# จาวา – Classes & Objects

**สมชาย ประสิทธิ์จูตระกูล**

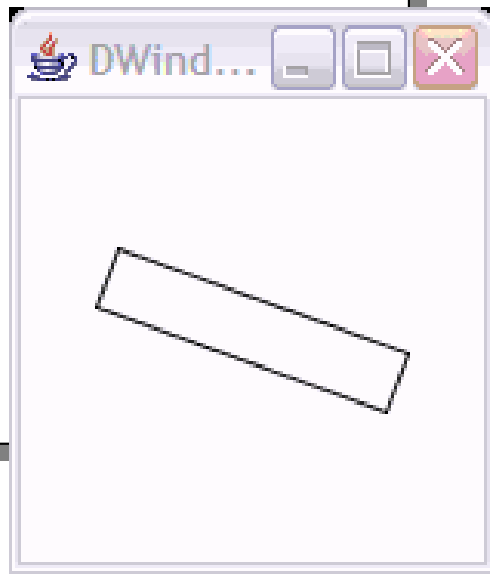
# หัวข้อ

- คลาสและออบเจ็กต์
- องค์ประกอบของคลาส
- Fields
- Methods
- Constructors
- Overloading

# คลาสและออปเจกต์

```
import jlab.graphics.*;

public class Demo {
  public static void main(String[] args) {
    DWindow w = new DWindow(200, 200);
    DRectangle r = new DRectangle(100, 20);
    w.add(r);
    r.setCenter(w.getCenter());
    while (true) {
      Util.sleep(50);
      r.rotate(2);
    }
  }
}
```




objectReference . methodName( arguments )

# องค์ประกอบของคลาส

```
class Point {
  int x;
  int y;

  Point() { this(0, 0); }
  Point(int x, int y) {
    this.x = x;
    this.y = y;
  }

  void setXY(int x, int y) {
    this.x = x; this.y = y;
  }
  double distance(int x, int y) {
    double dx = this.x - x;
    double dy = this.y - y;
    return Math.sqrt(dx*dx + dy*dy);
  }
}
```

fields

constructors

methods

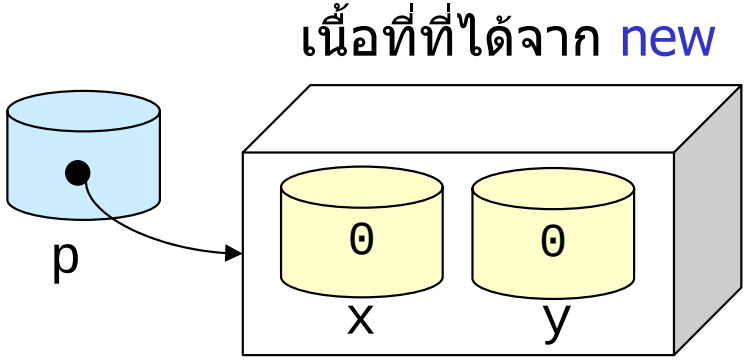
# การประกาศและการสร้าง

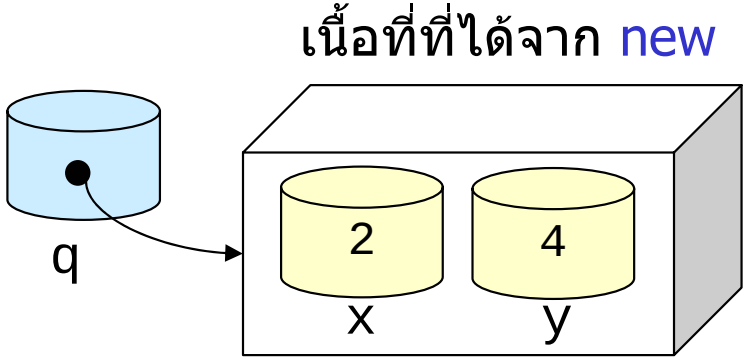
```
class Point {
  int  x, y;
  Point() { this(0, 0); }
  Point(int x, int y) {
    this.x = x;
    this.y = y;
  }
  ...
}
```

นิยามของคลาส

```
  Point  p;
  p = new Point();
```

```
  Point  q = new Point(2, 4);
```



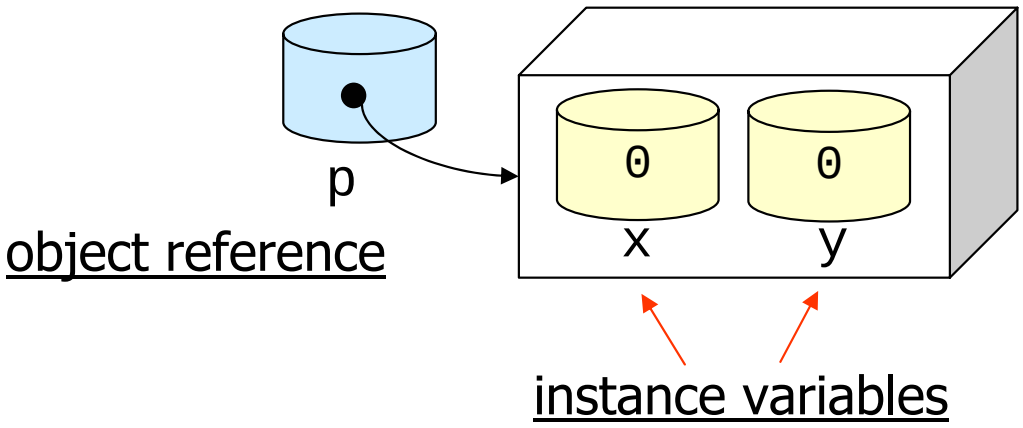



# ศัพท์เทคนิค

```
class Point {  // class
  int  x, y;   // fields
  ...
}
```

```
Point  p = new Point();
```

ตัวแปรอ้างอิง object
ของ class Point

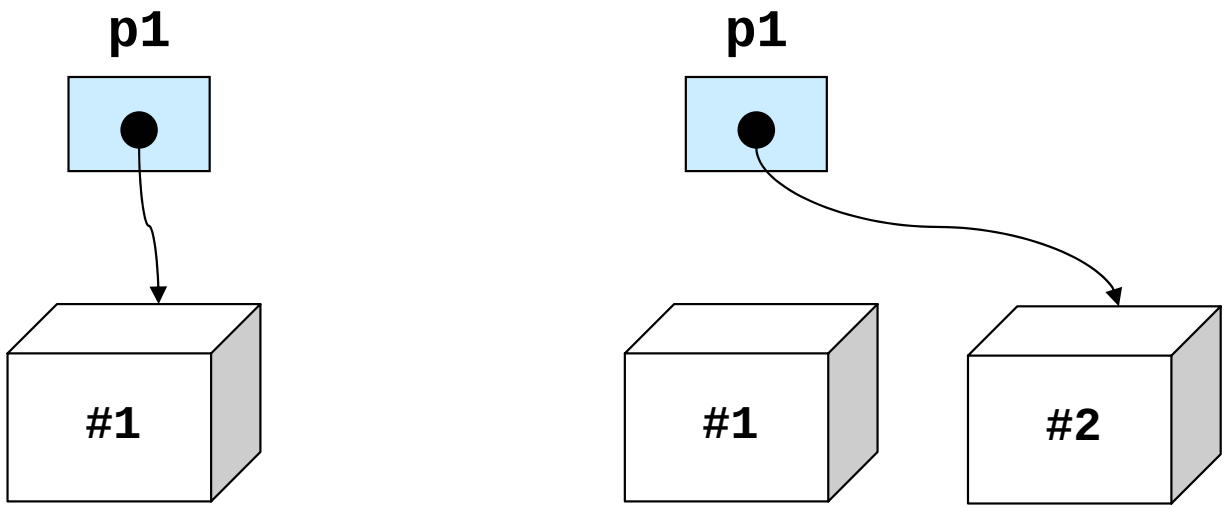



object (หรือ instance) ของ
คลาส Point

# ขยะ

- object ที่ถูกสร้างแต่ไม่มีตัวแปรใดอ้างอิง เป็นขยะ
- ระบบจาวาจะ "เก็บขยะ" นำเนื้อที่เหล่านี้ไปใช้ใหม่

```
Point p1 = new Point(1, 2);  // #1
...
p1 = new Point(1, 3);        // #2
```



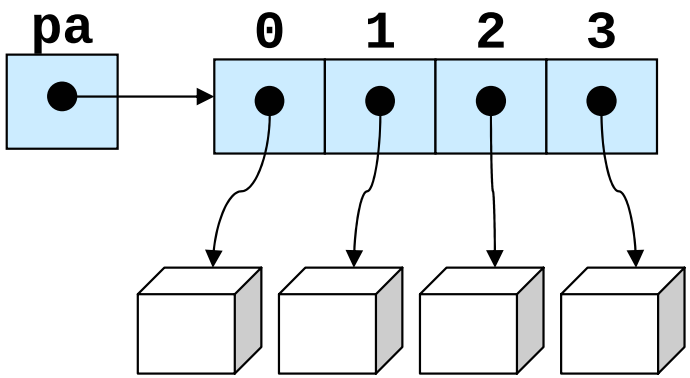
# การสร้างและทำลาย object

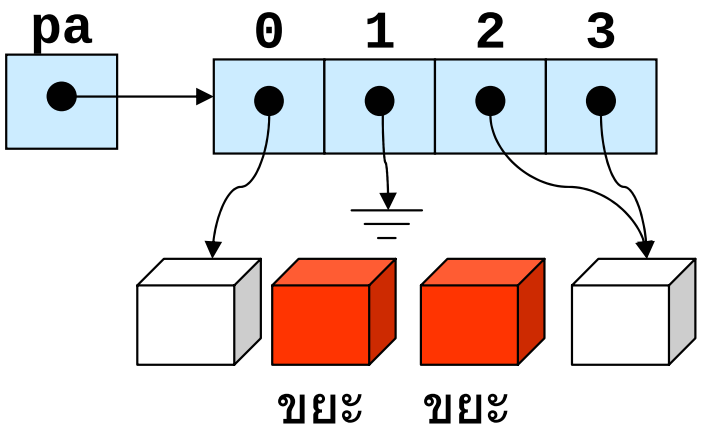
- สร้างโดยใช้ new
- ทำลายโดยอย่าให้มีตัวแปรใดไปอ้างอิงถึง

```
Point [] pa  = new Point[4];
for(int i=0; i< pa.length; i++) {
  pa[i] = new Point();
}

pa[2] = pa[3];
pa[1] = null;
```

null เป็นค่าคงตัวพิเศษแทน object reference ที่ไม่ได้อ้างอิงอะไร



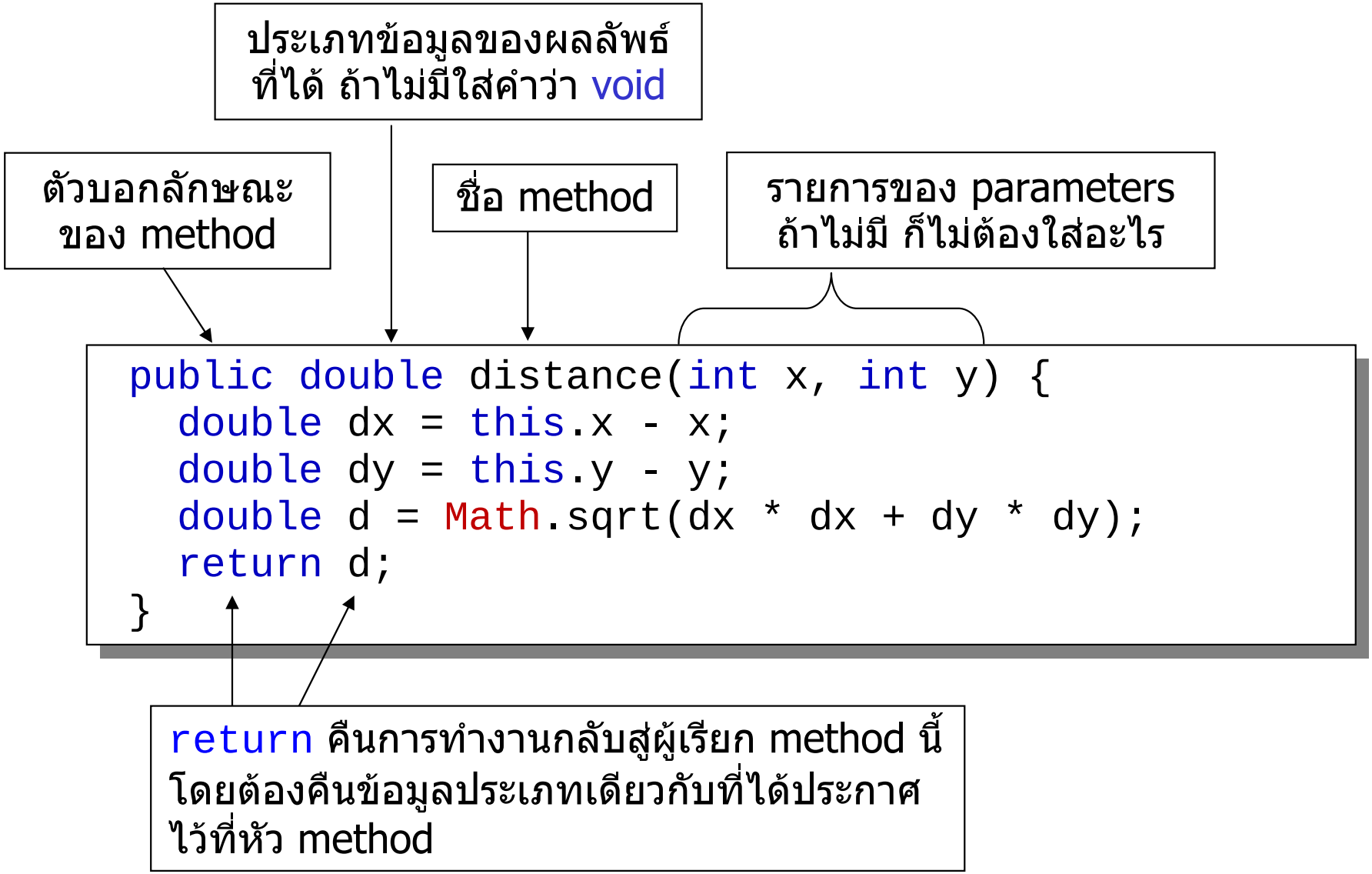
# การใช้ Instance Variables

- instance variables คือตัวแปรต่างๆ ภายในของ object ที่ได้ถูกสร้างขึ้น
- อยากจะใช้ ให้เขียนในรูป *object* . *variableName*

```
class Point {
  int  x, y;
  ...
}
```

```
Point p1 = new Point(3, 4);
Point p2 = new Point(5, 7);
...
double dx = p1.x - p2.x;
double dy = p1.y - p2.y;
double d = Math.sqrt(dx * dx + dy * dy);
```

# การเขียนเมท็อด



# this

this แทนออปเจกต์ที่ถูกเรียกเมท็อด

```
class Point  {
  int x, y;
  Point(int x, int y) { this.x = x; this.y = y; }

  double distance(int x, int y) {
    double dx = this.x - x;
    double dy = this.y - y;
    double d = Math.sqrt(dx * dx + dy * dy);
    return d;
  }
}
```

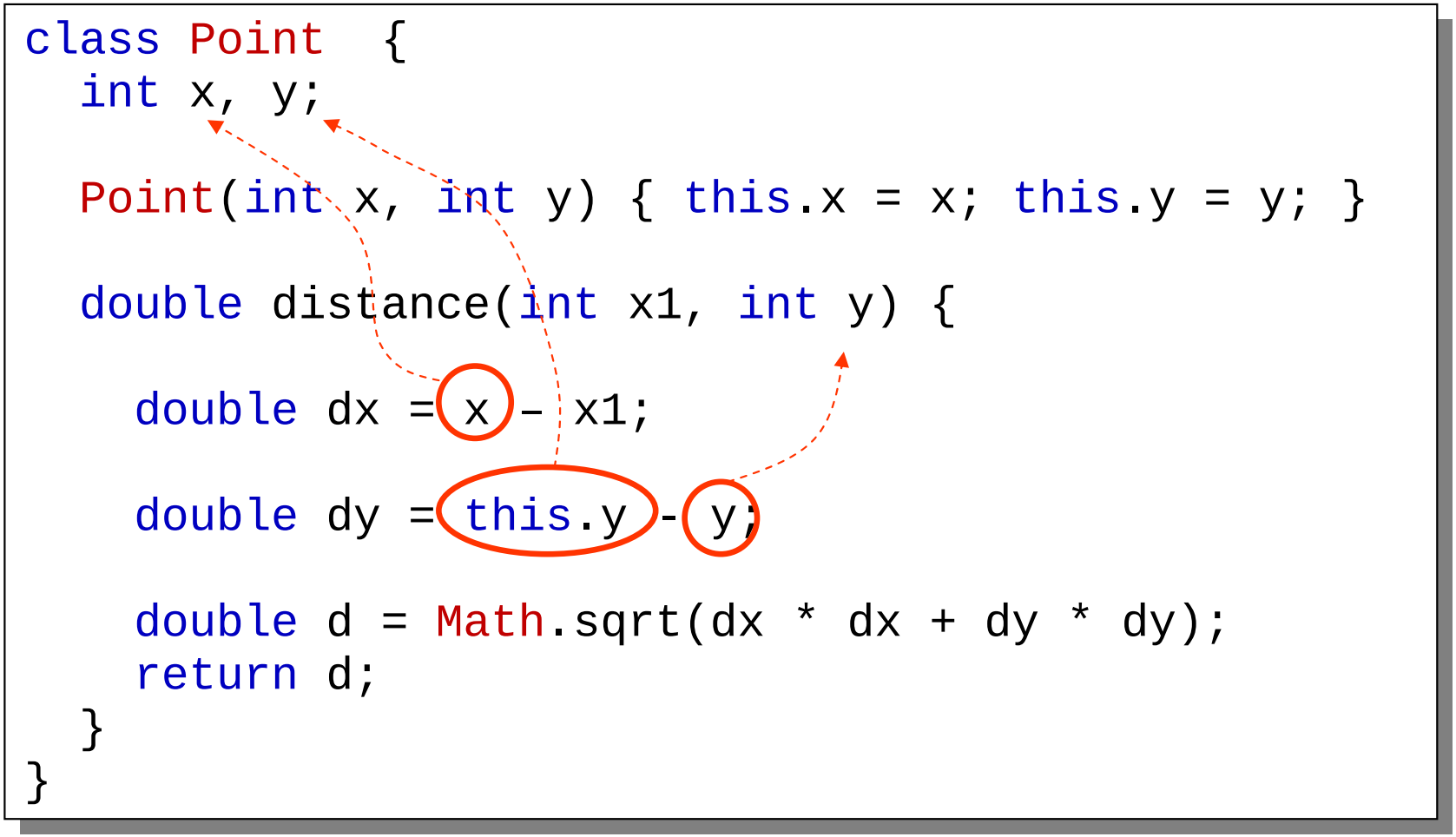
this

```
class Test {
  public static void main(String[] a) {
    Point p = new Point(2, 3);

    double d = p.distance(4, 6);

    System.out.println(d);
  }
}
```

# ละ this. ได้ถ้าไม่กำกวม



```java
class Point  {
  int x, y;
  Point(int x, int y) { this.x = x; this.y = y; }

  double distance(int x, int y) {
    double dx = this.x - x;
    double dy = this.y - y;
    double d = Math.sqrt(dx * dx + dy * dy);
    return d;
  }
  void setXY(int x, int y) {
    this.x = x;
    this.y = y;
  }
}
```

```java
public static void main(String[] a) {
  Point p = new Point(2, 3);
  p.setXY(3, 8);
  double d = p.distance(4, 6);
  System.out.println(p.x);
  System.out.println(d);
}
```

# Overloading

- ชื่อเมท็อดซ้ำได้ ถ้า parameter list ไม่เหมือนกัน

```
class Point  {
  int x, y;
  Point(int x, int y) { this.x = x; this.y = y; }

  double distance(int x, int y) {
    double dx = this.x - x;
    double dy = this.y - y;
    double d = Math.sqrt(dx * dx + dy * dy);
    return d;
  }

  double distance(Point p) {
    return this.distance(p.x, p.y);
  }
  ...
}
```

# Constructors

- เป็นเมท็อดพิเศษ ที่ระบบเรียกเมื่อมีการ new
- มีไว้เพื่อตั้งค่าเริ่มต้นให้กับ instance variables ของ object ใหม่ซึ่งเพิ่งถูกสร้าง
- เป็นเมท็อดพิเศษ
  - ชื่อต้องเหมือนชื่อ class
  - ไม่ระบุ return type ของเมท็อด

```
class Point {
  int  x, y;

  Point(int x, int y) {
    this.x = x;
    this.y = y;
  }
  ...
}
```

# Constructors

- overload constructors ได้

```
class Point {
  int  x, y;

  Point() {
    this.x = this.y = 0;
  }
  Point(int x, int y) {
    this.x = x;
    this.y = y;
  }
  Point(Point p) {
    this.x = p.x;
    this.y = p.y;
  }
  ...
}
```

```
public static void main(String[] a) {
  Point p1 = new Point();
  Point p2 = new Point(2, 33);
  Point p3 = new Point(p2);
  ...
}
```

# Constructors

- Constructor เรียกกันเองได้ โดยใช้ this( ... )
- การเรียก this( ... ) ในลักษณะนี้
  - ต้องปรากฏที่บรรทัดแรกของเมท็อดเท่านั้น
  - ใช้ได้เฉพาะใน constructors ด้วยกันเท่านั้น

```
class Point {
  int x, y;
  Point() {
    this(0, 0);
  }
  Point(int x, int y) {
    this.x = x;
    this.y = y;
  }
  Point(Point p) {
    this(p.x, p.y);
  }
  ...
```

# No-Arg Constructors

- ถ้าไม่เขียน constructor สักตัว ระบบจะเพิ่ม no-arg constructor ที่ไม่มีอะไรภายใน

```
class Point  {
  int x, y;
  ...
}
```

```
class Point  {
  int x, y;

  public Point() {}
  ...
}
```

# การให้ค่าเริ่มต้นกับ instance variable

```
class Point {
  int x, y;

  Point() {
    this.x = this.y = 0;
  }
  ...
}
```

```
class Point {
  int x = 0, y = 0;

  Point() {
  }
  ...
}
```

ใส่ค่าเริ่มต้นเองก็ได้ เช่นเดียวกับการตั้งค่าเริ่มต้นของตัวแปรภายในเมท็อด

```
class Point {
  int x, y;

  Point() {
  }
  ...
}
```

ปกติเมื่อ object หนึ่งถูกสร้างขึ้น จะมีการให้ค่า 0 null และ false กับ instance variables ต่างๆ ของ object

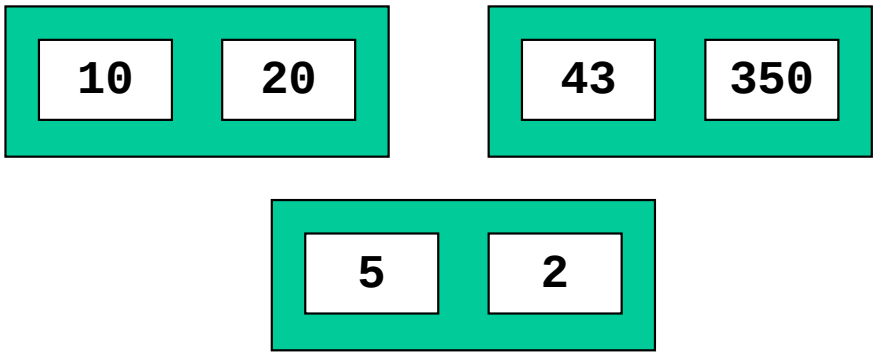
# Static Fields

- static field คือที่เก็บข้อมูลประจำคลาส
- ไม่จำเป็นต้องมีออปเจกต์ใดๆ ก็ใช้ static fields ได้

```
class Point {
  static int nPoints;
  int x, y;

  Point(int x, int y) {
    this.x = x;
    this.y = y;
    ++(Point.nPoints);
  }
  ...
}
```

```
Point p1 = new Point(10, 20);
Point p2 = new Point(5, 2);
Point p3 = new Point(43, 350);
```




การอ้างอิง static field สามารถใช้วิธีปกติ (ที่ทำผ่าน object reference) หรือจะใช้ชื่อคลาสตามด้วยจุด ตามด้วยชื่อ static field (ข้อแนะนำ : ควรใช้วิธีหลัง)

# Static Methods

- เมท็อดของคลาสที่ถูกเรียกใช้ได้โดยไม่ต้องมีออปเจกต์ของคลาสนั้น

การอ้างอิง static method สามารถใช้วิธีปกติ (ที่ทำผ่าน object reference) หรือจะใช้ ชื่อคลาสตามด้วยจุด ตามด้วยชื่อ static method (ข้อแนะนำ : ควรใช้วิธีหลัง)

```
class Point {
  int x, y;
  double distance(int
    double dx = this
    double dy = this.y - y;
    double d = Math.sqrt(dx * dx + dy * dy);
    return d;
  }
  double distance(Point p) {
    return this.distance(p.x, p.y);
  }
  static double distance(Point p1, Point p2) {
    double dx = p1.x - p2.x;
    double dy = p1.y - p2.y;
    return Math.sqrt(dx * dx + dy * dy);
  }
}
```

# ใช้ static methods และ static fields ได้เสมอ

```
class Point {
  int x, y;
  double distance(int x, int y) {
    double dx = this.x - x;
    double dy = this.y - y;
    double d = Math.sqrt(dx * dx + dy * dy);
    return d;
  }
  double distance(Point p) {
    return Point.distance(this, p);
  }
  static double distance(Point p1, Point p2) {
    double dx = p1.x - p2.x;
    double dy = p1.y - p2.y;
    return Math.sqrt(dx * dx + dy * dy);
  }
}
```

# ไม่มี this ให้อ้างอิงใน class methods

```
class Point {
  int x, y;
  double distance(int x, int y) {
    double dx = this.x - x;
    double dy = this.y - y;
    double d = Math.sqrt(dx * dx + dy * dy);
    return d;
  }
  double distance(Point p) {
    return this.distance(p.x, p.y);
  }
  static double distance(Point p1, Point p2) {
    double d1 = p1.distance(p2.x, p2.y); //correct
    d1 = p1.distance(p2);                //correct
    d1 = this.distance(p2);              //wrong
    d1 = distance(p2);                   //wrong
    return p1.distance(x, this.y); //wrong
  }
}
```

# Lab 2 : Rational Number

```
Rational r1 = new Rational(3, 5);  //    3/5
Rational r2 = r1.reciprocate();    //    5/3
Rational r3 = r1.add(3);           //    18/5
Rational r4 = r1.add(r3);          //    21/5
Rational r5 = r1.multiply(r2)      //    1/1
```

$$\frac{\text{numerator}}{\text{denominator}}$$

หลังจากสร้างออปเจกต์แบบ Rational แล้ว ไม่มี method ของ Rational ที่เปลี่ยนค่าภายในออปเจกต์ได้ (เรียกว่า immutable object)

ทุก method (reciprocate, add, multiply) คืนออปเจกต์ใหม่ที่แทนผลลัพธ์

```
public class Rational {
  int numerator, denominator;

  Rational() { this(0, 1); }
  Rational(int n, int d) {
    this.numerator = n;
    this.denominator = d;
    simplify();
  }
  Rational reciprocate() {  ...  }
  Rational add(int a) {  ...  }
  Rational add(Rational a) {  ...  }
  Rational multiply(Rational a) {  ...  }
  void simplify() {
    int gcd = gcd(numerator, denominator);
    numerator /= gcd;
    denominator /= gcd;
  }
  static int gcd(int a, int b) {
    return b == 0 ? a : gcd(b, a % b);
  }
}
```

# Lab 2

ทดสอบ : Run JUnit ของ lab2