Drawing/Animation

- Coordinate modification
  - No variables in the shape coordinates
  - variables added to x and y
  - trigs involving angle variable added to x and y
  - scale factor multiplied to x and y
- Transformations
  - No variables in the shape coordinates
  - shape is drawn centered on (0, 0)
  - translate
  - rotate
  - scale

Program Structure

```
Class Leaf{
```

- fields: `x`, `y`, `size`, `angle`, `spin` etc
- display()

```
pushMatrix();
translate(x, y);
rotate(angle);
scale(size);
// drawing …
popMatrix();
```

- move(): updates `x` and `y`
- spin(): updates `angle`

```
}
```

- `Leaf[] leaves = new Leaf[20];`
- `int idx = 0;`
- keyPressed()

```
if (key == 's') {
  spin = true;
  //spin = !spin;
}
```

- mousePressed()

```
leaves[idx] = new Leaf(mouseX, mouseY);
idx++;
```

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.

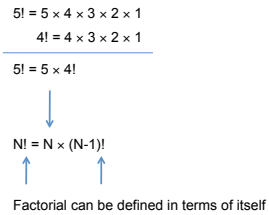
4! = 4 × 3 × 2 × 1 = 24

30! = 30 × 29 × … × 2 × 1 =
265252859812191058636308480000000

Factorial - Iterative Implementation

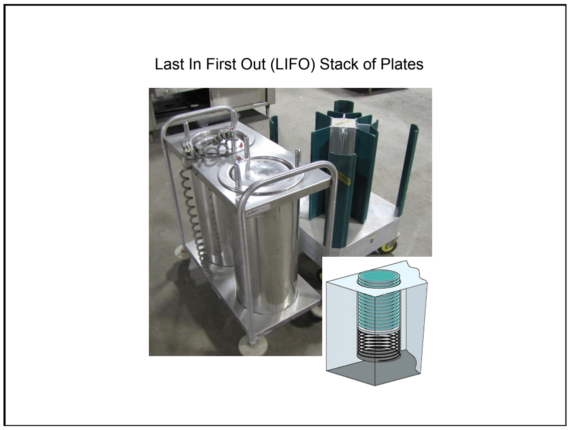
```
1.  void setup() {
2.    int A = 10;
3.    int B = factorial(5);
4.    println( B );
5.  }

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. }
```

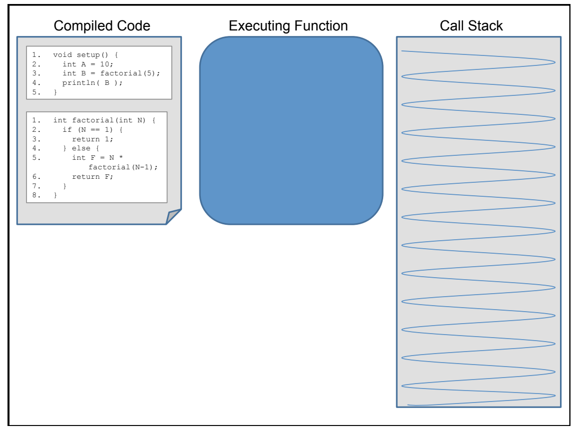
Trace it.

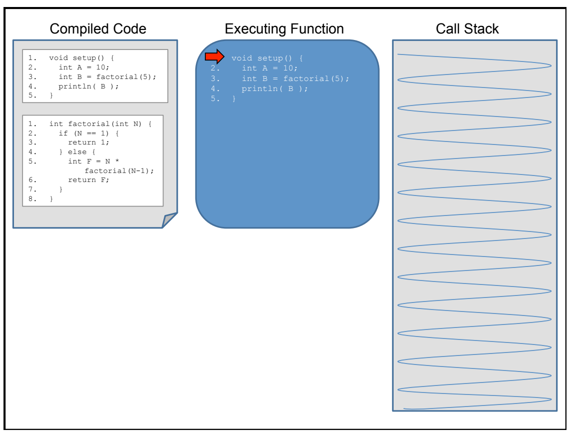


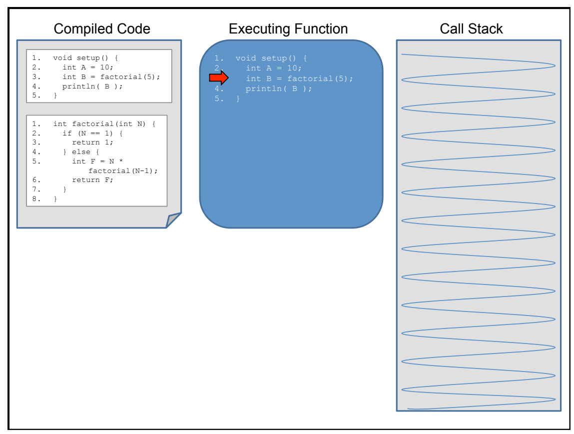
Factorial – Recursive Implementation

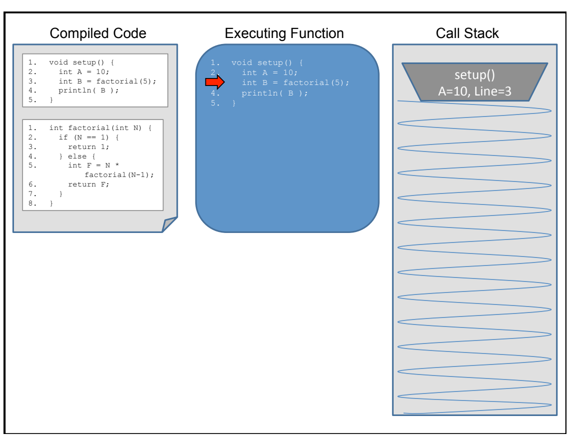
```
1.  void setup() {
2.    int A = 10;
3.    int B = factorial(5);
4.    println( B );
5.  }

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

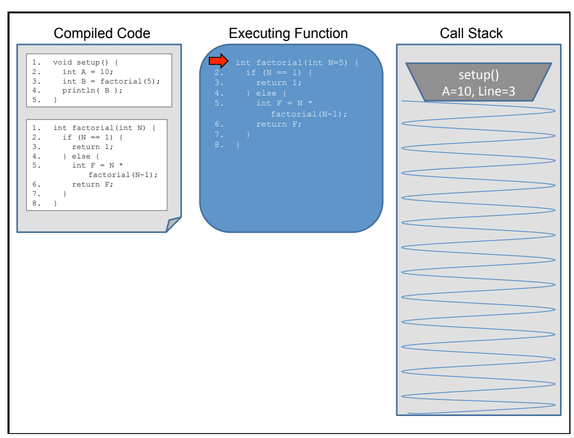

Last In First Out (LIFO) Stack of Plates

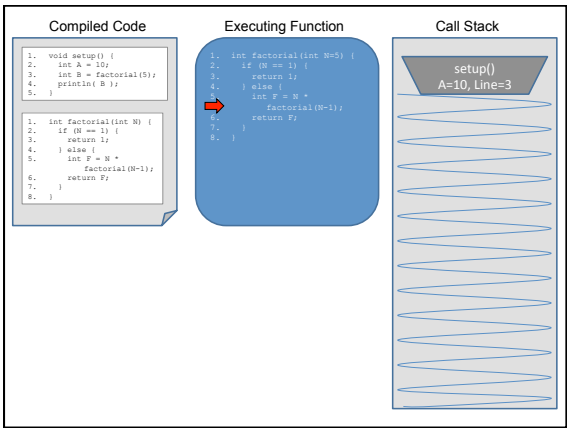

Compiled Code
Executing Function
Call Stack
1. void setup() {
2. int A = 10;
3. int B = factorial(5);
4. println(B);
5. }
1. int factorial(int N) {
2. if (N == 1) {
3. return 1;
4. } else {
5. int F = N *
factorial(N-1);
6. return F;
7. }
8. }

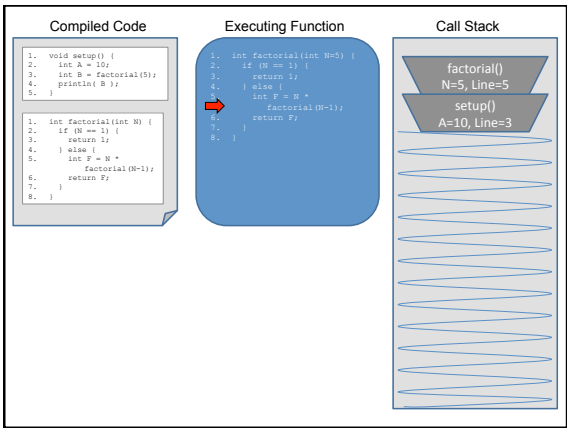

Compiled Code
Executing Function
Call Stack
1. void setup() {
2. int A = 10;
3. int B = factorial(5);
4. println(B);
5. }
1. int factorial(int N) {
2. if (N == 1) {
3. return 1;
4. } else {
5. int F = N *
factorial(N-1);
6. return F;
7. }
8. }
void setup() {
2. int A = 10;
3. int B = factorial(5);
4. println(B);
5. }

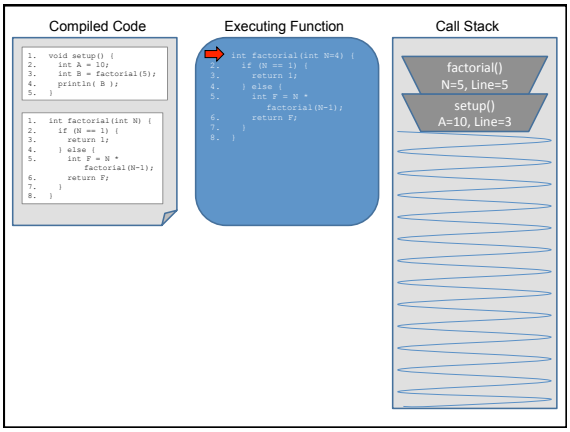

Compiled Code
Executing Function
Call Stack
1. void setup() {
2. int A = 10;
3. int B = factorial(5);
4. println(B);
5. }
1. int factorial(int N) {
2. if (N == 1) {
3. return 1;
4. } else {
5. int F = N *
factorial(N-1);
6. return F;
7. }
8. }
1. void setup() {
int A = 10;
3. int B = factorial(5);
4. println(B);
5. }

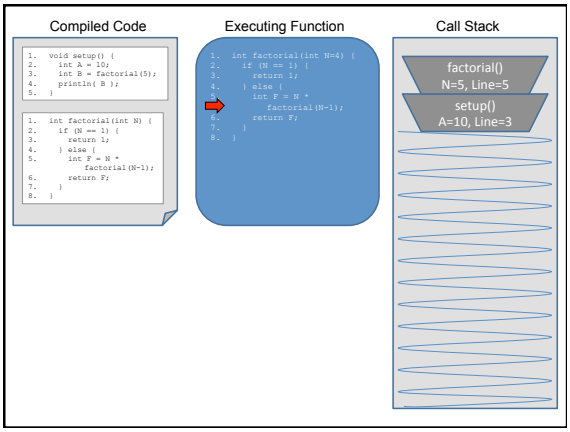

Compiled Code
Executing Function
Call Stack
1. void setup() {
2. int A = 10;
3. int B = factorial(5);
4. println(B);
5. }
1. int factorial(int N) {
2. if (N == 1) {
3. return 1;
4. } else {
5. int F = N *
factorial(N-1);
6. return F;
7. }
8. }
1. void setup() {
2. int A = 10;
int B = factorial(5);
4. println(B);
5. }
setup()
A=10, Line=3

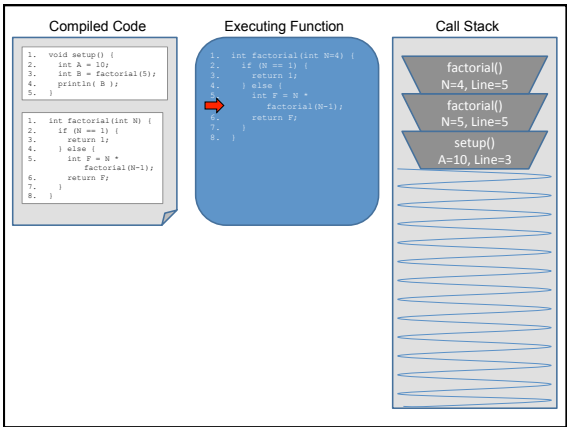

Compiled Code
Executing Function
Call Stack
1. void setup() {
2. int A = 10;
3. int B = factorial(5);
4. println(B);
5. }
1. int factorial(int N) {
2. if (N == 1) {
3. return 1;
4. } else {
5. int F = N *
factorial(N-1);
6. return F;
7. }
8. }
int factorial(int N=5) {
2. if (N == 1) {
3. return 1;
4. } else {
5. int F = N *
factorial(N-1);
6. return F;
7. }
8. }
setup()
A=10, Line=3

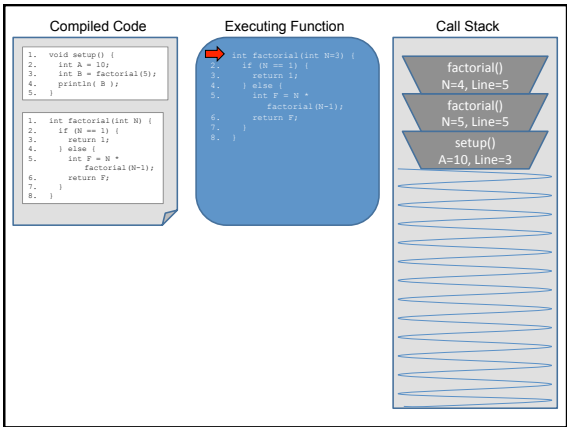

Compiled Code
Executing Function
Call Stack
setup()
A=10, Line=3

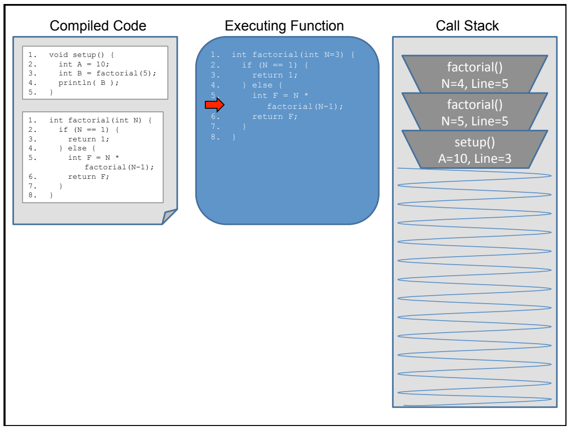

Compiled Code
Executing Function
Call Stack
factorial()
N=5, Line=5
setup()
A=10, Line=3

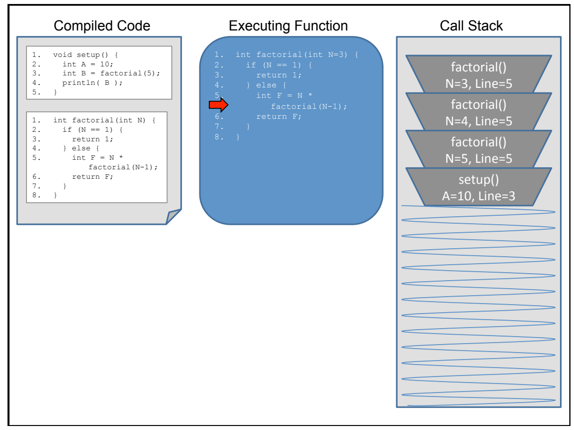

Compiled Code
Executing Function
Call Stack
factorial()
N=5, Line=5
setup()
A=10, Line=3

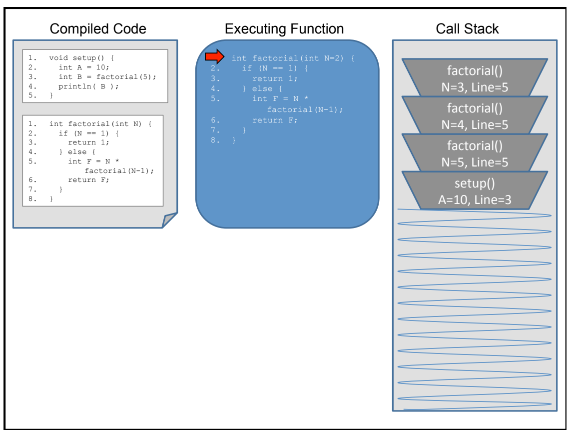

Compiled Code
Executing Function
Call Stack
factorial()
N=5, Line=5
setup()
A=10, Line=3

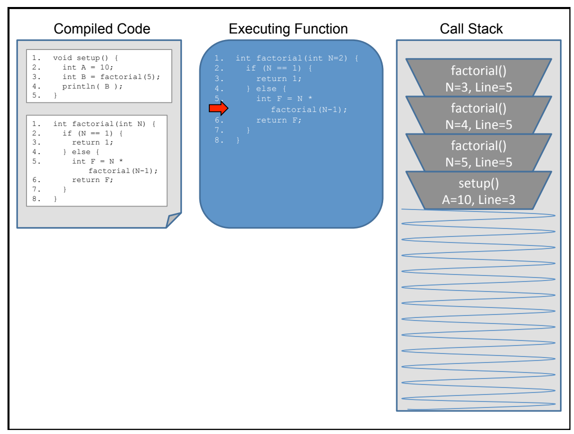

Compiled Code
Executing Function
Call Stack
factorial()
N=4, Line=5
factorial()
N=5, Line=5
setup()
A=10, Line=3

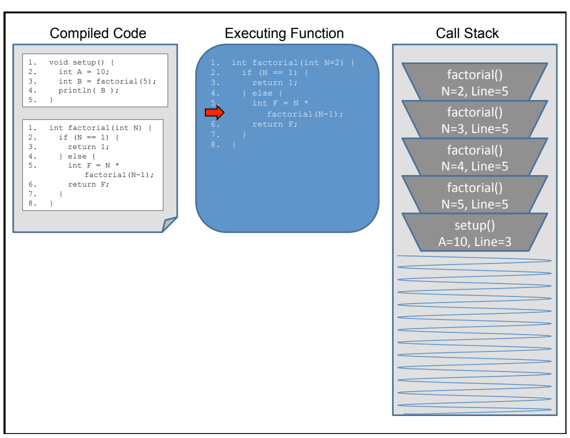

Compiled Code
Executing Function
Call Stack
factorial()
N=4, Line=5
factorial()
N=5, Line=5
setup()
A=10, Line=3

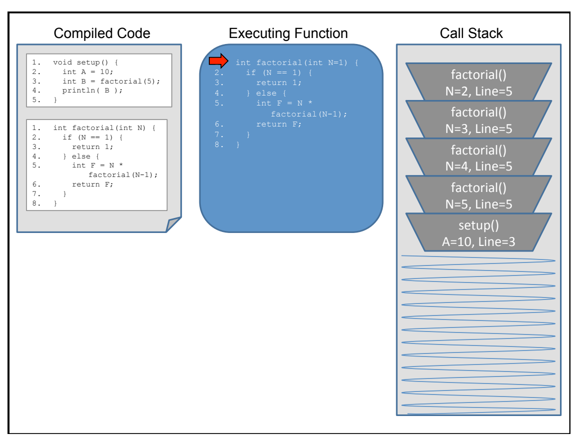

Compiled Code
Executing Function
Call Stack
factorial()
N=4, Line=5
factorial()
N=5, Line=5
setup()
A=10, Line=3

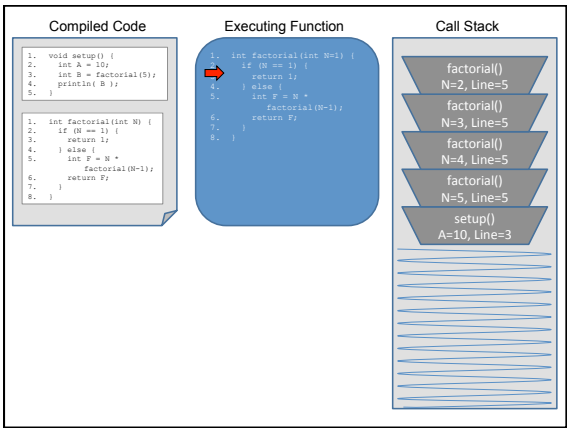

Compiled Code
Executing Function
Call Stack
factorial()
N=3, Line=5
factorial()
N=4, Line=5
factorial()
N=5, Line=5
setup()
A=10, Line=3

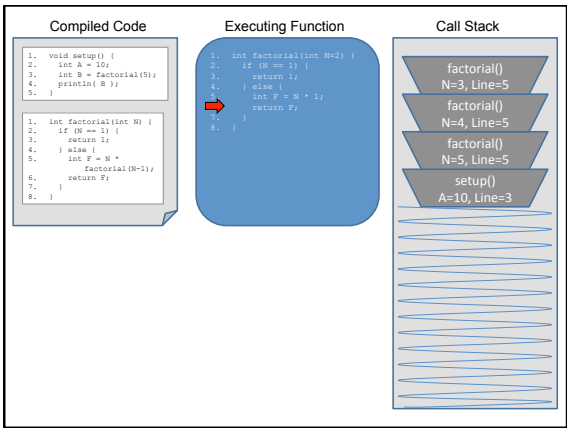

Compiled Code
Executing Function
Call Stack
factorial()
N=3, Line=5
factorial()
N=4, Line=5
factorial()
N=5, Line=5
setup()
A=10, Line=3

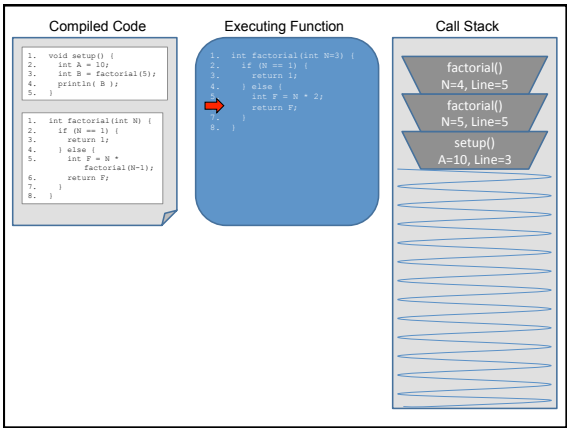

Compiled Code
Executing Function
Call Stack
factorial()
N=3, Line=5
factorial()
N=4, Line=5
factorial()
N=5, Line=5
setup()
A=10, Line=3

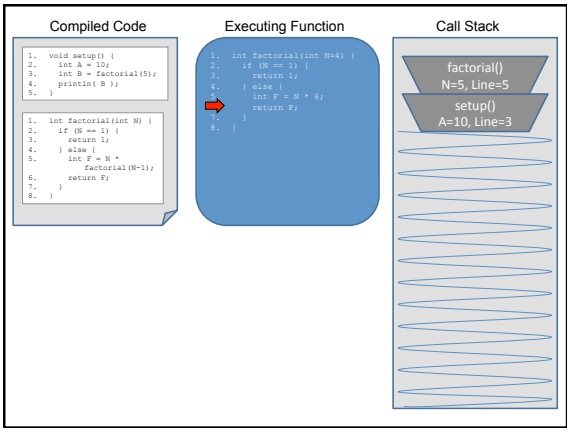

Compiled Code
Executing Function
Call Stack
factorial()
N=2, Line=5
factorial()
N=3, Line=5
factorial()
N=4, Line=5
factorial()
N=5, Line=5
setup()
A=10, Line=3

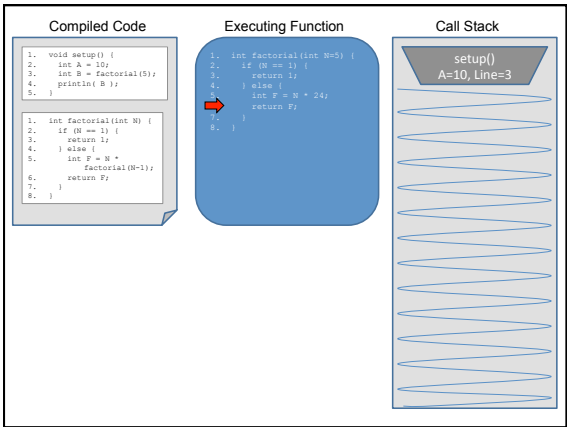

Compiled Code
Executing Function
Call Stack
factorial()
N=2, Line=5
factorial()
N=3, Line=5
factorial()
N=4, Line=5
factorial()
N=5, Line=5
setup()
A=10, Line=3

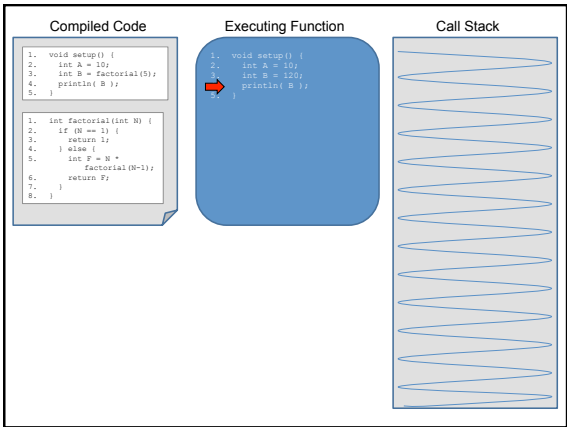

Compiled Code
Executing Function
Call Stack
factorial()
N=2, Line=5
factorial()
N=3, Line=5
factorial()
N=4, Line=5
factorial()
N=5, Line=5
setup()
A=10, Line=3


Compiled Code
Executing Function
Call Stack
factorial()
N=3, Line=5
factorial()
N=4, Line=5
factorial()
N=5, Line=5
setup()
A=10, Line=3


Compiled Code
Executing Function
Call Stack
factorial()
N=4, Line=5
factorial()
N=5, Line=5
setup()
A=10, Line=3


Compiled Code
Executing Function
Call Stack
factorial()
N=5, Line=5
setup()
A=10, Line=3


Compiled Code
Executing Function
Call Stack
setup()
A=10, Line=3

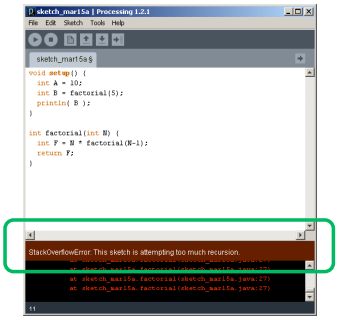

Compiled Code
Executing Function
Call Stack

Call Stack

The Call Stack keeps track of …

1. all functions that are suspended, in order
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"?



What does this do?

```
void setup() {
  println(F(12));
}

int F(int n) {
  if (n == 0) {
    return 0;
  } else if (n == 1) {
    return 1;
  } else {
    return F(n-1) + F(n-2);
  }
}
```

Examples

- recursiveCircles
- recursiveTree
- recursiveTreeTransform