

CS312 Fall 2017 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. "falseH101.5"

B. 59.0

C. 20.5

D. 25.0

E. true

F. false

G. -4.0 (-5.0 also accepted due to unexpected weirdness of Math.round())

public static long round(double a)

Returns the closest long to the argument, with ties rounding to positive infinity.

H. false

I. 10

J. -4

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.

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. [3, 2, 12, 7, 1]

B. 4.0 2.0

C. 0 7

D. true false

E. false true

F. 10 40

G. [5]

H. [2, 3, 9]

I. [1, 2, 4, 8, 16, 33]

J. [-3, -3, -1, -9, -9, -35]

K. 3 SYSTEMS

L. Syntax error (va++ illegal)

M. [K, B, A, G]

N. 0

O. .5 each

syntax error (no 0 arg
constructor)

legal

P. .5 each

syntax error

syntax error

Q. room: 30

R. room: 30

S. room: 25

T. syntax error (access private
field)

U. room: 20

V. room: 10 true

W. true false

X. [-2, 4][-2, 5]2 4 3

3. Critters - 16 Points. JumpinBean

```
public class JumpinBean extends Critter {

    private int stepsSoFar;
    private int maxStepsThisLeg;
    private boolean moving;
    private Direction dir;

    // Constructor not necessary. Default value okay
    // for all fields okay.

    public Attack fight(String opp) {
        if (moving)
            return Attack.FORFEIT;
        moving = true;
        maxStepsThisLeg += 2;
        stepsSoFar = 0;
        dir = Direction.NORTH;
        if (Math.random() >= 0.5)
            dir = Direction.WEST;
        return Attack.SCRATCH;
    }

    public Direction getMove() {
        if (!moving)
            return Direction.CENTER;
        stepsSoFar++;
        if (stepsSoFar == maxStepsThisLeg)
            moving = false;
        // OR moving = stepsSoFar == maxStepsThisLeg;
        return dir;
    }
}
```

Points:

header correct with extends clause: 2 points

instance variables: 1 point (can be different, but must track steps, leg length, current direction, and if moving / celebrating)

instance vars private: 1 point

Attack method:

forfeit if moving: 1 point

if not moving, prep for moving correctly: 5 points

return SCRATCH if not moving: 1 point

getMove

if not moving, returns CENTER: 1 point

if moving increment steps: 1 point

check to see if end of leg and stop moving: 2 points

return direction if moving: 1 point

Other Deductions:

-6 if fight calls getMove or getMove calls fight

-5 no setting values in fight method so we know if we won

-5 picks a random direction every step

-7 loop in getMove method / trying to move multiple times in getMove

4. Data Processing - 18 Points.

```
public static void printMoney(Scanner sc) {
    final double GALLEON_PER_SICKLE = 1 / 17.0;
    final double GALLEON_PER_KNUT = 1 / 29.0 / 17.0;
    while(sc.hasNextLine()) {
        Scanner line = new Scanner(sc.nextLine());
        while(!line.hasNextInt()) {
            System.out.print(line.next() + " ");
        }
        double total = 0.0;
        while (line.hasNext()) {
            int num = line.nextInt();
            String symbol = line.next();
            if (symbol.equals("K")) {
                total += GALLEON_PER_KNUT * num;
            } else if (symbol.equals("S")) {
                total += GALLEON_PER_SICKLE * num;
            } else if (symbol.equals("G")) {
                total += num;
            }
        }
        System.out.print(total + " Galleon");
        if (total != 1.0) {
            System.out.print("s");
        }
        System.out.println();
    }
}
```

loop through lines correctly: 2 points

create Scanner for line: 2 points

print out or save name as String correctly: 3 points (-1 if forget space " ")

variable for total galleons for current person: 1 point

loop while has next in line correctly: 2 points

read in int value correctly: 1 point

get and check symbol correctly: 4 points (-3 for == instead of .equals)

add correct value to running total (okay if 3 separate variables): 2 points

print number of Galleon correctly, including no s if 1.0: 1 point

Other Deductions:

5. Arrays 12 Points.

```
public static int[] getReversedSubList(int[] vals, int startIndex,
    int stopIndex) {

    int[] result = new int[stopIndex - startIndex];
    int indexVals = stopIndex - 1;
    for (int i = 0; i < result.length; i++) {
        result[i] = vals[indexVals];
        // or stopIndex - i - 1 for vals[]
        indexVals--;
    }
    return result;
}
```

creating resulting array of correct size: 2 points
loop of correct bounds (multiple correct alternatives): 4 points
copy value into correct spot in resulting array: 4 points
return result: 2 points

Other Deductions:

-4 for altering values in the parameter vals

6. Arrays. 16 Points.

```
public static void insertElementsAtFront (int[] one, int[] two) {
    int numMove = one.length - two.length;
    // move items in original array down
    for (int i = numMove - 1; i >= 0; i--) {
        one[i + two.length] = one[i];
    }
    // determine limiting length
    int limit = two.length;
    if (one.length < two.length)
        limit = one.length;
    for (int i = 0; i < limit; i++) {
        one[i] = two[i];
    }
}
```

shift elements down in first array correctly: 7 points (partial credit possible, OBOE -2)

determine limit on number of elements to copy in case two is larger: 5 points

copy elements from second array into first correctly: 4 points (partial credit possible, OBOE -2)

Other Deductions:

-4 Nested loop, efficiency

-4 if creates a new array

-2 if try and return a value

-3 if forget that these are value parameters. In other words creating a temp array, altering it, and then saying
one = temp;

7. ArrayList - 16 points

```
public static int removeValues(ArrayList<String> list, char c, int n){
    // start from the back to avoid skipping over values
    int numRemoved = 0;
    for (int i = list.size() - 1; i >= 0; i--) {
        String s = list.get(i);
        // look for c in first n characters of current String
        boolean found = false;
        int index = 0;
        while (index < n && !found && index < s.length()) {
            found = s.charAt(index) == c;
            index++;
        }
        if (found) {
            list.remove(i);
            numRemoved++;
        }
    }
    return numRemoved;
}
```

counter for number removed: 1 point

loop from back of ArrayList (or other technique) to avoid skipping elements on remove: 5 points

loop through current String:

 bounds check length of String: 3 points

 bounds check n: 1 point

 stop when found: 1 point

 access char correctly: 1 point

if char found in String

 remove from list: 2 points

 increment counter: 1 point

return number removed: 1 point

Other Deductions:

-4 treating list like an array, [index] instead of get(index)

-4 creating new Strings

-4 removing excess elements due to logic error

8. 2D Arrays - 18 points

```
public static int coinsCollected(int[][] mat, int initialRow) {
    int col = 0;
    int row = initialRow;
    int total = 0;
    while (col < mat[0].length - 1) {
        total += mat[row][col];
        mat[row][col] = -1; // so we don't come back
        int up = -1;
        if (row > 0)
            up = mat[row - 1][col];
        int down = -1;
        if (row + 1 < mat.length)
            down = mat[row + 1][col];
        int right = mat[row][col + 1];
        if (up >= down && up >= right) {
            row--;
        } else if (down >= up && down >= right) {
            row++;
        } else {
            col++;
        }
    }
    // get the last cell
    total += mat[row][col];
    return total;
}
```

Track row and column correctly: 2 points

Loop while not in the last column: 4 points

Check 3 directions for max coin with bounds checks: 4 points

Alter cells visited so robot doesn't go back: 5 points

Pick correct direction and break ties correctly: 3 points

Other Deductions:

-4 unnecessary nested for loops

-6 infinite loop due to logic error

-2 solution does not handle single row matrix correctly

-1 return missing