COP 3330 Spring 2026 Final Exam Part B Solutions

Please print your name in CAPITAL letters.

1) (10 pts) You've found an amazing investment. If you put in D dollars one year, by the next year you get 2D – 5 dollars. For obvious reasons, the minimum initial investment is 6 dollars. You've decided that you'll start with some amount D ≥ 6 dollars and reinvest your return each year until you reach L dollars. Write a static method that takes in both the values of D and L, respectively, and returns the fewest number of years you need to mature your initial investment of D dollars to have L or more dollars. Do not worry about overflow errors in your implementation. (Formally, you may assume that both D and L are in between 6 and 1 billion, inclusive. Note that if D ≥ L, then your method should return 0.)

```
public static int timeInvest(int D, int L) {

    int res = 0;                            // 2 pts

    while (D < L) {                         // 2 pts
        D = 2*D - 5;                        // 2 pts
        res++;                              // 2 pts
    }

    return res;                             // 2 pts

}
```

2) (10 pts) Write a static method that takes in a Random object, rndObj, a double lowBound, a double highBound, and an integer numValues, and returns an array with numValues randomly generated doubles in between lowBound and highBound, respectively. Generate the random values using rndObj.

```
public static double[] rndReals(Random rndObj, double lowBound,
                                double highBound, int numValues) {

    double[] res = new double[numValues];                                  // 2 pts

    for (int i=0; i<res.length; i++)                                       // 2 pts
        res[i] = rndObj.nextDouble()*(highBound-lowBound)+lowBound;        // 5 pts

    return res;                                                            // 1 pt
}
```

Grading: Breakdown long stmt – 1 pt LHS, 1 pt nextDouble call, 1 pt mult
1 pt for (hB-lB), 1 pt for adding lB

For the next few questions we'll deal with creating a class called CaffeinatedBeverage. The instance variables for the class will be:

```
protected int numOz;
protected double mgCaffeine;
```

We'll maintain the object such that the number of ounces (first instance variable) is always an integer, while the second instance variable may equal not always be integral. At all points in time, we assume that all individual CaffeinatedBeverage objects have their caffeine perfectly equally distributed throughout the whole beverage. Thus, if you drink half of the ounces of the beverage, you'll also consume half of its caffeine. Here is the constructor for the class:

```
public CaffeinatedBeverage(int oz, double caff) {
    numOz = oz;
    mgCaffeine = caff;
}
```

Each of the following questions will ask you to add methods to the CaffeinatedBeverage class.

3) (6 pts) Write a method called pour. This method will take in a CaffeinatedBeverage other, and "pour" the contents of it into this object. You should add the full contents of other into this AND empty out the other object.

```
public void pour(CaffeinatedBeverage other) {

    this.numOz += other.numOz;                  // 2 pts
    this.mgCaffeine += other.mgCaffeine;        // 2 pts
    other.numOz = 0;                            // 1 pt
    other.mgCaffeine = 0;                       // 1 pt
}                                          // order matters, this doesn't =)
```

4) (8 pts) Write a method drink. This method will take in a positive integer, quantity, modify this object as if quantity ounces of this object were consumed. Assume that 0 < quantity ≤ this.numOz.

```
public void drink(int quantity) {

    double caffDrink = mgCaffeine*quantity/numOz;     // 4 pts
    numOz -= quantity;                                // 2 pts
    mgCaffeine -= caffDrink;                          // 2 pts

}
```

Now, we'll create a class ProteinCaffeinatedBeverage which inherits from CaffeinatedBeverage. For the following question, you'll write the whole class. It will contain a single instance variable:

```
protected double gProtein;
```

representing the amount of protein in the beverage in grams. Just like caffeine, we assume the protein is equally distributed amongst the beverage.

5) (15 pts) Complete ProteinCaffeinatedBeverage class below. For full credit, **you must use super calls any time they are possible to use. Be careful when writing the drink method!**

```
public class ProteinCaffeinatedBeverage extends CaffeinatedBeverage { // 1 pt

    protected double gProtein;

    public ProteinCaffeinatedBeverage(int oz, double caff, double protein) {

        super(oz, caff);                    // 2 pts, must use super to get pts.
        gProtein = protein;                 // 1 pt

    }

    public void pour(ProteinCaffeinatedBeverage other) {

        super.pour(other);                  // 2 pts, must use super to get pts.
        gProtein += other.gProtein;         // 2 pts
        other.gProtein = 0;                 // 1 pt

    }

    public void drink(int quantity) {

        gProtein -= gProtein*quantity/numOz;    // 3 pts
        super.drink(quantity);                  // 3 pts, order matters here.
                                                // 2 pts call, 1 pt order
    }
}
```

6) (20 pts) Consider a 2D integer array storing elevations for plots of land in a mountainous region. A valley is a plot of land where the 8 adjacent plots of land (horizontally, vertically or diagonally on the grid) all have strictly greater elevation. Valleys are desired since the adjacent plots of land block wind. We define the safety of a valley to be the minimum of the differences of elevation between any adjacent plot of land and the valley plot of land. In the example plot below, the valley square with elevation 4 shown in bold, has a safety value of 10, since of its 8 adjacent plots of land, the minimum elevation is 14 (bottom right) and 14 – 4 = 10.

16	20	30	12	18
15	**4**	22	22	16
17	31	14	14	19
47	18	19	15	33

Complete the method below so that it takes in a 2D integer array (all positive distinct values) of size 3 by 3 or larger and returns the maximum safety value of any valley square. You are guaranteed that at least one valley square exists in the input array. Valley squares **must** be in the interior of the grid with 8 adjacent grid squares. For full credit, use the DR, DC arrays provided.

```
final public static int[] DR = {-1,-1,-1,0,0,1,1,1};
final public static int[] DC = {-1,0,1,-1,1,-1,0,1};

public static int maxSafety(int[][] land) {


    int res = 0;                                                    // 1 pt

    for (int i=1; i<land.length-1; i++) {                           // 2 pts
        for (int j=1; j<land[i].length-1; j++) {                    // 2 pts

            int minAround = land[i+DR[0]][j+DC[0]];                 // 1 pt

            for (int z=0; z<DR.length; z++)                         // 1 pt

                // 8 pts
                minAround = Math.min(minAround, land[i+DR[z]][j+DC[z]]);

            // 4 pts total
            if (minAround > land[i][j])
                res = Math.max(res, minAround-land[i][j]);
        }
    }

    return res;                                                     // 1 pt
}
```

7) (16 pts) Common set operations are union and intersection. The union of two sets is the set of unique items that appear in either set. The intersection of two sets is the set of items that appear in both sets. These are not built in functions in the HashSet class in Java. Write two static methods that take in two sets, a and b, and return the union and intersection, respectively, of the two sets in a new HashSet. You should NOT make any changes to the sets a and b in either method. Rather, you should create and return a new set that is the result of the appropriate operation.

```
public static HashSet<Integer> union(HashSet<Integer> a, HashSet<Integer> b) {

    HashSet<Integer> res = new HashSet<Integer>();          // 1 pt
    for (Integer x: a) res.add(x);                          // 3 pts
    for (Integer x: b) res.add(x);                          // 3 pts
    return res;                                             // 1 pt

}

public static HashSet<Integer> intersect(HashSet<Integer> a, HashSet<Integer>
b) {

    HashSet<Integer> res = new HashSet<Integer>();          // 1 pt
    for (Integer x: a)                                      // 2 pts
        if (b.contains(x))                                  // 2 pts
            res.add(x);                                     // 2 pts

    return res;                                             // 1 pt

}
```

8) (8 pts) Write a segment of code that attempts to open a file called "testscores.txt" from which to read to. If the file exists, simply print out, "File Opened Successfully." If the file doesn't exist, catch the "FileNotFoundException" and print out an error message "File Not Found." without the program crashing.

```
try {                                                       // 1 pt

    Scanner fin = new Scanner(new File("testscores.txt"));  // 3 pts
    System.out.println("File Opened Successfully.");        // 1 pt

}
catch (FileNotFoundException e) {                           // 2 pts

    System.out.println("File Not Found.");                  // 1 pt
}
```

9) (5 pts) In my last lecture, I had 3 takeaway messages. Summarize any two of them in your own words.

Notes are linked here:
https://www.cs.ucf.edu/~dmarino/ucf/cop3330/lectures/OOP-38-ClassLessons.pdf

My intended answers are:

#1: Most important class lessons are language agnostic (independent of programming language)

#2: Go to class!!!

Grading: Arup's discretion 100%

10) (2 pts) What type of activity does the student organization Volunteer UCF help students find opportunities for?

Volunteer Opportunities! **(Grading: Give to All)**