PIC 20A: Homework 1 (due 1/20 at 5pm)

Submitting your homework

The zip file you extracted to find this pdf includes a file called `HW1.java`.
In this assignment, you will edit this file and submit it to Gradescope.

- Upload `HW1.java` to **Gradescope** before the deadline.
- **Name** the file exactly as just stated.
- Do **not** enclose `HW1.java` in a folder or zip it.

  You should be submitting exactly **one file** and it should have the **extension** `.java`.
- Be sure that your code **compiles and runs** using **Adoptium's Temurin Version 11 (LTS)**.

Tasks

- For each function (public static member function) that is defined in `HW1.java`, read its function comment, and correct its definition. The definitions given are only provided so that `HW1.java` compiles even before you edit it.

Comments

1. `public static double average(int[] arr)`

   I hope that you can solve this in under 5 minutes. If not, please review PIC 10A thoroughly.

   I did not think about "the mean of no `int`s" when I coded my solution. Your solution is likely to handle this correctly automatically because `Double.isNaN(0.0 / 0)` evaluates to `true`.

2. `public static String binary(int n)`

   This question is more difficult than the first, but I hope that your PIC 10A instructor set you a similar problem. Here are some suggestions...

   - For full credit and to maximize learning, do not use any special functions.
   - Handle the case when `n == 0` at the beginning.
   - To avoid running into problems with overflow, work with `long`s in the function body.
   - If the number you wish to express in binary is negative, add a dash to the `String` that you're preparing to return, and replace the number by its negative (making it positive).
   - After that, I used two `while` loops to find the binary representation of a positive integer.

3. `public static boolean isMagicSquare(int[][] square)`

   If you need review on `for`-loops, this question is a good one.
   Here are some suggestions and help...

   - Store the `length` of the array of arrays.
   - You're going to check the conditions to be a magic square one by one. If a condition fails, you can `return false`. Otherwise, you can move onto the next condition. There is no reason to have a load of `boolean`s acting as "flags". Such code is nasty to read.
   - Check the array of arrays is square by looping over the rows with an enhanced `for`-loop.
   - For an $(N \times N)$-square, check each of the numbers 1 to N^2 appears exactly once. I'd use nested enhanced `for`-loops to loop through all the values in the square; if they're too big or small, `return false`; otherwise, record their presence in an array of `boolean`s called `used`. Finally, loop through `used` checking it is full of `true`s. This ensures every value is used exactly once because using a value twice would cause us to miss another value.
   - For an $(N \times N)$-square, the "magic total" is given by $\frac{1}{2} \cdot N \cdot (N^2 + 1)$. This is the value the rows, columns, and diagonals need to sum to.
   - Use nested (standard) `for`-loops to check the rows sum to the magic total.
   - Use nested `for`-loops to check the columns sum to the magic total. I'd avoid trying to check the rows and columns simultaneously. It's more confusing to read that way.
   - Use a single `for`-loop to check the first diagonal sums to the magic total.
   - Use a single `for`-loop to check the second diagonal sums to the magic total.

4. `public static int[] firstPrimes(int N)`

   - Recall that `ControlFlow.java` solves a very closely related problem.
   - Note that the function comment in `ControlFlow.java` says it is a "slow algorithm."
     If you solve this question using that algorithm with no extra optimizations,
     `firstPrimes(3_000_000);` will take much longer than 3 minutes to execute.
   - For full credit, your code must execute in less than 12 seconds in the PIC Lab.
     An unoptimized version that is otherwise correct **will only lose one point**.
   - Note that the new MacBooks are substantially faster than the PIC Lab computers.
     An optimized version can execute in a little over 2 seconds on a new MacBook.

Grading

1. 4 test cases will be used to give a score out of 4.

2. 6 test cases will be used to give a score out of 6.

3. 8 test cases will be used to give a score out of 8.

4. 5 test cases will be used to give a score out of 5.

   If you've optimized your code to make it run as fast as necessary, you'll receive a sixth point.

5. Total: 24 points.