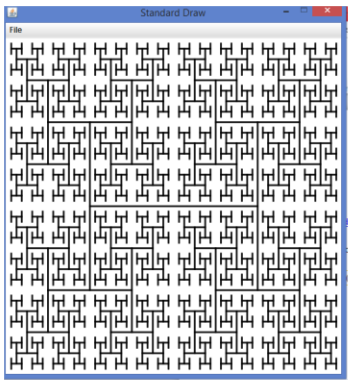
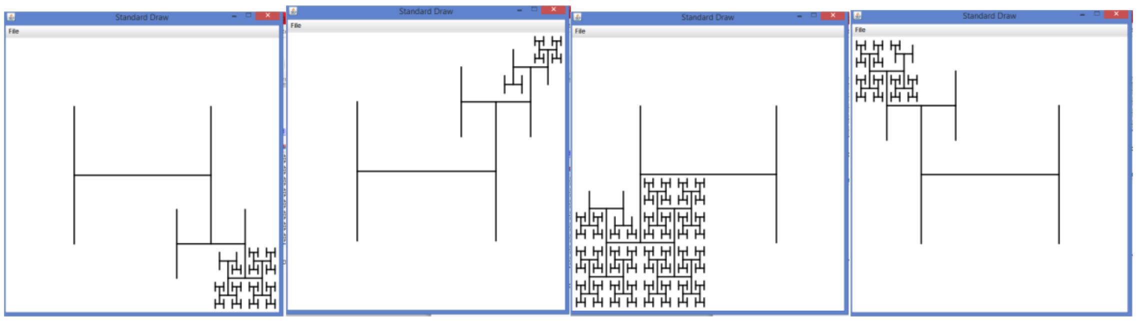
Ivan Lizcano Activity 4

Exercises 14;

Write a program AnimatedHtree.java that animates the drawing of the H-tree.



Next, rearrange the order of the recursive calls (and the base case), view the resulting animation, and explain each outcome



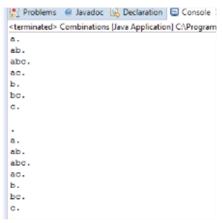
The recursivity start of the position Depending start the recursivity, of the first recursivity is from lower left, this draw all level to lower left ... and so with every turn.

Creative Exercises: 19, 22, 33;

19. **Combinations.** Write a program Combinations.java that takes one command-line argument n and prints out all 2^n *combinations* of any size. A combination is a subset of the n elements, independent of order. As an example, when n = 3 you should get the following output.

```
         a  ab  abc  ac  b  bc  c
```

Note that the first element printed is the empty string (subset of size 0).

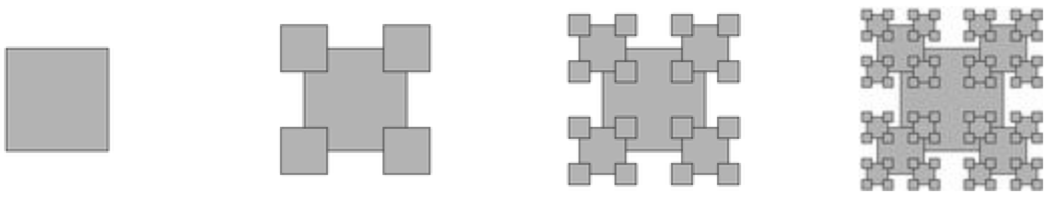


```
<terminated> Combinations [Java Application] C:\Program
a.
ab.
abc.
ac.
b.
bc.
c.

.
a.
ab.
abc.
ac.
b.
bc.
c.
```

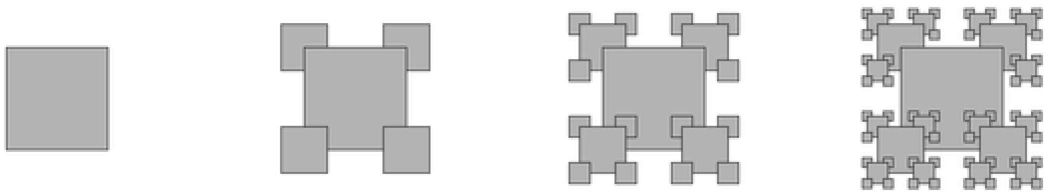
```
a.
ab.
abc.
ac.
b.
bc.
c.

.
a.
ab.
abc.
ac.
b.
bc.
c.

I adde a dot, and is more easy identify the blank positions.
```

22. Recursive squares. Write a program to produce each of the following recursive patterns. The ratio of the sizes of the squares is 2.2. To draw a shaded square, draw a filled gray square, then an unfilled black square.

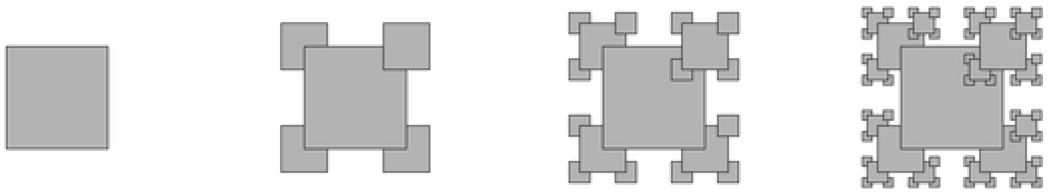
1.



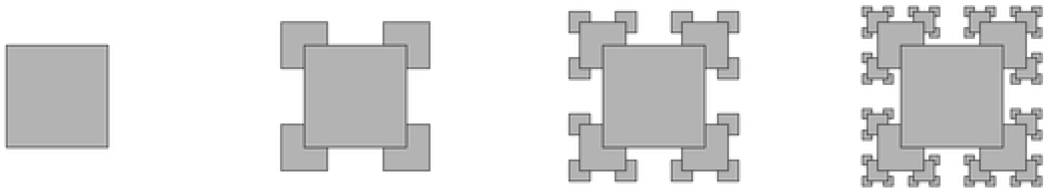
2.



3.

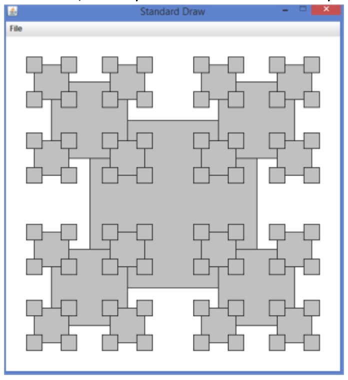


4.

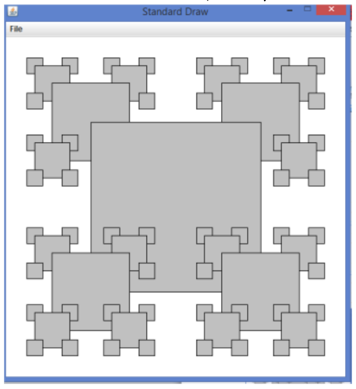


RecursiveSquares.java gives a solution to part a.

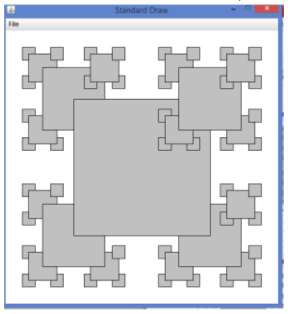
```
For case 1
       drawSquare(x, y, size);
       draw(n-1, x - size/2, y - size/2, size/ratio);    // lower left
       draw(n-1, x - size/2, y + size/2, size/ratio);    // upper left
       draw(n-1, x + size/2, y - size/2, size/ratio);    // lower right
       draw(n-1, x + size/2, y + size/2, size/ratio);    // upper right
```



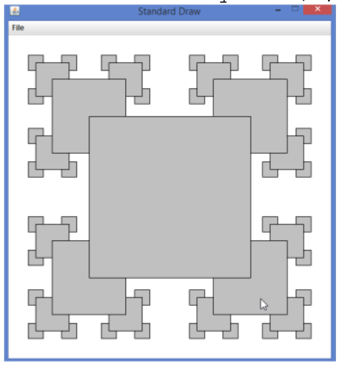
```
For Case 2
         draw(n-1, x - size/2, y + size/2, size/ratio);    // upper left
         draw(n-1, x + size/2, y + size/2, size/ratio);    // upper right
         drawSquare(x, y, size);
         draw(n-1, x - size/2, y - size/2, size/ratio);    // lower left
         draw(n-1, x + size/2, y - size/2, size/ratio);    // lower right
```



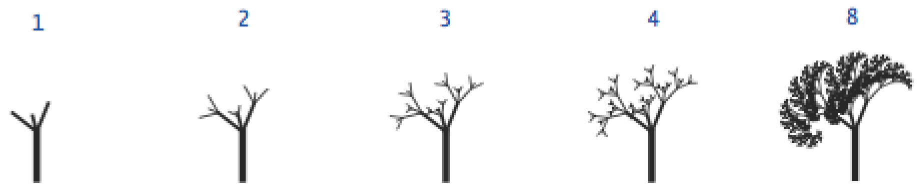
```
For case 3
            draw(n-1, x - size/2, y + size/2, size/ratio);    // upper left
            draw(n-1, x + size/2, y - size/2, size/ratio);    // lower right
            draw(n-1, x - size/2, y - size/2, size/ratio);    // lower left
            drawSquare(x, y, size);
            draw(n-1, x + size/2, y + size/2, size/ratio);    // upper right
```



```
For case 4
          draw(n-1, x - size/2, y + size/2, size/ratio);    // upper left
          draw(n-1, x + size/2, y + size/2, size/ratio);    // upper right
          draw(n-1, x - size/2, y - size/2, size/ratio);    // lower left
          draw(n-1, x + size/2, y - size/2, size/ratio);    // lower right
          drawSquare(x, y, size);
```



33. Recursive tree. Write a program Tree.java that takes a command-line arguemnt `N` and produces the following recurisve patterns for `N` equal to 1, 2, 3, 4, and 8.



Great… apparently the tree grew up in 10 seconds …

Web Exercises: 2, 10, 11, 20, 39,42.

2 Write a recursive program GoldenRatio.java that takes an integer input N and computes an approximation to the golden ratio using the following recursive formula:

```
      f(N) = 1                 if N = 0
           = 1 + 1 / f(N-1)    if N > 0
```

Redo, but do not use recursion.

With recursion:

```
2.0
1.5
1.6666666666666665
1.6
1.625
1.6153846153846154
1.619047619047619
1.6176470588235294
1.6181818181818182
1.6179775280898876
1.6180555555555556
1.6180257510729614
1.6180371352785146
1.6180327868852458
```

```
With loop
```

```
1.0
2.0
1.5
1.6666666666666667
1.6
1.625
1.6153846153846154
1.619047619047619
1.6176470588235294
1.6181818181818182
1.6179775280898876
1.6180555555555556
1.6180257510729614
```

10 Write a program Fibonacci2.java that takes a command-line argument N and prints out the first N Fibonacci numbers using the following alternate definition:

```
F(n)    = 1                                  if n = 1 or n = 2
        = F((n+1)/2)^2 + F((n-1)/2)^2        if n is odd
        = F(n/2 + 1)^2 - F(n/2 - 1)^2        if n is even
```

What is the biggest Fibonacci number you can compute in under a minute using this definition? Compare this to Fibonacci.java.

```
1: 1 -  4.60682E-4 segundos.
2: 1 -  0.0036274260000000004 segundos.
3: 2 -  0.003749407 segundos.
4: 3 -  0.0038212520000000002 segundos.
5: 5 -  0.00420772 segundos.
6: 8 -  0.004375492 segundos.
7: 13 -  0.005187113 segundos.
8: 21 -  0.005334752000000001 segundos.
9: 34 -  0.005488313000000001 segundos.
10: 55 -  0.005632794 segundos.
11: 89 -  0.005706219 segundos.
12: 144 -  0.005784776 segundos.
13: 233 -  0.005878334000000001 segundos.
14: 377 -  0.005942679 segundos.
15: 610 -  0.006023999 segundos.
16: 987 -  0.006148348000000005 segundos.
17: 1597 -  0.006290855000000001 segundos.
18: 2584 -  0.0064270460000000005 segundos.
19: 4181 -  0.006679691000000001 segundos.
20: 6765 -  0.006913782 segundos.
21: 10946 -  0.007293539000000001 segundos.
22: 17711 -  0.007483812 segundos.
23: 28657 -  0.0077052710000000005 segundos.
24: 46368 -  0.008031341000000001 segundos.
25: 75025 -  0.008753352 segundos.
26: 121393 -  0.009572474000000001 segundos.
27: 196418 -  0.010717665000000001 segundos.
28: 317811 -  0.012552893 segundos.
29: 514229 -  0.015965966 segundos.
30: 832040 -  0.022424593000000003 segundos.
31: 1346269 -  0.031375323000000004 segundos.
32: 2178309 -  0.044055723000000005 segundos.
33: 3524578 -  0.063562663 segundos.
34: 5702887 -  0.093310401 segundos.
35: 9227465 -  0.140918545 segundos.
36: 14930352 -  0.217627026 segundos.
37: 24157817 -  0.34005829000000004 segundos.
38: 39088169 -  0.53703279 segundos.
39: 63245986 -  0.8529374500000001 segundos.
40: 102334155 -  1.412167057 segundos.
41: 165580141 -  2.2561143 segundos.
42: 267914296 -  3.617142501 segundos.
```

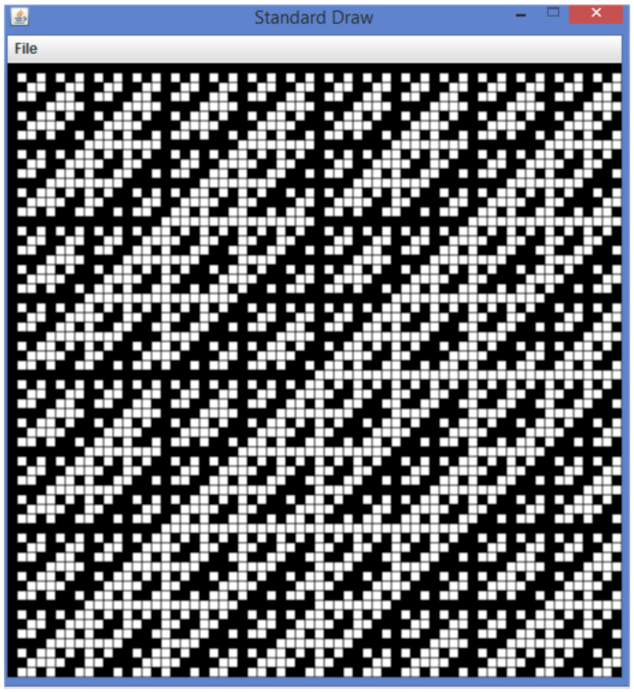
```
43: 433494437 -  5.8017259590000005 segundos.
44: 701408733 -  9.476293038000001 segundos.
45: 1134903170 -  15.174474114 segundos.
46: 1836311903 -  24.413780825 segundos.
47: 2971215073 -  39.323409394 segundos.
```
48: 4807526976 - 63.454599142000006 segundos.

Fibbonacci 4807526976 en 1 min

11 Write a program that takes a command-line argument N and prints out the first N Fibonacci numbers using the following method proposed by Dijkstra:

```
F(0) =  0
F(1) =  1
F(2n-1) = F(n-1)^2 + F(n)^2
F(2n) = (2F(n-1) + F(n)) * F(n)

0: 0.0
1: 1.0
2: 1.0
3: Infinity
4: 3.0
5: Infinity
```

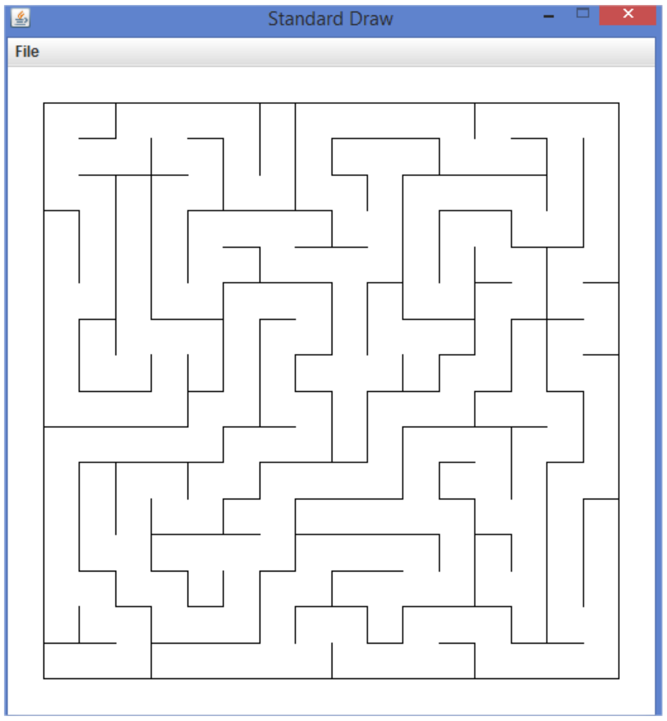
20. Hadamard matrix. Write a recursive program Hadamard.java that takes a command-line argument n and plots an N-by-N Hadamard pattern where N = 2^n. Do *not* use an array. A 1-by-1 Hadamard pattern is a single black square. In general a 2N-by-2N Hadamard pattern is obtained by aligning 4 copies of the N-by-N pattern in the form of a 2-by-2 grid, and then inverting the colors of all the squares in the lower right N-by-N copy. The N-by-N Hadamard H(N) matrix is a boolean matrix with the remarkable property that any two rows differ in exactly N/2 bits. This property makes it useful for designing *error-correcting* *codes*. Here are the first few Hadamard matrices.



39 **Tribonacci numbers.** The *tribonacci numbers* are similar to the Fibonacci numbers, except that each term is the sum of the three previous terms in the sequence. The first few terms are 0, 0, 1, 1, 2, 4, 7, 13, 24, 44, 81. Write a program to compute tribonacci numbers. What is the ratio successive terms? *Answer*. Root of x^3 - x^2 - x - 1, which is approximately 1.83929

```
0: 0 -  3.1304200000000003E-4 segundos.
1: 0 -  0.002328279 segundos.
2: 1 -  0.002420652 segundos.
3: 1 -  0.002484998 segundos.
4: 2 -  0.002549738 segundos.
5: 4 -  0.002610531 segundos.
6: 7 -  0.002706062 segundos.
7: 13 -  0.002781855 segundos.
```

42**Maze generation.** Create a maze using divide-and-conquer: Begin with a rectangular region with no walls. Choose a random gridpoint in the rectangle and construct two perpendicular walls, dividing the square into 4 subregions. Choose 3 of the four regions at random and open a one cell hole at a random point in each of the 3. Recur until each subregion has width or height 1.


Standard Draw
File