# MIDTERM EXAMINATION KEY
# CSC 212 ♦ FALL 2013

*You may use one 8.5"x11" sheet of notes on this exam. You may not consult other sources of information. You will have the entire period (80 minutes) to complete your work. All work should be written in the exam booklet. Partial credit will be granted where appropriate if intermediate steps are shown.*

## Vocabulary

Consider the short program below to answer the questions that follow. (The line numbers are for reference but are not part of the program.)

```
1      public class SphereVolumes {
2          public static double volume(double r) {
3              double result = (4.0/3.0)*Math.PI*r*r*r;
4              return result;
5          }
6          public static void main(String[] args) {
7              for (int i = 0; i < args.length; i++) {
8                  double r, v;
9                  r = Double.valueOf(args[i]);
10                 v = volume(r);
11                 System.out.println("Radius="+r+", volume="+v+".");
12             }
13         }
14     }
```

a.) List all tokens appearing in the program above that are names of classes.
*SphereVolumes, Math, String, Double, System*

b.) List all tokens appearing in the program above that are fields of a class.
*PI, (length)*

c.) List all the local variables used in the main method.
*i, r, v, (args)*

d.) List the numbers of all lines containing method calls.
*9, 10, 11*

e.) List the numbers of all lines containing a variable declaration (excluding method parameters).
3, *7, 8*

f.) List the numbers of all lines that perform memory allocation on the heap.
*None*

g.) List the numbers of all lines containing variable assignments.
*3, 7, 9, 10*

h.) If the volume method had not been declared static, how could it be called from main? (Write a code fragment to replace line 10.)
*SphereVolumes sv = new SphereVolumes(); v = sv.volume(r); // create an instance to call on*

## Graphical User Interfaces

Identify a class or interface from either the **java.awt**, **java.awt.event** or **javax.swing** packages that matches the descriptions below.

a.) Represents a top-level window with a title and border
*JFrame*

b.) The parent class for all objects that can appear in a content pane
*JComponent*

c.) Represents a hue in terms of its red, green, and blue components
*Color*

d.) Holds the event handler for a clicked button
*ActionListener*

e.) Defines empty handler methods for mouse clicks, drags, wheel motions, etc.
*MouseAdapter*

f.) Can arrange window components in a 3x3 grid
*GridLayout*

g.) Allows you to fit one component layout within another in a GUI
*JPanel*

h.) Provides methods for drawing lines, circles, rectangles, etc. within a given area of the screen
*Graphics*

## Stacks

Consider the following pseudocode:

```
given string T, stack S initially empty
err = 0;
n ← length(T)
loop for i from 0 to n-1
   if i is less than (n-1)/2
      S.push(T[i])
   elseif i is greater than (n-1)/2
      if S.pop() not equal to T[i]
         err = err+1;
      endif
   endif
endloop
```

a.) If **T** = "REVERSE" what would be the value of **err** at the end?
*3*

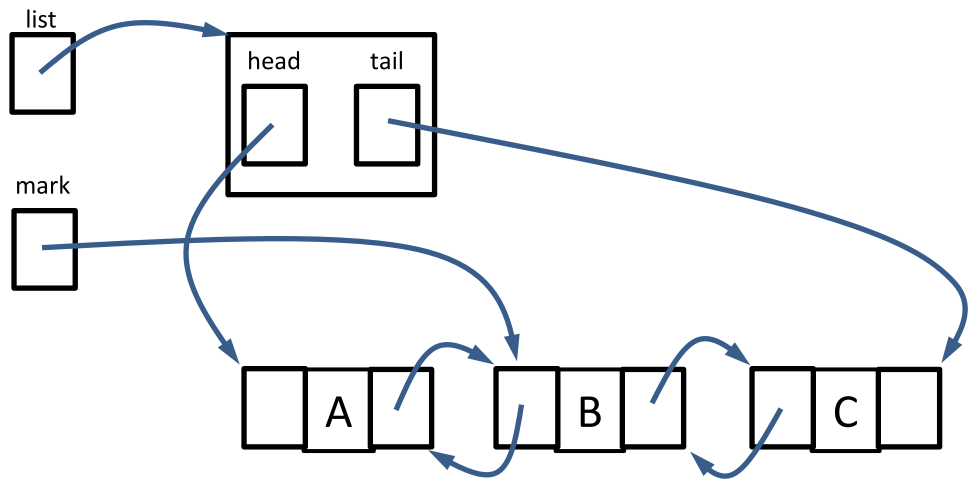
b.) If **T** = "RACECAR" what would be the value of **err** at the end?
*0*

c.) Draw the contents of the stack after the completion of each loop iteration for **T** = "STOPS".
Contents of stack (top is rightmost): S – ST – ST – S - <empty>

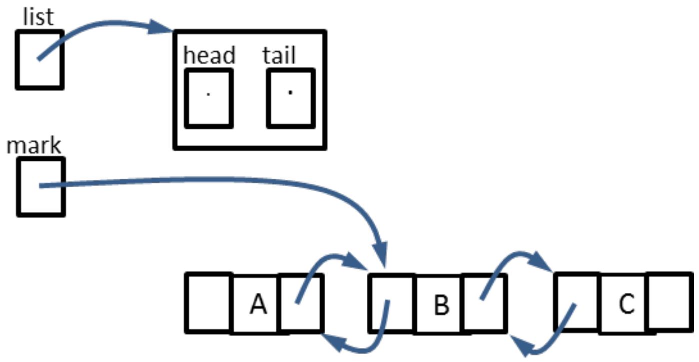
d.) If **T** = "POPS" and **S** is not empty but holds 'S' initially, what would be the value of **err** at the end?
*It turns out that this question is ambiguous. Since the algorithm was given in pseudocode, I had intended the (n-1)/2 to be interpreted mathematically as 1.5. However, if written in Java using ints, the result would be rounded down to 1, meaning that the O would be skipped. This dramatically*

*changes the answer – in the former interpretation, err would be 2 (the initial contents of the stack don't affect the algorithm). In the latter, err would be 0. I accepted either answer.*

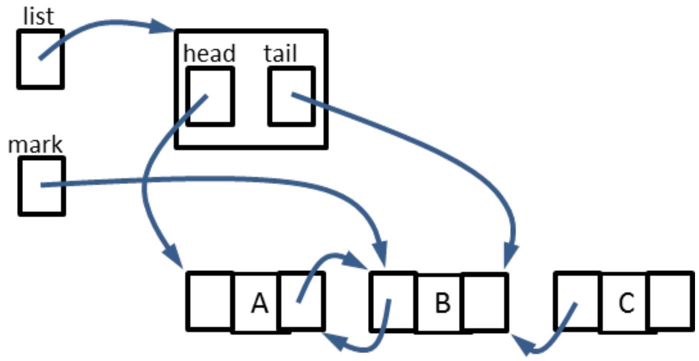
Linked Lists
Consider the diagram below, which visualizes the memory structure of a doubly linked list of the sort developed in class. The fields shown in each node are from left to right **previous**, **data**, and **next**. For each part below, draw the state of the memory structure as it would look after the commands given. If an exception would occur then you should draw no picture but identify the line that would cause the exception. Finally, for all the items where you have drawn a picture, state whether or not the structure referred to by **list** represents an internally consistent structure for a linked list.



a.) list.tail = list.head = null;
*Consistent (empty list)*

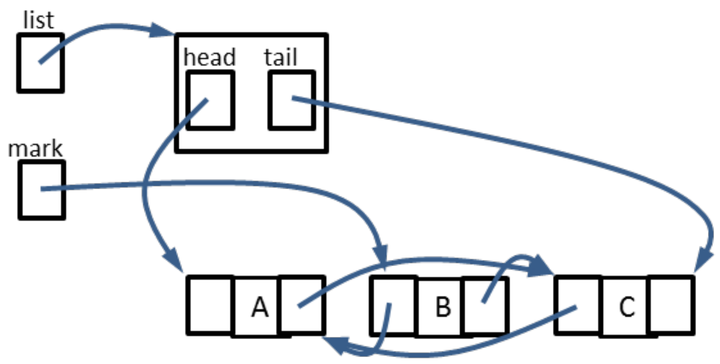


b.) list.tail = mark; list.head.next.next = null;
*Consistent (last element has been removed)*



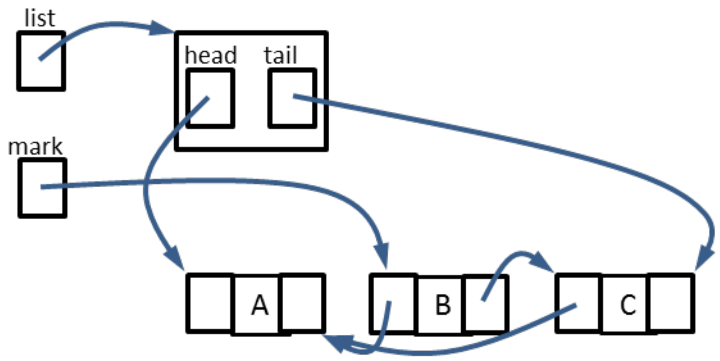
c.) mark.next.previous = mark.previous; mark.previous.next = mark.next; *(as intended)*

*Consistent (middle element has been removed)*



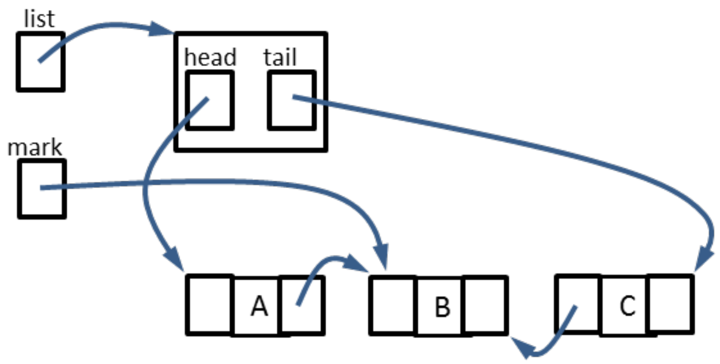
c.) mark.next.previous = mark.prev; mark.previous.next = mark.previous; *(as written)*

*Inconsistent (self-loop)*



d.) mark.next = mark.previous = null;

*Inconsistent (middle element does not link to neighbors any more)*



e.) head = head.next; head.previous = null; head.previous.next = null;

*NullPointerException on third command*

Complexity Analysis

The method below implements bubble sort on an array of integers. How many times would the comparison inside the inner loop be executed? Give an exact formula in terms of **n**, the array length, and also relate this to a standard complexity class (big-O bound).

*Outer loop n-1 times, inner loop n/2 times on average; overall* $(n^2-n)/2$

*This is* $O(n^2)$

```
public static void sort(int[] arr) {
    for (int i = 1; i < arr.length; i++) {
        for (int j = 0; j < arr.length-i; j++) {
            if (arr[j] > arr[j+1]) {
                int tmp = arr[j];
                arr[j] = arr[j+1];
                arr[j+1] = tmp;
            }
        }
    }
}
```

Consider the version below, which follows the same implementation using a **LinkedList<Integer>**. What, if anything, will change about the complexity analysis as compared to the case above?

*Yes, something changes. The **get** and **set** methods using indices require O(n) time to compute (specifically, n/4 time on average). Since these are already being executed* $O(n^2)$ *times, the complexity of the entire algorithm becomes* $O(n^3)$*.*

```
public static void sort(LinkedList<Integer> list) {
    for (int i = 1; i < list.size(); i++) {
        for (int j = 0; j < list.size()-i; j++) {
            if (list.get(j) > list.get(j+1)) {
                int tmp = list.get(j);
                list.set(j,list.get(j+1));
                list.set(j+1,tmp);
            }
        }
    }
}
```

Class Design

Write a few short paragraphs describing the role of constructors, accessors, and manipulators. Include in your discussion their relationship to the fields of a class, and the considerations that apply when deciding which constructors, accessors, and manipulators should be included in the class definition. Refer to specific examples from class where possible.

*Constructors, accessors, and manipulators are intimately connected to the fields of a class. In fact, they owe their existence and makeup to the fields that the interact with. Under normal circumstances, each constructor will interact with every field of the class, since a constructor's job is to initialize all the fields. In most cases, there will also be one accessor per field, responsible for reporting the field's value, and one manipulator per field, responsible for making changes to the field. So a stereotypical class will include at least one constructor and possibly more, plas accessors and manipulators for every field.*

*Sometimes special cases arise where it is unwise to provide manipulators that allow the alteration of individual fields directly. One example is the class we wrote implementing linked lists, where updating a single link could cause an inconsistent data structure. In this case it was preferable to write a structured manipulator that alters several links in concert, so that the list structure remains consistent. Likewise, there may be rare cases where one would not want to provide an accessor for a field – for example, the one storing a secret password.*

*We haven't really studied examples of classes where a constructor is not needed, but these do occasionally arise where the instances of a class come from a fixed set. An example from the homework assignment is the ProtoCard class, where there are exactly 52 instances,one for each card type in the deck, and no more should ever be created.*

*Rubric: 3 points per basic description, 3 for connection to fields, 4 for description of when to include, 4 for examples.*