CS312 Fall 2016 Final 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.)
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.
RTQ - Read the question. Violated restrictions or made incorrect assumption.

1. Expressions - 1 point each. -1 for missing or extra " OR .0. differences in capitalization of `true` and `false` okay

A. `3.5`
B. `7`
C. `"5dog45"`
D `1.4`
E. `39`
F. `true` (Not a runtime error. Thanks to short-circuiting, second part not evaluated.)
G. `"true false"`
H. `"sw"`
I. `false`
J. `false`

2. Program Logic - 1 point each

	temp < num	result <= temp	temp == 0	half == true
Point A	**S**	**A**	**A**	**N**
Point B	**A**	**S**	**S**	**S**
Point C	**A**	**N**	**S**	**A**
POINT D	**S**	**S**	**S**	**S**
POINT E	**N**	**S**	**N**	**A**

3. Code Tracing - 1 point each. Answer as shown or -1. -1 for first two occurrences of "'s. Differences in spacing, commas, and grouping symbols okay for array output.

A. 8 4 5 3
B. [4.5, 6.0, 2.0
C. [0, 5, 0], [3, 7]
D [GC, 4, B, C]
E. Runtime error or exception or Null pointer exception (Error or syntax error not okay)
F. 5 0
G. 10.0 true
H. [D, B, A, C, B]
I. NO (circle NO)
J. syntax error (.5 EACH ON J and K)
legal
K. syntax error
legal
L. TI5 15
M. 5 15
N. WD15 TT
O. 1. no constructor in parent that has 2 int parameters
2. cannot access private field dataRate
(Order of two answers does not matter. -.5 for each)

4. Critters - 15 Points. Sidewinder

```
public class Sidewinder extends Critter {

    private int stepsSoFarThisLeg;
    private int totalLegLength;
    private int addToLeg;
    private Direction move;

    public Sidewinder(int addToLeg) {
        totalLegLength = 1;
        move = Direction.NORTH;
        this.addToLeg = addToLeg;
    }

    public Color getColor() {
        return Color.GRAY;
    }

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

    public Attack fight(String opp) {
        Attack result = Attack.SCRATCH;
        if (opp.equals("%")) {
            result = Attack.ROAR;
        }
        return result;
    }

    public boolean eat() {
        return totalLegLength > 20;
    }

    public Direction getMove() {
        Direction result = move;
        stepsSoFarThisLeg++;
        if (stepsSoFarThisLeg == totalLegLength) {
            stepsSoFarThisLeg = 0;
            totalLegLength += addToLeg;
            if (move.equals(Direction.NORTH)) {
                move = Direction.WEST;
            } else {
                move = Direction.NORTH;
            }
        }
        return result;
    }
}
```

Points:
instance variables: 2 (can be different, but must track steps, leg length, add to leg, and current direction)
Constructor: 2
getMove: 5 total -> increment steps: 1, check if time to change direction: 2: change direction 2
fight: 2 :-2 if == on String instead of .equals
eat: 2
toString: 1
getColor: 1

5. Arrays 15 Points.

```
public static boolean difficultTrail(int[] elevations,
        int minChange, int minSegments) {

    // special case if not possible to have enough segments
    if (minSegments > elevations.length - 1)
        return false;

    // general case
    int difficultSegments = 0;
    for (int i = 1; i < elevations.length; i++) {
        int change = elevations[i] - elevations[i - 1];
        if (change >= minChange) {
            difficultSegments++;
            if (difficultSegments == minSegments) {
                return true; // no need to check any more
            }
        }
    }
    // not enough segments after loop
    return false;
}
```

return false immediaitely if not enough segments: 2 points
counter for number of difficult segmets: 1 point
loop with correct bounds: 4 points (-3 if AIOBE will occur)
correctly calculate current elevation change: 3 points
correctly check if current segment change >= minChange: 1 point
counter for number of difficult segmets: 1 point
return true as soon as number of difficult segment equals minSegments: 2 points
return false if not enough segments: 1 point

6. Strings 15 Points.

```
public static String acronym(String phrase) {
    String result = "";
    phrase = phrase.toUpperCase();
    boolean inWord = false;
    for (int i = 0; i < phrase.length(); i++) {
        char ch = phrase.charAt(i);
        if (ch == ' ' || ch == '-') {
            inWord = false;
        } else if (!inWord) {
            inWord = true;
            result += ch;
        }
    }
    return result;
}
```

create resulting String, initially empty: 1 point
loop through all characters: 3 points
correctly add first letter of words to result: 5 points (partial credit possible)
no other chars added to result: 3 points
ensure result is uppercase: 2 points
return result: 1 point

Okay to try and create new string with single space between words.

Only allowed methods: size, charAt, toUpperCase. Concatenation okay.
No other classes or methods. Take off 3 to 6 points depending on severity.

7. 2D Arrays. 15 Points.

```
    public static int indexOfMaxPointsDifference(int[][] scores) {
        int best = -1; // if data in scores is correct must be a team
        // with a positive point differential
        int indexOfBest = -1;

        for (int teamIndex = 0; teamIndex < scores.length; teamIndex++) {
            int totalPointsFor = 0;
            int totalPointsAgainst = 0;
            for (int oppIndex = 0; oppIndex < scores.length; oppIndex++) {
                if (scores[teamIndex][oppIndex] != -1) {
                    totalPointsFor += scores[teamIndex][oppIndex];
                    totalPointsAgainst += scores[oppIndex][teamIndex];
                }
            }
            int pointsDiff = totalPointsFor - totalPointsAgainst;
            if (pointsDiff > best) {
                best = pointsDiff;
                indexOfBest = teamIndex;
            }
        }
        return indexOfBest;
    }
```

variable for index of best: 1 point
variable for best point differential, initialized appropriately: 1 point
outer loop for rows (or columns): 2 points
inner loop: 1 point
check to ensure cell does not contains a -1: 2 points
track current team points for and against correctly: 5 points
correctly check if current team has best point difference so far and update tracking vars: 2 points
return correct result: 1 point

OTHER PROBLEMS:

hard coded values: -9
returns max points instead of index of max, -3

8. ArrayList - 10 points

```
    public static void removeValuesTwiceAverage(ArrayList<Double> list) {
        double total = 0.0;
        for (int i = 0; i < list.size(); i++) {
            total += list.get(i);
        }
        double twiceAve = (total / list.size()) * 2;

        // start from the back and move forwards to avoid
        // skipping elements when removing:
        for (int i = list.size() - 1; i >= 0; i--) {
            if (list.get(i) >= twiceAve) {
                list.remove(i);
            }
        }
    }
```

[] instead of get(index) - 2,

Calculate ave: 5 points Attempt to remove correct values: 2 points Remove correct: 3 points

9. File Processing - 20 Points:

```
public static void printAPScoreAverage(Scanner sc, int grade,
        boolean STEM) {

    while(sc.hasNextLine()) {
        int studentGrade = sc.nextInt();
        String name = sc.nextLine(); // skips next line if not needed
        String testData = sc.nextLine();
        if (studentGrade == grade) {
            double average = averageScore(testData, STEM);
            System.out.print(name + ": ");
            if (average == -1.0) {
                System.out.println("NO EXAMS");
            } else {
                System.out.println(average);
            }
        }
    }
}

private static double averageScore(String lineData, boolean STEM) {
    double totalScore = 0.0;
    int numExams = 0;
    Scanner line = new Scanner(lineData);
    while (line.hasNextInt()) {
        // we know at least one more score
        int score = line.nextInt();
        boolean currentScoreSTEM = false;
        if (line.hasNext() && !line.hasNextInt()) {
            // We know there is another token that isn't an int
            // so we know this is a STEM score
            currentScoreSTEM = true;
            // consume the token
            line.next();
        }
        if (currentScoreSTEM == STEM) {
            totalScore += score;
            numExams++;
        }
    }
    return numExams == 0 ? -1 : totalScore / numExams;
}
```

Loop for more lines: 1 point
Get grade from line correctly: 1 point
Get name correctly: 1 point
Check to ensure grade matched correctly: 2 points
Attempt to process grade line: 3 points
Gets one score: 2 points
Correctly checks if score has asterisks following it: 3 points
Processing entire line of scores: 1 point
Track total and number of correct kind of test: 3 points
Correctly prints out average of NO EXAMS: 3 points