Week6

Programming Fundamentals II

Course Outline

1. P2J (Basic)
2. P2J (Control structures)
3. P2J (Collection types)
4. Classes and methods
5. Inheritance
6. Polymorphism
7. Interfaces
8. Testing and debugging
9. Events
10. UI programming
11. Exceptions
12. Generics
13. Concurrency
14. Team project

Ex3 Inheritance

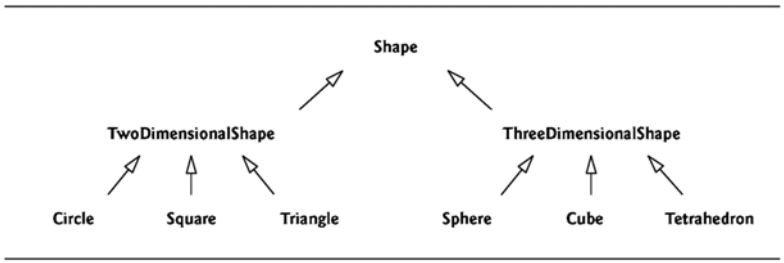


Fig. 9.3 | Inheritance hierarchy UML class diagram for Shapes.

Superclass

จาก Ex3 มาทำความรู้จักกับ Superclass Shape

รูปแบบ

```
[modifier] class SuperClassName {
        [AttributeName]
        [MethodName]
}
```

โดยที่

modifier	Keyword ที่กำหนดคุณสมบัติการเข้าถึง Class
SuperClassName	ชื่อของ Superclass
AttributeName	การประกาศ Attribute
MethodName	การประกาศ Method

Subclass

จาก Ex3 มาทำความรู้จักกับ Subclass TwoDimensionalShap

รูปแบบ

```
[modifier] class SubClassName extends SuperClassName {
        [AttributeName]
        [MethodName]
}
```

โดยที่

modifier	Keyword ที่กำหนดคุณสมบัติการเข้าถึง Class
SubClassName	ชื่อของ Subclass
SuperClassName	ชื่อของ Superclass
AttributeName	การประกาศ Attribute
MethodName	การประกาศ Method

protected

1. Subclass สามารถใช้ Method + Attribute ที่มีการเข้าใช้แบบ **protected** ได้
2. Class ที่อยู่ในแพ็คเกจเดียวกันก็สามารถใช้ Method + Attribute ที่มีการเข้าใช้แบบ **protected** ได้
3. การเข้าใช้แบบนี้มีความเข้มงวดน้อยกว่าแบบ **private** ซึ่งห้าม Class อื่นใดเข้าใช้แต่ก็อิสระน้อยกว่าแบบ **public** ที่ใครๆก็สามารถเข้าใช้ได้

Class Object

- All classes in Java inherit directly or indirectly from class `Object`, so its 11 methods are inherited by all other classes.
- Every array has an overridden `clone` method that copies the array.
  - If the array stores references to objects, the objects are not copied—a *shallow copy* is performed.

toString(), equals(), clone()

ข้อดี/เสียของการสืบทอด

ข้อดี	ข้อเสีย
■ การนำกลับมาใช้ใหม่	■ โปรแกรมทำงานช้าลง
■ ความเป็นมาตรฐานเดียวกัน	■ โปรแกรมมีขนาดใหญ่ขึ้น
■ เข้าใจสาระสำคัญได้ง่าย	■ ความซับซ้อนเพิ่มขึ้น

Polymorphism

Programming Fundamentals II

Week 6 **Polymorphism**

1. Intro Polymorphism
2. Polymorphism Example
3. Demostrating Polymorphic Behavior
4. Abstract Classes and Method
5. Case Study
6. Problem inheritance
7. Intro Interface
8. Abstract Class & Interface

Inheritance

Class นั้นสืบทอดมาจากอีก Class นึงโดยที่เราจะได้ Class ใหม่ที่มี Attribute และ Method ที่เหมือน Class แม่ทุกประการและเรายังสามารถเพิ่ม Attribute และ Method ได้อีกด้วย

```
2  class Shape
3  {
4      protected float height;
5      protected float width;
6      protected float radius;
7
8      protected float getHeight()
9      {
10         return this.height;
11     }
12 }
```

```
15 class TwoDimensionalShap extends Shape
16 {
17     protected float area;
18
19     protected float getArea()
20     {
21         return this.area;
22     }
23 }
```

```
25 class Triangle extends TwoDimensionalSha
26 {
27     float base;
28 }
```


Ex1 Inheritance

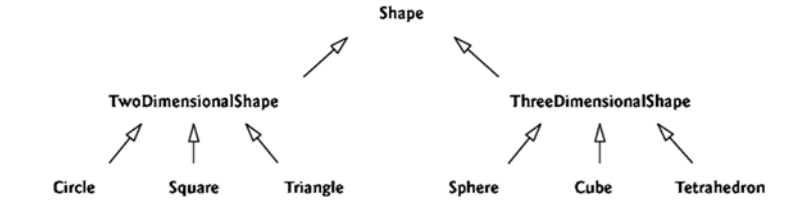


Fig. 9.3 | Inheritance hierarchy UML class diagram for Shapes.

OOP

1. Encapsulation (public, private, protect)
2. Inheritance (extend .....)

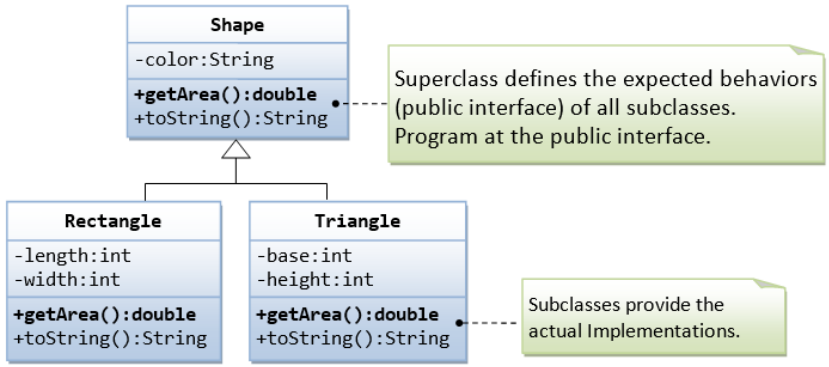
➡ 3. Polymorphism

Intro Polymorphism

- คุณสมบัติของ OOP ที่ Object ที่เกิดจาก Superclass เดียวกัน
- สามารถมีความสามารถเหมือน Superclass
- แต่อาจจะมี Method ที่ความสามารถไม่เหมือนกัน
- นั่นก็คือ Object จะสามารถมีลักษณะเฉพาะตัวได้

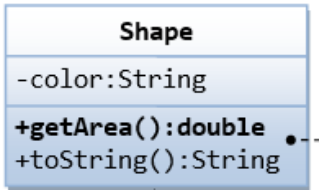
Intro Polymorphism

Shape ทุก Shape ควรจะมี getArea



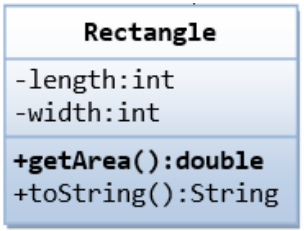
getArea แต่ละ Shape (3,4,5 เหลี่ยม) เหมือนกัน?

Superclass: **Shape**




```
3  // Superclass Shape
4  public class Shape
5  {
6     private String color;
7
8     // Constructor
9     public Shape (String color)
10    {
11       this.color = color;
12    }
13
14    @Override
15    public String toString()
16    {
17       return "Shape[color=" + color + "]";
18    }
19
20    // All shapes must have a method called getArea().
21    public double getArea()
22    {
23       System.err.println("Shape unknown! Cannot compute area!");
24       return 0;
25    }
26 }
```

Subclass: **Rectangle**



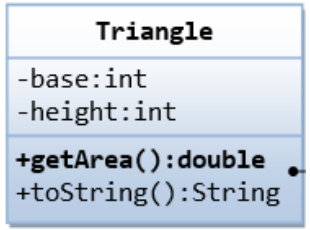

```java
// The Rectangle class, subclass of Shape
public class Rectangle extends Shape
{
    private int length;
    private int width;

    // Constructor
    public Rectangle(String color, int length, int width)
    {
        super(color);
        this.length = length;
        this.width = width;
    }

    @Override
    public String toString()
    {
        return "Rectangle[length=" + length + ",width=" + width + "," + super.toString() + "]";
    }

    @Override
    public double getArea()
    {
        return length*width;
    }
}
```


Subclass: **Triangle**




```java
// The Triangle class, subclass of Shape
public class Triangle extends Shape
{
    private int base;
    private int height;

    // Constructor
    public Triangle(String color, int base, int height)
    {
        super(color);
        this.base = base;
        this.height = height;
    }

    @Override
    public String toString()
    {
        return "Triangle[base=" + base + ",height=" + height + "," + super.toString() + "]";
    }

    @Override
    public double getArea()
    {
        return 0.5*base*height;
    }
}
```


Run: **ShapeTest**

```java
public class ShapeTest
{
    public static void main(String[] args)
    {
        Shape s1 = new Rectangle("red", 4, 5);      // create s1 Rectangle

        System.out.println(s1);                     // Run Rectangle's toString()
        System.out.println("Area: "+s1.getArea());  // Run Rectangle's getArea()

        Shape s2 = new Triangle("blue", 4, 5);      // create s2 triangle

        System.out.println(s2);                     // Run Triangle's toString()
        System.out.println("Area: "+s2.getArea());  // Run Triangle's getArea()
    }
}
```

```
Rectangle[length=4,width=5,Shape[color=red]]
Area: 20.0
Triangle[base=4,height=5,Shape[color=blue]]
Area: 10.0
```


Run: **ShapeTest**

```java
public class ShapeTest
{
    public static void main(String[] args)
    {
        Shape s1 = new Rectangle("red", 4, 5);      // create s1 Rectangle

        System.out.println(s1);                     // Run Rectangle's toString()
        System.out.println("Area: "+s1.getArea());  // Run Rectangle's getArea()

        Shape s2 = new Triangle("blue", 4, 5);      // create s2 triangle

        System.out.println(s2);                     // Run Triangle's toString()
        System.out.println("Area: "+s2.getArea());  // Run Triangle's getArea()
    }
}
```

Complier มอง Object Rectangle/Triangle เป็น Subclass ของ Shape ซึ่งก็คือ Object Shape เช่นกัน จึงสามารถประกาศชื่อตัวแปร Object ที่เป็น Object Superclass ได้เช่นกัน นี่คือความหลากหลายนั่นเอง

Run: **ShapeTest2**

```
3  public class ShapeTest2
4  {
5      public static void main(String[] args)
6      {
7          Rectangle s1 = new Rectangle("red", 4, 5);      // create s1 Rectangle
8
9          System.out.println(s1);                         // Run Rectangle's toString()
10         System.out.println("Area: "+s1.getArea());       // Run Rectangle's getArea()
11
12         Triangle s2 = new Triangle("blue", 4, 5);       // create s2 triangle
13
14         System.out.println(s2);                         // Run Triangle's toString()
15         System.out.println("Area: "+s2.getArea());       // Run Triangle's getArea()
16     }
17 }
```

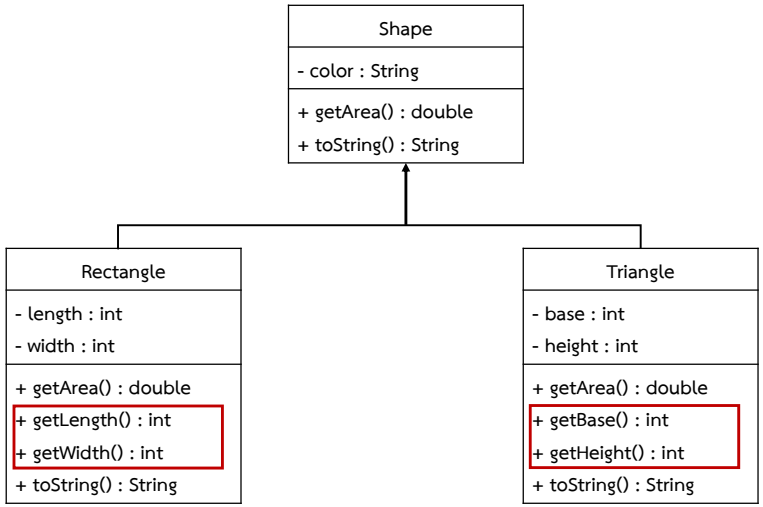
```
Rectangle[length=4,width=5,Shape[color=red]]
Area: 20.0
Triangle[base=4,height=5,Shape[color=blue]]
Area: 10.0
```


Run: **ShapeTest2**

```
3  public class ShapeTest2
4  {
5      public static void main(String[] args)
6      {
7          Shape s1 = new Rectangle("red", 4, 5);      // create s1 Rectangle
8
9          System.out.println(s1);                     // Run Rectangle's toString()
10         System.out.println("Area: "+s1.getArea());  // Run Rectangle's getArea()
11
12         Shape s2 = new Triangle("blue", 4, 5);      // create s2 triangle
13
14         System.out.println(s2);                     // Run Triangle's toString()
15         System.out.println("Area: "+s2.getArea());  // Run Triangle's getArea()
16
17         Shape s3 = new Shape("green");
18
19         System.out.println(s3);
20         System.out.println("Area: "+s3.getArea());  // Invalid output
21     }
22 }
```

```
Area: 20.0
Triangle[base=4,height=5,Shape[color=blue]]
Area: 10.0
Shape[color=green]
Area: 0.0
Shape unknown! Cannot compute area!
```


Polymorphism Example



Run: **ShapeTest3**

```
3  public class ShapeTest3
4  {
5      public static void main(String[] args)
6      {
7          Shape s1 = new Rectangle("red", 4, 5);      // create s1 Rectangle
8
9          System.out.println(s1);                     // Run Rectangle's toString()
10         System.out.println("Area: "+s1.getArea());  // Run Rectangle's getArea()
11
12         Shape s2 = new Triangle("blue", 4, 5);      // create s2 triangle
13
14         System.out.println(s2);                     // Run Triangle's toString()
15         System.out.println("Area: "+s2.getArea());  // Run Triangle's getArea()
16
17         Shape s3 = new Shape("green");
18
19         System.out.println(s3);
20         System.out.println("Base: "+s3.getLength());    // compile error
21         System.out.println("Base: "+s3.getWidth());     // compile error
22         System.out.println("Base: "+s3.getBase());      // compile error
23         System.out.println("Base: "+s3.getHeight());    // compile error
24     }
25 }
```

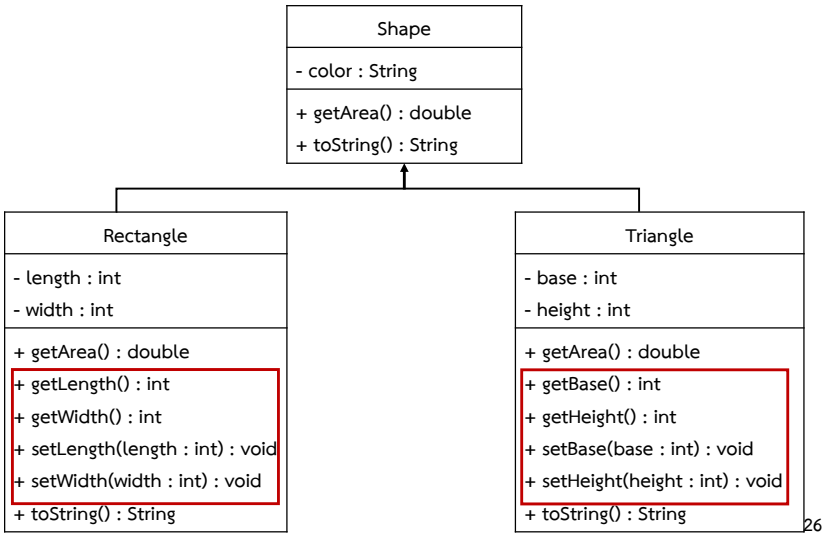
Polymorphism Example

ประโยชน์ Polymorphism ในการนำไปใช้

แก้ไข Class ให้มี Getter/Setter Method

1. Class Rectangle
2. Class Triangle

Polymorphism Example



Run: **ShapeTest4**

```
3  public class ShapeTest4
4  {
5      public static void main(String[] args)
6      {
7          Shape[] shapeList = new Shape[10];
8
9          Shape shape1 = new Shape();
10         shape1.setColor("Green");
11         shapeList[0] = shape1;
12
13         Rectangle shape2 = new Rectangle();
14         shape2.setColor("Navy");
15         shape2.setLength(10);
16         shape2.setWidth(5);
17         shapeList[1] = shape2;
18
19         Triangle shape3 = new Triangle();
20         shape3.setColor("Red");
21         shape3.setBase(3);
22         shape3.setHeight(2);
23         shapeList[2] = shape3;
24
25         for (int i = 0; i < shapeList.length; i++)
26         {
27             if (shapeList[i] != null)
28             {
29                 System.out.printf("\nList %d : Area = %.2f",i,shapeList[i].getArea());
30             }
31             else
32                 break;
33         }
34     }
35 }
```

```
Shape unknown! Cannot compute area!

List 0 : Area = 0.00
List 1 : Area = 50.00
List 2 : Area = 3.00
```


Run: **ShapeTest4**

```
3  public class ShapeTest4
4  {
5      public static void main(String[] args)
6      {
7          Shape[] shapeList = new Shape[10];
8
9          Shape shape1 = new Shape();
10         shape1.setColor("Green");
11         shapeList[0] = shape1;
12
13         Shape shape2 = new Rectangle();
14         shape2.setColor("Navy");
15         shape2.setLength(10);
16         shape2.setWidth(5);
17         shapeList[1] = shape2;
18
19         Shape shape3 = new Triangle();
20         shape3.setColor("Red");
21         shape3.setBase(3);
22         shape3.setHeight(2);
23         shapeList[2] = shape3;
24
25         for (int i = 0; i < shapeList.length; i++)
26         {
27             if (shapeList[i] != null)
28             {
29                 System.out.printf("\nList %d : Area = %.2f",i,shapeList[i].getArea());
30             }
31             else
32                 break;
33         }
34     }
35 }
```

Error!!!

Abstract

Classes & Method

Abstract

Classes & Method

Abstract Classes&Method

Abstract Class จะมีอย่างน้อยหนึ่ง Method เป็นประเภท Abstract Method

Abstract Method คือ Method ที่มีแต่ชื่อ ไม่มีการทำงานภายใน

Abstract Class สามารถมี state (พวก attribute หรือ instance variable ต่างๆ) ได้

Abstract Classes&Method

- Object จาก Class ประเภทนี้ จึงต้องนำไป extendสร้างเป็น Subclass
- เพิ่มการทำงานให้ Method ก่อน จึงจะสร้าง Object ได้ ซึ่งเรียกวิธีการดังกล่าวว่า การทำ **Override Method**
- Subclass ที่สืบทอดจาก Abstract Class จะต้องกำหนดการทำงานของ Abstract Method ให้ครบทุก Method ก่อน จึงจะนำไปใช้งานได้
- Abstract Class อนุญาตให้มี Method ที่การกำหนดการทำงานแล้วได้ เรียกว่า **Concrete Method**

Abstract Classes&Method

มี Keyword “abstract” อยู่ข้างหน้า class

รูปแบบ

```
abstract class AbstractClassName {
        [AbstractMethodNames]
        [ConcreteMethodNames]
}
```

โดยที่

AbstractClassName	ชื่อ Abstract Class
AbstractMethodNames	เป็นส่วนของการประกาศ Abstract Method
ConcreteMethodNames	เป็นส่วนของการประกาศ Concrete Method

Ex. Abstract Classes&Method

```java
// Superclass Shape
abstract class ShapeAb
{
   private String color;

   // Constructor
   public ShapeAb ()
   {
      this.color = "";
   }
   public ShapeAb (String color)
   {
      this.color = color;
   }

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

   @Override
   abstract public String toString();

   // All shapes must have a method called getArea().
   abstract public double getArea();

   // Rectangle
   abstract public int getLength();
   abstract public int getWidth();
   abstract public void setLength (int length);
   abstract public void setWidth (int width);

}
```


Abstract Classes&Method

การนำ Abstract Class ไปถ่ายทอดสร้างเป็น Subclass
ต้องมีการทำ Overriding Method มีรูปแบบดังนี้

รูปแบบ

```
class ClassName extends AbstractClassName {
        AbstractMethodNames() {
            [Statements];
        }
}
```

โดยที่

ClassName	ชื่อ Class
AbstractClassName	ชื่อ Abstract Class ที่ต้องการสืบทอด
AbstractMethodNames	ชื่อ Abstract Method ที่ต้องการทำ Override
Statements	statement ที่ override เมธอดใน Abstract Class

Ex. Abstract Classes&Method

```java
public class Rectangle extends ShapeAb
{
    private int length;
    private int width;
    // Constructor
    public Rectangle()
    {
        super("");
        this.length = 0;
        this.width = 0;
    }
    public Rectangle(String color, int length, int width)
    {
        super(color);
        this.length = length;
        this.width = width;
    }
    @Override
    public String toString() {
        return "Rectangle[length=" + length + ",width=" + width + "," + toString() + "]";
    }
    @Override
    public double getArea() {
        return length*width;
    }
    @Override
    public int getLength() {
        return this.length;
    }
    @Override
    public int getWidth() {
        return this.width;
    }
    @Override
    public void setLength(int length) {
        this.length = length;
    }
    @Override
    public void setWidth(int width) {
        this.width = width;
    }
}
```


Run: **ShapeTest5**

```java
public class ShapeTest5
{
    public static void main(String[] args)
    {
        ShapeAb[] shapeList = new ShapeAb[10];

        ShapeAb shape1 = new Rectangle();
        shape1.setColor("Navy");
        shape1.setLength(10);
        shape1.setWidth(5);
        shapeList[0] = shape1;

        ShapeAb shape2 = new Rectangle();
        shape2.setColor("Blue");
        shape2.setLength(20);
        shape2.setWidth(2);
        shapeList[1] = shape2;

        ShapeAb shape3 = new Rectangle();
        shape3.setColor("Red");
        shape3.setLength(5);
        shape3.setWidth(4);
        shapeList[2] = shape3;


        for (int i = 0; i < shapeList.length; i++)
        {
            if (shapeList[i] != null)
            {
                System.out.printf("\nList %d : Color = %4s, Area = %.2f",i,shapeList[i].getColor()
                        ,shapeList[i].getArea());
            }
            else
                break;
        }
    }
}
```

Triangle?

Abstract Classes&Method

Class ใดๆก็ตามจะสามารถ inherit
จาก abstract class ได้เพียงหนึ่งเท่านั้น

Java Compiler ไม่ อณุญาติให้ทำ Multiple inheritance
เหมือนภาษา Python แต่สามารถใช้การ **Interface** ได้

Triangle class ก็สามารถ inherit จาก ShapeAb
ก็ได้เช่นกัน แต่มันไม่เหมาะสม

Triangle extend ShapeAb

```
3  public class Triangle extends ShapeAb
4  {
5      @Override
6      public String toString() {
7          // TODO Auto-generated method stub
8          return null;
9      }
10
11     @Override
12     public double getArea() {
13         // TODO Auto-generated method stub
14         return 0;
15     }
16
17     @Override
18     public int getLength() {
19         // TODO Auto-generated method stub
20         return 0;
21     }
22
23     @Override
24     public int getWidth() {
25         // TODO Auto-generated method stub
26         return 0;
27     }
28
29     @Override
30     public void setLength(int length) {
31         // TODO Auto-generated method stub
32
33     }
34
35     @Override
36     public void setWidth(int width) {
37         // TODO Auto-generated method stub
38
39     }
40 }
```

Subclass ที่ inherit จาก Abstract Class จะต้องกำหนดการทำงานของ Abstract Method ให้ครบทุก Method ก่อน จึงจะนำไปใช้งานได้

Triangle class ของเราเมื่อ Inherit มาจาก ShapAb แล้วตัวมันเองจึงอุดมไปด้วยสารพัด Method จากคลาสแม่

Triangle extend ShapeAb

```
3  public class Triangle extends ShapeAb
4  {
5      @Override
6      public String toString() {
7          // TODO Auto-generated method stub
8          return null;
9      }
10
11     @Override
12     public double getArea() {
13         // TODO Auto-generated method stub
14         return 0;
15     }
16
17     @Override
18     public int getLength() {
19         // TODO Auto-generated method stub
20         return 0;
21     }
22
23     @Override
24     public int getWidth() {
25         // TODO Auto-generated method stub
26         return 0;
27     }
28
29     @Override
30     public void setLength(int length) {
31         // TODO Auto-generated method stub
32
33     }
34
35     @Override
36     public void setWidth(int width) {
37         // TODO Auto-generated method stub
38
39     }
40 }
```

แม้ว่าตัวมันเองจะไม่ได้ต้องการเลยก็ตาม การนำสองสิ่งที่ไม่สัมพันธ์กันมาสืบทอดกันจึงเป็นการผูกความสัมพันธ์ให้แน่นขึ้นและแยกออกจากกันยาก

** Abstract Class เหมาะกับการออกแบบ Class ที่มีลักษณะการทำงานเหมือนกัน

Abstract Classes&Method

Class ใดๆก็ตามจะสามารถ inherit
จาก abstract class ได้เพียงหนึ่งเท่านั้น

Triangle class ก็สามารถ inherit จาก ShapeAb
ก็ได้เช่นกัน แต่มันไม่เหมาะสม เป็นปัญหาหนึ่งของ Inheritance

Problem

Inheritance

Relationship

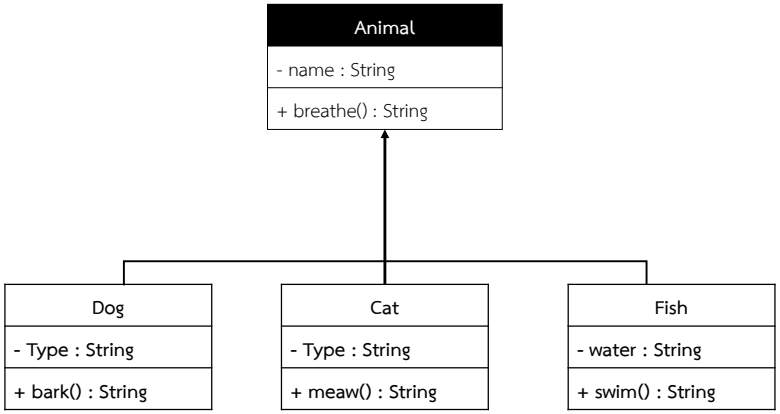
abstract class

Is-a relationship สิ่งหนึ่งเป็นอีกสิ่ง — extend

interface

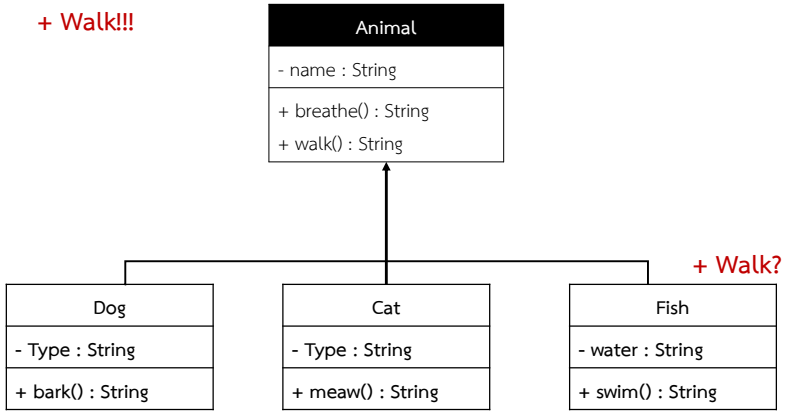
Has-a relationship สิ่งหนึ่งมีอีกสิ่งหนึ่ง — implement

Problem1

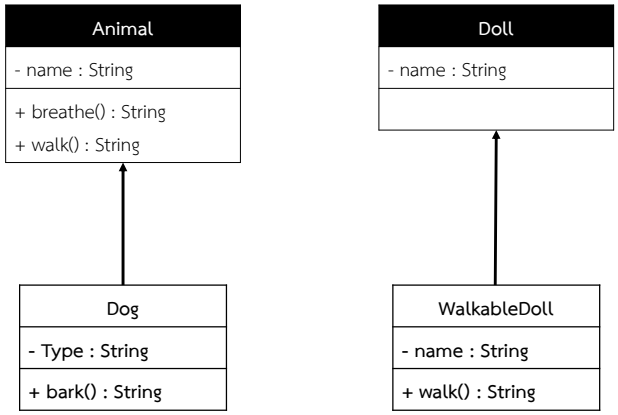


Problem1

+ Walk!!!

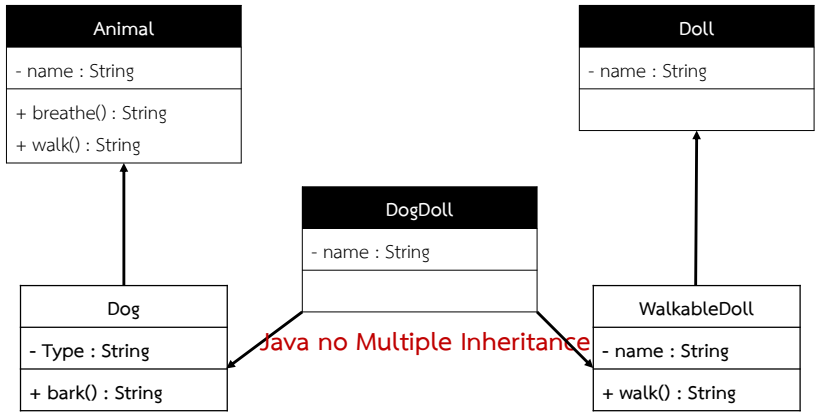


Problem2



Problem2

ต้องการสร้าง Dog Doll ที่มีความสามารถในการ Walk + Bark



Intro Interface

Interface

เป็น Class ที่มีทุก Method เป็น Abstract Method ซึ่งกำหนดเพียงว่ามีการรับค่าด้วยข้อมูลชนิดใด และคืนค่าข้อมูลเป็นข้อมูลชนิดใด

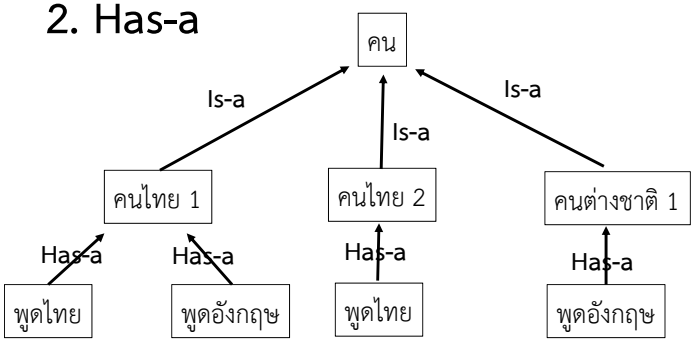
เปรียบเสมือนข้อตกลงที่มีไว้ให้ Object ติดต่อสื่อสารถึงกันได้นั้นเอง

Ex. คนไทย, คนต่างชาติ, สัญญาณไฟจราจร

Relationship

1. Is-a

2. Has-a



Interface

การสร้างและใช้งาน Interface

มี Keyword “interface” อยู่ข้างหน้า class มีรูปแบบ ดังนี้

```
[modifier]  interface interfaceName {

}
```

โดยที่ modifier Keyword ที่กำหนดคุณสมบัติการเข้าถึง Class

interfaceName ชื่อของ Interface Class

Interface

การใช้งาน Interface

1. ต้อง implements อินเตอร์เฟสเพื่อกำหนดหน้าที่การทำงานให้ Abstract Method

2. ต้องมีการทำ Overriding Method ก่อนเสมอ เพราะ Interface ประกอบด้วย Abstract Method ซึ่งรูปแบบการใช้งาน Interface (implements)

Interface

การใช้งาน Interface

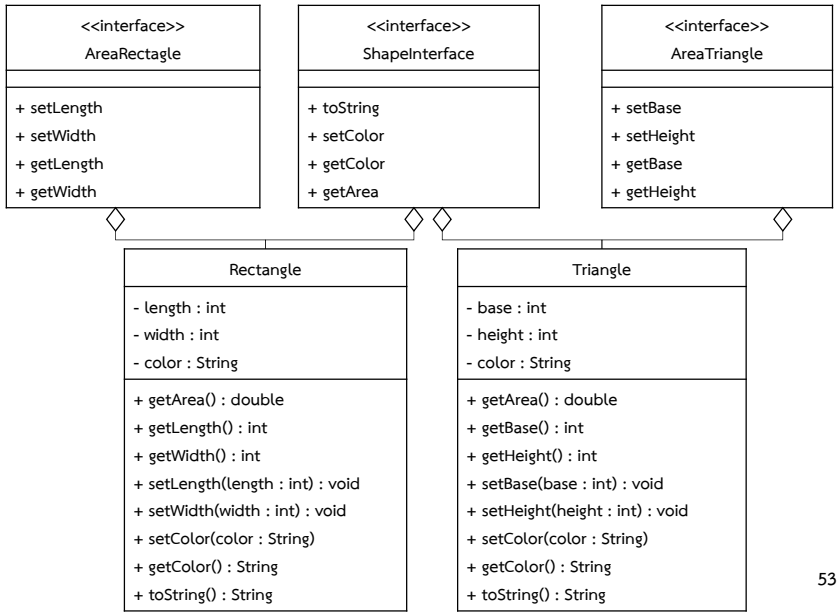
รูปแบบ

```
[modifier]  class ClassName implements InterfaceName {
            AbstractMethodNames() {
                [Statements];
            }
}
```

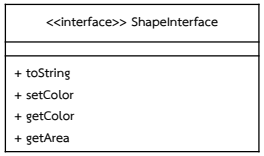
โดยที่

modifier	Keyword ที่กำหนดคุณสมบัติการเข้าถึง Class
ClassName	ชื่อ Class
interfaceName	ชื่อ interface ที่ต้องการใช้งาน
AbstractMethodNames	Abstract Method ที่ต้องการทำ Override
Statements	เป็นส่วนของชุดคำสั่งที่ทำ Overriding Method

Interface Example

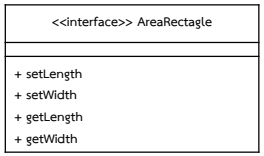


Interface Example

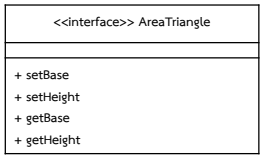



```java
public interface ShapeInterface
{
    public String toString();
    public void setColor (String color) ;
    public String getColor();
    public double getArea();

}
```

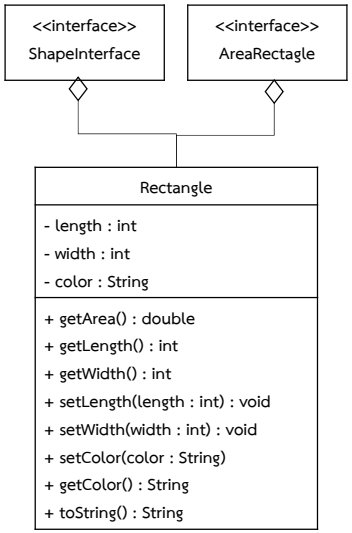



```java
public interface AreaRectagle
{
    public void setLength (int length);
    public void setWidth (int width);
    public int getLength();
    public int getWidth();
}
```



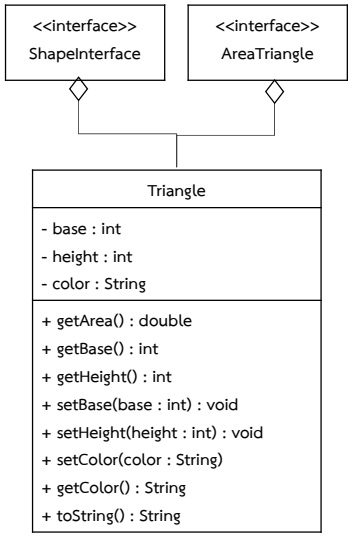

```java
public interface AreaTriangle
{
    public void setBase (int base);
    public void setHeight (int height);
    public int getBase();
    public int getHeight();
}
```


Interface Example




```java
public class Rectangle implements ShapeInterface,AreaRectagle
{
    @Override
    public void setLength(int length) {
        // TODO Auto-generated method stub
    }
    @Override
    public void setWidth(int width) {
        // TODO Auto-generated method stub
    }
    @Override
    public int getLength() {
        // TODO Auto-generated method stub
        return 0;
    }
    @Override
    public int getWidth() {
        // TODO Auto-generated method stub
        return 0;
    }
    @Override
    public void setColor(String color) {
        // TODO Auto-generated method stub
    }
    @Override
    public String getColor() {
        // TODO Auto-generated method stub
        return null;
    }
    @Override
    public double getArea() {
        // TODO Auto-generated method stub
        return 0;
    }
}
```


Interface Example




```java
public class Triangle implements ShapeInterface, AreaTriangle
{
    @Override
    public void setBase(int base) {
        // TODO Auto-generated method stub
    }
    @Override
    public void setHeight(int height) {
        // TODO Auto-generated method stub
    }
    @Override
    public int getBase() {
        // TODO Auto-generated method stub
        return 0;
    }
    @Override
    public int getHeight() {
        // TODO Auto-generated method stub
        return 0;
    }
    @Override
    public void setColor(String color) {
        // TODO Auto-generated method stub
    }
    @Override
    public String getColor() {
        // TODO Auto-generated method stub
        return null;
    }
    @Override
    public double getArea() {
        // TODO Auto-generated method stub
        return 0;
    }
}
```

Abstract Class & Interface

- Abstract Class ประกอบด้วย Method 2 แบบ คือ Concrete Method และ Abstract Metthod เรานำ Abstract Class มาใช้งานจะมีบาง Method ที่สามารถกำหนดการทำงานใน Subclass ภายหลังได้
- Interface เป็น Class ที่ยังไม่กำหนดการทำงาน เมื่อเรานำ Interface มาใช้งานใน Class เราจะต้องกำหนดการทำงานให้กับ Method ใน Interface นั้นก่อนเสมอ เรียกว่า เป็นการทำ implements interface
- Class จะสืบทอด Abstract Class ได้เพียง Class เดียวเท่านั้น
- Class จะใช้งาน Interface ได้มากกว่าหนึ่ง Interface เราจึงใช้ Interface สร้าง **Multiple Inheritance** ได้

Full Interface

Next week