但重要的是要记住，Java提供的仅仅是遵循递归思想的一种尝试。不是所有的数学递归函数都能被有效地(或正确地)由Java的递归模拟来实现。上面例子说的是递归函数 f 应该只用几行就能表示出来，正如非递归函数一样。图1-2指出了函数 f 的递归实现。

```
1       public static int f( int x )
2       {
3           if( x == 0 )
4               return 0;
5           else
6               return 2 * f( x - 1 ) + x * x;
7       }
```

图1-2 一个递归方法

第3行和第4行处理**基准情况**(base case)，即此时函数的值可以直接算出而不用求助递归。正如 $f(x)=2f(x-1)+x^2$ 若没有 $f(0)=0$ 这个事实在数学上没有意义一样，Java的递归方法若无基准情况也是毫无意义的。第6行执行的是递归调用。

关于递归，有几个重要并且可能会被混淆的概念。一个常见的问题是：它是否就是循环推理(circular logic)？答案是：虽然我们定义一个方法用的是这个方法本身，但是我们并没有用方法本身定义该方法的一个特定的实例。换句话说，通过使用 $f(5)$来得到 $f(5)$的值才是循环的。通过使用 $f(4)$得到 $f(5)$的值不是循环的，当然，除非 $f(4)$的求值又要用到对 $f(5)$的计算。两个最重要的问题恐怕就是如何做和为什么做的问题了。如何和为什么的问题将在第3章正式解决。这里，我们将给出一个不完全的描述。

实际上，递归调用在处理上与其他调用没有什么不同。如果以参数4的值调用函数 f，那么程序的第6行要求计算 $2*f(3)+4*4$。这样，就要执行一个计算 $f(3)$的调用，而这又导致计算 $2*f(2)+3*3$。因此，又要执行另一个计算 $f(2)$的调用，而这意味着必须求出 $2*f(1)+2*2$ 的值。为此，通过计算 $2*f(0)+1*1$ 而得到 $f(1)$。此时，$f(0)$必须被赋值。由于这属于基准情况，因此我们事先知道 $f(0)=0$。从而 $f(1)$的计算得以完成，其结果为1。然后，$f(2)$、$f(3)$以及最后 $f(4)$的值都能够计算出来。跟踪挂起的函数调用(这些调用已经开始但是正等待着递归调用来完成)以及它们的变量的记录工作都是由计算机自动完成的。然而，重要的问题在于，递归调用将反复进行直到基准情形出现。例如，计算 $f(-1)$的值将导致调用 $f(-2)$、$f(-3)$等等。由于这将不可能出现基准情形，因此程序也就不可能算出答案。偶尔还可能发生更加微妙的错误，我们将其展示在图1-3中。图1-3中程序的这种错误是第6行上的bad(1)定义为bad(1)。显然，实际上bad(1)究竟是多少，这个定义给不出任何线索。因此，计算机将会反复调用bad(1)以期解出它的值。最后，计算机簿记系统将占满内存空间，程序崩溃。一般情形下，我们会说该方法对一个特殊情形无效，而在其他情形是正确的。但此处这么说则不正确，因为bad(2)调用bad(1)。因此，bad(2)也不能求出值来。不仅如此，bad(3)、bad(4)和bad(5)都要调用bad(2)，bad(2)算不出值，它们的值也就不能求出。事实上，除了0之外，这个程序对 n 的任何非负值都无效。对于递归程序，不存在像“特殊情形”这样的情况。

```
1       public static int bad( int n )
2       {
3           if( n == 0 )
4               return 0;
5           else
6               return bad( n / 3 + 1 ) + n - 1;
7       }
```

图1-3 无终止递归方法