Introduction to Java

POWERPOINT BY, ASSOC. PROF. RANGSIT SIRIRANGSI

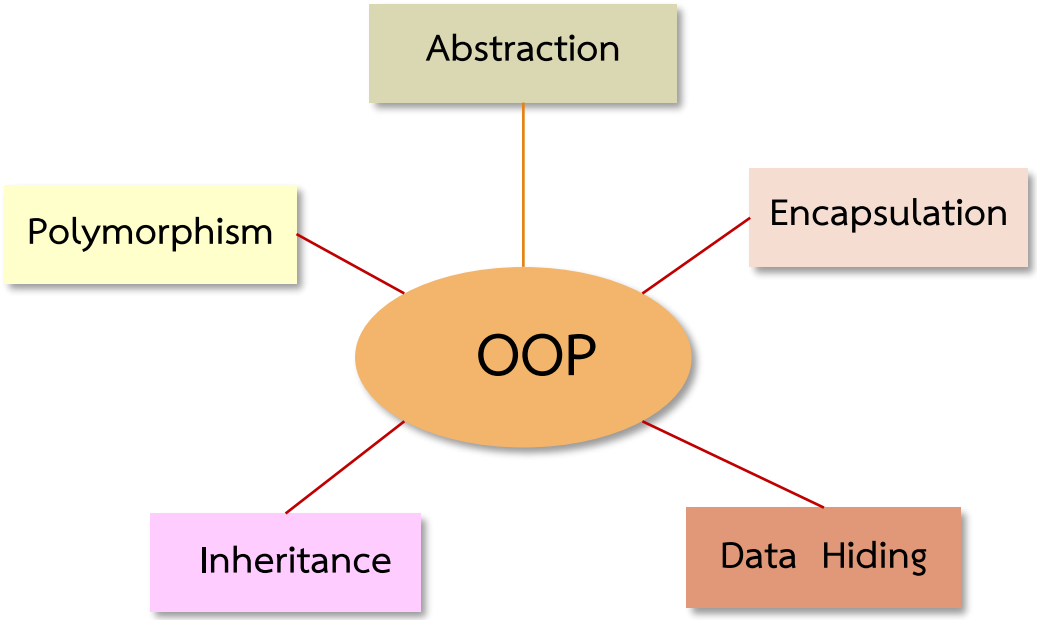
PRESENT BY DR,. CHIRAWAN RONRAN

INFORMATION TECHNOLOGY MJU

Object Oriented Concepts

ในทุก ๆ โปรแกรมภาษาที่สนับสนุนแนวคิดเชิงวัตถุ จะต้องประกอบไปด้วยคุณสมบัติพื้นฐานดังต่อไปนี้

- Abstraction
- Encapsulation
- Data Hiding
- Inheritance
- Polymorphism



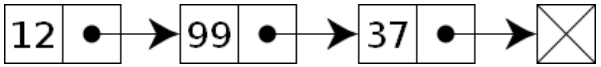
What Is an Object?

เป็นตัวแทนของสิ่งใดสิ่งหนึ่ง ซึ่งอาจอยู่ในรูป**ของวัตถุทางกายภาพ (physical) หรือแนวคิด (conceptual) หรือแม้กระทั่งซอฟต์แวร์ (software)**

- Physical Entity

- Conceptual Entity

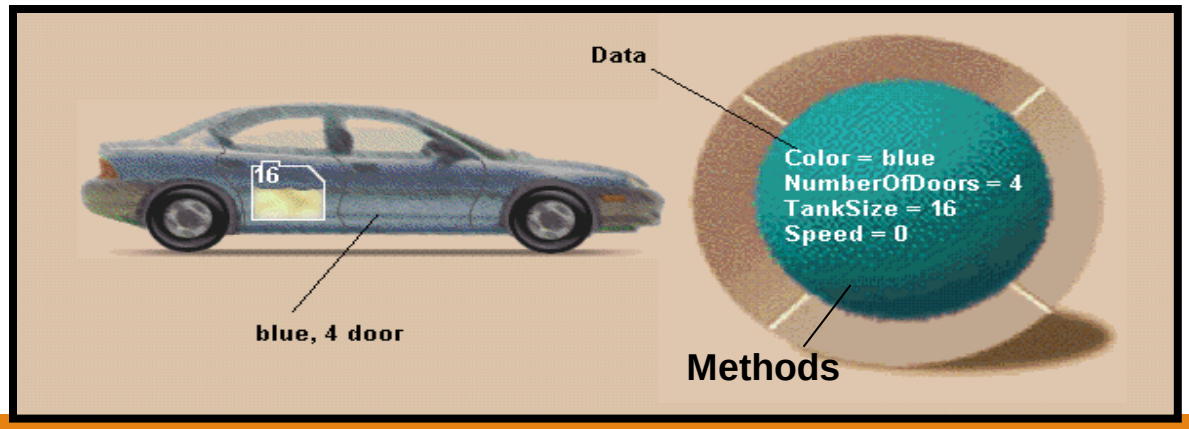
- Software Entity



Class

คลาสจะเป็นส่วนที่ใช้ในการกำหนดคุณสมบัติของออปเจคที่ใช้ข้อมูลและการทำงานร่วมกัน ทุก ๆ ออปเจคที่มีคุณสมบัติเหมือนกันจะถูกสร้างมาจากคลาสเดียวกัน

ในกระบวนการเชิงวัตถุ คลาสจะมีลักษณะเป็น **“พิมพ์เขียว”** ของออปเจค โดย คลาสคือนิยามที่มีลักษณะเป็น “นามธรรม” ส่วนออปเจคจะมีลักษณะเป็น “รูปธรรม” ที่ถูกสร้างขึ้นตามนิยามที่กำหนดไว้ใน “คลาส” นั่นเอง



Defining a Class

รูปแบบของการกำหนด Class

```
[modifier] class ClassName {
    // ClassBody => data + method()
}
```

- modifier เป็นคีย์เวิร์ดที่ใช้ในการกำหนดสิทธิในการใช้งานของ Class
- class เป็นคีย์เวิร์ดที่ใช้ในการระบุว่าเป็นการประกาศคลาส
- ClassName เป็นการระบุชื่อคลาส โดยปกติอักษรตัวแรกของคำจะถูกกำหนดให้เป็นตัวใหญ่เสมอ เช่น

```
public class Car {

}
```

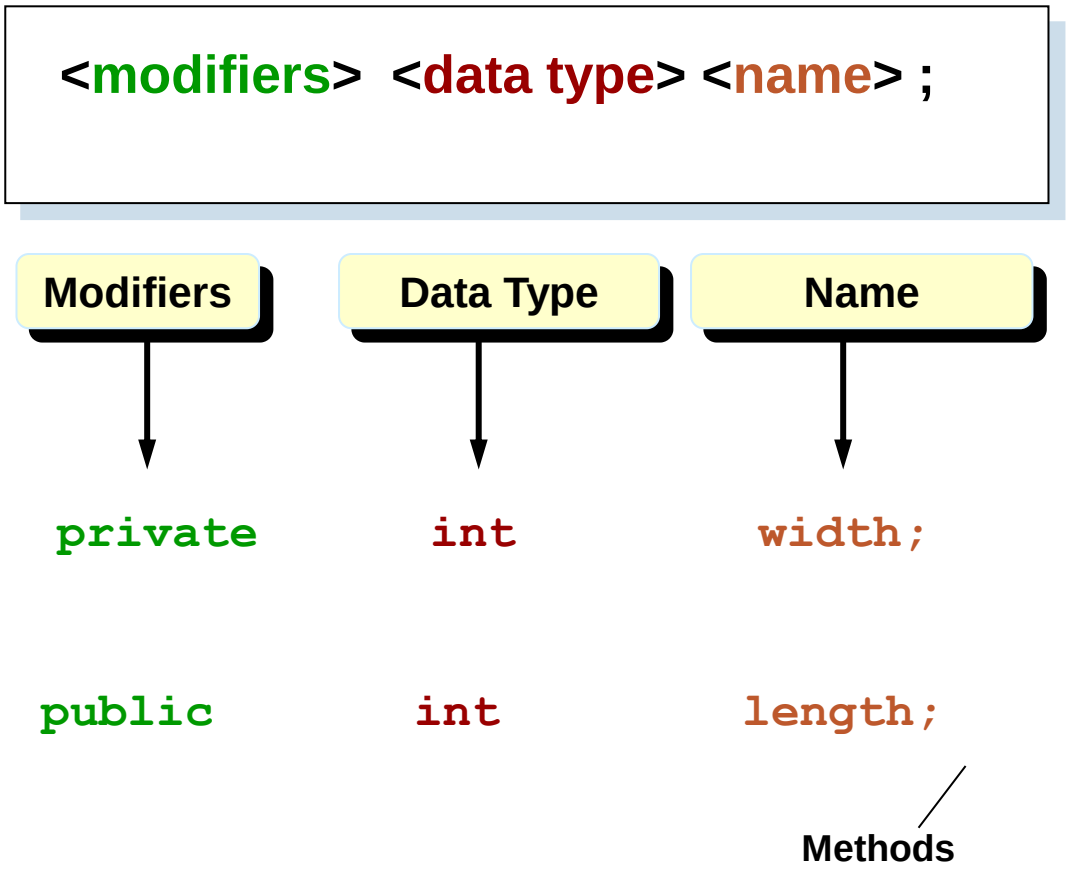
Modifiers of Classes

```
public class Car {

}
```

modifiers สำหรับ class จะถูกกำหนดไว้ดังรูปแบบต่อไปนี้ :

- public—**คลาสอาจถูกเรียกใช้ได้โดยตรงจากภายนอก package**
- abstract—เป็นการกำหนด class ที่ประกอบไปด้วย abstract เมธอดที่ไม่ได้มีการระบุรายละเอียดการทำงานไว้ แต่จะมีจุดประสงค์ในการใช้งานสำหรับการสืบทอดเท่านั้น
- final—คลาสที่ถูกกำหนดไว้ในลักษณะนี้จะไม่สามารถนำไปทำการสืบทอดต่อได้
- private—**จะยอมให้มีการเรียกใช้ได้เฉพาะจากภายใน class**
- protected—จะยอมให้มีการเรียกใช้ได้เฉพาะจากภายใน class และ Inherit class

Data Member Declaration



Defining an Attribute

แอททริบิวต์คือตัวแปรหรือค่าคงที่ซึ่งประกาศภายในออปเจคโดยมีรูปแบบดังต่อไปนี้

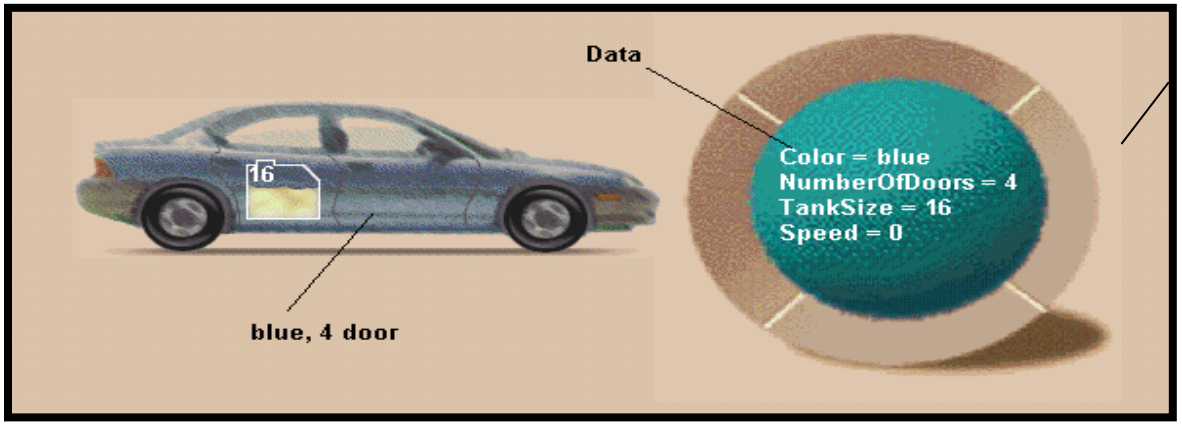
```
[modifier] dataType attributeName;
```

- modifier ใช้ในการกำหนดสิทธิในการใช้งานของตัวแปรหรือค่าคงที่
- dataType คือชนิดข้อมูลซึ่งอาจเป็นชนิดข้อมูลพื้นฐานหรือชนิดคลาส
- attributeName คือชื่อของแอททริบิวต์ เช่น

```
public class Car {
    public int numberOfWheel;
    public String color;
}
```

Defining an Attribute (TRY)

```
[modifier] dataType attributeName;
```



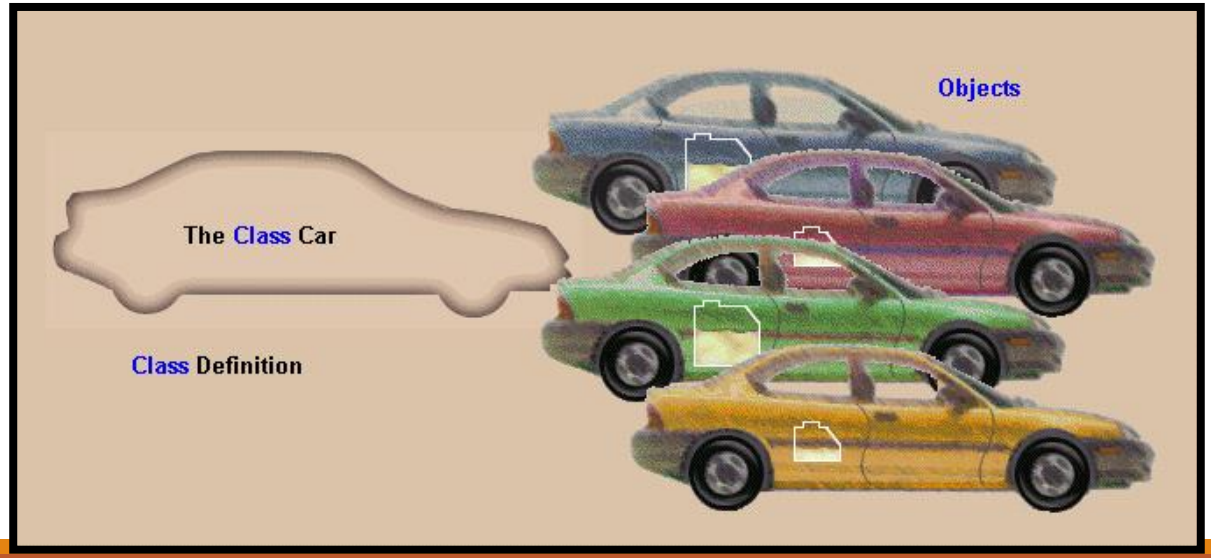

```
public class Car {
    public int numberOfWheel;
    public String color;

}
```

Instance

การสร้างออปเจคจะถูกเรียกว่า **instantiation** โดยออปเจคจะเป็น **instance** ของคลาส (New)

การสร้างออปเจคจากคลาสเดียวกันสามารถทำได้อย่างไม่จำกัด

ออปเจคที่ถูกสร้างขึ้นจะมีรายละเอียดตามคลาสที่ถูกกำหนดไว้



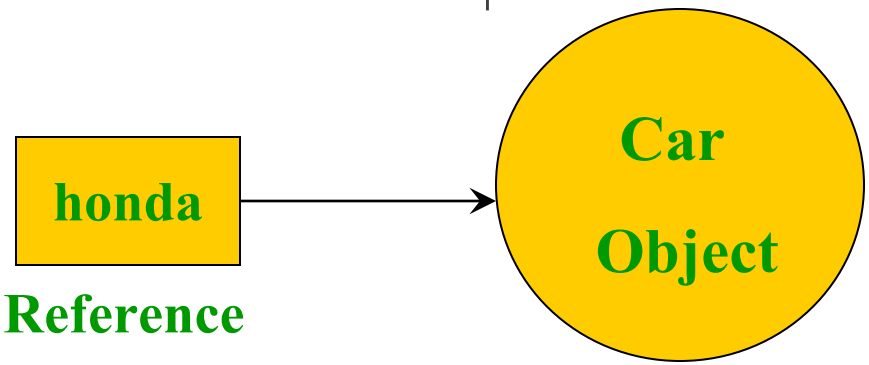
Creating a Class Instance

ในกรณีที่ไม่ได้มีการกำหนด Constructor ไว้ก่อน Java จะทำการสร้างให้โดยอัตโนมัติ

ใน Java ทุกครั้งที่มีการสร้างออปเจคใหม่เกิดขึ้นจะต้องอาศัยคำสั่ง new ระบุไว้ที่ Constructor ก่อนเสมอ :

```
Car honda = new Car()
```

ในกรณีนี้จะเป็นการกำหนด reference ที่ชื่อ p ที่ชี้ไปยังออปเจคที่แท้จริงภายในเมมโมรี่



Example: a Class

```
class Point    {
 public int x;
 public int y;
}
class Test   {
        public static void main(String[] args) {
          Point p = new Point();
          System.out.println(" Value of x = " + p.x + " Value of y = " + p.y);

   }
}
```

- การอ้างถึงค่าที่ถูกกำหนดไว้ภายในแอททริบิวต์จะทำได้โดยการระบุชื่อของออปเจคตามด้วยชื่อของแอททริบิวต์ที่ต้องการค้นด้วยเครื่องหมายจุด

Output : Value of x = 0 Value of y = 0

Try : New Class Car

Print ??
public int numberOfWheel;
public String color;

```
public class Car {
    public int numberOfWheel;
    public String color;

    public static void main(String[] args) {
        Car honda = new Car();
        System.out.println(honda.numberOfWheel);
        System.out.println(honda.color); }
}
```

Variable Declarations

Modifiers :

- public—ข้อมูลจะถูกเข้าถึงได้จากที่ใด ๆ ภายในโปรแกรม
- protected—ข้อมูลจะถูกเข้าถึงได้เฉพาะจาก subclass ที่อยู่ภายใน package เดียวกัน
- **private—ข้อมูลจะถูกเข้าถึงได้จาก code ภายใน class เดียวกัน**

final—ข้อมูลที่ถูกกำหนดในลักษณะนี้จะไม่สามารถเปลี่ยนแปลงได้

static— ในกรณีที่ต้องการพื้นที่ของข้อมูลภายในหน่วยความจำเพียงตำแหน่งเดียวเพื่อใช้สำหรับการเก็บข้อมูล โดยไม่คำนึงถึงจำนวนออปเจคที่ถูกสร้างขึ้น

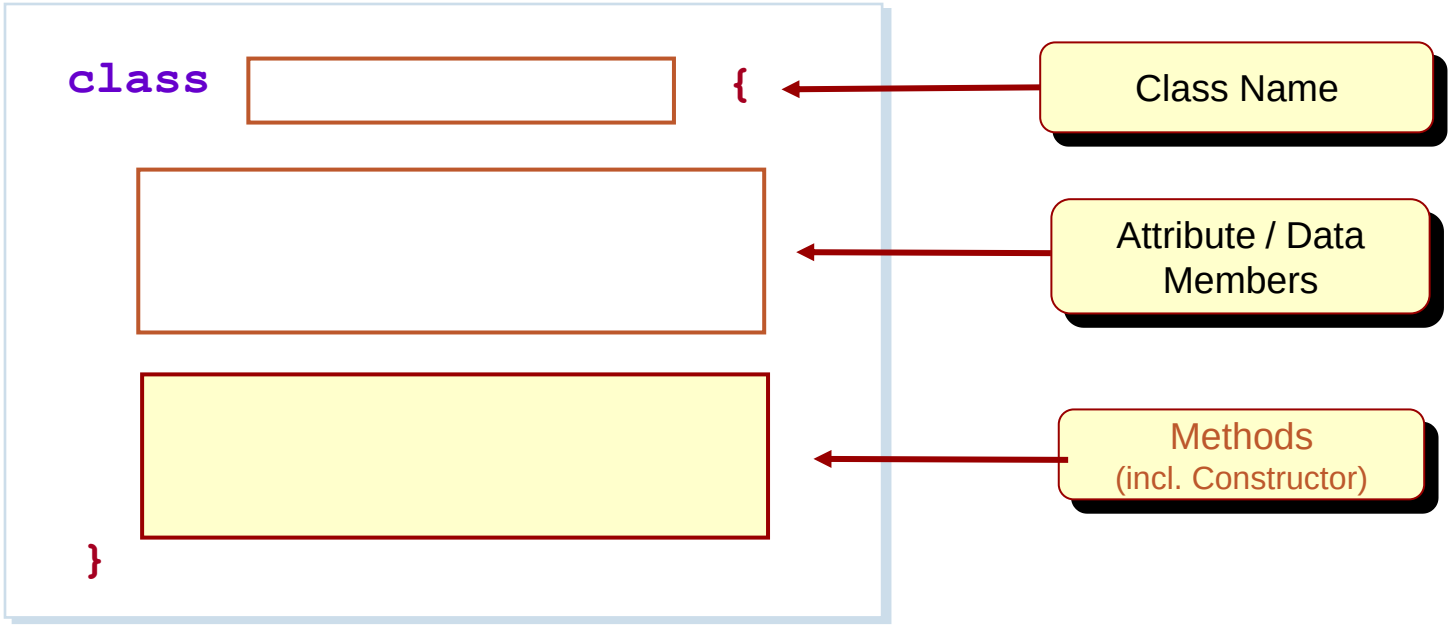
Defining a Private Attribute

```
class Point  // File Point.java
{
 private int x;
 private int y;
}
```

```
class Test  // File Test.java
{
 public static void main(String[] args) {
    Point p = new Point();

    System.out.println( " Value of x = " + p.x +  " Value of y = " + p.y);
 }
```

Output : Error

Class Definition : Review



Defining a Method

ภาษาจาวากำหนดรูปแบบของการประกาศเมธอดที่อยู่ในคลาสไว้ดังนี้

```
[modifier] return_type methodName ( [argument]) {
        // method_body
}
```

- modifier คือคีย์เวิร์ดที่ใช้ในการกำหนดสิทธิในการเรียกใช้เมธอด
- return_type เป็นส่วนที่ใช้กำหนดชนิดของข้อมูลที่ต้องการคืนค่า ภายหลังจากสิ้นสุดการทำงานภายในเมธอด
- methodName เป็นการระบุชื่อของเมธอด โดยปกติขึ้นต้นด้วยตัวเล็ก
- arguments ประกอบไปด้วยชื่อและชนิดข้อมูลที่ใช้ในการรับข้อมูลจากออปเจคที่เรียกใช้

Modifiers of Methods

ส่วนของ **modifiers** สำหรับเมธอดอาจกำหนดได้ดังนี้:

- **public—เมธอดจะถูกเรียกใช้ได้จากโค้ดใด ๆ**
- protected—เมธอดจะถูกเรียกใช้ได้จาก package เดียวกันหรือ subclass
- private—เมธอดสามารถจะถูกใช้ได้จากโค้ดที่อยู่ภายใน class เดียวกัน
- abstract—เป็นการกำหนดเฉพาะชื่อและพารามิเตอร์ของเมธอด โดยไม่มีการระบุ code ในการทำงานไว้
- static—เป็นเมธอดที่สามารถเรียกใช้ได้โดยตรง โดยไม่จำเป็นจะต้องทำการสร้างออปเจคไว้ก่อน
- final—เป็นการกำหนดเมธอดไม่ให้สามารถทำการ overriden ได้

Method Definitions

- ในกรณีที่มีการประกาศตัวแปรภายในเมธอด จะถือเป็นการทำงานภายในบล็อกโค้ด ซึ่งตัวแปรดังกล่าวไม่สามารถเรียกใช้ได้จากภายนอก
- เมธอดไม่สามารถประกาศไว้ภายในเมธอดอื่น ๆ ได้
- **ในกรณีที่มีการระบุชนิดของการคืนค่าไว้ บรรทัดสุดท้ายภายในเมธอดจะต้องระบุ (ไม่ใช่ void)**
  - return *expression*;
  - Expression อาจเป็นนิพจน์หรือค่าที่มีชนิดข้อมูลแบบเดียวกับที่ระบุไว้ใน return_type
- **หมายเหตุ** เมธอดในจาวาคืนค่าได้เพียงค่าเดียวเช่นเดียวกับภาษาซี

Type of Methods

```
<modifier>  <return type>  <method name>  ( <parameters> ){
    <statements>
}
```

นอกเหนือจาก constructor **เมธอดในการจัดการตัวแปรสามารถแบ่งออกได้เป็น 2 ชนิด ได้แก่:**

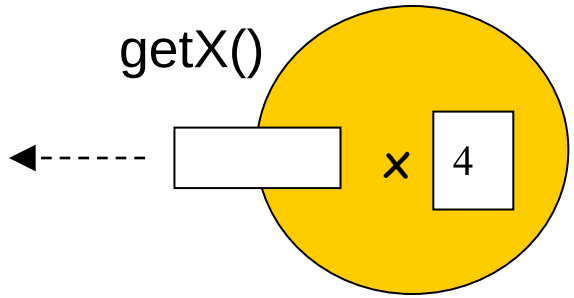
- เมธอดแบบ accessor (หรือที่เรียกว่าเมธอดแบบ get) เป็นเมธอดที่ใช้สำหรับการ**คืนค่าของออปเจคที่ถูกระบุ**
- เมธอดแบบ mutator (หรือที่เรียกว่าเมธอดแบบ set) เป็นเมธอดที่ใช้สำหรับการเซทค่าหรือเปลี่ยนแปลงค่าข้อมูลที่อยู่ภายในออปเจค

Accessor method

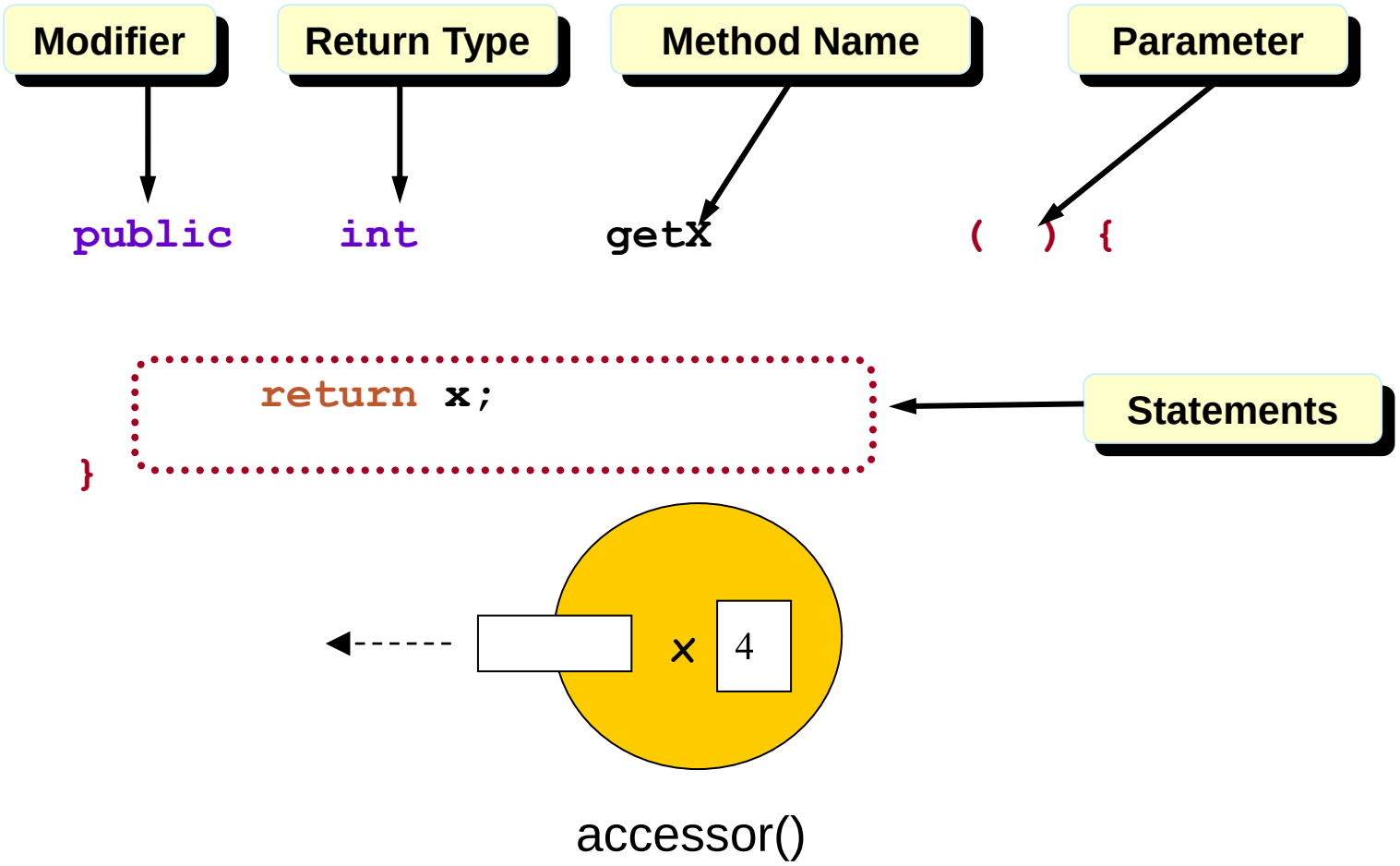
เมธอดแบบนี้ใช้สำหรับการเข้าถึงโดยการอ่านค่าข้อมูลหรือแอทริบิวต์ที่อยู่ภายในออปเจค โดยปกติจะมีชื่อขึ้นต้นด้วย get?() เสมอ ส่วนเครื่องหมาย ? ใช้แทนชื่อของข้อมูลหรือแอทริบิวต์ที่ถูกเรียกใช้

เมธอดแบบนี้จะคืนค่าเสมอ โดยปกติจะเป็นค่าชนิดเดียวกับค่าข้อมูลภายในออปเจค ดังนั้นจึงมีคีย์เวิร์ด return เสมอ

```
class Value
{
    private int x;
            public int getX()
            {
                    return(x);
            }
}
```



Method Declaration : Accessor



Try : Class Car -> Create Get Method

Print ??
public int numberOfWheel;
public String color;

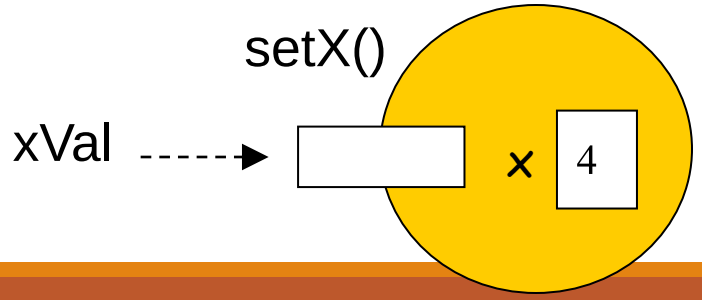
```
public class Car {
    private int numberOfWheel;
    private String color;

    public int getNumberOfWheel() {
        return numberOfWheel;
    }

    public String getColor() {
        return color;
    }
}
```

Mutator methods

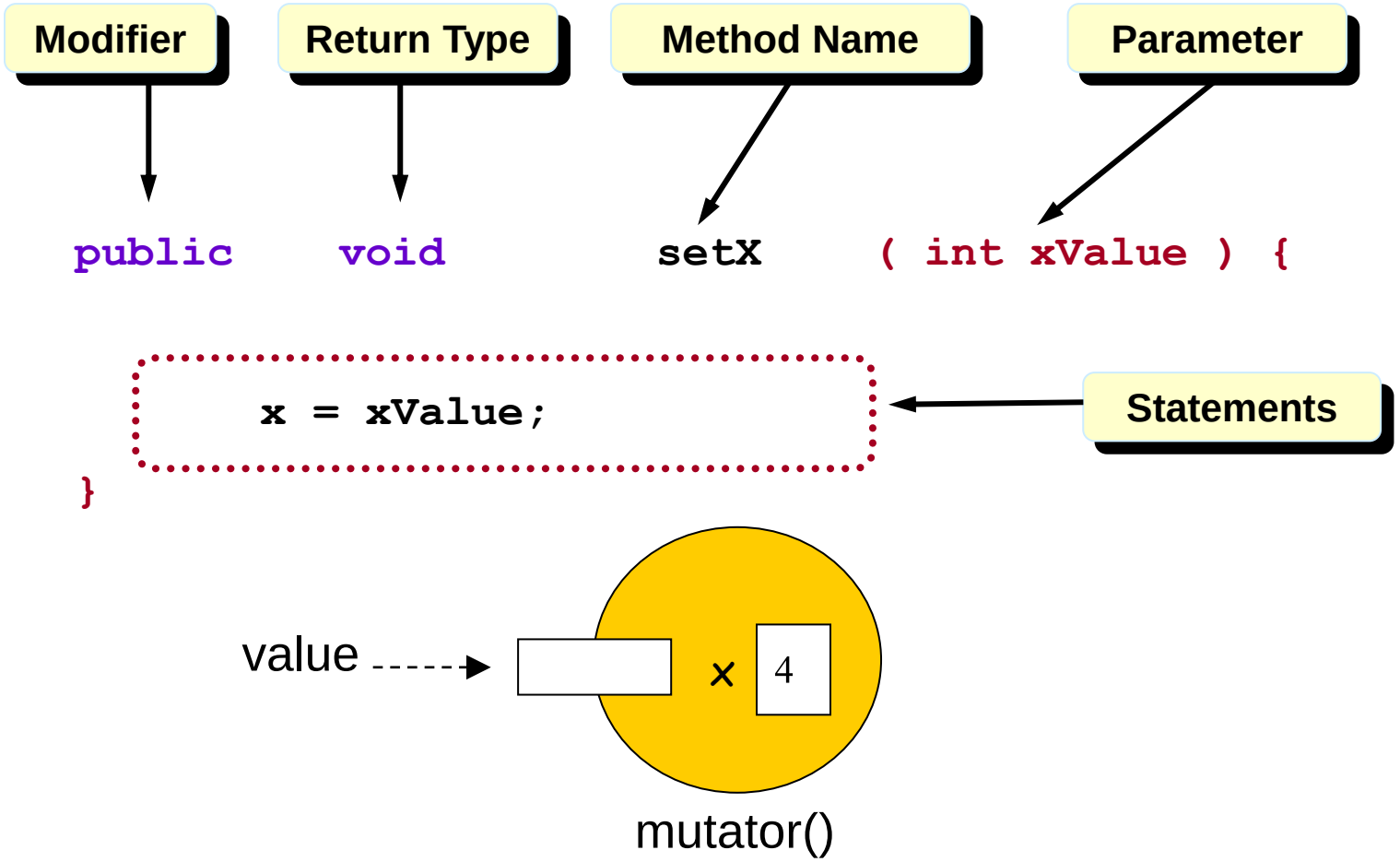
เมธอดแบบนี้ใช้สำหรับการแก้ไขค่าของข้อมูลหรือแอทริบิวต์ที่อยู่ภายในออปเจค โดยปกติจะมีรายการพารามิเตอร์ชนิดเดียวกับแอทริบิวต์ที่ต้องการแก้ไขค่า

เนื่องจากมีจุดประสงค์ในการแก้ไขค่า เมธอดแบบนี้จึงไม่มีการคืนค่าเสมอ return_type ถูกกำหนดให้เป็น void เสมอ

```
class Value
{
   private int x;
         public void setX(int xVal)
         {
             x = xVal;
         }
}
```



Method Declaration : Mutator

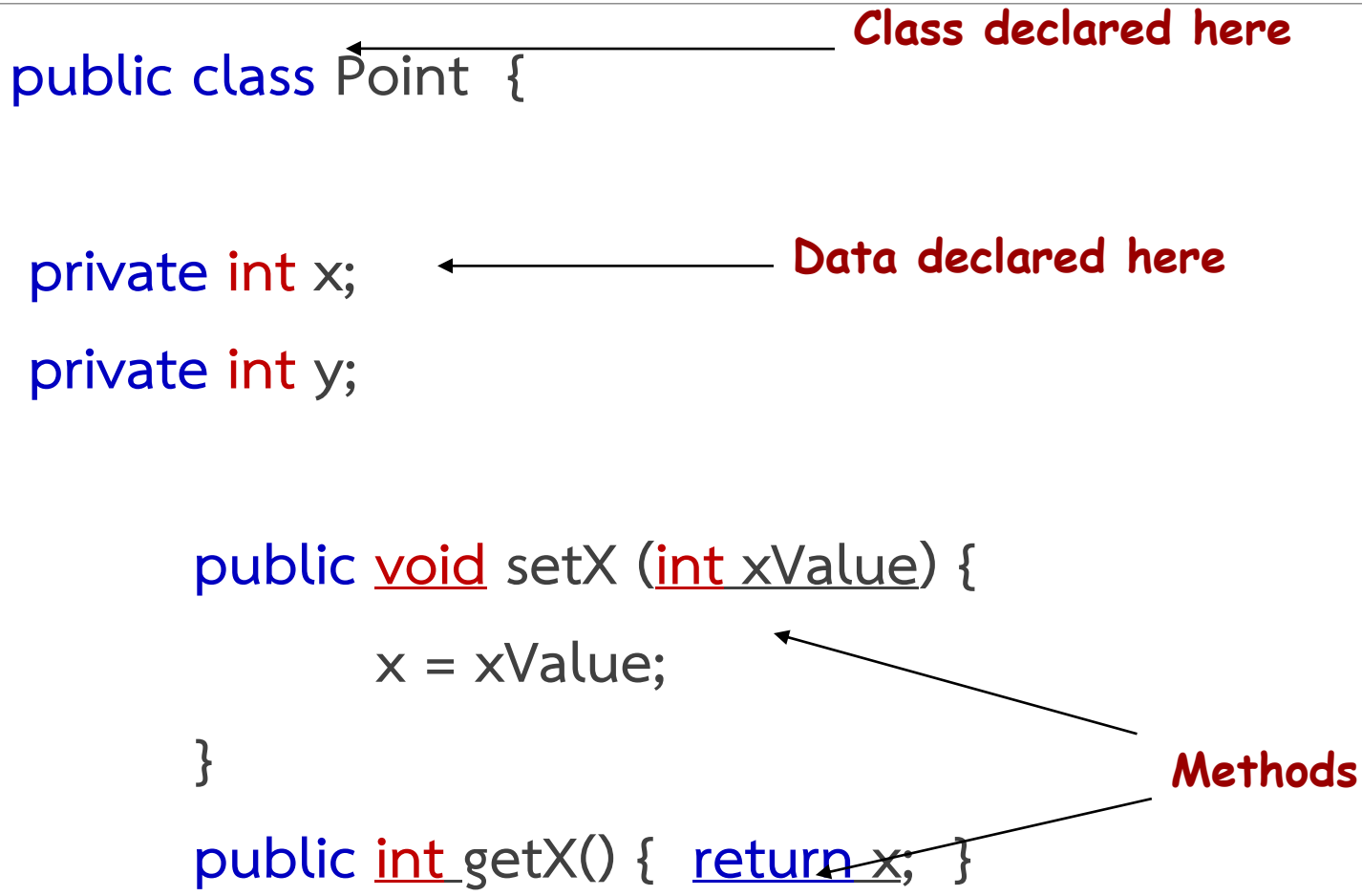


Try : Class Car -> Create Set Method

Print ??
public int numberOfWheel;
public String color;

```
public class Car {
    private int numberOfWheel;
    private String color;
    public void setNumberOfWheel(int numberOfWheel) {
        this.numberOfWheel = numberOfWheel;
    }

    public void setColor(String color) {
        this.color = color;
    }
}
```

Class with Accessor & Mutator Method



Class with Accessor & Mutator Method

```
class Point {
  private int x;
  private int y;
        public void setX(int xValue) {
                x = xValue;
        }
        public int getX() {  return x;  }

        public void setY(int yValue) {
                y = yValue;
        }
        public int getY() {  return y; }
}
```

Object Type declared here

Data declared here

Methods

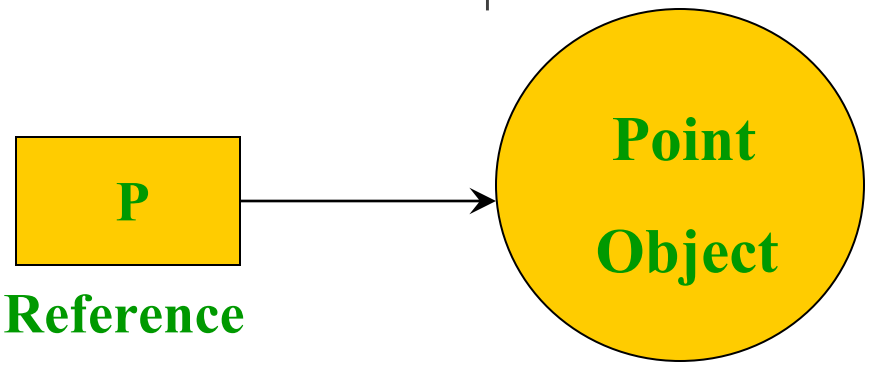
Creating a Class Instance

ในกรณีที่ไม่ได้มีการกำหนด Constructor ไว้ก่อน Java จะทำการสร้างให้โดยอัตโนมัติ

ใน Java ทุกครั้งที่มีการสร้างออปเจคใหม่เกิดขึ้นจะต้องอาศัยคำสั่ง new ระบุไว้ที่ Constructor ก่อนเสมอ :

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

ในกรณีนี้จะเป็นการกำหนด reference ที่ชื่อ p ที่ชี้ไปยังออปเจคที่แท้จริงภายในเมมโมรี่



Method calls

โดยปกติแล้วการเรียกใช้เมธอดจะประกอบไปด้วย 3 ส่วนที่สำคัญดังนี้ :

- ชื่อของออปเจคที่ต้องการเรียกใช้เมธอดที่ต้องการ
- ชื่อของเมธอดที่ต้องการเรียกใช้
- พารามิเตอร์ที่เมธอดต้องการ

การเรียกใช้เมธอดในโปรแกรมเชิงวัตถุจะถือเป็นการติดต่อกันระหว่างออปเจค ดังนั้นจึงจำเป็นต้องสร้างออปเจคด้วยคีย์เวิร์ด new ก่อนเสมอ

```
className name = new className();
```

การเรียกใช้เมธอดภายในออปเจคจะเหมือนกับการระบุค่าแอททริบิวต์ภายในออปเจค โดยมีรูปแบบดังนี้

```
name.methodName();
```

Using objects

```
Point p = new Point();

p.setX(5);

p.setY(5);
```

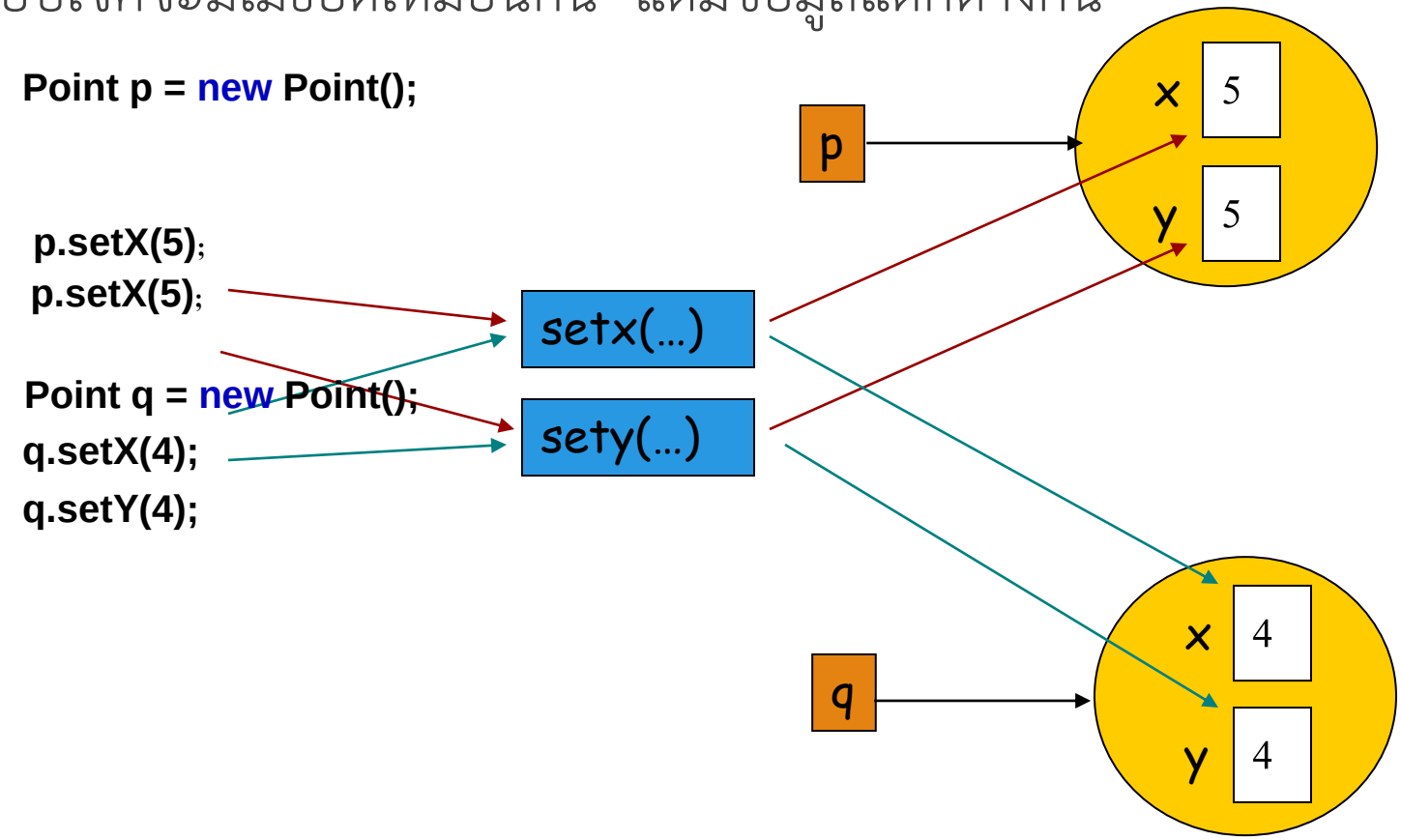
```
class Point {
  private int x;
  private int y;

  public void setX(int xValue)  {
    x = xValue;
  }
  public int getX() {
    return x;
  }
  public void setY(int yValue) {
    y = yValue;
  }
  public int getY() {  return y; }
}
```


```
class TestRun
{
        public static void main(String[] args) {

            Point p = new Point();

            p.setX(5);
            p.setY(5);
            System.out.println(" Value of x = "+ p.getX() +" Value of y = "+ p.getY());
  }
}
```

Output : Value of x = 5 Value of y = 5

Using objects

แต่ละออปเจคจะมีเมธอดเหมือนกัน แต่มีข้อมูลแตกต่างกัน



Try : Class Car

set color = Black

Set numberOfWheel = 4

```
public static void main(String[] args) {
    Car1 honda = new Car1();
    honda.setColor("Black");
    honda.setNumberOfWheel(4);
    System.out.println(honda.getNumberOfWheel());
    System.out.println(honda.getColor());
}
```

Methods

การสร้างเมธอดไม่ได้มีเพียงค่า Set และ get เท่านั้น

```
class Shape  {
    int x = 10;
    int  y = 10;
    public int calcalteSquare() {
        return x*y;
    }
}

Class Run {
     public static void main(String[] args) {
        Shape shape = new Shape();
        System.out.println(shape.calcalteSquare());
    }
}
```

Review Method How to : calculateTax()

ต้องการคำนวณราคา**สินค้ารวมภาษีจาก จำนวนสินค้า (Qty) และราคาสินค้า (price)** พร้อมทั้งภาษีจำนวน 7 % -> Invoice

1 รูปแบบของการเขียนเมธอด

```
modifier return_type  methodName( [argument] )
{
   // method body
}
```

2 ระบุชนิดข้อมูลที่ต้องการคืนค่า

ในกรณีนี้ต้องการคำนวณราคาสินค้ารวมภาษี ซึ่งมีแนวโน้มเป็นทศนิยม ดังนั้นกำหนดข้อมูลชนิด double พร้อมกำหนดคำสั่ง return ในเมธอด

```
public double () {
      // body
      return;
}
```

Description	Rate	Qty	Line Total
Freelance Writing Hourly rate for article on flight maneuvers	$100.00 +Tax	3.5	$350.00
Social Media Promotion of article on Twitter and Instagram	$500.00	1	$500.00
		1	

Review Method How to

3 กำหนดชื่อของเมธอด

ชื่อของเมธอดควรขึ้นต้นด้วยอักษรตัวเล็กและอักษรตัวใหญ่ในคำถัด ๆ ไป

```
public double calculateTax() {
    //body
return;
}
```

4 ระบุพารามิเตอร์ที่ต้องการ

เริ่มจาก ชนิดข้อมูล ชื่อตัวแปรคั่นด้วยเครื่องหมาย comma ในกรณีที่มีพารามิเตอร์หลายตัว ส่วน body ของเมธอดจะอยู่ภายในเครื่องหมายปีกกา

```
public double calculateTax (int qty, double price ) {
    //  body
return ;
}
```

Description	Rate	Qty	Line Total
Freelance Writing Hourly rate for article on flight maneuvers	$100.00 +Tax	3.5	$350.00
Social Media Promotion of article on Twitter and Instagram	$500.00	1	$500.00
		1	

Review Method How to

5 เขียนโค้ดภายในเมธอด

ในกรณีนี้ค่าราคารวมภาษีคำนวณได้จากจำนวนสินค่า (Qty) และราคาสินค้า (price) พร้อมทั้งภาษีจำนวน 7 % ซึ่งสามารถเขียนเป็นโค้ดสำหรับการคำนวณได้ดังต่อไปนี้

```
public  double  calculateTax( int qty, double price )
 {
   double subtotal;

                    subtotal = price * qty;

   return (subtotal* 0.07);
}
```

Constructor

คอนสตรัคเตอร์เป็นเมธอดชนิด**พิเศษ**ที่ใช้สำหรับการกำหนดค่าเริ่มต้นของออปเจค โดยปกติจะใช้ร่วมกับคีย์เวิร์ดตัวกระทำ (Operator) ที่เรียกว่า new เมื่อมีการสร้างออปเจคใหม่เกิดขึ้น

การประกาศคอนสตรัคเตอร์จะแตกต่างไปจากเมธอดอื่น ๆ โดยมีรูปแบบดังนี้:

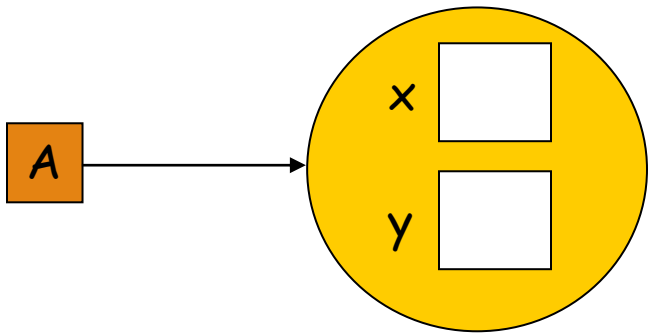
```
AccessSpecifier ConstructorName (Parameter lists)
{
        // ConstructorBody
}
```

Constructors

- ชื่อของ Constructor จะเป็นชื่อเดียวกับคลาส และอาจรับค่าพารามิเตอร์ได้
- เนื่องจากคอนสรัคเตอร์ทำหน้าที่เป็นกลไกในการกำหนดค่าเริ่มต้นของออปเจคที่ถูกเรียกใช้โดยอัตโนมัติทุกครั้งที่ออปเจคถูกสร้างขึ้นจากคลาส
- การกำหนดคอนสรัคเตอร์จะไม่มีการกำหนดชนิดของการคืนค่า (return) ใด ๆ ไว้
- โดยปกติคอนสรัคเตอร์จะเป็น **public**
- หากการประกาศคลาสไม่ได้มีการกำหนดคอนสตรักเตอร์ใด ๆ ไว้ จาวาจะทำการสร้าง **default constructor ให้โดยอัตโนมัติ**
- นอกจากนั้นแล้วคอนสตรักเตอร์ที่ไม่ได้มีการกำหนดพารามิเตอร์ไว้จะถือเป็น default constructor

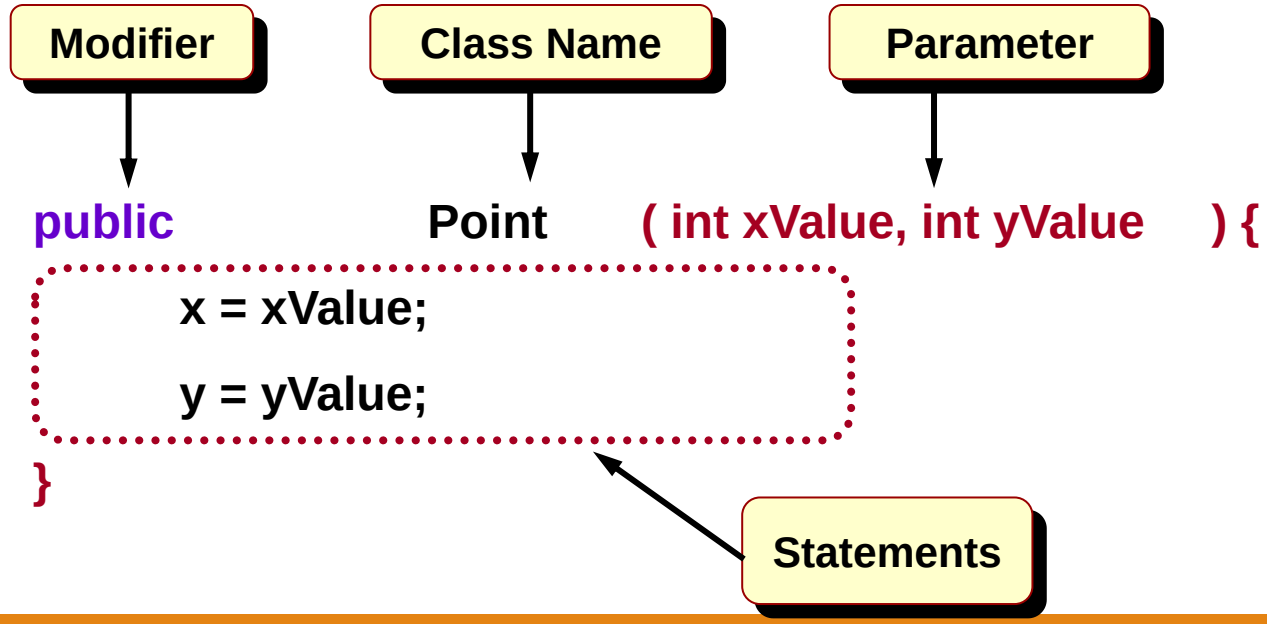
Constructor :Initialized Object

```
public class Point {

    private int x = 0;

    private int y = 0;

    // a constructor!

    public Point() {     }

Point A = new Point();
```



Constructor (1/2)

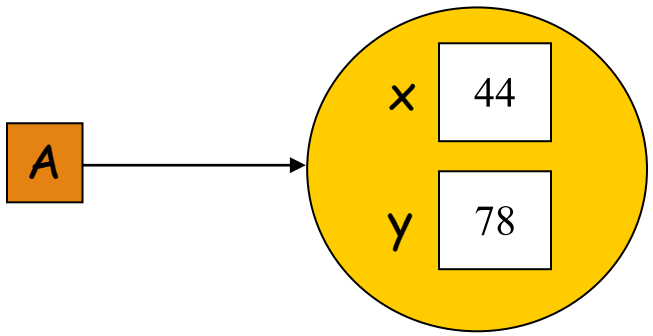
```
public <class name> ( <parameters> )     {
    <statements>
}
```



Constructor :Initialized Object

```
public class Point {


  private int x = 0;
  private int y = 0;
  // a constructor!
  public Point() {}
  public Point(int xValue, int yValue) {
            x = xValue;
            y = yValue;
  }
}
```

```
Point A = new Point(44,78);
```



Class Point with Constructor

```
class Point {
        private int x;
        private int y;
        public Point(int x, int y) {
                        this.x = x;
                        this.y = y;
        }
        public void setX(int xValue) { x = xValue; }
        public int getX() {  return x;  }
        public void setY(int yValue) { y = yValue; }
        public int getY() {  return y; }
}
```

Class Point with Constructor

```
class Test {
        public static void main(String[] args) {
        Point a = new Point(44, 78);
         System.out.println(" Value of p.x = " + a.getX()+ " Value of p.y = " +
         a.getY());

         Point q = new Point(0,0);
        q.setX(5);
        q.setY(5);
         System.out.println(" Value of q.x = " + q.getX() + " Value of q.y = " +
        q.getY());
         }
}
```

Output : **Value of a.x = 44 Value of a.y = 78**
Value of q.x = 5 Value of q.y = 5

Overloaded

เกิดขึ้นเมื่อสองเมธอดหรือมากกว่ามีชื่อที่เหมือนกันและถูกประกาศไว้ภายในคลาสเดียวกัน โดยขึ้นอยู่กับความต้องการของผู้ใช้

โดยปกติแล้วการ overloading เมธอดจะแตกต่างกันเฉพาะในส่วนของรายละเอียดของพารามิเตอร์เท่านั้น

การทำงานของเมธอดที่ถูก overloaded ในลักษณะนี้จะเกิดขึ้นในช่วงระหว่างการประมวลผลเท่านั้น โดยที่จำนวนและชนิดของพารามิเตอร์ภายในเมธอดจะถูกเรียกใช้ในช่วงเวลาที่กำหนดไว้ช่วงใดช่วงหนึ่งเสมอ

คอมไพเลอร์จะทำหน้าที่ในการแยกความแตกต่างของเมธอดจากจำนวนของพารามิเตอร์ที่ถูกระบุนั่นเอง เมื่อคอมไพเลอร์ทำการแปลรหัสโค้ดส่วนนี้ ก็จะสามารถจำแนกได้ว่าคอนสตรักเตอร์ใดที่กำลังถูกเรียกใช้ในช่วงเวลาการทำงาน

Overloaded Constructor

```
public class Point {
        private int x;
        private int y;
        public Point() {
                x = 0;  y = 0;
        }
        public Point(int xValue, int yValue) {
                x = xValue; y = yValue;
        }
        public Point(Point p) {
                x = p.getx(); y = p.gety();
        }
}
```

Overloaded Constructor

public Point() คอนสตรักเตอร์แบบแรกจะไม่มีการระบุพารามิเตอร์ไว้ จึงใช้สำหรับการกำหนดค่าเริ่มต้นของ x และ y ให้เป็น 0

public Point(int xValue, int yValue) คอนสตรักเตอร์แบบที่สองจะประกอบไปด้วยพารามิเตอร์ 2 จำนวน จึงใช้สำหรับกำหนดค่าของตัวแปร x และ y พร้อมกับการสร้างออปเจคได้ในคราวเดียว

public Point(Point p) คอนสตรักเตอร์แบบที่สามจะรับค่าพารามิเตอร์ในรูปของออปเจคจากคลาสเดียวกัน และทำการกำหนดค่าตัวแปร x และ y จากค่าของออปเจค p เป็นหลัก ในกรณีนี้จะเป็นตัวอย่างการทำงานที่เรียกว่า *copy constructor* เนื่องจากใช้สำหรับการสร้างออปเจคใหม่จากคลาส point ที่ทำการสำเนาค่าจากออปเจคที่ผ่านค่ามาในรูปของพารามิเตอร์นั่นเอง

Overloaded Constructor with Point class

```
class Point  {
        private int x;  private int y;
        public Point() {  x = 0;   y = 0; }
        public Point( int xValue, int yValue) { x = xValue; y = yValue; }
        public Point(Point p) { x = p.x; y = p.y; }
        public int getX() { return x; }
        public int getY() { return y;}
}
```

Overloaded Constructor with Point class

```
class Test
{
    public static void main(String[] args) {
            Point p = new Point();
            System.out.println(" Value of p.x = " + p.getX() +" Value of p.y = " + p.getY());
            Point q = new Point(5,5);
            System.out.println(" Value of q.x = " + q.getX() + " Value of q.y = " + q.getY());

            Point r = new Point(q);

            System.out.println(" Value of r.x = " + r.getX() + " Value of r.y = " + r.getY());

    }

}
```

Output :

Value of p.x = 0 Value of p.y = 0
Value of q.x = 5 Value of q.y = 5
Value of r.x = 5 Value of r.y = 5

Overload สามารถใช้กับเมธอดทั่วไปได้