CSE 121 Winter 2024 Final Exam (Alternate) April 5, 2024

Name: _____

Section: _____ Student **Number** (not UWNetID): _____

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. However, 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 agree to these rules:

Name: _____

1. Code Comprehension

(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 `String`s.) Write your answer in the line to the right of each expression.

i. `8 / 5 + 3 * (1 / 2) - 9 % 5`

Solution: `-3`

ii. `4 * 2 + "1 + 3" + "e -" + 8 / 3`

Solution: `"81 + 3e -2"`

iii. `14 / 4 > 3.0 || !(4 - 3.0 == 1)`

Solution: `false`

(b) Consider the following code:

```
public static void m(int x, int y) {
    while (x > 1) {
        System.out.println(x + "," + y);
        if (x % y == 0) {
            x = x / y;
        } else {
            x = x + 1;
        }
    }
}
```

Select **all** calls to m that would **print out four or more non-empty lines**.

- ☐ m(1, 1);
- ■ m(5, 2);
- ☐ m(3, 4);
- ☐ m(3, 5);
- ☐ m(0, 6);

Name: _____

(c) Consider the following code:

```
public static void mystery(int x, int a) {
    if (x < 1) {
        x = 1;
    }

    int y = x;
    // Point A

    while (x < a) {
        // Point B
        int temp = y;
        y = y + x;
        // Point C
        x = temp;
        // Point D
    }

    // Point E
    System.out.println(x);
}
```

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

- ☐ At Point A, both `x > a` and `y > a` must be true.
- ☐ At Point B, `x < y` must be true.
- ■ At Point C, `x < y` must be true.
- ■ At Point D, `x < y` must be true.
- ☐ At Point E, both `x > a` and `y > a` must be true.

Name: _____

2. Array Code Tracing

Consider the following method:

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

(a) Consider the following code:

```
int[] arr = {2, 4, 6};
int[][] result = mystery(arr);
```

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

Solution: `[-2, -12, -30]`

(b) Consider the following code:

```
int[] arr = {2, 4, 6};
int[][] result = mystery(arr);
```

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

Solution: `[[0, 8, 12], [8, 0, 24], [12, 24, 0]]`

(c) Which <u>two</u> statements are true about the `mystery` method?

- ☐ Each `int` element of the returned 2D array will <u>always</u> be nonnegative (i.e. has a value greater than or equal to zero).
- ■ There will always be <u>at least</u> `list.length` items in the returned 2D array with a value of exactly `0`.
- ☐ Depending on the input, it is possible for `result.length` to be different from `list.length`.
- ☐ Because we don't use a nested for loop to traverse a 2D array, it is possible for us to have an array index out of bounds error at `Point X`.
- ■ If the parameter `list` is empty (i.e., has zero items), then the returned 2D array will also be empty.

Name: _____

3. Debugging

   Consider a static method called `seasonInformation` that prompts the user for month and day, and then uses some logic to determine and return the current season as a string.

   In particular, the `seasonInformation` method is passed one parameter: a `Scanner` object (which we'll call `console`). The method uses `console` to prompt the user for the current month and day.

   To determine the season, there is a specific set of steps that should determine the returned string:

   1. The user is prompted for the current month, as a number from 1 to 12.
   2. If the input is invalid (i.e., a number that is not between 1 and 12 inclusive), the string `"Invalid"` should be returned.
   3. For some inputs, we can determine the season without needing to ask for any more information. For each of these inputs, the method should immediately return the corresponding string, without prompting the user for any more input. In particular:
      - for months 1 and 2 (Jan and Feb), return `"Winter"`
      - for months 4 and 5 (Apr and May), return `"Spring"`
      - for months 7 and 8 (Jul and Aug), return `"Summer"`
      - for months 10 and 11 (Oct and Nov), return `"Fall"`
   4. For the other months (3, 6, 9, 12), we need to ask the user for the date. In particular,
      - for month 3 (Mar), return `"Winter"` if it is before the 21st and `"Spring"` otherwise
      - for month 6 (Jun), return `"Spring"` if it is before the 21st and `"Summer"` otherwise
      - for month 9 (Sep), return `"Summer"` if it is before the 21st and `"Fall"` otherwise
      - for month 12 (Dec), return `"Fall"` if it is before the 21st and `"Winter"` otherwise

   You may assume that the provided `Scanner` is valid, and that the user will always provide numbers as input. In addition, you may assume that the user never inputs a day that does not exist (e.g. March 32nd or December -1st).

   Here are the expected results of three different example calls to `seasonInformation` (with a valid Scanner); **(User input is in bold and underlined):**

   Example #1: "simpler" case.

   ```
   Please enter the month (1-12):  2
   ```

   The method would return the season string `"Winter"`.

   Example #2: "complicated" case.

   ```
   Please enter the month (1-12):  6
   Please enter the day (1-31):  15
   ```

   The method would return the season string `"Spring"`.

   Example #3: invalid input.

   ```
   Please enter the month (1-12):  -42
   ```

   The method would return the season string `"Invalid"`.

Name: _____

Consider the following proposed buggy implementation of `seasonInformation`. This implementation contains three bugs that are causing it to not work as intended!

Your task: Annotate (write on) the code below to indicate how you would fix the three bugs. You may add (using arrows to indicate where to insert), remove (by crossing out), or modify (with a combination) any code you choose. Each fix is "local" (i.e. it should not require significant work).

- You must correctly identify three of the lines with issues, or correctly identify and fix two of the bugs for an S grade.
- You must correctly identify all three lines with the bugs and correctly fix all three of the bugs for an E grade.

```
public static String seasonInformation(Scanner console) {
    System.out.print("Please enter the month (1-12): ");
    int month = console.nextInt();

    if (month < 1 && month > 12) { // Bug #1, should be ||, not &&
        return "Invalid";
    }

    if (month == 1 || month == 2) {
        return "Winter";
    } else if (month == 4 || month == 4) { // Bug #2, should be
                                           // month == 4 || month == 5
        return "Spring";
    } else if (month == 7 || month == 8) {
        return "Summer";
    } else if (month == 10 || month == 11) {
        return "Fall";
    }

    System.out.print("Please enter the day (1-31): ");
    int day = console.next(); // Bug #3, should be console.nextInt();

    if ((month == 12 && day >= 21) || (month == 3 && day < 21)) {
        return "Winter";
    } else if ((month == 3 && day >= 21) || (month == 6 && day < 21)) {
        return "Spring";
    } else if ((month == 6 && day >= 21) || (month == 9 && day < 21)) {
        return "Summer";
    } else if (month == 9 && day >= 21 || (month == 12 && day < 21)) {
        return "Fall";
    }

    return "Invalid";
}
```

Name: _____

4. General Programming 1

Write a static method called `findChars` that takes two parameters: a `Scanner input` and an String `targetChars`. The method should prompt the user for words using the given `Scanner` until the user inputs a word that contains **all** of the unique characters in `targetChars` (in any order). Finally, the method should return the total number of characters the user has entered.

For example, assume the following variable is declared:

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

The following table shows some sample calls to `findChars` and resulting output (user input is **bold and underlined**):

Call	`findChars(con, "abc");`	`findChars(con, "mat");`	`findChars(con, "ELBA");`
Output	`Next word:` **I** `Next word:` **wanna** `Next word:` **get** `Next word:` **him** `Next word:` **back!**	`Next word:` **Cause** `Next word:` **we** `Next word:` **are** `Next word:` **living** `Next word:` **in** `Next word:` **a** `Next word:` **material**	`Next word:` **And** `Next word:` **at** `Next word:` **every** `Next word:` **table** `Next word:` **oops** `Next word:` **TABLE!**
Return	17	27	25

As shown in the above examples, your program should be able to handle lowercase and uppercase alphabet characters, numbers, and punctuation. Your comparisons should be **case-sensitive** (as shown in the `ELBA` example).

You may assume that `targetChars` always has at least one character. You may also assume that `targetChars` has **no duplicate characters**.

Outside of whitespace, you should aim to exactly reproduce the output format above. As a reminder, you are restricted to the methods provided on the reference sheet.

Name: _____

Write your solution to problem #4 here:

Solution:

```
public static int findChars(Scanner console, String targetChars) {
    int charCount = 0;
    boolean foundAllChars = false;;
    while(!foundAllChars) {
        System.out.print("Next word: ");
        String word = console.next();
        charCount += word.length();

        int charsFound = 0;

        for (int i = 0; i < targetChars.length(); i++) {
            // alternatively, targetChars.charAt(i) + ""
            String currentChar = targetChars.substring(i, i+1);
            if (word.contains(currentChar)) {
                charsFound++;
            }
        }

        if (charsFound == targetChars.length()) {
            foundAllChars = true;
        }
    }
    return charCount;
}
```

Name: _____

5. General Programming 2

After their Lottery Dreams (on Quiz 2) failed, Matt and Elba have one more Hail Mary plan to make money: betting on coin flips! But they need help with the math – that's where you come in!

Write a static method named `howTheFlip` that accepts a `Random` object and an integer (which we'll call `numMoreTails`) and simulates coin flips (using the `Random` object) until you flip `numMoreTails` **more** tails than heads. In other words, you should stop only when the number of tails you've seen is exactly `numMoreTails` more than the number of heads you've seen.

There are some specific notes about how this method should work:

- your coin flip should be "unbiased"; in other words, there should be a 50% chance of getting heads and a 50% chance of getting tails on each flip.
- you **may not assume that `numMoreTails` is positive**. If the user passes in a value that is less than 1, your method should not flip any coins. Instead, print out an "Invalid" message and return 0.

Assuming that the following variable has been initialized:

```
Random randy = new Random();
```

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

Call	Console Output	Returned
`howTheFlip(randy, 1)`	`Flips: H, T, T`	3
`howTheFlip(randy, 2)`	`Flips: T, H, T, H, T, T`	6
`howTheFlip(randy, 2)`	`Flips: T, H, H, T, T, H, T, H, T, T`	10
`howTheFlip(randy, 0)`	`Invalid numMoreTails: 0`	0
`howTheFlip(randy, -2)`	`Invalid numMoreTails: -4`	0

You must exactly reproduce the format of the console output shown above (other than whitespace), though the actual output may differ due to randomness.

You are not permitted to create any additional data structures (e.g. arrays, `ArrayList`s) to solve this problem. However, new variables for primitive data types are allowed. In addition, you should not declare a new `Random` object within your method; you must use the one passed in by the user.

Name: _____

Write your solution to problem #5 here:

Solution:

```
public static int howTheFlip(Random randy, int numMoreTails) {
    if (numMoreTails < 1) {
        System.out.println("Invalid numMoreTails: " + numMoreTails);
        return 0;
    }

    System.out.print("Flips: ");

    int tailsBalance = 0;
    int totalFlips = 0;

    // pulling out one flip - fencepost
    int flip = randy.nextInt(2);
    totalFlips++;

    if (flip == 1) {
        tailsBalance--;
        System.out.print("H");
    } else {
        tailsBalance++;
        System.out.print("T");
    }

    while (tailsBalance != numMoreTails) {
        System.out.print(", ");

        flip = randy.nextInt(2);
        totalFlips++;

        if (flip == 1) {
            tailsBalance--;
            System.out.print("H");
        } else {
            tailsBalance++;
            System.out.print("T");
        }
    }
    return totalFlips;
}
```

Name: _____

6. Array Programming

Write a static method named `repeatWords` that accepts two parameters: an array of strings (which we'll call `words`), and an array of integers (which we'll call `repetitions`). This method should not return anything; instead, it should modify `words` by "repeating" each of its entries for the corresponding number of times.

The simple case is when `words` and `repetitions` are the same length. This behavior is best explained with an example. Assume that the following variables have been declared:

```
String[] words = {"Olivia", "Rodrigo", "GUTS "};
int[] repetitions  = {1, 2, 4};
```

Calling `repeatWords(words, repetitions)` would return nothing and leave `repetitions` unchanged. However, printing `words` would yield:

```
["Olivia", "RodrigoRodrigo", "GUTS GUTS GUTS GUTS "]
```

To break this example down:

1. for the first item in `words`, we repeat it `1` time (the corresponding value in `repetitions`), giving us the original string: `"Olivia"`
2. for the second item in `words`, we repeat it `2` times (the corresponding value in `repetitions`), converting `"Rodrigo"` to `"RodrigoRodrigo"`
3. for the third item in `words`, we repeat it `4` times (the corresponding value in `repetitions`), converting `"GUTS "` to `"GUTS GUTS GUTS GUTS "`;

When `words` and `repetitions` are not the same length, you should ignore the "extra" items in the larger array. In addition, if a value in `repetitions` is nonpositive (i.e. less than or equal to 0), you should replace the corresponding entry in "words" with the empty string (i.e. `""`). These rules apply if one (or both) of the arrays is empty. Here are two examples that showcase this behaviour:

```
String[] words1 = {"Laufey", "Bewitched"};
String[] words2 = {"Laufey", "Bewitched"};
int[] repetitions1  = {3};
int[] repetitions2  = {2, -1, 4};
```

After calling `repeatWords(words1, repetitions1)`, printing `words1` would yield:

```
["LaufeyLaufeyLaufey", "Bewitched"]
```

After calling `repeatWords(words2, repetitions2)`, printing `words2` would yield:

```
["LaufeyLaufey", ""]
```

You are not permitted to create any additional data structures (e.g. arrays, `ArrayList`s) to solve this problem. However, new variables for primitive data types are allowed. As a reminder, you are restricted to the methods provided on the reference sheet.

Name: _____

Write your solution to problem #6 here:

Solution:

```
public static void repeatWords(String[] words, int[] repetitions) {
    // this properly deals with if the two lengths are not equal
    // if words.length > repetitions.length, skip the remaining words
    // if reptitions.length > words.length, skip the remaining
   repetitions
    int wordsToRepeat = Math.min(words.length, repetitions.length);

    for (int i = 0; i < wordsToRepeat; i++) {
        // temp variable: necessary since we "reset" words[i]
        String word = words[i];
        words[i] = "";

        // if repetitions[i] is <= 0, this will never run
        // so, words[i] will remain "" - as desired
        for (int j = 0; j < repetitions[i]; j++) {
            words[i] += word;
        }
    }
}
```

Name: _____

Just for fun: As a thanks for all their work, draw your TA a picture of what you think they will be up to during Spring Quarter! (not graded, not mandatory!)

TA Name: _____

Solution:

This page is intentionally left blank and will not be graded; do not put exam answers here.