Programming Assignment #7: Recursive Methods

Due Date: April 13, 2000

1 The Problem

In this program, you will write up to five (unrelated) recursive static methods and place them in a class called RecursiveMethods. Each method is described below. Try to code up as many of the five methods as possible. You must write at least three of them (any three) to receive full credit on this assignment. If you write more than three, you will receive EXTRA CREDIT for the ones you write (assuming of course that they are correctly written!). For the methods you do not write, just put a simple println statement in the body of the method saying that you did not code this method. For example:

```java
public static [void or datatype] methodName(...) {
    System.out.println("Sorry. I did not code this method.");
}
```

Please use the same names for the methods as given in the descriptions below!

1.1 Binary Numbers

Computers do all their calculations using the binary (base-2) number system. (Humans use decimal (that is, base-10) numbers.) To understand how binary numbers work, recall how numbers are represented in our own base-10 system. For example, the number 382 can be expanded as

\[ 382 = 3 \times 10^2 + 8 \times 10^1 + 2 \times 10^0. \]
Table 1: Binary Numbers From 1 to 10

<table>
<thead>
<tr>
<th>Decimal Number</th>
<th>Binary Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>11</td>
</tr>
<tr>
<td>4</td>
<td>100</td>
</tr>
<tr>
<td>5</td>
<td>101</td>
</tr>
<tr>
<td>6</td>
<td>110</td>
</tr>
<tr>
<td>7</td>
<td>111</td>
</tr>
<tr>
<td>8</td>
<td>1000</td>
</tr>
<tr>
<td>9</td>
<td>1001</td>
</tr>
<tr>
<td>10</td>
<td>1010</td>
</tr>
</tbody>
</table>

That is, every decimal number can be written in terms of the powers of 10, and the number 382 is simply a listing of the coefficients of those powers from the highest power to the lowest.

To represent numbers in the binary number system, first expand the number in powers of 2. For example, consider the decimal number 234. This number can be expanded in powers of 2 as

\[ 234 = 1 \times 2^7 + 1 \times 2^6 + 1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0. \]

Therefore, the representation of the decimal number 234 in binary is 11101010. Notice that the coefficients of the powers of 2 can only be 0 and 1. Therefore, a binary number contains only 0s and 1s. Table 1 presents more examples.

Write a recursive method that prompts the user to input a positive integer \( n \) and then converts it to binary. A sample header for such a method is given below.

```java
public static void convertToBinary(int n) {
    ...
}
```

Here, the integer \( n \) is the (decimal) number to be converted. The method `convertToBinary` must call itself. In other words, it must be recursive.

How can we do the needed conversion? Here is a way to do the conversion. Let us suppose that we want to convert the number 234 to a binary number.
Consider the following sequence of computations.

\[
\begin{align*}
234 &= 117 \times 2 + 0 \\
117 &= 58 \times 2 + 1 \\
58 &= 29 \times 2 + 0 \\
29 &= 14 \times 2 + 1 \\
14 &= 7 \times 2 + 0 \\
7 &= 3 \times 2 + 1 \\
3 &= 1 \times 2 + 1 \\
1 &= 0 \times 2 + 1
\end{align*}
\]

The number 11101010 is the binary representation of the decimal number 234. The sequence of remainders (read bottom to top) constitutes the binary number! Notice that the quotient in the first line, 117, is used as the dividend in the second line and so on. Obviously, these computations can be done recursively!

### 1.2 Teddy Bear Game

This method involves a game with Teddy bears. The game starts when you are given \( n \) Teddy bears. You can then give back some bears, but you must follow these rules where \( n \) is the number of bears that you have.

- If \( n \) is even, then you *may* give back exactly \( n/2 \) bears.
- If \( n \) is divisible by 3 or 4, then you *may* multiply the last two digits of \( n \) and give back this many bears. By the way, the last digit of \( n \) is \( n \% 10 \), and the next-to-last digit is \( (n \% 100)/10 \).
- If \( n \) is divisible by 5, then you *may* give back exactly 42 bears.

The goal of the game is to end up with EXACTLY 42 bears.

For example, suppose that you start with 250 bears. Then you could make the following moves:

- Start with 250 bears.
- Since 250 is divisible by 5, you *may* return 42 of the bears, leaving you with 208 bears.
Since 208 is even, you *may* return half of the bears, leaving you with 104 bears.

Since 104 is even, you *may* return half of the bears, leaving you with 52 bears.

Since 52 is divisible by 4, you *may* multiply the last two digits (resulting in 10) and return these 10 bears. This leaves you with 42 bears.

You have reached the goal!

The situation is more complicated than it looks! For example, suppose \( n \) is divisible by 60 (for example, \( n = 180 \)). In this case, \( n \) is even, \( n \) is divisible by both 3 and 4, and \( n \) is divisible by 5. Therefore, you can take *one of three actions*: give back exactly \( n/2 \) bears, give back \( k \) bears where \( k \) is the product of the last two digits of \( n \), or give back exactly 42 bears!

Write a recursive method to decide whether it is possible to win the Teddy bear game if you start with \( n \) bears. *You do not have to figure out the exact sequence of decisions.* Just determine whether it is possible or not.

```java
public static boolean teddyBearGame(int n) {
    ...
}
```

As an example, your method should return

- *true* for \( n = 250 \),
- *true* for \( n = 42 \),
- *false* for \( n = 53 \),
- *false* for \( n = 41 \),
- etc.

### 1.3 A Fractal Image

Examine the following pattern of asterisks and blanks, and write a recursive method that can generate such patterns.
The longest line has \( n \) asterisks and is a power of two. For example, \( n \) might be 1, 2, 4, 8, 16, etc.

```java
public static void fractal(int n, int i) {
    ...
}
```

The parameter \( n \) is the number of asterisks to be printed in the longest line and \( i \) is the column number where the asterisks should start printing. (The leftmost column is column number 0.) Therefore, the pattern above results from the method call `RecursiveMethods.fractal(8,0);` (`RecursiveMethods` is the name of the class that will contain the recursive methods you will write.)

### 1.4 A Letter Pattern

Write a recursive method that prints out a pattern of letters using the following rules. (The method has one formal parameter \( c \) of type `char`.)

- If the parameter \( c \) is 'A', then the output is 'A'.
- For other values of \( c \), the output consists of three parts:
  - the output for the previous letter (for example, if \( c \) is 'D', then the previous letter is 'C'),

```java

```
followed by the letter c itself,
followed by a second copy of the output for the previous letter.

For example, if c = 'D', then the method will print ABACABADABACABA.

```java
public static void letterPattern(char c) {
    ...
}
```

### 1.5 Longest Common Subsequence

Write a recursive method that has two `String` objects for its formal parameters and returns the length of the longest common subsequence of the two strings. This is a problem that has been of interest to biologists who try to measure the closeness (or homology) of two sequences (for example, RNA or DNA). For example, if the two strings are AZBBACXCABACCC and CBBACCCAABBCXYZ, then the longest common subsequence (BBACCABC) has length 8. Note that the method does not actually produce a longest common subsequence, but only measures its length.

How might this method be written? Suppose that `str1` and `str2` are given. If either string has length 0 (or is null), then the method should return 0. Next, suppose that `str1` and `str2` each has length at least one. If `str1.charAt(0)` equals `str2.charAt(0)`, then the length of a longest common subsequence of `str1` and `str2` is one more than the length of a longest common subsequence of `str1.substring(1)` and `str2.substring(1)`. If `str1.charAt(0)` is different from `str2.charAt(0)`, then the length of the longest common subsequence of `str1` and `str2` is the larger of the lengths of a longest common subsequence of the pair `str1` and `str2.substring(1)` and of the pair `str2` and `str1.substring(1)`.  

```java
public static int longComSubSeq(String str1, String str2) {
    ...
}
```

### 2 What To Turn In

Please submit your `.java` source file in `e:\submit\DelGreco\Comp170\Prog7\<yourFolder>`. All you have to submit is the class that contains
your recursive methods. Dr. Del Greco will write a driver class to test your recursive methods. Therefore, you will be submitting only one file.

3 Late Program Policy

A program will lose 10% of its value each day it is late (excluding weekends). Starting early on your programs will maximize your chances of earning full credit on your work!