พหุลักษณ์ (Polymorphism)

Benjamas Panyangam
Matinee Kiewkanya
Computer Science, CMU

พหุลักษณ์ (Polymorphism)

พหุลักษณ์หรือการพ้องรูป แปลมาจากคำว่า **"Polymorphism"** ในภาษากรีก ซึ่งหมายถึง มีหลายรูปแบบ **(Many shapes)**

ในการเขียนโปรแกรมเชิงวัตถุ พหุลักษณ์ คือ ภาวะที่เมท็อดมีการทำงานได้หลายรูปแบบ ถึงแม้ว่าจะมีการเรียกใช้เมท็อดเดียวกัน แต่ผลลัพธ์จากการทำงานของเมท็อดจะแตกต่างกันออกไป

Binding

ในการทำงานของโปรแกรมที่มีเมท็อดชื่อซ้ำกัน การจะเลือกว่าจะเรียกใช้เมท็อดใด มี 2 วิธี คือ

1) **Early Binding หรือ Static Binding**
   คือ การเลือกเมท็อดตอนแปลโปรแกรม (Compile Time)

เช่น เมื่อคลาสมีการสร้างเมท็อดพ้องชื่อ (Overloaded Method) ซึ่งเป็นเมท็อดชื่อเดียวกัน แต่มี Method Signature ต่างกัน เช่น

- void calArea(int);
- void calArea(int, int);
- void calArea(string, int);

เมื่อวัตถุมีการเรียกใช้เมท็อด calArea() การพิจารณาว่าจะใช้เมท็อดใด จะพิจารณาจากอาร์กิวเมนต์ที่ส่งมา
ดังตัวอย่าง

- myObj.calArea(5); จะเรียกใช้ void calArea(int);
- myObj.calArea(5,10); จะเรียกใช้ void calArea(int, int);
- myObj.calArea("circle",5); จะเรียกใช้ void calArea(string, int);

ซึ่งการพิจารณาว่าจะใช้เมท็อดใดสามารถเกิดขึ้นได้ในขณะแปลโปรแกรม เนื่องจากมีความแตกต่างในจำนวนและชนิดของพารามิเตอร์ของเมท็อด หรืออาจกล่าวได้ว่า Overloaded Method จะก่อให้เกิด Polymorphism ในช่วงแปลโปรแกรม นั่นเอง

2) Late Binding หรือ Dynamic Binding

คือ การเลือกเมท็อดในขณะที่โปรแกรมกำลังดำเนินงาน (Run Time) โดยการหาวัตถุเพื่อกำหนดเมท็อดที่จะทำงานให้เหมาะสม
ตัวอย่างของ Late Binding สามารถพบได้ในการทำงานของเมท็อดเสมือน (Virtual Method)

หรืออาจกล่าวได้ว่า Virtual Method จะก่อให้เกิด Polymorphism ในขณะที่โปรแกรมดำเนินงานนั่นเอง
อย่างไรก็ตามในบางตำราจะถือว่า Polymorphism เกี่ยวข้องกับ Late Binding เท่านั้น

Polymorphism

- รูปแบบคำสั่งสำหรับประกาศออบเจ็กต์ใดๆ ให้เป็นออบเจ็กต์ของคลาสแม่

  SuperClassName objName;

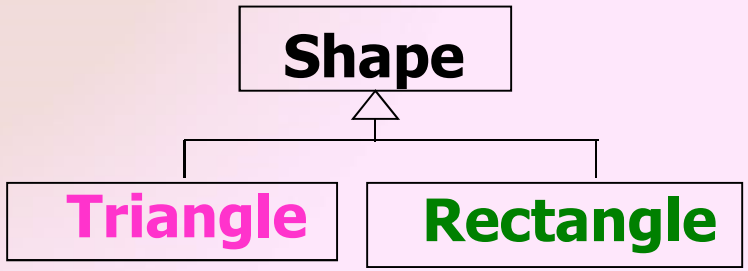
- ใช้หลักการของ Polymorphism ทำให้สร้างออบเจ็กต์ได้หลายรูปแบบ โดยใช้รูปแบบคำสั่งดังนี้

  objName = new ***SuperClassName***();

  หรือ ***objName*** = new ***SubClassName***();

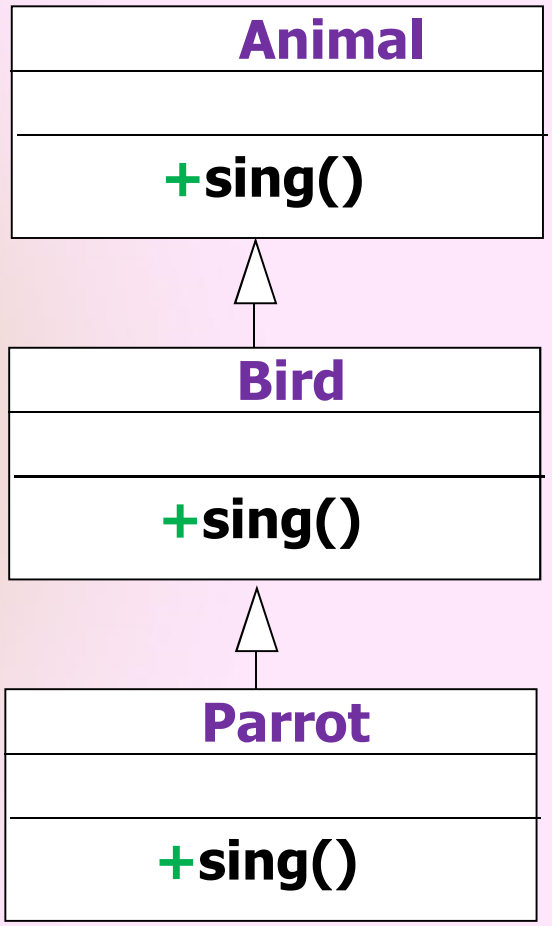
ตัวอย่างเช่น

```
Shape a= new Shape();
Shape a = new Triangle();
Shape a = new Rectangle();
```



ตัวอย่างโปรแกรมที่ 1

```
class Animal {
    public void  sing(){
            System.out.println("La La...");
    }
}
class Bird extends Animal {
   public void  sing(){
            System.out.println("Jib Jib...") ;
   }
}
class Parrot extends Bird {
   public void  sing(){
            System.out.println("Woo Woo...");
   }
}
```



```
public class JavaAppEx1 {
   public static void main(String[] args) { //main
      Animal aniObj = new Animal();
      aniObj.sing();

      aniObj = new Bird();
      aniObj.sing();

      aniObj = new Parrot();
      aniObj.sing();
  }
}
```

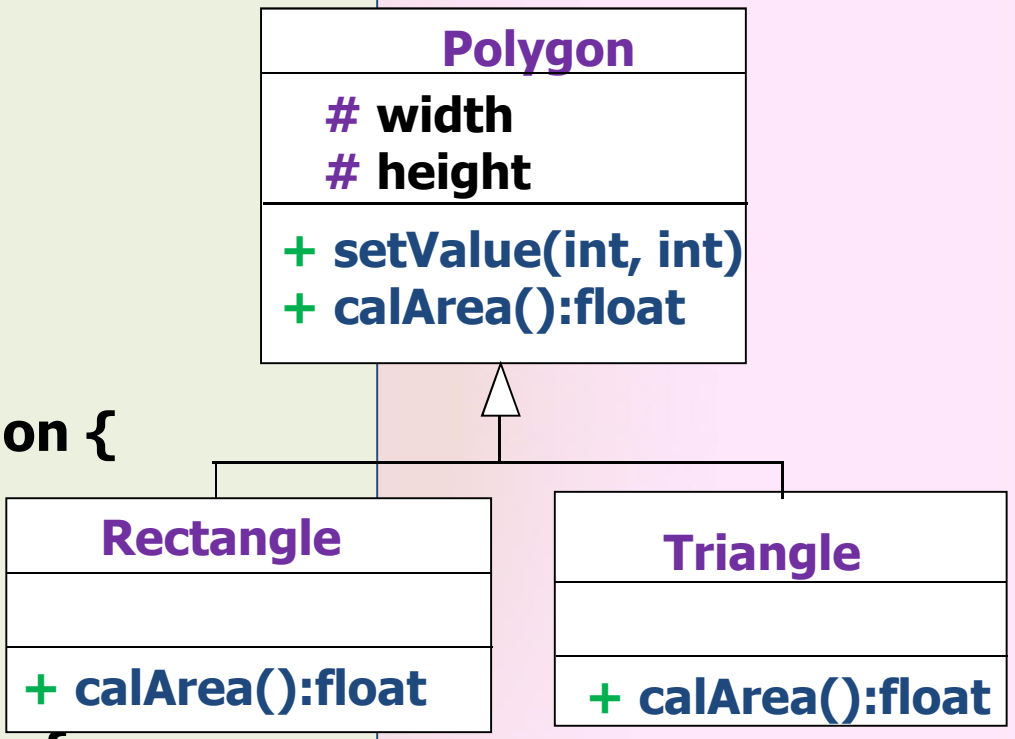
ผลการทำงาน

```
La La...
Jib Jib...
Woo Woo...
```

- การสร้าง Polymorphism ด้วยการใช้ overriding method (พฤติกรรมของลูกลบล้างพฤติกรรมของแม่)
- ทำให้เกิดการตอบสนองต่อเมท็อดเดียวกัน แต่ได้ผลลัพธ์หลายรูปแบบ

ตัวอย่าง โปรแกรมที่ 2

```
class Polygon {
  protected int width, height;
  public void setValue(int a, int b) {
        width=a;
        height=b;
  }
  public float calArea() {
        return 0.0f;
  }
}
class Rectangle extends Polygon {
  public float calArea() {
        return width*height;
  }
}
class Triangle extends Polygon {
  public float calArea() {
        return width*height/2.0f;
  }
}
```

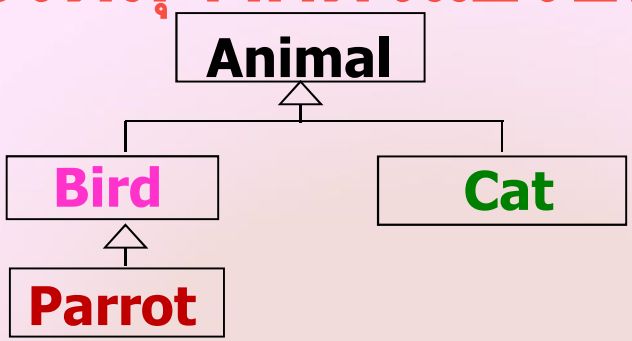


```
public class JavaAppEx2 {
   public static void main(String[] args) { //main
        Polygon  pPoly;
        pPoly = new Rectangle();
        pPoly.setValue (10,5);
        System.out.println(pPoly.calArea());

        pPoly = new Triangle();
        pPoly.setValue (10,5);
        System.out.println(pPoly.calArea());
  }
}
```

ผลการทำงาน

```
50.0
25.0
```

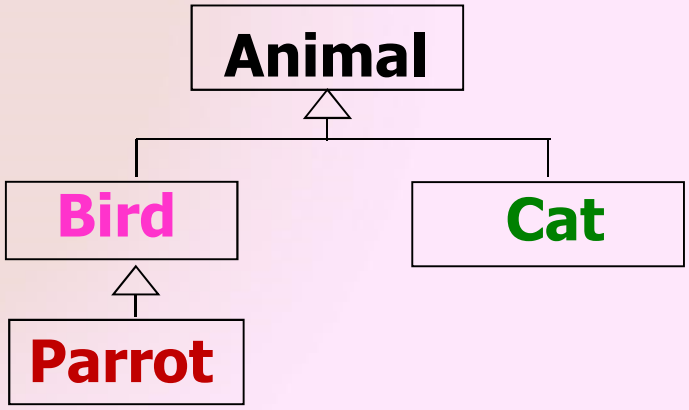
การสร้างวัตถุจากตัวแปรชนิด superclass



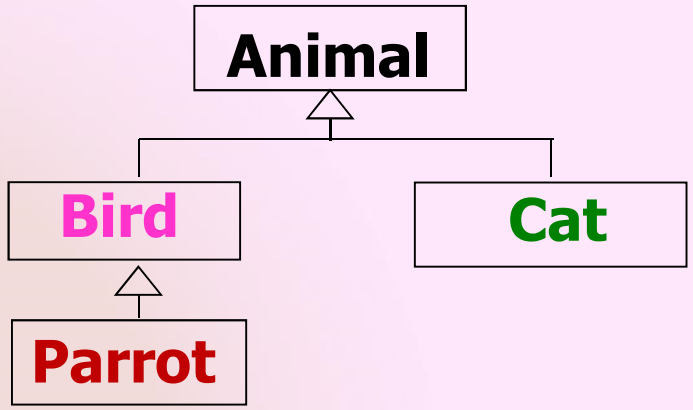
- สามารถสร้าง superclass object จากตัวแปรชนิด superclass ได้
  เช่น Animal a = new Animal();
- สามารถสร้าง subclass object จากตัวแปรชนิด subclass ได้
  เช่น Cat a = new Cat();
- สามารถสร้าง subclass object จากตัวแปรชนิด superclass ได้
  เช่น Animal a = new Cat();
- ไม่สามารถสร้าง superclass object จากตัวแปรชนิด subclass
  เช่น Cat a = new Animal(); //Compiler error เพราะobject
  //แม่จะขาดคุณสมบัติบางอย่าง
  //ที่ลูกมี

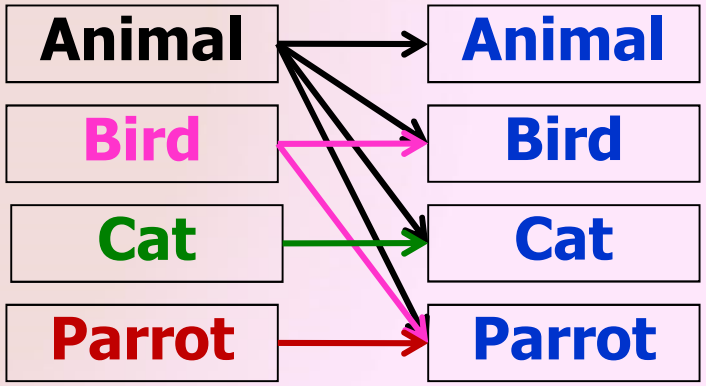
- สามารถเก็บข้อมูลที่มีชนิดข้อมูลต่างกันในกลุ่มเดียวกันได้
- ตัวอย่างการเก็บข้อมูลแบบ Heterogeneous ในอาร์เรย์ของ object เช่น

```
Animal a[] = new Animal[4];
a[0]  =  new Animal();
a[1]  =  new Bird();
a[2]  =  new Cat();
a[3]  =  new Parrot();
```



```
Animal a = new Animal();
Animal a = new Bird();
Animal a = new Cat();
Animal a = new Parrot();

Bird b = new Bird();
Bird b = new Parrot();

Cat c = new Cat();
Parrot d = new Parrot();

Bird x = new Cat(); //ผิด
Bird x = new Animal(); // ผิด
```



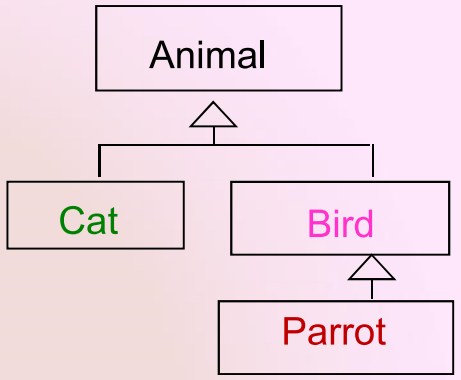


การเรียกใช้ Method ของ Subclass objects

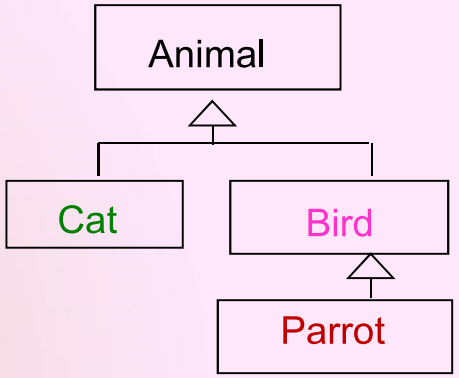
- เมื่อกำหนดให้ subclass object เป็นตัวแปรชนิด superclass
- คอมไพเลอร์ภาษาจาวาไม่อนุญาตให้เรียกใช้ method ที่ไม่มีการประกาศอยู่ใน superclass ที่กำหนดไว้
- ในการเรียกใช้เมท็อดที่ไม่อนุญาตให้ใช้ ของ subclass objects สามารถใช้เทคนิคดังนี้
  - Type Casting
  - ตัวดำเนินการ Instanceof

ตัวอย่างโปรแกรมที่ 3

```
class Animal {
  void print() {
      System.out.println("in Animal Class");
  }
  void eat() {
      System.out.println("Eat eat eat");
  }
}

class Bird extends Animal {
  void print() {    //override
     System.out.println("in Bird Class");
  }
  void fly() {
      System.out.println("I am flying");
  }
}
```



```
class Cat extends Animal  {
   void print() {            //override
      System.out.println("in Cat Class");
   }
   void walk(){
       System.out.println("I am walking");
    }
}

class Parrot extends Bird {
   void print() {            //override
     System.out.println("in Parrot Class");
   }
   void speak(){
     System.out.println("I can speak");
   }
}
```



```
Animal a = new Animal();
a.print();  // in Animal Class
a.eat();    // Eat eat eat
a.fly();    // compile error
a.walk();   // compile error
a.speak(); // compile error
```

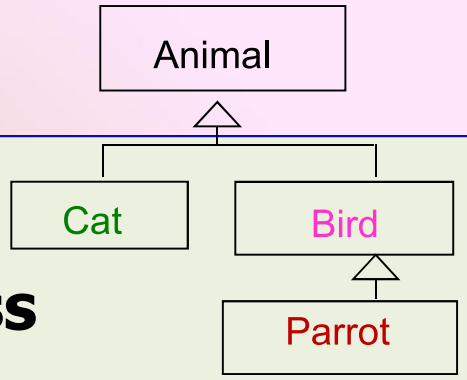
```
Animal a = new Bird();
a.print();   // in Bird Class
a.eat();     // Eat eat eat
a.fly();     // compile error
a.walk();  // compile error
a.speak(); // compile error
```

a.print(); ทำงานได้หลายรูปแบบโดยอัตโนมัติขึ้นอยู่กับว่า **Object** เป็นของ **class** ใด

การเรียกใช้ **method** มีข้อห้ามดังนี้

1. ไม่อนุญาตให้เรียกใช้ **method** ที่ไม่ได้ประกาศใน **Class**
2. ไม่อนุญาต **subclass object** ที่เป็นตัวแปรชนิด **superclass** เรียกใช้ **method** ที่ไม่ได้ประกาศใน **superclass**

การใช้ Type Casting




```
Animal a = new Bird();
a.print();                  // in Bird Class
a.fly();                    //compile error
((Bird)a).fly();            //I am flying
((Cat)a).walk();            // runtime error
((Parrot)a).speak();        // runtime error
((Animal)a).print();        // in Bird Class  เนื่องจาก overriding
```

การใช้ตัวดำเนินการ instanceof

- ตัวดำเนินการ instanceof จะใช้ตรวจสอบว่าเป็นออบเจ็กต์ของคลาสนั้นหรือไม่ โดยจะให้ผลลัพธ์เป็นข้อมูลชนิด boolean

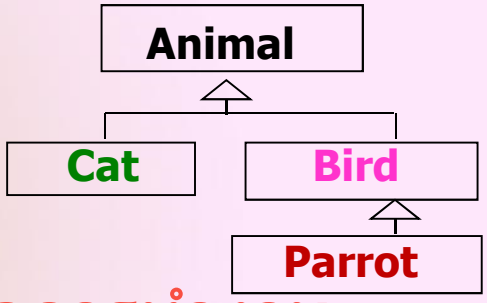
- ตัวอย่าง

```
Bird a = new Bird();
System.out.println(a instanceof Animal ); // true
System.out.println(a instanceof Bird );   // true
System.out.println(a instanceof Cat );    // compile error
System.out.println(a instanceof Parrot ); // false
```

ตัวอย่างโปรแกรมที่ 4

```
Animal a[] = new Animal[4];
a[0]  =  new Animal();
a[1]  =  new Bird();
a[2]  =  new Cat();
a[3]  =  new Parrot();
```

- คำสั่ง a[i].print(); การทำงานของเมท็อด print จะสามารถตอบสนองการทำงานได้หลายรูปแบบโดยอัตโนมัติขึ้นอยู่กับว่า Object เป็นของ class ใด



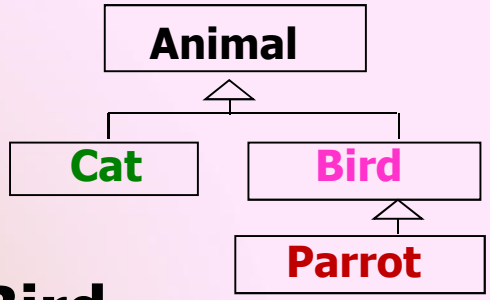

```
for (int i = 0; i < 4; i++){
    a[i].print();
    if (a[i] instanceof Bird)
        ((Bird)a[i]).fly();
    else if (a[i] instanceof Cat)
            ((Cat)a[i]).walk();
        else  if ( a[i] instanceof Parrot) {
                ((Parrot)a[i]).fly();
                ((Parrot)a[i]).speak();
              }
    System.out.println();
}
```

ผลการทำงาน

```
in Animal Class

in Bird Class
I'am flying

in Cat Class
I'am walking

in Parrot Class
I'am flying
```

a[0] a[1] a[2] a[3]

- เนื่องจาก a[3] = new Parrot();
- ความสัมพันธ์ระหว่างคลาสคือ “is-a”



“is-a” relationship → A Parrot is a Bird

```
if (a[i] instanceof Bird)
        ((Bird)a[i]).fly();
```

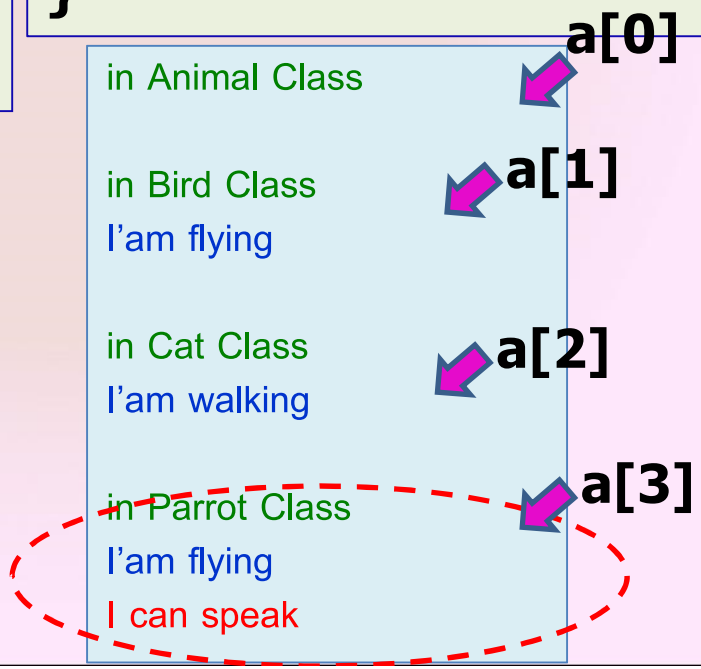
- ดังนั้นเช็คว่า a[3] เป็น Bird จริงไหม
  ผลการเช็คจะเป็น true จึงเข้าทำงานที่ if อันแรก

หากแก้ไข logic ของโปรแกรมใหม่ ดังนี้ (ด้านซ้าย หรือด้านขวาก็ได้)

```
for (int i = 0; i < 4; i++)
{
   a[i].print();
   if (a[i] instanceof Bird){
      ((Bird)a[i]).fly();
       if ( a[i] instanceof Parrot)
         (Parrot)a[i]).speak();
    }
    else if (a[i] instanceof Cat)
              ((Cat)a[i]).walk();
   System.out.println();
}
```

```
for (int i = 0; i < 4; i++)
{
    a[i].print();
    if (a[i] instanceof Bird)
          ((Bird)a[i]).fly();
    if (a[i] instanceof Cat)
          ((Cat)a[i]).walk();
    if ( a[i] instanceof Parrot)
          ((Parrot)a[i]).speak();
    System.out.println();
}
```

ผลการทำงาน

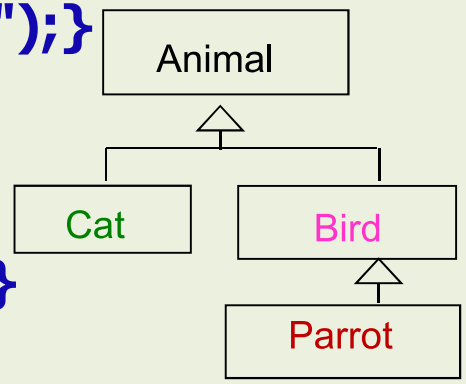


เมท็อดเสมือน (Virtual Method)

- **Virtual Method** คือ เมท็อดที่สร้างขึ้นในคลาสแม่ และมีการสร้างเมท็อดซ้อนทับขึ้นมาในคลาสลูก โดยมี **Method Signature** เช่นเดียวกันกับเมท็อดของคลาสแม่
- ในภาษา **C++** จะมีคำสงวน **"virtual"** นำหน้า
- ในภาษา **Java** จะใช้การนิยามเมท็อดให้เป็น **Empty method** ในคลาสแม่ แล้วจึงสร้างรายละเอียดของเมท็อดขึ้นในคลาสลูก

ตัวอย่างโปรแกรมที่ 5

```
class Animal {
   void print() {System.out.println("in Animal Class");}
   void sing(){ }          // virtual method
}
class Bird extends Animal {
   void print() {System.out.println("in Bird Class");}
   void fly() {System.out.println("I' am flying");}
   void sing(){ System.out.println("Jib Jib"); }
}
class Cat extends Animal  {
   void print() {System.out.println("in Cat Class");}
   void walk(){System.out.println("I' am walking");}
   void sing(){ System.out.println("Miew Miew"); }
}
class Parrot extends Bird {
   void print() {System.out.println("in Parrot Class");}
   void speak(){ System.out.println("I can speak"); }
}
```



```
Animal a[] = new Animal[4];
a[0]  =  new Animal();
a[1]  =  new Bird();
a[2]  =  new Cat();
a[3]  =  new Parrot();
```

```
for (int i = 0; i < 4; i++){
      a[i].print();
      a[i].sing();
      if (a[i] instanceof Bird)
            ((Bird)a[i]).fly();
      if (a[i] instanceof Cat)
            ((Cat)a[i]).walk();
      if ( a[i] instanceof Parrot)
            ((Parrot)a[i]).speak();
      System.out.println();
}
```

ผลการทำงาน

a[0]

```
in Animal Class
```

a[1]

```
in Bird Class
Jib Jib
I'am flying
```

a[2]

```
in Cat Class
Miew  Miew
I'am walking
```

a[3]

```
in Parrot Class
Jib Jib
I'am flying
I can speak
```

คลาสแบบนามธรรม

- คลาสแบบนามธรรม (Abstract Class) คือ คลาสที่ไม่สมบูรณ์ มีการออกแบบรายละเอียดไว้เพียงบางส่วน เพื่อใช้เป็นคลาสแม่ของคลาสอื่น (รายละเอียดที่เหลือจะมีการนิยามในคลาสลูก) โดยคลาสแบบนามธรรมจะต้องมีสมาชิกที่เป็นเมท็อดนามธรรม อย่างน้อย 1 เมท็อด และไม่สามารถสร้างวัตถุจากคลาสแบบนามธรรมได้
- เมท็อดนามธรรม (Abstract Method) คือ เมท็อดที่จะไม่กำหนดการทำงานไว้ภายใน ถูกสร้างขึ้นเพื่อให้เป็นต้นแบบสำหรับคลาสลูกเท่านั้น โดยในคลาสลูกจะมีสร้างเมท็อดขึ้นมาซ้อนทับ เพื่อกำหนดรายละเอียดการทำงานของเมท็อดดังกล่าว และมีการเรียกใช้งานเมท็อดดังกล่าวผ่านวัตถุของคลาสลูก

รูปแบบของ Abstract Class

[*modifier*] abstract class *class_name* { .....}

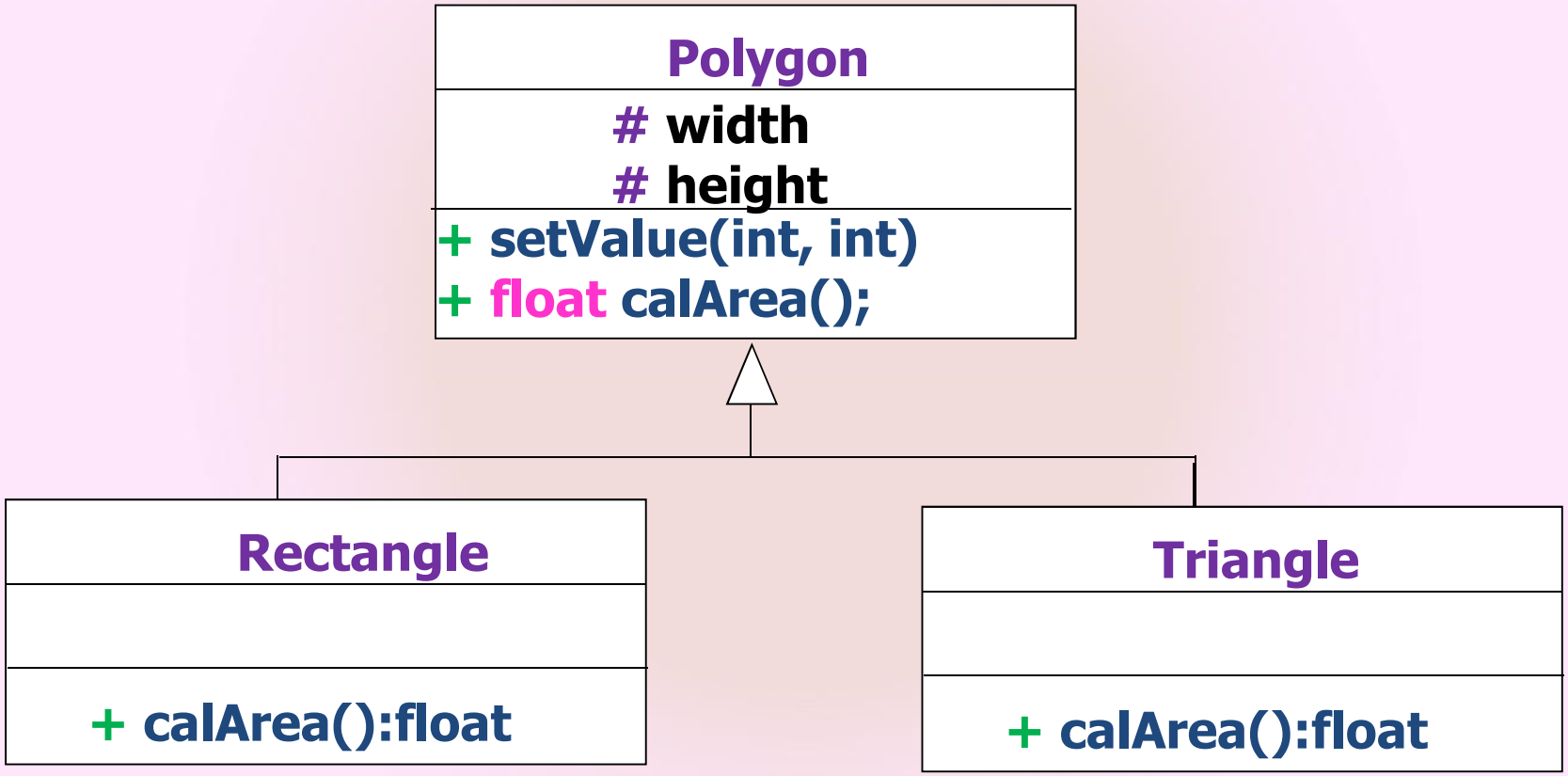
รูปแบบของ Abstract Method

[*modifier*] abstract *return_type methodName*([*arguments*]);

- อยู่ในรูปแบบของ Prototype
- ไม่ต้องมีเครื่องหมาย { } เพื่อบอกขอบเขต (scope) ของเมท็อด
- ใช้สำหรับให้คลาสที่มาสืบทอด มาทำการ override

ตัวอย่างโปรแกรมที่ 6

โปรแกรมเพื่อแสดงการใช้งานเมท็อดนามธรรม(Abstract Method) และคลาสแบบนามธรรม (Abstract Class)



```
abstract class Polygon {
  protected int width, height;
  public void setValue(int a, int b) {
        width=a;
        height=b;
  }
   abstract float calArea(); //abstract method
}
class Rectangle extends Polygon {
  public float calArea() {
    return width*height;
  }
}
class Triangle extends Polygon {
  public float calArea() {
        return width*height/2.0f;
  }
}
```

โปรแกรม

(สังเกต code จะคล้ายกับ ex.2)

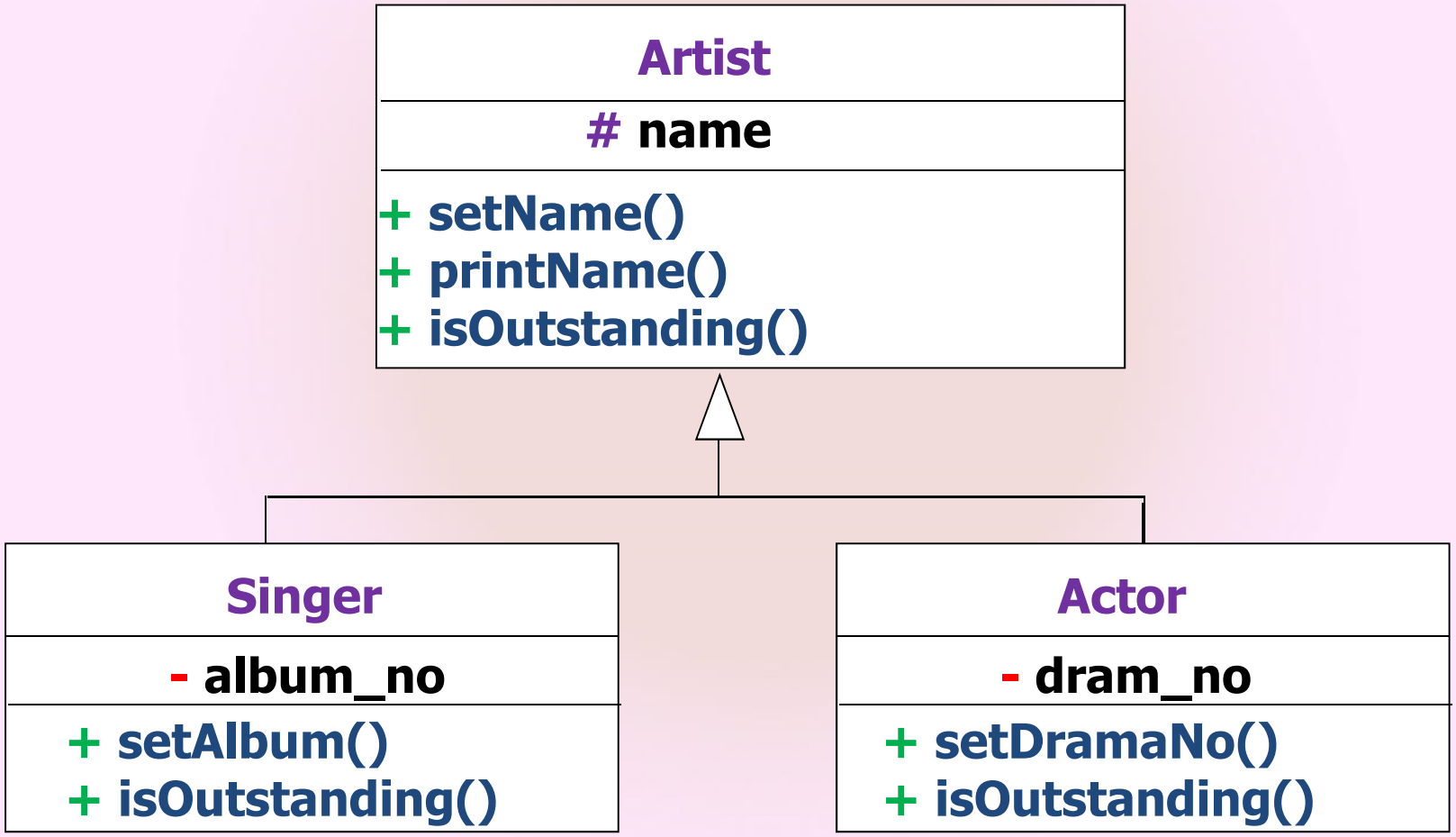
```
public class JavaAppEx {
    public static void main(String[] args) {
        Polygon  pPoly = new Rectangle();
        pPoly.setValue (10,5);
        System.out.println(pPoly.calArea());

        pPoly = new Triangle();
        pPoly.setValue (10,5);
        System.out.println(pPoly.calArea());
    }
}
```

ผลการทำงาน

```
50.0
25.0
```

ตัวอย่างโปรแกรมที่ 7

โปรแกรมเพื่อแสดงการใช้งานเมท็อดนามธรรม(Abstract Method) และคลาสแบบนามธรรม (Abstract Class)



โปรแกรม

```
abstract class Artist  {
 protected String name;

 public void  setName() {
    Scanner inVar = new Scanner(System.in);
    System.out.print("Enter Name: ") ;
    name = inVar.nextLine();
 }

 public void printName() {
    System.out.println (" Name : " + name );
 }

 public abstract  boolean isOutstanding();
}
```

โปรแกรม

```
class Singer extends Artist {
  private int  album_no;
  public void setAlbumNo() {
       Scanner inVar = new Scanner(System.in);
      System.out.print(" Enter number of albums:  ");
      album_no = inVar.nextInt();
    }

    public boolean isOutstanding() {
      return (album_no >= 10) ? true : false;
    }
}
```

```
class  Actor extends Artist {
  private int  drama_no;
  public void setDramaNo() {
     Scanner inVar = new Scanner(System.in);
     System.out.print("Enter number of dramas:  ");
     drama_no = inVar.nextInt();
  }

  public boolean isOutstanding() {
        return (drama_no >= 20) ? true : false;
  }
}
```

```
public class JavaAppEx2 {
    public static void main(String[] args) {
        Artist[] artPtr = new Artist[100];
        Scanner inVar = new Scanner(System.in);
        Singer sPtr;
        Actor aPtr;

        int n = 0, i ;
        char choice;
```

```
do {
        System.out.print("Select singer or actor (s/a): ");
         choice = inVar.nextLine().charAt(0);
         if (choice == 's') {
                  artPtr[n] = new Singer();
                  artPtr[n].setName();
                  ((Singer)artPtr[n]).setAlbumNo();
                  n++;
         }
         else{
                  artPtr[n] = new Actor();
                  artPtr[n].setName();
                  ((Actor)artPtr[n]).setDramaNo();
                  n++;
         }
         System.out.print("Enter another (y/n) ? ");
         choice = inVar.nextLine().charAt(0);
         System.out.println(".............................");
} while (choice == 'y');
```

```
  for (i=0; i<n; i++) {
          artPtr[i].printName();
          if (artPtr[i].isOutstanding())
              System.out.print("This person is outstanding\n\n");
          else
              System.out.print("This person is not outstanding\n\n");
    }
  }
}
```

Select singer or actor (s/a): s
Enter Name: Tata
Enter number of albums: 15
Enter another (y/n) ? y
.....

Select singer or actor (s/a): a
Enter Name: Nadech
Enter number of dramas : 10
Enter another (y/n) ? y
.....

Select singer or actor (s/a): a
Enter Name: Theeradech
Enter number of dramas : 23
Enter another (y/n) ? y
.....

ตัวอย่างผลการทำงาน

```
Select singer or actor (s/a): s
Enter Name: Baitoey
Enter number of albums: 5
Enter another (y/n) ? n
.............................
Name : Tata
This person is outstanding

Name : Nadech
This person is not outstanding

Name : Theeradech
This person is outstanding

Name : Baitoey
This person is not outstanding
```

Interface Class

- คล้ายกับ **abstract class** แต่หากมี **method** ภายในคลาส จะมีเฉพาะ **method** ที่ยังไม่สมบูรณ์(**Abstract Method**)เท่านั้น

```
[modifier] interface InterfaceName {
    [modifier] return_type methodName([arguments]);
        .....
}
```

Interface Class

- คลาสอื่นนำไปใช้งานโดยใช้คีย์เวิร์ด implements

```
[modifier] class ClassName implements InterfaceName {

    methods();
      ......
}
```

เช่น class Teacher implements Person

Interface Class

- ไม่สามารถสร้างออบเจ็กต์จาก Interface Class เช่นเดียวกับ abstract class
- ประโยชน์ของ Interface class คือ
  - ใช้กำหนดรูปแบบของเมท็อดต่างๆ ที่คลาสอื่นๆ จะต้อง implements ไว้ล่วงหน้า และอาศัยหลักการของ polymorphism (การมีได้หลายรูปแบบ) มาเรียกใช้เมท็อดเหล่านั้น
  - ภาษาจาวากำหนดให้คลาสใดๆสามารถสืบทอดคลาสอื่นได้เพียงคลาสเดียวเท่านั้น แต่จะสามารถ implements จากหลาย Interface class ได้

ตัวอย่างที่ 1

```
import  java.util.Scanner;
interface  Person {
      void setName(String n);  //abstract method
      void setAge(int a);          //abstract method
      void display(int i);          //abstract method
}
class Student implements Person {
       private  String name;
       private  int age;
       private double gpa;
       public void setName(String n){  //implement
          name = n;
       }
      public void setAge(int a){          //implement
         age = a;
      }
       public void setGPA(double g){
           gpa = g;
      }
       public void display(int i){          //implement
          System.out.println("["+i+"] A student  Name = "
                    +name + " Age = "+ age+ " GPA = "+ gpa);
       }
}
```

```
class Teacher implements Person {
      private  String name;
      private  int age;
      private String deptName;
      public void setName(String n){   //implement
         name = n;
      }
      public void setAge(int a){          //implement
           age = a;
      }
      public void setDept(String dept){
           deptName = dept;
      }
      public void display(int i){          //implement
          System.out.println("["+i+"] A teacher  Name = "
           +name+ " Age = " +age + " Dept= " + deptName);
      }
}
```

```
public class JavaApp {
   public static void main(String[] args) { //main
   Person [] p = new Person[5];
   Scanner input = new Scanner(System.in);
   int N,i=0;  char type;

   do {
     System.out.print("Number of person (<=5) : ");
     N= input.nextInt();
   } while (N > 5 || N < 0);
```

```
while (i<N) {
    System.out.print("Type of person (s/t) : ");
    type = input.next().charAt(0);
    if (type == 's') {
        p[i]=new Student();
        System.out.print("Name : ");
        p[i].setName(input.nextLine());
        System.out.print("Age : ");
        p[i].setAge(input.nextInt());
        System.out.print("GPA : ");
        ((Student) p[i]).setGPA(input.nextDouble());
        i++;
 }
```

```
    else if (type == 't') {
        p[i]=new Teacher();
        System.out.print("Name : ");
        p[i].setName(input.nextLine());
        System.out.print("Age : ");
        p[i].setAge(input.nextInt());
        System.out.print("Dept. Name : ");
        ((Teacher) p[i]).setDept(input.next());
        i++;
      }
    else
      System.out.println("Invalid Type of person");
}
for(i=0;i<N;i++)
      p[i].display(i);
```

ผลการทำงาน

```
Number of person (<=5) : 3
Type of person (s/t) : s
Name : Nida
Age : 15
GPA : 4.00
Type of person (s/t) : s
Name : Rattana
Age : 18
GPA : 3.5
Type of person (s/t) : t
Name : Benjamas
Age : 35
Dept. Name : Computer
[0] A student  Name = Nida  Age = 15 GPA = 4.0
[1] A student  Name = Rattana Age = 18 GPA = 3.5
[2] A teacher  Name = Benjamas Age = 35 Dept= Computer
```

Interface Class

- คลาสอื่นที่ implement จาก interface จะต้องกำหนดคำสั่งของเมท็อดนามธรรมของ interface ให้ครบทุกเมท็อด

```
interface A {
   void printA();
}

class C implements A {          //Compile error
    void printC() {
       System.out.println("print C");
    }
}
```

-คลาส C ต้องนิยามคำสั่งของเมท็อด printA()

Interface Class

- คลาสอื่นสามารถ implement จากหลาย interface ได้ แต่ต้อง implement ให้ครบทุกเมท็อด

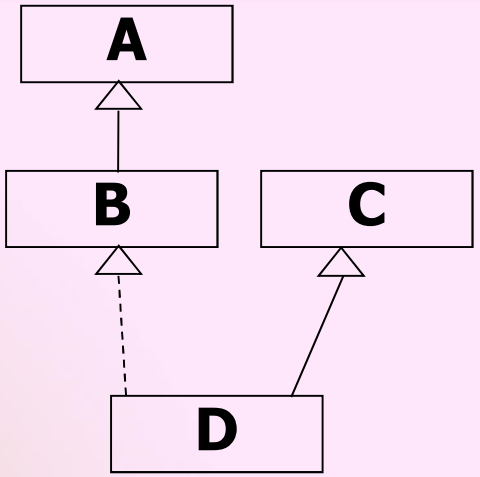
```
interface A {
    void printA();
}

interface B {
    void printB();
}
```

```
class C implements A,B {
   void printA() {  //implement
          System.out.println("print A");
   }
   void printB() { //implement
          System.out.println("print B");
    }
   void printC() {
         System.out.println("print C");
   }
}
```

-คลาส C ต้องนิยามคำสั่งของเมท็อดต่างๆ ใน interface A และ B ให้ครบ

Interface Class

- คลาสอื่นสามารถ implement จากหลาย interface ได้ แต่ต้อง implement ให้ครบทุกเมท็อด

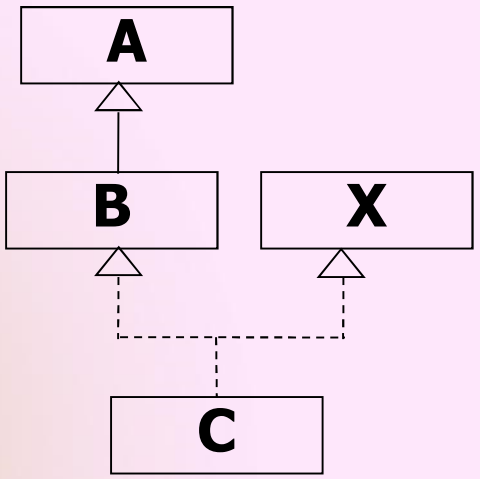



```
interface A {
  void printA();
}

interface B extends A {
   void printB();
}

class C {
   public void printC(){
     System.out.print("C");
  }
}
```

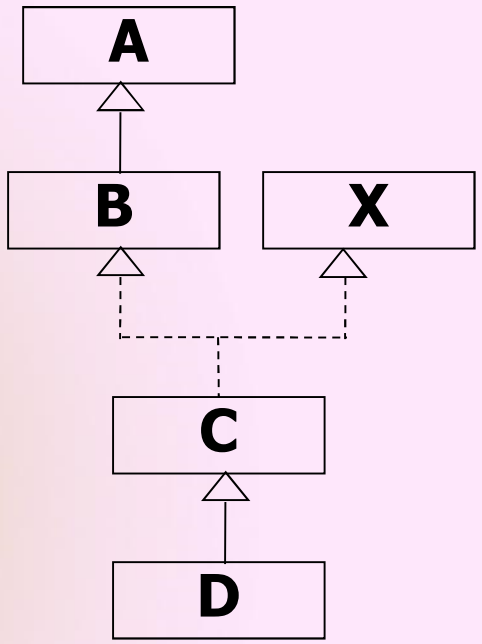
```
class D extends C implements B {
   public void printA() {  //implement
         System.out.println("A");
   }
   public void printB() { //implement
         System.out.println("B");
    }
   public void printC() { //override
         System.out.println("print C");
   }
}
```

ตัวอย่างที่ 2

```
interface A {
  void printA();
}
interface B extends A {
  void printB();
}
interface X {
  void printX();
}
class C implements B,X {
    public void printA() {  //implement
          System.out.println("print A");
    }
    public void printB() {//implement
          System.out.println("print B");
    }
    public void printX() {//implement
         System.out.println("print X");
    }
    public void printC() {
         System.out.println("print C");
    }
}
```



```
class D extends C {
   public void printD() {
          System.out.println("print D");
    }
}
```

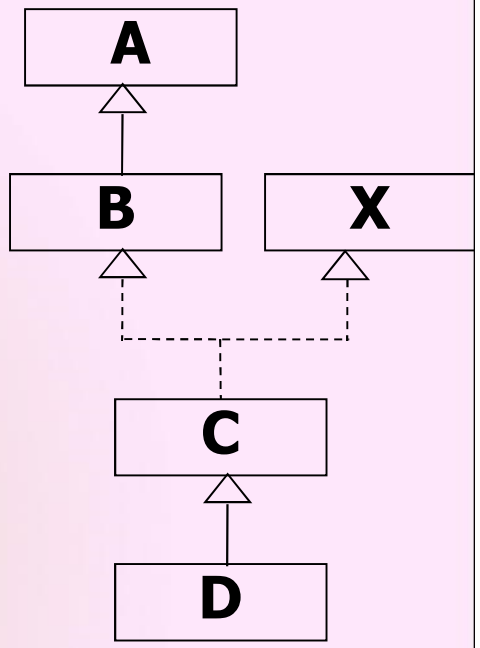


```
class JavaExInterface{
   public static void main(String args[]){
    C x= new C();
    x.printA();
    x.printB();
    x.printC();
    x.printX();
    System.out.println("---------");
    D y= new D();
    y.printA();
    y.printB();
    y.printC();
    y.printX();
    y.printD();
    System.out.println("---------");
    C z = new D();
    z.printA();
    z.printB();
    z.printC();
    z.printX();
    ((D)z).printD();
    }
```

ใช้ **printD** เฉย ๆ ไม่ได้ เพราะ **type** เป็น **C**

ผลการทำงาน

```
print A
print B
print C
print X
---------
print A
print B
print C
print X
print D
---------
print A
print B
print C
print X
print D
```

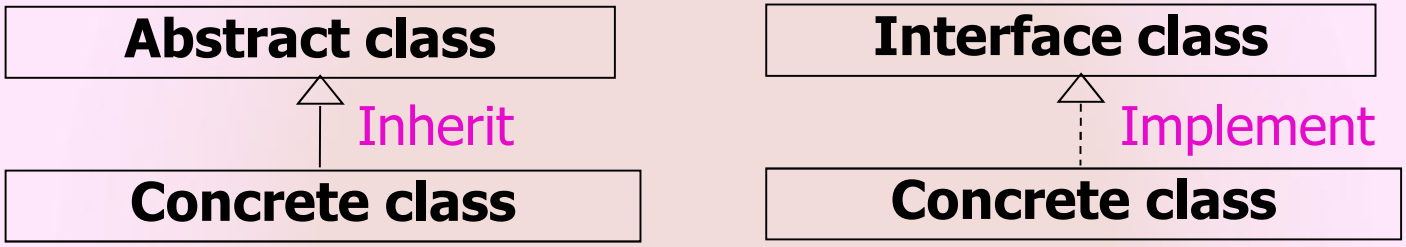


สรุปความเหมือนและความต่างของ Abstract Class กับInterface Class

- ไม่สามารถสร้างออบเจ็กต์จาก Interface Class และ Abstract Class ได้
- Interface Class จะประกอบด้วย Abstract Method เท่านั้น ไม่มี data ไม่มีการ implement
- Abstract Class จะประกอบด้วย Abstract Method อย่างน้อย 1 เมท็อด และสามารถจะมี Concrete Method (เมท็อดที่ implement แล้ว) ด้วยได้ และสามารถมี data ได้

สรุปความเหมือนและความต่างของ Abstract Class กับInterface Class

- มีความสัมพันธ์กับ Concrete Class (คลาสที่มา implement) ที่แตกต่างกัน



Generalization Realization

- คลาสใด ๆ จะสามารถ inherit จาก Abstract Class ได้เพียง 1 คลาสเท่านั้น แต่สามารถ implement จาก Interface Class ได้หลายคลาส (คล้าย multiple inheritance)