Lecture 3

Wednesday September 11

- Notes on a Programming Pattern
  (Point, PointCollector, PointTester)
- Java Tutorial Series

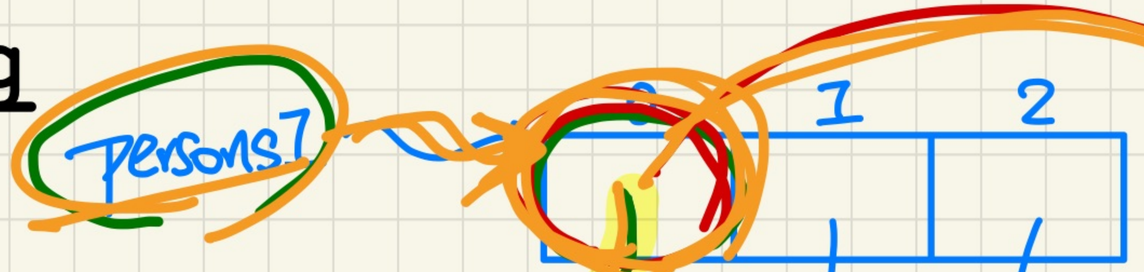
Arrays and Aliasing

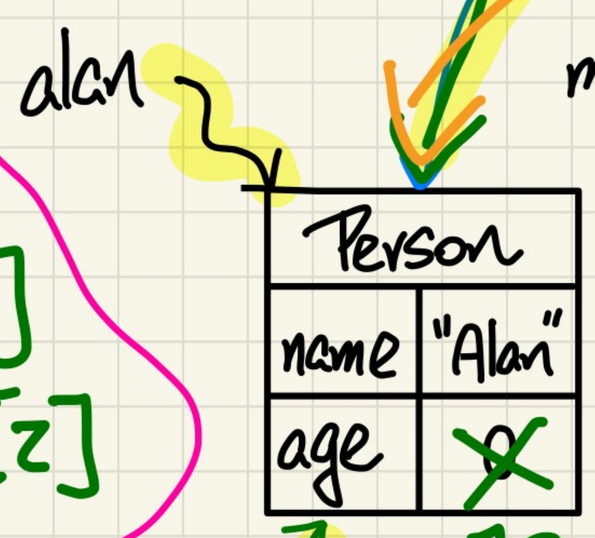
All alias paths to "Alan"?

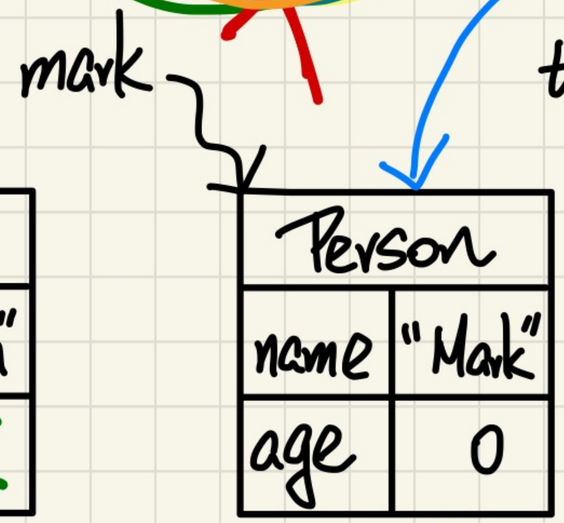
alan
persons1[0]
persons2[2]

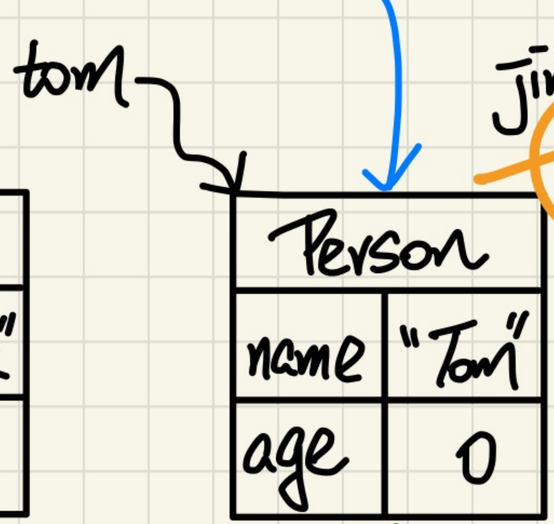
alan == persons1[0]

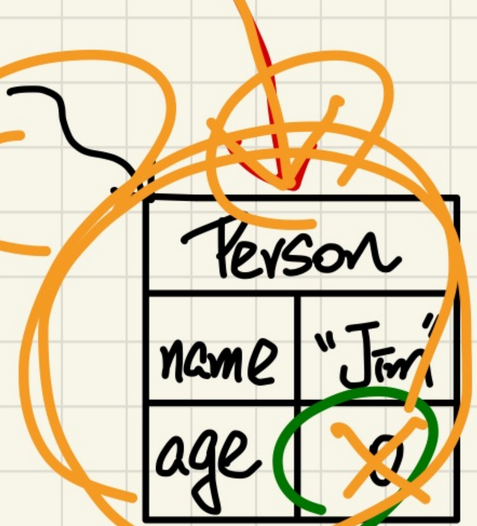
T.

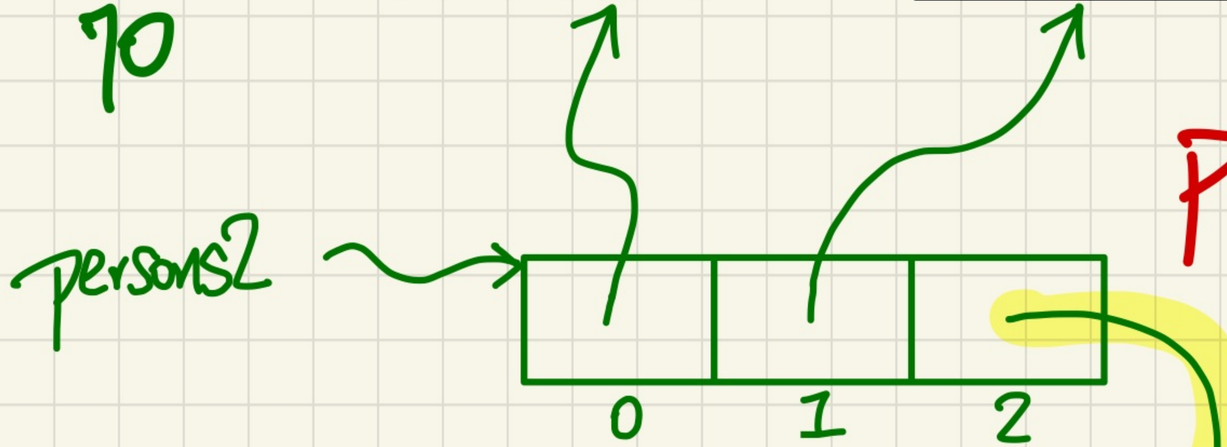








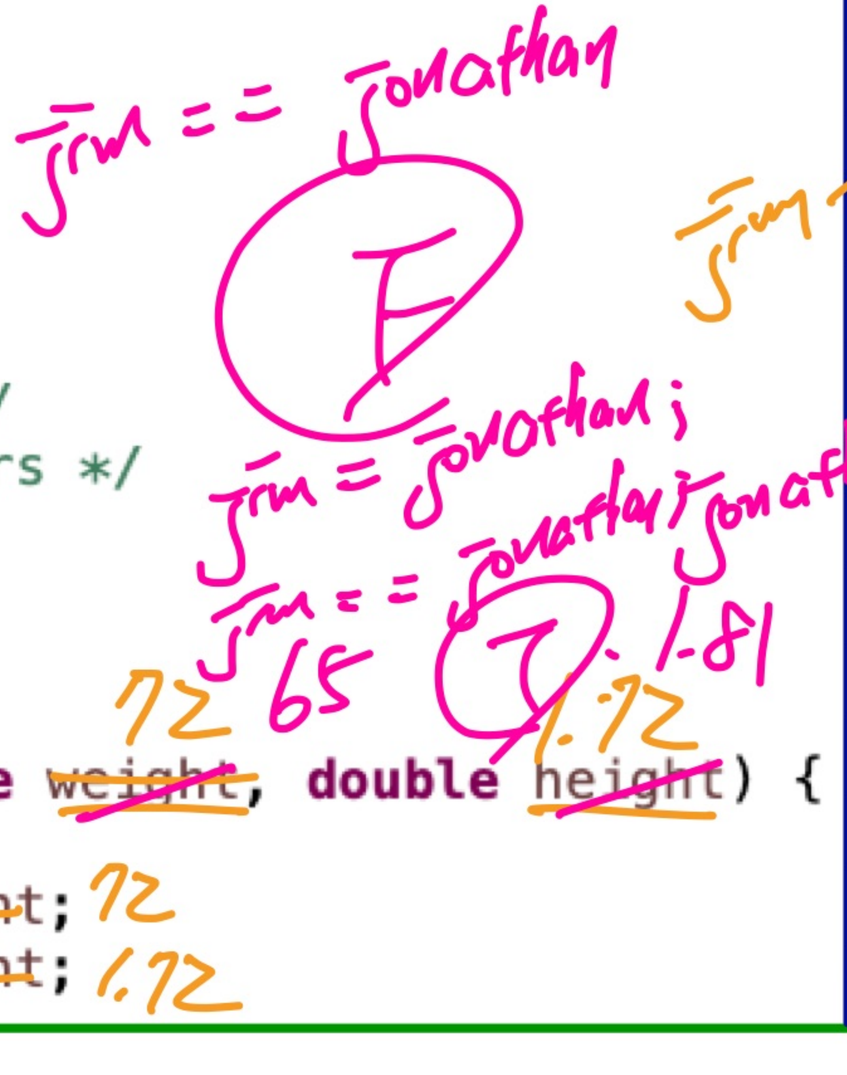




persons1[0] = jim
persons1[0].setAge(75)

Constructors using this Keyword

```java
public class Person {
    /*
     * Attributes
     */
    int age;
    String nationality;
    double weight; /* kg */
    double height; /* meters */

    /*
     * Constructors
     */
    Person (int age, double weight, double height) {
        this.age = age;
        this.weight = weight;
        this.height = height;
    }
```




```java
    public static void main(String[] args) {
        Person jim = new Person(45, 72, 1.72);
        Person jonathan = new Person(62, 65, 1.81);
    }

}
```

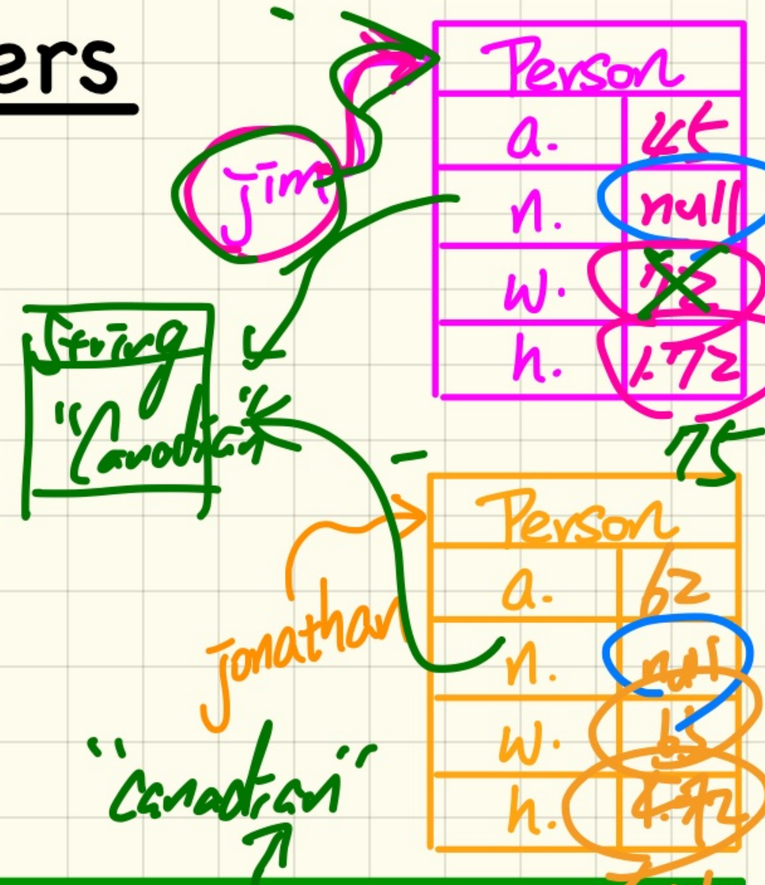
Accessors/Getters vs. Mutators/Setters

```java
public class Person {
    int age;
    String nationality;
    double weight; /* kg */
    double height; /* meters */

    double getBMI() {
        double bmi = this.weight / (this.height * this.height);
        return bmi;
    }

    void gainWeight(double amount) {
        this.weight = this.weight + amount;
    }
}
```

jim == jonathan (F)

jim.age == jonathan.age (F)

jim.nationality == jonathan.nat. (T)




```java
Person jim = new Person(45, 72, 1.72);
Person jonathan = new Person(62, 65, 1.81);

double jimBMI = jim.getBMI();
double jonathanBMI = jonathan.getBMI();
System.out.println("Jim's BMI: " + jimBMI);
System.out.println("Jonathan's BMI: " + jonathanBMI);

jim.gainWeight(3);
jonathan.gainWeight(3);

jimBMI = jim.getBMI();
jonathanBMI = jonathan.getBMI();
System.out.println("Jim's BMI: " + jimBMI);
System.out.println("Jonathan's BMI: " + jonathanBMI);
```

main (. . -) {

double d1 = 2.0;

double d2 = 2.0;

String s1 = "akn"; d1 == d2

String s2 = "alan";

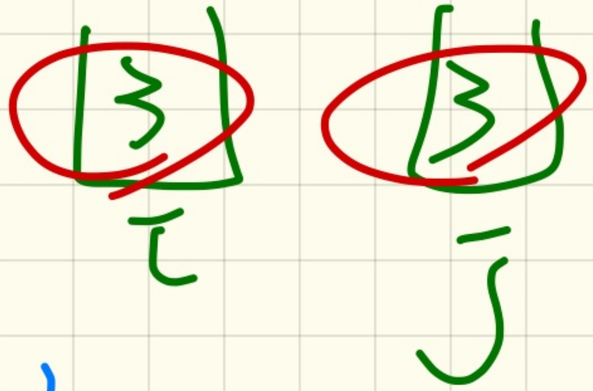
s1 == s2 (T).

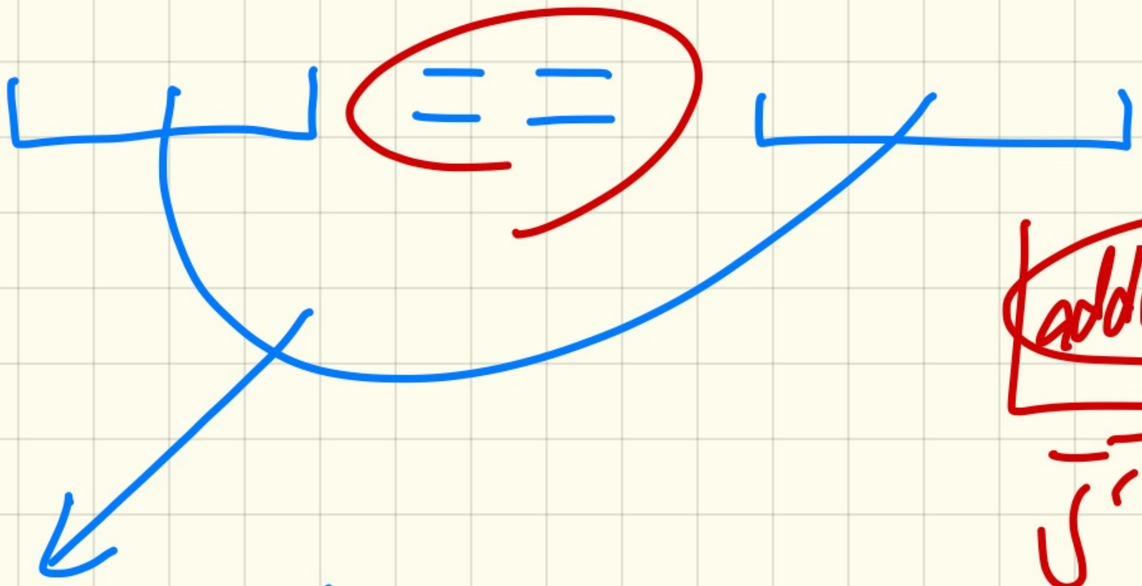
s1 → "akn"

s2 }

int i = 3;

int j = 3;





1. both are primitives (int)

↳ compare values

2. both are references (jim, jonathan)

↳ compare addresses

OOP: Use of Accessors vs Use of Mutators

- Calls to *mutator methods* *cannot* be used as values.
  - e.g., `System.out.println(jim.setWeight(78.5));` ×
  - e.g., `double w = jim.setWeight(78.5);` ×
  - e.g., `jim.setWeight(78.5);` ✓

- Calls to *accessor methods* *should* be used as values.
  - e.g., `jim.getBMI();`
  - e.g., `System.out.println(jim.getBMI());`
  - e.g., `double w = jim.getBMI();`

valid but useless

```
void setWeight(...) {...}
double getBMI() {...}
```

OOP: Choice of Method Parameters

- **Principle 1:** A *constructor* needs an *input parameter* for every attribute that you wish to initialize.

  e.g., `Person(double w, double h)` vs. `Person(String fName, String lName)`

- **Principle 2:** A *mutator* method needs an *input parameter* for every attribute that you wish to modify.

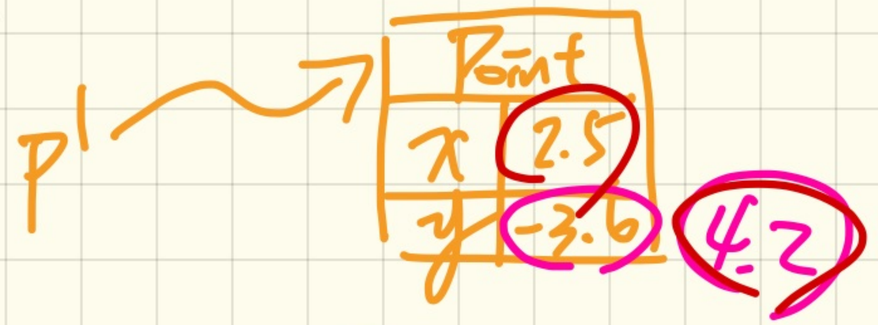
  e.g., In `Point`, `void moveToXAxis()` vs. `void moveUpBy(double unit)`

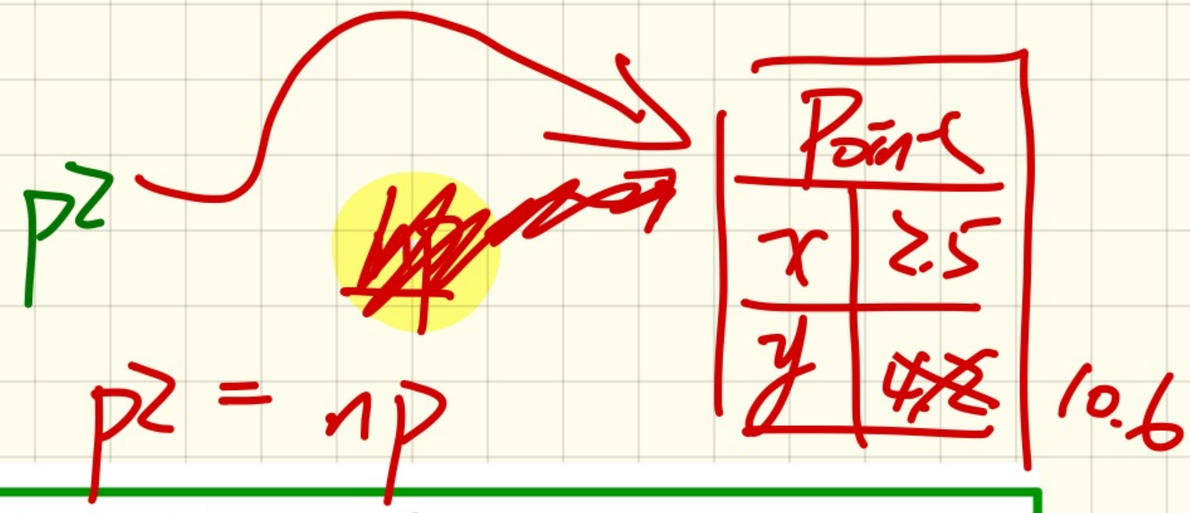
- **Principle 3:** An *accessor method* needs *input parameters* if the attributes alone are not sufficient for the intended computation to complete.

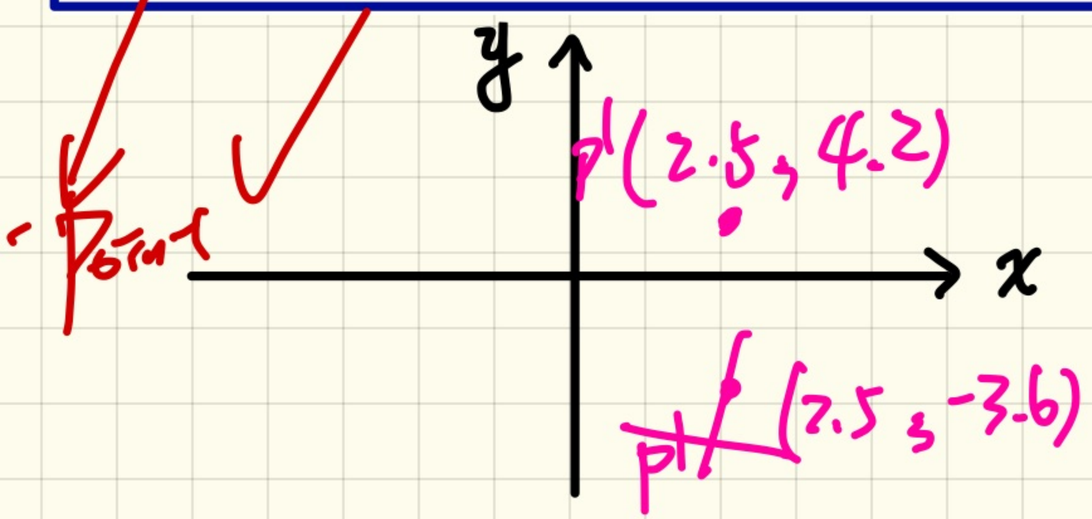
  e.g., In `Point`, `double getDistFromOrigin()` vs. `double getDistFrom(Point other)`

Return Type: Reference Type

```
class Point {
   Point(double x, double y) {...}

   void moveUpBy(double units) {
      this.y = this.y + units;
   }

   Point movedUpBy(double units) {
      Point np = new Point(this.x, this.y);
      np.moveUpBy(units);
      return np;
   }
}
```








```
class PointTester {
   static void main(String[] args) {
      Point p1 = new Point(2.5, -3.6);
      p1.moveUp(7.8);
      Point p2 = p1.movedUpBy(6.4);
      System.out.println(p1 == p2);
   }
}
```

np

F

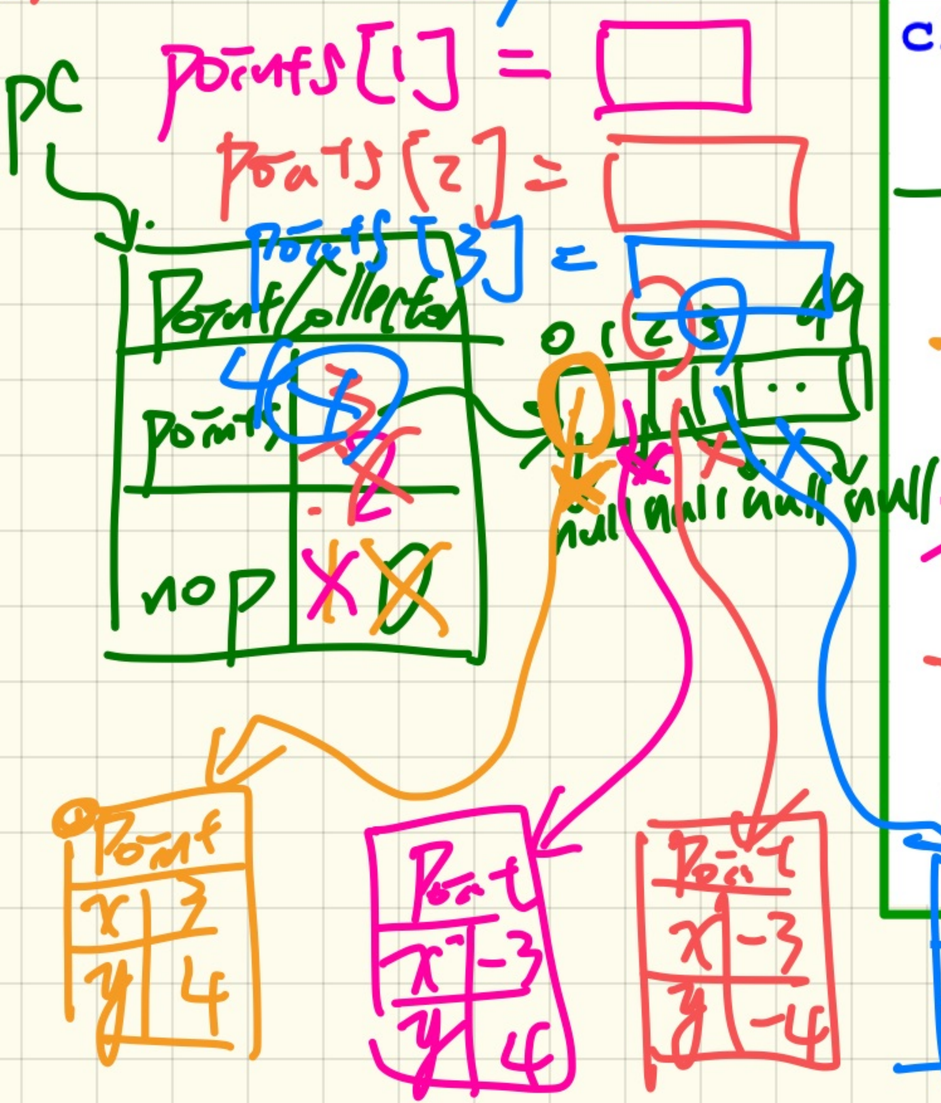
Programming Pattern: Mutator

array of Point addresses.

nop: 1. how points have been stored
2. where to store next point

```
class PointCollector {
 Point[] points; int nop; /* number of points */
 PointCollector() { this.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 */
```



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");
    }
}
```

P && Q

E left to right

Test Case :

x = 0

y = 10

Test Case :

x = 5

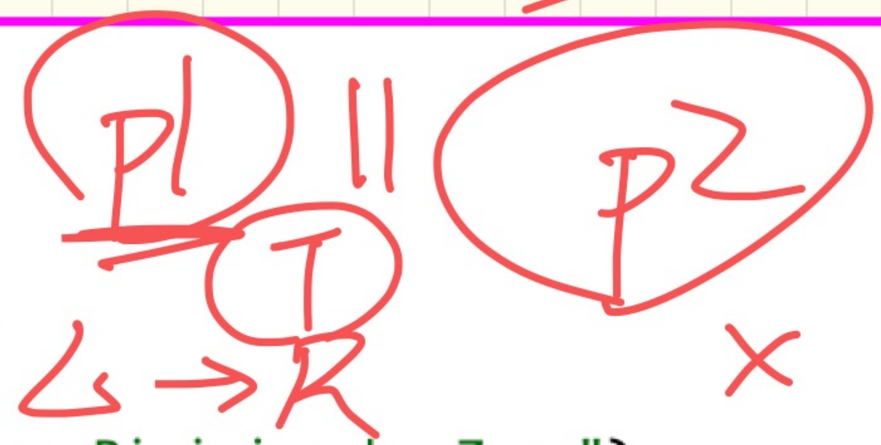
y = 10

Short-Circuit Evaluation: ||

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

Guard

```
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");
}
```



Test Case :

$x = 0$

$y = 10$

Test Case :

$x = 5$

$y = 10$