CS314 Fall 2025 Exam 1 Solution and Grading Criteria.

Grading acronyms:
AIOBE - Array Index out of Bounds Exception may occur.
BOD - Benefit Of the Doubt. Not certain code works, but, can't prove otherwise.
Gacky or Gack - Code very hard to understand even though it works. (Solution is not elegant. Lack of Zen.)
LE - Logic Error in code.
MCE - Major Conceptual Error. Answer is way off base, question not understood based on answer provided.
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.
RTQ - Read The question. Violated restrictions or made incorrect assumption.
EFF - Efficiency. Order is worse than expected or unnecessary computations done.

1. Answer as shown or -2 unless question allows partial credit. Quotes on output counted as long unless an actual quote gets printed out. (None on this exam.)
Text in parenthesis not required. It is simply grading guidance and / or a brief explanation for answer.

A. 32 seconds
B. 13N + 7 (range: +/- 1.0 allowed on each coefficient)
C. 10 seconds
D. $3N^3 + 5N^2 + 5N + 4$ (range: +/- 1.0 allowed on each coefficient)
E. Thrown out. Everyone +2
F. $O(N^3)$
G. O(N) (Always even, remove last element, no shifting)
H. [C, GO, C, CS]
I. false (class could be abstract)
J. false (don't want extra capacity elements included)
K. 12 seconds (given array of all zeros inner most loop never runs so $O(N^2)$, best case)
L. No compiler errors. (Runtime error yes, but no compile errors)
M. 4 and 6 (can only add Strings)
N. false (not all data structures implement Iterable interface)
O. error compiles
P. compiles error
Q. 105 0
R. t: 5
S. 19
T. Compile error (declared type is Package, no setInsure method)
U. false (uses inherited equals method from Object which is simply this == obj)
V. 52
W. 15 26 15
X. Compile error (Child class cannot access private instance variable of ancestor)
Y. 413 (order of constructor calls, implicit calls to super())

2. Comments: Meant to be a straight forward question. Very similar in nature to past GenericList questions. Mostly about dealing with the abstractions.

```
public GenericList<E> copyWithoutTarget(E target) {

    GenericList<E> result = new GenericList<>();

    // Result could be < size, but add some extra capacity just in case.
    // A more space efficient solution would determine needed
    // capacity of result first OR copy array afterwards.
    result.con = (E[]) new Object[size  + 10];

    for (int i = 0; i < size; i++) {
        if (!con[i].equals(target)) {
            // we want element at con[i] in the result
            result.con[result.size] = con[i];
            result.size++;
        }
    }
    return result;
}
```

17 points, Criteria:

- -1 did not create new GenericList for result, 1 point (lose if any constructor called other than 0 arg unless implemented)
- -1 did not create new array for new GenericList, 1 point (lose if new E[int_exp])
- -1 Not at least 1 element of extra capacity in new array (assuming worst case of size of this list) 1 point
- -3 does not correctly loop through elements of this list, bounded by size, 3 points
- -3 does not check if current element meets requirement to be added using equals method, 3 points (lose if ==)
- -4 does not add new element to correct spot in resulting array, 4 points -- lose if result[i] (Lose if call add method without implementing)
- -3 does not return resulting GenericList
- -1 does not return resulting GenericList

Other possible deductions:

- -3 alter this
- disallowed methods if not alrady points off for other issues related to these methods. size() -1, get(int) -1, add(val) -7
- -3 adding inappropriate public methods that violates encapsulation, (size, get not necessary, but okay)
- -3 Worse that O(N)
- -3 AIOBE or NPE not covered by other criteria
- -3 infinite loop not covered by other criteria
- calling get on an array

3. Comments: Not a Baby Names question. Again, s straightforward question if one completed assignment 2, the MathMatrix assignment. 3 methods (add, subtract, multiply) in that assignment where there are two MathMatrix objects used to create a 3 just like this question. Lots of confusion on the difference between a MathMatrix object and the internal 2d array of ints. Lots of confusion on how to access number of rows and number of columns in a 2d array in Java.

The only real difficulty was handling the column offset of the elements of rhs.

Per the example this was a horizontal concatenation. Not a vertical concatenation. Some solutions though the number of rows in the result would be the sum of the rows of this and rhs.

Saw some nice solutions with an outer loop for rows and then two, not nested, inner loops for columns of each. I think that is a cleaner solution than below.

Lots of confusion on the difference between a MathMatrix object and its internal 2d array of ints, cells.

```
public MathMatrix concatenate(MathMatrix rhs) {

    int newColumns = cells[0].length + ths.cells[0].length;
    MathMatrix result = new MathMatrix(cells.length, newColumns);

    // Copy over the elements of this
    for (int r = 0; r < cells.length; r++) {
        for (int c = 0; c < cells[0].length; c++) {
            result.cells[r][c] = cells[r][c];
        }
    }

    // Now copy over cells from rhs. Tricky part is getting the
    // column in the result correct. Offset by this.cells[0].length
    int offset = cells[0].length;

    // Maybe a helper method with offset as parameter?
    for (int r = 0; r < rhs.cells.length; r++) {
        for (int c = 0; c < rhs.cells[0].length; c++) {
            result.cells[r][c + offset] = rhs.cells[r][c];
        }
    }

    return result;
}
```

16 points, Criteria:

- -1 Does not create resulting MathMatrix correctly using given constructor or with constructor added in solution.
- -2 Dimensions of result not correct. Lose if do not access con.length, con[0].length, rhs.con[0].length correctly. Lose if do not call constructor correctly with proper dimensions.
- -4 does not copy elements from this to result correctly. Lose if does not access result.cells correctly. Lose if don't access this.cells correctly.

- -2 no attempt to copy elements from rhs to result
- -4 does not correctly copy elements from rhs to result. Must account for offset. (Error carried forward if did not access cells 2d array correctly.) Typically lose if a vertical concatenation.
- -2 does not access elements from MathMartix correctly. result[r][c] or rhs[r][c] or rhs[r, c] rhs[0].length are wrong.
- -1 does not return resulting MathMatrix

Other Possible Deductions:

- -1 creates more than one 2d array of the needed size
- -3 worse than O(N^2) assuming 2 NxN matrices.
- -x depends on severity use methods without defining them

4. Comments: The most difficult of the 3 coding questions. If you completed the Spring 2024 practice exam it likely helped tremendously dealing with the abstractions involved. Question 4 on that exam was also a MultiSet question with the same internal instance variables. I chose NOT to take off for stopping the inner loop as soon as a value found.

Common issues included:

- unclear on the difference between distinct elements and size (total elements)
- looping or possibly looping into the extra capacity section of the internal array which would cause null pointer exceptions
- not ensuring some extra capacity in result
- altering this or other
- thinking it was necessary for the outer loop to be the smaller (or longer) of the two MultiSets. Turns out it does not matter. There were many correct solutions that wrote both, but that likely took up a lot of precious time.
- Not creating new ValueAndFrequency objects as necessary. Those are the internal instance variables and fields we need. Shallow copies VERY dangerous as we are likely altering frequency.
- Creating **public** helper methods that broke the encapsulation of the MultiSet. The client does not want to know about ValueAndFrequency objects OR positions. From the client's perspective the elements are in no particular order.

```
/* pre: other != null  post: per the problem description. */
public MultiSet<E> getIntersection(MultiSet<E> other) {

    MultiSet<E> result = new MultiSet<>(numDistinct + 5); // extra cap

    for (int i = 0; i < numDistinct; i++) {
        E target = con[i].element;
        int indexInOther = other.indexOf(target);
        if (indexInOther != -1) {
            int newFreq = Math.min(con[i].frequency,
                                   other.con[indexInOther].frequency);
            result.con[result.numDistinct] = new
                    ValueAndFrequency<>(target, newFreq);
            result.numDistinct++;
            result.size += newFreq;
        }
    }

    return result;
}

private int indexOf(E tgt) {
    for (int i = 0; i < numDistinct; i++)
        if (con[i].element.equals(tgt))
            return i;
    return -1; // never found tgt.
}
```

17 points, Criteria:

- -1 does not create the resulting MultiSet<E>, using the given constructor
- -1 result not always guaranteed to have some extra capacity
- -2 does not have outer loop for number of distinct elements. Can be this or other. (Lose if use for-each loop due to possible nulls)
- -3 does not correctly determine if current element in this is in other (or vice versa), In other words, the inner loop to search for a match for the current outer loop element is not correct. (Can still earn even if use ==, lose if don't check correct portion of array in other.)
- -1 does not use .equals correctly - Must use on elements, not the ValueFrequencyObjects themselves.
- -2 for values added to result, does not determine the correct frequency, min of the two frequencies
- -2 does not create new ValueAndFrequency objects with correct element for the result. Shallow copies NOT okay.
- -2 does not add to correct spot in resulting array. (Lose if don't access result.con correctly.)
- -1 does not update or set size (total number of elements including duplicates) in result correctly
- -1 does not update or set numDistinct in result correctly
- -1 does not return the resulting MultiSet<E>

Other:

- -2 Incorrect use of ValueAndFreqeuncy object from internal array. Typically treating it as an E or not accessing the internal element.
- -2 alter this or other
- -2 create any arrays besides the array for the resulting MultiSet<E>
- -2 method(s) added that break encapsulation

For 1.P through 1.Y, refer to the following classes. Detach this page from the exam. *Courtesy of Miranda*

```
public class Candy {
    private String brand;
    private int price;

    public Candy(String b, int p) {
        brand = b;
        price = p;
    }

    public Candy() {
        price = 3;
    }

    public void inc() { price++; }

    public int getPrice() { return price; }

    public String toString() { return brand + " " + price; }
}

public class Chocolate extends Candy {
    private String name;

    public Chocolate(String n) { name = n; }

    public Chocolate() { super("Hershey", 4);   }

}

public class Jellys extends Candy {
    private String color;

    public Jellys() { color = "rainbow"; }

    public Jellys(String c) { color = c; }

    public String getColor() { return color; }

}


public class MilkyWay extends Chocolate {

    public MilkyWay (){  super("Milk Chocolate"); }

    public String toString() { return "MW"; }

}
```