4. Recursion :: Recursive Factorial Method

In programming, recursion is involved when a method calls itself:

```
public T recursiveMethod(params) {
  ...
  T result = recursiveMethod(params);
  ...
}
```

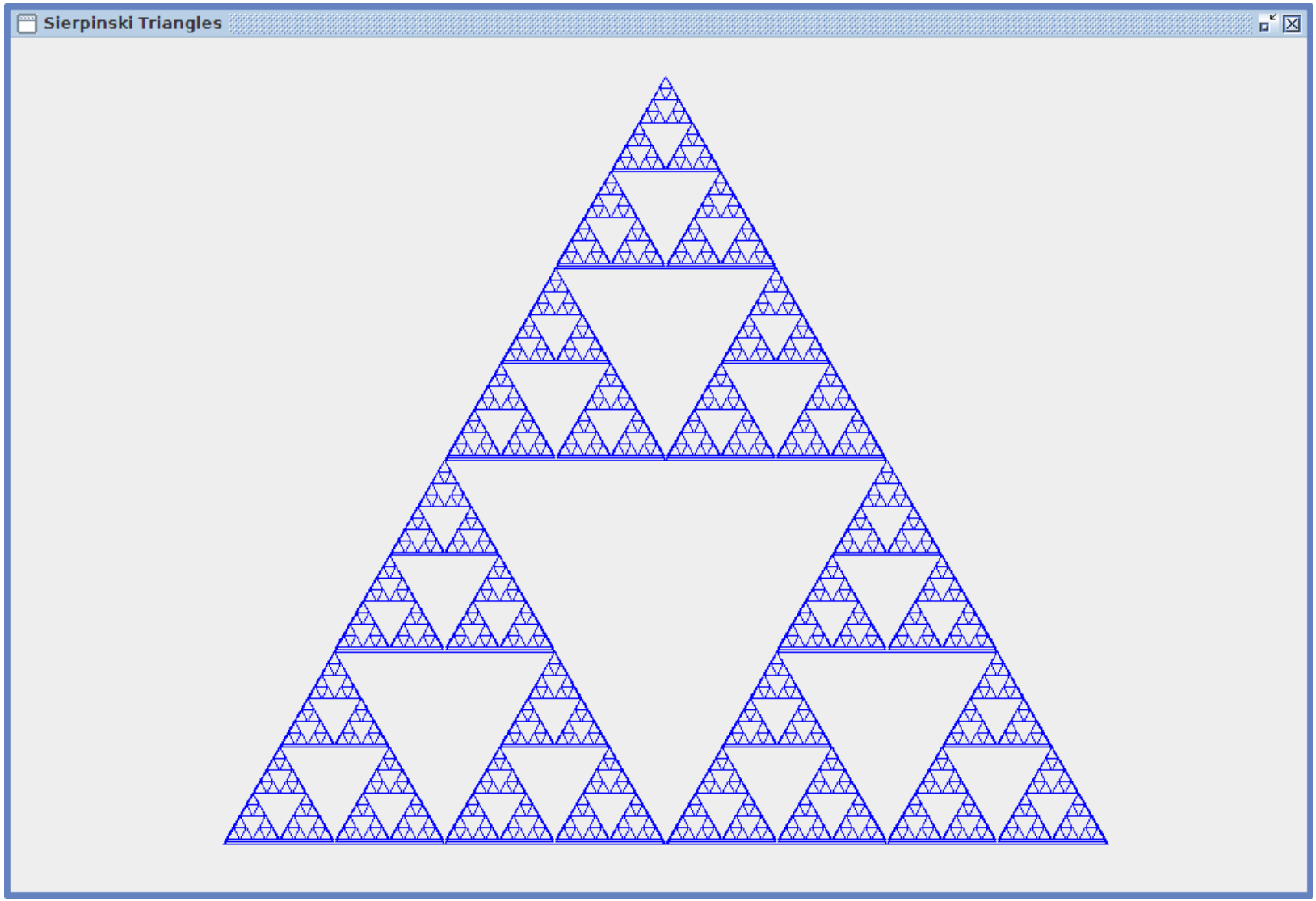
Therefore, we can write a recursive method to compute $n!$:

```
public long fact(long n) {
  // The base case is n = 1 and since 1! = 1, we return 1.
  if (n == 1) {
    return 1;
  // Otherwise, n > 1 so we first determine (n-1)! and then
  // multiply n by (n-1)!.
  } else {
    return n * fact(n-1);
  }
}
```

Lines 3–4 implement the base case: when the input n is 1 we return 1 (since $1! = 1$). Otherwise, we reduce the size of our problem to n - 1, pass n - 1 as the parameter to a recursive call to *fact*(), wait for *fact*(n - 1) to return the value of $(n$ - 1)! and then calculate and return $n!$ as $n \times (n$ - 1)!.

4. Recursion :: The Sierpinski Triangle

Around 1916 a Polish mathematician Waclaw Sierpinski described a recursive mathematical structure (a fractal) which is now called Sierpinski's Triangle. It looks like this:



4. Recursion :: The Sierpinski Triangle (continued)

It is pretty easy to draw one of these:

1. Draw an equilateral triangle
2. Within the equilateral triangle, draw three smaller equilateral triangles.
3. For each of the three smaller equilateral triangles: goto Step 2.

Of course, recursion cannot occur indefinitely, i.e., there must be a base case. We can define the base case for the Sierpinski Triangle by specifying the size of the smallest equilateral triangle we wish to draw.

4. Recursion :: The Sierpinski Triangle (continued)

Here is pseudocode for a recursive method that draws the Sierpinski Triangle:

```
public void drawSierpinskiTriangle(SierpinskiTriangle pTriangle, int pHeight) {
   // The base case is when the height of the triangle we are asked to draw is less than
   // 5 pixels. In this problem, we are simply drawing triangles, so there is no "solution"
   // to be determined and returned.
   if (pHheight <= 5) { return; }

   // Draw the larger triangle.
   myFrame.draw(pTriangle);

   // Reduce the size of the problem to three smaller equilateral triangles.
   int newHeight = some-new-height-which-is-less-than-pHeight;
   SierpinskiTriangle a = new SierpinksiTriangle(determine-location-of-a);
   SierpinskiTriangle b = new SierpinksiTriangle(determine-location-of-b);
   SierpinskiTriangle c = new SierpinksiTriangle(determine-location-of-c);

   // Recursively draw each of the three smaller equilateral triangles.
   drawSierpinskiTriangle(a, newHeight);
   drawSierpinskiTriangle(b, newHeight);
   drawSierpinskiTriangle(c, newHeight);
}
```