Faculty of Information Technology
King Mongkut's Institute of Technology Ladkrabang

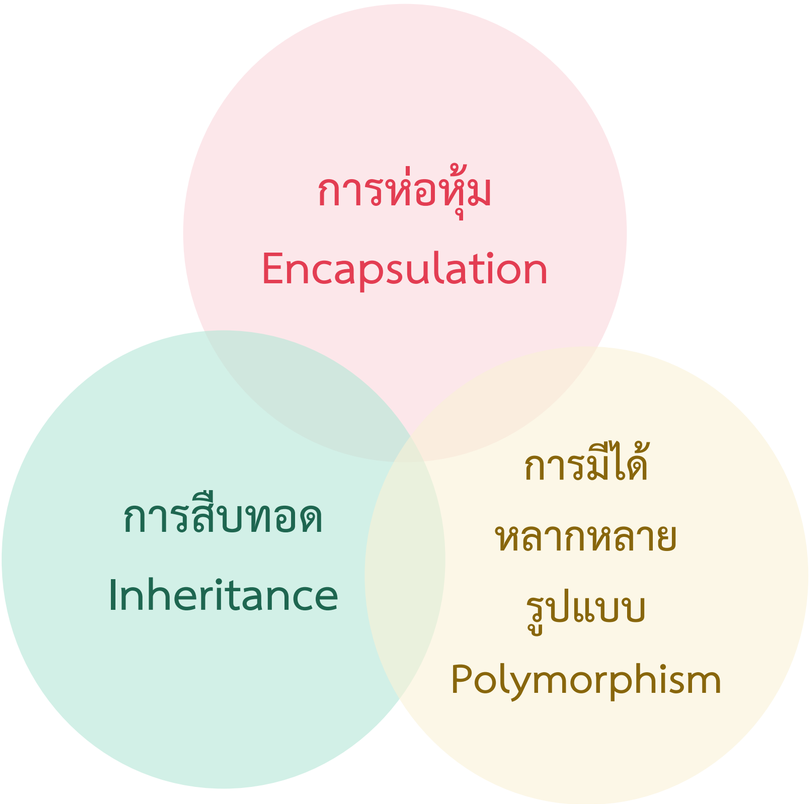
บทที่ 6 การเขียนโปรแกรมเชิงวัตถุ โดยอาศัยการมีได้หลากหลายรูปแบบ

บรรยายโดย ผศ.ดร.ธราวิเชษฐ์ ธิติจรูญโรจน์

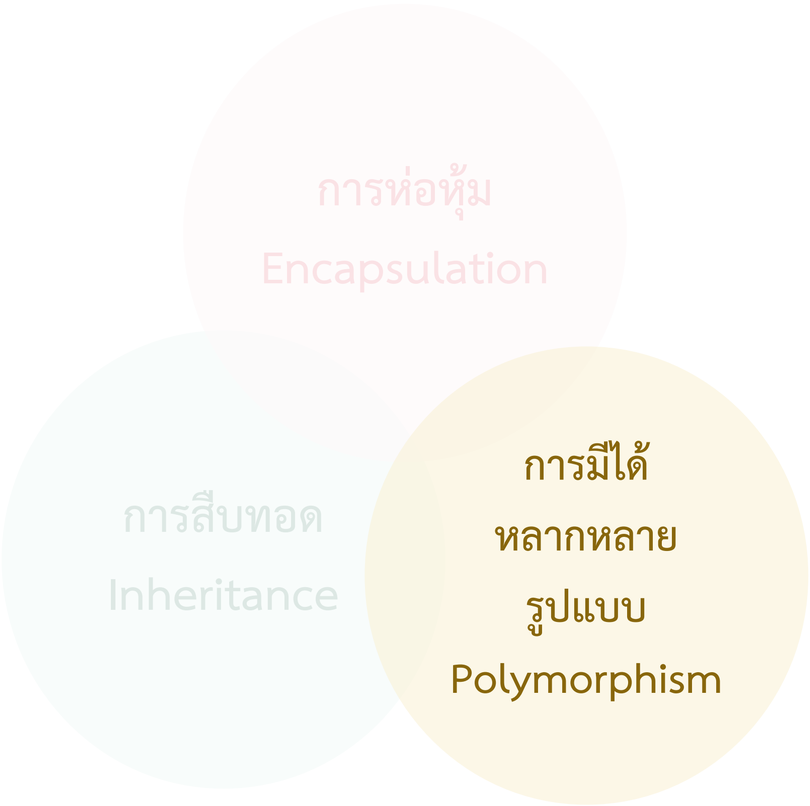
คณะเทคโนโลยีสารสนเทศ

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

การเขียนโปรแกรมเชิงวัตถุ



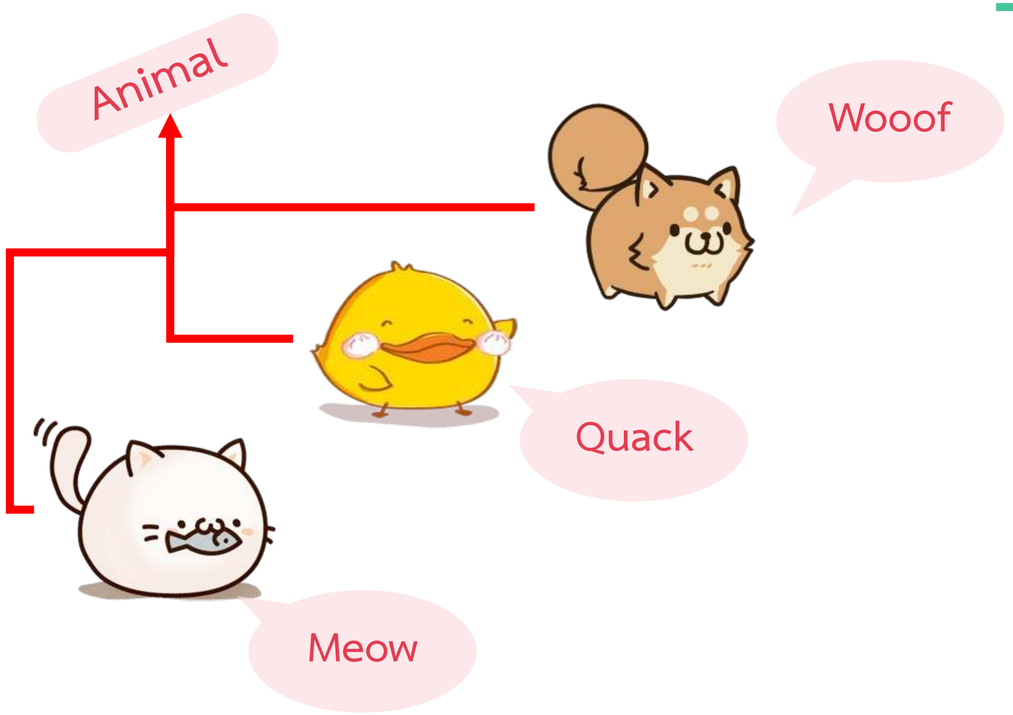
การเขียนโปรแกรมเชิงวัตถุ



หัวข้อ

- การมีหลากหลายรูปแบบ (Polymorphism)
- หลักการ Upcasting และ Downcasting
- การ Overloaded เมธอด
- การ Overridden เมธอด
- หลักการทำงานของ Dynamic Binding
- หลักการทำงานของ Virtual Method Invocation

การมีหลากหลายรูปแบบ (Polymorphism)



Polymorphism มีรากศัพท์มาจากภาษากรีกคำว่า “poly” และ “morphs” โดยที่ "Poly" หมายถึงมากมาย และ "Morphs" หมายถึงรูปแบบ ดังนั้น จึงมีความหมายว่ารูปแบบนับไม่ถ้วนหรือรูปแบบที่หลากหลาย สำหรับการเขียนโปรแกรมเชิงวัตถุการมีได้หลายรูปแบบหมายถึง คุณสมบัติของอ็อบเจกต์ของคลาสที่ต่างกันสามารถตอบสนองต่อเมธอดเดียวกันในวิธีการที่ต่างกันได้ ซึ่งหมายถึงการเขียนเมธอดแบบ Overloaded, Overridden และการใช้ Dynamic Binding

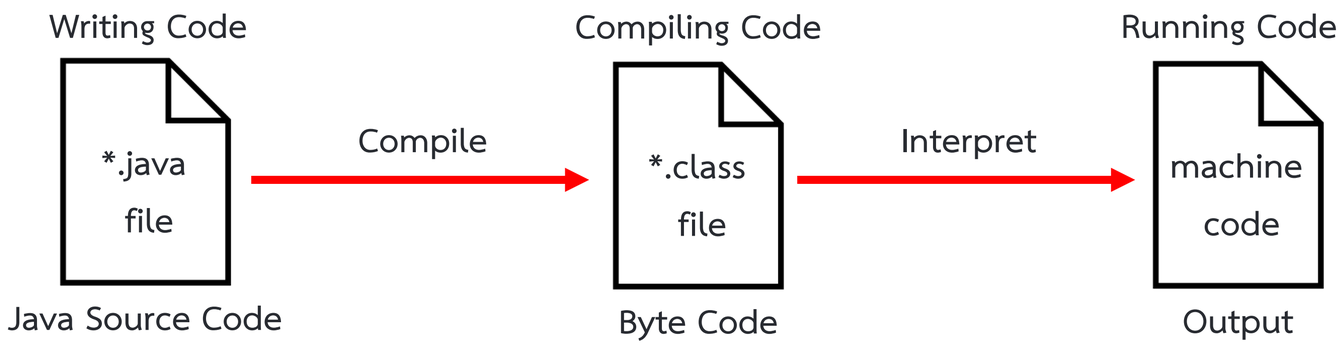
การมีหลากหลายรูปแบบ (Polymorphism)

ขณะเวลาที่อยู่ที่	บริบท
บ้าน	ลูก
โรงเรียน	นักเรียน
ร้านอาหาร	ลูกค้า
รถโดยสาร	ผู้โดยสาร

สมมติว่าถ้านักศึกษาอยู่ที่โรงเรียนเวลานั้นนักศึกษาทำตัวเหมือนนักเรียน เวลานักศึกษาไปตลาดเวลานั้นนักศึกษาทำตัวเหมือนลูกค้า เวลานักศึกษาอยู่บ้านเวลานั้นนักศึกษาทำตัวเหมือนลูกชายหรือลูกสาว ดังนั้น จะพบบุคคลหนึ่งแสดงพฤติกรรมแตกต่างกันในสถานการณ์แตกต่างกัน

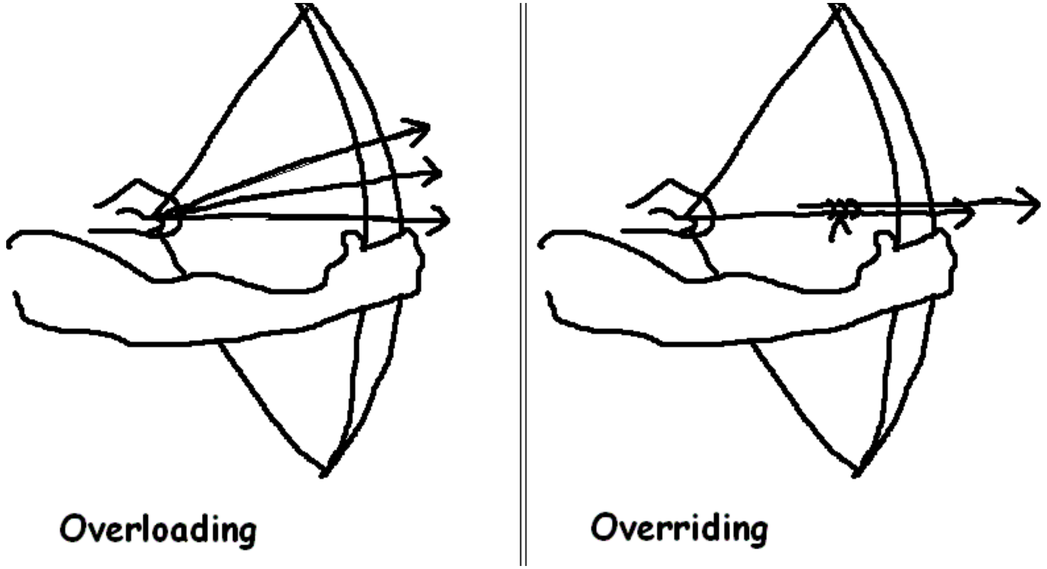
การมีหลากหลายรูปแบบ (Polymorphism)

การมีหลากหลายรูปแบบ (Polymorphism) ในภาษาจาวายังสามารถแบ่งออกได้เป็น 2 ประเภท ได้แก่



- “Compile-Time Polymorphism” เรียกอีกอย่างว่า “Static Polymorphism” ซึ่งการเรียกใช้เมธอดจะได้รับการกำหนดช่วงเวลาคอมไพล์ การ Compile-Time Polymorphism สามารถทำได้โดยอาศัยหลักการ Overloading เมธอด
- “Runtime Polymorphism” เรียกอีกอย่างว่า “Dynamic Binding” หรือ “Dynamic Method Dispatch” ในกระบวนการนี้ การเรียกใช้เมธอดจะได้รับการกำหนดหรือปรับเปลี่ยนแบบ Dynamic ขณะรันไทม์ การ “Runtime Polymorphism” สามารถทำได้โดยอาศัยหลักการ Overriding เมธอด

การมีหลากหลายรูปแบบ (Polymorphism)



Overloaded และ Overridden

เป็นการกำหนดให้เมธอด (ที่มีชื่อเหมือนกัน) ของคลาส (ทั้ง**ภายในคลาสเดียวกัน**และ**คลาสที่สืบทอด**กัน) มีวิธีการทำงานที่แตกต่างกัน ซึ่งสามารถทำได้ 2 รูปแบบ (1) Overloaded และ (2) Overridden

หัวข้อ

- การมีหลากหลายรูปแบบ (Polymorphism)
- หลักการ Upcasting และ Downcasting
- **การ Overloaded เมธอด**
- การ Overridden เมธอด
- หลักการทำงานของ Dynamic Binding
- หลักการทำงานของ Virtual Method Invocation

Overloaded

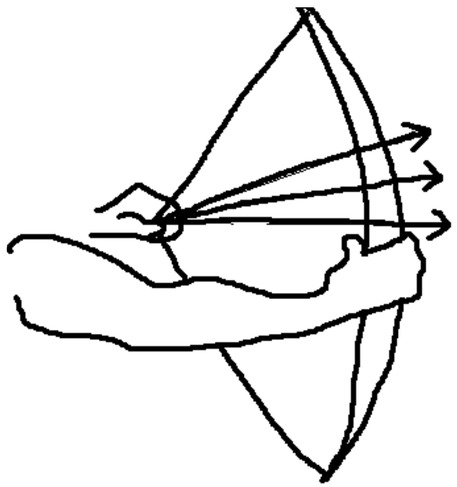
Woof

speak();

Hae Hae

speak( );

Overloaded



Overloading

เป็นการกำหนดให้เมธอด (ที่มีชื่อเหมือนกัน) **ภายในคลาสเดียวกัน** มี (1) การทำงาน หรือ (2) มีอินพุท หรือ (3) มีผลลัพธ์ ที่แตกต่างกัน เช่น คลาส `PrintStream` มีการ `overloaded` เมธอด `println()` ไว้ทั้งหมด 9 รูปแบบ เพื่อรองรับอินพุท `(input)` อย่างหลากหลาย ส่งผลให้เมธอดมีความยืดหยุ่นในการใช้งานมากยิ่งขึ้น

void	**println**(boolean x)Prints a boolean and then terminate the line.
void	**println**(char x)Prints a character and then terminate the line.
void	**println**(char[] x)Prints an array of characters and then terminate the line.
void	**println**(double x)Prints a double and then terminate the line.
void	**println**(float x)Prints a float and then terminate the line.
void	**println**(int x)Prints an integer and then terminate the line.
void	**println**(long x)Prints a long and then terminate the line.
void	**println**(**Object** x)Prints an Object and then terminate the line.
void	**println**(**String** x)Prints a String and then terminate the line.

หลักการ Overloaded

ในภาษาจาวา นักศึกษาสามารถ overloading เมธอดได้ด้วย 2 แนวทาง ได้แก่

ข้อที่ 1 ปรับเปลี่ยนจำนวนของ parameter

```
public class MyMath {
    public int minus(int a, int b) {
        return (a-b);
    }
    public int minus(int a, int b, int c){
        return (a-b-c);
    }
}
public class Main {
    public static void main(String[] args) {
        MyMath m = new MyMath();
        System.out.println(m.minus(10,5));
        System.out.println(m.minus(10,2,5));
    }
}
```

หลักการ Overloaded

ในภาษาจาวา นักศึกษาสามารถ overloading เมธอดได้ด้วย 2 แนวทาง ได้แก่

ข้อที่ 2 ปรับเปลี่ยน data type ของ parameter

Type

③ ลำดับไม่เหมือนได้

```
public class MyMath {
    public int minus(int a, int b) {
        return (a-b);
    }
    public double minus(String a, String b){
        return (Double.parseDouble(a)- Double.parseDouble(b));
    }
}
public class Main {
    public static void main(String[] args) {
        MyMath m = new MyMath();
        System.out.println(m.minus(10,5));
        System.out.println(m.minus("10","2"));
    }
}
```

หลักการ Overloaded

การ overloading เมธอดไม่สามารถทำได้โดยเปลี่ยนการคืนค่าของเมธอด กล่าวคือ overloading เมธอดไม่เกี่ยวข้องกับประเภทการส่งคืนค่า การ overloading เมธอด 2 เมธอดต้องมีพารามิเตอร์ต่างกัน ดังนั้น หากมีสองเมธอดที่มีพารามิเตอร์เดียวกันภายในคลาส จะก่อให้เกิดข้อผิดพลาดถึงแม้ว่าสองเมธอดจะมีการส่งคืนค่าที่แตกต่างกันก็ตาม **หมายความว่าการ overloading ไม่มีความสัมพันธ์กับการส่งคืนค่า**

```
public class MyMath{
        public int minus(int x, int y){
                return x-y;
        }
        public double minus(int x, int y){
                return 1.0*(x-y);
        }
        public static void main(String args[]){
                MyMath m = new MyMath();
                System.out.printIn(m.minus(5,2));
        }
}
```

```
MyMath.java:5: error: method
minus(int, int) is already
defined in class Sample
```

ตัวอย่าง Overloaded ที่ถูกต้อง

```
public class Main {
  public static void main(String[] args) {
      Main m = new Main();
      m.hi();
      m.hi("Bank");
      m.hi("Tara", "Vichet");
  }

  public void hi(){
    System.out.println("Hi");
  }
  public void hi(String n){
    System.out.println("Hi "+n);
  }
  public void hi(String f, String l){
    System.out.println("Hi " + f + " " + l);
  }
}
```

ผลลัพธ์

```
Hi
Hi Bank
Hi Tara Vichet
```

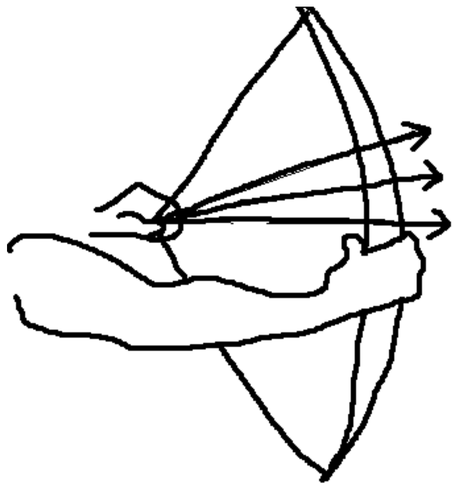
ตัวอย่าง Overloaded ที่ไม่ถูกต้อง

```
public class Main {
  public static void main(String[] args) {
    Main m = new Main();
    m.hi();
    m.hi("Bank");
  }
  public void hi(){
    System.out.println("Hi");
  }
  public String hi(){
    return "Hi";
  }
  public void hi(String n){
    System.out.println("Hi "+n);
  }
  public void hi(String f){
    System.out.println("Hi " + f);
  }
}
```

ผลลัพธ์

```
Main.java:11: error: method hi() is already defined in
class Main
  public String hi(){
                 ^
Main.java:17: error: method hi(String) is already
defined in class Main
  public void hi(String f){
              ^
2 errors
```

ตัวอย่าง Overloaded



Overloading

ตัวอย่าง การเขียนเมธอดแบบ Overloaded **ถูกต้อง**

```
public void setDetail(String ID, String n)
public void setDetail(String ID, double GPA)
public double setDetail(double GPA, String ID)
```

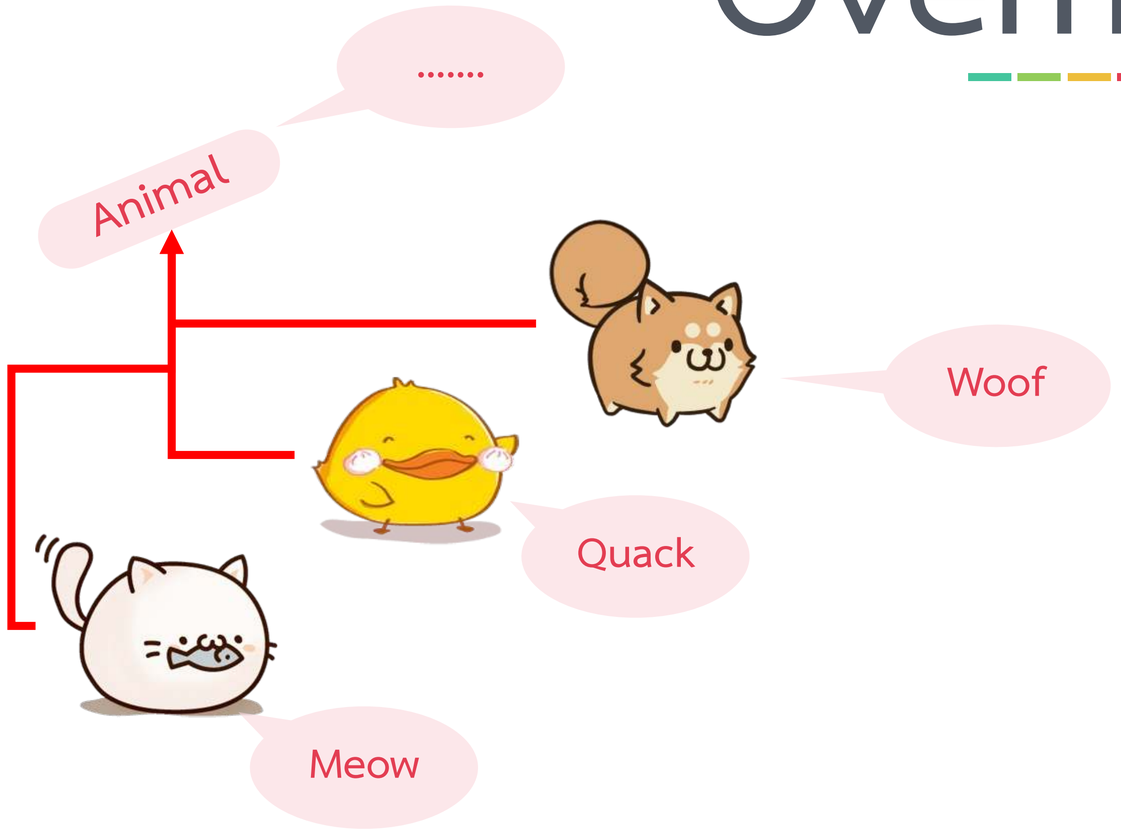
ตัวอย่าง การเขียนเมธอดแบบ Overloaded **ไม่ถูกต้อง**

```
public void setDetail(String ID, double GPA)
public void setDetail(String n, double GPA)
```

หัวข้อ

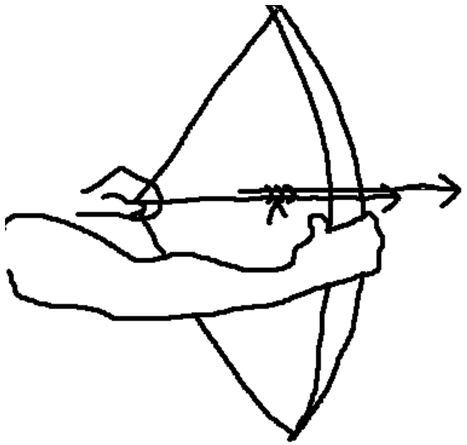
- การมีหลากหลายรูปแบบ (Polymorphism)
- หลักการ Upcasting และ Downcasting
- การ Overloaded เมธอด
- **การ Overridden เมธอด**
- หลักการทำงานของ Dynamic Binding
- หลักการทำงานของ Virtual Method Invocation

Overridden



การ Overriding เมธอดในภาษาจาวาคือคลาสลูก (Subclass) มีเมธอดที่ชื่อเหมือนในเมธอดที่คลาสแม่ (Superclass) จุดประสงค์มีไว้เพื่ออนุญาตให้เมธอดในคลาสลูกสามารถเพิ่มเติมการทำงานจำเพาะ (เพิ่มโค้ด) ในเมธอด

Overridden



Overriding

เป็นการอนุญาตให้คลาสลูก (Subclass) สามารถเพิ่มเติมการทำงานจำเพาะ (เพิ่มโค้ด) ในเมธอดที่คลาสแม่ (Superclass) มีไว้แล้วได้ โดยมีเงื่อนไขดังนี้

ข้อที่ 1 parameter

จำนวนและชนิดของ argument จะต้องเหมือนเดิม

ข้อที่ 2 return

ชนิดข้อมูลของค่าที่ส่งกลับ จะต้องเหมือนเดิม

ข้อที่ 3

Access Modifier จะต้องมีระดับไม่ต่ำกว่าเดิม

ตัวอย่าง Overridden

```
public class Student{
    protected String id;
    protected String name;
    protected double gpa;
    public void printDetail(){
        System.out.println("ID "+id);
        System.out.println("Name "+name);
    }
}public class GradStudent extends Student{
    protected String thesisTitle;
    public String getThesisTitle(){ return this.thesisTitle; }
    public void printDetail(){
        System.out.println("ID "+id);
        System.out.println("Name "+name);
        System.out.println("GPA "+gpa);
        System.out.println("Title "+thesisTitle);
    }
}
```

ใช้แทน

super.printDetail();

ตัวอย่าง Overridden

```
public class Vehicle{
    public void run(){
        System.out.println("Vehicle is running");
    }
}

public class Motorcycle extends Vehicle{
    public void run(){
        System.out.println("Bike is running safely");
    }
}
```

ตัวอย่าง Overridden

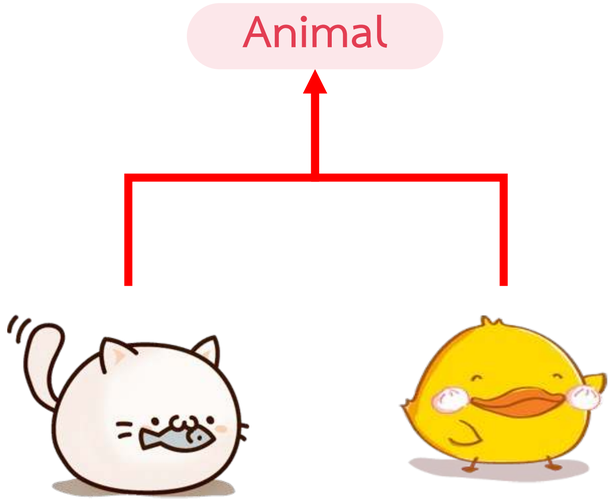
```
public class Bank{
      public int getRateOfInterest(){return 0;}
}
public class SCB extends Bank{
      public double getRateOfInterest(){return 2.5;}
}

public class KTB extends Bank{
      public double getRateOfInterest(){return 1.75;}
}
public class KBANK extends Bank{
      public double getRateOfInterest(){return 2.90;}
}
```

หัวข้อ

- การมีหลากหลายรูปแบบ (Polymorphism)
- หลักการ Upcasting และ Downcasting
- การ Overloaded เมธอด
- การ Overridden เมธอด
- **หลักการทำงานของ Dynamic Binding**
- หลักการทำงานของ Virtual Method Invocation

Dynamic Binding



โดยทั่วไปแล้ว **ตัวแปรชนิดอ้างอิง** สามารถเป็นได้เพียงชนิดเดียว (Data type) และไม่สามารถปรับเปลี่ยนได้ ซึ่งเป็นลักษณะการทำงานแบบ **Static Binding**

- **Static Binding** คือ ชนิดข้อมูล (Data type) ของตัวแปรชนิดอ้างอิง (Object) จะถูกกำหนดก็ต่อเมื่อ**โปรแกรมถูก Compile**

อย่างไรก็ตาม การที่ตัวแปรชนิดอ้างอิงสามารถย้ายการอ้างอิงจากวัตถุเดิมไปสู่วัตถุใหม่ได้ กรณีเป็นคลาสแม่ลูกกัน

- **Dynamic Binding** คือ ชนิดข้อมูล (Data type) ของตัวแปรชนิดอ้างอิง (Object) จะถูกกำหนดก็ต่อเมื่อ**โปรแกรมประมวลผล (runtime)**

การมีหลากหลายรูปแบบ (Polymorphism)

Dynamic Binding

เป็นหลักการของการสร้างวัตถุของคลาสที่มีการสืบทอดกันได้หลากหลายรูปแบบ ตัวอย่างการประกาศให้วัตถุของคลาสใด ๆ

```
Superclass obj;
```

ซึ่งสามารถสร้างได้ 2 แบบ ได้แก่

(1) คลาสนั้นเอง

```
obj = new Superclass();
```

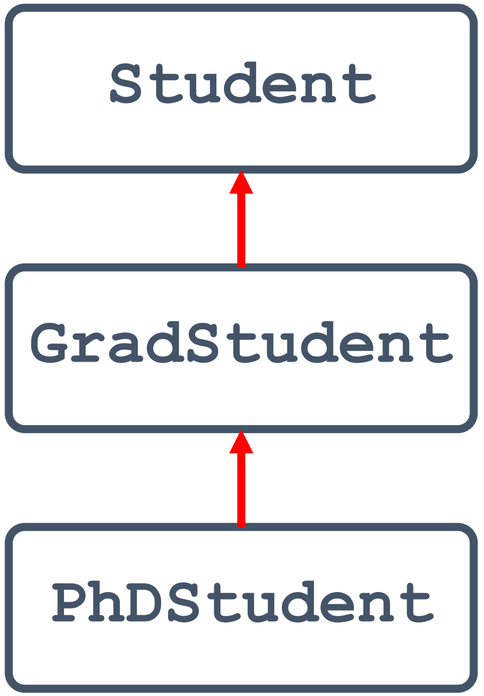
แม่สร้าง แม่

(2) คลาสอื่น ๆ ที่เป็นคลาสลูก (subclass)

```
obj = new Subclass();
```

แม่สร้าง ลูก

การมีหลากหลายรูปแบบ (Polymorphism)



ตัวอย่าง การประกาศ **Student** แล้วสร้างวัตถุเป็น **Student** หรือ **GradStudent** หรือ **PhDStudent**

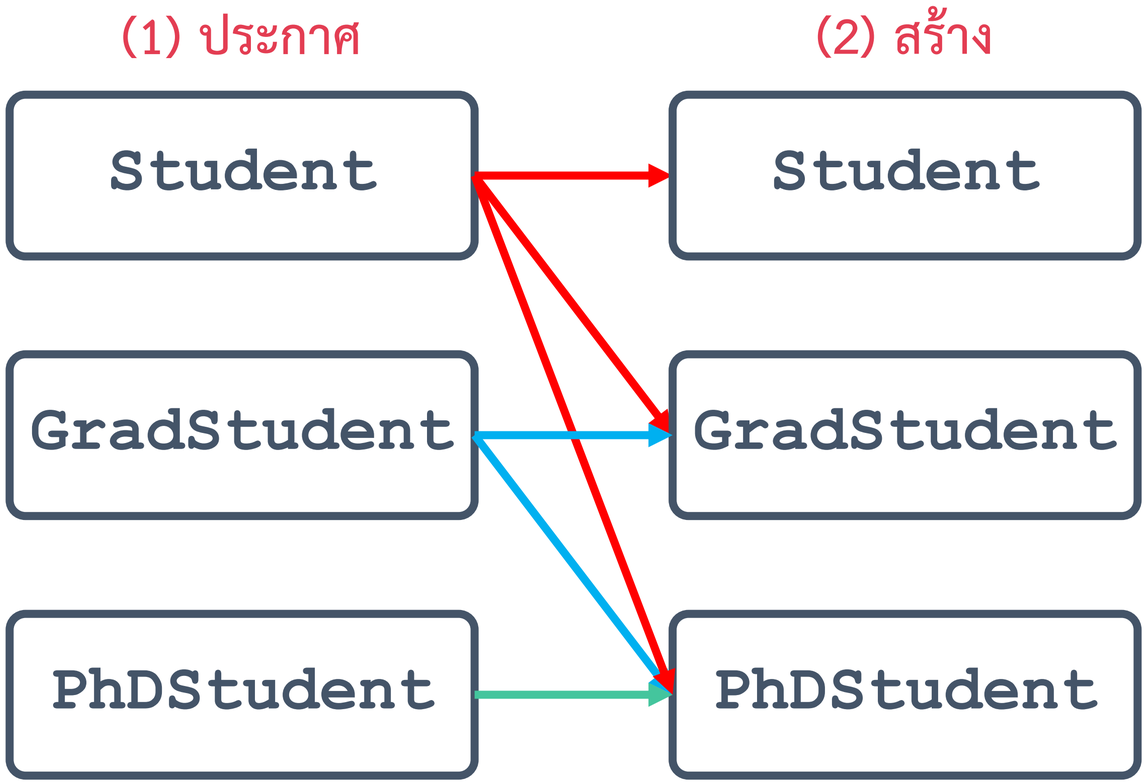
```
Student stu1 = new Student();
Student stu1 = new GradStudent();
Student stu1 = new PhDStudent();
```

เช่นเดียวกับ การประกาศ **GradStudent** แล้วสร้างวัตถุเป็น **GradStudent** หรือ **PhDStudent**

```
GradStudent stu2 = new GradStudent();
GradStudent stu2 = new PhDStudent();
```

**อย่างไรก็ตาม ภาษาจาวาไม่สามารถประกาศ Subclass แต่สร้างวัตถุเป็น Superclass ได้

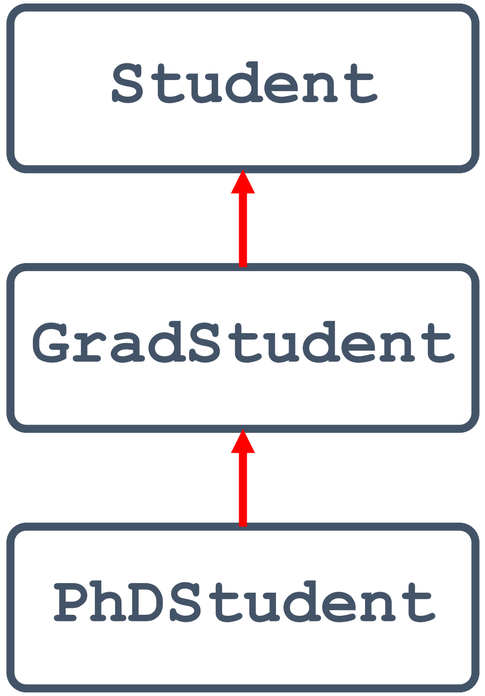
การมีหลากหลายรูปแบบ (Polymorphism)



ตัวอย่างวัตถุของคลาสที่มีได้หลากหลายรูปแบบ

การส่งผ่าน argument ได้หลายรูปแบบ

ในกรณีที่เมธอดมี argument เป็นข้อมูลชนิดคลาส เราสามารถที่จะส่งอ็อบเจกต์ของคลาสที่เป็น subclass ของคลาสนั้นแทนได้ ตัวอย่างเช่น




```
Student s1 = new Student();
GradStudent s2 = new GradStudent();
```

**** เราสามารถที่จะเรียกใช้เมธอด **printInfo(Student s)** ได้หลายรูปแบบดังนี้ *****

```
printInfo(s1)
printInfo(s2)
```

ตัวอย่างที่ 1 Dynamic Binding

```
public class Student{
        protected String id, name;
        public void printInfo(Student s) {
                System.out.println("ID "+s.id);
                System.out.println("Name "+s.name);
        }
}
public class GradStudent extends Student{
        protected String thesisTitle;
}
public class Main {
    public static void main(String[] args) {
            Student s1 = new Student();
            Student s2 = new Student();
            GradStudent g1 = new GradStudent();
            s1.printInfo(s2);
            s1.printInfo(g1);
    }
}
```

ตัวอย่างที่ 2 Dynamic Binding

```
public class Fruit {
    private int energy;
    public void setEnergy(int energy) { this.energy = energy;  }
    public int getEnergy() { return this.energy;  }
}
public class Apple extends Fruit {
    public Apple() {
        this.setEnergy(10);
    }
}
public class Orange extends Fruit {
    public Orange() {
        this.setEnergy(15);
   }
}
public class Banana extends Fruit{
    public Banana() {
        this.setEnergy(20);
    }
}
```

ตัวอย่างที่ 2 Dynamic Binding

```
public class Dog {
    private int energy;
    public int getEnergy() {    return energy;   }
    public void setEnergy(int energy) {   this.energy = energy;    }
    public void eat(Fruit f){
        this.setEnergy(this.getEnergy() + f.getEnergy());
    }
}
public class Main {
    public static void main(String[] args) {
        Dog d1 = new Dog();
        Apple a = new Apple();
        Orange o = new Orange();
        Banana b = new Banana();
        System.out.println("Dog's Energy is "+d1.getEnergy());        d1.eat(a);
        System.out.println("Dog's Energy is "+d1.getEnergy());        d1.eat(o);
        System.out.println("Dog's Energy is "+d1.getEnergy());        d1.eat(b);
        System.out.println("Dog's Energy is "+d1.getEnergy() + " after ate " + b.getName());
    }
}
```

หัวข้อ

- การมีหลากหลายรูปแบบ (Polymorphism)
- หลักการ Upcasting และ Downcasting
- การ Overloaded เมธอด
- การ Overridden เมธอด
- หลักการทำงานของ Dynamic Binding
- หลักการทำงานของ Virtual Method Invocation

Virtual Method Invocation

- โปรแกรมภาษาจาวาพิจารณา**เรียกใช้เมธอดจากชนิดของอ็อบเจกต์ที่สร้างขึ้น** ตัวอย่างเช่น

```
Student s1 = new GradStudent();
s1.printDetail();
```

เป็นคำสั่งที่เรียกใช้เมธอด printDetail() ของคลาส GradStudent ไม่ใช่เมธอดของคลาส Student

- **แต่คอมไพเลอร์ของภาษาจาวาจะไม่อนุญาตให้เรียกใช้เมธอดใด ๆ ก็ตามที่ไม่มีการประกาศอยู่ในเมธอดของ superclass ที่กำหนดไว้** ตัวอย่างเช่น

```
Student s1 = new GradStudent();
s1.getThesisTitle();    // Fail to compile
```

ตัวอย่างที่ 1

```
public class Vehicle{
  public void move(){
      System.out.println("Move in Vehicle");
  }
}
public class Car extends Vehicle{
  public void move(){
      System.out.println("Move in Car");
  }
}
public  class Main {
  public static void main(String[] args) {
    Vehicle v1 = new Car();
    v1.move();
    Car c1 = new Car();
    c1.move();
  }
}
```

ผลลัพธ์

```
Move in Car
Move in Car
```

ตัวอย่างที่ 2

```
public class Vehicle{
  public void move(){     System.out.println("Move in Vehicle"); }
}

public class Car extends Vehicle{
  public void move(){     System.out.println("Move in Car");  }
  public void move2(){    System.out.println("Move in Car");  }
}

public  class Main {
  public static void main(String[] args) {
    Vehicle v1 = new Car();
    v1.move2();
    Car c1 = new Car();
    c1.move2();
  }
}
```

ผลลัพธ์

```
Main.java:4: error: cannot find symbol
    v1.move2();
      ^
symbol:   method move2()
location: variable v1 of type Vehicle
1 error
```

การ Casting Object

การ Casting Object จึงถูกนำมาใช้งานสำหรับกรณีนี้

```
(ClassName) ObjectName
```

เช่น

```
Vehicle v1 = new Car();
((Car)v1).move2();
```

ตัวอย่างที่ 3

```
public class Vehicle{
  public void move(){       System.out.println("Move in Vehicle");    }
}

public class Car extends Vehicle{
  public void move(){       System.out.println("Move in Car");        }
  public void move2(){      System.out.println("Move in Car");        }
}

public  class Main {
  public static void main(String[] args) {
    Vehicle v1 = new Car();
    ((Car)v1).move2();
    Car c1 = new Car();
    c1.move2();
  }
}
```

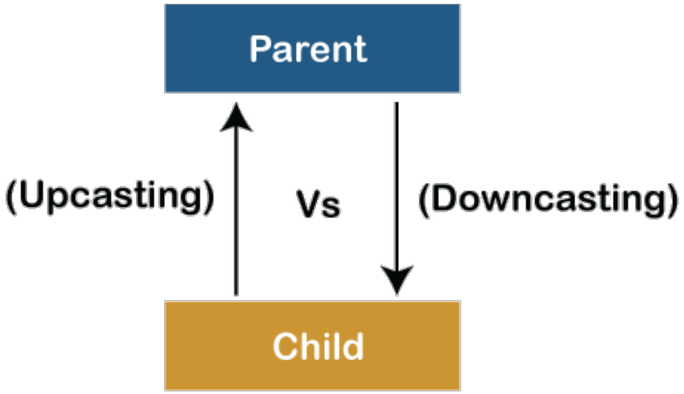
ผลลัพธ์

```
Move in Car
Move in Car
```

หัวข้อ

- การมีหลากหลายรูปแบบ (Polymorphism)
- **หลักการ Upcasting และ Downcasting**
- การ Overloaded เมธอด
- การ Overridden เมธอด
- หลักการทำงานของ Dynamic Binding
- หลักการทำงานของ Virtual Method Invocation

หลักการ Upcasting และ Downcasting



Upcasting คือการแปลงชนิดข้อมูล (typecasting) รูปแบบหนึ่งที่พยายามแปลงออปเจ็คจากคลาสลูก (Child Object) ไปเป็นออปเจ็คของคลาสแม่ (Parent Class Object) ซึ่งการทำ Upcasting นิยมนำมาใช้เมื่อต้องการเข้าถึงแอททริบิวหรือและเมธอดของคลาสลูก อย่างไรก็ตาม นักศึกษาไม่สามารถเข้าถึงแอททริบิวหรือและเมธอดทั้งหมดได้ มีสิทธิ์เข้าถึงเพียงแอททริบิวหรือและเมธอดของคลาสลูกเท่านั้น

```
Parent a = new Child();
```

ตัวอย่างการ Upcasting

```
public class Main{
      public static void main(String args[]){
          Parent obj1 = new Child();
           obj1.printData();
           // obj1.sayHi();
           // System.out.println(obj1.num);
      }
}
public class Parent{
   public void printData() {  System.out.println("parent class");  }
}

public class Child extends Parent {
   public int num;
   public void printData() {  System.out.println("child class");   }
   public void sayHi() {      System.out.println("Hi");            }
}
```

กรณีไม่มี Comment

```
/Main.java:6: error: cannot find symbol
          obj1.sayHi();
  symbol:   method sayHi()
  location: variable obj1 of type Parent
/Main.java:7: error: cannot find symbol
          System.out.println(obj1.num);
  symbol:   variable num
  location: variable obj1 of type Parent
```

กรณี Comment

```
child class
```

หลักการ Upcasting และ Downcasting

Downcasting คือการแปลงชนิดข้อมูล (typecasting) รูปแบบหนึ่งที่พยายามแปลงออปเจ็คจากคลาสแม่ (Parent Class Object) ไปเป็นออปเจ็คของคลาสลูก (Child Object) ซึ่งการทำ downcasting จะไม่เกิดข้อผิดพลาดใน Compile-Time

```
Parent p = new Parent();
Child c = (Child) p;
```

ตัวอย่างการ Downcasting

```
public class Main{
    public static void main(String[] args) {
        Parent p = new Child();
        p.name = "Taravichet";
        Child c = (Child)p;
        c.age = 18;
        System.out.println(c.name + " " +c.age);
        c.showMessage();
    }
}
public class Parent {
    public String name;
    public void showMessage() { System.out.println("Parent method");   }
}
public class Child extends Parent {
    public int age;
    @Override
    public void showMessage() {  System.out.println("Child method");   }
}
```

```
run:
Taravichet 18
Child method
BUILD SUCCESSFUL (total time: 0 seconds)
```

ตัวอย่างการ Upcasting และ Downcasting

```
public class Animal{…}
public class Bird extends Animal {…}
public class Main{
     public static void main(String args[]){
          // Upcasting
          Animal a = new Bird();

          // Downcasting
          Animal a1 = new Animal();
          Bird b = (Bird) a1;
     }
}
```

Dependency Injection

Dependency Non-Injection

```
public class App {
      private DatabaseThingie myDatabase;
      public App() { myDatabase = new DatabaseThingie(); }
      public void doSomething() { myDatabase.getData();  }
}
```

Dependency Injection

```
public class App {
  private DatabaseThingie myDatabase;
  public App() { myDatabase = new DatabaseThingie();  }
  public App(DatabaseThingie useThisDatabaseInstead) {
    myDatabase = useThisDatabaseInstead;
  }
  public void doSomething() { myDatabase.getData(); }
}
```

Dependency Injection

```
public class AppTest {
  public void testDoSomething() {
    MockDatabase mockDatabase = new MockDatabase();

    // MockDatabase is a subclass of DatabaseThingie
    // "inject" it here:
    App app = new App(mockDatabase);

    app.doSomething();
  }
}
```

```
public class App {
  private DatabaseThingie myDatabase;
  public App() { myDatabase = new DatabaseThingie();  }
  public App(DatabaseThingie useThisDatabaseInstead) {
    myDatabase = useThisDatabaseInstead;
  }
  public void doSomething() { myDatabase.getData(); }
}
```

ดังนั้น นักศึกษาสามารถแก้ไขได้โดย

- สร้าง Constructor ที่รองรับรองรับพารามิเตอร์ DatabaseThingie เพิ่ม
- กำหนดค่าแอททริบิวผ่านเมธอด (setter)

สิ่งนี้ทำให้ทั้งสองคลาสลดการผูกมัดกันลงและทำให้สะดวกต่อการทดสอบการทำงานของแต่ละส่วน