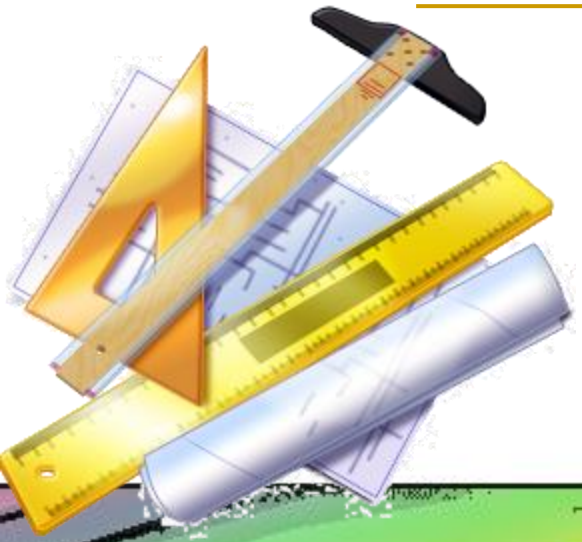
การสืบทอด

Inheritance



Lecture 9

Yaowadee Temtanapat

เยาวดี เต็มธนาภัทร์

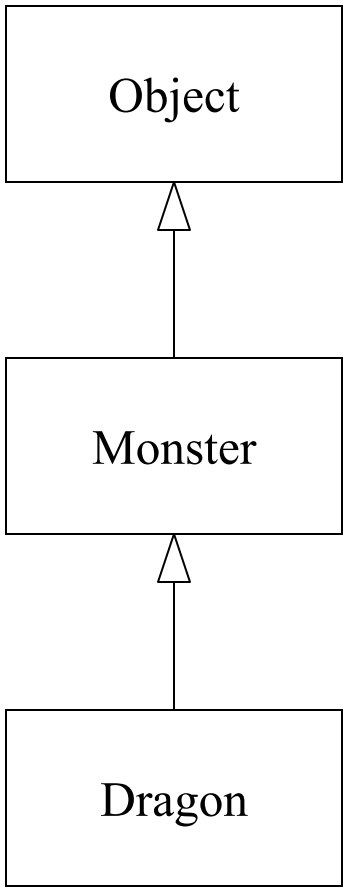
วัตถุประสงค์ของการเรียนวันนี้

- เรียนรู้เกี่ยวกับการสืบทอดและการเขียนทับ (override) เมท็อดของ superclass
- สามารถเรียก constructors ของ superclass
- เรียนรู้เกี่ยวกับจาวา access control ระดับ protected และ package
- สามารถแปลงระหว่าง supertype และ subtype reference ได้
- เข้าใจเกี่ยวกับ superclass Object และการเขียนทับเมท็อด: toString, equals และ clone

Inheritance

- Class สืบทอดสถานะและพฤติกรรมจาก Superclass
  - นอกจากนี้อาจเพิ่ม attributes หรือเพิ่ม/กำหนด method ใหม่ (add หรือ redefine method)
- Inheritance เตรียมกลไกสำหรับโครงสร้างและองค์ประกอบของโปรแกรม เพื่อรองรับการสืบทอด
- ประโยชน์ของ Inheritance
  - Subclasses จะทำพฤติกรรมพิเศษที่เพิ่มแตกต่างไปจากส่วนพื้นฐานที่มีอยู่ใน superclass นั่นคือทำให้สามารถจะ reuse code ใน superclass ได้

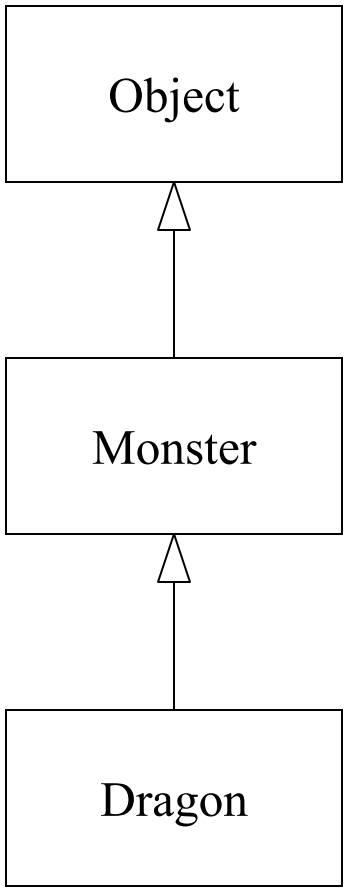
Inheritance Diagram




```
public class Monster {
    private final double EYE_SIGHT = 5;
    private boolean sleep;
    private int positionX;
    private int positionY;
    private double eyeSight;

    public Monster(){ … }
    public boolean isSleep(){ … }
    public void setSleep(boolean status){ … }
    public int getPositionX(){ … }
    public void setPositionX(int x){ … }
    public boolean canSee(Person player){ … }
// … omit other methods
```

Inheritance Diagram



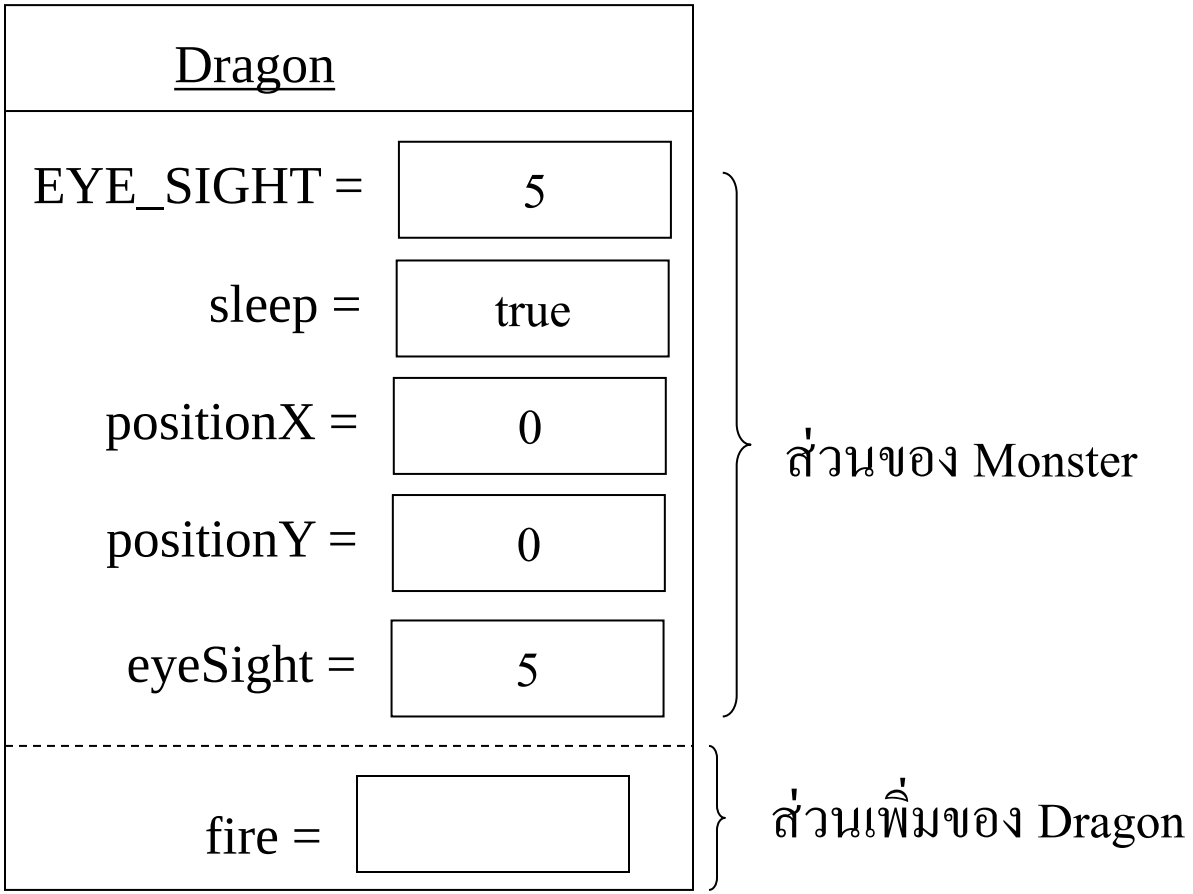
Syntax:

```
class SubclassName extends SuperclassName {
    new methods
    new instance variables
}
```

ตัวอย่าง: Dragon คือ Monster ที่สามารถพ่นไฟได้

```
public class Dragon extends Monster
{
    private int fire;
    public int getFirePower{...}
}
```

Layout ของ subclass object



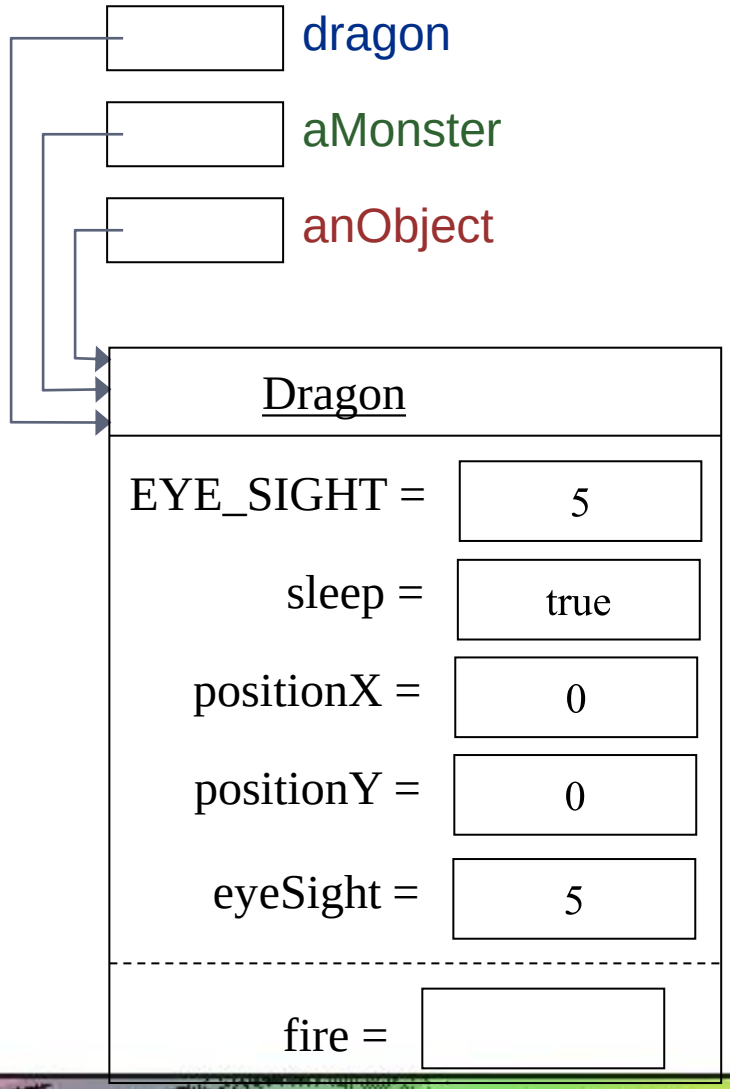
สังเกตว่า

- **extends** บอกให้รู้ว่า Dragon เป็น object ที่ขยายเพิ่มจาก Monster
  - Dragon มีรายละเอียดเพิ่มเติม (fire) จาก Monster
  - Dragon มีความสามารถเพิ่มเติม (getFirePower()) จาก Monster
  - แต่เรียก Dragon ว่า **subclass** และ Monster ว่า **superclass** สอดคล้องกับทฤษฎีเซต เซตของ Monster เป็นเซตที่มีขนาดใหญ่กว่าเซตของ Dragon

สังเกตว่า

- ข้อแตกต่างระหว่าง Inheritance กับการ realize อินเทอร์เฟส
  - Interface ไม่ใช่ class
  - Interface ไม่มี instance attributes หรือ methods ให้สืบทอดได้
    - เมท็อดเป็นเพียงข้อกำหนด ไม่มี body
    - สามารถมีตัวแปรค่าคงที่ได้
  - ในจาวา
    - Realize ได้**มากกว่า 1** อินเทอร์เฟส
    - Inherit ได้จาก**เพียง 1 คลาสเท่านั้น**

การแปลงระหว่าง class types



- Dragon ขยายจาก Monster หรือเป็นกรณีพิเศษของ Monster
- ดังนั้นวัตถุของตัวแปร Dragon สามารถถูกเก็บในตัวแปรของ Monster

```
Dragon dragon = new  Dragon();
Monster aMonster = dragon;
Object anObject = dragon;
```

- แต่ aMonster ไม่รู้ method ใน Dragon

```
aMonster.setPositionX(10); // OK
aMonster.getFirePower(); // Not OK
```

- ไม่สามารถแปลงตัวชี้ ข้าม class ที่ไม่เกี่ยวข้องกัน

```
Rectangle r = dragon; // Not OK
```

การตรวจสอบชนิดของ object: instanceof

- สามารถแปลงกลับได้โดยการทำ cast

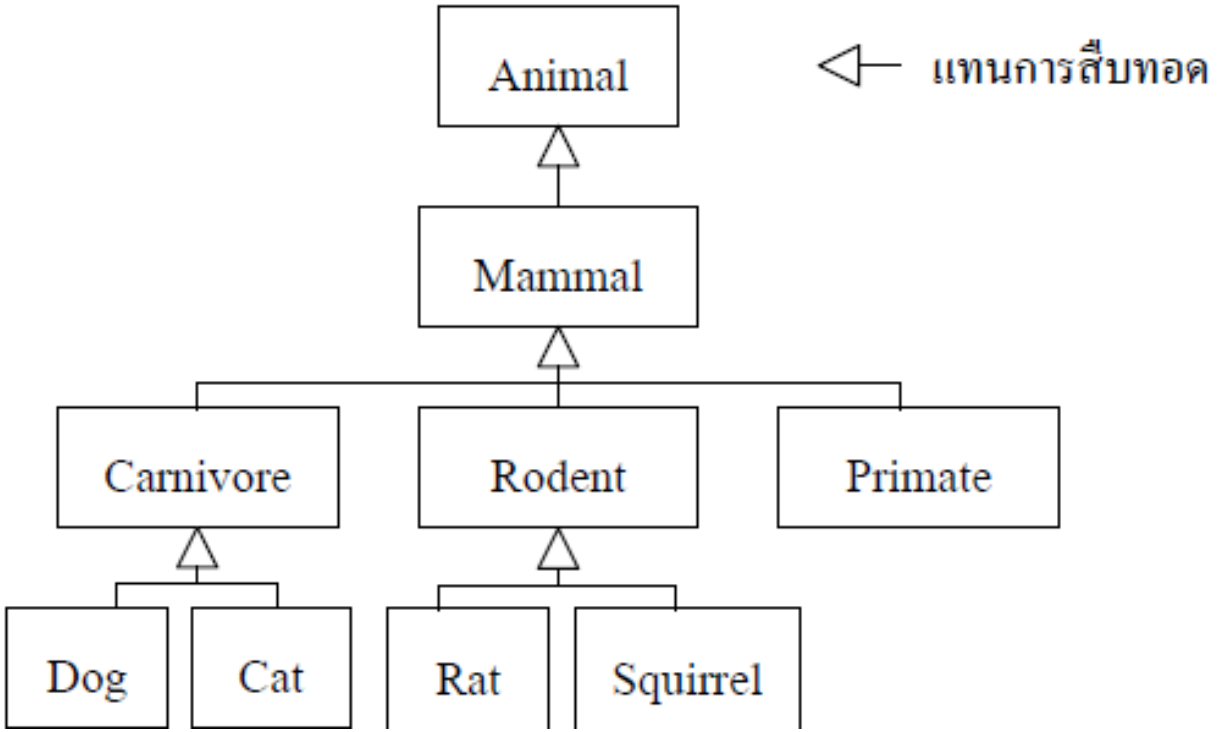
```
Dragon aDragon = (Dragon) anObject; // OK
```

  - ถ้าแปลงผิด class จะเกิด ClassCastException เมื่อ run

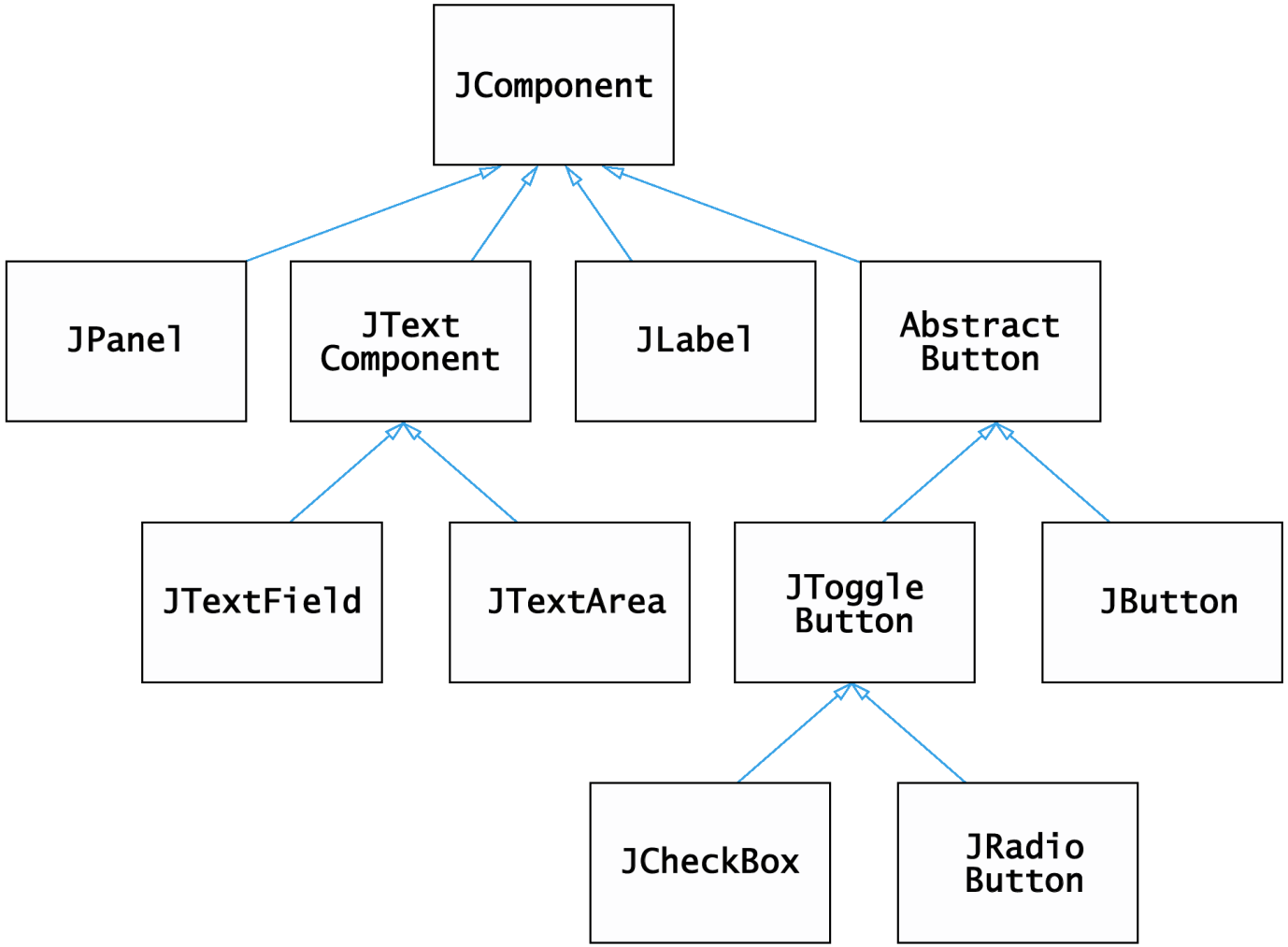
- การทดสอบก่อนการแปลงทำได้ (เช่นเดียวกับการทดสอบ interface) โดย operator **instanceof**

```
Dragon aDragon;
if (anObject instanceof Dragon)
   aDragon = (Dragon) anObject;
else
   aDragon = null;
```

Inheritance Hierarchy: Animal



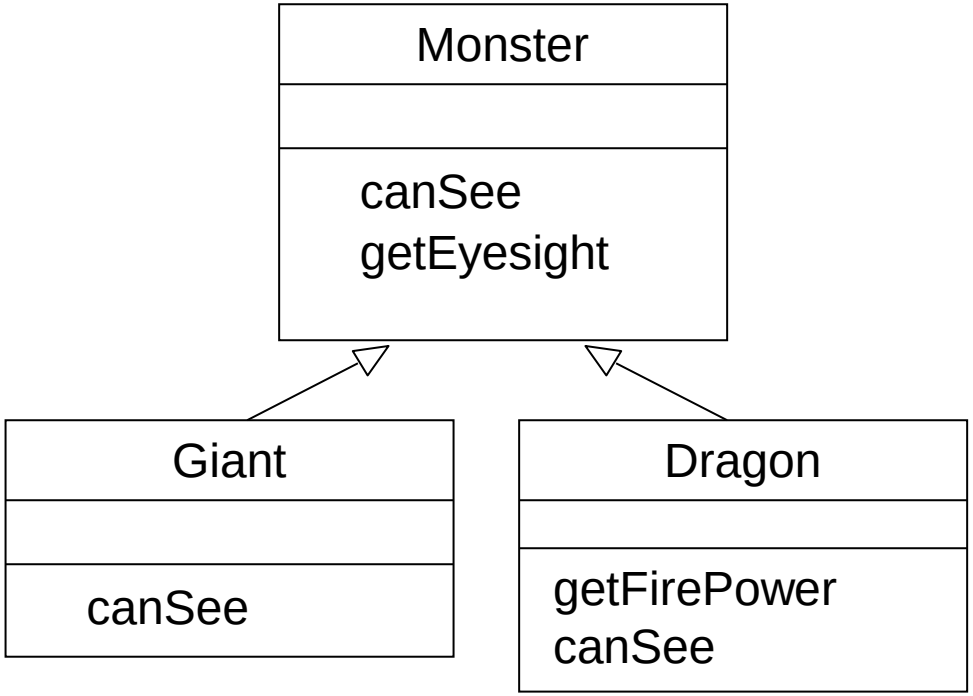
Inheritance Hierarchy: Swing UI Components



Inheritance Hierarchy: Monster class

สัตว์ประหลาด:

1. Dragon สามารถพ่นไฟ
2. Giant สามารถขว้างก้อนหินได้ โดยขว้างได้ไกลเป็นสองเท่าของระยะมองเห็น



เมท็อด canSee คืนจริง หากคนอยู่ในระยะที่สัตว์ประหลาดทำร้ายได้ (จากการเห็น พ่นไฟ หรือขว้างหินถึง)

การสืบทอด methods

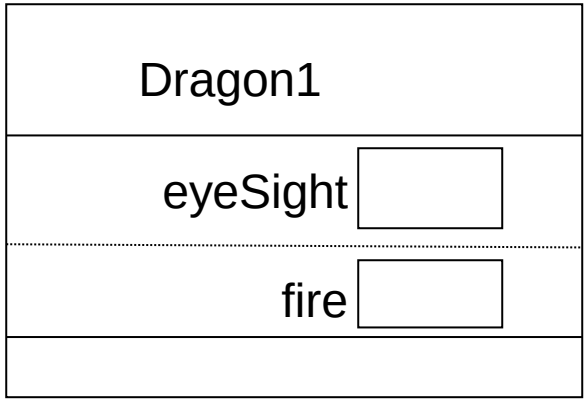
- เมท็อดใน subclass มีได้ใน 3 ลักษณะ
  - **สืบทอด method มาจาก superclass**: method จะถูกสืบทอดจากแม่มายังลูกหากไม่มีการกำหนดทับ
  - **Override methods ของ superclass**:หากกำหนด method ให้มีชื่อและชนิดของ parameters ของ method เหมือนกับ superclass
    - subclass object จะเรียกใช้ overridden method แทน method แม่
  - **เพิ่ม method ใหม่**: กำหนด method ใหม่ที่ไม่เคยมีใน superclass
    - สามารถเรียกใช้ได้จากเพียง subclass ที่กำหนด method ใหม่นี้ขึ้นเท่านั้น

การสืบทอดตัวแปร

- ตัวแปรจะ**ไม่ถูกเขียนทับ**โดย subclass
- ดังนั้นตัวแปรใน subclass มีได้ใน 2 ลักษณะ
  - **สืบทอดตัวแปรมาจาก superclass**: ตัวแปรของ superclass จะถูกสืบทอดจากแม่มายังลูกโดยอัตโนมัติ
  - **ตัวแปรใหม่**: กำหนดตัวแปรใหม่ที่ไม่เคยมีใน superclass
    - สามารถเรียกใช้ได้จากเพียง subclass ที่กำหนด method ใหม่นี้ขึ้นเท่านั้น

Shadow Instance Variable

กรณีที่กำหนดตัวแปรทับกับตัวแปรใน superclass ตัวแปรนั้น จะมีอยู่แต่เข้าถึงไม่ได้ใน class ลูก



ส่วนของ Monster1

```
public class Dragon1 extends Monster1 {
    private int fire;
    public double eyeSight;
    ...
}
```

```
public class Monster1Test{
    public static void main(String[] args){
        Dragon1 dragon = new Dragon1(30);
        System.out.println("Eyesight for dragon: " +
dragon.getEyeSight());
    ...
}
```

อ่านค่า eyeSight ใน Dragon1

แต่ getEyeSight() ซึ่งสืบทอดจาก superclass จึงอ่านค่า eyeSight ที่อยู่ใน superclass

การเขียนทับและเรียกเมท็อดซุปเปอร์คลาส

```
public class Dragon extends Monster {
    private int fire;

    public int getFirePower(){
       return fire;
    }
    @Override
    public boolean canSee(Person player){
       double distance = getDistanceFrom(player);
       if(distance > eyeSight && distance > fire)
           return false;
       else
           return true;
    }
}
```

ไม่สามารถเข้าถึงตัวแปร eyeSight ใน Monster เพราะเป็น private

การเรียก superclass method

- **จุดมุ่งหมาย** เพื่อที่จะเรียก method ที่อยู่ใน superclass แทน method ที่อยู่ใน class ปัจจุบัน
- **Syntax:**

```
returnType methodName(parameters) {
    super.methodName(parameters);
}
```

- **ตัวอย่าง**

```
public boolean canSee(Person player) {
    double distance = getDistanceFrom(player);
    boolean seen = super.canSee(player) || distance <= fire;
    return seen;
}
```

หากเรียก `canSee(player);`
หมายถึง
`this.canSee(player);`

การเรียก superclass constructor

- **จุดมุ่งหมาย** เพื่อที่จะเรียก constructor ที่อยู่ใน superclass
  - ต้องเป็น statement แรกใน subclass constructor
- **Syntax:**

```
ClassName(parameters) {
    super(parameters);
}
```

- **ตัวอย่าง**

```
public class Dragon extends Monster{
   public Dragon() {
     super("Dragon");
     fire = 3;
   }
}
```

Inheritance และ Constructor: กรณีไม่มี constructor เอง

```
class Person {
    public void sayHello( ) {
       System.out.println(“Hello”);
    }
}
```

มีความหมายเท่ากับ

```
class Person {
    public Person( ) {
        super( );
    }
    public void sayHello( ) {
        System.out.println("Hello");
    }
}
```

Compiler เพิ่มให้โดยอัตโนมัติ

ในจาวา, ถ้าไม่ extends class ใด extends โดยปริยายจาก Object

Constructor: กรณีมี Constructor เอง (1)

- หากประกาศ Constructor ภายใน class เอง ไม่มีการสร้าง default constructor ให้อัตโนมัติ

```
class Student extends Person {
    public Student(String name) {
        ...
    }
}

Student aStudent = new Student(); //INVALID
                                  //no matching constructor
```

Constructor: กรณีมี Constructor เอง (2)

- หากกำหนด Constructor โดยไม่เรียก super class constructor, compiler เพิ่มให้โดยอัตโนมัติ

```
class Student
      extends Person {
   private String name;

   public Student( ) {
        name = "unknown";
   }
}
```

```
class Student
      extends Person {
   private String name;

   public Student( ) {
        super();
        name = "unknown";
   }
}
```

ระวัง (Caution) !!

```
class Vehicle {
  private String vid;
  public Vehicle(String vNo)
  {
      vid = vNo;
  }

  public String getVid() {
      return vid;
  }
}
```

```
class Car extends Vehicle {
  private int noOfSeats;
  public void setSeat(int n)
  {
      noOfSeats = n;
  }

  public int getSeat() {
      return noOfSeats;
  }
}
```

Compilation Error เนื่องจากไม่มีการกำหนด constructor สำหรับ Car( )
compiler เพิ่ม **`public Car() { super( );}`** แต่ไม่มี
constructor ที่ไม่รับพารามิเตอร์ใน superclass Vehicle

สังเกตุ

- Constructors ของ super class ไม่สืบทอดมายัง subclass
- แยกความแตกต่างจากการที่ compiler เพิ่ม default constructor ให้ในกรณีไม่กำหนด

```
person1 = new Student( );
person2 = new Person( );
```

- ถูกต้องเนื่องจากใช้ default constructor ที่เพิ่มโดย compiler ไม่ใช่เนื่องจาก inheritance

Polymorphism (ความสามารถที่จะปรากฏได้ในหลายรูป)

- ความสัมพันธ์เชิงสืบทอด มักถูกเรียกว่าเป็นความสัมพันธ์**แบบ Is-a**
- Object ใน subclass นั้นเป็น superclass object ที่มีคุณลักษณะเพิ่ม

- polymorphism *ทำให้สามารถเรียกใช้ method ชื่อเดียวกันจาก class ที่แตกต่างกัน และให้ผลการทำงานที่ต่างกัน โดยพิจารณาตามชนิดที่แท้จริงของ object*
  - อาศัยคุณลักษณะของ Overridden methods ที่แตกต่างกันใน subclass และ superclass

Monster และ Giant

```
public class Monster {
    // … omit
    public double getEyeSight() { return eyeSight; }
}

public class Giant extends Monster {
   private double boulderPower;
   public Giant(int x, int y) {
       super(x, y);
       boulderPower = 2 * getEyeSight();
   }
   public boolean canSee(Person player) {
       return getDistanceFrom(player) <= boulderPower;
   }
}
```

MonsterTest.java

```
public class MonsterTest {
    public static void main(String[] args) {
        Monster monster = new Monster();
        monster.setPositionX(30);
        monster.setPositionY(40);
        Dragon dragon = new Dragon(7);
        dragon.setPositionX(30);
        dragon.setPositionY(40);
        Giant giant = new Giant(30, 40);
        Person player = new Person();
        player.setPositionX(38);
        player.setPositionY(40);
        print("Monster ", monster, player);
        print("Dragon ", dragon, player);
        print("Giant ", giant, player);
    }
    public static void print(String name, Monster beast, Person person) {
        String seen = beast.canSee(person)? "see": "cannot see";
        System.out.println(name + seen + " the person");
    }
}
```

เมท็อด print ทางานได้กับวัตถุหลายชนิด และให้พฤติกรรมการทำงานตามแต่วัตถุจริงในขณะ runtime

Abstract classes

- Abstract class =
  - class ที่กำหนดโดยมี modifier abstract
  - ไม่สามารถสร้าง instance จากมันได้

Abstract class

```
public abstract class Animal {
    protected int age;
    protected String name;
     :
    abstract public void move( );
    public String getName( ) {
        return name;
    }
}
```

Abstract method

```
public class Dog extends Animal {
    public void move(){ … }
}
```

Abstract methods

- Abstract method =
  - method ที่มี keyword abstract เป็น modifier
  - ปิดท้ายด้วย ; ==> ไม่มี method body (ไม่กำหนดการทำงาน)
- Private และ Static methods ประกาศเป็น abstract methods **ไม่ได้**
- Class เป็น abstract class ถ้า
  - Class ประกาศตัวขยายเป็น abstract
  - Class มีเมท็อดที่เป็น abstract method หรือไม่ได้ทำ implementation ของ inherited abstract method

Interfaces VS Abstract Class

- ดังนั้นอาจมองได้ว่า Interface ก็คือ Abstract class ที่
  - ไม่มี instance variables
  - ทุก method ใน interface **ต้อง**เป็น abstract นั่นคือไม่มีการ implement body
    - ไม่จำเป็นต้องประกาศด้วยคำสงวน abstract นำหน้าเมท็อด
  - ทุก method **เป็น public** โดยอัตโนมัติ

การเรียกใช้ Interface (Revisited)

- **จุดมุ่งหมาย** เพื่อที่จะกำหนด class ใหม่ซึ่งจะ implement methods ของ interface
- **Syntax:**

```
class SubclassName
      implements InterfaceName1, InterfaceName2 ... {
    methods
}
```

- **ตัวอย่าง**

```
public class Dragon extends Monster
      implements Movable {
    public void setPostionX() { ... }
    public void setPostionY() { ... }
}
```

Inheritance และ member accessibility

- การเข้าถึง (accessibility) ข้อมูลจากการใช้ inheritance สำหรับ modifier
  - public
  - private
  - protected (เข้าถึงได้โดย subclass และ package)
  - ไม่มีตัวขยาย: package access

Specifier	class	package	subclass	world
private	X			
-	X	X		
protected	X	X	X	
public	X	X	X	X

ข้อแนะนำการใช้ระดับของ Access

- Attribute: ใช้ private เสมอ
- Methods: ใช้ public หรือ private
- Class: ใช้ public หรือ package
- ระวัง! การใช้ package access แบบไม่ตั้งใจ (ลืมใส่ modifier)

Superclass ของทุกวัตถุ

- *Object*: ทุกคลาสในจาวาที่ไม่ extends จากคลาสอื่น ๆ extends จาก Object โดยตรง (อัตโนมัติ)
- **methods ที่มีประโยชน์ใน Object**
  - `String toString()` : คืน String ที่ใช้แสดง object นั้นเพื่อการดีบัก
  - `boolean equals(Object other)` : ทดสอบว่า content ของ object นั้นเท่ากับอีก object หรือไม่
  - `Object clone()` : ทำสำเนา object แบบเต็มรูปแบบ
- สามารถ override method เหล่านี้เพื่อให้เกิดผลตามต้องการ

Monster.java: ตัวอย่างการ override

อาจใช้ annotation @Override เพื่อบอกให้ compiler รู้ว่าตั้งใจที่จะทำ override

```
public class Dragon extends Monster { ...
    public String toString() {
        return getClass().getName() +" [" + getStatus() +"]";
    }
    public boolean equals(Object otherObject) {
        if (otherObject instanceof Dragon) {
            Dragon other = (Dragon)otherObject;
            return fire == other.fire;
        }
        else
            return false;
    }
    public Dragon clone() {
        Dragon clonedDragon = new Dragon(fire);
        clonedDragon.setPositionX(getPositionX());
        clonedDragon.setPositionY(getPositionY());
        clonedDragon.setSleep(isSleep());
        clonedDragon.fire = this.fire;
        return clonedDragon;   }
}
```

สรุปการเรียนในวันนี้

- การสืบทอด: วัตถุใน subclass สืบทอดลักษณะจาก superclass
- ลักษณะ method ที่สืบทอดอาจ override ได้ใน subclass เพื่อให้มีพฤติกรรมการทำงานที่แตกต่างจาก superclass → polymorphism
- Constructors ของ superclass จะไม่ถูกสืบทอดมายัง subclass
- Subclass มีลักษณะทั้งหมดของ superclass จึงสามารถใช้ตัวแปรที่อ้างถึงวัตถุระดับ superclass ในการอ้างถึง subclass ได้ แต่ไม่เป็นในทางตรงข้าม
- ในจาวาทุกวัตถุสืบทอดจาก superclass Object
  - การ override methods: toString, equals และ clone ช่วยให้การพิมพ์ การเปรียบเทียบ และการสร้างแบบ clone ทำได้สะดวกขึ้น