Tracing

Chris Piech
CS106A, Stanford University

This is Method Man. He is part of the Wu Tang Clan. ☺

Learn How To:

Trace programs step by step

Anatomy of a method

```
public void run() {
    double mid = average(5.0, 10.2);
    println(mid);
}
```

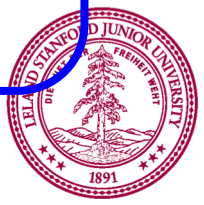
```
private double average(double a, double b) {
    double sum = a + b;
    return sum / 2;
}
```


Anatomy of a method

```
public void run() {
    double mid = average(5.0, 10.2);
    println(mid);
}
```

method "definition"

```
private double average(double a, double b) {
    double sum = a + b;
    return sum / 2;
}
```



Anatomy of a method

```
public void run() {
    double mid = average(5.0, 10.2);
    println(mid);
}
```

Output expected

Input expected

```
private double average(double a, double b) {
    double sum = a + b;
    return sum / 2;
}
```


Anatomy of a method

```
public void run() {
    double mid = average(5.0, 10.2);
    println(mid);
}
```

Return Type

Parameters

```
private double average(double a, double b) {
    double sum = a + b;
    return sum / 2;
}
```


Anatomy of a method

```
public void run() {
    double mid = average(5.0, 10.2);
    println(mid);
}
```

name

```
private double average(double a, double b) {
    double sum = a + b;
    return sum / 2;
}
```


Anatomy of a method

```
public void run() {
    double mid = average(5.0, 10.2);
    println(mid);
}
```

```
private double average(double a, double b) {
    double sum = a + b;
    return sum / 2;
}
```

body

Anatomy of a method

```
public void run() {
    double mid = average(5.0, 10.2);
    println(mid);
}
```

```
private double average(double a, double b) {
    double sum = a + b;
    return sum / 2;
}
```

Ends the method and gives back a single value

Anatomy of a method

```
public void run() {
    double mid = average(5.0, 10.2);
    println(mid);
}
```

```
private double average(double a, double b) {
    double sum = a + b;
    return sum / 2;
}
```

This statement is necessary because **`average`** promised to return a double

Anatomy of a method

method "call"

```
public void run() {
    double mid = average(5.0, 10.2);
    println(mid);
}

private double average(double a, double b) {
    double sum = a + b;
    return sum / 2;
}
```


Anatomy of a method

arguments

```
public void run() {
    double mid = average(5.0, 10.2);
    println(mid);
}

private double average(double a, double b) {
    double sum = a + b;
    return sum / 2;
}
```


Anatomy of a method

Return Type Parameters

```
public void run() {
    double mid = average(5.0, 10.2);
    println(mid);
}


private double average(double a, double b) {
    double sum = a + b;
    return sum / 2;
}
```


Anatomy of a method

```
public void run() {
    double mid = average(5.0, 10.2);
    println(mid);
}
```

When a method ends it "returns"

```
private double average(double a, double b) {
    double sum = a + b;
    return sum / 2;
}
```


Trace

```
public void run() {
    double mid = average(5.0, 10.2);
    println(mid);
}

private double average(double a, double b) {
    double sum = a + b;
    return sum / 2;
}
```


Trace

Run memory

No variables

```
public void run() {
    double mid = average(5.0, 10.2);
    println(mid);
}


private double average(double a, double b) {
    double sum = a + b;
    return sum / 2;
}
```


Trace

Run memory

No variables

```
public void run() {
    double mid = average(5.0, 10.2);
    println(mid);
}


private double average(double a, double b) {
    double sum = a + b;
    return sum / 2;
}
```


Trace

Run memory

No variables

```
public void run() {
    double mid = average(5.0, 10.2);
    println(mid);
}


private double average(double a, double b) {
    double sum = a + b;
    return sum / 2;
}
```


Trace

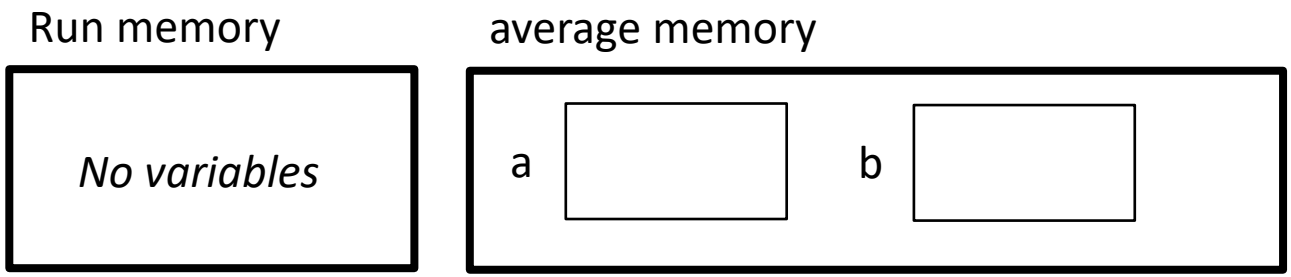
Run memory

No variables

```
public void run() {
    double mid = average(5.0, 10.2);
    println(mid);
}
```

```
private double average(double a, double b) {
    double sum = a + b;
    return sum / 2;
}
```

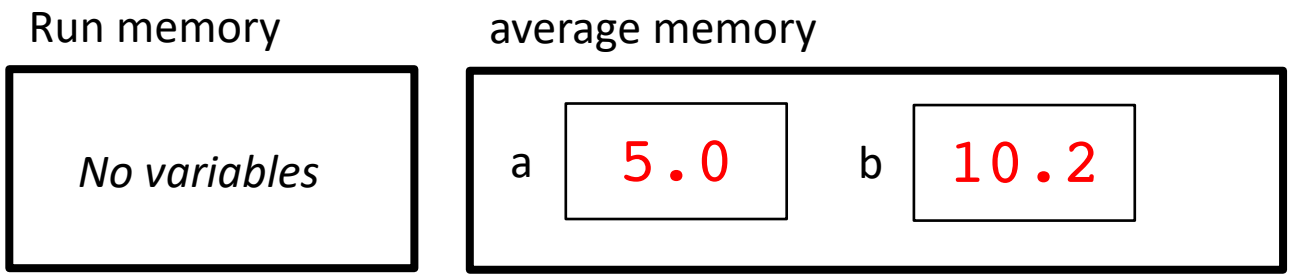

Trace




```
public void run() {
    double mid = average(5.0, 10.2);
    println(mid);
}
```

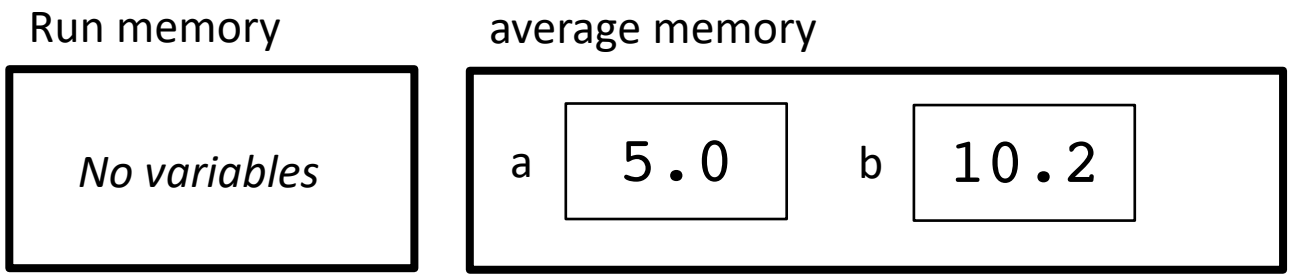
```
private double average(double a, double b) {
    double sum = a + b;
    return sum / 2;
}
```

Trace




```
public void run() {
    double mid = average(5.0, 10.2);
    println(mid);
}


private double average(double a, double b) {
    double sum = a + b;
    return sum / 2;
}
```

Trace




```
public void run() {
    double mid = average(5.0, 10.2);
    println(mid);
}


private double average(double a, double b) {
    double sum = a + b;
    return sum / 2;
}
```

Trace

Run memory

No variables

average memory

a 5.0 b 10.2 sum 15.2

```
public void run() {
    double mid = average(5.0, 10.2);
    println(mid);
}


private double average(double a, double b) {
    double sum = a + b;
    return sum / 2;
}
```


Trace

Run memory

No variables

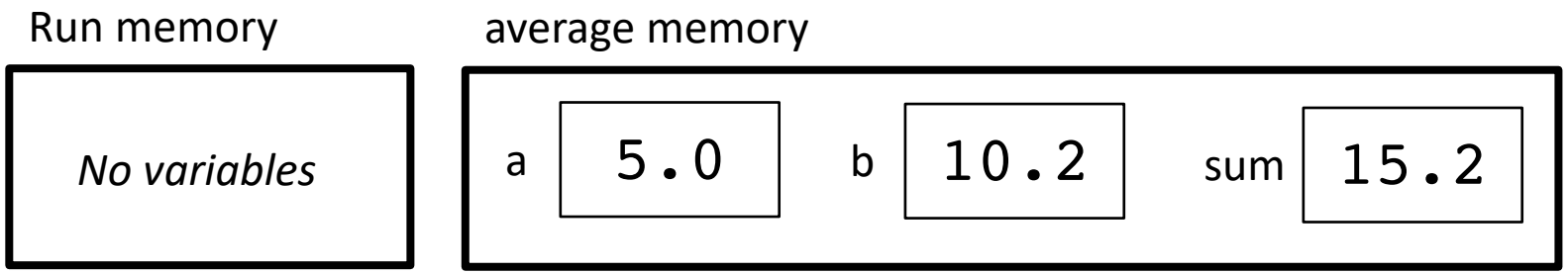
average memory

a	5.0	b	10.2	sum	15.2

```
public void run() {
    double mid = average(5.0, 10.2);
    println(mid);
}


private double average(double a, double b) {
    double sum = a + b;
    return sum / 2;
}
```

7.6

Trace




```
public void run() {
    double mid = average(5.0, 10.2);
    println(mid);
}
```

7.6

```
private double average(double a, double b) {
    double sum = a + b;
    return sum / 2;
}
```

Trace

Run memory

No variables

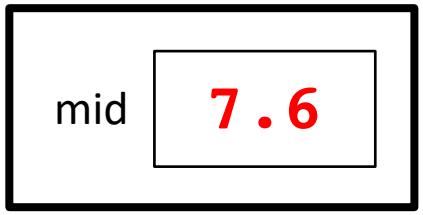
```
public void run() {
    double mid = average(5.0, 10.2);
    println(mid);
}
```

7.6

```
private double average(double a, double b) {
    double sum = a + b;
    return sum / 2;
}
```


Trace

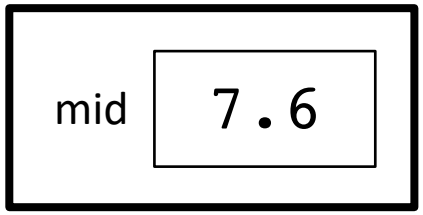
Run memory




```
public void run() {
    double mid = average(5.0, 10.2);
    println(mid);
}


private double average(double a, double b) {
    double sum = a + b;
    return sum / 2;
}
```

Trace

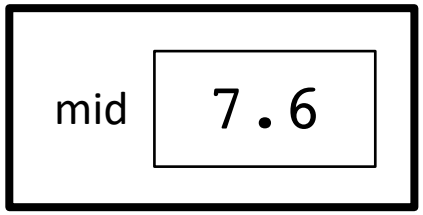
Run memory




```
public void run() {
    double mid = average(5.0, 10.2);
    println(mid);
}


private double average(double a, double b) {
    double sum = a + b;
    return sum / 2;
}
```

Trace

Run memory

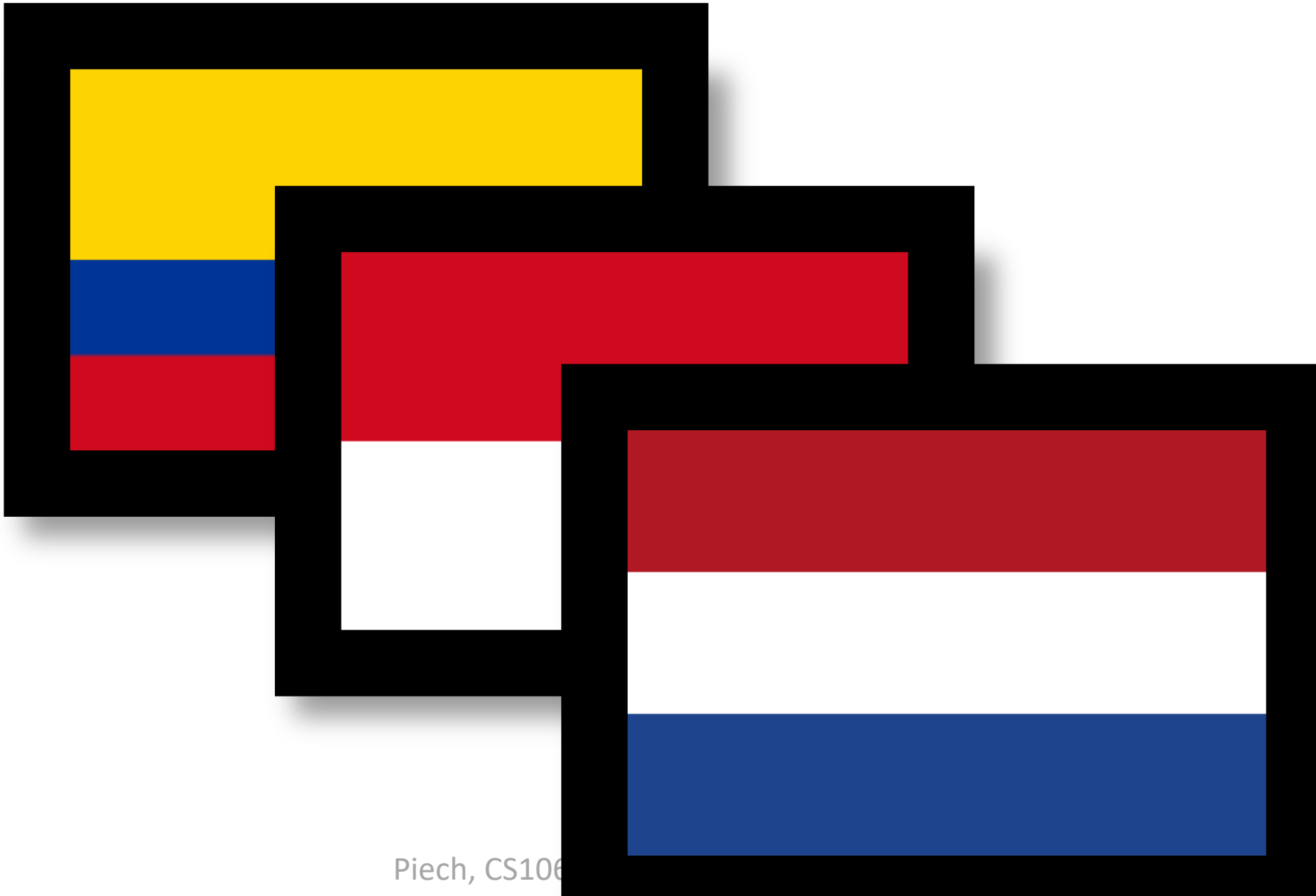



```
public void run() {
    double mid = average(5.0, 10.2);
    println(mid);
}

private double average(double a, double b) {
    double sum = a + b;
    return sum / 2;
}
```

Trace

```
public void run() {
    double mid = average(5.0, 10.2);
    println(mid);
}

private double average(double a, double b) {
    double sum = a + b;
    return sum / 2;
}
```


Drawing Flags



Flags

```
private void drawColombiasFlag() {
    // Call drawStripe three times
    // with different inputs each time
    drawStripe(Color.YELLOW, 0);
    drawStripe(Color.BLUE, 0.5);
    drawStripe(Color.RED, 0.75);
}
```

```
// Define drawStripe
// drawStripe needs two inputs (which come as variables)
// First a color, then a fraction down the screen
private void drawStripe(Color color, double fractionDown) {
    double y = getHeight() * fractionDown;
    GRect rect = new GRect(0, y, getWidth(), getHeight() - y);
    rect.setColor(color);
    rect.setFilled(true);
    add(rect);
}
```

method “definition”

Flags

```
private void drawColombiasFlag() {
    // Call drawStripe three times
    // with different inputs each time
    drawStripe(Color.YELLOW, 0);
    drawStripe(Color.BLUE, 0.5);
    drawStripe(Color.RED, 0.75);
}
```

```
// Define drawStripe
// drawStripe needs tw
// First a color, then a fraction down the screen
private void drawStripe(Color color, double fractionDown) {
    double y = getHeight() * fractionDown;
    GRect rect = new GRect(0, y, getWidth(), getHeight() - y);
    rect.setColor(color);
    rect.setFilled(true);
    add(rect);
}
```

Takes two inputs: (1) a color, (2) a fraction

Flags

```
private void drawColombiasFlag() {
    // Call drawStripe three times
    // with different inputs each
    drawStripe(Color.YELLOW, 0);
    drawStripe(Color.BLUE, 0.5);
    drawStripe(Color.RED, 0.75);
}


// Define drawStripe
// drawStripe needs two inputs (which come as variables)
// First a color, then a fraction down the screen
private void drawStripe(Color color, double fractionDown) {
    double y = getHeight() * fractionDown;
    GRect rect = new GRect(0, y, getWidth(), getHeight() - y);
    rect.setColor(color);
    rect.setFilled(true);
    add(rect);
}
```

Thus, every time the method is called, two inputs must be given!

Flags

```
private void drawColombiasFlag() {
    // Call drawStripe three times
    // with different inputs each time
    drawStripe(Color.YELLOW, 0);
    drawStripe(Color.BLUE, 0.5);
    drawStripe(Color.RED, 0.75);
}
```

The method receives the first input in a box (aka as a variable), with the name color

```
private void drawStripe(Color color, double fractionDown) {
    double y = getHeight() * fractionDown;
    GRect rect = new GRect(0, y, getWidth(), getHeight() - y);
    rect.setColor(color);
    rect.setFilled(true);
    add(rect);
}
```

It can use the value in the color variable

Flags

```
private void drawColombiasFlag() {
    // Call drawStripe three times
    // with different inputs each time
    drawStripe(Color.YELLOW, 0);
    drawStripe(Color.BLUE, 0.5);
    drawStripe(Color.RED, 0.75);
}
```

The method receives the second input in a box (aka as a variable), with the name fractionDown

```
// Define drawSt
// drawStripe ne
// First a color, then a fraction down the screen
private void drawStripe(Color color, double fractionDown) {
    double y = getHeight() * fractionDown;
    GRect rect = new GRe                    Height() - y);
    rect.setColor(color)
    rect.setFilled(true)
    add(rect);
}
```

It can use the value in the fractionDown variable

A Full Program

```
public class FactorialExample extends ConsoleProgram {

   private static final int MAX_NUM = 4;

   public void run() {
       for(int i = 0; i < MAX_NUM; i++) {
             println(i + "! = " + factorial(i));
       }
   }

   private int factorial(int n) {
       int result = 1;
       for (int i = 1; i <= n; i++) {
             result *= i;
       }
       return result;
   }
}
```

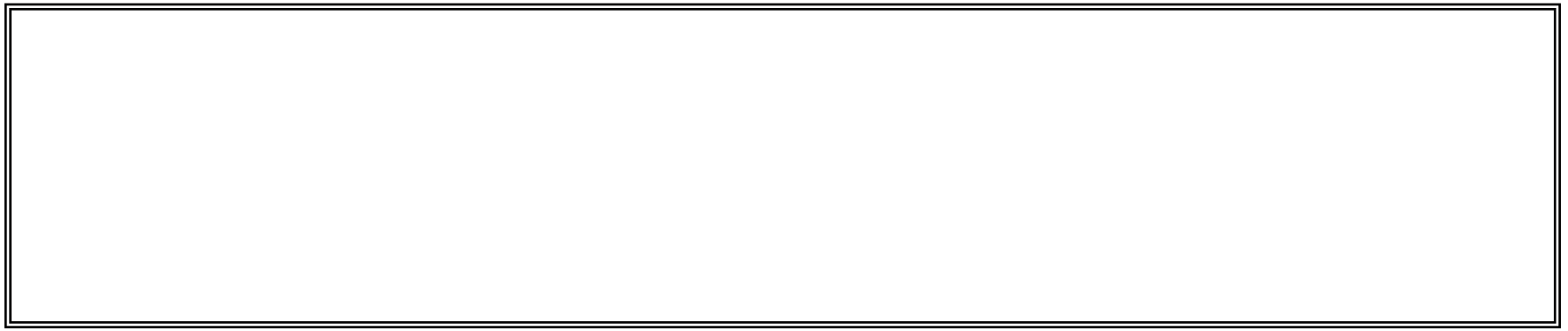
A Full Program

```
public class FactorialExample extends ConsoleProgram {

   private static final int MAX_NUM = 4;

   public void run() {
       for(int i = 0; i < MAX_NUM; i++) {
             println(i + "! = " + factorial(i));
       }
   }

   private int factorial(int n) {
       int result = 1;
       for (int i = 1; i <= n; i++) {
             result *= i;
       }
       return result;
   }
}
```

Understand the Mechanism

```
public void run() {
   for(int i = 0; i < MAX_NUM; i++) {
      println(i + "! = " + factorial(i));
   }
}
```

i 



```
public void run() {
    for(int i = 0; i < MAX_NUM; i++) {
        println(i + "! = " + factorial(i));
    }
}
```

i | 0

```
public void run() {
    for(int i = 0; i < MAX_NUM; i++) {
        println(i + "! = " + factorial(i));
    }
}
```

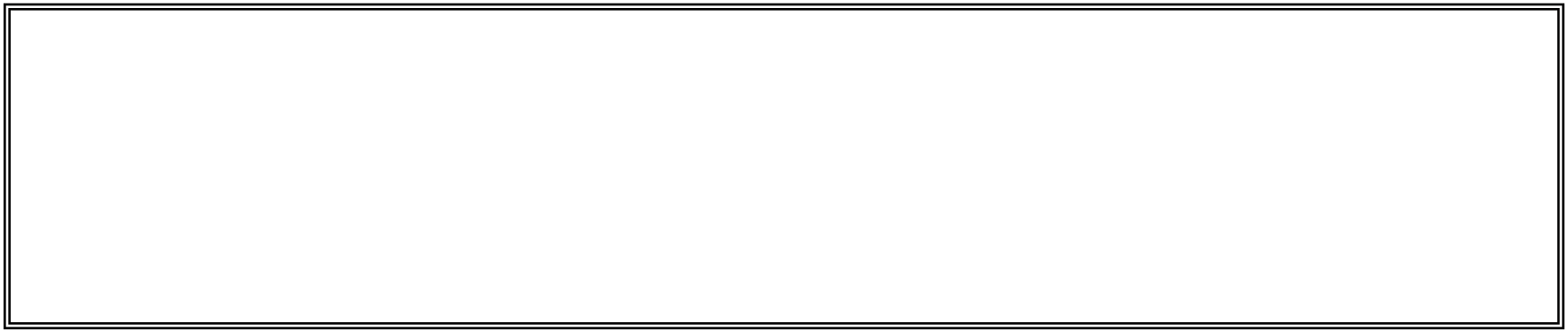
i | 0

```
public void run() {
    for(int i = 0; i < MAX_NUM; i++) {
        println(i + "! = " + factorial(i));
    }
}
```

i | 0

```
public void run() {
   for(int i = 0; i < MAX_NUM; i++) {
      println(i + "! = " + factorial(i));
   }
}
```

i | 0



```
private int factorial(int n) {
    int result = 1;
    for (int i = 1; i <= n; i++) {
        result *= i;
    }
    return result;
}
```

n 0 result i

```
private int factorial(int n) {
    int result = 1;
    for (int i = 1; i <= n; i++) {
        result *= i;
    }
    return result;
}
```

n 0 result 1 i

```
private int factorial(int n) {
    int result = 1;
    for (int i = 1; i <= n; i++) {
        result *= i;
    }
    return result;
}
```

n	0	result	1	i	1

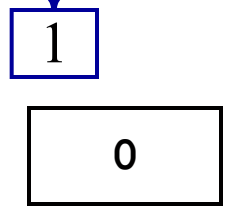
```
private int factorial(int n) {
    int result = 1;
    for (int i = 1; i <= n; i++) {
        result *= i;
    }
    return result;
}
```

n	0	result	1	i	1

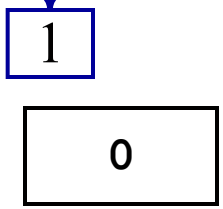
```
private int factorial(int n) {
    int result = 1;
    for (int i = 1; i <= n; i++) {
        result *= i;
    }
    return result;
}
```

n	0	result	1	i	1

```
public void run() {
   for(int i = 0; i < MAX_NUM; i++) {
      println(i + "! = " + factorial(i));
   }
}
```



```
public void run() {
    for(int i = 0; i < MAX_NUM; i++) {
        println(i + "! = " + factorial(i));
    }
}
```




```
0! = 1
```

```
public void run() {
    for(int i = 0; i < MAX_NUM; i++) {
        println(i + "! = " + factorial(i));
    }
}
```

i | 1

```
0! = 1
```

```
public void run() {
    for(int i = 0; i < MAX_NUM; i++) {
        println(i + "! = " + factorial(i));
    }
}
```

i | 1

```
0! = 1
```

```
public void run() {
    for(int i = 0; i < MAX_NUM; i++) {
        println(i + "! = " + factorial(i));
    }
}
```

i | 1

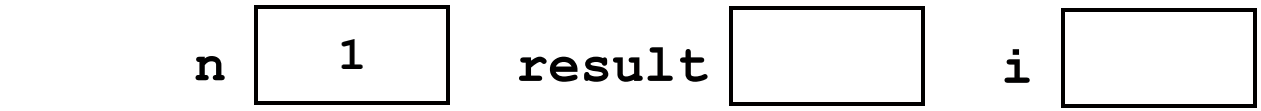
```
0! = 1
```

```
public void run() {
   for(int i = 0; i < MAX_NUM; i++) {
      println(i + "! = " + factorial(i));
   }
}
```

i | 1

```
0! = 1
```

```
private int factorial(int n) {
    int result = 1;
    for (int i = 1; i <= n; i++) {
        result *= i;
    }
    return result;
}
```



0! = 1

```
private int factorial(int n) {
    int result = 1;
    for (int i = 1; i <= n; i++) {
        result *= i;
    }
    return result;
}
```

n	1	result	1	i	

0! = 1

```
private int factorial(int n) {
    int result = 1;
    for (int i = 1; i <= n; i++) {
        result *= i;
    }
    return result;
}
```

n	1	result	1	i	1

0! = 1

```
private int factorial(int n) {
    int result = 1;
    for (int i = 1; i <= n; i++) {
        result *= i;
    }
    return result;
}
```

n	1	result	1	i	1

0! = 1

```
private int factorial(int n) {
    int result = 1;
    for (int i = 1; i <= n; i++) {
        result *= i;
    }
    return result;
}
```

n	1	result	1	i	1

0! = 1

```
private int factorial(int n) {
    int result = 1;
    for (int i = 1; i <= n; i++) {
        result *= i;
    }
    return result;
}
```

n	1	result	1	i	2

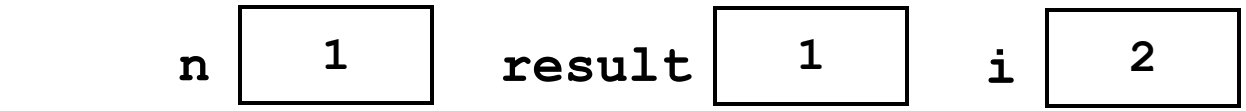
0! = 1

```
private int factorial(int n) {
    int result = 1;
    for (int i = 1; i <= n; i++) {
        result *= i;
    }
    return result;
}
```

n	1	result	1	i	2

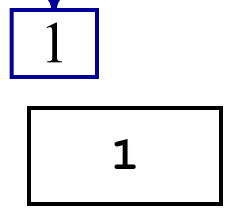
0! = 1

```
private int factorial(int n) {
    int result = 1;
    for (int i = 1; i <= n; i++) {
        result *= i;
    }
    return result;
}
```



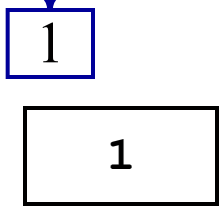
0! = 1

```
public void run() {
   for(int i = 0; i < MAX_NUM; i++) {
      println(i + "! = " + factorial(i));
   }
}
```




```
0! = 1
```

```
public void run() {
    for(int i = 0; i < MAX_NUM; i++) {
        println(i + "! = " + factorial(i));
    }
}
```




```
0! = 1
1! = 1
```

```
public void run() {
    for(int i = 0; i < MAX_NUM; i++) {
        println(i + "! = " + factorial(i));
    }
}
```

i | 2

```
0! = 1
1! = 1
```

```
public void run() {
    for(int i = 0; i < MAX_NUM; i++) {
        println(i + "! = " + factorial(i));
    }
}
```

i | 2

```
0! = 1
1! = 1
```

```
public void run() {
    for(int i = 0; i < MAX_NUM; i++) {
        println(i + "! = " + factorial(i));
    }
}
```

i | 2

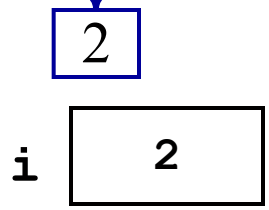
```
0! = 1
1! = 1
```

```
public void run() {
    for(int i = 0; i < MAX_NUM; i++) {
        println(i + "! = " + factorial(i));
    }
}
```

i | 2

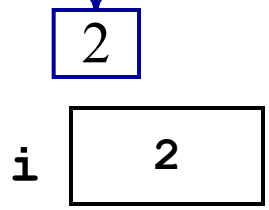
```
0! = 1
1! = 1
```

```
public void run() {
   for(int i = 0; i < MAX_NUM; i++) {
      println(i + "! = " + factorial(i));
   }
}
```




```
0! = 1
1! = 1
```

```
public void run() {
    for(int i = 0; i < MAX_NUM; i++) {
        println(i + "! = " + factorial(i));
    }
}
```




```
0! = 1
1! = 1
2! = 2
```

```
public void run() {
    for(int i = 0; i < MAX_NUM; i++) {
        println(i + "! = " + factorial(i));
    }
}
```

i | 3

```
0! = 1
1! = 1
2! = 2
```

```
public void run() {
    for(int i = 0; i < MAX_NUM; i++) {
        println(i + "! = " + factorial(i));
    }
}
```

i | 3

```
0! = 1
1! = 1
2! = 2
```

```
public void run() {
    for(int i = 0; i < MAX_NUM; i++) {
        println(i + "! = " + factorial(i));
    }
}
```

i | 3

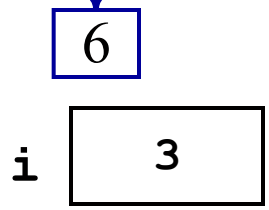
```
0! = 1
1! = 1
2! = 2
```

```
public void run() {
   for(int i = 0; i < MAX_NUM; i++) {
      println(i + "! = " + factorial(i));
   }
}
```

i 3

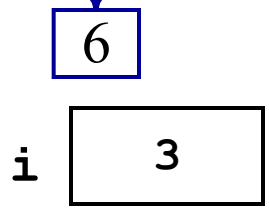
```
0! = 1
1! = 1
2! = 2
```

```
public void run() {
   for(int i = 0; i < MAX_NUM; i++) {
      println(i + "! = " + factorial(i));
   }
}
```




```
0! = 1
1! = 1
2! = 2
```

```
public void run() {
    for(int i = 0; i < MAX_NUM; i++) {
        println(i + "! = " + factorial(i));
    }
}
```




```
0! = 1
1! = 1
2! = 2
3! = 6
```

```
public void run() {
    for(int i = 0; i < MAX_NUM; i++) {
        println(i + "! = " + factorial(i));
    }
}
```

i | 4

```
0! = 1
1! = 1
2! = 2
3! = 6
```

```
public void run() {
    for(int i = 0; i < MAX_NUM; i++) {
        println(i + "! = " + factorial(i));
    }
}
```

i | 4

```
0! = 1
1! = 1
2! = 2
3! = 6
```

Parameters

Every time a method is called, new memory is created for the call.

Bad Times With Methods

```
// NOTE: This program is buggy!!

private void addFive(int x) {
  x += 5;
}


public void run() {
  int x = 3;
  addFive(x);
  println("x = " + x);
}
```

Let's "trace" this program on the board

Good Times With Methods

```
// NOTE: This program is feeling just fine...

private int addFive(int x) {
   x += 5;
   return x;
}

public void run() {
   int x = 3;
   x = addFive(x);
   println("x = " + x);
}
```


For primitives: Variables are **not** passed when you use parameters. Values are passed

Pass by “Value”

- Thanks Mehran

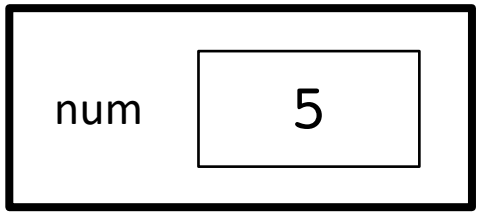
More Examples

Changed Name

```
private void run() {
    int num = 5;
    cow(num);
}

private void cow(int grass) {
    println(grass);
}
```


Changed Name

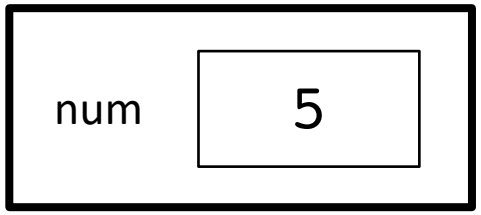
Run memory




```
private void run() {
    int num = 5;
    cow(num);
}

private void cow(int grass) {
    println(grass);
}
```

Changed Name

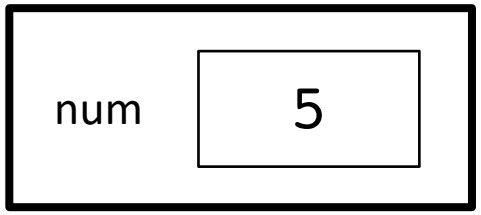
Run memory




```
private void run() {
    int num = 5;
    cow(num);
}

private void cow(int grass) {
    println(grass);
}
```

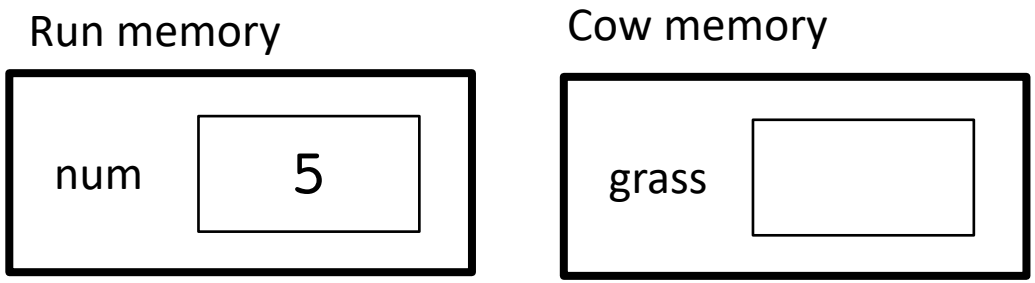
Changed Name

Run memory



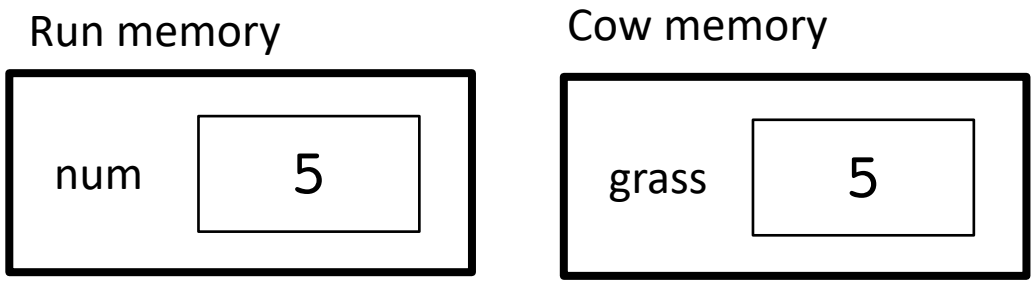

```
private void run() {
    int num = 5;
    cow(num);
}

private void cow(int grass) {
    println(grass);
}
```

Changed Name



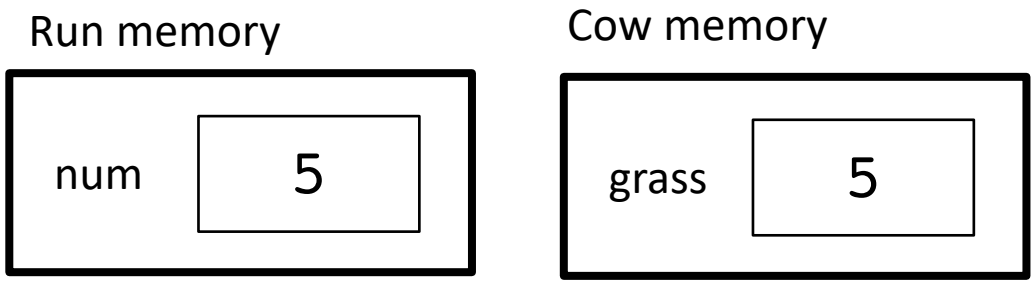

```
private void run() {
    int num = 5;
    cow(num);
}

private void cow(int grass) {
    println(grass);
}
```

Changed Name



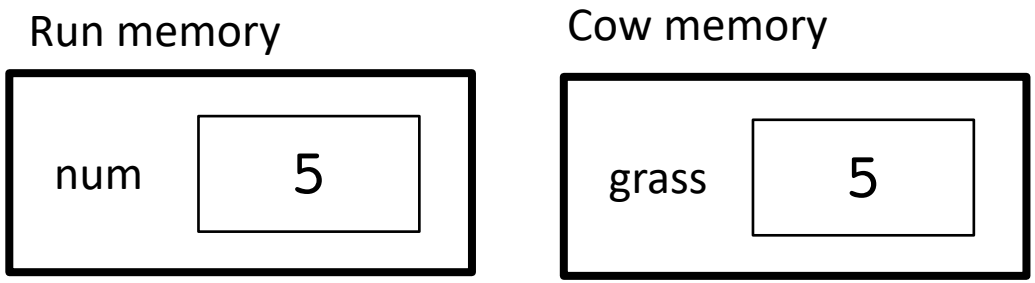

```
private void run() {
    int num = 5;
    cow(num);
}

private void cow(int grass) {
    println(grass);
}
```

Changed Name




```
private void run() {
    int num = 5;
    cow(num);
}

private void cow(int grass) {
    println(grass);
}
```

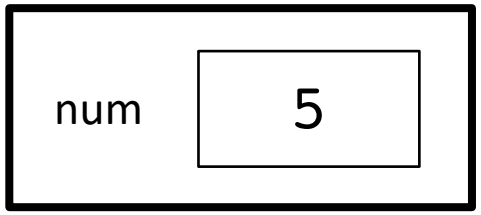
Changed Name




```
private void run() {
    int num = 5;
    cow(num);
}

private void cow(int grass) {
    println(grass);
}
```

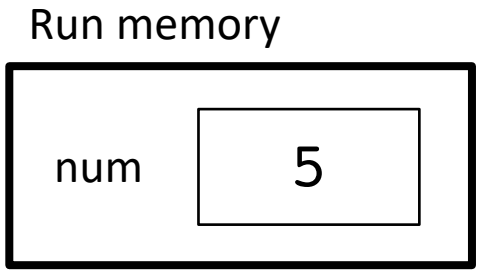
Changed Name

Run memory




```
private void run() {
    int num = 5;
    cow(num);
}

private void cow(int grass) {
    println(grass);
}
```

Changed Name




```
private void run() {
    int num = 5;
    cow(num);
}

private void cow(int grass) {
    println(grass);
}
```

Changed Name

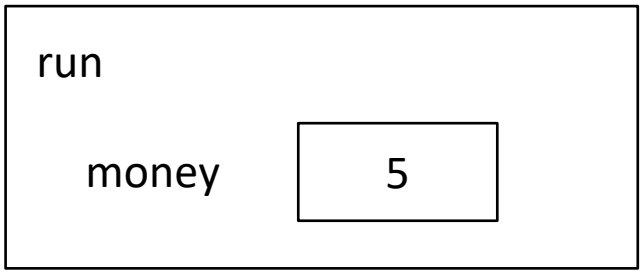
```
private void run() {
    int num = 5;
    cow(num);
}

private void cow(int grass) {
    println(grass);
}
```


Same Variable Name

```
private void run() {
    int money = 5;
    retireEarly();
    println(money);
}

private void retireEarly(){
    int money = 1200000;
    println(money);
}
```

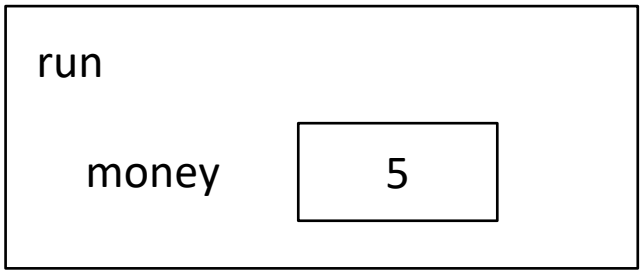
Same Variable Name

```
private void run() {
    int money = 5;
    retireEarly();
    println(money);
}

private void retireEarly(){
    int money = 1200000;
    println(money);
}
```



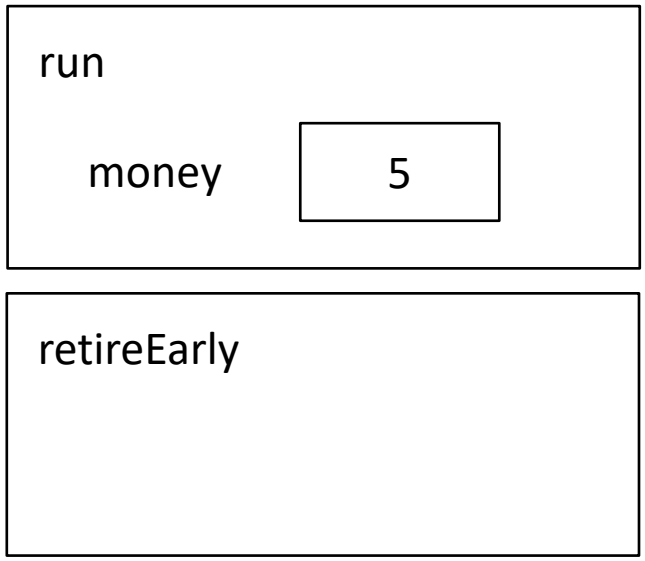
Same Variable Name

```
private void run() {
    int money = 5;
    retireEarly();
    println(money);
}

private void retireEarly(){
    int money = 1200000;
    println(money);
}
```



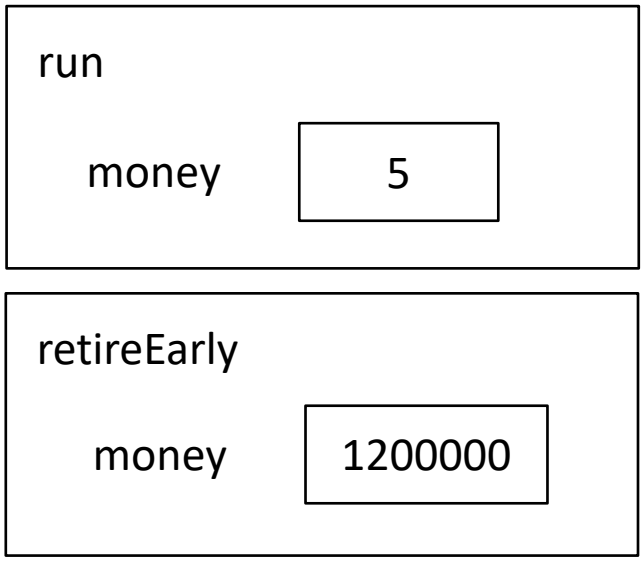
Same Variable Name

```
private void run() {
    int money = 5;
    retireEarly();
    println(money);
}

private void retireEarly(){
    int money = 1200000;
    println(money);
}
```



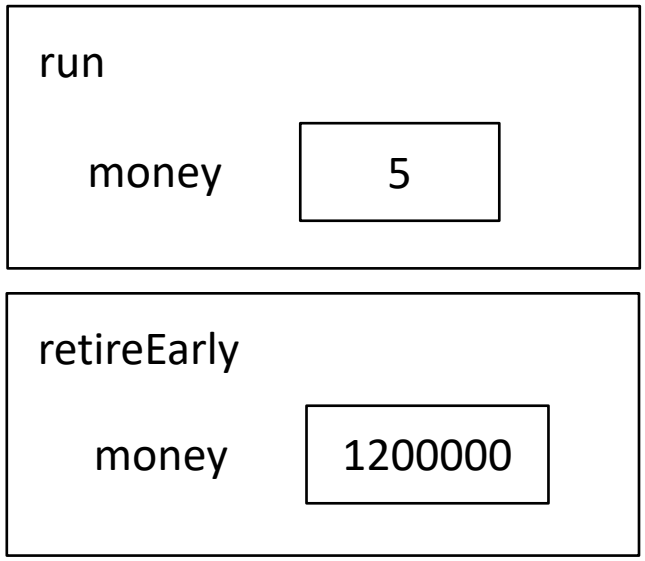
Same Variable Name

```
private void run() {
    int money = 5;
    retireEarly();
    println(money);
}

private void retireEarly(){
    int money = 1200000;
    println(money);
}
```



Same Variable Name

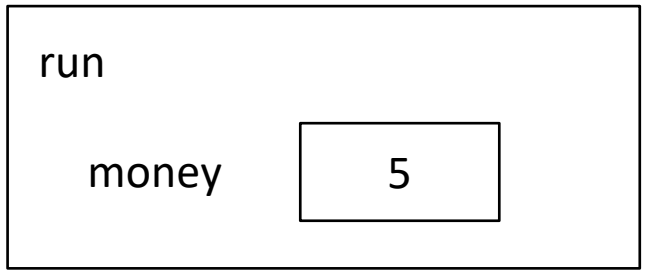
```
private void run() {
    int money = 5;
    retireEarly();
    println(money);
}

private void retireEarly(){
    int money = 1200000;
    println(money);
}
```

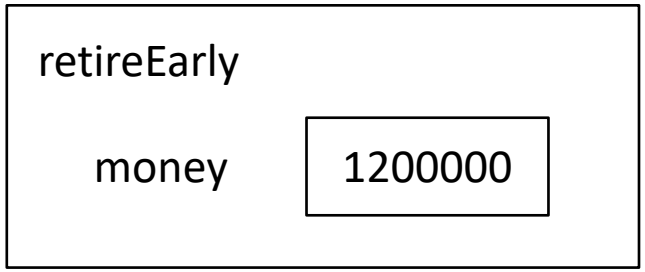



```
1200000
```

Same Variable Name

```
private void run() {
    int money = 5;
    retireEarly();
    println(money);
}

private void retireEarly(){
    int money = 1200000;
    println(money);
}
```

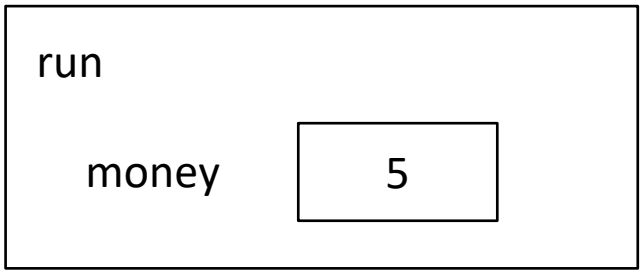





```
1200000
```

Same Variable Name

```
private void run() {
    int money = 5;
    retireEarly();
    println(money);
}

private void retireEarly(){
    int money = 1200000;
    println(money);
}
```




```
1200000
5
```

No Methods in Methods

```
private void run() {
    println("hello world");
    private void sayGoodbye() {
        println("goodbye!");
    }
}
```

Illegal modifier for parameter goodbye, only final is permitted

Huh?!?

No Methods in Methods

```
private void run() {
    println("hello world");
    sayGoodbye();
}

private void sayGoodbye() {
    println("goodbye!");
}
```


Remember Booleans?

Boolean Variable

```
boolean karelIsAwesome = true;

boolean myBool = 1 < 2;
```


Boolean Operations

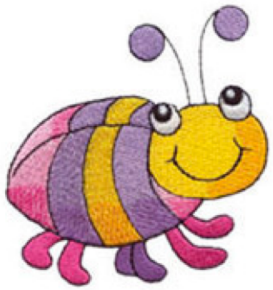
```
boolean a = true;

boolean b = false;


boolean and = a && b;

boolean or = a || b;

boolean not = !a;
```


Is Square

```
public void run() {
    for(int i = 1; i <= 100; i++) {
        if(isSquare(i)) {
            println(i);
        }
    }
}
```

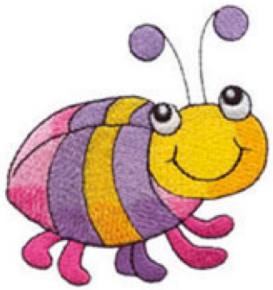

Boolean Return

```
public void run() {
   for(int i = 1; i <= 100; i++) {
      if(isSquare(i)) {
         println(i);
      }
   }
}

private boolean isSquare(int x) {
   double root = Math.sqrt(x);
   if(root == Math.floor(root)) {
      return true;
   } else {
      return false;
   }
}
```



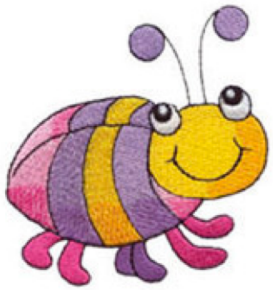
Boolean Return

```
public void run() {
   for(int i = 1; i <= 100; i++) {
      if(isSquare(i)) {
         println(i);
      }
   }
}

private boolean isSquare(int x) {
   double root = Math.sqrt(x);
   return root == Math.floor(root);
}
```

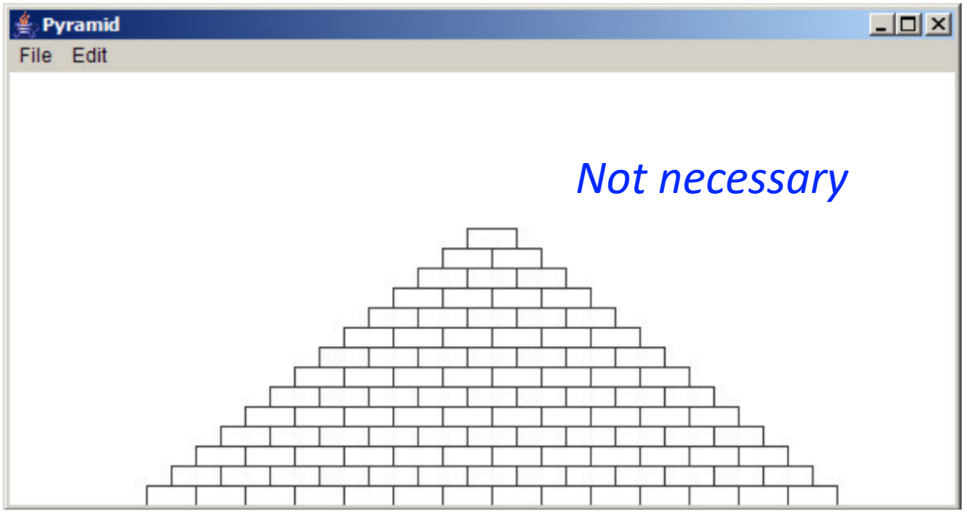


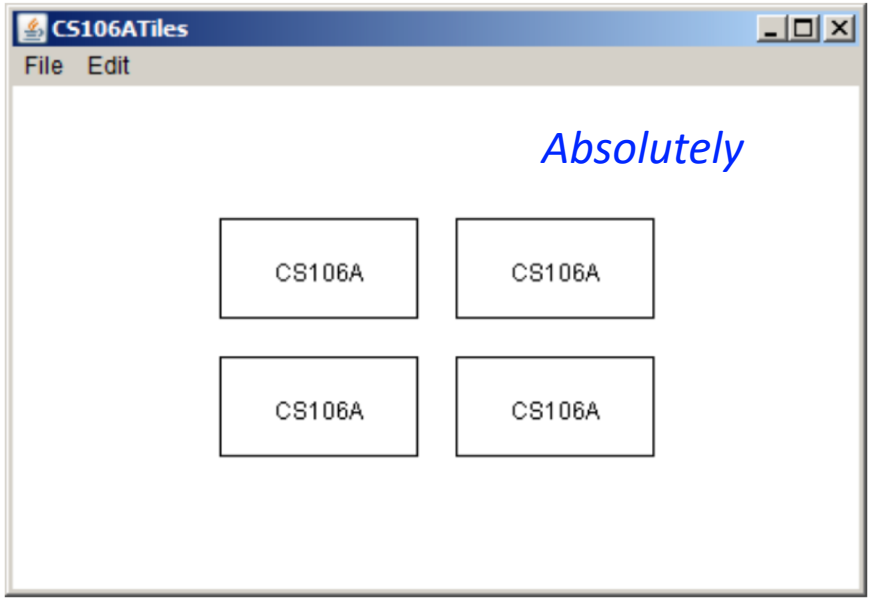
Boolean Return

```
public void run() {
   for(int i = 1; i <= 100; i++) {
      if(isSquare(i)) {
         println(i);
      }
   }
}

private boolean isSquare(int x) {
   double root = Math.sqrt(x);
   return root == (int)root;
}
```

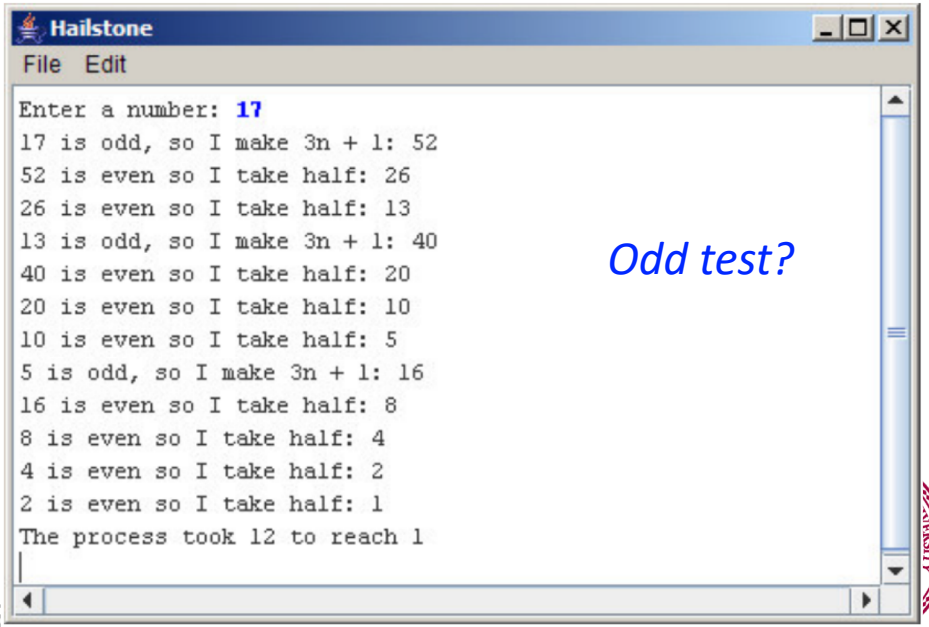


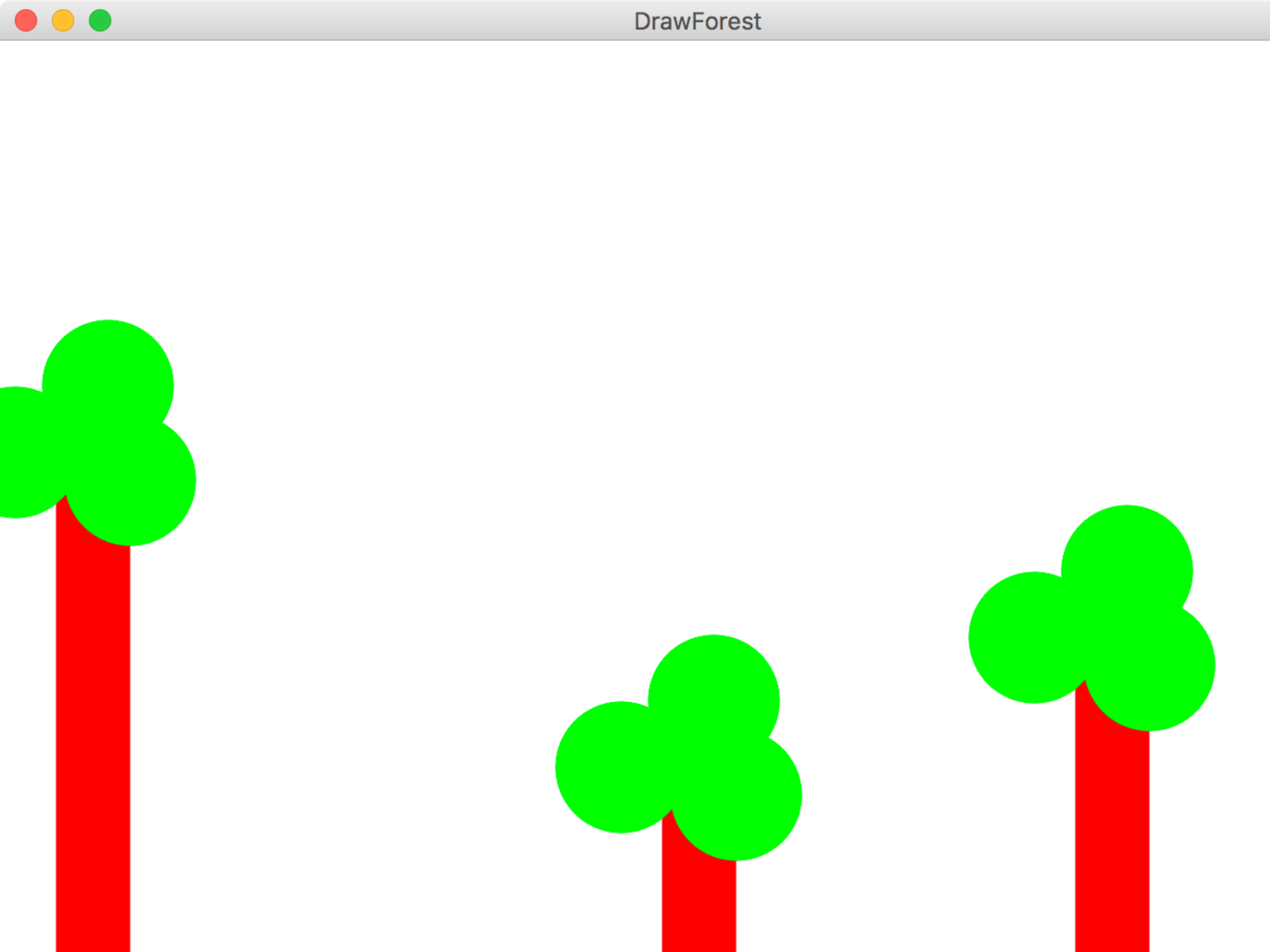
Assn 2


Pyramid
File Edit
Not necessary


CS106ATiles
File Edit
Absolutely
CS106A
CS106A
CS106A
CS106A


Target
File Edit
Yes


Hailstone
File Edit
Enter a number: 17
17 is odd, so I make 3n + 1: 52
52 is even so I take half: 26
26 is even so I take half: 13
13 is odd, so I make 3n + 1: 40
40 is even so I take half: 20
20 is even so I take half: 10
10 is even so I take half: 5
5 is odd, so I make 3n + 1: 16
16 is even so I take half: 8
8 is even so I take half: 4
4 is even so I take half: 2
2 is even so I take half: 1
The process took 12 to reach 1
Odd test?

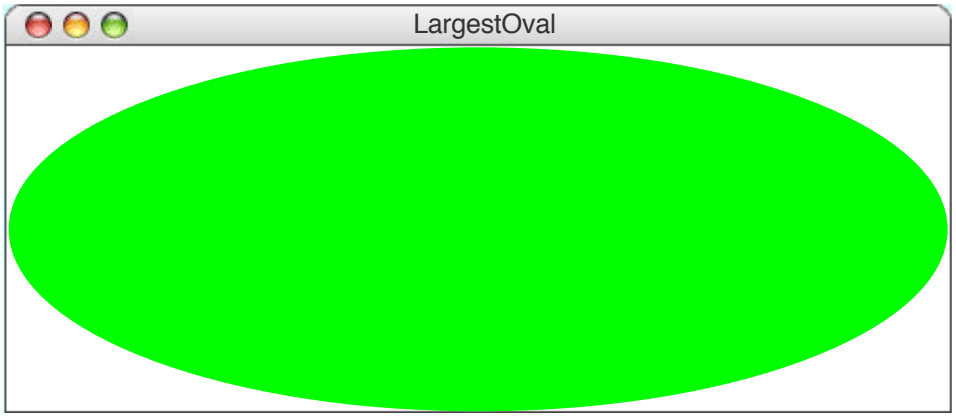

DrawForest

GOval

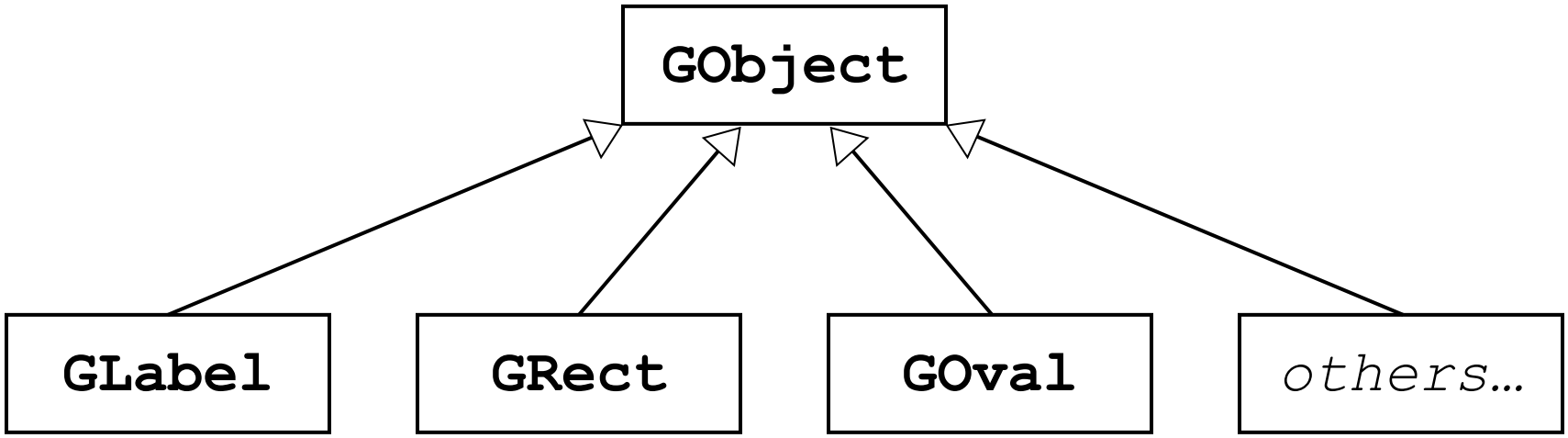
The `GOval` class represents an elliptical shape defined by the boundaries of its enclosing rectangle.

As an example, the following `run` method creates the largest oval that fits within the canvas:

```
public void run() {
   GOval oval = new GOval(0, 0, getWidth(), getHeight());
   oval.setFilled(true);
   oval.setColor(Color.GREEN);
   add(oval);
}
```

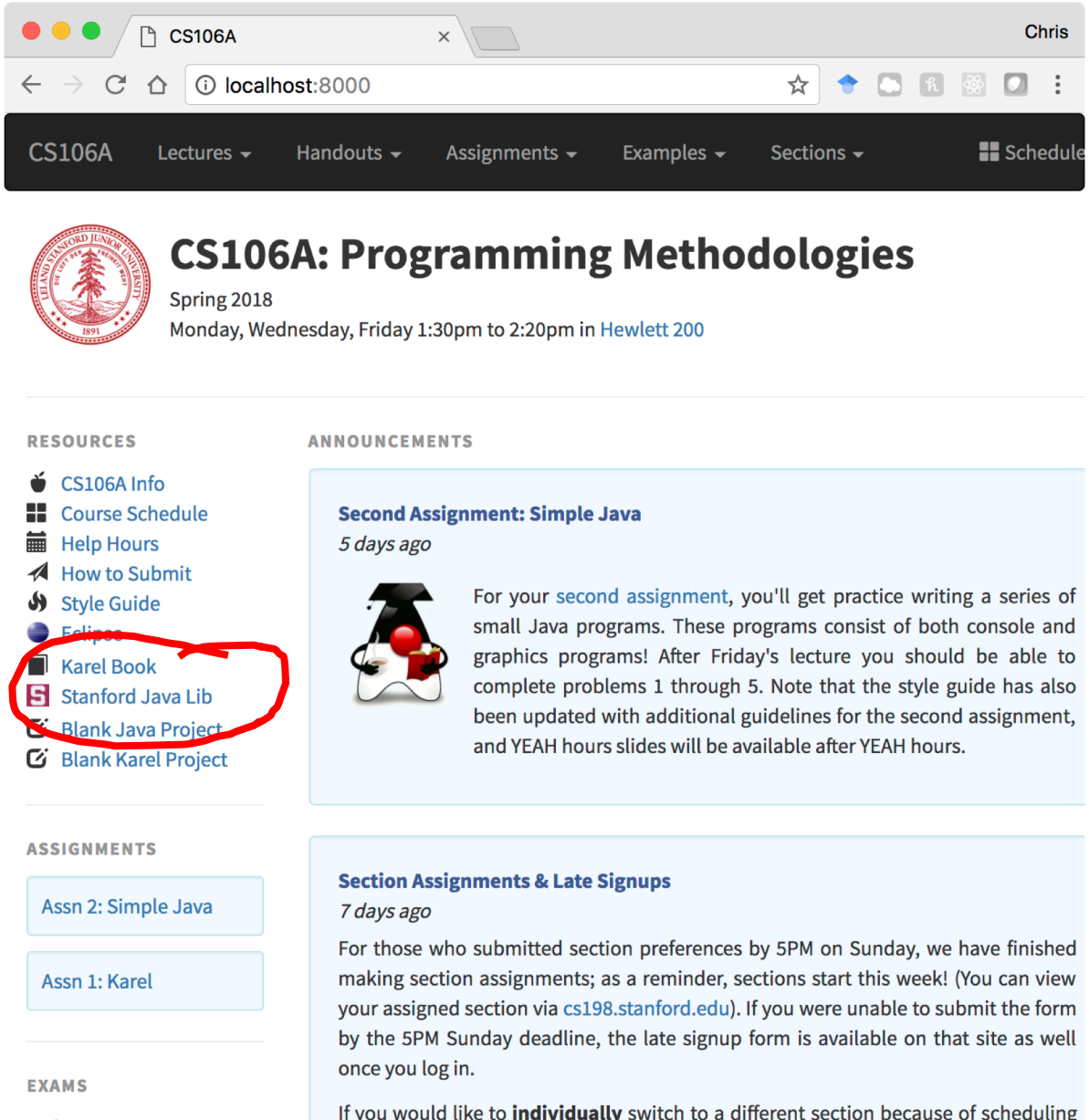


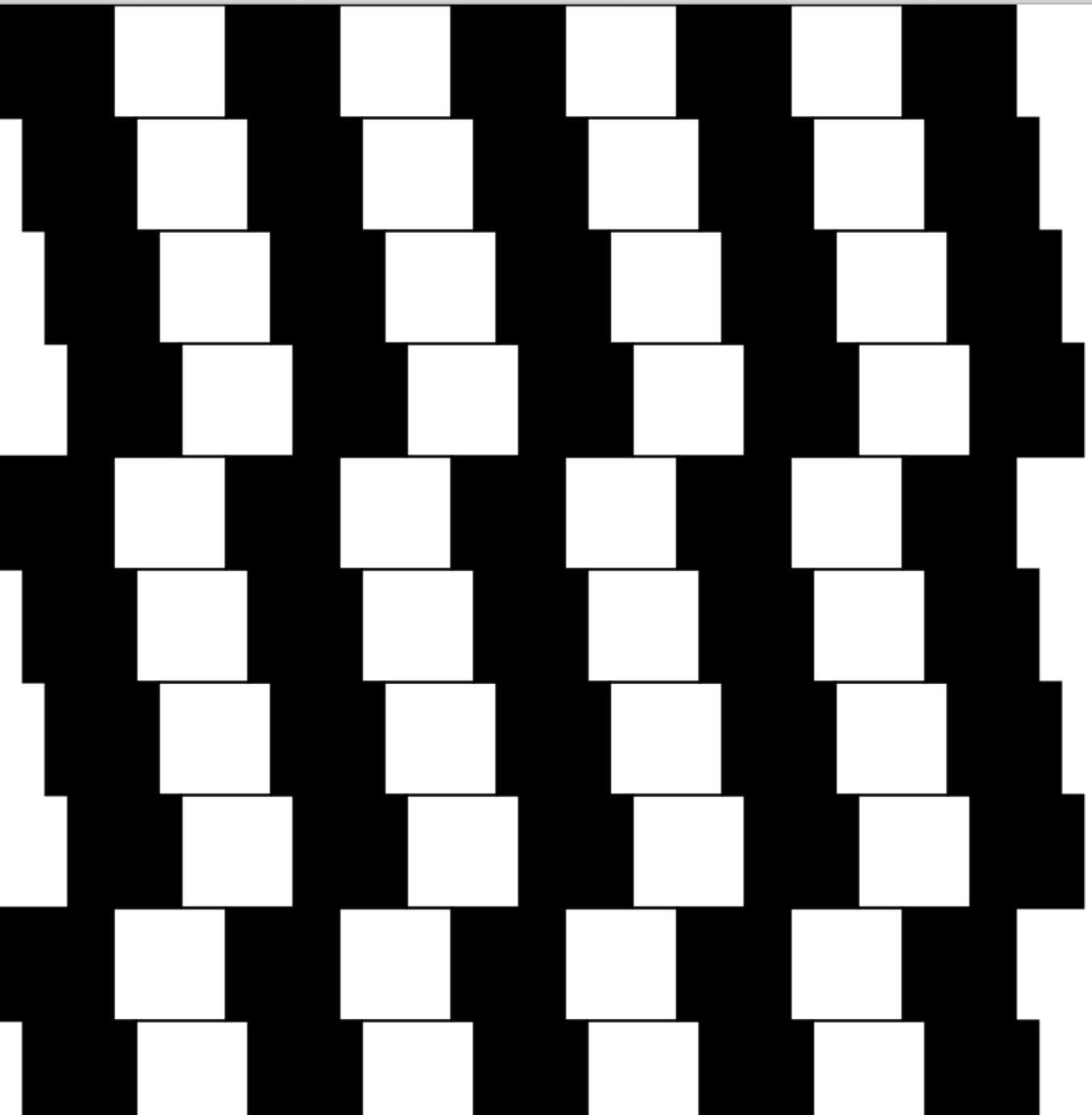
Graphics Variable Types




```
GLine line = new GLine(0, 0, 50, 50);
line.setColor(Color.BLUE);
```

Graphics Variable Types




IllusionSoln