

Review

– Inheritance

- A relationship established between two classes
- Fields and methods of the superclass become available to all subclass by default
- Subclasses can replace (override) superclass members (fields and methods) by declaring new versions
- Inheritance implements the concept of subtype polymorphism
 - Objects of a subclass type can be assigned to variables declared as one of its superclass types

– Keywords

- extends, this, super

Function Call Tracing

```
void setup() {}
void draw() {}

void mousePressed() {
  println("Before paragraph.");
  printParagraph();
  println("After paragraph.");
}

void printParagraph() {
  println("Before sentence.");
  printSentence();
  println("After sentence.");
}

void printSentence() {
  println("Go!");
}
```

```
Before paragraph.
Before sentence.
Go!
After sentence.
After paragraph.
```

Old Lady Who Swallowed a Fly

There was an old lady who swallowed a fly.
I dunno why she swallowed that fly,
Perhaps she'll die.

There was an old lady who swallowed a spider,
That wiggled and wiggled and tickled inside her.
She swallowed the spider to catch the fly.
But I dunno why she swallowed that fly -
Perhaps she'll die.

There was an old lady who swallowed a bird;
How absurd, to swallow a bird!
She swallowed the bird to catch the spider
That wiggled and wiggled and tickled inside her.
She swallowed the spider to catch the fly.
But I dunno why she swallowed that fly -
Perhaps she'll die

There was an old lady who swallowed a cat.
Imagine that, she swallowed a cat.
She swallowed the cat to catch the bird ...
She swallowed the bird to catch the spider
That wiggled and wiggled and tickled inside her.
She swallowed the spider to catch the fly.
But I dunno why she swallowed that fly
Perhaps she'll die

There was an old lady who swallowed a dog.
What a hog! To swallow a dog!
She swallowed the dog to catch the cat...
She swallowed the cat to catch the bird ...
She swallowed the bird to catch the spider
That wiggled and wiggled and tickled inside her.
She swallowed the spider to catch the fly.
But I dunno why she swallowed that fly
Perhaps she'll die.

There was an old lady who swallowed a goat.
Just opened her throat and swallowed a goat!
She swallowed the goat to catch the dog ...
She swallowed the dog to catch the cat.
She swallowed the cat to catch the bird ...
She swallowed the bird to catch the spider
That wiggled and wiggled and tickled inside her.
She swallowed the spider to catch the fly.
But I dunno why she swallowed that fly
Perhaps she'll die.

There was an old lady who swallowed a cow.
I don't know how she swallowed a cow!
She swallowed the cow to catch the goat...
She swallowed the goat to catch the dog...
...

The "catching" goal of each animal is *Recursive* that is, it is defined in terms of itself

```
Cow – "I don't know how."
  ↳ Goat – "She just opened her throat."
    ↳ Pig – "Her mouth was so big."
      ↳ Snake – "What a mistake."
        ↳ Dog – "What a hog."
          ↳ Cat – "Fancy that."
            ↳ Bird – "Quite absurd."
              ↳ Spider – "It wriggled and jiggled and tickled inside her."
                ↳ Fly – "I don't know why."
```

- The data are different (Cow, Goat, Pig, ..., Fly)
- The function is the same (catch)
- The goal of each animal is nested within a larger animal's goal
- Fly is the "base case" after which no more catching takes place

Factorial

- The factorial of a positive integer N is computed as the product of N with all positive integers less than or equal to N.

$$4! = 4 \times 3 \times 2 \times 1 = 24$$

$$5! = 5 \times 4 \times 3 \times 2 \times 1 = 120$$

$$30! = 30 \times 29 \times \dots \times 2 \times 1 = 265252859812191058636308480000000$$

Factorial – Iterative Implementation

```
1. void setup() {
2.   int A = 10;
3.   int B = factorial(5);
4.   println( B );
5. }

6. int factorial(int N) {
7.   int F = 1;
8.
9.   for( int i=N; i>=1; i-- ) {
10.    F = F * i;
11.  }
12.
13.  return F;
14. }
```

Trace it.

FactorialIterative.pde

$$5! = 5 \times 4 \times 3 \times 2 \times 1$$

$$4! = 4 \times 3 \times 2 \times 1$$

$$5! = 5 \times 4!$$

$$N! = N \times (N-1)!$$

Factorial can be defined in terms of itself

$$4! = 4 \times 3 \times 2 \times 1 = 24$$

$$5! = 5 \times 4 \times 3 \times 2 \times 1 = 120$$

$$5! = 5 \times 4!$$

$$4! = 4 \times 3!$$

$$3! = 3 \times 2!$$

$$2! = 2 \times 1$$

Factorial – Recursive Implementation

```

1. void setup() {
2.   int A = 10;
3.   int B = factorial(5);
4.   println( B );
5. }

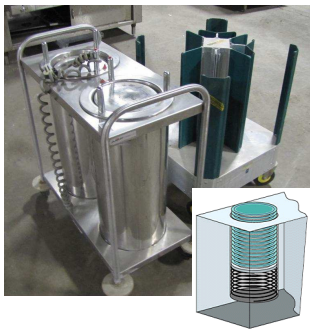
6. int factorial(int N) {
7.   int F;
8.   if (N == 1) {
9.     return 1;
10.  } else {
11.    F = N * factorial(N-1);
12.    return F;
13.  }
14. }

```

Trace it.

Factorial.pde

Last In First Out (LIFO) Stack of Plates



Compiled Code

```

1. void setup() {
2.   int A = 10;
3.   int B = factorial(5);
4.   println( B );
5. }

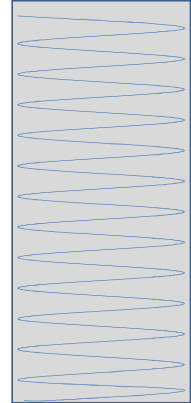
1. int factorial(int N) {
2.   if (N == 1) {
3.     return 1;
4.   } else {
5.     int F = N *
      factorial(N-1);
6.     return F;
7.   }
8. }

```

Executing Function



Call Stack



Compiled Code

```

1. void setup() {
2.   int A = 10;
3.   int B = factorial(5);
4.   println( B );
5. }

1. int factorial(int N) {
2.   if (N == 1) {
3.     return 1;
4.   } else {
5.     int F = N *
      factorial(N-1);
6.     return F;
7.   }
8. }

```

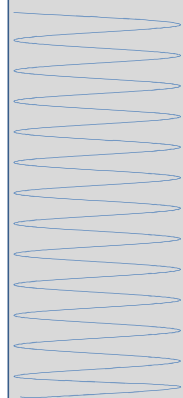
Executing Function

```

1. void setup() {
2.   int A = 10;
3.   int B = factorial(5);
4.   println( B );
5. }

```

Call Stack



Compiled Code

```

1. void setup() {
2.   int A = 10;
3.   int B = factorial(5);
4.   println( B );
5. }

1. int factorial(int N) {
2.   if (N == 1) {
3.     return 1;
4.   } else {
5.     int F = N *
      factorial(N-1);
6.     return F;
7.   }
8. }

```

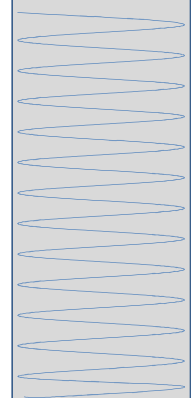
Executing Function

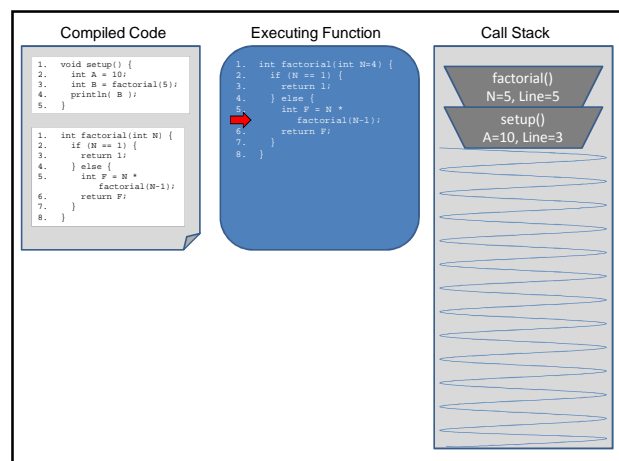
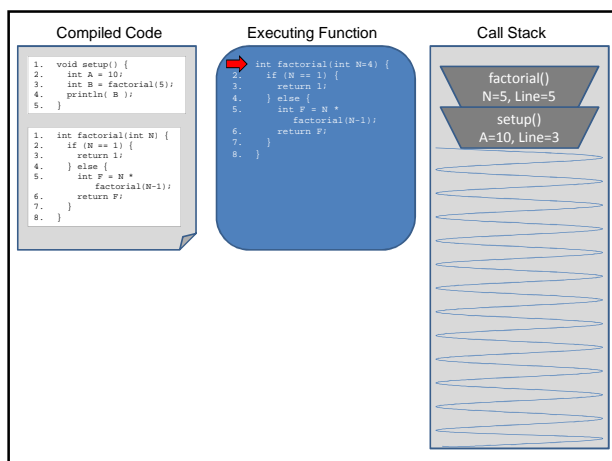
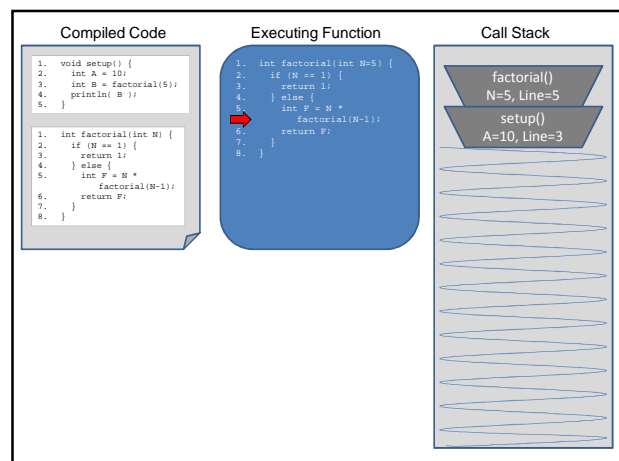
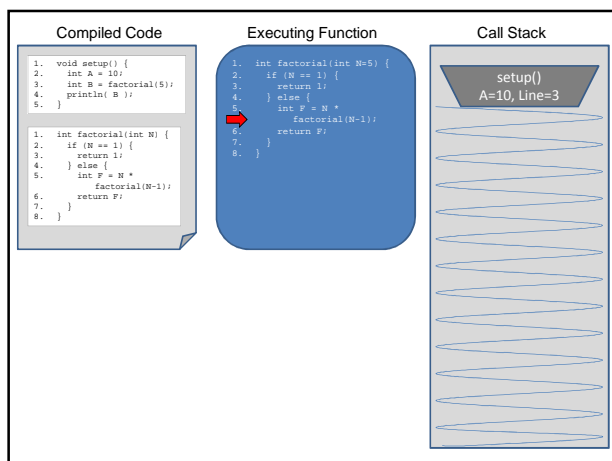
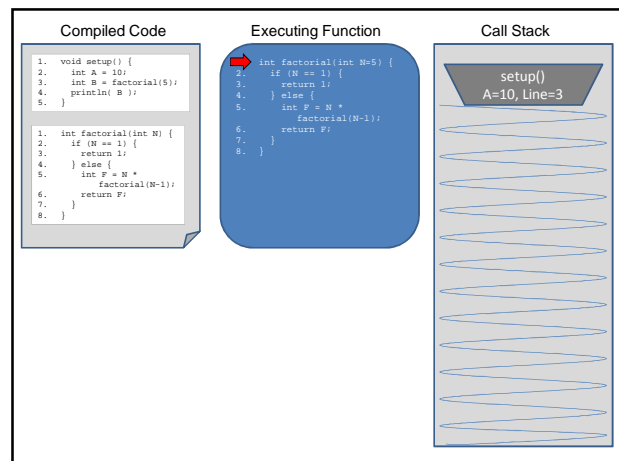
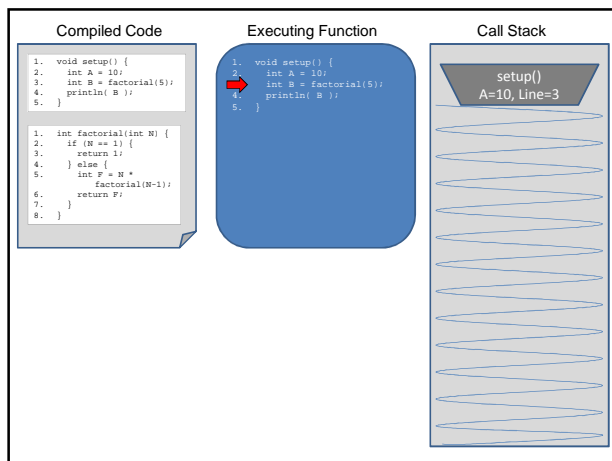
```

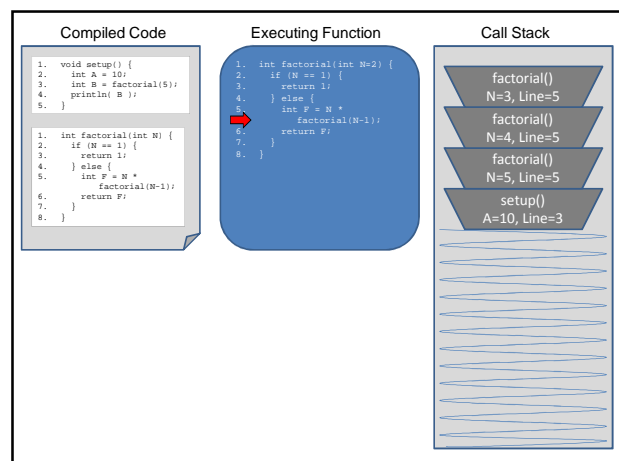
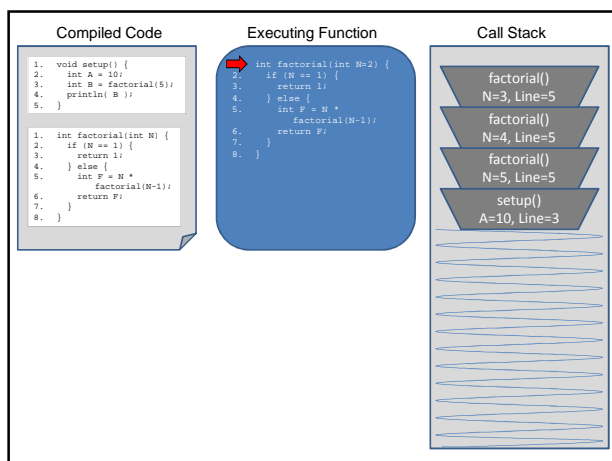
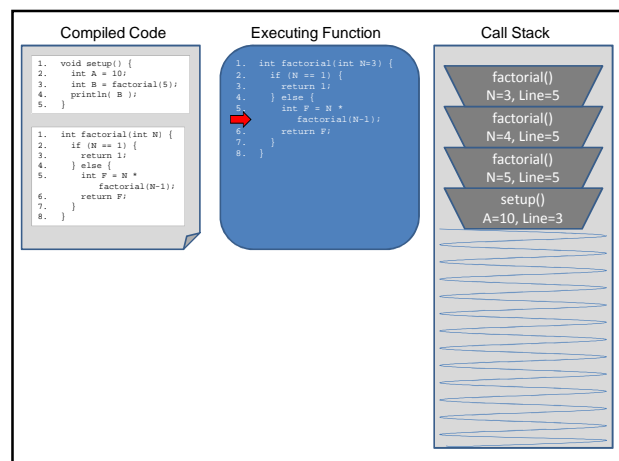
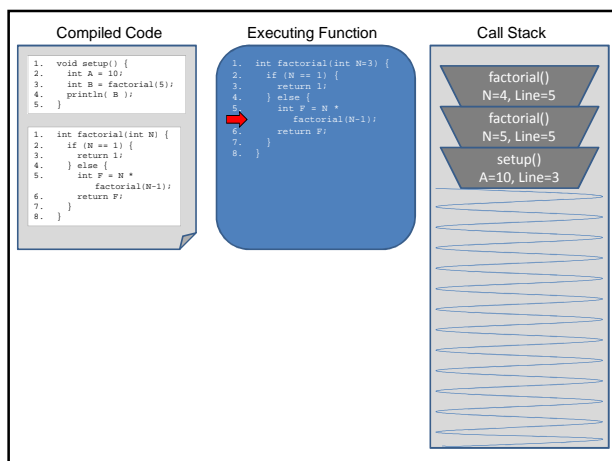
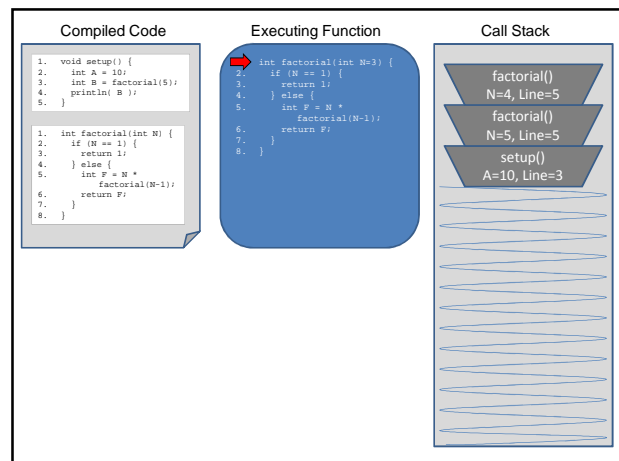
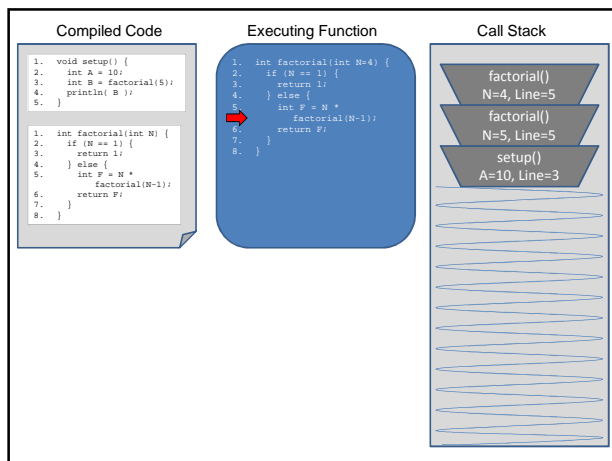
1. void setup() {
2.   int A = 10;
3.   int B = factorial(5);
4.   println( B );
5. }

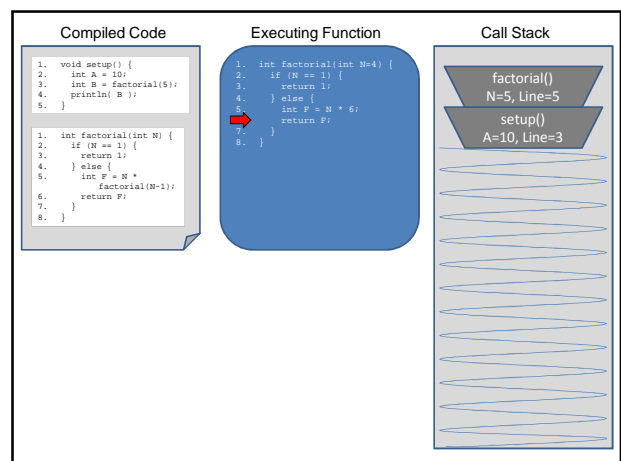
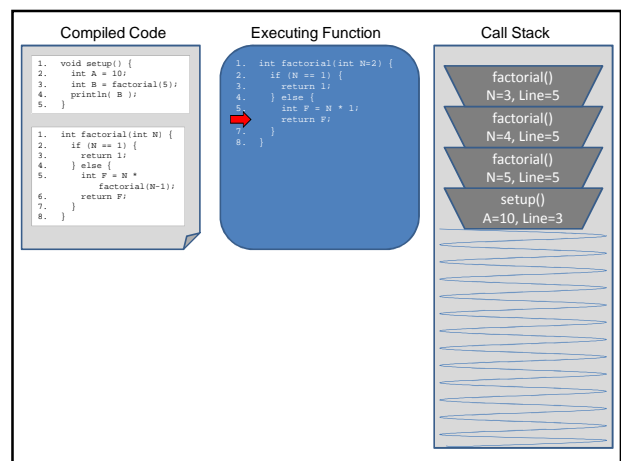
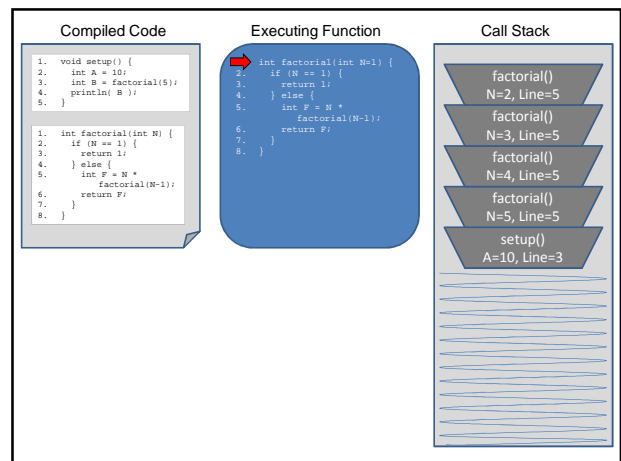
```

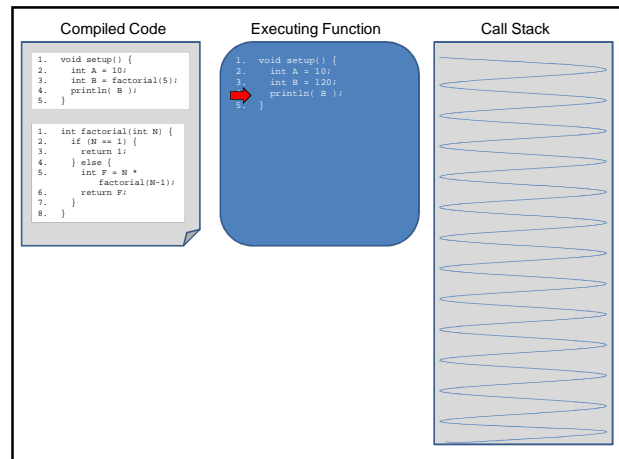
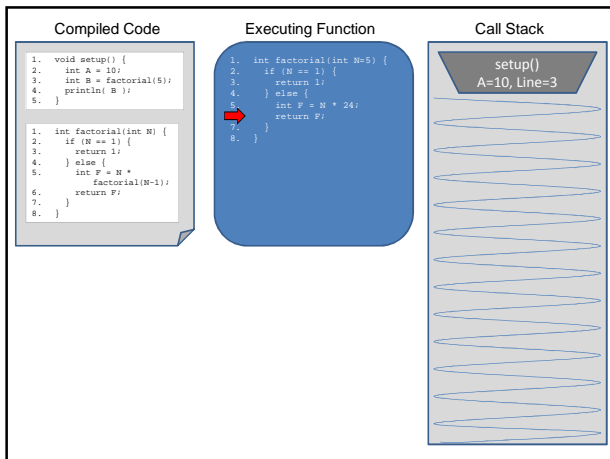
Call Stack







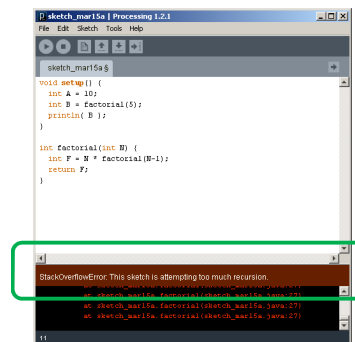




The Call Stack keeps track of ...

1. all functions that are suspended, in the reverse order in which they were suspended (LIFO)
2. the point in the function where execution should resume after the invoked subordinate function returns
3. a snapshot of all variables and values within the scope of the suspended function so these can be restored upon continuing execution

What happens if there is no stopping condition, or "base case"?



```
// Fibonacci sequence
// n = 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, ...
// f = 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, ...

void setup() {}
void draw() {}

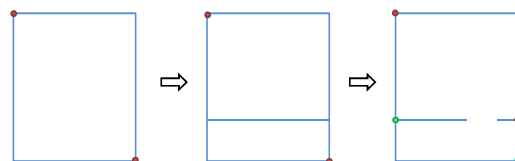
void mousePressed() {
    int f = fibonacci(12);
    println(f);
}

// Compute and return the nth Fibonacci number
int fibonacci(int n) {
    if (n == 0) {
        return 0;
    } else if (n == 1) {
        return 1;
    } else {
        int f = fibonacci(n-1) + fibonacci(n-2);
        return f;
    }
}
```

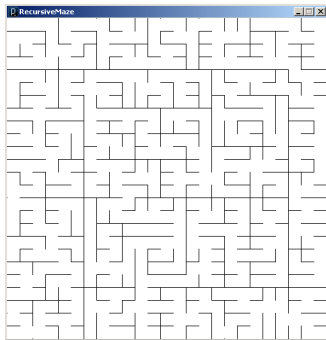
Fibonacci.pde

Creating a maze, recursively

1. Start with a rectangular region defined by its upper left and lower right corners
2. Divide the region at a random location through its more narrow dimension
3. Add an opening at a random location
4. Recursively, repeat on two rectangular subregions



Inspired by <http://weblog.jamisbuck.org/2011/1/12/maze-generation-recursive-division-algorithm>



See also StackMaze.pde

RecursiveMaze.pde

```
// RecursiveMaze

int N = 25; // Grid dimension
int gsize = 20; // Grid size

int V = 1; // Vertical constant
int H = 2; // Horizontal constant

void setup() { // Determine the direction for dividing
  // Setup sketch // Stop when too small.
  size(N*gsize+1, N*gsize+1);
  noLoop();
  background(255);
  stroke(0);

  // Kick off the recursive divide
  // on entire sketch window
  divide(0,0,N,N);
}

int divDir(int r1, int c1, int r2, int c2) {
  int dr = r2 - r1; // Deltas
  int dc = c2 - c1;
  if (dr <= 1 || dc <= 1) // Too small
    return 0; // No division
  else if (dr < dc) // Flat and wide
    return V; // Vertical division
  else // Tall and narrow
    return H; // Horizontal div
}

// Return a random integer in the range
int randomInt(int min, int max) {
  return round(random(min-0.5,max+0.5));
}

// Draw a line on a grid segment
void gridLine(int r1, int c1, int r2, int c2) {
  line(r1*gsize, c1*gsize, r2*gsize, c2*gsize);
}
```

```
// Divide the region given upper left and
// lower right grid corner points

void divide(int r1, int c1, int r2, int c2)
{
  int cr, rr;

  // Get divide direction (V, H or 0)
  int dir = divDir(r1, c1, r2, c2);

  // Divide in vertical direction // Divide in horizontal direction
  if (dir == V) {
    // Wall and opening locations
    cr = randomInt(c1+1, c2-1);
    rr = randomInt(r1, r2-1);

    // Draw wall
    gridLine(cr,r1,cr,rr);
    gridLine(cr,rr+1,cr,r2);

    // Recursively divide two subregions
    divide(r1,c1,r2,cr);
    divide(r1,cr,r2,c2);
  } else if (dir == H) {
    // Wall and opening locations
    cr = randomInt(c1, c2-1);
    rr = randomInt(r1+1, r2-1);

    // Draw wall
    gridLine(c1,rr,cr,rr);
    gridLine(cr+1,rr,c2,rr);

    // Recursively divide two subregions
    divide(r1,c1,rr,c2);
    divide(rr,c1,r2,c2);
  } else {
    // No division. We're done.
    return;
  }
}
```