Hints for BrilliantNumber and NumberRange Problems
4/5/2019

Tester

Add "Tester" classes, e.g. BrilliantNumberTester and NumberRangeTester that "test" or "drive" your code to ensure it works. These do [not] need to be handed in, but are really beneficial (fundamental software engineering rule)

Reuse, Reuse, Reuse Code

And then reuse code. You've engineered a lot of functionality into BrilliantNumber. So NumberRange can be pretty "lightweight". Construct instances of BrilliantNumber whenever you can and let it do it's thing.

Also, even within BrilliantNumber add and use helper methods at every corner. E.g. see " isDivisibleBy" below.

Discussion Board (D2L – Communication – Discussions)

Find hints and tips here. Load it up with all the questions and comments you like.

BrilliantNumber

Assume BrilliantNumber's int "n" is a positive natural number (i.e. an int >= 1) Otherwise we'd have difficulty with e.g. "factorCount" for n=0 (factorCount is infinite). In other words, do not worry about n=0.

BrilliantNumber -- isDivisibleBy

First program "isDivisibleBy", then call it as much as you can (to make things easier for other methods). This is a general practice -- try to use what you've already coded.

BrilliantNumber -- factorCount

See:

"Chapter 9 Loops -- For Loop -- Examples"
Example 6 -- Counting Factors Of a Number

BrilliantNumber -- isPrime

Could isPrime delegate and call factorCount?

<u>“largestPrimeGap”</u>

One possible algorithm:

```
Set maxGap to 0
Set prevPrime to 0
Iterate from start to stop (call loop variable 'n')
      If n is prime
            If prevPrime > 0
                  gap = n - prevPrime - 1
                  If gap > maxGap
                        maxGap = gap
            prevPrime = n
return maxGap
```

hint: Employ "BrilliantNumber"

<u>“largestPrimeGapPair”</u>

One possible algorithm is to use the algorithm from “largestPrimeGap” with this tweak:

```
If gap > maxGap
      maxGap = gap
      gapStart = prevPrime
      gapStop = n
```

Then at the end of the method return the specified friendly string showing the gap (build the string using gapStart and gapStop)

<u>Jotting down algorithms</u>

Many times the problem description will be much longer than the solution code, and jotting down a “paper” quick alrogithm may be helpful. A couple examples are shown here, where each algorithm is short and the Java code could be short as well.

possible algorithm for “shrink” (2 statements)

```
1. newLength = size / factor
2. Set new length to newLength
```

hint -- Employ "BrilliantNumber" e.g. "divide", for #1 (and see “General Hints” at top of problems pdf).
hint -- Add little helper method "setLength(int newLength)" for #2

possible algorithm for “countSquareAndCubeNumbers” (3 statements)

```
1. count = 0
2. Loop from our start to stop (loop variable "n")
      2a. If n is a square and is a cube
            Increment count
3. Return Count
```

hint -- Employ a NewNumber for algorithm statement #2a (see "General Hints" at top of problems PDF, for constructing objects on the fly)