CS312 Fall 2018 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. No limit on points off.

A. 5
B. 17
C. false
D. 'D'
E. ""
F. true
G. false (always false due to Math.random() return values.)
H. 5.6
I. "9RING55"
J. 5

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

A. [3, 4, 8]
B. [4, 0, 2]
C. [0, 0, 0, 0]
D. [1, 3, 5, 12, 9]
E. 0 [F, E, GC]
F. 4 8 0.0
G. false
H. [1, 10, 4]
I. [0, 1, 5, 12, 0]
J. RUNTIME ERROR (Due to / by 0)
K. 5 [0, 2] 7
L. INFINITE LOOP (due to boolean expression in while loop never equaling false.)
M. .5 each syntax error (no constructor for PC that takes an int)
   legal
N. .5 each syntax error (Computer not a subtype of MAC)
   syntax error (Mac not a subtype of PC)
O. 8 4
P. 10
Q. SYNTAX ERROR (cp's declared type is Computer which does not have a getColors method.)
R. 10
S. false false
T. 16
U. true
V. Because we don't have access to the private instance variable memory OR words to that affect.
   (Including no variable named memory in scope.)
public class Yak extends Critter {
    private static Direction[] dirs = Direction.values();
    private Direction dir;
    private int maxSteps;
    private int steps;

    public Yak () {
        dir = getRandomDirection();
        maxSteps = 1;
    }

    private Direction getRandomDirection() {
        int index = (int) (Math.random() * 4);
        return dirs[index];
    }

    public boolean eat() { return true; }

    public Direction move() {
        Direction result = dir;
        steps++;
        if (steps == maxSteps) {
            steps = 0;
            maxSteps++;
            dir = getRandomDirection();
        }
        return result;
    }

    public Attack fight(String opp) {
        Attack result = Attack.ROAR;
        if (dir == Direction.EAST || dir == Direction.WEST) {
            result = Attack.POUNCE;
        }
        return result;
    }
}

Points:
header correct with extends clause: 1 point
instance variables: 1 point (can be different, but must track steps, leg length and current direction)
instance vars private: 1 point
correctly picks random direction, 2 points
eat overridden correctly: 1 point
fight method:
    check correct next direction and return correct value 4 points
    -4 if call getMove in fight
getMove
    correctly handles incrementing steps this leg and direction to return, 2 points
    correctly handles case when end of leg reached and steps, leg length, and direction updated, 4 points

Other Deductions:
-6 loop in getMove method / trying to move multiple times in getMove
public static boolean capitalLettersPresent(Scanner sc, int[] required) {
    while (sc.hasNext()) {
        String s = sc.next();
        for (int i = 0; i < s.length(); i++) {
            char ch = s.charAt(i);
            if ('A' <= ch && ch <= 'Z') {
                int index = ch - 'A';
                if (required[index] > 0) {
                    // if check not required in soln's
                    required[index]--;
                }
            }
        }
    }
    // now check that all values are 0
    for (int numLeft : required) {
        if (numLeft > 0) {
            return false;
        }
    }
    // All zeros.
    return true;
}

- correctly loop while hasNext true for Scanner, 2 points
- get next token from Scanner, 2 points
- loop through all characters of token, 2 points
- correctly check that current char is a capital letter, 4 points
- correctly update counter for given letter, 3 points
- after reading all tokens, correctly check if any counters > 0 and return false, 3 points
- return true (or correct answer if requirements met), 1 point

Other Deductions:
-4 loop through String of capital letters or use indexOf instead of mapping
-4 scanner loop on inside of some other loop, attempting to scan file multiple times
-3 concat to create new String version of input
-2 convert input to upper case
public static int[] copyWithOutRange(int[] vals, int start, int stop) {
    int newLen = vals.length - (stop - start);
    int[] result = new int[newLen];
    // copy elements up to index start in vals
    for (int i = 0; i < start; i++) {
        result[i] = vals[i];
    }
    // copy elements from index stop to end of vals;
    int indexResult = start;
    for (int i = stop; i < vals.length; i++) {
        result[indexResult] = vals[i];
        indexResult++;
    }
    return result;
}

- creating resulting array of correct size: 2 points
- correctly add elements up to index start to result, 4 points
- loop to add elements from stop to end of original array to result, 4 points
- index correct for resulting array, 4 points
- return result, 2 point

Other Deductions:
-4 return original array (vals) if no elements removed (instead of making copy)
-4 for altering values in the parameter vals
-2 one loop, with if statement, possibly very inefficient if most elements removed.
Arrays and Objects. 17 Points.

```java
public static double minDistance(Point[] pts) {
    double min = pts[0].distance(pts[1]);
    for (int i = 0; i < pts.length; i++) {
        Point p1 = pts[i];
        for (int j = i + 1; j < pts.length; j++) {
            Point p2 = pts[j];
            double distance = p1.distance(p2);
            if (distance < min) {
                min = distance;
                /* Stop if hit 0. Questionable if this is worthwhile. Certainly not required. */
                if (min == 0.0) {
                    return 0.0; // can't do better than that
                }
            }
        }
    }
    return min;
}
```

- variable for min distance of type double, 1 point
- correctly initialize min distance to distance between first two Points or Double.MAX_VALUE. Also okay to use flag for first time. (Gacky) Lose this if use Integer.MAX_VALUE or some arbitraty value, 2 points
- outer loop correct for all Points, 3 points (okay if subtract 1 from length)
- inner loop correct with bounds so we check all Points after current Point based on outer loop, 5 points
  - lose if check all Points
  - -2 efficiency if check all Points but guard with if
- correctly access Point objects from array, 1 point
- correctly calculate distance using distance method, 2 points
- correctly check if current distance less than min so far and update distance, 2 points
- return correct result, 1 point (not necessary for min = 0.0 check.)

Other Deductions:
-7 no inner loop /only check adjacent pairs
7. ArrayList - 16 points

```java
public static int removeStrings(Scanner sc, ArrayList<String> list) {
    int count = 0;
    while (sc.hasNext()) {
        String s = sc.next();
        boolean search = true;
        int i = 0;
        while (i < list.size() && search) {
            if (list.get(i).equals(s)) {
                search = false;
                list.remove(i);
                count++;
            }
            i++;
        }
    }
    return count;
}
```

- counter for number removed: 1 point
- while loop correct for Scanner (hasNext or hasNextLine): 2 points
- get next token: 1 point
- loop for elements on ArrayList, 3 points
- stop when find first occurrence, 2 points (-1 if use boolean, but loop through whole list)
- use size method and get method for ArrayList, 2 points (1 each)
- use equals method from String class, 2 points
- remove from list and increment counter if matches, 2 points (1 each)
- returns correct result, 1 point

Other Deductions:
- 5 swapped loop, main Scanner loop inside a loop for elements in list,
- 5 removing multiple or all elements from list on first occurrence of word
- 4 treating list like an array, [index] instead of get(index)
- 4 using indexOf or contains methods for ArrayList
8. 2D Arrays - 16 points

```java
public static void clampValue(int[][] mat, int r, int c, int w, int h, int tgt) {
    int startRow = r - h + 1;
    if (startRow < 0) {
        startRow = 0;
    }
    int startCol = c - w + 1;
    if (startCol < 0) {
        startCol = 0;
    }
    for (int row = startRow; row <= r; row++) {
        for (int col = startCol; col <= c; col++) {
            if (mat[row][col] < tgt) {
                mat[row][col] = tgt;
            }
        }
    }
}
```

```
// alternate solution with while loops
public void clampValues(int[][] mat, int r, int c, int w, int h, int tgt) {
    int row = r;
    int endRow = r - h + 1;
    int endCol = c - w + 1;
    while (row >= 0 && row >= endRow) {
        int col = c;
        while (col >= 0 && col >= endCol) {
            if (mat[row][col] < tgt) {
                mat[row][col] = tgt;
            } else { // else is added here for completeness
            }
            col++;
        }
        row++;
    }
}
```

- nested loop, 3 points
- bounds check row correctly, 3 points
- bounds check column correctly, 3 points
- treat r, c as lower right corner, 2 points
- access array elements correctly, 1 point
- check if element less than target and set if necessary, 3 points
- don't perform unnecessary checks, 1 point

Other Deductions:
disallowed methods, -5 points off
AIOBE, -4
OBOE, first -2, second and subsequent, -1
infinite loop, -6