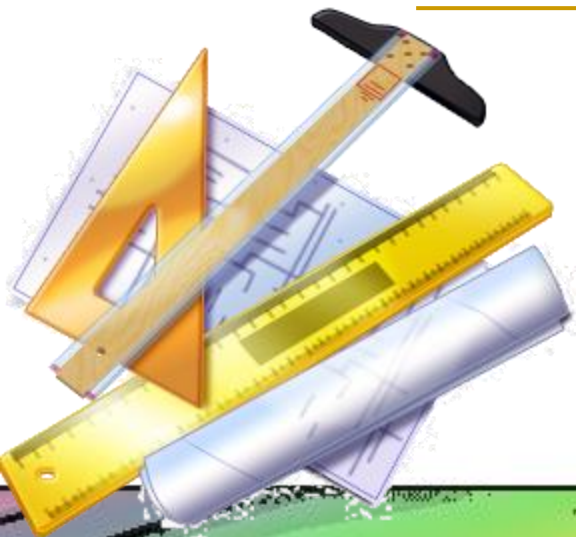


เพิ่มเติมเกี่ยวกับ Methods

Lecture 5

เยาวดี เต็มธนาภักดิ์

Yaowadee Temtanapat



วัตถุประสงค์ของการเรียนวันนี้

- เข้าใจเกี่ยวกับการส่งและคืน parameters ผ่านและจาก method
- เข้าใจความแตกต่างระหว่าง instance method กับ static method
- เข้าใจจุดประสงค์และการใช้ constructors
- เข้าใจแนวคิดเกี่ยวกับ static variables
- เข้าใจเกี่ยวกับขอบเขตของตัวแปร
- เข้าใจการเขียนโปรแกรมแบบ recursive method
- การบรรจุและใช้ class ใน package
- เพื่อเรียนรู้การแปลง class diagram ให้เป็นโค้ด

การ implement Class อย่างง่าย

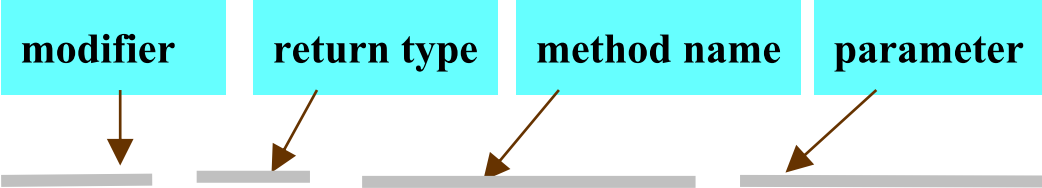
```
// Comment  
  
import statement  
  
public class ClassName  
{  
    instance variables  
    Constructors  
    Methods  
}
```

ลักษณะ

ความสามารถ

การกำหนด method

```
<modifiers> <return type> <method name> (<parameter list>)  
{  
    <statement list>  
}
```



```
public void warpToBottom(int posX, int posY) {  
    ...  
}  
  
public int getPositionX() {  
    ...  
    return positionX;  
}
```

Static และ Instance methods

- **Instance** หรือ **Object method**: วิธีการในระดับวัตถุ มีอยู่ในทุกวัตถุ
 - ตัวอย่างเมท็อดของ slide ก่อนหน้า
- **Static** หรือ **Class method**: วิธีการในระดับของคลาส มีอยู่เพียง 1 เดียวสำหรับ class นั้น ๆ

- **ตัวอย่าง** static methods ใน Math class

```
public static double abs(double a)  
public static double sqrt(double a)
```

- ทำให้สามารถเรียกใช้เมท็อดผ่านคลาสได้โดยตรง (หรือจะใช้ผ่านวัตถุก็ได้)

```
Math.sqrt(4);
```

Method Parameters

■ ใน method จะมี parameters 2 อย่าง

- ❑ parameter โดยนัย: **this**

- ❑ parameter ที่รับเข้ามาจากภายนอก

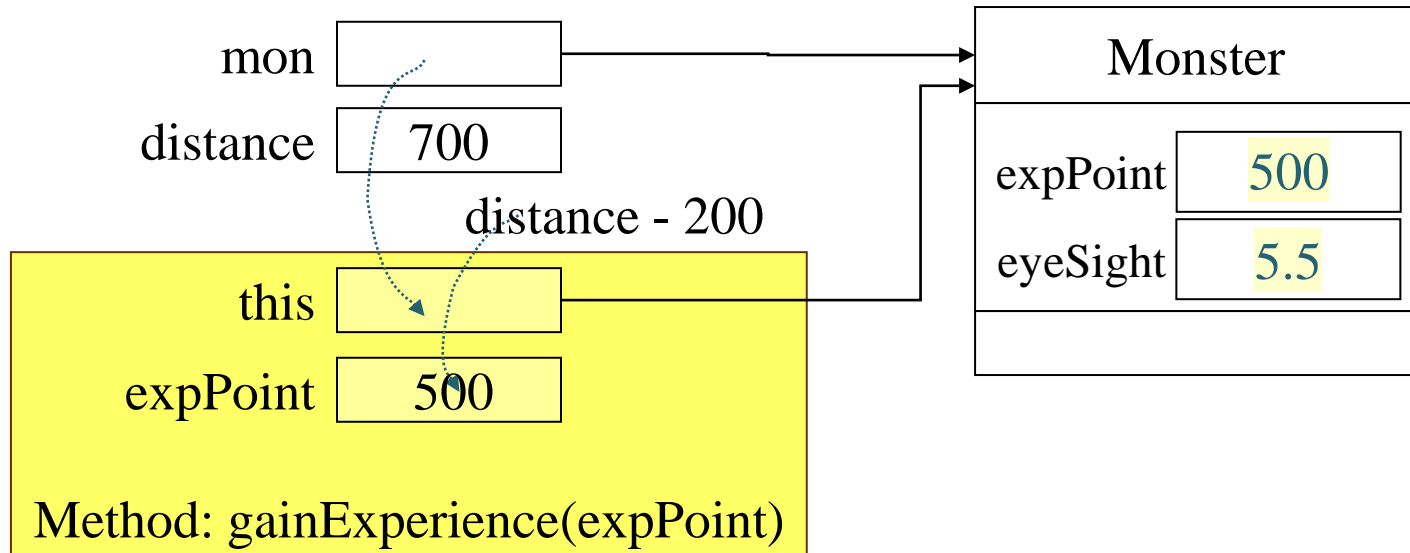
```
public class Monster {  
    . . . . .  
    public void gainExperience(long expPoint) {  
        this.expPoint = expPoint;  
        eyeSight = BASE_EYE_SIGHT + expPoint/1000.0;  
    }  
    . . . . .  
}
```

■ การเรียกใช้

```
mon.gainExperience(expPoint);
```

การส่ง Parameter

```
public class Monster {  
    private final double BASE_EYE_SIGHT = 5.0;  
    private long expPoint;  
    public void gainExperience(long expPoint) {  
        this.expPoint = expPoint;  
        eyeSight = BASE_EYE_SIGHT + expPoint/1000.0;  
    }  
    public static void main(String args[]) {  
        :  
        mon.gainExperience(distance - 200);  
    }  
}
```



การเรียกเมทอด

■ การเรียกเมทอด อาจเป็น 2 แบบกว้าง ๆ

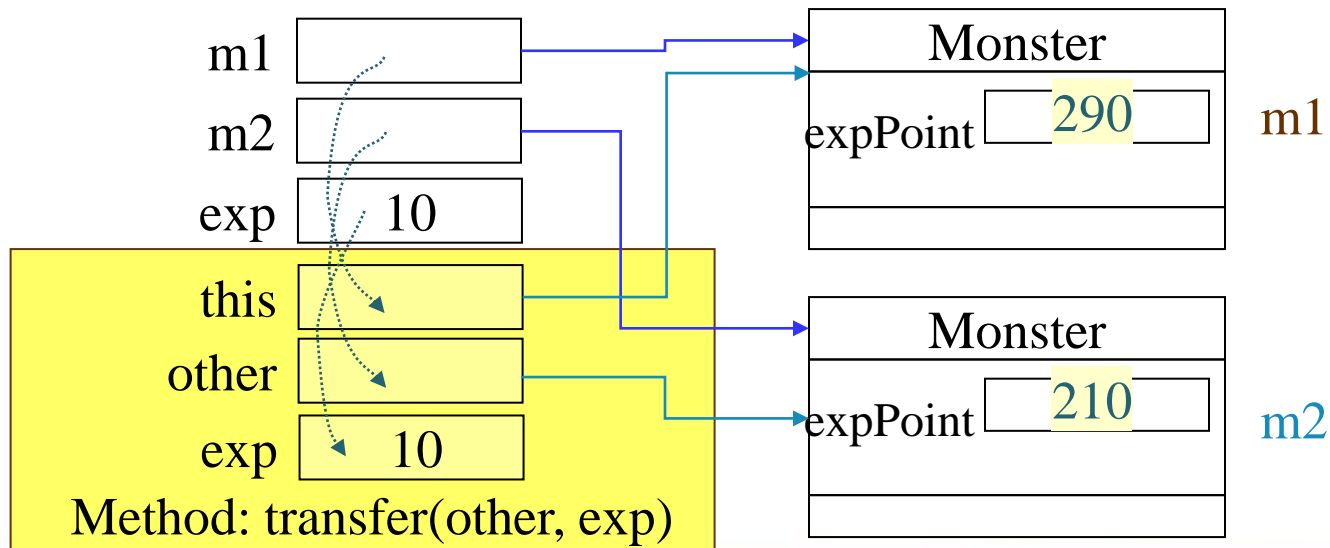
- **การเรียกด้วยค่า (Call by value):** สำเนาค่าจากตัวเรียกไปให้ตัวแปรพารามิเตอร์ของเมทอด เมื่อเมทอดเริ่มทำงาน
- **การเรียกด้วยตัวอ้างอิง (Call by reference):** เมทอดแก้ไขพารามิเตอร์ได้

■ จาวาทำ **Call by value** แบบเดียว

- เมทอดสามารถเปลี่ยน**สถานะ**ของวัตถุ แต่**ไม่**สามารถเปลี่ยนตัววัตถุใหม่ได้

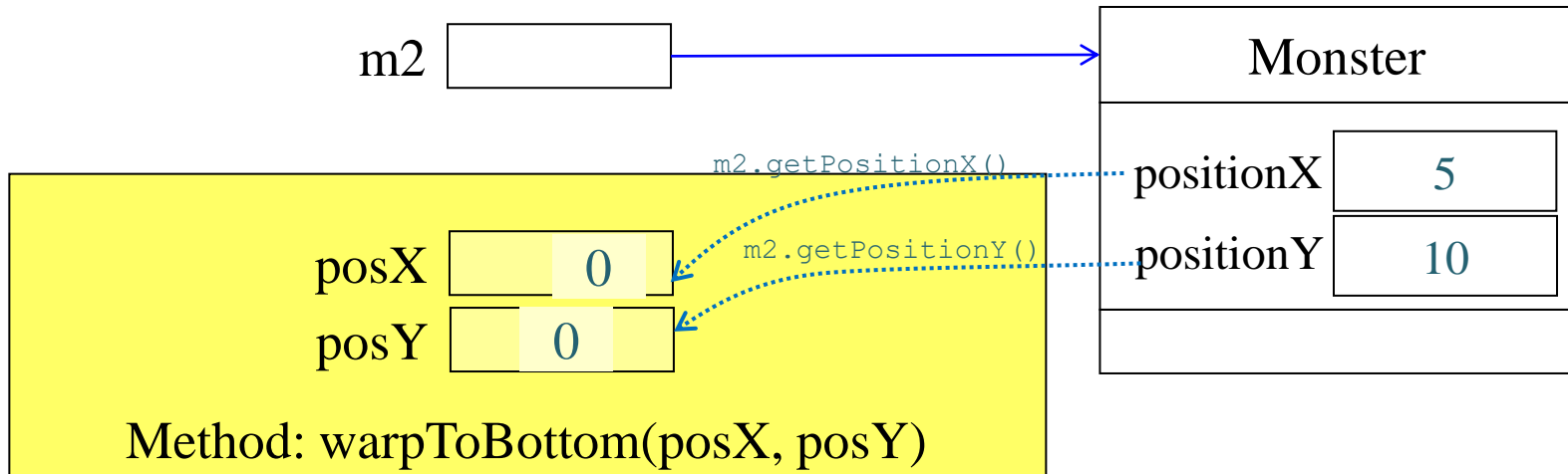
ตัวแปรที่อ้างถึงค่า VS ตัวแปรที่อ้างถึงวัตถุ

```
public class Monster {  
    private long expPoint;  
    public void transfer(Monster other, long exp) {  
        expPoint -= exp;  
        other.gainExperience(other.getExpPoint() + exp);  
    }  
:  
    m1.transfer(m2, 10);  
}  
}
```



Method ไม่สามารถเปลี่ยนค่าชนิดพื้นฐาน

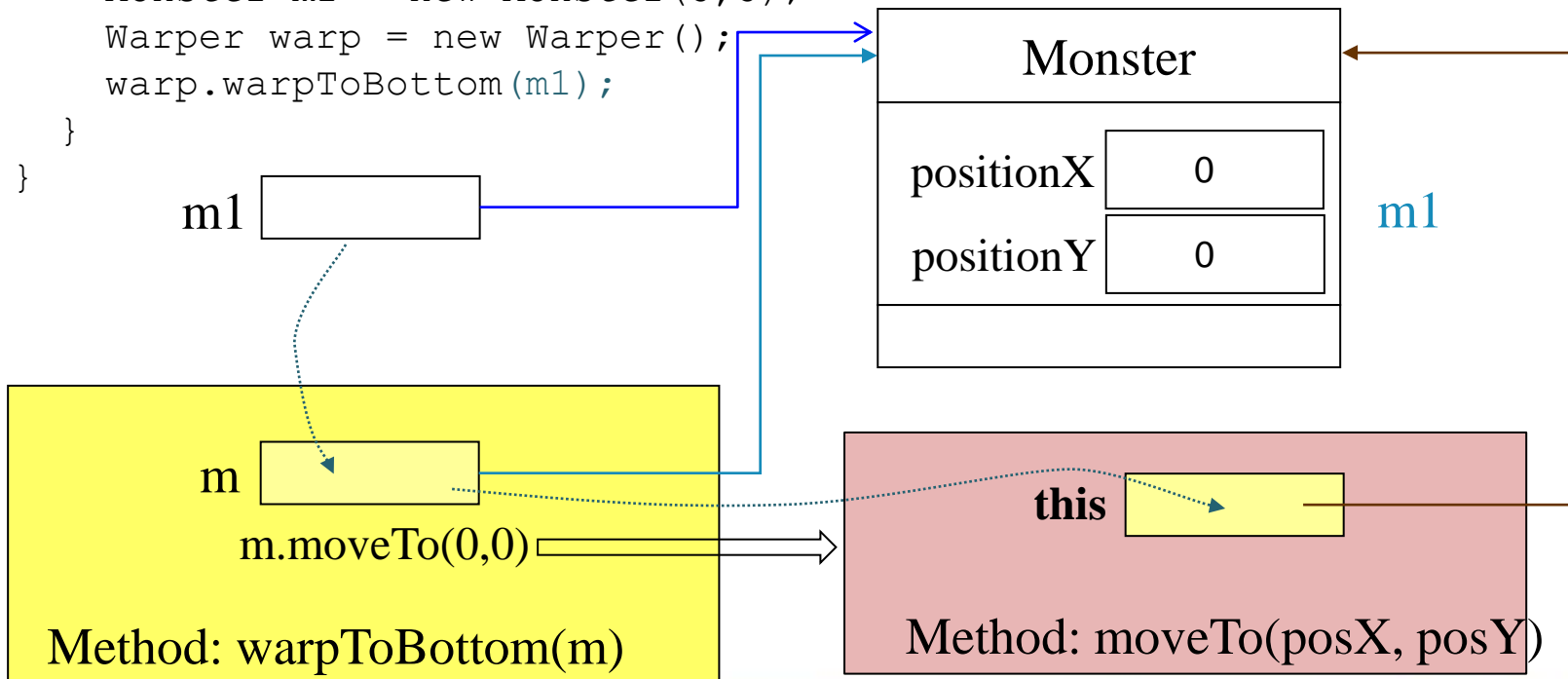
```
public class Warper {  
    public void warpToBottom(int posX, int posY)  
    {  
        posX = 0;  
        posY = 0;  
    }  
    public static void main(String args[]) {  
        . . .  
        Monster m2 = new Monster(5, 10);  
        Warper warp = new Warper();  
        warp.warpToBottom(m2.getPositionX(), m2.getPositionY());  
    }  
}
```



Method สามารถเปลี่ยนสถานะของวัตถุ

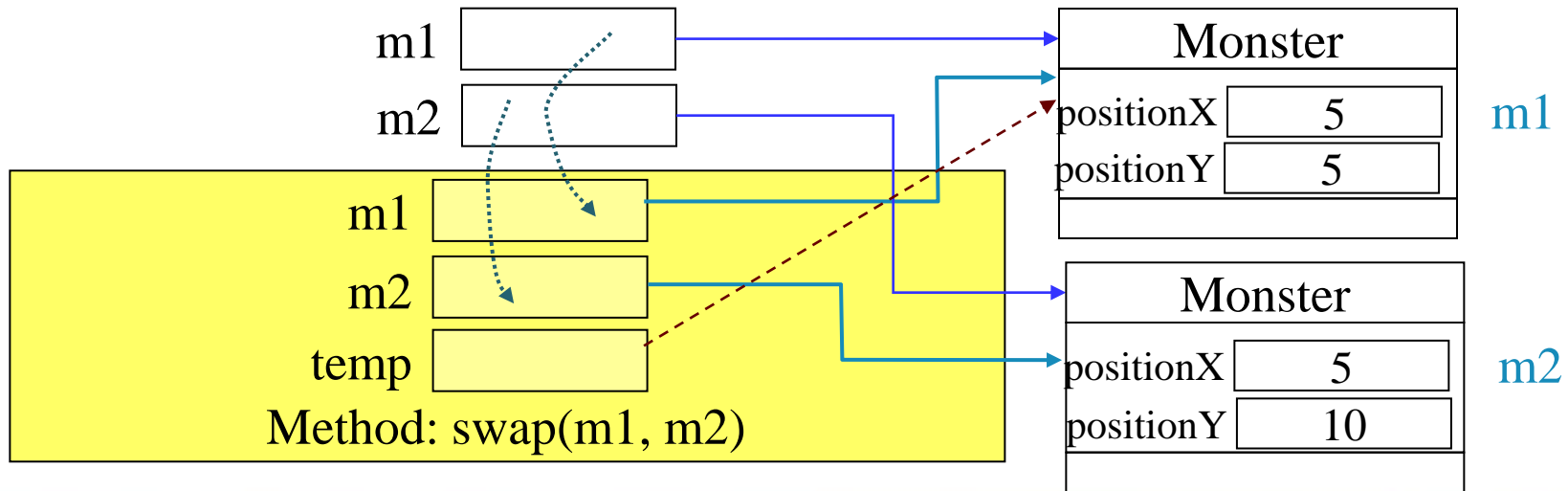
```
public class Warper {  
    public void warpToBottom(Monster m)  
    {  
        m.moveTo(0,0);  
    }  
  
    public static void main(String args[]) {  
        Monster m1 = new Monster(5,5);  
        Warper warp = new Warper();  
        warp.warpToBottom(m1);  
    }  
}
```

```
public void moveTo(int posX, int posY)  
{  
    positionX = posX;  
    positionY = posY;  
}
```



Method ไม่สามารถเปลี่ยนตัววัตถุ

```
public class Warper {  
    . . .  
    public void swap(Monster m1, Monster m2) {  
        Monster temp = m1;  
        m1 = m2;  
        m2 = temp;  
    }  
    public static void main(String args[]) {  
        Monster m1 = new Monster(5,5);  
        Monster m2 = new Monster(5,10);  
        swap(m1, m2);  
        :  
    }  
}
```



Accessor และ Mutator methods

- ประเภทเมทอดที่พบบ่อยในวัตถุ (เพื่อรองรับแนวคิดเกี่ยวกับ Information Hiding)
 - **Accessor** หรือ **Getter method**: วิธีการที่ใช้ในการคืนค่าให้กับผู้ใช้โดยไม่เปลี่ยนแปลงค่านั้น ๆ
 - **Mutator** หรือ **Setter method**: วิธีการที่ใช้ในการเปลี่ยนค่าข้อมูลตามค่าที่กำหนดให้โดยผู้ใช้

Overloading Method

- *ลายเซ็นของเมทอด (Method signature):* ส่วนเชื่อมต่อสำหรับการส่งสาร กำหนดจากชื่อของเมทอด และกลุ่มพารามิเตอร์ของเมทอด

```
double getDistanceFrom(int positionX, int positionY)
```

- *Method Overloading* เป็นการประกาศเมทอดที่ใช้ชื่อเดียวกันแต่มีลายเซ็นที่แตกต่างกัน เพื่อช่วยการโปรแกรมให้สะดวกและง่ายต่อการนำไปใช้

- **Overloading** ไม่คำนึงถึง return type ของลายเซ็นของเมทอด

```
public double getDistanceFrom(int positionX, int positionY)  
public double getDistanceFrom(Monster m)
```

Monster

Monster

- BASE_EYE_SIGHT: double
- isSleep: boolean
- positionX: int
- positionY: int
- eyeSight: double
- expPoint: long

+getDistanceFrom(int, int): double
+getDistanceFrom(Monster): double

Overloaded Methods

```
public class Monster {  
    private final double BASE_EYE_SIGHT = 5.0;  
    private Boolean isSleep;  
    private int positionX;  
    private int positionY;  
    private double eyeSight;  
    private long expPoint;  
    public double getDistanceFrom(int posX, int posY){  
        int xDiff = posX - getPositionX();  
        int yDiff = posY - getPositionY();  
        double distance = Math.sqrt(Math.pow(xDiff,2)+Math.pow(yDiff,2));  
        return Math.abs(distance);  
    }  
    public double getDistanceFrom(Monster m){  
        return getDistanceFrom(m.getPositionX(), m.getpositionY());  
    }  
}
```

การใช้ Overloaded

Execute Code Monster

```
public double getDistanceFrom(int posX, int posY){  
    int xDiff = posX - getPositionX();  
    int yDiff = posY - getPositionY();  
    double distance =  
        Math.sqrt(Math.pow(xDiff,2)+Math.pow(yDiff,2));  
    return Math.abs(distance);  
}  
public double getDistanceFrom(Monster m){  
    return getDistanceFrom(m.getPositionX(),  
        m.getpositionY());  
}
```

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

```
    Monster mon1 = new Monster();
```

```
    mon1.moveTo(0,0);
```

```
    Monster mon2 = new Monster();
```

```
    mon1.moveTo(0,5);
```

```
    System.out.printf("Monster 1 is at x:%d,y:%d %n",mon1.getPositionX(),mon1.getPositionY());
```

```
    System.out.printf("Monster 2 is at x:%d,y:%d %n",mon1.getPositionX(),mon1.getPositionY());
```

```
    System.out.println("Distance from mon1 to 9,0 : " + mon1.getDistanceFrom(9, 0));
```

```
    System.out.println("Distance from mon1 to mon2 : " + mon1.getDistanceFrom(mon2));
```

เรียกเมทอด

พารามิเตอร์ (Parameter) ชนิดวัตถุ

- เมท็อดสามารถรับเข้าพารามิเตอร์ที่เป็นวัตถุได้
- **พารามิเตอร์ชนิดวัตถุ:** ตัวแปรอ้างอิงถึงวัตถุ (object reference)
 - สามารถใช้ความสามารถของวัตถุนั้นในการทำงาน และ
 - อาจเปลี่ยนค่าภายในวัตถุได้ แต่เปลี่ยนตัววัตถุไม่ได้

```
public static void main(String[] args) {  
    Monster mon1 = new Monster(2,2);  
    Monster mon2 = new Monster(5,5);  
    mon2.call(mon1);  
}
```

เรียกเมท็อด

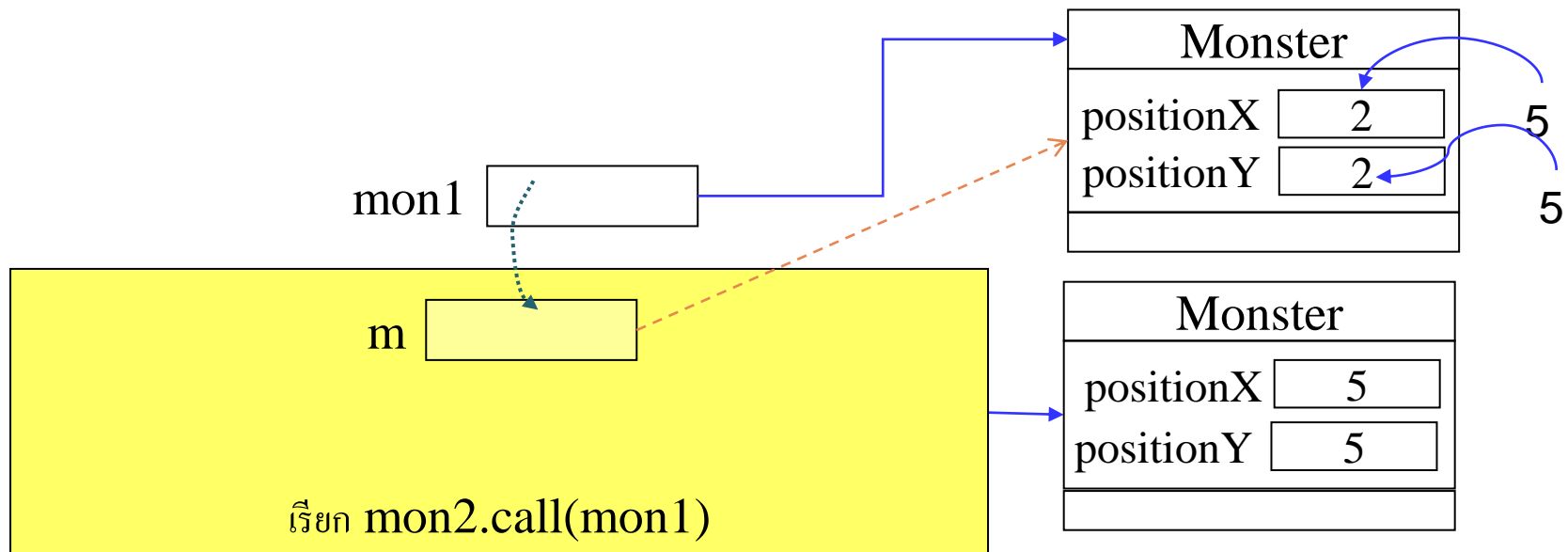
คืนผลลัพธ์

Execute Code

```
public void moveTo(int posX, int posY) {  
    positionX = posX;  
    positionY = posY;  
}  
  
public void call(Monster m) {  
    m.moveTo(positionX, positionY);  
}
```

พารามิเตอร์ชนิดวัตถุ

```
public void call(Monster m) {  
    m.moveTo(positionX, positionY);  
}
```



เมทอดตัวสร้าง (Constructors)

- เป็นเมทอดพิเศษเพื่อใช้ในการสร้างวัตถุของคลาสนั้น
- เป็นเมทอดที่ไม่ใช่ void และไม่ใช่ method ที่คืนค่า
- ชื่อของเมทอดต้องเหมือนกับชื่อของคลาส
- ใช้ **new** เพื่อส่ง message ไปยัง constructor เพื่อให้เกิดการทำงาน
- จุดมุ่งหมายของ constructor
 - เพื่อกำหนดค่าเริ่มต้นให้กับสมาชิกต่าง ๆ ที่อยู่ใน object ให้อยู่ในสถานะที่ถูกต้อง
- Syntax:

```
public <class name> ( <parameter> ) {  
  
}
```

ชนิดของ Constructors

■ Constructor แบบไม่มีพารามิเตอร์ : ทั่วไปมีให้โดยปริยาย

```
public Monster ( ) { <statements> }
```

- เป็น default constructor ที่ถูกกำหนดโดยนัยภายในคลาส หากไม่มีการกำหนด constructor อื่นใด -- สามารถเรียกใช้ได้โดยไม่ต้องประกาศ
- ถ้ากำหนด constructor อื่นไว้ ในคลาสต้องประกาศ constructor แบบนี้เอง

■ Constructor อื่น – ทำ Overloading

- สามารถประกาศ constructor ได้ > 1 constructor แต่ละ constructor ต้องมีรายการของ parameter (parameter list) ที่แตกต่างกัน (จำนวน และ/หรือ data type)

ตัวอย่าง

// 1

```
public ClassA(int x) { ... }
```

// 2 จำนวน parameter ต่างกับ 1

```
public ClassA( ) { ... }
```

// 3 ชนิดของ parameter ต่างจาก 1 และต่างจำนวนกับ 2

```
public ClassA(float x) { ... }
```

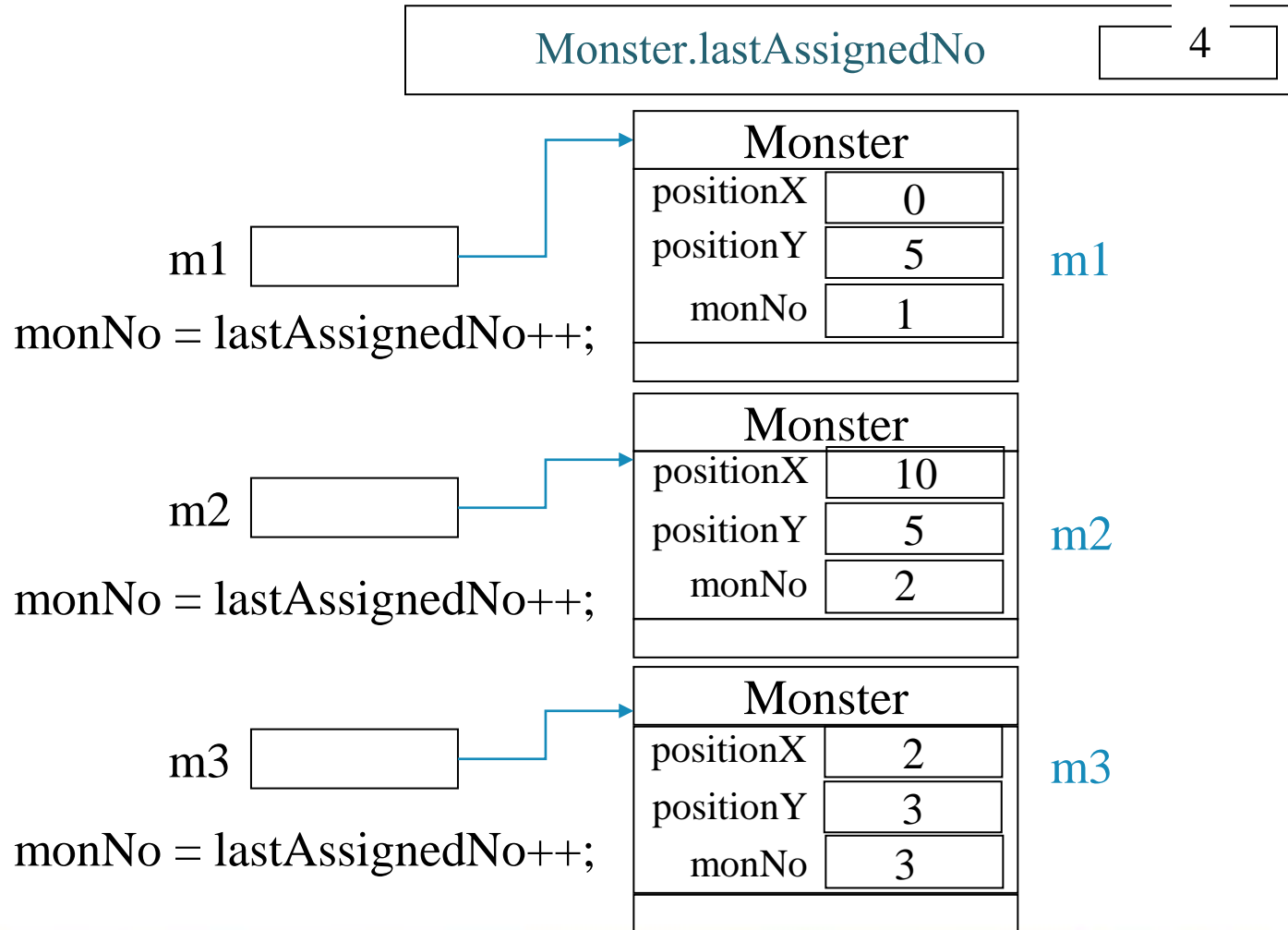
```
public ClassB(int x, int y) { ... }
```

// ชนิดของ parameter และจำนวนเท่ากับ ตัวบน

```
public ClassB(int a, int b) { ... } // invalid constructor
```

ตัวแปร Static VS ตัวแปรวัตถุ

```
public static int lastAssignedNo = 1;
```



ตัวแปรในภาษาจาวา

- อายุหรือขอบเขตของตัวแปร ขึ้นกับ block ที่ครอบตัวแปรนั้น ๆ
 - ตัวแปรของวัตถุ
 - ตัวแปร static
 - ตัวแปร parameter
 - ตัวแปร local

```
public class Monster {  
    private int positionX, positionY; // instance variable  
    private static int lastAssignedNo // static variable  
    public double getDistanceFrom(int posX, int posY) {  
        // posX and posY are parameter variable  
        int xDiff= posX - getPositionX(); // local variable  
        .  
        .  
        .  
    } // scope of local and parameter variable end here  
}
```

ขอบเขตของตัวแปร (1)

- ไม่สามารถประกาศตัวแปรระดับเดียวกัน ที่มีชื่อเดียวกัน ในขอบเขตเดียวกันได้
 - ตัวแปร local กับ ตัวแปรพารามิเตอร์จัดเป็นระดับเดียวกัน
- ประกาศตัวแปรชื่อเดียวกัน อยู่ได้ขอบเขตต่างอันได้

```
public class Monster {  
    private int positionX, positionY; // instance variable  
  
    public void getDistanceFrom(int posX, int posY) {  
        double distance = 0.0; // local variable  
    }  
  
    public void getDistanceFrom(Monster m) {  
        float distance = 0.0F; // local variable; different scope  
    }  
}
```


ขอบเขตของตัวแปร (2)

- ประกาศตัวแปร local ชื่อเดียวกัน ภายใต้ขอบเขตอันเดียวกันไม่ได้

```
public class Monster {  
    private long expPoint; // instance variable  
  
    public void calculateEyeSight(double factor) {  
        double factor = expPoint/1000;  
        // violate! duplicate variable  
    }  
}
```

ขอบเขตของตัวแปร (3)

■ ประกาศตัวแปรต่างระดับ ใช้ชื่อเดียวกัน

```
public class Monster {  
    private long expPoint; // instance variable  
  
    public Monster(long expPoint) {  
        expPoint = expPoint; // OK; wrong meaning  
    }  
}
```

■ ตัวแปร local จะบังตัวแปรวัตถุ ไม่เช่นนั้นต้องระบุ **this**

```
public class Monster {  
    private long expPoint; // instance variable  
  
    public Monster(long expPoint) {  
        this.expPoint = expPoint;  
    }  
}
```

Recursive Methods

- Method ที่เรียกตัวเอง
- ตัวอย่าง

```
methodOne (...) {  
    :  
    methodOne (...) ;  
    :  
}
```

องค์ประกอบที่สำคัญของ recursive method

■ Recursion ประกอบด้วย 2 ส่วนหลัก

- เงื่อนไขตรวจสอบที่จะบอกว่า ทำต่อหรือหยุด

- โดยทั่วไป เมื่อหยุดแล้ว ถ้าต้องคืนค่า จะคืนค่าอะไร

- ตัวเรียก recursive (recursive call) เพื่อให้เกิดการเรียกตัวมันเอง

■ ตัวอย่าง

```
public int factorial (int n) {  
    if (n <= 1) return 1;  
    else return n * factorial(n - 1);  
}
```

ลำดับของการทำงานแบบ recursive

n = 3

```
int factorial (int n) {  
    if (n <= 1) return 1;  
    else return n * factorial(n - 1);  
}
```

n = 2

```
int factorial (int n) {  
    if (n <= 1) return 1;  
    else return n * factorial(n - 1);  
}
```

n = 1

```
int factorial (int n) {  
    if (n <= 1) return 1;  
    else return n * factorial(n - 1);  
}
```

3*2

2*1

1

ใช้ recursion ถ้า

- ทำให้การแก้ปัญหาง่ายและดูเป็นธรรมชาติต่อการเข้าใจ
- การทำ recursion นั้นไม่ก่อให้เกิดการคำนวณซ้ำอย่างมาก
- หากทางแก้ปัญหาโดยการทำ iterative นั้นยุ่งยาก

Packages

- A **package** = กลุ่มของ classes และ interfaces ที่เกี่ยวข้อง ซึ่งจะสนับสนุนการเข้าถึงและการจัดการกับ namespace
- ประโยชน์
 - ทำให้ง่ายที่จะบอกได้ว่า classes และ interfaces เหล่านั้นสัมพันธ์กัน
 - ทำให้รู้ว่าสามารถจะค้นหา classes และ interfaces ได้จากที่ใด
 - ชื่อของ classes นั้นจะไม่ขัดกันกับชื่อของ class ใน packages อื่น เนื่องจาก packages สร้าง namespace อันใหม่
 - ใช้กำหนดการเข้าถึง (access) ให้ต่างกันระหว่าง class ใน package เดียวกันและ class ใน package ที่ต่างกันได้

การตั้งชื่อ package

- ชื่อของ Package ควรชัดเจน และเพื่อให้ชำนาญใช้ชื่อ domain name กลับหลังเป็นคำแนะนำ เช่น
`th.ac.tu.game`
- โดยจะต้องเก็บไว้ภายใต้ directory ที่ชื่อตรงกันกับชื่อของ Package เช่น
`/th/ac/tu/game`
- Scope ของ package statement ครอบคลุมทั้ง file
- หากประกาศมากกว่า 1 class ใน 1 file จะมีเพียง class เดียวที่เป็น public ได้ และ class นั้นต้องชื่อเดียวกับ file
 - class อื่นๆ จะไม่สามารถถูก access จาก package อื่นได้เนื่องจากไม่ได้ประกาศไว้เป็น public

การบรรจุ class ใน package

- เริ่มต้นไฟล์ด้วย ประโยค package

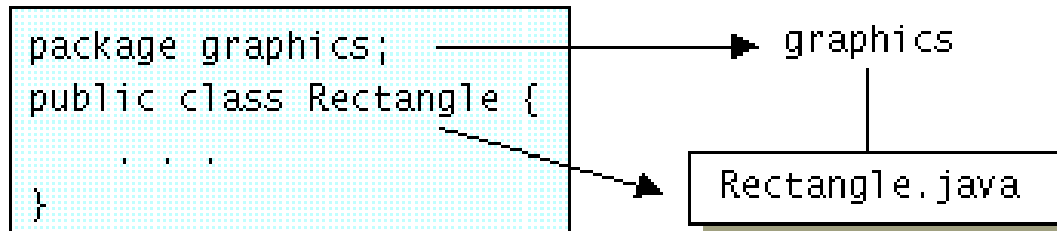
```
package packageName;
```

- ตัวอย่างเช่น

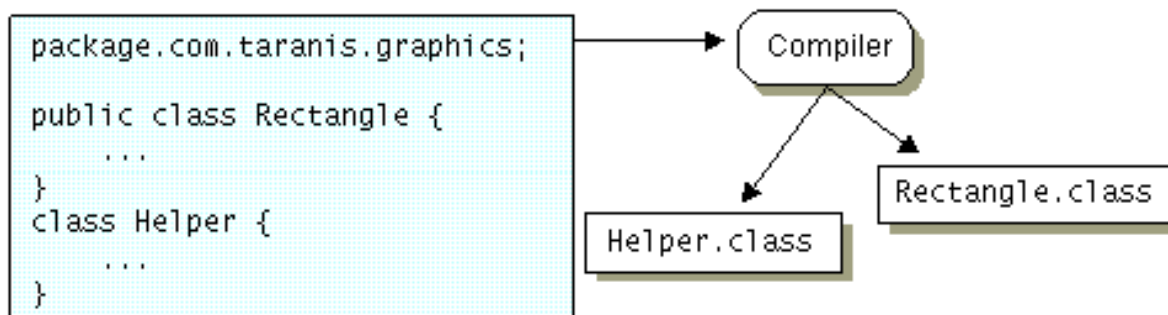
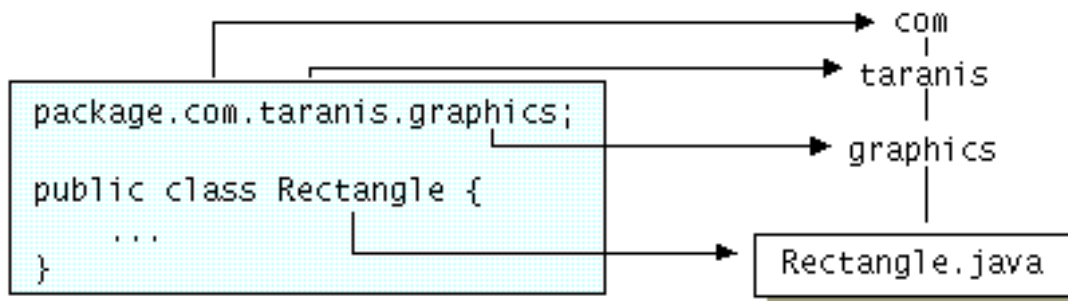
```
package game;  
public class Monster {  
    :  
}
```

- ❑ สร้าง public class *Monster* ซึ่งเป็นสมาชิกใน package *game*
- เก็บ class ใน Package ภายใต้ directory ชื่อเดียวกันกับชื่อ package
 - ❑ Package ไม่มีชื่อถือเป็น default ไม่ต้องประกาศ
 - ❑ ในทุกไฟล์ที่ต้องการกำหนดการเป็นสมาชิกของ package ต้องใส่ package statement ไว้เสมอ
 - ❑ คลาสภายใน package เดียวกัน เห็นกันได้โดยไม่ต้องสั่ง import

ความสัมพันธ์ของชื่อ package กับ directory และชื่อ File



graphics.Rectangle
class name
graphics/Rectangle.java
pathname to file



การเรียกใช้ package

■ ยกเว้น package java.lang การเรียกใช้ package ทำได้ใน 3 ลักษณะ

- ❑ การอ้างถึง member โดยการเรียกชื่อเต็ม (ไม่ต้อง import)

```
java.util.Random r = new java.util.Random( );
```

- ❑ import เฉพาะ package member ที่ต้องการ

```
import java.util.Random;
```

- ❑ import ทุก member ที่อยู่ใน package

```
import java.util.*;
```

- ❑ กรณีที่มีการ import มากกว่า 1 package และใน package นั้นมีชื่อ member เหมือนกัน ต้องระบุให้ชัดเจนว่าอ้างถึงตัวใด

แปลงไคอะแกรมคลาสเป็นโค้ด

Monster

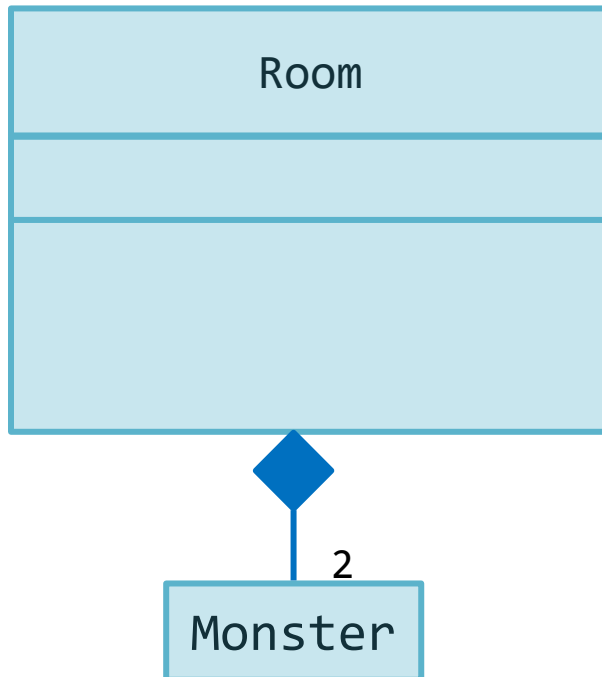
-expPoint: long
-positionX: int
-positionY: int
+lastAssignedNo: int
-BASE_EYE_SIGHT: double = 5.0
+Monster(long)
+getDistanceFrom(int,int): double

```
public class Monster {  
    private long expPoint;  
    private int positionX, positionY;  
    public static int lastAssignedNo;  
    private final double BASE_EYE_SIGHT = 5.0;  
  
    public Monster(long expPoint) { }  
    public double getDistanceFrom(int posX,int posY)  
    {  
        return 0.0;  
    }  
}
```

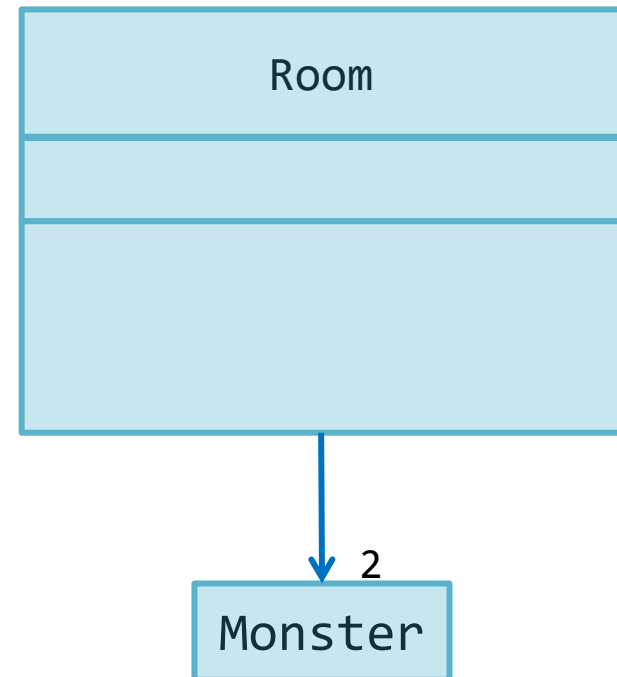
ความสัมพันธ์ระหว่าง Room กับ Monsters

Class Diagram

แบบ Composition

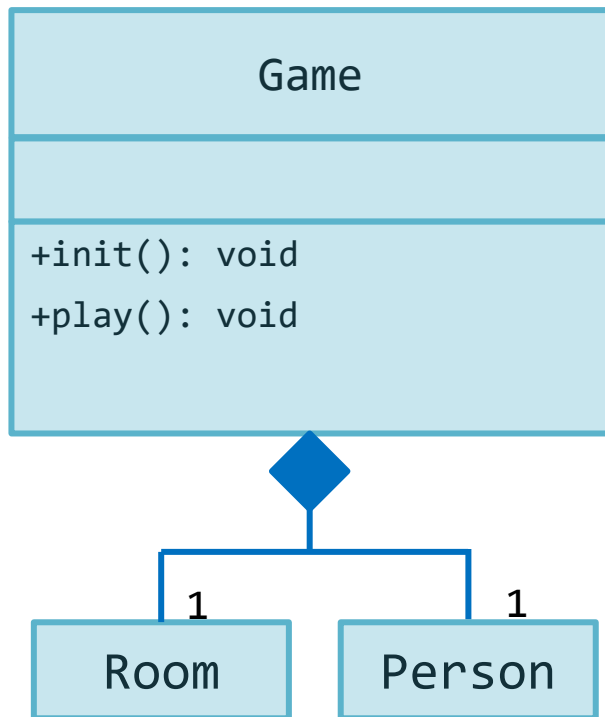


แบบ Association



การแปลง Class Diagram เป็น โค้ด (1)

Class Diagram



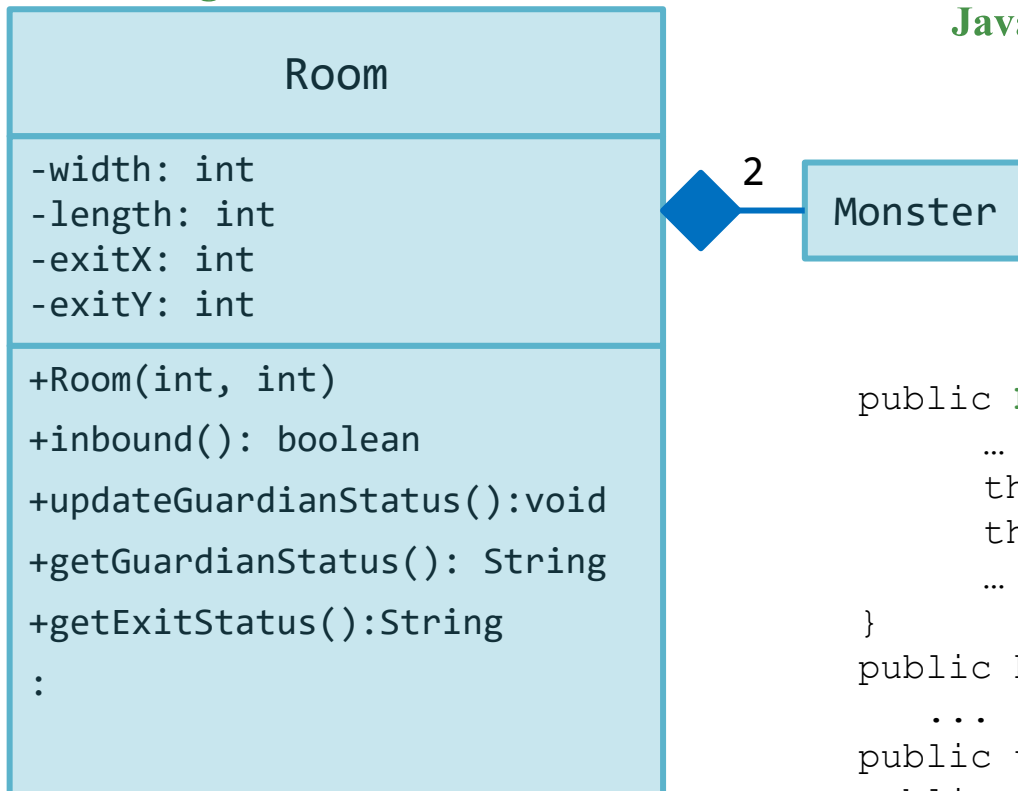
Java Code

```
public class Game {  
  
    private Person player;  
    private Room room;  
  
    public Game() {  
        player = new Player();  
        room = new Room(...);  
    }  
  
    public void init() { }  
    public void play() { }  
  
}
```

การแปลง Class Diagram เป็น โค้ด (2)

กรณีความสัมพันธ์ระหว่าง Room และ Monster เป็นแบบ **Composition**

Class Diagram



Java Code

```
public class Room {
    private Monster guardian1;
    private Monster guardian2;
    private int width;
    private int length;
    private int exitX;
    private int exitY;

    public Room(int width, int length){
        ...
        this.guardian1 = new Monster(...);
        this.guardian2 = new Monster(...);
        ...
    }

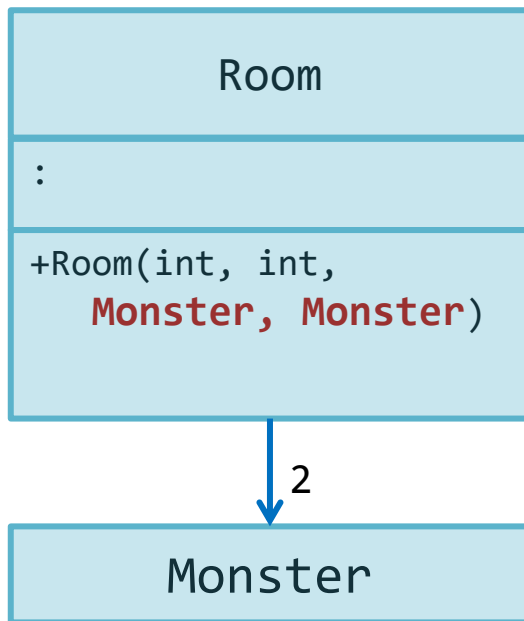
    public boolean inbound(int posX, int posY) {
        ... }

    public void updateGuardianStatus() {...}
    public String getGuardianStatus() {...}
    public String getExitStatus() {...}
    ...
}
```

การแปลง Class Diagram เป็น โค้ด (3)

กรณีความสัมพันธ์ระหว่าง Room และ Monster เป็นแบบ **Association**

Class Diagram



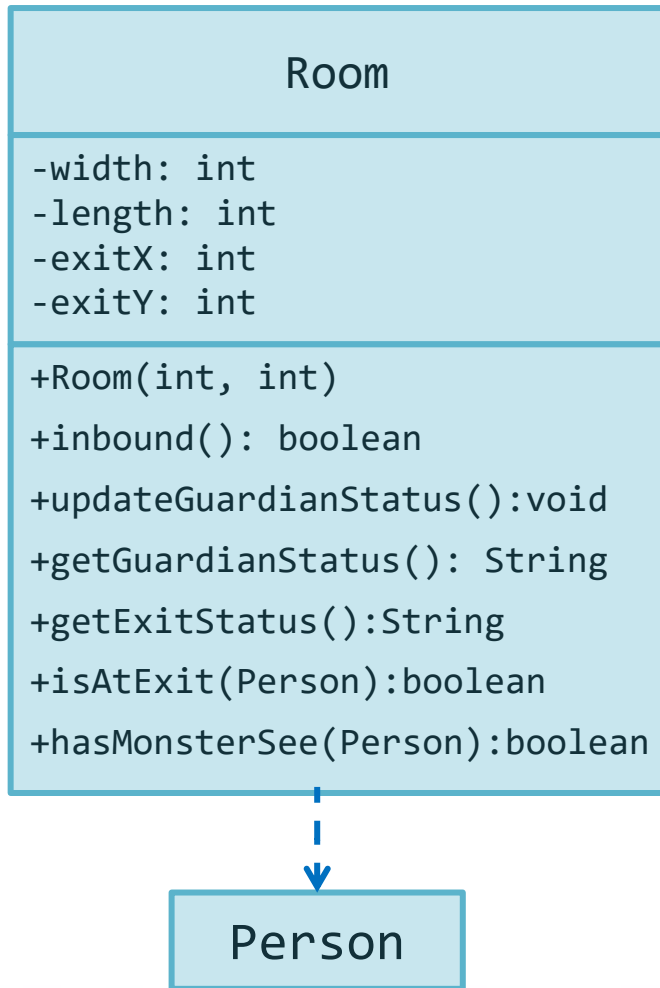
Java Code

```
public class Room {
    private Monster guardian1;
    private Monster guardian2;
    ...

    public Room(int width, int length,
        Monster mon1, Monster mon2) {
        ...
        guardian1 = mon1;
        guardian2 = mon2;
    }
}
```


การแปลง Class Diagram เป็น โค้ด (4)

Class Diagram



Java Code

```
public class Room {
    private Monster guardian1;
    private Monster guardian2;
    private int width;
    private int length;
    private int exitX;
    private int exitY;

    public Room(int width, int length){
        ...
        this.guardian1 = new Monster(...);
        this.guardian2 = new Monster(...);
        ...
    }

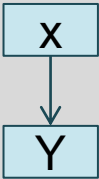
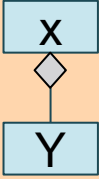
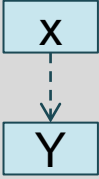
    public boolean inbound(int posX, int posY) {
        ... }

    public void updateGuardianStatus(){...}
    public String getGuardianStatus(){...}
    public String getExitStatus(){...}
    public boolean isAtExit(Person player){...}
    public boolean hasMonsterSee(Person player) {
        ...}
}
```

สรุปการแปลงไคอะแกรมเป็นโค้ด

ไคอะแกรม	จาวา	ตัวอย่าง
ชื่อคลาส	คลาส	Game → <code>public class Game { }</code>
+/-	public/private	<code>+init():void</code> → <code>public void init() { }</code>
Attribute	ตัวแปรวัตถุ	<code>-player:Person</code> → <code>private Person player;</code>
Method	เมทอด	<code>+play():void</code> → <code>public void play() { }</code>
= ค่าคงที่	final	<code>-VALUE:int = 2</code> → <code>private final int VALUE=2;</code>
จีดเส้นใต้	static	<code>-<u>lastNo</u>:int</code> → <code>private static int lastNo;</code>

สรุปการแปลงไคอะแกรมเป็นโค้ด

ไคอะแกรม	จาวา	ตัวอย่าง
ความสัมพันธ์ (Association)	ตัวแปรวัตถุ (ชนิดคลาสที่ สัมพันธ์ด้วย)	 <pre>→ public class X { private Y yObject; ... }</pre>
องค์ประกอบ (Composition)	ตัวแปรวัตถุ (ชนิดคลาสที่ สัมพันธ์ด้วย)	 <pre>→ public class X { private Y yObject; public X() { yObject = new Y(); } }</pre>
Dependency	พารามิเตอร์หรือ ตัวแปรท้องถิ่น ในเมทอด	 <pre>→ public class X { public void someMethod(Y y) { ... } }</pre>

สรุปการเรียนรู้วันนี้

- เรียนรู้เกี่ยวกับการส่งและคืน parameters ผ่านและจาก method
- เข้าใจความแตกต่างระหว่าง instance method กับ static method
- เข้าใจจุดประสงค์และการใช้ constructors
- เข้าใจแนวคิดเกี่ยวกับ static variables
- เข้าใจเกี่ยวกับขอบเขตของตัวแปร
- เข้าใจการเขียนโปรแกรมแบบ recursive method
- การบรรจุและใช้ class ใน package
- การแปลงไคอะแกรมให้เป็นโค้ด