CS307 Spring 2010 Midterm 1 Solution and Grading Criteria.

Grading acronyms:
ABA - Answer by Accident
AIOBE - Array Index out of Bounds Exception may occur
BOD - Benefit of the Doubt. Not certain code works, but, can't prove otherwise
ECF - Error carried forward.
Gacky or Gack - Code very hard to understand even though it works or solution is not elegant. (Generally no points off for this.)
GCE - Gross Conceptual Error. Did not answer the question asked or showed fundamental misunderstanding
LE - Logic error in code.
NAP - No answer provided. No answer given on test
NN - Not necessary. Code is unneeded. Generally no points off
NPE - Null Pointer Exception may occur
OBOE - Off by one error. Calculation is off by one.

1. Answer as shown or -2 unless question allows partial credit.
No points off for differences in spacing and capitalization. If quotes included,
okay.

A. 4 5

B. [4, 2, 5] (comma differences and brace differences okay, just need 4, 2, and 5 in that order.)

C. [2, 5, 4] (comma differences and brace differences okay, just need 2, 5, and 4 in that order.)

D. 0

E. The declared type of obj is Object. Object does not have an add method so the compiler will not allow obj.add("Kelly"). (Or words to that effect.)

F. Runtime error or Exception. (No explanation required.)

G. IPod is an abstract class. Abstract classes cannot be created. (or instantiated) (Or words to that effect.)

H. flash 30

I. flash 40
hard drive 20
hard drive 30

J and K. Can be other than valid / invalid. Some way of saying declaration is okay or not. Explanations NOT needed. -1 each part

J. 1. invalid
2. valid

K. 1. valid
2. invalid

L. flash 30

M. hard drive 1

N. hard drive 2

O. Zero or more. (Or words to that effect.)

2. Comments. I thought this would be an easy problem because we had spent so much time on the array based list data structure in class and there were many similar questions on old tests. However a lot of students had trouble dealing with the list abstraction and the concrete array. The real difficulty was in determining the size of the resulting list, ensuring capacity, and then correctly copying elements from the original two lists to the resulting list.

Common problems:
- confusion between a GenericList and an array
- not updating size of resulting list
- calling methods (add, resize, constructor with initial capacity) without defining them
- error in maintaining index into result and index into original
- not ensuring enough capacity in the resulting lists internal array
- confusion between size of list and length of internal array.

Suggested Solution:

```
public GenericList interleave(GenericList other) {
    GenericList result = new GenericList();
    result.size = Math.min(size, other.size) * 2;
    result.container = new Object[result.size];
    int limit = result.size / 2;
    // alternate elements from this list and other
    for(int i = 0; i < limit; i++) {
        result.container[i * 2] = this.container[i];
        result.container[i * 2 + 1] = other.container[i];
    }
    return result;
}
```

Grading Criteria: 25 points

Create result: 2 points

Determine number of elements in result: 3 points

ensure results container has adequate storage: 3 points

set result.size: 2 points

loop to add elements to result: attempt 2 points, correct 4 points

access elements from this and other and place them in correct elements in result. attempt, 2 points, correct 5 points

return result: 2 point

3. Comments: A 2d array problem with objects. Dealing with the neighborhood of cells instead of the whole 2d array was the real challenge as well as ensuring an ArrayIndexOutOfBoundsException did not occur.

Overall students did well on this question.

Common Problems:
- not determining bounds of neighborhood to prevent AIOBE
- not calling methods on color objects correctly

Suggested Solution

```
public static int numGreaterBlueGreen(Color[][] grid, int row,
                                      int col, int size) {

    int result = 0;
    int g = grid[row][col].getGreen();
    int b = grid[row][col].getBlue();
    for(int r = row - size; r <= row + size; r++) {
        for(int c = col - size; c <= col + size; c++) {
            if( 0 <= r && r < grid.length && 0 <= c && c < grid[0].length) {
                if( grid[r][c].getGreen() > g && grid[r][c].getBlue() > b)
                    result++;
            }
        }
    }
    return result;
}
```

Grading Criteria: 20 points

loop through all elements in neighborhood: attempt 2 points, correct 4 points

Ensure coordinates are for valid cell to avoid AIOBE:
attempt 2 points, correct 2 points

access object methods getBlue() and getGreen correctly: 3 points

increment result if color object has more blue and green than color object at center of neighborhood: attempt: 2 points, correct 4 points

return result: 1 point

4A. Comments. People generally did well on this question. I think the people who did the assignment did well on the question as they were familiar with the NameRecord class and using ArrayLists.

Common problems included:

- not accounting for 0's correctly.
- trying to permanently change 0's to 1001's
- not knowing how to use the ArryList
- not taking the absolute value of the difference in ranks

Suggested Solution

```
public boolean inSynch(NameRecord other, int maxDiff) {
    boolean synch = true;
    int decadeIndex = 0;
    final int UNRANKED = 1001;
    while(synch && decadeIndex < this.ranks.size()){
        int r1 = this.getRank(decadeIndex);
        int r2 = other.getRank(decadeIndex);
        if(r1 == 0)
            r1 = UNRANKED;
        if(r2 == 0)
            r2 = UNRANKED;
        int diff = Math.abs(r1 - r2);
        synch = diff <= maxDiff;
        decadeIndex++;
    }
    return synch;
}
```

Grading criteria: 15 points

- loop through correct number of decades: attempt 1, correct 3
- get ranks correctly from each NameRecord object: 2
- correct 0s to 1001: 3
- correctly check absolute value of difference compare to ma allowed: 2
- determine answer correctly: 3
- return result: 1

4B. Comments. The algorithm for this question was not hard (a simple iteration problem), but dealing with a lot of different data types was a challenge. The question included a String, an ArrayList of NameRecords, an ArrayList of Strings, individual NameRecords, and the Names class itself. That was a lot. Again, I think students who did the assignment tended to do well. It was okay to use the String equals method. I did not count off for using == or != which might not give the correct result.

Common problems:

- using [] on ArrayList instead of get method
- confusion on data types
- not ensuring the String name is present.

```
public ArrayList<String> inSynch(String name, int maxDiff){
    ArrayList<String> result = new ArrayList<String>();
    NameRecord target = getName(name);
    if(target != null){
        for(NameRecord other : namesList){
            if(!other.getName().equals(name)
                    && target.inSynch(other, maxDiff)){
                result.add(other.getName());
            }
        }
    }
    return result;
}
```

Grading criteria:

- create result, ArrayList<String>, 1
- get target NameRecord (via method or search of ArrayList<NameRecord), 1
- check if resulting NameRecord is null indicating name is not present and method must return an empty ArrayList (not null), 1
- loop through ArrayList<NameRecord> correctly, 3
- ensure same name not added to results, 1
- check two NameRecords in sync, 1
- if in sync add to result, 1
- return result, 1