

CS 1723, Simple List of Ints compared to Generic List Using Comparable

```

// List: a simple list of ints
public interface List {
    // isEmpty: is the list empty? (initially true)
    public boolean isEmpty();

    // add: add x to the list. No check for dupls
    public boolean add(int x);

    // contains: is x in the list?
    public boolean contains(int x);

    // size: return number of elements in the list
    public int size();

    // remove: remove the parameter
    public boolean remove(int x);

    // removeMin: return the minimum and remove it
    public int removeMin();

    // removeMax: return the minimum and remove it
    public int removeMax();
}

// IntList: a simple list of ints
public class IntList implements List {
    private int[] listArray; // the actual list
    private int currentSize; // current # of elts
    private final int INITIAL_SIZE = 6; // max size

    // IntList: constructor
    public IntList () {
        currentSize = 0; // not needed (already done)
        listArray = new int[INITIAL_SIZE]; // allocate
    }

    // isEmpty: is the list empty? (initially true)
    public boolean isEmpty() {
        return size() == 0;
    }

    // add: add x to the list. No check for dups
    public boolean add(int x) {
        if (currentSize >= listArray.length)
            doubleArray();
        listArray[currentSize++] = x;
        return true;
    }

    // contains: is x in the list?
    public boolean contains(int x) {
        for (int i = 0; i < currentSize; i++)
            if (listArray[i] == x) return true;
        return false;
    }

    // size: return number of elements in the list
    public int size() {
        return currentSize;
    }

    // remove: remove the parameter
    public boolean remove(int x) {
        for (int i = 0; i < currentSize; i++)
            if (listArray[i] == x) {
                for (int j = i; j < currentSize - 1; j++)
                    listArray[j] = listArray[j+1];
                currentSize--;
                return true;
            }
        return false;
    }

    // removeMin: return the minimum and remove it
    public int removeMin() {
        if (size() == 0) return 0;
        int min = listArray[0];
        for (int i = 1; i < currentSize; i++)
            if (listArray[i] < min)
                min = listArray[i];
        remove(min);
        return min;
    }

    // removeMax: return the maximum and remove it
    public int removeMax() {
        if (size() == 0) return 0;
        int max = listArray[0];
        for (int i = 1; i < currentSize; i++)
            if (listArray[i] > max)
                max = listArray[i];
        remove(max);
        return max;
    }
}

```

```

// List: a simple list of comparable items
public interface List {
    // isEmpty: is the list empty? (initially true)
    public boolean isEmpty();

    // add: add x to the list. No check for duplicates
    public boolean add(Comparable x);

    // contains: is x in the list?
    public boolean contains(Comparable x);

    // size: return number of elements in the list
    public int size();

    // remove: remove the parameter
    public boolean remove(Comparable x);

    // removeMin: return the minimum and remove it
    public Comparable removeMin();

    // removeMax: return the minimum and remove it
    public Comparable removeMax();
}

// CompList: a simple list of comparables
public class CompList implements List {
    private Comparable[] listArray; // the actual list
    private int currentSize; // current # of elements
    private final int INITIAL_SIZE = 6; // max size

    // CompList: constructor
    public CompList () {
        currentSize = 0; // not needed (already done)
        listArray = new Comparable[INITIAL_SIZE];
    }

    // isEmpty: is the list empty? (initially true)
    public boolean isEmpty() {
        return size() == 0;
    }

    // add: add x to the list. No check for dups
    public boolean add(Comparable x) {
        if (currentSize >= listArray.length)
            doubleArray();
        listArray[currentSize++] = x;
        return true;
    }

    // contains: is x in the list?
    public boolean contains(Comparable x) {
        for (int i = 0; i < currentSize; i++)
            if (listArray[i].compareTo(x) == 0) return true;
        return false;
    }

    // size: return number of elements in the list
    public int size() {
        return currentSize;
    }

    // remove: remove the parameter
    public boolean remove(Comparable x) {
        for (int i = 0; i < currentSize; i++)
            if (listArray[i].compareTo(x) == 0) {
                for (int j = i; j < currentSize - 1; j++)
                    listArray[j] = listArray[j+1];
                currentSize--;
                return true;
            }
        return false;
    }

    // removeMin: return the minimum and remove it
    public Comparable removeMin() {
        if (size() == 0) return null;
        Comparable min = listArray[0];
        for (int i = 1; i < currentSize; i++)
            if (listArray[i].compareTo(min) < 0)
                min = listArray[i];
        remove(min);
        return min;
    }

    // removeMax: return the maximum and remove it
    public Comparable removeMax() {
        if (size() == 0) return null;
        Comparable max = listArray[0];
        for (int i = 1; i < currentSize; i++)
            if (listArray[i].compareTo(max) > 0)
                max = listArray[i];
        remove(max);
        return max;
    }
}

```