Assignment 1.3: Methods, Recursion, and Loops

Due: 8am Wednesday, October 1st, 2008

Directions

The problems for this assignment are described below. You will submit this assignment by emailing it do Mr. Wulsin. The file will be named "Assignment 1.3-LastName.zip", where [LastName] is obviously your last name. All of your submitted assignments will be named in this fashion. The Zip file will contain a folder of the same name (so, "Assignment 1.3-LastName"), which will contain a folder for each of the subsequent problems. Each of these subfolders will have the name of its particular problem. Each of these problem subfolders will contain all of the necessary files for that particular problem. It's very important that you follow all of these naming conventions exactly as specified.

Problems

1) nFactorial

You will create a program that uses recursive methods to calculate the factorial of a number. Remember, the factorial of a number is defined as

$$n! = (n)(n-1)(n-2)\cdots(1)$$

The program should prompt the user to enter a number (say, 10) and then output the factorial (for 10, 3628800). You will make a `public static long` (note the `long` return type since the result of factorials can often be quite large) method that you use in your recusion and that you call from your `public static void main`.

2) piApprox

You will create a program that approximates π. As you will learn after taking calculus, all functions can be defined by a finite or infinite series. The series that defines π is

$$\pi = 4\sum_{i=1}^{\infty} \frac{(-1)^{i-1}}{2i-1}$$

or in expanded form

$$\pi = 4(1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \frac{1}{11}\cdots)$$

You should use a `for` loop for this program to calculate the π up to n elements in the series. You may specify n in your code and do not need to have the user input it. Once you have finished this program, notice how the number of elements affects the closeness of the approximation. You may find it useful to print out Java's approximation of π for comparison using the `Math.PI` constant.

3) binConvert

You will create a program that converts between binary and decimal values. We haven't yet learned about arrays, which are the best way to store numbers of various bases, so for now, we will store both our binary and our decimal values as `int`s. Your program will prompt the user for either binary or decimal number and then display the conversion to decimal (or binary) and then that conversion's conversion back to binary (or decimal). Here is a sample output of such a program:

```
Enter a binary number:  10110000
The decimal value of 10110000 is 176
The binary value of 176 is 10110000
```

You will know that your conversion methods are working correctly if your binary→decimal→binary conversion ends up with the same binary number you started with (or decimal→binary→decimal if you prefer). Your program will have two different `static` methods, `binToDec` and `decToBin` that will perform your conversions. **Important:** since this algorithm is probably the trickiest we've done so far, you *must* outline it (both binary→decimal and decimal→binary) on paper before you start writing any code.