Assignment #3

Due

Part 1: Submit (multiple submissions possible) before 8:30 pm on Tuesday, February 25
Part 2: Submit before 8:30pm on Friday, February 28

Learning Outcomes: Upon successful completion of this assignment you will be able to:

- Explain what is a recursive technique to solve a problem
- Write and test Java recursive methods

Part 1 (Milestone)

NOTE: In this *milestone* part (part 1) of the assignment you will only be asked to hand in the answers to the questions (No code will be submitted.) However, the methods you write for the milestone will be submitted in Part 2.

Problem statement:

Write three Java static methods, all that calculate x^n for some $n \geq 0$, following the specifications (below), and then answer questions (below) about them. A simple tester (**MathTest.java**) is provided: it will not compile until your methods or method stubs are added.

- The first method, called `powerOne()`, must be an iterative method to compute x^n for some $n \geq 0$. Use the signature: `static double powerOne(double x, long n).`
- The second method, called `powerTwo()`, must be a recursive method (ie, a method that calls itself) to compute x^n by using the following recursive formulation:

  $x^0 = 1$

  $x^n = x * x^{(n-1)}$ if $n>0$

  Use the signature: `static double powerTwo(double x, long n).`
- The third method, called `powerThree()`, must be a recursive method to compute x^n by using the following recursive formulation:

  $x^0 = 1$

  $x^n = (x^{(n/2)})^2$ if $n>0$ and n is even

  $x^n = x * (x^{(n/2)})^2$ if $n>0$ and n is odd

  Use the signature: `static double powerThree(double x, long n).`

Questions:

- How may multiplications will each of the methods, `powerOne()`,`powerTwo()`, and `powerThree()` perform when computing 3.15^{17}?
- How many calls to each of `powerTwo()` and `powerThree()` will be performed when computing 3.15^{17}?

Submitting your Solution:

When complete, submit your answers to the Questions (only!) to the CSc 115 Connex Site using the Tests & Quizzes: Assignment 3 Milestone link before 8:30 on Tuesday, February 25, 2014.

Part 2

Problem statement:
Complete the Java program in **`a3Tester.java`** that performs mathematical computations. Note that **`a3Tester.java`** will not compile until your methods or method stubs are added. (You will be creating some of the calculations that are available in Java's Math class!)

Your program needs the following static methods:

- An iterative static method to compute x^n for some $n \geq 0$.
  (Call this method: `powerOne()`,
  ie, use the signature `static double powerOne(double x, long n)`.)
- A recursive static method that computes x^n by using the following recursive formulation:
  $x^0 = 1$
  $x^n = x * x^{(n-1)}$ if $n>0$ (Call this method: `powerTwo()`.)
- a recursive static method that computes x^n by using the following recursive formulation:
  $x^0 = 1$
  $x^n = (x^{(n/2)})^2$ if $n>0$ and n is even
  $x^n = x * (x^{(n/2)})^2$ if $n>0$ and n is odd (Call this method: `powerThree()`.)
- an iterative static method to calculate $x!$ for some $x \geq 0$.
  (Call this method: `factOne()`,
  ie, use the signature `static double factOne(double x)`.)
- a recursive static method that computes $x!$ using the following recursive formulation:
  $0! = 1$
  $x! = x * (x-1)!$ if $x>0$ (Call this method: `factTwo()`.)
- a iterative static method that uses an $(n+1)$ term MacLaurin series to compute e^x, using the formula

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!} = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots \quad \text{for all } x$$

  (Call this method `eOne()`,
  ie, use the signature `static double eOne(double x, long n)`.)
- a recursive static method that uses an $(n+1)$ term MacLaurin series to compute e^x, called $e(x,n)$, by using the following recursive formulation:
  $e(x,0) = 1$
  $e(x,n) = e(x,n-1) + x\text{^}n/n!$, if $n>0$
  (Call this method `eTwo()`.)
- a second recursive static method that uses the MacLaurin series to compute e^x, called $e(x, n)$. It should use less recursive calls and/or less multiplications. Using the following observation:
  $1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \cdots$ can be factored as follows:

$$1 + x\left(1 + \frac{x}{2}\left(1 + \frac{x}{3}\left(1 + \frac{x}{4}(1 + \cdots)\right)\right)\right)$$

  (Call this method `eThree()`.)

Put all your methods into the **a3Tester.java** program and pass the tests.

Submitting your Solution:

- When complete submit your **`a3Tester.java`** file to the CSc 115 Connex Site using the Assignments: Assignment 3 Submission link before 8:30pm on Friday, February 28.

Marking:

Part 1: 5 marks (one for the correct answer to each question.)
Part 2, Power calculations: 3 marks (1 mark for passing each of the 3 tests.)
Part 2, Factorial calculations: 2 marks (1 mark for passing each of the 2 tests.)
Part 2, e^x calculations: 5 marks (1 mark for the iterative test, 2 marks for each recursive test.)
Documentation: 3 marks for appropriate documentation of your program and the methods.

Total: 18 Marks

Background Information

Did you wonder why we were counting the number of multiplications that were needed to complete a calculation or the number of times a method was called? All of the techniques modeled in this assignment come from a computing field called *Numerical Analysis*. Scholaropedia (http://www.scholarpedia.org/article/Numerical_analysis) defines **Numerical analysis** as "the area of mathematics and computer science that creates, analyzes, and implements algorithms for solving numerically the problems of continuous mathematics."

In this example we are approximating results using real (in this case Java double) values. One problem with real numbers is that the very last digit of every number is certainly an approximation. (For example, have you ever seen a program output 0.999999999999999996, or something similar, when, in fact, the result should have been 1.000000000000000000? Both are approximately the same number, except that the last digit's approximation is different.)

With every calculation on real numbers an additional digit on the end of the result becomes approximated. Performing a long calculation, like `eTwo(3.15, 17)`, could mean that most, if not all digits, of the result are approximate. Changing the order and, thus, reducing the number of calculations (as in `eThree()`) maintains more accuracy digits.

For an entire course on useful numerical calculations, look up CSc 349a.