COS125 - Precept 11 (Functions II)

1 Array Mutation

Download `precept11.zip` - from the precepts webpage, unzip and open the project folder.

Open `Arrays.java` - and create several functions that will collectively act as a library of functions an array user might want to use. The functions that the user can call are collectively called the API, whereas the library is the set of implementations you will create:

- `printArray(int[] a)` - prints a 1D array in a nice to read format like:

  {1, 2, 3}

- `printArray2D(int[][] a)` - prints a 2D array in a nice to read format like:

  {1, 2, 3}

  {4, 5, 6}

  {7, 8, 9}

- `copyArray(int[] a)` - copies the values of `a` into a new array `b`, then returns `b`.
- `reverseInPlace(int[] a)` - reverses the array `a` in-place, i.e., without creating a new array.
- `reversedCopy(int[] a)` - copies the values of `a` in reverse order into a new array and returns it.
- `slice(int[] a, int start, int end)` - copies the values of `a` between the start and end index (inclusive) into a new array and returns it.
- `convertToDoubles(int[] a)` - creates a `double` array and converts all integers to doubles, returning the new doubles array.
- `median(int[] a)` - finds the median in a sorted array. The median is the middle index of an odd-length array and the average of the two middle indices in an even-length array.
- `max(int[] a)` - finds the largest value of an array and returns it.
- `min(int[] a)` - finds the smallest value of an array and returns it.

Functions should also handle invalid inputs and corner cases. You can assume every integer array argument contains at least one element.

Also, remember to leave a comment before each function in a class!

2 Modularizing `ColorContrast.java`

Copy your submission of `ColorContrast.java` from the Conditionals assignment into the folder and modularize your code by implementing the methods

- `double calcLuminance(int colorLevel)`
- `double totalLuminance(double luminanceRed, double luminanceGreen, double luminanceBlue)`
- `double colorContrast(double luminanceText, double luminanceBackground)`

and calling them from `main()`.

You can place all constants outside of main so that every function within the class can access them, as long as you add the keyword `static` in front of them like this:

```
public class MyClass{
    static int MYCONSTANT = 5;

    public static void main(String[] args){
        System.out.println(MYCONSTANT);
    }
}
```

Compare this version to your original submission. Did the number of lines decrease at all? Did the legibility change? Which version do you prefer?

3 Database Search

In Precept 9, we created several versions of the `Search` program to find all records that match certain search terms using fake data in `healthcare_data.txt`. Just like the bonus problem in that precept, we will create `DatabaseSearch.java` that takes in an arbitrary number of search terms as pairs, each pair consisting of a category and then the term itself. For instance:

```
java-introcs Search AGE 23 RACE white ETHNICITY hispanic < healthcare_data.txt
```

This time, however, we will write break our program up into several functions. You may start with one of your `Search` programs from last precept and refactor it or start a new program from scratch, but either way, your program should pass each new line under analysis to a separate search function. This function will go through the process of deciding whether to print the line or not. Feel free to create other helper functions that help with legibility.