CS312 Fall 2018 Exam 2 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. Code Trace:

A. `7`
B. `false`
C. `yball`
D `TeXaS 5  (TEXAS 5 is wrong, substring returns a String but code doesn't set d1 to refer to that new String)`
E. `ack_`
F. `5`
G. `ge--2.72g27.50 (2 clear underscores between ge and 2.72g27.50`
H. `47`
I. `RUNTIME ERROR (Exception okay as well)`
J. `The Scanner creation must be in a try-catch block OR the method must throw a FileNotFound (or IOException). (Or words to that affect.)`
K. `3 6`
L. `19 vector`
M. `0 [0, 0, 0]  (differences in bracket style okay, but some form of bracket must be present)`
N. `[-3, 3, 4, 13] (differences in bracket style okay)`

2. Program Logic (0.5 each)

	c < x	e < 5	e > 0
POINT A	A	A	N
POINT B	A	A	S
POINT C	S	A	S
POINT D	S	S	A
POINT E	S	S	S

3. Strings - 10 Points. Write a method `getEqualStart` that accepts two `String`s as parameters.

```
    public static String getEqualStart(String s1, String s2) {
        int i = 0;
        int limit = s1.length();
        if (s2.length() < s1.length)
           limit = s2.length();
        while (i < limit && s1.charAt(i) == s2.charAt(i)) {
            i++;
        }
        return s1.substring(0, i);
    }
```

```
Alternate solution using a boolean and concatenation

public static String getEqualStart(String s1, String s2) {
        int i = 0;
        String result = "";
        boolean same = true;
        while (i < s1.length() && i < s2.length() && same)) {
            if (s1.charAt(i) == s2.charAt(i)) {
                result += s1.charAt(i);
                i++;
            } else
                 same = false;
        }
        return result;
```

correctly check index for charAt is less than both String lengths (no String index out of bounds errors), 3 points
correctly access characters via charAt, 1 point (lose this if create many new Strings using substring instead of charAt)
correctly track and increment current index, 1 point
stop as soon as characters mismatch. (return from inside loop okay), 2 points
correctly build up result via concatenation or creating String at end via substring method, 2 points
return result, 1 point

OTHER:
infinite loop, - 6
bounds check wrong in this way: s1.charAt(i) == s2.charAt(i) && i < limit, -2
adding too many chars in addition to bounds check off, -2
nested loop (typically not correct, too many chars added) -5

4. 17 points Write a method that given a `Scanner` already connected to a file and a target `char`, returns the token in the file that contains the target `char` the largest number of times.

```
    public static String tokenWithMost(Scanner sc, char tgt) {
        String result = "";
        int max = 0;
        while (sc.hasNext()) {
            String token = sc.next();
            int count = 0;
            for (int i = 0; i < token.length(); i++) {
                if (token.charAt(i) == tgt) {
                    count++;
                }
            }
            if (count > max) {
                result = token;
                max = count;
            }
        }
        return result;
    }
```

variables to track max frequency and init to value <= 0, 1 point
variable to track String with most, 1 point
while with hasNext(), 3 points
get next token correctny, 1 point
loop through token correctly, 3 points (lose this if use indexOf or anything besides charAt and legnth)
count number of occurrences of target char in current token correctly, 3 points (if statement, variable, increment on matches)
correctly check if current token has largest number of occurrences of target and re assign variables correctly, 3 points (-1 of do this inside inner loop)
correctly return empty String if no occurrences, 1 point (can achieve via init var to "")
return correct result, 1 point

early return, -6
not resetting inner count variable for number of chars in token, -3
creating new Scanners, -4
substring method, -4

5. programming. Write a method, `lines` that generates random line segments until the sum of the length of the generated line segments is greater than a given limit

```
    public static int lines(int min, int max, int limit) {
        int count = 0;
        int total = 0;
        int range = max - min + 1;
        Random r = new Random();
        while (total <= limit) {
            count++;
            int len = r.nextInt(range) + min;
            total += len;
        }
        return count;
    }
```

variable for count of lines and cumulative sum, both initialized to 0, 1 point (missing either -1)
correctly create object of type Random, 1 point (lose if do this inside loop)
correct while loop, 3 points (-1 if < instead of <=, limit >= total okay)
correctly increment counter in loop ,1 point
correctly get random line length, equation for length correct, 4 points

- range = max - min instead of max - min + 1 -> -1
- not adding min to result, -2
- two parameters to nextInt, lose all 4

add current line to running total, 1point
return correct result, 1 point

Other:
Any output, -2
Calling nextInt more than once per number. -4
(can be a significant efficiency hit. min = 90,000, max = 90,020??)

6. Scanners. 17 points. Write a method `aveOfDoubles` that given a `Scanner` already connected to a file, prints out the average of the doubles in each line in the file.

```java
    public static void aveOfDoubles(Scanner sc) {
        int line = 0;
        while (sc.hasNextLine()) {
            line++;
            Scanner lineScanner = new Scanner(sc.nextLine());
            int count = 0;
            double sum = 0.0;
            while (lineScanner.hasNext()) {
                if (lineScanner.hasNextDouble() && !lineScanner.hasNextInt()) {
                    count++;
                    sum += lineScanner.nextDouble();
                } else {
                    lineScanner.next();
                }
            }
            System.out.print(line + ": ");
            if (count == 0) {
                System.out.println("no doubles");
            } else {
                System.out.println("sum = " + sum + ", ave = " + sum / count);
            }
        }
    }
```

line counter, 1 point
outer while loop for hasNextLine(), 1 point
create new Scanner from next line, 1 point
increment line counter correctly, 1 point
variables for sum and number of doubles, 1 point
correct loop for hasNext on line scanner, 3 points
correctly check next token is a double but not an int, 4 points
if next is only a double, read and increment total and count correctly, 2 points
correctly consume next token if not a double, 1 point (-4 if too many calls to next and skips possible double tokens)
output for line correct, 2 points (lose if don't handle case when there are no doubles on a line)

Other:
line counter declared inside instead of outside of loop (all lines are #1), -3
not resetting sum of line and counter for line, - 4
skips lines incorrectly, -5
infinite loop, -7
reads while file as a single line, -8

7. Arrays. 8 points. Write a method named `equalsTarget` that given an array of `int`s and an `int` that represents a target sum, returns `true` if the sum of all the elements in the array equal the target sum, `false` otherwise.

```
public static boolean equalsTarget(int[] data, int tgt) {
    int total = 0;
    for (int i = 0; i < data.length; i++) {
        total += data[i];
    }
    return total == tgt;
}
```

method header correct, 1 point (lose if Boolean instead of boolean, compiles, but use primitive)
variable for total, initialized to 0, 1 point (or subtract from parameter target)
for loop with correct bounds, 3 points (-1 if length() instead of length)
correctly access array elements, 1 point
add elements to running total correctly, 1 point
return correct result, comparing sum to target, 1 point

off by one on number of elements checked -3
off by more than one on number of elements checked -5 total (doesn't sum all elements)