Perfect Number [60 points]

A perfect number is a positive integer that is equal to the sum of its positive divisors, excluding the number itself. In other words, a perfect number is one where the sum of its divisors (excluding itself) equals the number itself.

For example: the number 6 has divisors 1, 2, and 3 (excluding itself), and (1 + 2 + 3 = 6), making it a perfect number. The next perfect number is 28, since (1 + 2 + 4 + 7 + 14 = 28).

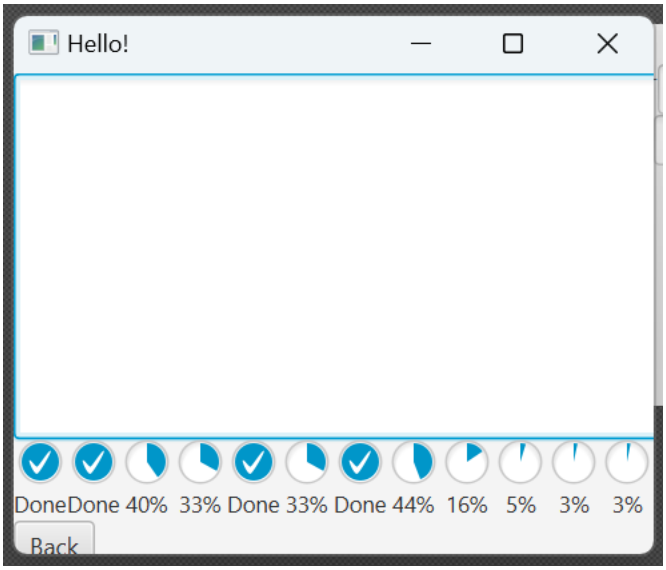
Create a GUI that shall take two numbers, one is the num and the other is the number of threads cnt. Your goal is to show all the perfect numbers from 1 to num using cnt threads. You should also have a way to handle errors and showing the appropriate error message in the GUI.

Next is you should create Threads by unequally dividing the tasks to the threads. Divide them unequally to highlight the differences in progress of individual threads. In order to visualize the progress of the threads, dynamically create the same number of ProgressIndicators as the number of threads in your GUI and use their setProgress() method that will indicate how close each threads are to finishing the task. The setProgress() method will take a double from 0 to 1, where 1 means 100% done.

When all threads are done, aside from showing the perfect numbers (preferably in GUI), also output how many are less than perfect (i.e., the sum of factors is less than itself) and more than perfect (i.e. the sum of factors is greater than itself).

In creating GUI, you can use either FXML or coding.

Sample Output (you may add more elements):



Fibonacci Sequence [40 points]

The Fibonacci sequence is a mathematical sequence in which each number is the sum of the two preceding ones. The sequence typically starts with 0 and 1, and then continues as follows: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ...

Your task is to create a program that will ask for an integer num and shall print the first num elements of the Fibonacci sequence.

However you are to achieve this using Threads.

Create a Runnable, called FibRunnable, that has a number n as its field. The run() method shall be to wait for the values of the two FibRunnables before it. This will be discussed in the preceding paragraph.

To have access to those FibRunnables, create a static array of Threads in the Main class with size num corresponding to the user input. These Threads of FibRunnables must be created such that the first FibRunnable in the array will have n = 1, the second n = 2, and so on.

In the main method, only run the last FibRunnable. This should make sense for the user to only be able to run that FibRunnable according to its input and hide the "recursive" nature of the algorithm.

As mentioned, the run() method shall be to wait for the results of the two FibRunnables before it. This being said, there should also be a static array of integers with the same size num to store the result of the FibRunnable. The FibRunnable will first wait for the result to be completed by the preceding two FibRunnables in the static array. Make sure that these FibRunnables are running or already have ran.

There are two ways on how to do this: either wait and notify on the result, or wait for those two threads to die. I will leave the decision as to which way is best.

Note that the goal is to print the first num Fibonacci sequence. Do this in the main method.

Smarter BruteForce [50 points]

Improve the BruteForce password by having n*5 threads, where n is the length of the password.

Instead of having each thread running on different starting character, this improvement involves the use of vowels wherein each thread will focus on a vowel and a position. For example, the password "gwapo", having 5 as its length, it shall create 5*5=25 threads to perform the following:

Thread 01: _ _ _ _ a

Thread 02: _ _ _ a _

Thread 03: _ _ a _ _

Thread 04: _ a _ _ _

Thread 05: a _ _ _ _

Thread 06: _ _ _ _ e

Thread 07: _ _ _ e _

...

Thread 10: e _ _ _ _

...

Thread 24: _ u _ _ _

Thread 25: u _ _ _ _

Hence, there will be two threads that can possibly find "gwapo" -- the __a__ and ____o. In your output as you print your attempts, print it similarly as the one above, having the thread number before the attempt. In a similar way as the laboratory activity, stop the other threads if it has been found by one thread.

With this technique, we may not be able to have an attempt for the words without vowels -- to which this problem can live with.

FREE [30 points]

You can choose to improve the login GUI that you've done in the previous laboratory. You can add more functionalities like adding Night Mode, file reading, exception handling, CSS styles, adding more FXML navigation – anything to show your craft and creativity.

The points to be garnered here will depend on the complexity of your implementation.