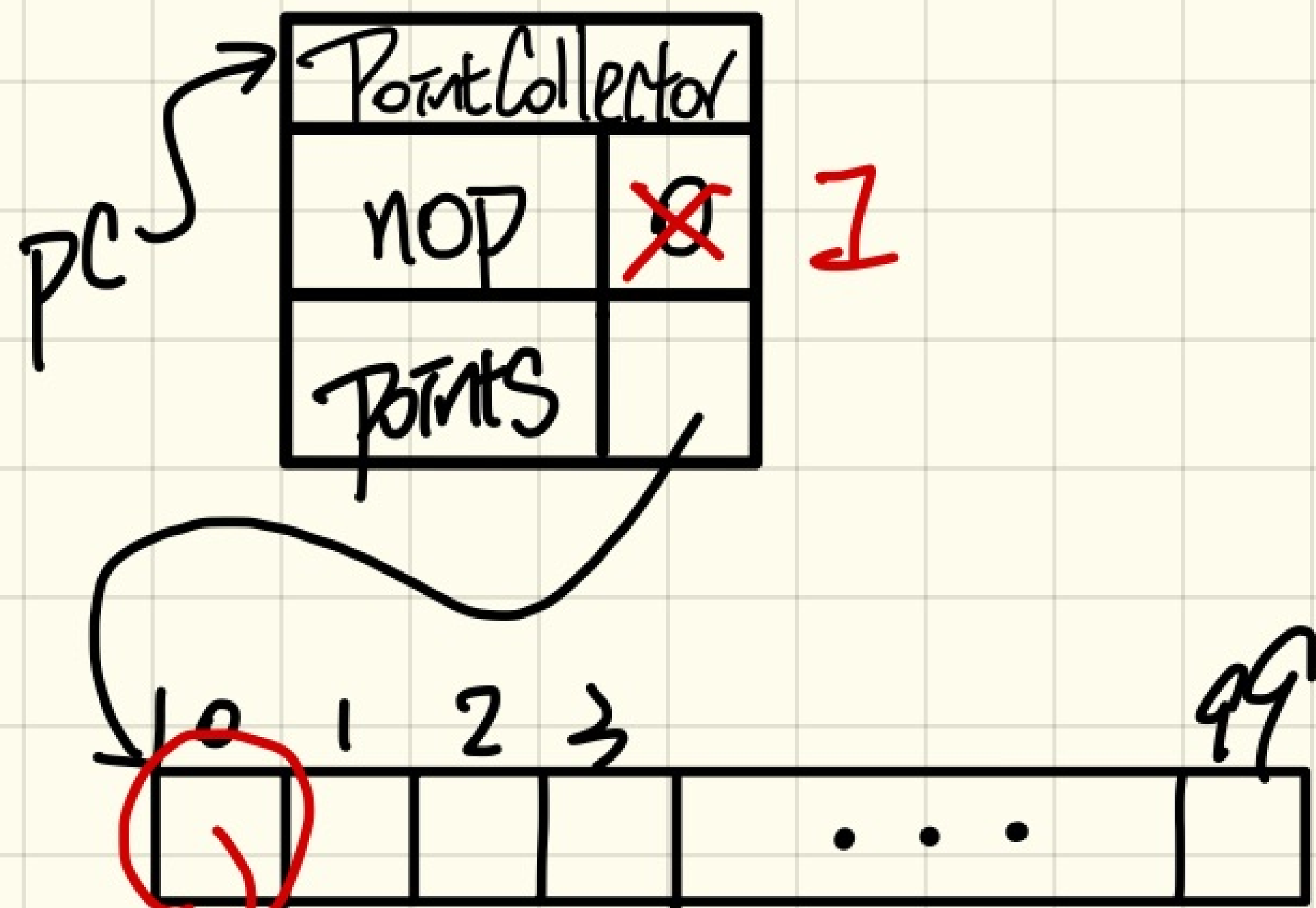
Lecture 4

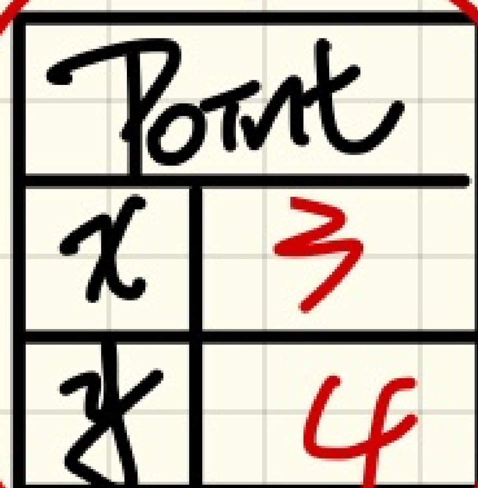
Monday September 16

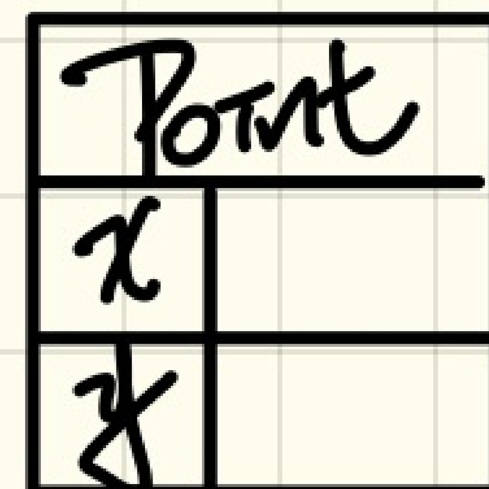
Programming Pattern: Mutator

```
class PointCollector {
  Point[] points; int nop; /* number of points */
  PointCollector() { points = new Point[100]; }
  void addPoint(double x, double y) {
   points[nop] = new Point(x, y); nop++; }
```

```
class PointCollectorTester {
  public static void main(String[] args) {
    PointCollector pc = new PointCollector();
    System.out.println(pc.nop);   /* 0 */
    pc.addPoint(3, 4);
    System.out.println(pc.nop);   /* 1 */
    pc.addPoint(-3, 4);
    System.out.println(pc.nop);   /* 2 */
    pc.addPoint(-3, -4);
    System.out.println(pc.nop);   /* 3 */
    pc.addPoint(3, -4);
    System.out.println(pc.nop);   /* 4 */
```





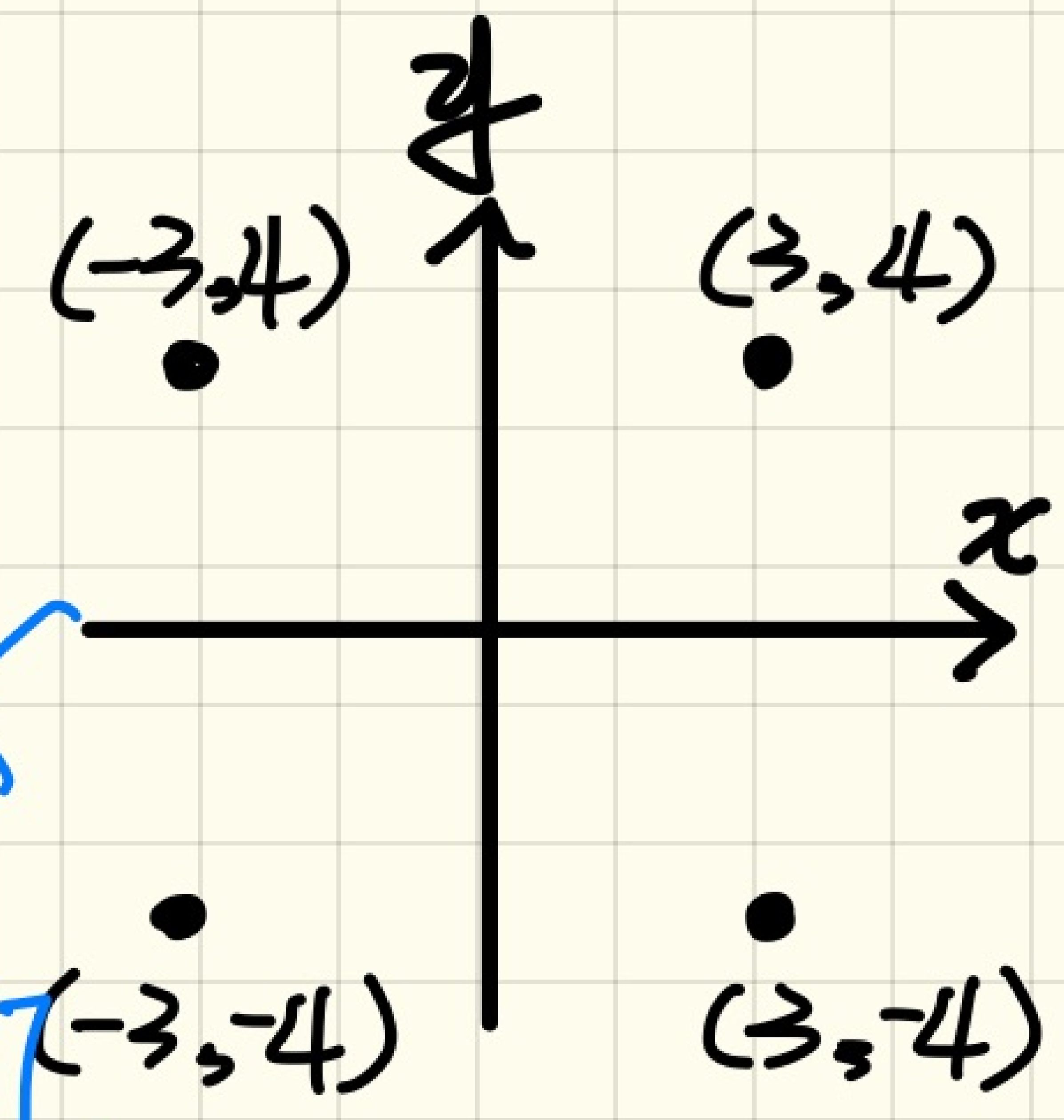


nop:

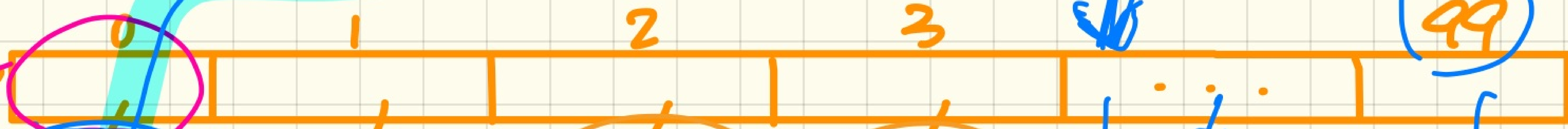
1. number of Point objects stored
2. index to store the next Point object

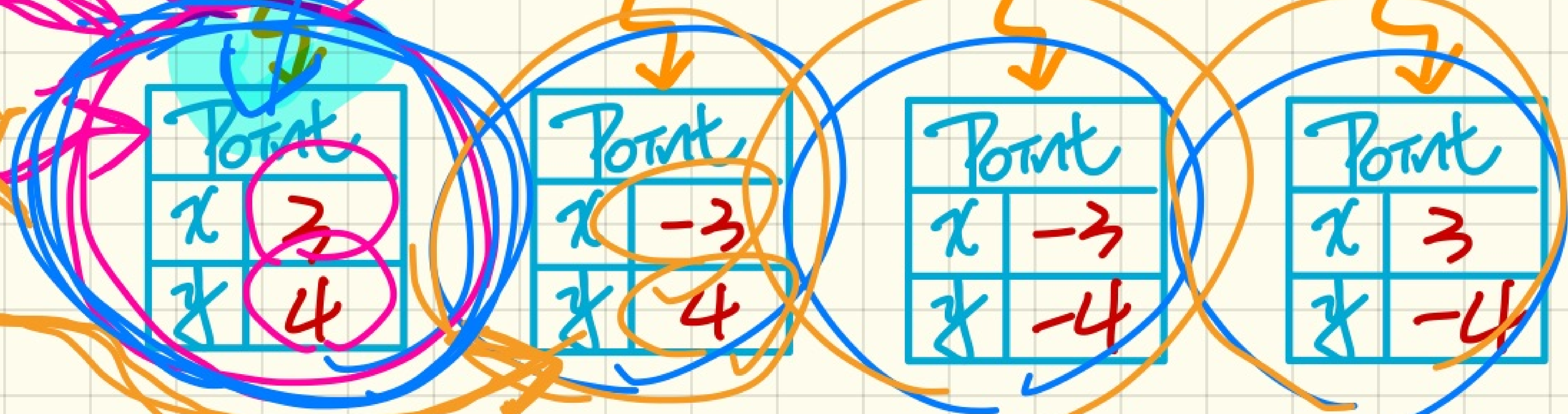
Programming Pattern: Accessor

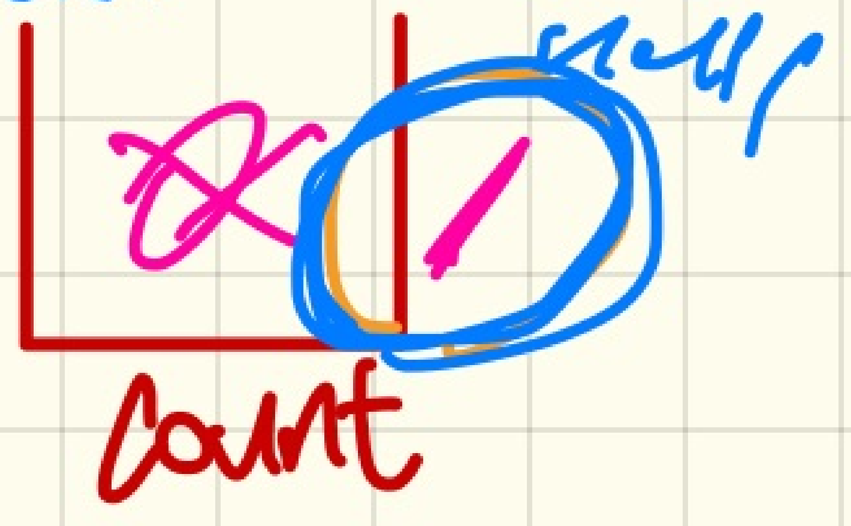
```
Point[] getPointsInQuadrantI() {
  Point[] ps = new Point[nop];
  int count = 0; /* number of points in Quadrant I */
  for(int i = 0; i < nop; i ++) {
    Point p = points[i];
    if(p.x > 0 && p.y > 0) { ps[count] = p; count ++; } }
  Point[] q1Points = new Point[count];
  /* ps contains null if count < nop */
  for(int i = 0; i < count; i ++) { q1Points[i] = ps[i] }
  return q1Points ;
} }
```




```
Point[] ps = pc.getPointsInQuadrantI();
System.out.println(ps.length); /* 1 */
System.out.println("(" + ps[0].x + ", " + ps[0].y + ")");
/* (3, 4) */
```







Short-Circuit Evaluation: &&

Left Operand op1	Right Operand op2	op1 && op2
true	true	true
true	false	false
false	true	false
false	false	false

```
System.out.println("Enter x:");
int x = input.nextInt();
System.out.println("Enter y:");
int y = input.nextInt();
if(x != 0 && y / x > 2) {
    System.out.println("y / x is greater than 2");
}
else { /* !(x != 0 && y / x > 2) == (x == 0 || y / x <= 2) */
    if(x == 0) {
        System.out.println("Error: Division by Zero");
    }
    else {
        System.out.println("y / x is not greater than 2");
    }
}
```

b1 && b2

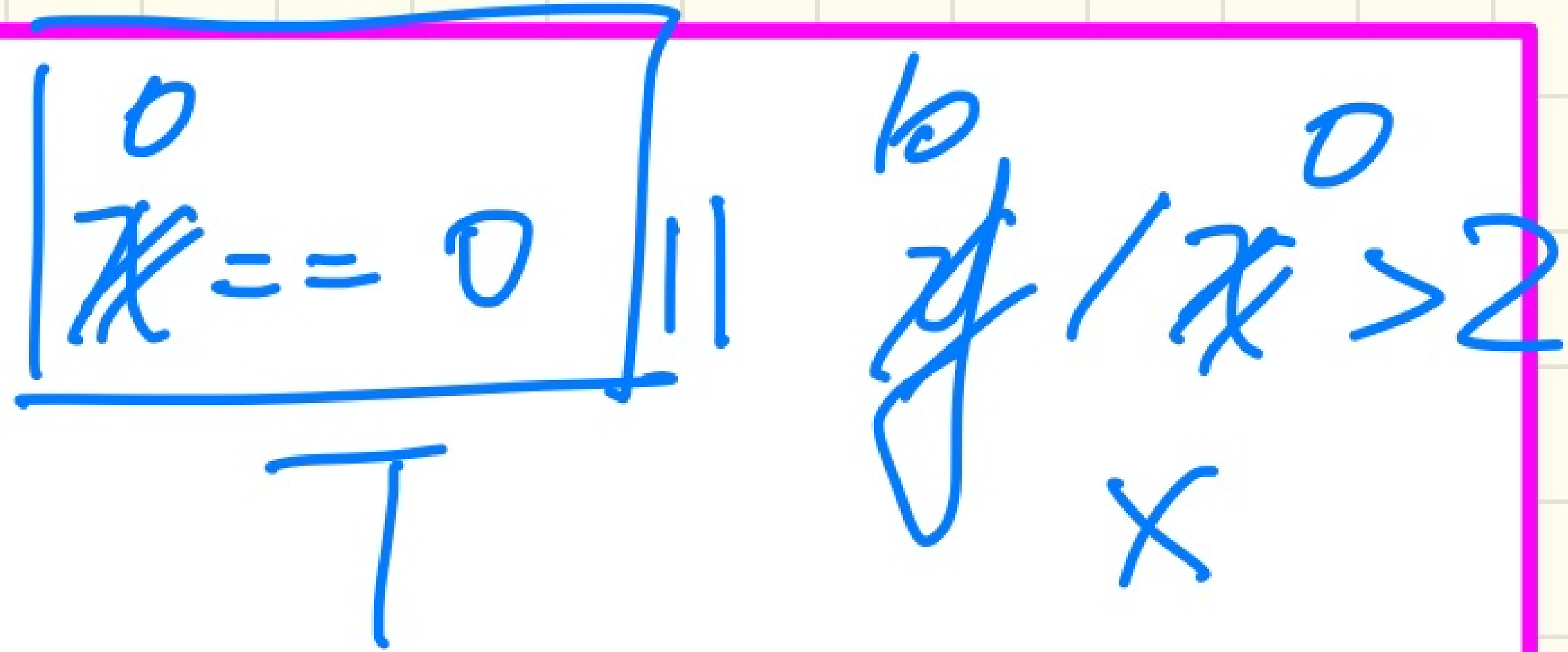
Test Case:
x = 0
y = 10

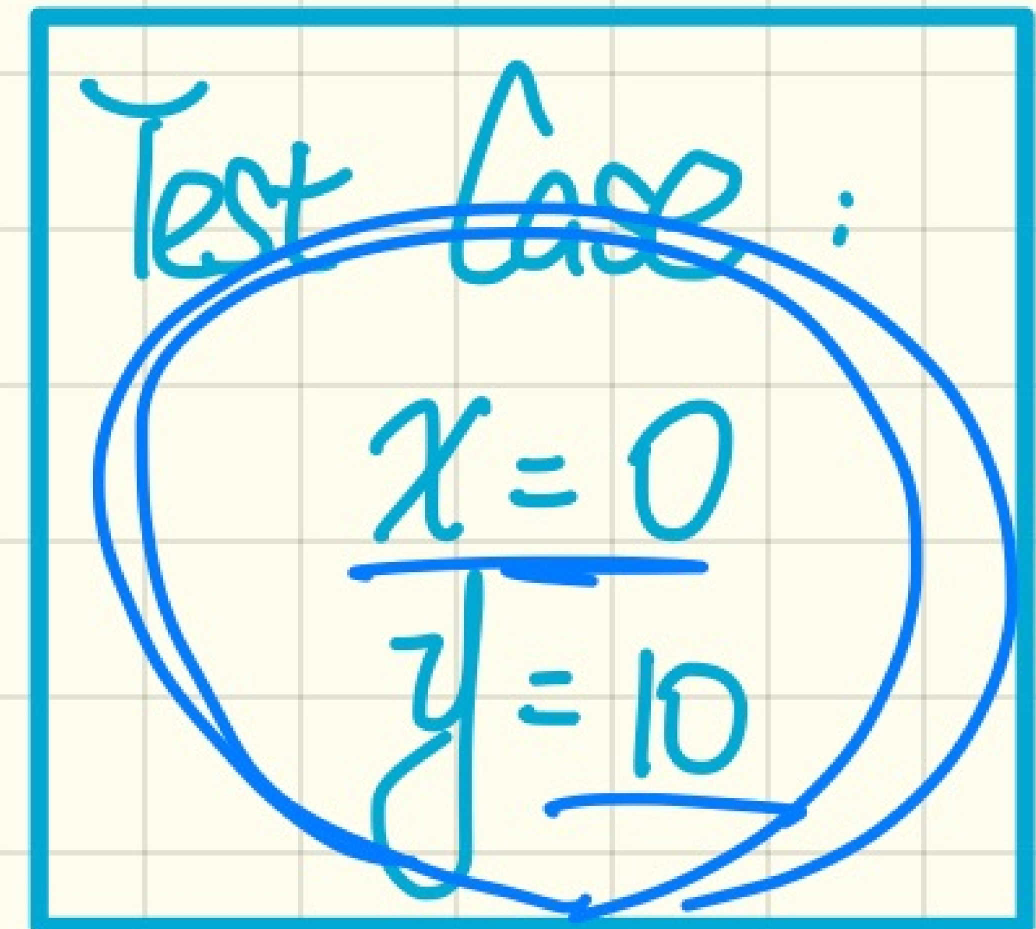
Test Case:
x = 5
y = 10

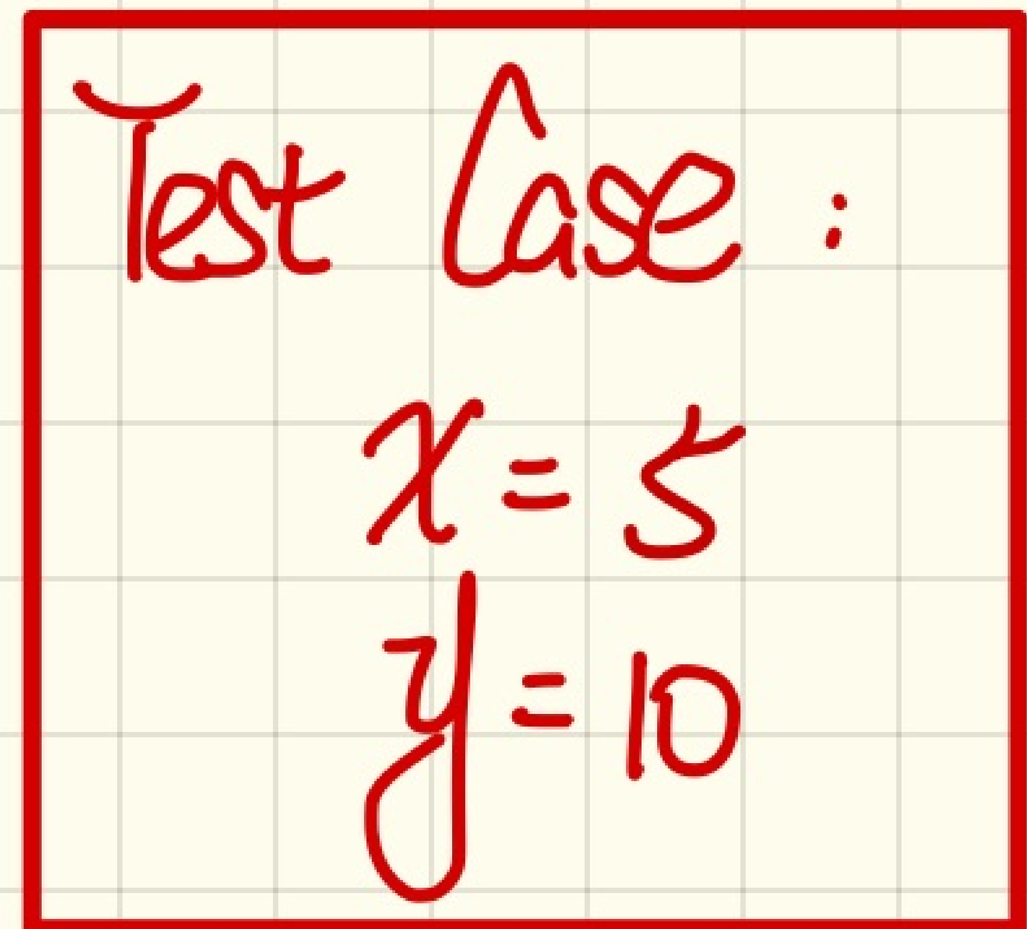
Short-Circuit Evaluation: ||

Left Operand `op1`	Right Operand `op2`	`op1 || op2`
false	*false*	*false*
true	*false*	*true*
false	*true*	*true*
true	*true*	*true*

```java
System.out.println("Enter x:");
int x = input.nextInt();
System.out.println("Enter y:");
int y = input.nextInt();
if(x == 0 || y / x > 2) {
    if(x == 0) {
        System.out.println("Error: Division by Zero");
    }
    else {
        System.out.println("y / x is greater than 2");
    }
}
else { /* !(x == 0 || y / x > 2) == (x != 0 && y / x <= 2) */
    System.out.println("y / x is not greater than 2");
}
```







Short-Circuit Evaluation: Common Errors

Short-Circuit Evaluation is not exploited: crash when `x == 0`

```
if (y / x > 2 && x != 0) {
  /* do something */
}
else {
  /* print error */ }
```

Short-Circuit Evaluation is not exploited: crash when `x == 0`

```
if (y / x <= 2 || x == 0) {
  /* print error */
}
else {
  /* do something */ }
```

Test Case:
x = 0
y = 10

SCE goes from L → R

y / x > 2 && x != 0

y / x <= 2 || x == 0

DBZ

Anonymous Objects

```
1  double square(double x) {
2    double sqr = x * x;
3    return sqr; }
```

```
1  double square(double x) {
2    return x * x; }
```

```
1  Person getP(String n) {
2    Person p = new Person(n);
3    return p; }
```

```
1  Person getP(String n) {
2    return new Person(n); }
```

between line 2 and return line 3, p never used.

```
class Member {
  Order[] orders;
  int noo;
  /* constructor ommitted */
  void addOrder(Order o) {
    orders[noo] = o;
    noo ++;
  }
  void addOrder(String n, double p, double q) {
    addOrder(new Order(n, p, q));
    /* Equivalent implementation:
     * orders[noo] = new Order(n, p, q);
    noo ++; */
  }
}
```

```
class MemberTester {
  public static void main(String[] args) {
    Member m = new Member("Alan");
    Order o = new Order("Americano", 4.7, 3);
    m.addOrder(o);
    m.addOrder(new Order("Cafe Latte", 5.1, 4));
  }
}
```

Overloadding

```
int divide (int x, int y)
double divide (double x, double y)
```

```
void addOrder (Order o) { - - - }
void addOrder (String n, double p, int q)
```

Tester:

```
Member m = - - -
m.addOrder ("A", 2.3, 4);
m.addOrder (
  new Order ("A", 2.3, 4));
```

compilation error ✓

m. addOrder("A", 2.3, 4);

m. addOrder(2.3, 4, "A");

usage.

Overloading

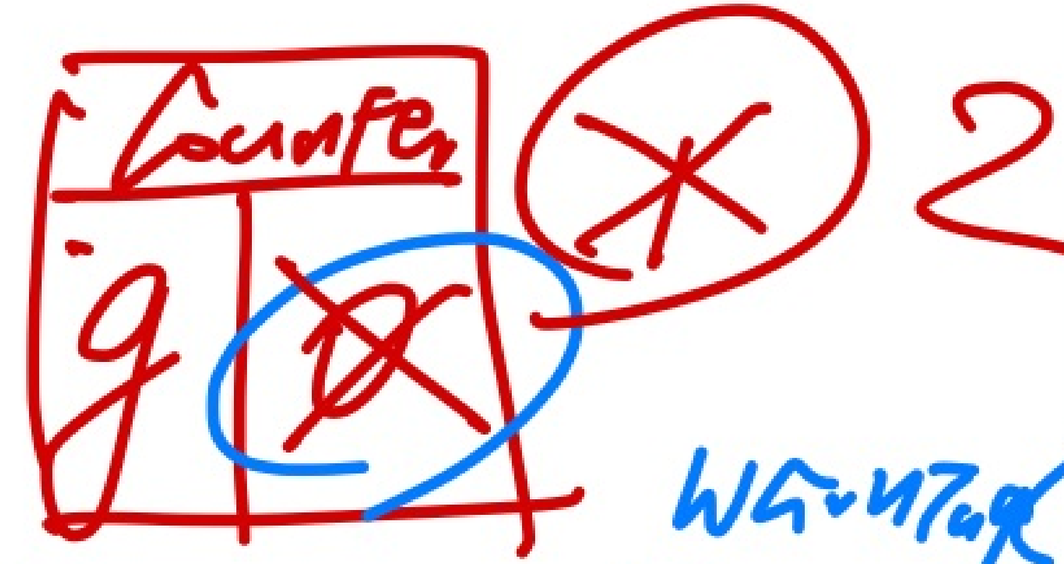
Methods with the same name:

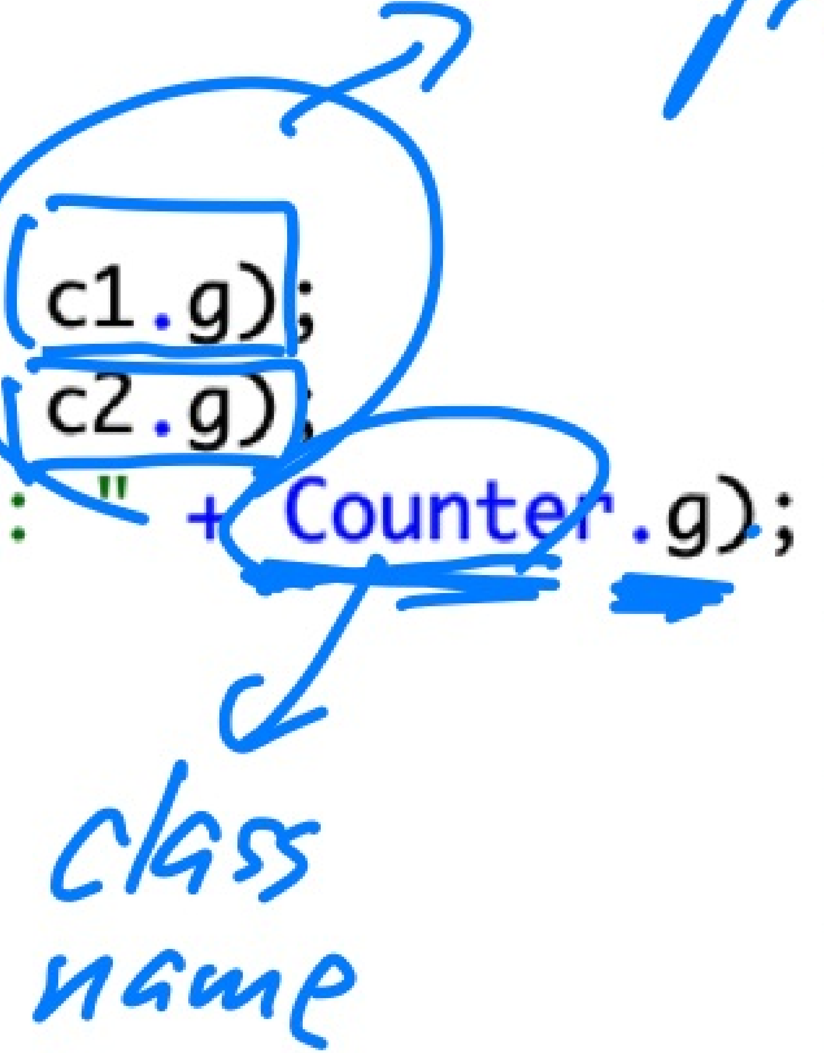
1. different # of parameters
2. same # of parameters, types must be different.

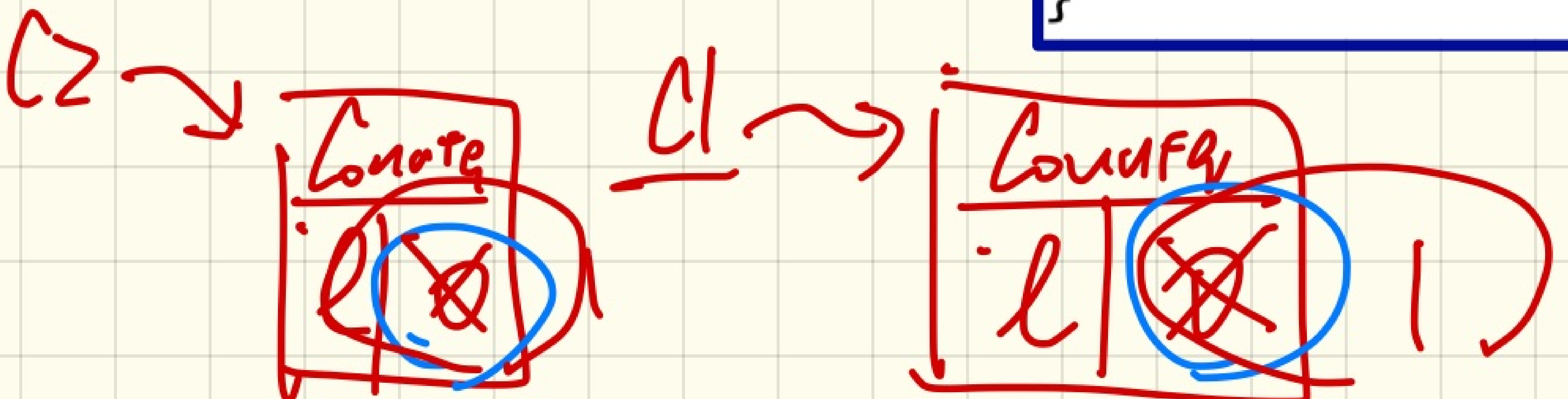
Static vs Non-Static Variables

```java
public class Counter {
  int l;
  static int g = 0;

  Counter() {
    l = 0;
  }

  void incrementLocal() {
    l ++;
  }

  void incrementGlobal() {
    g ++;
  }
}
```

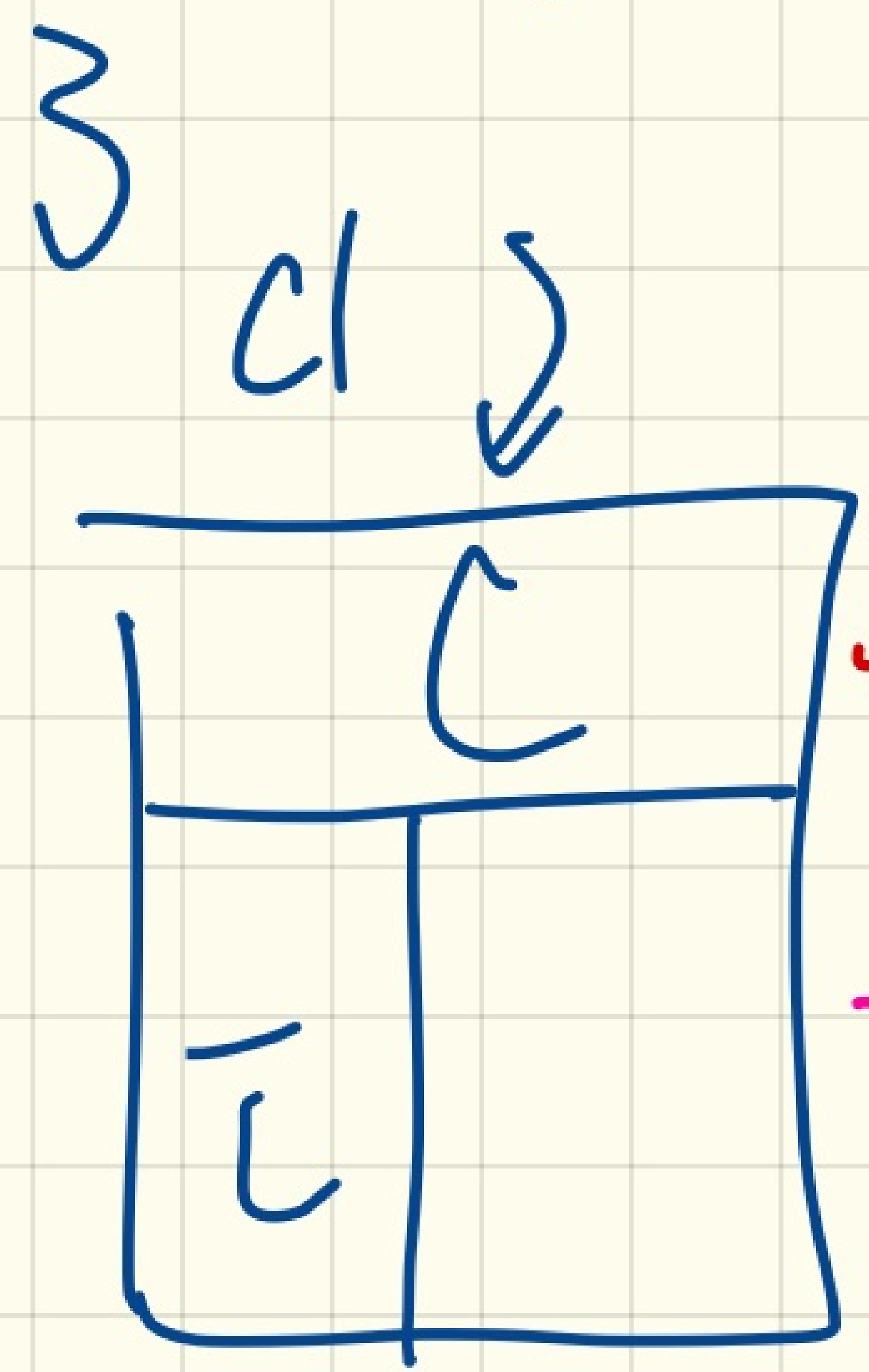
```java
public class CounterTester {

    public static void main(String[] args) {
        Counter c1 = new Counter();
        Counter c2 = new Counter();

        System.out.println("c1's local: " + c1.l);
        System.out.println("c2's local: " + c2.l);
        System.out.println("Global accessed via c1: " + c1.g);
        System.out.println("Global accessed via c2: " + c2.g);
        System.out.println("Global accessed via Counter: " + Counter.g);

        c1.incrementLocal();

        c2.incrementLocal();

        c1.incrementGlobal();

        c2.incrementGlobal();

        Counter.g = Counter.g + 1; // Counter.global ++;
    }

}
```







```
class C {
  static int si;
  int i;
}
```



C

si 0

static variables

↳ 1. belongs to the class

↳ 2. global to all objects

→ non-static variables (attributes)

↳ 1. belongs to an instance

2. local