);
        list.add(9);
        list.add(9);
        System.out.println("The original list is: ");
        print(list);
        System.out.println("The list after method call is: ");
        removeDuplicates(list);
```

```
        print(list);
    }

    public static void removeDuplicates(ArrayList<Integer> list) {
        ...

    }

    public static void print(ArrayList <Integer> someList) {
        for(Integer i:someList)
            System.out.print(i + "  ");
        System.out.println();
    }
}
```
