AP Computer Science — Activity – Calculating Pi

BACKGROUND

Pi, or π, is the ratio of any circle's circumference to its diameter.

$$Pi(\pi) = \frac{circumference}{diameter} \approx 3.14159265...$$

Pi is one example of a *transcendental* number: a real or complex number that is not algebraic, i.e. not a root of a non-zero polynomial equation with rational coefficients. The search for increasingly accurate determinations of the value of *pi* (π) is an activity that goes back thousands of years.

OBJECTIVE

The purpose of this assignment is implement an algorithm for calculating *pi* in two different ways: one using recursion, and one using a loop.

In this activity, we'll be using a portion of this infinite series to calculate π:

$$\frac{\pi^2}{6} = \frac{1}{1^2} + \frac{1}{2^2} + \frac{1}{3^2} + ... + \frac{1}{n^2}$$

You'll also be identifying how many steps it takes us to get within 1% of the true value of π.

PROCEDURE

Complete the steps indicated below. At the conclusion of this activity submit a zipped file `pi.zip` that contains `PiLoops.java` and `PiRecursion.java`, along with any supporting files you choose to include.

I. Calculating π using a loop

In this version of the calculation, write the program `PiLoops.java`, which will use only a `main` class to calculate *pi*. The program should ask the user how many iterations they want to run (the value `n` in the equation above), and then use a loop to sum that series. Once the loop has completed, perform additional calculations as needed to determine a value for *pi*.

Additionally, the program should print out the value of *pi* as given by `Math.PI`, and calculate the percent error between the two values according to this formula:

$$\%\,error = \frac{|Calculated\ Value - Correct\ Value|}{Correct\ Value} \times 100$$

In the JavaDoc comments at the top of your program, include a statement indicating how many iterations it takes to arrive at a value of *pi* that is within 1% of the correct value.

II. Calculating π using recursion

This version of the program—`PiRecursion.java`—should produce exactly the same results as the version above, but it will implement the formula using *recursion* instead of a loop.

Recursion in this context involves performing calculations using a self-referential method—a method that calls itself. If you aren't familiar with the concept of recursion, a brief example is helpful.

The factorial of an integer **n** *is indicated as* **n!**, *and calculated as the product of that number and all the integers below it, down to 1. So:*

$$n! = (n) \times (n-1) \times (n-2) \times \ldots \times (1)$$

Using actual values:

$$6! = (6) \times (5) \times (4) \times (3) \times (2) \times (1)$$

Note that $6!$ *can be considered as* $6 \times 5!$ *, where* $5!$ *is the product of all the integers between it and 1. With this understanding we can define the factorial function recursively:*

$$n! = (n) \times (n-1)!$$

Getting back to recursive methods, then, we could define a method `factorial` recursively as follows:

```
01    /**
02     *   Returns the factorial of n by calculating recursively.
03     */
04
05    public static int factorial(int n)
06    {
07        if (n == 1)
08            return 1;
09        else
10            return n * factorial(n - 1);
11    }
```

In this example, when the `factorial` method is called, line 10 returns a value that depends in part on calculating the value of the factorial of `n - 1`. The method is calling itself again—*recursing*—with a different value of `n`. This process continues, with each repeated call of the method working on a value of `n` that is one less than it was before. Ultimately, the `if` statement in line 07 is required to indicate to the method the limit of its recursion: when we get down to 1, we send back that value to the previous call. At that point, returned values are passed all the way back up to previous calls until a final result is returned to the original call, and the value of `n!` is returned.

The program `PiRecursion.py` will include a main method as just as the loop version did, but the value of the infinite series will be calculated recursively. Write a static method `recursion` that takes a value `n` and operates on it recursively to calculate the sum of that series. Then use your result to calculate π as before.

QUESTIONS FOR YOU TO CONSIDER (NOT HAND IN)

1. What happens if you try to recurse too far? What are the limits of recursion for your machine/program/Java?

2. I was trying to calculate *pi* more precisely by using the `BigDecimal` class for my calculation, but I got this error. Why is this error being thrown? What should I do to fix it?

```
java.lang.ArithmeticException: Non-terminating decimal expansion; no
exact representable decimal result.
      at java.math.BigDecimal.divide(BigDecimal.java:1690)
      at piCalculator3.main(piCaclulator3.java:29)
```

3. Look up the Bailey–Borwein–Plouffe formula for calculating *pi*. What makes this particular formula for determining *pi* so different from most other series?

REFERENCES

http://www.numberworld.org/misc_runs/pi-10t/details.html
http://www.jensign.com/JavaScience/www/bigpi/index.html
http://crd-legacy.lbl.gov/~dhbailey/dhbpapers/bbp-alg.pdf
http://crd-legacy.lbl.gov/~dhbailey/dhbpapers/bbp-formulas.pdf
https://en.wikipedia.org/wiki/Bailey-Borwein-Plouffe_formula
http://mathworld.wolfram.com/BBPFormula.html