csTask57 Assignment

The activities in this assignment require you to create and modify Java programs using BlueJ. The BlueJ project csTask57 already contains various classes for you to work on.

Follow the directions given for each part. At the beginning of the next class, all modified programs (class files) must be in your project, and the project must be copied to your shared Dropbox folder.

When questions are posed, write your responses as full sentences in the Readme.txt file in your project.

There are four parts to this assignment:

I. Computing Powers
II. Counting and Summing Digits
III. Base Conversion
IV. Efficient Computation of Fibonacci Numbers

I. Computing Powers

Computing a positive integer power of a number is easily seen as a recursive process.

Consider a^n:

- If n = 0, a^n is 1 (by definition)
- If n > 0, a^n is a * a^{n-1}

The Power class contains a main program that reads in integers `base` and `exp` and calls method `power` to compute $\texttt{base}^{\texttt{exp}}$. Fill in the code for `power` to make it a recursive method to do the power computation.

The comments provide guidance.

II. Counting and Summing Digits

The problem of counting the digits in a positive integer or summing those digits can be solved recursively. For example, to count the number of digits think as follows:

- If the integer is less than 10 there is only one digit (the base case).
- Otherwise, the number of digits is 1 (for the units digit) plus the number of digits in the rest of the integer (what's left after the units digit is taken off).
- For example, the number of digits in 3278 is 1 + the number of digits in 327.

The following is the recursive algorithm implemented in Java.

```
public int numDigits (int num)
{
      if (num < 10)
      return (1); // a number < 10 has only one digit
      else
      return (1 + numDigits (num / 10));
}
```

Note that in the recursive step, the value returned is 1 (counts the units digit) + the result of the call to determine the number of digits in `num / 10`. Recall that `num/10` is the quotient when `num` is divided by 10 (recall that integer division truncates) so it would be all the digits except the units digit.

The class DigitPlay contains the recursive method `numDigits` (note that the method is static—it must be since it is called by the static method main). Compile it, and run it several times to see how it works.

Modify the program as follows:

1. Add a static method named sumDigits that finds the sum of the digits in a positive integer. Also add code to `main` to test your method. The algorithm for sumDigits is very similar to numDigits; you only have to change two lines!

2. Most identification numbers, such as the ISBN number on books or the Universal Product Code (UPC) on grocery products or the identification number on a traveller's check, have at least one digit in the number that is a check digit. The check digit is used to detect errors in the number. The simplest check digit scheme is to add one digit to the identification number so that the sum of all the digits, including the check digit, is evenly divisible by some particular integer. For example, American Express Traveller's checks add a check digit so that the sum of the digits in the id number is evenly divisible by 9. United Parcel Service adds a check digit to its pick up numbers so that a weighted sum of the digits (some of the digits in the number are multiplied by numbers other than 1) is divisible by 7. Modify the main method that tests your sumDigits method to do the following: input an identification number (a positive integer), then determine if the sum of the digits in the identification number is divisible by 7 (use your sumDigits method but don't change it—the only changes should be in main). If the sum is not divisible by 7 print a message indicating the id number is in error; otherwise print an ok message. (FYI: If the sum is divisible by 7, the identification number could still be incorrect. For example, two digits could be transposed.) Test your program on the following input:

    3429072 --- error
    1800237 --- ok
    88231256 --- ok
    3180012 --- error

III. Base Conversion

One algorithm for converting a base 10 number to base b involves repeated division by the base b. Initially one divides the number by b. The remainder from this division is the units digit (the rightmost digit) in the base b representation of the number (it is the part of the number that contains no powers of b). The quotient is then divided by b on the next iteration. The remainder from this division gives the next base b digit from the right. The quotient from this division is used in the next iteration. The algorithm stops when the quotient is 0. Note that at each iteration the remainder from the division is the next base b digit from the right—that is, this algorithm finds the digits for the base b number in reverse order.

Here is an example for converting 30 to base 4:

	quotient	remainder
	---	---
30/4 =	7	2
7/4 =	1	3
1/4 =	0	1

The answer is read bottom to top in the remainder column, so 30 (base 10) = 132 (base 4).

Think about how this is recursive in nature: If you want to convert x (30 in our example) to base b (4 in our example), the rightmost digit is the remainder x % b. To get the rest of the digits, you perform the same process on what is left; that is, you convert the quotient x / b to base b. If x / b is 0, there is no rest; x is a single base b digit and that digit is x % b (which also is just x).

The class BaseConversion contains the shell of a method `convert` to do the base conversion and a `main` method to test the conversion. The `convert` method returns a string representing the base b number, hence for example in the base case when the remainder is what is to be returned it must be converted to a String object. This is done by concatenating the remainder with a null string. The outline of the convert method is as follows:

```
public static String convert (int num, int b)
{
  int quotient; // the quotient when num is divided by base b
  int remainder; // the remainder when num is divided by base b
  quotient = _____________________________;
  remainder = _____________________________;
  if ( ___________________________________ )//fill in base case
      {
        return ("" + ___________________________________ );
      }
  else
      {
        // Recursive step: the number is the base b
      //representation of
        // the quotient concatenated with the remainder
  return ( ____________________________________________________ );
      }
}
```

Fill in the blanks above (for now don't worry about bases greater than 10), then in the BaseConversion class, complete the `convert` and `main` methods. The `main` method currently asks the user for the number and the base and reads these in. Add a statement to print the string returned by `convert` (appropriately labeled).

Test your function on the following input:

Number: 89 Base: 2 ---> should print 1011001
Number: 347 Base: 5 ---> should print 2342
Number: 3289 Base: 8 ---> should print 6331

Improving the program: Currently the program doesn't print the correct digits for bases greater than 10. Add code to your convert method so the digits are correct for bases up to and including 16.

IV. Efficient Computation of Fibonacci Numbers

The Fibonacci sequence is a well-known mathematical sequence in which each term is the sum of the two previous terms. More specifically, if fib(n) is the nth term of the sequence, then the sequence can be defined as follows:

fib(0) = 1
fib(1) = 1
fib(n) = fib(n-1) + fib(n-2) n>1

1. Because the Fibonacci sequence is defined recursively, it is natural to write a recursive method to determine the nth number in the sequence. The Fib class contains the skeleton for a class containing a method to compute Fibonacci numbers. Following the specification above, fill in the code for method fib1 so that it recursively computes and returns the nth number in the sequence.

2. The driver class TestFib contains a simple driver that asks the user for an integer and uses the fib1 method to compute that element in the Fibonacci sequence. Use it to test your fib1 method. First try small integers, then larger ones. You'll notice that the number doesn't have to get very big before the calculation takes a very long time. The problem is that the fib1 method is making lots and lots of recursive calls. To see this, add a print statement at the beginning of your fib1 method that indicates what call is being computed, e.g., "In fib1(3)" if the parameter is 3. Now run TestFib again and enter 5—you should get a number of messages from your print statement. Examine these messages and figure out the sequence of calls that generated them. (This is easiest if you first draw the call tree on paper.) . Since fib(5) is fib(4) + fib(3),you should not be surprised to find calls to fib(4) and fib(3) in the printout. But why are there two calls to fib(3)? Because both fib(4) and fib(5) need fib(3), so they both compute it—very inefficient. Run the program again with a slightly larger number and again note the repetition in the calls.

3. The fundamental source of the inefficiency is not the fact that recursive calls are being made, but that values are being recomputed. One way around this is to compute the values from the beginning of the sequence instead of from the end, saving them in an array as you go. Although this could be done recursively, it is more natural to do it iteratively. Proceed as follows:

a. Add a method fib2 to your Fib class. Like fib1, fib2 should be static and should take an integer and return an integer.
b. Inside fib2, create an array of integers the size of the value passed in.
c. Initialize the first two elements of the array to 1 and 1, corresponding to the first two elements of the Fibonacci sequence. Then loop through the integers up to the value passed in, computing each element of the array as the sum of the two previous elements. When the array is full, its last element is the element requested. Return this value.
d. Modify your TestFib class so that it calls fib2 (first) and prints the result, then calls fib1 and prints that result. You should get the same answers, but very different computation times.