CIS 210

Fall 2010 Final Exam Key

Try to work the exam yourself before reading this key. Also, remember that there may be more than one way to solve the programming problems. This key shows just one possible solution to each programming problem.

1. [5 points] What does this code print?

```
int x = 7;
int y = 3;
y = y + x;
y = y + x;
y = y / 2;
System.out.println(y);
```

8

2. [5 points] What does this code print?

```
int sum = 0;
for (int k=1; k <= 10; ++k) {
   if (k % 3 == 0) {
       sum = sum + 1 ;
   }
}
System.out.println(sum);
```

3

3. [8 points] What is printed if we call method q3?

```
static int abdiff(int x, int y) {
    if ( x > y ) {
        x = x - y;
    } else {
        x = y - x;
    }
    return x;
}

static void q3() {
    int x = 7;
    int y = 4;
    x = abdiff( x, 5  );
    y = abdiff( y, 14 );
    y = y + x;
    System.out.println(y);
}
```

12

4. [10 points] Fill in a method to go with the following javadoc header comment. Remember that the header comment is like a contract that your method must fulfill.

```
/**
* Find the average (arithmetic mean) of the
*             positive entries 0 .. k-1 of an array.
* @param arr  The array to average.  It has at least k entries.
* @param k    Number of items in the array to average
*             (i.e., we average arr[0] through arr[k-1],
*              but only the positive numbers are considered)
* @return Average of the positive numbers in the range
*         arr[0] ... arr[k-1], or -1.0 if there are no
*         positive numbers in that range.
* Example:  If arr is [ 2.0, -2.0, 3.0, 0.0, 5.0 ] and k is 4,
*       we return (2.0 + 3.0) / 2, or 2.5
*       We omit -2.0 and 0.0 because they are not positive,
*       and we omit 5.0 because it is at index 4.
*/
public static double avgPos( double[ ] arr, int k ) {
   double sum = 0.0;
   int count = 0;
   for (int i=0; i < k; ++i ) {
       if (arr[i] > 0.0) {
           ++count;
           sum += arr[i];
        }
    }
    if (count == 0) {
       return -1.0;
    }
    return sum / (double) count;
}
```

Here is a ListCell class used by the NameList class on the next page. The code on this page is for reference while completing the question on the next page. You do not need to write anything on this page, and you may remove it from the exam to use as reference while working on the next page.

```
/** Cell in the attendance list. */

class ListCell {

    private String name;
    private int count;
    private ListCell next;

    /**
     * Construct one attendance record in an attendance list.
     * @param name  Keep attendance record for this name
     * @param next  Attach to this list
     */
    public ListCell(String name, ListCell next) {
        this.name = name;
        this.count = 0;
        this.next = next;
    }

    /**
     * Is this the attendance record for name?
     * @param name  Student name to compare to
     * @return true if this attendance record matches the name
     */
    public boolean sameName(String name) {
        return name.equals(this.name);
    }

    /**
     * Increment the attendance count.
     * (Do this after finding or creating a
     *  list cell with a matching name.)
     */
    public void increment() { ++count; }

    // Access to fields
    public String getName() { return name; }
    public int getCount() { return count; }
    public ListCell getNext() { return next; }

}
```

5. [10 points] This code and the code you write may use the ListCell class on the previous page. Assume there is another method that calls `present( name )` with the name of each student present in class each day. You just write the `present()` method to mark attendance in a record matching the name.

```
/**
 * An attendance list. Maintains a count of
 * of attendance for each name.
 */
class NameList {

    private ListCell head = null;

    /**
     * Mark attendance for a student by incrementing the count
     * in the matching ListCell.  The list is not sorted.
     * @param name  Name of student whose attendance should be marked.
     */
    public void present(String name) {
                ListCell cur = head;
                while (cur != null) {
                        if (cur.sameName(name)) {
                                cur.increment();
                                return;
                        }
                        cur = cur.getNext();
                }
                head = new ListCell(name, head);
                head.increment();
    }
```

The next problem asks you to write a search method for looking up a word in a sorted array. Here is the beginning of the class in which that search method appears. You do not need to write on this page, and you may remove it from the exam to use as reference.

```
/**
 * A dictionary, loaded from an external file,
 * with binary search to find a word.
 */
public class Dict {

    private final int CAPACITY = 50000;
    private String[ ] words;
    private int wordCount = 0;

    public Dict( File f ) throws FileNotFoundException {
        words = new String[ CAPACITY ];
        ... omitted code that reads the file into words
        ... and sets wordCount to the number of words
    }

    /**
     * Find a word in the dictionary.
     *   We assume words[0..wordCount-1] are in alphabetical order.
     * @param w A word to look up
     * @return true if the word is present in the dictionary,
     *         otherwise false.
     */
    public boolean find(String w) {
        return search(w,  0, wordCount - 1);
    }
```

(class Dict continues on the next page, where you must fill in the `search` method.)

6. [15 points] The class Dict started on the previous page needs a search method to find a word `w` in the array `words`, which contains around 40,000 words in alphabetical order. You can earn 15 points by writing a binary search, which would be need less than 16 comparisons to search 40,000 words. You can earn up to 10 points by writing a *linear* search method that might have to compare a word to each and every word in the dictionary. You may use recursion or a loop, as you prefer.

Useful methods: Recall that the `s1.equals(s2)` determines whether strings s1 and s2 are equal. Another useful method, `s1.compareTo(s2)`, returns a negative number if s1 is before s2 in alphabetical order, a positive integer if s1 is after s2 in alphabetical order, and zero if they are equal.

```
/**
 * Search to find a word in a sorted array.
 * @param w The word to find
 * @param low   The lowest possible position the word could be at
 * @param high  The highest possible position the word could be at
 * @return true if the word is present, false otherwise.
 */
private boolean search(String w,  int low, int high) {
    if (low > high) return false;
    int probe = (low + high) / 2;
    int order = words[probe].compareTo(w);
    if (order < 0) {
        // The probed word is before the word we want
        return search( w,  probe + 1, high );
    }
    if (order > 0) {
        // The probed word is after the word we want
        return search( w,  low, probe - 1);
    }
    return true;
}
```