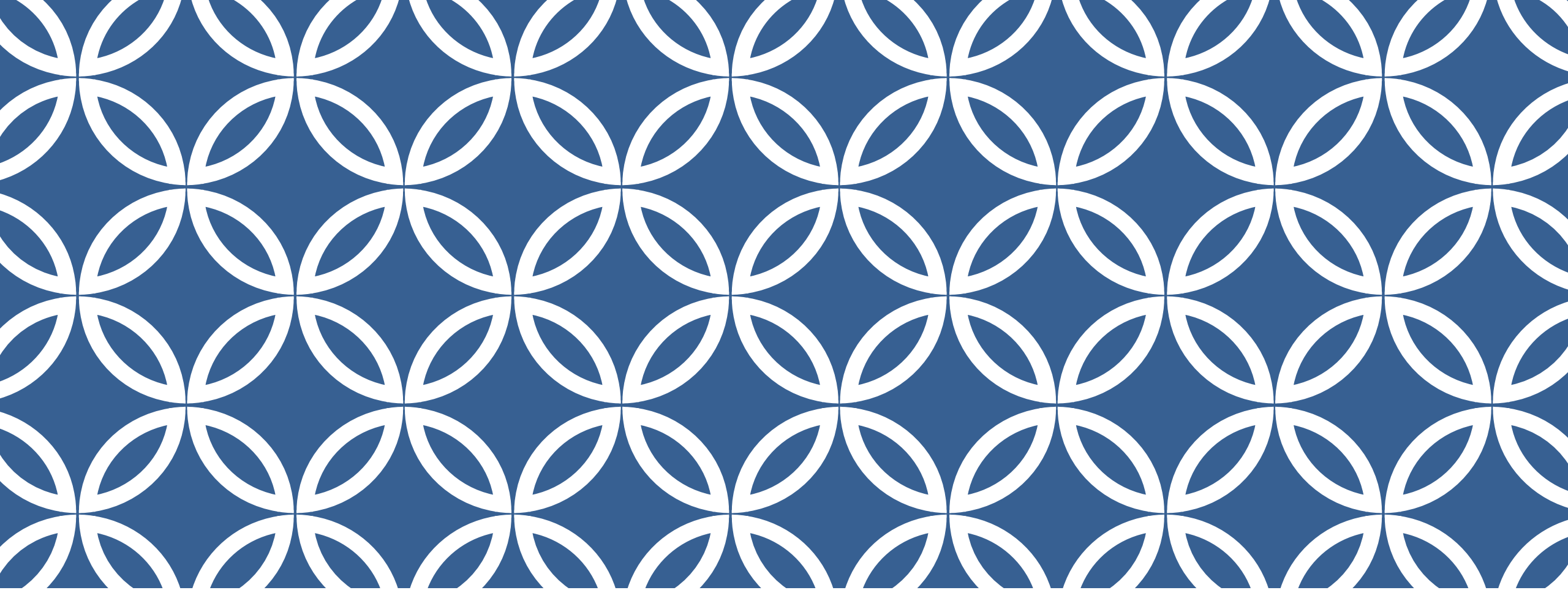


บทที่ 5
POLYMORPHISM

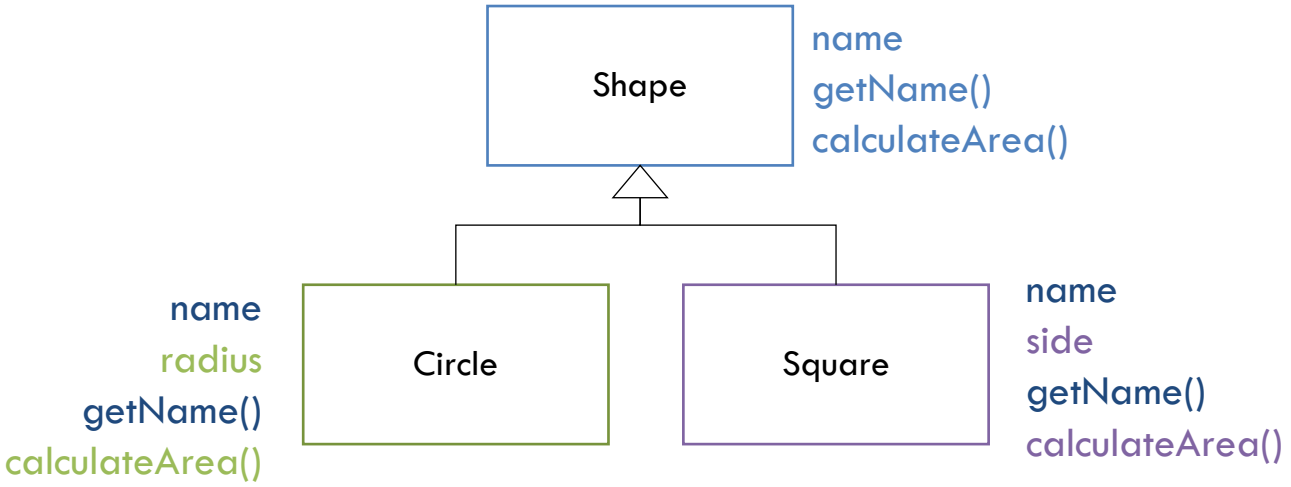
อ.สกรณ์ บุษบง
สาขาวิทยาการคอมพิวเตอร์ คณะวิทยาศาสตร์
มหาวิทยาลัยราชภัฏบุรีรัมย์

OVERVIEW

- Static binding
- Dynamic binding
- Operation overloading
- Polymorphism

STATIC BINDING

- ในบทที่ผ่านมา เราได้พบกับ **class Shape** ไปแล้ว



STATIC BINDING

- ตอนนี้มาลองแปลงโครงสร้างการเขียน code ของ class Shape แต่เขียนในแบบ procedural programming
- โดยการจำลอง pseudo-code ได้ดังนี้

STATIC BINDING

```
Data Section
Type
  Circle   = Record
               String name;
               int radius;
             End
  Square = Record
               String name;
               int side;
             End
```

```
Variable
shapeArray : Array [1..5] of char;
  c : Circle;
  s : Square;
Main Code Section
  c.name = "Circle C";
  c.radius = 2;
  s.name = "Square S";
  s.side = 3;
  A[1] = 'c';
  A[2] = 's';
  For int i = 1 to 2 do {
    Switch shapeArray[i]
      'c' : calculateCircleArea();
      's' : calculateSquareArea();
      'n' : do nothing;
    End (Case)
  }
```

```
Routine Section
calculateCircleArea() {
  float area = 3.14 * c.radius * c.radius;
  writeln ("The area of ", c.name, " is ", area, " sq. cm.");
}
calculateSquareArea() {
  float area = s.side * s.side;
  writeln ("The area of ", s.name, " is ", area, " sq. cm.");
}
```

STATIC BINDING

- เพิ่ม ส่วนของ Triangle เข้าไปในโครงสร้างได้ดังนี้

```
Data Section
Type
  Circle   = Record
               String name;
               int radius;
             End
  Square = Record
               String name;
               int side;
             End
  Triangle = Record
               String name;
               Int base;
               Int height;
             End
```

```
Variable
shapeArray : Array [1..5] of char;
  c : Circle;
  s : Square;
  t : Triangle;
Main Code Section
  c.name = "Circle C";
  c.radius = 2;
  s.name = "Square S";
  s.side = 3;
  t.name = "Trianlge T";
  t.base = 4; t.height = 5;
  A[1] = 'c';
  A[2] = 's';
  A[3] = 't';
  For int i = 1 to 2 do {
    Switch shapeArray[i]
      'c' : calculateCircleArea();
      's' : calculateSquareArea();
      't' : calculateTriangleArea();
      'n' : do nothing;
    End (Case)
  }
```

```
Routine Section
calculateCircleArea() {
  float area = 3.14 * c.radius * c.radius;
  writeln ("The area of ", c.name, " is ", area, " sq. cm.");
}
calculateSquareArea() {
  float area = s.side * s.side;
  writeln ("The area of ", s.name, " is ", area, " sq. cm.");
}
calculateTriangleArea() {
  float area = 0.5f * t.base * t.height;
  writeln ("The area of ", t.name, " is ", area, " sq. cm.");
}
```

DYNAMIC BINDING

- จาก Static Binding ทำให้เกิดความยุ่งยากในการจัดการ code
- จึงได้มีการออกแบบ Dynamic Binding

DYNAMIC BINDING

```
public class Shape {
    private String name;
    public Shape(String aName) {
        name=aName;
    }
    public String getName() {
        return name;
    }
    public float calculateArea() {
        return 0.0f;
    }
}
```

```
public class Square extends Shape{
    private int side;
    public Square(String aName) {
        super(aName);
        side = 3;
    }
    public float calculateArea() {
        int area;
        area = side * side;
        return area;
    }
}
```

```
public class Circle extends Shape {
    private int radius;
    public Circle(String aName) {
        super(aName);
        radius = 3;
    }
    public float calculateArea() {
        float area;
        area = (float) (3.14 * radius * radius);
        return area;
    }
}
```

```
public class Triangle extends Shape {
    private int base, height;
    Triangle(String aName) {
        super(aName);
        base = 4;
        height = 5;
    }
    public float calculateArea() {
        float area = 0.5f * base * height;
        return area;
    }
}
```

```
public class Main {
    public static void main(String[] args) {
        Circle c = new Circle("Circle C");
        Square s = new Square("Square S");
        Triangle t = new Triangle("Triangle T");
        Shape shapeArray[] = {c, s, t};
        for (int i=0; i<shapeArray.length; i++) {
            System.out.print("The area of " + shapeArray[i].getName() + " is " +
              shapeArray[i].calculateArea()+ " sq. cm.\n");
        }
    }
}
```

OPERATION OVERLOADING

- Class Circle และ class Shape ต่างก็ประกาศ calculateArea() ของตนเอง
- ถึงแม้ว่า method ทั้ง 2 จะมี method signature ที่เหมือนกัน
- แต่มี implementation ที่ต่างกัน เนื่องจากสูตรในการคำนวณของแต่ละรูปทรางต่างกัน

OPERATION OVERLOADING

- เป็นไปไม่ได้เลยที่จะสร้าง **method** ที่มี **method name** เหมือนกัน แต่ **Implementation** ที่ต่างกัน ในการเขียนโปรแกรมภาษาเดิมๆ
- แต่สามารถทำได้ใน OOP
- ความสามารถนี้เป็นการใช้ **method name** แต่มีการ **implementation** ที่ต่างกันได้มากกว่า 1 **method**
- เราเรียกความสามารถนี้ว่า Operation Overloading

OPERATION OVERLOADING >SAME METHOD SIGNATURE

- Method 2 method จะถูกเรียกว่ามี Signatureที่เหมือนกัน (same method signature) ก็ต่อเมื่อ
  - the name of the methods are the same; and
  - the number and type of formal parameters are the same.

OPERATION OVERLOADING >SAME METHOD SIGNATURE

```
public class Shape {
    private String name;
    public Shape(String aName) {
        name=aName;
    }
    public String getName() {
        return name;
    }
    public float calculateArea() {
        return 0.0f;
    }
}
```

```
public class Square extends Shape{
    private int side;
    public Square(String aName) {
        super(aName);
        side = 3;
    }
    public float calculateArea() {
        int area;
        area = side * side;
        return area;
    }
}
```

```
public class Circle extends Shape {
    private int radius;
    public Circle(String aName) {
        super(aName);
        radius = 3;
    }
    public float calculateArea() {
        float area;
        area = (float) (3.14 * radius * radius);
        return area;
    }
}
```

```
public class Triangle extends Shape {
    private int base, height;
    Triangle(String aName) {
        super(aName);
        base = 4;
        height = 5;
    }
    public float calculateArea() {
        float area = 0.5f * base * height;
        return area;
    }
}
```

```
public class Main {
    public static void main(String[] args) {
        Circle c = new Circle("Circle C");
        Square s = new Square("Square S");
        Triangle t = new Triangle("Triangle T");
        Shape shapeArray[] = {c, s, t};
        for (int i=0; i<shapeArray.length; i++) {
            System.out.print("The area of " + shapeArray[i].getName() + " is " +
              shapeArray[i].calculateArea()+ " sq. cm.\n");
        }
    }
}
```

OPERATION OVERLOADING >OVERLOADING METHOD NAMES

```
class A {
    ...
    A() { ... }
    A(int x) { ... }
    A(int x, int y) { ... }
    A(String x) { ... }
    ...
}
```

- Method A(int x), A(int x, int y), A(String s) ได้ overloading method A()
- Method เหล่านี้จะถูกแยกแยะตอน compile ด้วย จำนวน และ type of parameters

OPERATION OVERLOADING >OVERLOADING METHOD NAMES

- A thisA = new A(); → A()
- A thisA = new A(3); → A(int x)
- A thisA = new A(4, 5); → A(int x, int y)
- A thisA = new A("hello"); → A(String x)

OPERATION OVERLOADING >OVERLOADING METHOD NAMES

- ซ้าย ขวา อันไหน **compile** ผ่าน

```
public class A {
    public A(){}
    public void a1(){}
    public void a1(){}
}
```

```
public class A {
    public A(){}
    public void a1(){}
    public int a1(){return 5;}
}
```

OPERATION OVERLOADING >OVERLOADING METHOD NAMES

- อย่างไรก็ตาม การประกาศ **method** ที่ **Same Method Signature** แต่อยู่ใน **class** ที่ต่างกันสามารถทำได้ใน OOP
- ให้นักศึกษา **implement code**

```
public class A {
    public A(){}
    public void a1(){}
}
```

```
public class B {
    public B(){}
    public void a1(){}
    public void b1(){}
}
```

OPERATION OVERLOADING >OVERLOADING METHOD NAMES

- Implement code

Redefine = Override

```
public class C {
    public C(){}
    public void c1(){
        System.out.println("C.c1()");
    }
}
```

```
public class D extends C {
    public D(){}
    public void c1(){
        super.c1();
        System.out.println("D.c1()");
    }
    public void d1(){}
}
```

```
public static void main(String[] args) {
    D thisD = new D();
    thisD.c1();
}
```

C.c1()
D.c1()

OPERATION OVERLOADING >OVERLOADING METHOD NAMES

- Redefine = Override
- เกิดจากการประกาศ **method** ที่สืบทอดมาจาก **superclass** ซ้ำเพื่อกำหนดการทำงานที่ต่างจากเดิม

OPERATION OVERLOADING >OVERLOADING METHOD NAMES

- Implement code

```
public static void main(String[] args) {
    D thisD = new D();
    thisD.c1();
    thisD.c1(3);
}
```

Overloading

```
public class D extends C {
    public D(){}
    public void c1(int i){
        super.c1();
        System.out.println("D.c1()");
    }
    public void d1(){}
}
```

```
C.c1()
C.c1()
D.c1()
```

POLYMORPHISM

- ในช่วงต้น **slide** เราได้กล่าวถึง **Dynamic binding**
- ยกตัวอย่าง กลุ่มของ **class shape**

POLYMORPHISM

```
public class Shape {
    private String name;
    public Shape(String aName) {
        name=aName;
    }
    public String getName() {
        return name;
    }
    public float calculateArea() {
        return 0.0f;
    }
}
```

```
public class Square extends Shape{
    private int side;
    public Square(String aName) {
        super(aName);
        side = 3;
    }
    public float calculateArea() {
        int area;
        area = side * side;
        return area;
    }
}
```

```
public class Circle extends Shape {
    private int radius;
    public Circle(String aName) {
        super(aName);
        radius = 3;
    }
    public float calculateArea() {
        float area;
        area = (float) (3.14 * radius * radius);
        return area;
    }
}
```

```
public class Triangle extends Shape {
    private int base, height;
    Triangle(String aName) {
        super(aName);
        base = 4;
        height = 5;
    }
    public float calculateArea() {
        float area = 0.5f * base * height;
        return area;
    }
}
```

```
public class Main {
    public static void main(String[] args) {
        Circle c = new Circle("Circle C");
        Square s = new Square("Square S");
        Triangle t = new Triangle("Triangle T");
        Shape shapeArray[] = {c, s, t};
        for (int i=0; i<shapeArray.length; i++) {
            System.out.print("The area of " + shapeArray[i].getName() + " is " +
              shapeArray[i].calculateArea()+ " sq. cm.\n");
        }
    }
}
```

POLYMORPHISM

- message (calculateArea()) ถูกส่งจาก sender (main()) ไปยัง receiver
- class circle เมื่อได้รับ message ก็จะทำงานตาม method calculateArea() ของตนเพื่อคำนวณหา area
- Class square เมื่อได้รับ message ก็จะทำงานตาม method calculateArea() ของตนเพื่อคำนวณหา area เช่นกัน

POLYMORPHISM

- ความสามารถของ Object ที่แตกต่างกัน ทำงานกับ **method** ของตนเพื่อตอบสนอง Message เดียวกัน เรียกความสามารถนี้ว่า Polymorphism

POLYMORPHISM > SELECTION OF METHOD

- **Polymorphism** การตีความ **message** นั้นไม่ขึ้นอยู่กับ **sender**
- **Sender** รู้เพียงว่า **object** สามารถตอบสนองต่อ **message** ได้หรือไม่
- แต่ไม่รู้ว่า **object** เป็นของ **class** อะไร หรือตอบสนองต่อ **message** ยังไง

POLYMORPHISM
> SELECTION OF METHOD

- เช่น message shapeArray[i].calculateArea() จาก method main() ส่งไปที่ Shape object (a circle or a square)
- sender (main()) จะไม่รู้ว่า Shape objects จะตอบสนองต่อ message อย่างไร

POLYMORPHISM
> SELECTION OF METHOD

- การเลือก **method** ที่เหมาะสมเป็นหน้าที่ของ **class** ของ object
- เช่น **object circle** จะเรียก **calculateArea()** จากนั้น **object square** ก็จะเรียก **calculateArea()** เนื่องจาก **shapeArray** ตัวแรกเป็น **circle, square, triangle**

POLYMORPHISM
> SELECTION OF METHOD

- Method calculateArea() ของ class Circle, Square, Triangle จะถูกเรียกว่า Polymorphic Method

POLYMORPHISM > INCREMENTAL DEVELOPMENT

- **Polymorphism** ถูกสนับสนุนจาก **dynamic binding** และ ความสามารถในการใช้ชื่อของ **method** ที่เหมือนกัน ใน **class** ที่ต่างกัน
- **Polymorphism** จะช่วยให้การเขียนโปรแกรมเน้นไปที่
  - **what method should happen** มากกว่า
  - how it should happen

POLYMORPHISM
> INCREMENTAL DEVELOPMENT

- วิธีการนี้ช่วยให้เกิดความยืดหยุ่นในการออกแบบ Code และช่วยในการพัฒนาโปรแกรม
- ดังนี้

POLYMORPHISM > INCREMENTAL DEVELOPMENT

- Code จาก บทที่ 5 part 2

```
public class Shape {
    private String name;
    public Shape(String aName) {
        name=aName;
    }
    public String getName() {
        return name;
    }
    public float calculateArea() {
        return 0.0f;
    }
}
```

```
public class Square extends Shape{
    private int side;
    public Square(String aName) {
        super(aName);
        side = 3;
    }
    public float calculateArea() {
        int area;
        area = side * side;
        return area;
    }
}
```

```
public class Circle extends Shape {
    private int radius;
    public Circle(String aName) {
        super(aName);
        radius = 3;
    }
    public float calculateArea() {
        float area;
        area = (float) (3.14 * radius * radius);
        return area;
    }
}
```

```
public class Main {
    public static void main(String[] args) {
        Circle c = new Circle("Circle C");
        Square s = new Square("Square S");
        Shape shapeArray[] = {c, s};
        for (int i=0; i<shapeArray.length; i++) {
            System.out.print("The area of " + shapeArray[i].getName() + " is " +
             shapeArray[i].calculateArea()+ " sq. cm.\n");
        }
    }
}
```

POLYMORPHISM
> INCREMENTAL DEVELOPMENT

- จากนั้นเพิ่ม class

```
public class Triangle extends Shape {
    private int base, height;
    Triangle(String aName) {
        super(aName);
        base = 4;
        height = 5;
    }
    public float calculateArea() {
        float area = 0.5f * base * height;
        return area;
    }
}
```

```
public class Main {
    public static void main(String[] args) {
        Circle c = new Circle("Circle C");
        Square s = new Square("Square S");
        Triangle t = new Triangle("Triangle T");
        Shape shapeArray[] = {c, s, t};
        for (int i=0; i<shapeArray.length; i++) {
            System.out.print("The area of " + shapeArray[i].getName() + " is " +
              shapeArray[i].calculateArea()+ " sq. cm.\n");
        }
    }
}
```

POLYMORPHISM
> INCREMENTAL DEVELOPMENT

- จากการเพิ่ม **class Triangle** พบว่า
  - ไม่ส่งผลกระทบต่อ **class** อื่นๆในโครงสร้าง
  - เพิ่ม **code** ใน **main()** เพื่อสร้าง **object Triangle**
  - ใน **main()** ได้เพิ่ม **object Triangle** ใน **shapeArray**
- ไม่มีการแก้ไขในส่วน **println** ใน **main()**
- ไม่ต้องการ **switch statement** เพื่อระบุ **method**

POLYMORPHISM
> INCREASED CODE READABILITY

- **Polymorphism** ช่วยเพิ่ม **Code Readability** เนื่องจากมีการใช้งาน **message** เดียวที่สามารถเรียกใช้ **object** ที่ต่างการในการทำงานให้เหมาะสม

SUMMARY

- Static binding—the binding of variables to operations at compile time;
- Dynamic binding—the binding of variables to operations at run time;
- Operation overloading—the ability to use the same name for two or more methods in a class
- Polymorphism—the ability of different objects to perform the appropriate method in response to the same message.

THANK