Midterm

October 11, 2001. 5:00–6:15 PM

There are six problems each worth five points for a total of 30 points. Show all your work, partial credit will be awarded. When there is not enough room on the test page itself, write in the provided blue books and write and sign your name on each one. No notes, no collaboration.

Name: ____

Problem	Credit
1	
2	
3	
4	
5	
6	
Total	

1. Fill in the method bodies of addToCounter and getCounter to finish the following program.

```
class ProblemOne
{
   public static void main(String [] args)
   {
       MyObjectOne moo = new MyObjectOne() ;
       moo.addToCounter(5) ;
       moo.addToCounter(2) ;
       System.out.println(moo.getCounter()) ;
   }
}

class MyObjectOne
{
    int counter = 0 ;
    void addToCounter( int numberToAdd )
    {
        // increment counter by numberToAdd

    }

    int getCounter()
    {
        // return the value of counter

    }
}
```

2. Suppose we have a LinkedList ADT which is required to efficiently return the current length of the list, where length of a linked list is the number of nodes on the list. It is too time-consuming to count afresh the number of nodes each time getLength is invoked, so you get the great idea of keeping a "hidden" variable which is always updated with the current linked list length. Complete the following two methods inside the class ProblemTwo to implement this idea.

```
class ProblemTwo
{
    private int count = 0 ;
    private Node root = null ;

    void insertAtHead(String s)
    {
        Node n = new Node() ;
        n.content = s ;
        n.next = root ;
        root = n ;
        // add code to keep count variable current

     }

     int getLength()
     {
        // return the length of the linked list

     }
}
```

3. In this version of a linked list, rather than insert the same string twice, we first try to find the string on the linked list, and if it is on the list, instead we increment its count. Finish the code below.

```
class ProblemThree {

    private int count = 0 ;
    private NodeThree root = null ;

    void insertAtHead(String s) {
        NodeThree n = findOnList(s) ;
        if ( if n!=null ) {
            // ASSERT n.contents.equals(s)==true
            // don't re-insert, adjust count

        }
        else {
           // insertAtHead, as Problem Threee
           // code omitted, assume it's here
        }
    }

    NodeThree findOnList(String s) {
        // returns n such that n.content.equals(s)==true
        // or null, if no such n exists.

        // code omitted, assume it's here and it works!
    }
}

class NodeThree {

   NodeThree next = null ;
   String content = null ;
   int count = 1 ;
}
```

4. So then you have another great idea. To do a delete, rather than removing the element from the list, because this is a pain, you simply decrement its count. Assume class ProblemThree is reproduced below, you just have to fill in the deleteFromList method.

```
class ProblemFour
{
   // instance variables and methods as in class ProblemThree

   void deleteFromList(String s)
   {
      // do the find, if found,  check count>0, and if so
      // decrement count







   }
}
```

5. Now you are troubled. You imagine your list full of elements with count zero. So you decide to write a method, cleanUpList, which runs the list and deletes nodes with count zero. Write it.

```
class ProblemFive
{
   //  instance variables and methods as in class ProblemFour

   void cleanUpList() {









   }
}
```

6. Then you begin to wonder about who will call cleanUpList? So you decide that the linked list object should take care of cleaning up after itself. Also, since the user of the linked list object needn't know about count zero nodes, they should be "hidden" from the public version of the find method. That is, the post-condition on the return value n of the public find method is:

   ```
   ASSERT: (n==null) || (n.count>0)
   ```

   Discuss how you decided to do all this and produce Java code for the affected methods.

   Justify why this "lazy deletion" (marking a record as deleted and really deleting it at some later time, when it is more convenient) is just as efficient as non-lazy (immediate) deletion.

6. (extra workspace)