CSE 121 Autumn 2022 Final Exam

December 14, 2022 - 12:30-2:20pm - Kane Hall

Name of Student: ____

Section (e.g., AA):____________________ Student Number: ____________________________

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: Check all statements that are true. The entirety of the statement should be true to be selected.

- ☐ Primitive data types in Java (e.g. `int`, `double`, `boolean`, `char`) obey value semantics. This means assigning a new primitive variable to another primitive variable results in the new variable holding a copy of the data type's value.
- ☐ Reference semantics allows us to pass an object as a parameter to a method, and to modify it without need for a `return` statement to "get back" the modified object.
- ☐ Objects in Java (e.g. `Scanner`, `File`, `Random`) obey reference semantics. This means when an object is passed as a parameter to a method, a reference to the object is passed, and no copy of the object is made for the scope of the method.
- ☐ Value semantics implies that any primitive data type variable being passed as a parameter to a method remains unchanged outside the scope of the method call.

Part B: Suppose we want to generate a random integer with an already-declared Random object called `rand`. Which line will give back a random value in the range of 17 (inclusive) to 23 (exclusive)?

- ☐ `int num = rand.nextInt(17) + 7;`
- ☐ `int num = rand.nextInt(17) + 23;`
- ☐ `int num = rand.nextInt(7) + 17;`
- ☐ `int num = rand.nextInt(6) + 17;`

Part C: 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.

`"r" + 2 * (22 % 7) + "d" + 7 / 3`	
`7 % 8 / 4 * 10.0 / 5 + 1`	
`7 <= (12 / 2) && 3 > 10 % 5`	

2. Array Code Tracing

Consider the following method:

```
public static int[] mystery(int[] list, int ind1, int ind2) {
    int length = list.length - (ind2 - ind1) - 1;
    int[] result = new int[length];

    for (int i = 0; i < ind1; i++) {
        result[i] = list[i];
    }

    int index = ind1;

    for (int i = (ind2 + 1); i < list.length; i++) {
        result[index] = list[i];
        index++;
    }

    return result;
}
```

Part A: Consider the following code:

```
int[] arr = {4, 9, 3, 6, 8, 3, 4, 0};
int[] result = mystery(arr, 2, 5);
```

What values are present in **`arr`** after this code is executed?

- ☐ [4, 9, 4, 0]
- ☐ [4, 9, 2, 6, 8, 5, 4, 0]
- ☐ [4, 9, 3, 6, 8, 3, 4, 0]
- ☐ [4, 9, 6, 8, 4, 0]

Part B: Consider the following code:

```
int[] arr = {4, 9, 3, 6, 8, 3, 4, 0};
int[] result = mystery(arr, 0, arr.length - 1);
```

What values are present in **`result`** after this code is executed?

- ☐ [0 9 3 6 8 3 4 7]
- ☐ [4 9 3 6 8 3 4 0]
- ☐ [9 3 6 0 8 3 4]
- ☐ [ ]

Part C: Which of the following best describes what the method `mystery` does?

- ☐ Modify `list` by removing the elements at indexes between `ind1` and `ind2`.
- ☐ Modify `list` by removing all elements except those at indexes between `ind1` and `ind2`.
- ☐ Return a new array containing only the elements in `list` at indexes between `ind1` and `ind2`.
- ☐ Return a new array containing the elements in `list` except those at indexes between `ind1` and `ind2`.

3. Debugging

Consider a static method called **`findNumber`** that generates random numbers looking for a particular target a particular number of times. The method takes two parameters:

- `int target` - the number to search for
- `int count` - the number of times to find the target

The method generates and prints random integers between 1 and 10 (inclusive) until the number `target` has been generated `count` times. It then returns how many numbers were generated.

For example, suppose the following call was made:

```
int tries = findNumber(2, 2);
```

This call might produce output like the following:

```
Searching for 2 2s
Generating numbers: 5 2 4 8 6 6 10 2
```

In the example above, at the end of this call, the value 8 would be returned from the method and stored in the variable `tries`, because 8 numbers were generated before two 2s were seen. (Note that, due to randomness, this call would always not produce the exact same output or return value.)

Consider the following proposed buggy implementation of `findNumber()`:

```
public static int findNumber(int target, int count) {
    Random rand = new Random();
    int found = 0;
    int total = 0;

    System.out.println("Searching for " + count + " " + target + "s");
    System.out.print("Generating numbers: ");
    while (total < count) {
        int num = rand.nextInt(10) + 1;
        System.out.print(num + " ");
        if (num == target) {
            found++;
        }
        total++;
    }
    System.out.println();

    return total;
}
```

This implementation contains a single bug that is causing it to not work as intended.

(continued on next page…)

As an example, if the following code is executed:

```
int tries = findNumber(2, 2);
```

The buggy implementation might produce the following output:

```
Searching for 2 2s
Generating numbers: 3 1
```

After this call to the buggy implementation, the variable `tries` would contain the value 2.

Part A: Identify the single line of code that contains the bug. Write your answer in the box to the right as a single number.

Part B: Annotate (write on) the code below to indicate how you would fix the bug. 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.

```
1  public static int findNumber(int target, int count) {
2      Random rand = new Random();
3      int found = 0;
4      int total = 0;
5
6      System.out.println("Searching for " + count + " " + target + "s");
7      System.out.print("Generating numbers: ");
8      while (total < count) {
9          int num = rand.nextInt(10) + 1;
10         System.out.print(num + " ");
11         if (num == target) {
12             found++;
13         }
14         total++;
15     }
16     System.out.println();
17
18     return total;
19 }
```

4. General Programming

Write a static method named **`testMindReading`** that tests a user's ability to read your mind by guessing secret numbers. Your method should take two parameters:

- `Scanner console` - the `Scanner` to use for user input
- `int[] secrets` - an array of secret numbers the user will try to guess

Your method should prompt the user to enter a guess for each number in `secrets` and inform them if their guess is correct. The user should only be asked for one guess for each number in `secrets`, meaning the total number of guesses will be equal to the number of values in `secrets`. Your method should then *return* the number of correct guesses.

For example, suppose the following code was executed:

```
Scanner console = new Scanner(System.in);
int[] secrets = {4, 8, 15, 16, 23, 42};
int score = testMindReading(console, secrets);
```

This code would produce output like the following (user input **bold and underlined**):

```
Enter your guess: 4
Correct!
Enter your guess: 8
Correct!
Enter your guess: 12
Incorrect. It was 15.
Enter your guess: 16
Correct!
Enter your guess: 20
Incorrect. It was 23.
Enter your guess: 42
Correct!
```

Notice that the user's first guess is tested against the first element of `secrets`, their second guess is tested against the second element, and so on. At the end of this call, the value 4 would be returned from the method and stored in the variable `score`, since the user guessed 4 numbers correctly.

You may assume that `secrets` contains at least one value and that the user inputs exactly one integer when prompted.

Write your solution in the box on the next page.

Write your solution to problem #4 here:

5. File I/O Programming

Write a static method called **`payEmployees`** that computes how much to pay a group of employees based on hours and rate read from a text file. Your method should take one parameter:

- `Scanner payroll` - a Scanner connected to the input file to read from

Each line of the input file will contain an employee's name (as a single token), followed by their hour pay rate (a real number, e.g. `7.50`) and the number of hours they worked on each of a series of days (as integers). Your method should count the number of days the employee worked and compute how much they earned over those days, then output that information.

For example, suppose the file "payroll.txt" contained the following data:

```
Aang 21.50 3 7 5 0 2 8
Sokka 19 4 12 6
Katara 24.65 8 8 8 8
Zuko 20.55 21
```

Then suppose the following code was executed:

```java
Scanner input = new Scanner(new File("payroll.txt"));
payEmployees(input);
```

This code would produce the following output:

```
Aang worked 6 days and earned $537.5
Sokka worked 3 days and earned $418.0
Katara worked 4 days and earned $788.8
Zuko worked 1 days and earned $431.55
```

In this example, Aang earns $21.50 per hour, and worked days of 3, 7, 5, 0, 2, and 8 hours, for a total of 25 hours, which works out to $537.50. Sokka earns $19.00 per hour and worked days of 4, 12, and 6 hours, for a total of 22 hours and $418.00. Notice that total pay is not rounded or formatted in the output. You should not round or format any number as part of your output–simply output the values exactly as they are computed by Java.

You may assume that each employee worked at least one day, that the input file contains at least one line, and that each line of the input is formatted as described above.

Write your solution in the box on the next page.

Write your solution to problem #5 here:

6. Array Programming

Write a static method called **`chooseShorter`** that finds the shorter string at each element from a pair of arrays. Your method should take two parameters:

- `String[] words1` - the first array of strings to consider
- `String[] words2` - the second array of strings to consider

Your method should *return* a new array where each element of the new array contains the shorter of the two strings at that index in the parameter arrays. If the two strings are the same length, your method may choose either one. If one array is longer than the other, the result array should contain the extra strings from the longer array.

For example, suppose the following code was executed:

```
String[] words1 = {"alpha", "bravo", "charlie", "delta", "echo"};
String[] words2 = {"zulu", "yankee", "x-ray", "whiskey", "victor"};
String[] result = chooseShorter(words1, words2);
```

After this code is executed, the array result would contain the following values:

```
["zulu", "bravo", "x-ray", "delta", "echo"]
```

Note that the shorter string at each index was put into the result array (for instance “zulu” is shorter than “alpha” and “echo” is shorter than “victor”).

As another example, suppose the following code was executed:

```
String[] words1 = {"alpha", "bravo", "charlie", "delta", "echo"};
String[] words2 = {"zulu", "yankee", "x-ray", "whiskey", "victor", "uniform"};
String[] result = chooseShorter(words1, words2);
```

In this case, after this code is executed, `result` would contain the following values:

```
["zulu", "bravo", "x-ray", "delta", "echo", "uniform"]
```

Notice that the extra element “uniform” from the longer parameter array was also added to the result.

You may assume that both parameter arrays contain at least one element.

Write your solution in the box on the next page.

Write your solution to problem #6 here:

This page intentionally left blank for scratch work

CSE 121 Final Cheat Sheet

Declaring and using arrays

```
type[] name = new type[length];
type[] name = new type[]{VAL1, VAL2, VAL3, ...};
type[] name = {VAL1, VAL2, VAL3, ...};
name[index] = value;
name.length // Number of elements in array
```

String Method	Description
`charAt(i)`	returns character in this String at index **i**
`contains(str)`	returns `true` if this String contains **str** inside it
`endsWith(str), startsWith(str)`	returns `true` if this String ends/starts with **str**
`equals(str)`	returns `true` if this String is the same as **str**
`equalsIgnoreCase(str)`	returns `true` if this String is the same as **str**, ignoring capitalization
`indexOf(str)`	returns the first index in this String where **str** begins (-1 if not found)
`length()`	returns the number of characters in this String
`replace(str, new)`	returns a new String with all **str** in this String replaced with **new**
`substring(i)`	returns characters in this String from index **i** (inclusive) to end (inclusive)
`substring(i, j)`	returns characters in this String from index **i** (inclusive) to **j** (exclusive)
`toLowerCase()`	returns a new String with all lowercase letters
`toUpperCase()`	returns a new String with all uppercase letters

Random Method	Description
`nextInt(max)`	random integer from 0 (included) to **max** (excluded)
`nextInt(min, max)`	random integer from **min** (included) to **max** (excluded)

Math Method	Returns
`Math.abs(val)`	returns the absolute value
`Math.max(val1, val2)`	returns the larger of two values
`Math.min(val1, val2)`	returns the smaller of two values
`Math.pow(base, exp)`	returns the **base** to the **exp** power
`Math.sqrt(num)`	returns the square root

Scanner Method	Description
`hasNext()`	returns `true` if there is a next token to read
`hasNextLine()`	returns `true` if there is a next line to read
`hasNextInt()`	returns `true` if there is a next token and it can be read as an `int`, `false` otherwise
`hasNextDouble()`	returns `true` if there is a next token and it can be read as an `double`, `false` otherwise
`next()`	returns one token as `String`
`nextLine()`	returns entire line as `String`
`nextInt()`	returns one token as `int`, throws exception if not
`nextDouble()`	returns one token as `double`, throws exception if not

PrintStream Method	Description
`print(str)`	prints **str**
`println(str)`	prints **str** and then terminates the line