# บทที่ 6

# Polymorphism

# &

# Composition

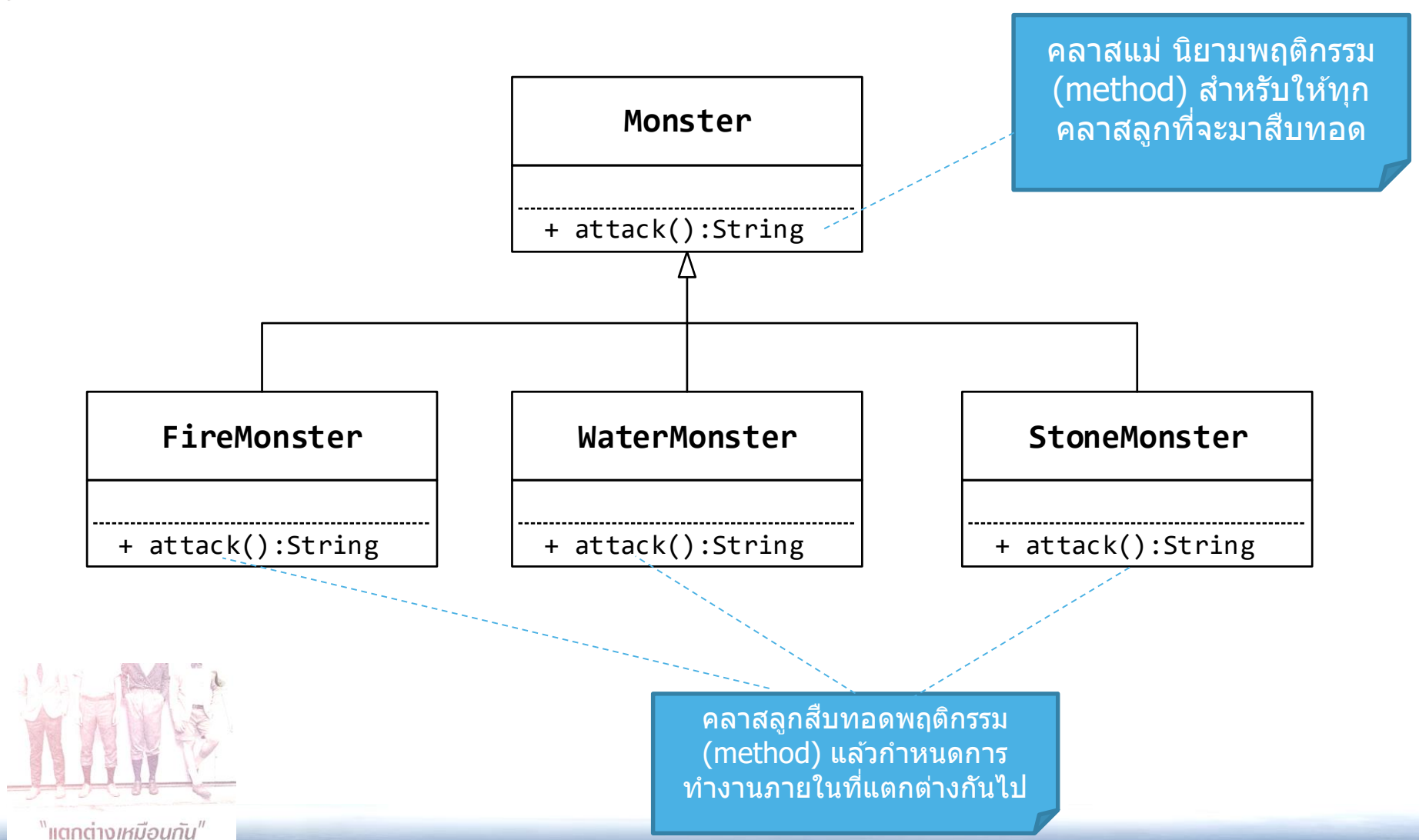
# Polymorphism

- Polymorphism แปลว่า "*ความหลากหลาย*"

- ในทาง OOP คือ รูปแบบของ object ที่สามารถมีหลากหลายการทำงาน ซึ่งเกิดจากการสืบทอดคลาส โดยยังคงคุณสมบัติของคลาสแม่เอาไว้อยู่

- ช่วยให้การรับและส่งข้อมูลไม่เฉพาะเจาะจงเฉพาะ object ของคลาสเดียวเท่านั้น อาจมีทั้งคลาสลูกหรือคลาสหลานก็ได้

# คลาสแม่ กำหนดพฤติกรรมแก่คลาสลูก



# ตัวอย่าง

```
public class TestApp {
  public static void main(String args[]) {
    // ประกาศชนิด object ในชื่อคลาสลูกและสร้าง object จากคลาสเดียวกับที่ประกาศ
    FireMonster fm = new FireMonster();
    callAttack1(fm);

    // ประกาศชนิด object เป็นคลาสแม่ แต่สร้าง object ของคลาสลูก
    Monster m1 = new FireMonster();
    Monster m2 = new WaterMonster();
    Monster m3 = new StoneMonster();
    Monster m4 = new Monster();

    callAttack2(m1);
    callAttack2(m2);
    callAttack2(m3);
    callAttack2(m4);
  }

  public static void callAttack1(FireMonster m) {
    // เรียก attract จาก FireMonster
    System.out.println(m.attack());
  }

  // เมธอดรับพารามิเตอร์ได้หลากหลาย object
  public static void callAttack2(Monster m) {
    // เรียก attract ตามชนิดของ object ที่ส่งมา
    System.out.println(m.attack());
  }
}
```

เมธอดรับได้เฉพาะ object ของคลาส FireMonster เท่านั้น

เมธอดรับได้ทั้ง object ของคลาสแม่และลูก ทำให้มีความหลากหลาย (Polymorphism)

*ตัวอย่างหน้าจอ*

```
Attack with fire!
Attack with fire!
Attack with water!
Attack with stones!
I don't know how to attack!
```

**คลาสแม่**

```
class Monster {
  // นิยามพฤติกรรมให้แก่คลาสลูกที่จะมาสืบทอด
  public String attack() {
    return "I don't know how to attack!";
  }
}
```

**คลาสลูก**

```
class FireMonster extends Monster {
    public String attack() {
      return "Attack with fire!";
    }
}

class WaterMonster extends Monster {
    public String attack() {
      return "Attack with water!";
    }
}

class StoneMonster extends Monster {
    public String attack() {
      return "Attack with stones!";
    }
}
```

# Interface

- Interface คือ คลาสที่มีเพียงต้นแบบของฟังก์ชันเท่านั้น ไม่มีส่วนการทำงานจริง

- Interface ใช้ในการกำหนดว่าคลาสนั้นทำอะไรได้บ้าง ทำให้คลาสที่สืบทอดมีการทำงานแตกต่างกันไปตามที่นักพัฒนากำหนด แต่ยังคงชื่อและตัวแปรรับค่า (parameter) สำหรับเรียกใช้เหมือนเดิม

# รูปแบบ Interface

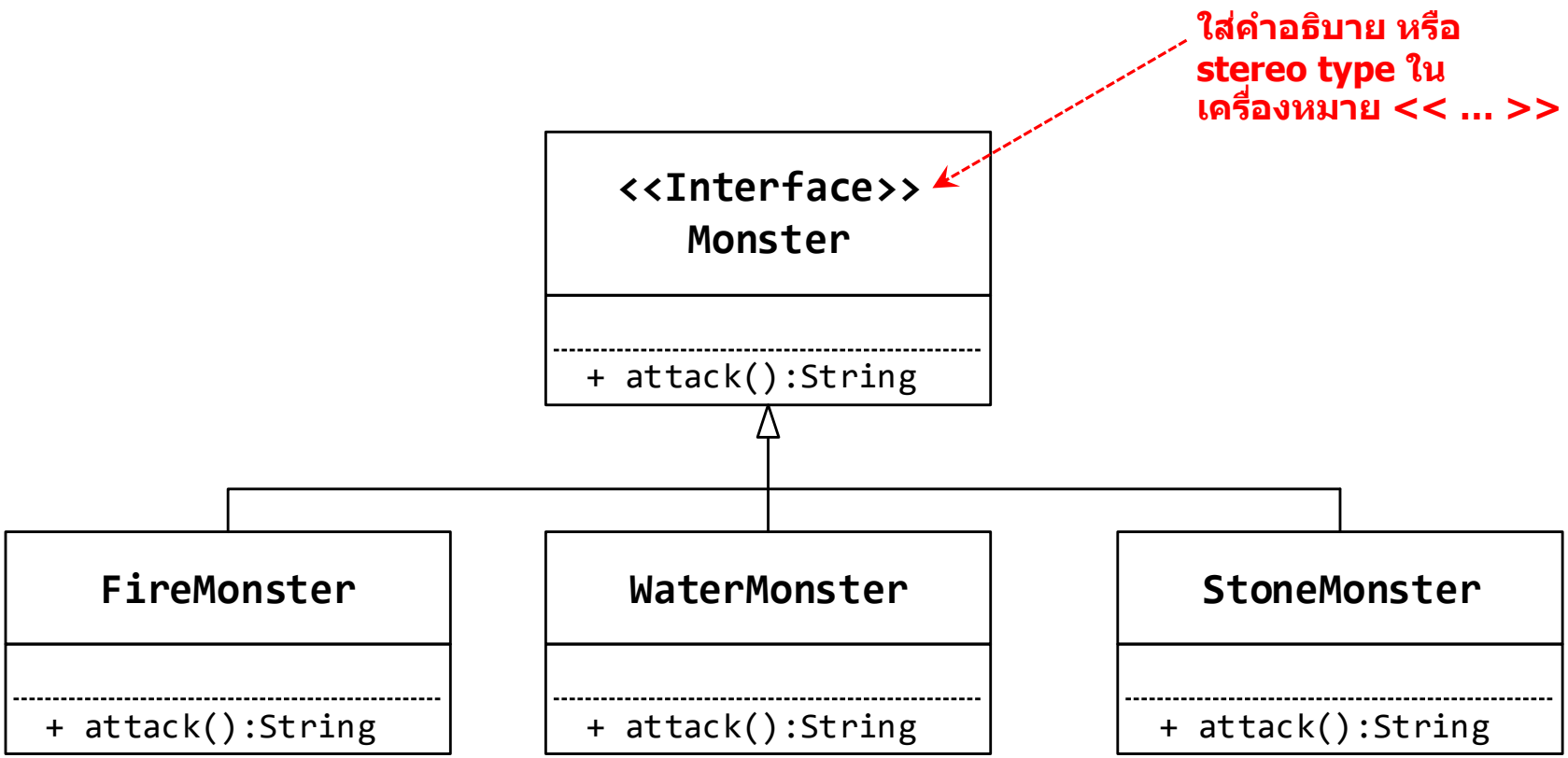
```
interface ชื่อคลาสแม่ {
        // Attribute
        final ชนิดตัวแปร  ชื่อตัวแปรที่1 = ค่าเริ่มต้น;
        final ชนิดตัวแปร  ชื่อตัวแปรที่2 = ค่าเริ่มต้น;
        final ชนิดตัวแปร  ชื่อตัวแปรที่N = ค่าเริ่มต้น;

        // เมธอดต้นแบบ
        public returnType  เมธอดที่1();
        public returnType  เมธอดที่2();
}

class ชื่อคลาสลูก implements ชื่อคลาสแม่ {
        public returnType  เมธอดที่1() { ... }
        public returnType  เมธอดที่2() { ... }
}
```

หากมี attribute ต้องกำหนดค่าเริ่มต้น และกำหนดเป็น final (ค่าคงที่)

มีเพียงหัวของเมธอดเท่านั้น การทำงานจะถูกนิยามในคลาสลูก

**การสืบทอด Interface ใช้คำสั่ง implements ไม่ใช่ ~~extends~~**

# สัญลักษณ์ Interface



# ตัวอย่าง

```
public class TestApp {
  public static void main(String args[]) {
    // ประกาศ object ในชื่อคลาสแม่
    // แต่สร้าง object ในชื่อคลาสลูก
    Monster m1 = new FireMonster();
    Monster m2 = new WaterMonster();
    Monster m3 = new StoneMonster();

    // เรียกเมธอด attack()
    System.out.println(m1.attack()); // จาก FireMonster
    System.out.println(m2.attack()); // จาก WaterMonster
    System.out.println(m3.attack()); // จาก StoneMonster

    // ประกาศ และสร้าง object ของคลาสแม่
    Monster m4 = new Monster();
    System.out.println(m4.attack());
  }
}
```

Interface ไม่สามารถสร้าง object ได้

*ตัวอย่างหน้าจอ*

```
Attack with fire!
Attack with water!
Attack with stones!
```

**คลาสแม่**

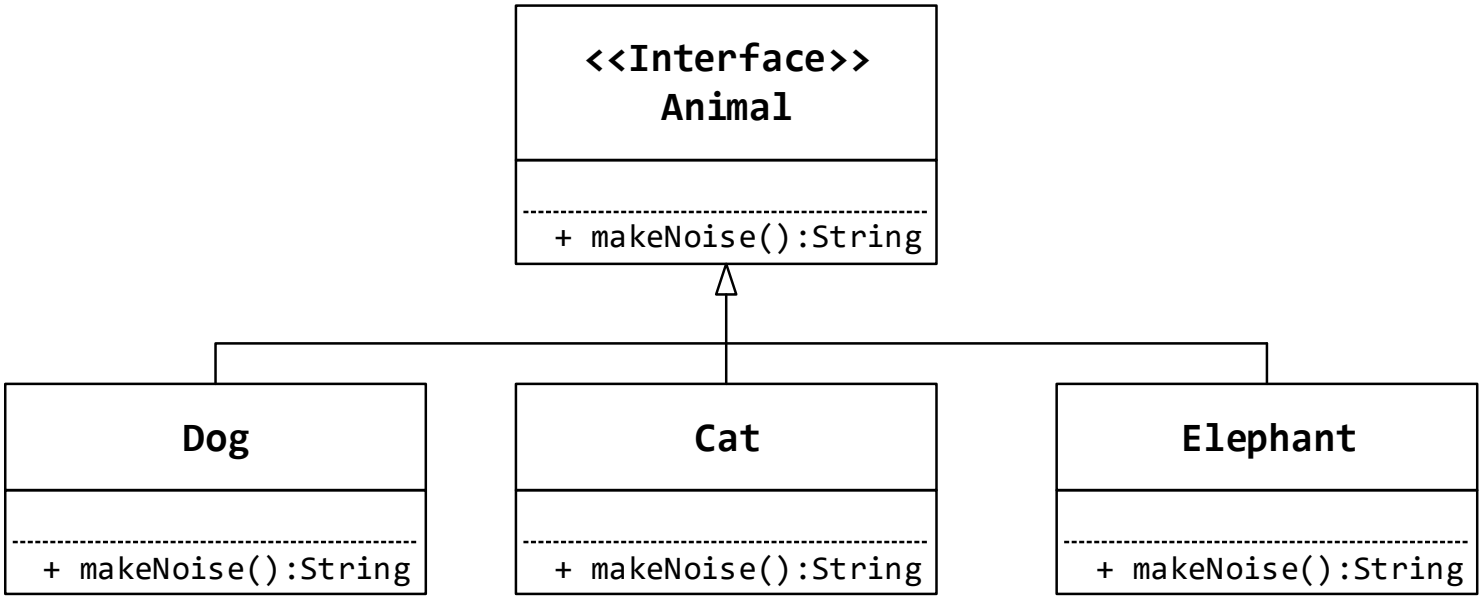
```
interface Monster {
   // นิยามพฤติกรรมให้แก่คลาสลูกที่จะมาสืบทอด
   public String attack() ;


}
```

**คลาสลูก**

```
class FireMonster implements Monster {
    public String attack() {
       return "Attack with fire!";
    }
}

class WaterMonster implements Monster {
    public String attack() {
       return "Attack with water!";
    }
}

class StoneMonster implements Monster {
    public String attack() {
       return "Attack with stones!";
    }
}
```

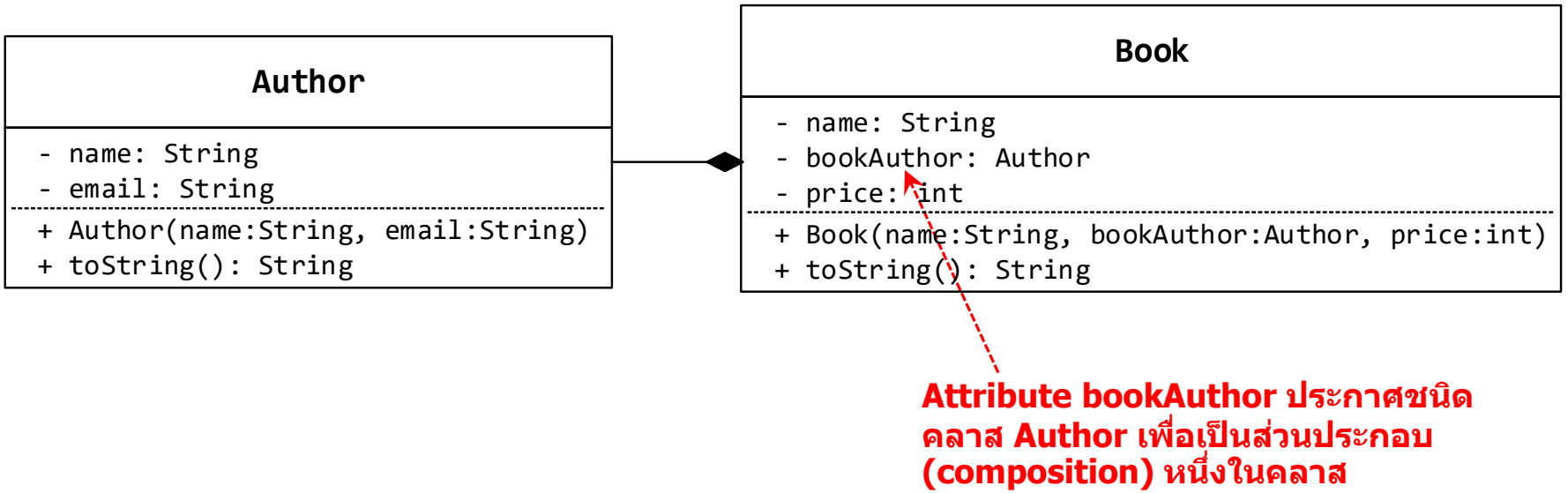
# กิจกรรม

- จงสร้าง interface Animal ที่มีการกำหนดให้คลาสลูกต้องสร้าง method ชื่อ makeNoise( ) โดยให้แสดงข้อความเสียงร้องของสัตว์ที่มีความแตกต่างกันตามชื่อคลาส



# Composition

- การ reuse คลาสที่มีอยู่แล้วมาใช้งาน โดยนำมาเป็น attribute หนึ่งของคลาสใหม่ เรียกว่า การประกอบคลาส หรือ Composition



```java
class Author {
   private String name;
   private String email;

   // Constructor
   public Author(String name, String email) {
      this.name = name;
      this.email = email;
   }

   public String toString() {
      return "[Author: " + name + " (" + email + ")]";
   }
}
```

```java
class Book {
   private String name;
   private Author bookAuthor;
   private int price;

   // Constructor
   public Book(String name, Author bookAuthor, int price) {
      this.name = name;
      this.bookAuthor = bookAuthor;
      this.price = price;
   }

   public String toString() {
      return name + " by " + bookAuthor
             + " - " + price + " baht";
   }
}
```

```java
public class TestApp {
   public static void main(String args[]) {
      // สร้าง object ของคลาส Author ก่อน
      Author john = new Author("John Smith", "smith@gmail.com");
      System.out.println(john);

      // สร้าง object ของ Book โดยกำหนด object ของคลาส Author ลงไปด้วย
      Book book1 = new Book("Java Tutor", john, 230);
      System.out.println(book1);

      // สร้าง object ของ Book โดยสร้าง object ของคลาส Author กำหนดลงไปด้วย
      Book book2 = new Book("Java EE", new Author("Robert", "rb@gmail.com"), 350);
      System.out.println(book2);
   }
}
```

*ตัวอย่างหน้าจอ*

```
[Author: John Smith (smith@gmail.com)]
Java Tutor by [Author: John Smith (smith@gmail.com)] - 230 baht
Java EE by [Author: Robert (rb@gmail.com)] - 350 baht
```

# กิจกรรม

- สร้างคลาส Point และนำมาประกอบเป็นส่วนหนึ่งของคลาส Line หลังจากนั้นให้สร้างคลาส TestLine เพื่อทดสอบการทำงาน