CSE 121 Autumn 2023 Final Exam

December 13, 2023 12:30pm - 2:20pm

Name of Student: _____

Section (e.g., AA):_____ Student **Number** (not UWNetID): _____

Do not turn the page until you are instructed to do so.

Rules/Guidelines:

- You must not begin working before time begins, and you must stop working **promptly** when time is called. Any modifications to your exam (writing *or* erasing) before time begins or after time is called will result in a penalty.
- You are allowed one page of notes, no larger than 8.5 x 11 inches. You may not access any other resources or use any electronic devices (including calculators, phones, or smart watches, among others) during the exam. Using unauthorized resources or devices will result in a penalty.
- In general, you are limited to Java concepts or syntax covered in class. You may not use `break`, `continue`, a `return` from a `void` method, `try/catch`, or Java 8 features.
- You are limited to the standard Java classes and methods listed on the provided reference sheet. You do not need to write import statements.
- If you abandon one answer and write another, ***clearly cross out*** the answer(s) you do not want graded and ***draw a circle or box*** around the answer you do want graded. When in doubt, we will grade the answer that appears in the space indicated, and the first such answer if there is more than one.
- If you require scratch paper, raise your hand and we will bring some to you.
- If you write an answer on scratch paper, please ***write your name and clearly label*** which question you are answering on the scratch paper, and ***clearly indicate*** on the question page that your answer is on scratch paper. Staple all scratch paper you want graded to the ***end*** of the exam before turning in.
- Answers must be written as proper Java code. Pseudocode or comments will not be graded.
- The exam is not graded on code quality. You are not required to include comments.
- You are also allowed to abbreviate "System.out.print" and "System.out.println" as "S.o.p" and "S.o.pln" respectively. You may **NOT** use any other abbreviations.

Grading:

- Each problem will receive a single E/S/N grade.
    - On problems 1 through 3, earning an E requires answering all parts correctly and earning an S requires answering almost all parts correctly.
    - On problems 4 through 6, earning an E requires an implementation that meets all stated requirements and behaves exactly correctly in *all* cases. Earning an S requires an implementation that meets all stated requirements and behaves exactly correctly in *most* cases or behaves nearly correctly in *all* cases.
- Minor syntax errors will be ignored as long as it is unambiguous what was intended (e.g. forgetting a semicolon, misspelling a variable name where there is only one close option). Major syntax errors, or errors where it is unclear what was intended, may have an impact on your grade.

Advice:

- Read all questions carefully. Be sure you understand the question *before* you begin your answer.
- The questions are not necessarily in order of difficulty. Feel free to skip around. Be sure you are able to at least attempt every question.
- Write clearly and legibly. We cannot award credit for answers we cannot read.
- If you have questions, raise your hand to ask. The worst that can happen is we will say "I can't answer that."
- Ask questions as soon as you have them. Do not wait until you have several questions.

Initial here to indicate you have read and agreed to these rules: _____

1. Code Comprehension

Part A: Trace the evaluation of the following expressions, and give their resulting values. Make sure to give a value of the appropriate type. (i.e. Be sure to include a `.0` at the end of a `double` value, or `""` around strings.) Write your answer in the box to the right of each expression.

```
(6 - 4) * 3 + 12 / 5 % 2
```

```
3 * 0.5 + 1 + "7" + 7 + (6 - 2)
```

```
2.5 * 3 >= 10 % 6 || 13 / 4 != 3
```

Part B: Consider the following code:

```
public static int m(int x, int y) {
    while (x > 0 && y > 0) {
        x = x - y;
        y--;
        System.out.print(x + ", ");
    }
    return y;
}
```

Select **all** calls to m that would return an **odd** value.

- ☐ `m(14, 9);`
- ☐ `m(5, 0);`
- ☐ `m(-17, -8);`
- ☐ `m(11, -3);`
- ☐ `m(10, 10);`

Part C: Consider the following code:

```
public static void mystery(int x, int y) {
    int z = 0;
    // Point A
    while (x < y) {
        // Point B
        z++;
        if (z % 2 == 0) {
            x = x * 2;
            // Point C
        } else {
            y--;
            // Point D
        }
    }
    // Point E
    System.out.println(z);
}
```

For each of the statements below, place a check (✓) in the corresponding box if it is true.

- ☐ At Point A, `x < y` must be true.
- ☐ At Point B, `x < y` must be true.
- ☐ At Point C, `z % 2 == 0` must be true.
- ☐ At Point D, `z % 2 == 0` must be true.
- ☐ At Point E, `x < y` must be true.

2. Array Code Tracing

Consider the following method:

```
public static int[] mystery(int[][] list) {
    int[] result = new int[list.length];
    for (int i = 0; i < list.length; i++) {
        int n = 1;
        for (int j = 0; j < list[0].length; j++) {
            n *= list[i][j];
        }
        result[i] = n;
    }
    return result;
}
```

Part A: Consider the following code:

```
int[][] arr = {{3, 15, 1},
               {-8, 1, 7},
               {7, 11, 0},
               {-1, -9, 4}};
int[] result = mystery(arr);
```

What are the contents in **`arr`** after this code is executed?

Part B: Consider the following code:

```
int[][] arr = {{3, 15, 1},
               {-8, 1, 7},
               {7, 11, 0},
               {-1, -9, 4}};
int[] result = mystery(arr);
```

What are the contents in **`result`** after this code is executed?

Part C: Which of the following best describes what the method `mystery` does? (Choose **one**)

- ☐ Returns a new array holding the last element in each row of arr, and modifies arr to contain the products of each row.
- ☐ Returns a new array holding the last element in each column of arr, and modifies arr to contain the products of each column.
- ☐ Returns a new array holding the products of the elements in each row of `arr`, and fills `arr` with 1's.
- ☐ Returns a new array holding the products of the elements in each column of `arr`, and fills `arr` with 1's.
- ☐ Returns a new array containing the products of the elements in each row of arr, leaving `arr` unchanged.
- ☐ Returns a new array containing the products of the elements in each column of arr, leaving `arr` unchanged.

3. Debugging

Consider a static method called **`battle`** that simulates a battle between two players, which takes two parameters:

- `int minDamage` - the minimum amount of damage a player can inflict upon the other (guaranteed to be at least 0)
- `int maxDamage` - the maximum amount of damage a player can inflict upon the other (guaranteed to be greater than `minDamage`)

The "health" of a player is represented by a number initially set to 100. Each player randomly attacks the other, subtracting damage from the attacked player's health, until one of the player's health falls below 1. The next player to attack is **randomly** determined, and the damage inflicted is a **random number** between the minimum damage value and the maximum damage value (inclusive).

For example, suppose the following call was made:

```
battle(20, 50);
```

This call to a *correct* implementation of the method might produce output like the following. (Due to the randomness involved in the method, this exact output may not be produced every time it is run.):

```
Let's get ready to rumble!!!
Player 2 attacks! 35 damage...P1: 65, P2: 100
Player 1 attacks! 37 damage...P1: 65, P2: 63
Player 1 attacks! 43 damage...P1: 65, P2: 20
Player 2 attacks! 21 damage...P1: 44, P2: 20
Player 2 attacks! 43 damage...P1: 1, P2: 20
Player 2 attacks! 32 damage...P1: -31, P2: 20
Player 2 wins!
```

Consider the following proposed buggy implementation of **`battle`**:

```
1  public static void battle(int minDamage, int maxDamage) {
2      System.out.println("Let's get ready to rumble!!!");
3      Random r = new Random();
4      int player = 0;
5      int playerOneHealth = 100;
6      int playerTwoHealth = 100;
7      while (playerOneHealth > 0 || playerTwoHealth > 0) {
8          int damage = r.nextInt(maxDamage - minDamage + 1);
9          player = r.nextInt(2) + 1;
10         if (player == 1) {
11             playerTwoHealth -= damage;
12         } else {
13             playerOneHealth -= damage;
14         }
15         System.out.print("Player " + player + " attacks! " + damage + " damage");
16         System.out.println("P1: " + playerOneHealth + ", P2: " + playerTwoHealth);
17     }
18     System.out.println("Player " + player + " wins!");
19 }
```

This implementation contains two bugs that are causing it to not work as intended!

For the same input as before, the buggy implementation might produce the following output:

```
Let's get ready to rumble!!!
Player 1 attacks! 4 damage...P1: 100, P2: 96
Player 2 attacks! 11 damage...P1: 89, P2: 96
Player 2 attacks! 27 damage...P1: 62, P2: 96
Player 1 attacks! 16 damage...P1: 62, P2: 80
Player 2 attacks! 22 damage...P1: 40, P2: 80
Player 2 attacks! 24 damage...P1: 16, P2: 80
Player 1 attacks! 17 damage...P1: 16, P2: 63
Player 2 attacks! 23 damage...P1: -7, P2: 63
Player 1 attacks! 27 damage...P1: -7, P2: 36
Player 1 attacks! 18 damage...P1: -7, P2: 18
Player 2 attacks! 23 damage...P1: -30, P2: 18
Player 1 attacks! 19 damage...P1: -30, P2: -1
Player 1 wins!
```

Your task: Annotate (write on) the code below to indicate how you would fix the two bugs. You may add (using arrows to indicate where to insert), remove (by crossing out), or modify (with a combination) any code you choose. However, the fix should not require a lot of work.

You must *correctly identify* both of the lines with issues, or *correctly identify and fix* **one** of the bugs for an S grade.
You must *correctly identify* both of the lines with the bugs **and** *correctly fix* both of the bugs for an E grade.

```
1  public static void battle(int minDamage, int maxDamage) {
2      System.out.println("Let's get ready to rumble!!!");
3      Random r = new Random();
4      int player = 0;
5      int playerOneHealth = 100;
6      int playerTwoHealth = 100;
7      while (playerOneHealth > 0 || playerTwoHealth > 0) {
8          int damage = r.nextInt(maxDamage - minDamage + 1);
9          player = r.nextInt(2) + 1;
10         if (player == 1) {
11             playerTwoHealth -= damage;
12         } else {
13             playerOneHealth -= damage;
14         }
15         System.out.print("Player " + player + " attacks! " + damage + " damage");
16         System.out.println("P1: " + playerOneHealth + ", P2: " + playerTwoHealth);
17     }
18     System.out.println("Player " + player + " wins!");
19 }
```

4. General Programming 1

Write a static method called **`longWords`** that takes two parameters: a `Scanner` `input`, and an integer, `numWords`. The method should prompt the user for `numWords` words using the given `Scanner`. After each word, if there are more words to be entered, the number of words remaining should be shown. The method should then print the longest word entered and return the total number of characters in all words entered. If two more words are tied for the longest word, the method should report the one entered earliest.

For example, assuming the following declaration is made, the table below shows some sample calls to `longWords` and resulting output (user input **bold and underlined**):

```
Scanner console = new Scanner(System.in);
```

Call	`longWords(console, 5);`	`longWords(console, 9);`	`longWords(console, 1);`

Output	`Next word? apple`
	`4 more words...`
	`Next word? blueberry`
	`3 more words...`
	`Next word? watermelon`
	`2 more words...`
	`Next word? pear`
	`1 more words...`
	`Next word? grape`
	`Longest word: watermelon`
		`Next word? alpha`
		`8 more words...`
		`Next word? bravo`
		`7 more words...`
		`Next word? charlie`
		`6 more words...`
		`Next word? delta`
		`5 more words...`
		`Next word? echo`
		`4 more words...`
		`Next word? foxtrot`
		`3 more words...`
		`Next word? golf`
		`2 more words...`
		`Next word? hotel`
		`1 more words...`
		`Next word? india`
		`Longest word: charlie`
			`Next word? elementary`
			`Longest word: elementary`

Return	`33`	`47`	`10`

Notice that we don't print out the "`_ more words…`" message after the last word is entered. You must exactly reproduce the format of this sample execution.

You may assume that `numWords` is always greater than 0, and that the user enters a single word with at least one character each time they are prompted. You must exactly reproduce the format of these logs.

Write your solution in the box on the next page.

Write your solution to problem #4 here:

5. General Programming 2

Write a static method named **`gumballTricks`** that accepts a `Random` object and an integer n that represents the number of tricks as parameters. Your method should use the `Random` object to randomly choose a trick from Gumball's repertoire: spin, bang!, and boop. Each outcome should be equally likely. Your method should print out each of the randomly-generated tricks followed by a space, and then both print and return the greatest number of spins that occurred in a row.

Assuming that the following variable has been initialized:

```
Random r = new Random();
```

Here are some example calls to the method with their resulting console output and return value:

Call	Console Output	Returned
`gumballTricks(r, 8);`	`spin boop bang! spin spin spin bang! spin` `Run of 3 spins after 8 tricks.`	3
`gumballTricks(r, 5);`	`bang! boop boop bang! boop` `Run of 0 spins after 5 tricks.`	0
`gumballTricks(r, 2);`	`spin bang!` `Run of 1 spins after 2 tricks.`	1
`gumballTricks(r, 10);`	`bang! spin spin boop spin spin spin spin bang! boop` `Run of 4 spins after 10 tricks.`	4
`gumballTricks(r, 3);`	`bang! bang! boop` `Run of 0 spins after 3 tricks.`	0
`gumballTricks(r, 1);`	`bang!` `Run of 0 spins after 1 tricks.`	0

Note that if there is only one trick, your final line should end with "1 tricks" (<u>not</u> 1 trick).

You must exactly reproduce the format of the console output shown above, though the actual output may differ due to randomness. It is okay for the line of tricks to end with a space. You may assume that the integer passed as a parameter to your method is greater than 0.

You may not construct any extra data structures (e.g. arrays, `ArrayLists`) to solve this problem.

Write your solution to problem #5 here:

6. Array Programming

Write a static method named **`weave`** that accepts two arrays of integers as a parameter and that returns a new array that is the result of alternating the values from the two arrays, starting with the first value of the first array. For example, if variables named a1 and a2 store the following values:

```
int[] a1 = {1, 2, 3};
int[] a2 = {4, 5, 6};
```

then the call of `weave(a1, a2)` would return a new array containing the following values:

```
[1, 4, 2, 5, 3, 6]
```

It is possible that the two arrays may have different lengths, in which case after running out of values from the shorter array, the remaining slots of the result array are filled with the leftover elements of the longer array. For example, if variables named a1 and a2 store the following values:

```
int[] a1 = {1, 2, 3, 4, 5, 6};
int[] a2 = {7, 8, 9};
```

then the call of `weave(a1, a2)` would return a new array containing the following values:

```
[1, 7, 2, 8, 3, 9, 4, 5, 6]
```

You are not permitted to create any additional data structures (e.g. arrays, `ArrayLists`, `Strings`) other than the result array that you return.

Write your solution to problem #6 here:

(Extra fun!) *Draw a picture of what you think your TA will be doing over Winter break! We'll give these drawings to your TA as thanks for all their hard work this quarter!*

Here's a picture of Gumball for inspiration!