Module 2: Recursion Lab

In this lab you will look at how recursion can be used to:

1. write the characters of a String in reverse order
2. very inefficiently calculate Fibonacci numbers
3. draw a ruler and a fractal star

Installation Instructions

Step 1

Download the RecursionDemo.zip file containing the source files for the demo. There should be 7 source files:

1. Driver.java (contains the "main" method)
2. FractalStar.java
3. GUI.java (the user interface)
4. RecursionStackDemo.java
5. RecursionTest.java
6. Ruler.java
7. Strings.java

Step 2

Create a Java project in the Java development environment you are using; e.g., Eclipse, JCreator, BlueJ, etc.

Step 3

Compile the project.

Step 4

Run the Recursion Demo program by executing the Driver class.

Running the Demo:

Writing a String in Reverse Order

The quickest way to do this would be to use a loop that goes from the last character to the first character. The "Strings.java" file shows a recursive algorithm.

Enter a sequence of characters in the text box provided. Click the "Reverse" button to recursively write the reverse of the entered sequence; click the "Reverse Loop" button to do the same thing using a loop instead of recursion. The result will appear below the text box.

Both algorithms will probably have comparable running times. The point here is just to show that the problem can be solved both recursively and iteratively (i.e., using a loop).

Fibonacci Example

1. Enter the integer '40' in the text box for the Fibonacci example. Click the "Recursive Algorithm" button to use the recursive algorithm shown below. You may have to wait several seconds before the result is displayed, depending on how fast your computer is.

```
public long fibRecursive(int n)
{
    if (n <= 1)
        return n;
    else
        return fibRecursive(n - 1) + fibRecursive(n - 2);
}
```

2. Now click the "Linear Algorithm" button to use a loop-based iterative algorithm. You should get the result immediately. Why does the recursive algorithm take so long? To find out, run the Stack Demo, described later on this page. The code for the linear algorithm is shown below. Note that it is not nearly as elegant as the recursive algorithm, although it is much faster.

```
public long fibLinear(int n)
{
    long nMinusTwo = 0L;
    long nMinusOne = 1L;
    long sum = 1L;
    int i;

    if (n == 0)
        sum = n;
    else
        for (i = 2; i <= n; i++)
        {
            sum = nMinusTwo + nMinusOne;
            nMinusTwo = nMinusOne;
            nMinusOne = sum;
        }

    return sum;
}
```

3. Click the "Dyn. Recursive Algorithm" button. This uses a recursive algorithm that utilizes dynamic programming to make sure each Fibonacci number only gets calculated once. It should run comparably to the linear algorithm. Here is the code for the dynamic recursive algorithm (assume the array, `knownFib[]`, has been declared outside the method):

```
public long dynamicFibRecursive(int n)
{
    if (knownFib[n] != 0)
        return knownFib[n];

    long sum = n;
    if (n < 0)
        return 0;

    if (n > 1)
        sum = dynamicFibRecursive(n - 1) + dynamicFibRecursive(n
- 2);
```

```
        knownFib[n] = sum;

        return knownFib[n];
    }
```

Stack Demo

Click the "Stack Demo" button. A new window should appear. This part of the lab serves two purposes: 1) to show you why the recursive Fibonacci algorithm is so inefficient, and 2) to illustrate how a stack can be used instead of the box trace technique to trace recursion.

The demo has two modes, which are toggled by the "Pause" button in the lower right corner. When the button's caption says "Pause On", you will be able to see how each recursive call adds a new activation record to the stack. A dialog box appears to let you proceed. When the button's caption says "Pause Off", the stack is not displayed but the algorithm runs faster.

Example:

1. Enter the number 5 for *N*.
2. Make sure the Pause button says "Pause On", and click the "Start" button.
3. Click "OK" on the dialog box that comes up and you should see the first activation record appear on the left side of the screen. The activation record shows the current value of *N* and what the return value will be. The return value is shown as an expression to make it clear which recursive calls have been fully evaluated.
4. Keep clicking on the dialog box until the activation record for *N* = 1 appears. Note that this is a base case, so the literal value 1 is returned.
5. Click on the dialog box until the topmost activation record disappears, to indicate the return from the base case to the next activation record on the stack. Note that the "fib(n - 1)" term is replaced with the literal value 1. This indicates the first of the 2 recursive calls has been evaulated, so now the "fib(n - 2)" term can be evaluated.
6. Continue until the algorithm is finished and the results appear in the statistics window. This window shows how many times the *N*th Fibonacci number was calculated for 0 <= *N* <= 5. Notice that the first Fibonacci number was calculated 5 times. This illustrates how violation of the 4th fundamental rule of recursion is violated.
7. Now click the Pause button and make sure the button reads "Pause Off". Enter the value 40 for *N* and click the "Start" button. The stack should not be displayed, and in a few seconds the results of the algorithm should appear. If you scroll up you will see that for *N* = 40 the first Fibonacci number is calculated a whopping 102,334,155 times!

Drawing a Ruler and a Fractal

These examples are more visual than calculating Fibonacci numbers. The source code for the algorithms is in the `Ruler.java` and `FractalStar.java` files, if you wish to see the code.

In the main window of the recursion demo, click the "Ruler" button. A new window will appear and you should see a ruler being drawn recursively, although pretty slowly. Obviously, this could be done much quicker using a simple loop to draw the ruler from left to right.

Next, click on the "Fractal" button to draw a square-based fractal image. A new window will appear, but nothing will happen for a while...be patient, it may take several minutes for the recursion to reach the point where the first square can be drawn. How easy do you think it would be to write a non-recursive algorithm to draw the same image?