Name: __________________________    Student ID: __________________________

For this exam you may use 1 two-sided 8.5”x11” piece of paper (“cheat sheet”) containing whatever you like. All other books, notes, materials -- especially computers and cell phones -- are prohibited.

Score:

Problem 1: _____/40

Problem 2: _____/20

Problem 3: _____/40

Total: _____/100
Problem 1: ArrayLists

Consider the ArrayList class written below. It offers the familiar add(o), get(index), and size() methods. Implement a new method -- removeAll(o) -- that removes every instance of the specified object o from the ArrayList. Make sure that the other methods will continue to function correctly given your implementation of removeAll(o). You may assume that the user will never add null to the list.

class ArrayList {
    private Object[] _underlyingStorage;
    private int _numElements;

    public ArrayList () {
        _numElements = 0;
        _underlyingStorage = new Object[64];
    }

    public int size () {
        return _numElements;
    }

    // Adds o to the end of the list. o may NOT be null.
    public void add (Object o) {
        if (_numElements == _underlyingStorage.length) {
            // Assume the “...” below “enlarges” the
            // _underlyingStorage array as we’ve discussed in class.
            ...
        }
        _underlyingStorage[_numElements] = o;
        _numElements++;
    }

    // Returns the object stored at the specified index within the
    // ArrayList. You can assume that index is valid, i.e.,
    // 0 <= index < size().
    public Object get (int index) {
        return _underlyingStorage[index];
    }

    // Removes every element of the ArrayList that equals,
    // in the equals(o) sense, the specified object. If o is
    // not contained in the list, then this method does nothing
    // and does not throw an exception.
    public void removeAll (Object o) {
        // Write your solution on the next page
    }
}
Write your implementation of the `removeAll(o)` method below:
Problem 2: Object-orientation in Java

The purpose of this problem is to make sure you understand the relationship between classes, interfaces, sub-interfaces, and implementations.

Consider the Java interfaces specified below. Write a (non-abstract) class $M$ that implements interface $C$. **Your class $M$ doesn’t have to do anything useful.** However, there are two requirements: (a) **your code must compile without errors**; and (b) **none of your methods may return null**, i.e., a method with return-type $B$ must return a valid reference to an object of type $B$. If you wish, you may define additional classes -- either inner-classes or “regular” classes -- to complete this task.

```java
interface A {
    void m ();
}

interface B extends A {
    void gimme (A a);
}

interface C extends A {
    B b ();
}

class M implements C {
    // Write your solution below. You may also create additional
    // classes if they help. Make sure all methods are public!
}
```
Problem 3: SinglyLinkedLists

Consider the partially implemented SinglyLinkedList class below, which uses a non-static inner-class Node and “dummy” head and tail nodes. Implement two methods: addToBack(o), which adds the specified object to the back (tail) of the list, and moveToFront(o), which finds the specified object o within the list (if it exists) and moves it to the front (head) of the list. You may not change the Node class.

class SinglyLinkedList {
    private static class Node {
        Node _next;
        Object _data;
    }
    private Node _head, _tail;  // dummy nodes
    private int _size;

    public SinglyLinkedList () {
        _head = new Node();
        _tail = new Node();
        _head._next = _tail;
        _size = 0;
    }

    // Returns the object stored at the specified index. Assume
    // index is valid, i.e., 0 <= index < _size.
    public Object get (int index) {
        Node cursor = _head._next;
        for (int i = 0; i < index; i++) {
            cursor = cursor._next;
        }
        return cursor._data;
    }

    // Adds the specified object to the back (tail) of the list.
    public void addToBack (Object o) {
        // Write your solution on the next page
    }

    // Searches the list for the specified object o. If found, this
    // method moves o to the front (head) of the list. If not found, it
    // does nothing. Example:
    //   list.addToBack("okra");
    //   list.addToBack("marzipan");
    //   list.addToBack("turnip");
    //   list.moveToFront("marzipan");
    //   list.get(0);  // returns “marzipan”
    //   list.get(1);  // returns “okra”
    //   list.get(2);  // returns “turnip”
    public void moveToFront (Object o) {
        // Write your solution on the next page
    }
}

CSE 12 Summer Session 1 Midterm Examination.......................17 July 2012
Write your implementation of the `addToBack(o)` method below:

Write your implementation of the `moveToFront(o)` method below: