CSE 121 Spring 2024 Final Exam June 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:

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 box to the right of each expression.

i. `3 * (4 % 2) * 1.0`

Solution: `0.0`

ii. `8 - 5.0 * 2 + ("2" + 5)`

Solution: `"-2.025"`

iii. `(15 % 5 == 3) || (!(5.0 - 4 == 1.0))`

Solution: `false`

(b) Select **all** choices whose statements are true.

☐ The `for` loop header `for (int i = 0; i < arr.length; i++)` would give an error if the array `arr` was empty (i.e. had zero elements).

■ Calling `nextInt(1)` on a `Random` object always gives you `0`.

■ The names of types (e.g. `boolean`) in Java are case-sensitive.

☐ `Scanner` can only be used with `while` loops, not `for` loops.

☐ Using == with Strings causes a compilation error.

Explanations:

1. this `for` loop wouldn't error (try it!)
2. `nextInt(1)` gives you a number between 0 and $1 - 1 = 0$, so ... 0
3. like all identifiers and keywords, Java is generally case-sensitive! try `string`!
4. not at all - see our weather example from lecture!
5. this is tricky, but == with Strings does not give a compilation or runtime error. However, it may not give you the result that you want (which is why it can still cause bugs).

Name: ______________________________

(c) Consider the following code:

```
public static int mystery(int n) {
    if (n <= 1) {
        return 0;
    }

    // Point A
    int i = 0;
    while (n > 1) {
        if (n % 2 == 0) {
            n = n / 2;
            // Point B
        } else {
            n = 3 * n + 1;
            // Point C
        }
        // Point D
        i++;
    }
    // Point E
    return i;
}
```

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, `n <= 1` could be `true`.
- ■ At Point B, `n <= 1` could be `true`.
- ☐ At Point C, `n <= 1` could be `true`.
- ■ At Point D, `n <= 1` could be `true`.
- ☐ At Point E, `n <= 1` could be `false`.

Explanations:

1. At Point A, it is not possible for `n <= 1`; if it was, we would have `return 0;`'d already.
2. At Point B, it is possible for `n <= 1` because we've just divided n by 2. For example, try tracing `mystery(2)`.
3. At Point C, it is not possible for `n <= 1`, since we would have only entered the `while` loop if `n > 1` (and we have not changed `n` - unlike Point B)
4. At Point D, it is possible for `n <= 1` - see Point B.
5. At Point E, it must be true that `n <= 1`, as we would only get here if the `while` loop was exited (i.e., `n > 1` is `false`).

Fun fact: this problem is based on something called the "Collatz conjecture".

Name: ______________________________

2. Array Code Tracing

Consider the following method:

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

(a) Consider the following code:

```
int[][] arr = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
int[] result = mystery(arr);
```

What are the contents in `arr` after this code is executed? (please try to format the array in "quick initialization" syntax, i.e. what is after `int[][] arr =` in the example)

Solution: `{{0, 1, 2}, {3, 4, 5}, {6, 7, 8}}`

(b) Consider the following code:

```
int[][] arr = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
int[] result = mystery(arr);
```

What are the contents in `result` after this code is executed? (please aim for clear syntax)

Solution: `{0, 1, 1}`

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

- [x] Depending on `arr`, it is possible to get an index out of bounds error at `Point X`.
- [x] Depending on `arr`, it is possible to get an index out of bounds error at `Point Y`.
- [ ] Depending on `arr`, it is possible to get an index out of bounds error at `Point Z`.
- [ ] If we declared `int[][] arr = {}`, calling `mystery(arr)` will cause an error.
- [ ] It is possible that `arr` and the returned array will have different lengths.

Name: ______________________________

Explanations:

1. This is possible if `arr` is not "square", in the case where it has more rows than columns. For example, try running this code with `int[][] arr = {{1}, {2}, {3}}`.
2. This is possible if `arr` is not "square", in the case where it has more columns than rows. For example, try running this code with `int[][] arr = {1, 2, 3}`. This is tricky because the error comes from `result[j]`, not `arr[i][j]`!
3. We properly check our bounds for both `i` and `j` for `arr[i][j]`!
4. The method would just return an empty array, since none of the `for` loops would run. Give it a spin!
5. Note that `result`'s length is always `arr.length` — they are the same by definition!

3. Debugging

Matt's been talking about Laufey so much ... but is he actually a real fan? Consider a static method called `spotifyRapped` that prompts the user for listening data, and then tells them if they are a "true fan" (using some data).

In particular, the `spotifyRapped` method takes in three parameters:

- a `Scanner` to get user input
- a `String` representing an artist
- an `int` array representing "listening data" for that artist, where each element is the number of minutes a specific person has spent listening to that artist (not including the user)

The method should not return anything. Instead, it should:

1. prompt the user for the number of minutes they've spent listening to the artist
2. count the number of people who have listened to the artist for less time than the user (this is how many people the user has "beat"), then calculate what percentage of the total number of people this is.
3. depending on that percentage, call them a "true fan", "bandwagoner", or "casual listener":
   - if the user beat greater than or equal to 80% of people, they are a "true fan"
   - if the user beat less than 40% of people, they are a "casual listener"
   - otherwise, they are a "bandwagoner"

You may assume that the user always provides valid input (i.e. a nonnegative `int`) to the `Scanner`. You may also assume that the array parameter has at least one element.

Here are results of two different example calls to a correct implementation of `spotifyRapped`. Assume that `console` is a valid `Scanner`.

Example #1: "true fan"

```
int[] listens = {842, 523, 2, 1482, 23, 78};
spotifyRapped(console, "Laufey", listens)
```

would print (with **user input in bold and underlined**):

```
How many minutes have you listened to Laufey?  6726
You beat 6/6 other people; you're a true fan!
```

Example #2: "bandwagoner"

```
int[] listens = {370, 76, 0, 37, 0, 6557, 39, 0, 0, 234};
spotifyRapped(console, "Reneé Rapp", listens)
```

would print (with **user input in bold and underlined**):

```
How many minutes have you listened to Reneé Rapp?  100
You beat 7/10 other people; you're a bandwagoner!
```

Example #3: "casual listener"

```
int[] listens = {2346, 647, 0, 128, 7346};
spotifyRapped(console, "Chappell Roan", listens)
```

would print (with **user input in bold and underlined**):

```
How many minutes have you listened to Chappell Roan?  128
You beat 1/5 other people; you're a casual listener!
```

Name: ______________________________

Consider the following proposed buggy implementation of `spotifyRapped`. 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.

```java
public static void spotifyRapped(Scanner console,
                                 String artist, int[] listens) {

    System.out.print("How many minutes have you listened to "
                          + artist + "?");

    // Bug #1: this should be nextInt(), not next()
    int minutesListened = console.nextInt();

    int numBeat = 0;

    for (int i = 0; i < listens.length; i++) {
        if (minutesListened > listens[i]) {
            numBeat++;
        }
    }
    // Bug #2: (numBeat / listens.length) will always be zero because
    //         of integer division. Many ways to fix!
    // note: the actual final had "numPeopleBeat", but that was a typo
    //        and was clarified during the final
    double beatPercentage = 100.0 * numBeat / listens.length;

    // Alternate fixes:     100.0 * (1.0 * numBeat / listens.length);
    //                      (100.0 * numBeat) / listens.length;
    //                      100.0 * (0.0 + numBeat) / listens.length

    String fracAsString = numBeat + "/" + listens.length;
    System.out.print("You beat " + fracAsString + " other people; ");

    if (beatPercentage >= 80.0) {
        System.out.println("you're a true fan!");
    }
    // Bug #3: this "if" should be an "else if" - previously,
    //         it printed bandwagoner for true fans too!
    else if (beatPercentage < 40.0) {
        System.out.println("you're a casual listener!");
    }
    else {
        System.out.println("you're a bandwagoner!");
    }

}
```

Name: ______________________________

4. General Programming 1

Write a static method called `worseWordle` that uses two parameters (a `Scanner` and a String `word`) to play a "worse" version of the hit game Wordle. It should use the `Scanner` to continually prompt the user for words until they properly guess the word, and finally <u>return</u> the number of guesses the player made (including the winning guess) as an `int`.

No knowledge of the "real" Wordle is necessary. The rules of `worseWordle` are as follows:

- for the first turn, we print `word.length()` underscores (`'_'`) and ask the user to guess
- until their guess is correct, we:
  - print out the correctness of their last guess: for each character in their guess, if `word` has the same character in that position, print the character; otherwise, print an underscore
  - ask them to guess again
- once the user correctly guesses the word, we print out the `word` and a `"You Won!"` message

This is best explained with an example. Assume the following variable is declared:

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

Then, calling `worseWordle(console, "LAUFEY")` would output the following
(user input is **<u>bold and underlined</u>**):

```
______ Guess: EILISH
______ Guess: WETLEG
____E_ Guess: MITSKI
______ Guess: LANADR
LA____ Guess: LAUFEY
LAUFEY You won!
```

And, the method call would return the value `5` (as we made `5` total guesses).

To break down this example:

1. before the first guess, we print six underscores (as `"LAUFEY"` has six characters)
2. the user's first guess (`"EILISH"`) has no characters in the same position with `"LAUFEY"`.
3. the user's second guess (`"WETLEG"`) has one character in the same position as `"LAUFEY"`: the `'E'` at index `4`. So, we print that `E`, <u>only for that turn.</u>
4. the user's third guess (`"MITSKI"`) has no characters in the same position with `"LAUFEY"`.
5. the user's fourth guess (`"LANADR"`) shares two characters in the same position as `"LAUFEY"`: the `'L'` at index `0` and the `'A'` at index `1`, so we print them both.
6. the user's fifth guess is correct, so we tell the user they won! (and return `5`)

You may assume that:

- both `word` and the user's guesses have at least one character (i.e. not empty or `null`)
- both `word` and the user's guesses only consist of uppercase English alphabet characters
- the user's guess will always be the same length as the `word`

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, and you do not need to write a main method or class declaration - just the method itself.

Name: ______________________

Write your solution to problem #4 here:

Solution:

```java
public static int worseWordle(Scanner console, String word) {
    // Overall approach: fenceposting with while, where we ask
    // *while* their current guess is wrong. (or until it's correct)


    // printing out the first set of underscores
    for (int i = 0; i < word.length(); i++) {
        System.out.print("_");
    }

    // fenceposting the first guess!
    int guesses = 1; // starting at 1 since they'll always have to guess
    System.out.print(" Guess: ");
    String guess = console.next();

    while (!guess.equals(word)) {
        // prints out either _ or the character, depending on if it matches
        for (int i = 0; i < word.length(); i++) {
            if (word.charAt(i) == guess.charAt(i)) {
                System.out.print(word.charAt(i));
            } else {
                System.out.print("_");
            }
        }

        // this is now at the end, since we fenceposted
        guesses++;
        System.out.print(" Guess: ");
        guess = console.next();
    }

    System.out.println(word + " You won!");

    return guesses;
}
```

Name: ____________________

5. General Programming 2

Write a method called `dateNight` that takes in a `Random` object as a parameter and returns a String that represents a random day, month, and year (in a particular format).

The logic for the method is as follows:

- the `year` can be any number between `2000` and `2099` (inclusive)
- the `month` can be any number between `1` and `12` (inclusive), where `1` represents January, `2` represents February, . . .
- the `day` has some rules. It should also start at 1, but:
  - some months always have `31` days: January (`1`), March (`3`), May (`5`), July (`7`), August (`8`), October (`10`), and December (`12`).
  - some months always have `30` days: April (`4`), June (`6`), September (`9`), and November (`11`).
  - February (2) normally has 28 days. However, during a leap year — for the purposes of this problem, years that are evenly divisible by 4 — February instead has 29 days.

While this method relies on randomness, its randomness should follow two properties:

1. any year and/or month should be equally likely
2. given a year and month, any valid day should be equally likely

Your method should return a random date in the format `year-month-day`. Unlike most calendars, you do not need to add a leading 0 (i.e. April 1st, 2024 is `2024-4-1`, not `2024-04-01`).

Assume that we've declared a `Random` object called `randy`. Here are some examples of valid return values from `dateNight(randy)`:

- `"2000-1-1"` (the earliest valid date) and `"2099-12-31"` (the latest valid date)
- `"2023-3-31"` (March has 31 days) and `"2019-11-30"` (November has 30 days)
- `"2024-2-29"` (2024 (this year!) is a leap year — as it's evenly divisible by 4 — so February has 29 days)
- `"2024-6-5"` (today's date!) and `"2024-10-31"` (Halloween, spooky!)

In contrast, here are some invalid dates:

- `"1999-12-31"` (too early) and `"2100-1-1"` (too late)
- `"2024-4-31"` (April only has 30 days) and `"2024-8-32"` (no month has 32 days)
- `"2023-2-29"` and `"2026-2-29"` (neither 2023 nor 2026 is a leap year — as neither is evenly divisible by 4 — so February only has 28 days)

Your return value should exactly match the format described above (though the actual output would differ due to randomness). You should not declare a new `Random` object within your method; you must use the one passed in by the user.

As a reminder, you are restricted to the methods provided on the reference sheet, and you do not need to write a main method or class declaration - just the method itself.

Write your solution to problem #5 here:

Solution:

```java
public static String dateNight(Random randy) {
    // Overall approach: pick year & month first, then day

    int year = randy.nextInt(100) + 2000; // this is [2000, 2099]
    int month = randy.nextInt(12) + 1;    // this is [1, 12]

    int day = 1; // starting at 1 as the "first day";
                 // could also start at zero, if we
                 // remember to add 1 later

    if (month == 2) {
        // leap year logic for february
        if (year % 4 == 0) {
            day += randy.nextInt(29);
        } else {
            day += randy.nextInt(28);
        }
    } else if (
        month == 4 || month == 6 || month == 9 || month == 11
    ) {
        // April, June, September, November - max day is 30
        day += randy.nextInt(30);
    } else {
        // otherwise, in a 31-day month
        day += randy.nextInt(31);
    }

    return year + "-" + month + "-" + day;
}
```

Name: ______________________________

6. Array Programming

   Write a static method named `meanSquaredError` that accepts two parameters, both arrays of `doubles` (which we'll call `prediction` and `actual`). It should calculate and return the "mean squared error" of the two arrays as a `double`.

   (as an aside: the mean squared error, or MSE, is a foundational tool in modern statistics!)

   For the purposes of this problem, we define the mean squared error in two pieces:

   - the **error** at an index `i` is the square of the difference between `prediction[i]` and `actual[i]`
   - the **mean squared error** of two arrays is the mean (i.e. average) of the errors for each individual pair of elements

   Let's walk through an example. Assume the following variables have been declared:

```
double[] prediction = {1.0, 5.0, 3.0};
double[] actual = {-1.0, 4.0, 4.0};
```

   Calling `meanSquaredError(prediction, actual)` would return the value `2.0`. This is because:

   - there are three elements; their individual errors are
     1. element 0: $(1.0 - (-1.0))^2 = 2.0^2 = 4.0$
     2. element 1: $(5.0 - 4.0)^2 = 1.0^2 = 1.0$
     3. element 2: $(3.0 - 4.0)^2 = (-1.0)^2 = 1.0$
   - thus, the **mean squared error** is $(4.0 + 1.0 + 1.0)/3 = 6.0/3 = 2.0$

   If `actual` has more elements than `prediction`, your method should use `0.0` as the `prediction`'s value. For example, given the following variables:

```
double[] prediction = {17.0};
double[] actual = {17.0, 6.0, 3.0};
```

   Calling `meanSquaredError(prediction, actual)` would return the value `15.0`. This is because:

   - the errors are: $(17.0 - 17.0)^2 = 0.0$, $(0.0 - 6.0)^2 = 36.0$, and $(0.0 - 3.0)^2 = 9.0$
   - thus, the **mean squared error** is $(0.0 + 36.0 + 9.0)/3 = 45.0/3 = 15.0$

   In addition, your method must deal with two cases:

   1. if `prediction` and `actual` are both empty, your method should return `0.0`
   2. if `prediction` has more elements than `actual`, your method should return `-1.0`

   Your method does not have to deal with the case where `prediction`, `actual`, or both are `null`.

   Your method should not modify the elements of `prediction` or `actual` in any way.

   You are not permitted to create any additional data structures (e.g. arrays, 2D arrays, `ArrayLists`) 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, and you do not need to write a main method or class declaration - just the method itself.

Name: ______________________________

Write your solution to problem #6 here:

Solution:

```java
public static double meanSquaredError(double[] prediction, double[]
    actual) {
     // Overall approach: check for edge cases first.
     // If not, iterate through arrays pairwise,
     // with logic to check if prediction is missing

     if (prediction.length == 0 && actual.length == 0) {
         return 0.0; // see spec
     }

     if (prediction.length > actual.length) {
         return -1.0; // see spec
     }

     double total = 0.0;

     for (int i = 0; i < actual.length; i++) {
         if (i >= prediction.length) {
             // "missing" prediction. so, just treat it as zero;
             // note that actual[i] - 0 is just actual[i]
             total += actual[i] * actual[i];
         } else {
             double diff = prediction[i] - actual[i];
             total += diff * diff; // no need to use Math.pow, etc.
         }
     }

     // note: actual.length can't be 0 here, since we check
     //        for the two edge cases above
     return total / actual.length;
}
```

Name: ______________________________

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 their Summer break! (not graded, not mandatory!)

TA Name: ____________________

Solution:

Name: ____________________

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