Object-Oriented Programming (OOP)
การเขียนโปรแกรมเชิงวัตถุ

SC363204
Java Web Application Development
การพัฒนาโปรแกรมประยุกต์บนเว็บด้วยภาษาจาวา

Jakkrit Kaewyotha, Ph.D.
College of Computing, Khon Kaen University

Class and Object

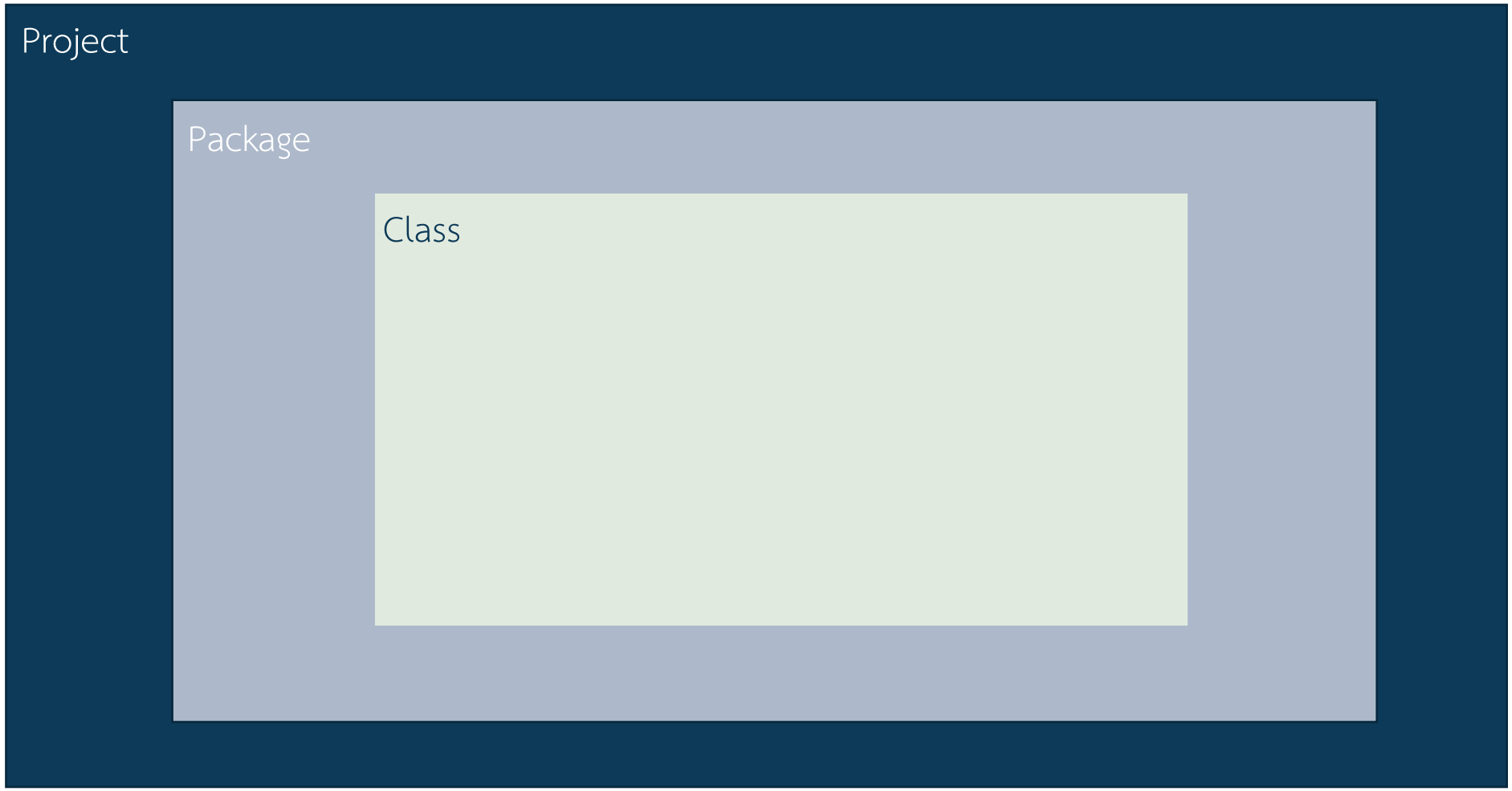
- หลักการเขียนโปรแกรมเชิงวัตถุ คือ มองทุกสิ่งทุกอย่างในโปรแกรมให้เป็นวัตถุ (object)
- ตัวอย่างระบบบัญชีเงินเดือน (Payroll System)

Class

Object

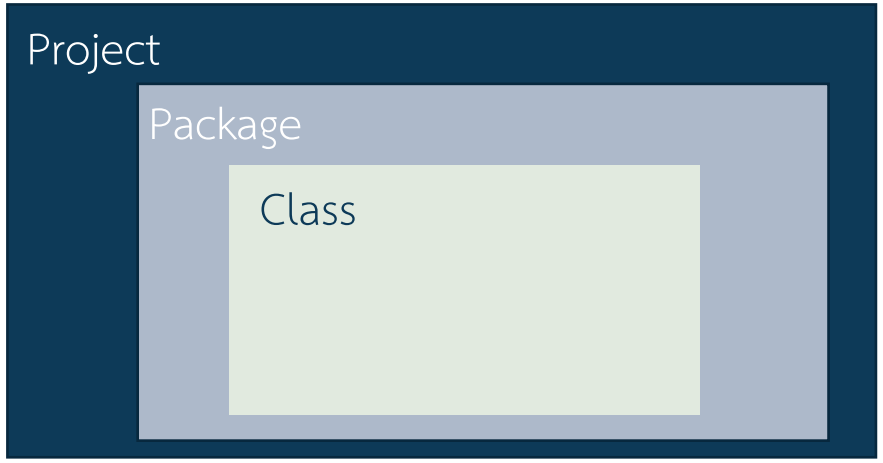
ชื่อ: แผนก: เงินเดือน: OT:	อนันต์ พนักงานทั่วไป 25000 2500
รวม: ผลลัพธ์:	25000 + 2500 = 27500 ชื่อ:อนันต์ แผนก: พนักงานทั่วไป เงินเดือน: 25000 OT: 2500 รวม: 27500

Class and Object



Class and Object

- **Package**: เป็นกลุ่มของ class หรือ library มาตรฐานของภาษา Java ที่มีฟังก์ชันต่างๆ
- **Class**: ในส่วนของการประกาศคลาส **จะต้องประกาศคลาสให้ชื่อตรงกับไฟล์เสมอ** โดยชื่อคลาสนั้น **ควรจะขึ้นต้นด้วยตัวใหญ่** และถ้ามีหลายคำให้ใช้ตัวพิมพ์ใหญ่แบ่ง
- **Method**: หลังจากคลาสสร้างแล้ว จะเป็นประกาศเมธอดภายในคลาส โดยในการที่จะรันโปรแกรมได้ **จะต้องมีเมธอดที่ชื่อว่า main**
- **Statements**: เป็นคำสั่งของโปรแกรมเพื่อให้โปรแกรมทำงานตามต้องการ



Class and Object

Employee.java

```
package com.payroll;

public class Employee {
//ประกาศตัวแปร
  public String name;
     public String department;
     public int salary;
     public int ot;
     }
// method คำนวณเงินเดือน
  public int calculateSalary() {
         return salary + ot;
     }
// method แสดงรายละเอียด
  public String display() {
         return "ชื่อ:" + name +
         ", แผนก:" + department +
         ", เงินเดือน" + salary +
         ", OT" + ot +
         ", Total" + calculateSalary();
     }
```

Object

ชื่อ:	อนันต์
แผนก:	พนักงานทั่วไป
เงินเดือน:	25000
OT:	2500
รวม:	25000 + 2500 = 27500
ผลลัพธ์:	ชื่อ:อนันต์ แผนก: พนักงานทั่วไป เงินเดือน: 25000 OT: 2500 รวม: 27500

ชื่อ:	นิมมาน
แผนก:	พนักงานทั่วไป
เงินเดือน:	30000
OT:	0
รวม:	30000 + 0 = 30000
ผลลัพธ์:	ชื่อ:นิมมาน แผนก: พนักงานทั่วไป เงินเดือน: 30000 OT: 0 รวม: 30000

ชื่อ:	จอมขวัญ
แผนก:	พนักงานทั่วไป
เงินเดือน:	45000
OT:	0
รวม:	45000 + 0 = 45000
ผลลัพธ์:	ชื่อ:จอมขวัญ แผนก: พนักงานทั่วไป เงินเดือน: 45000 OT: 0 รวม: 45000

Class and Object

- **รูปแบบการสร้าง Object**

ชื่อคลาส ชื่อ object = new ชื่อคลาส();

Ex. Employee Obj1 = new Employee();

Employee.java

```
public class Employee {
//ประกาศตัวแปร
 public String name;
    public String department;
    public int salary;
    public int ot;

// ...      }
```

TestObject.java

ชื่อ:	อนันต์
แผนก:	พนักงานทั่วไป
เงินเดือน:	25000
OT:	2500
รวม:	25000 + 2500 = 27500
ผลลัพธ์:	ชื่อ:อนันต์ แผนก: พนักงานทั่วไป เงินเดือน: 25000 OT: 2500 รวม: 27500

Class and Object

- รูปแบบการสร้าง Object

  ชื่อคลาส ชื่อ object = new ชื่อคลาส();

Ex. Employee somsak = new Employee();

Employee.java

```
public class Employee {
//ประกาศตัวแปร
 public String name;
    public String department;
    public int salary;
    public int ot;

// ...      }
```

TestObject.java

ชื่อ:	สมศักดิ์
แผนก:	พนักงานทั่วไป
เงินเดือน:	25000
OT:	2500
รวม:	25000 + 2500 = 27500
ผลลัพธ์:	ชื่อ: สมศักดิ์ แผนก: พนักงานทั่วไป เงินเดือน: 25000 OT: 2500 รวม: 27500

```
package com.payroll;

public class TestObj {

public static void main(String[] args) {

        Employee obj1 = new Employee();
        obj1.name = "สมศักดิ์";
   obj1.department = "พนักงานทั่วไป";
   obj1.salary = 25000;
        obj1.ot = 2500;
        System.out.println(obj1.display());
}
}
```

Class and Object

Employee.java

```
package com.payroll;

public class Employee {
//ประกาศตัวแปร
  public String name;
     public String department;
     public int salary;
     public int ot;
     }
// method คำนวณเงินเดือน
  public int calculateSalary() {
         return salary + ot;
     }
// method แสดงรายละเอียด
  public String display() {
         return "ชื่อ:" + name +
         ", แผนก:" + department +
         ", เงินเดือน" + salary +
         ", OT" + ot +
         ", Total" + calculateSalary();
     }
```

TestObject.java

ชื่อ: แผนก: เงินเดือน: OT:	อนันต์ พนักงานทั่วไป 25000 2500
รวม: ผลลัพธ์:	25000 + 2500 = 27500 ชื่อ:อนันต์ แผนก: พนักงานทั่วไป เงินเดือน: 25000 OT: 2500 รวม: 27500

ชื่อ: แผนก: เงินเดือน: OT:	นิมมาน พนักงานทั่วไป 30000 0
รวม: ผลลัพธ์:	30000 + 0 = 30000 ชื่อ:นิมมาน แผนก: พนักงานทั่วไป เงินเดือน: 30000 OT: 0 รวม: 30000

ชื่อ: แผนก: เงินเดือน: OT:	จอมขวัญ พนักงานทั่วไป 45000 0
รวม: ผลลัพธ์:	45000 + 0 = 45000 ชื่อ:จอมขวัญ แผนก: พนักงานทั่วไป เงินเดือน: 45000 OT: 0 รวม: 45000

Constructor

- การทำงานบางเมธอดในคลาส จำเป็นต้องกำหนดค่าเริ่มต้นให้กับบาง attribute ก่อนเรียกใช้เมธอด
- ช่วยในการสร้าง object และการกำหนดค่าให้กับตัวแปรใน object ภายในบรรทัดเดียว

Employee.java

```
public class Employee {
//เพิ่มคำสั่ง
     public Employee(){
    }

// ...      }
```

- หากไม่เขียน constructor ใดๆไว้ default constructor จะถูกสร้างโดยอัตโนมัติ
- แต่ถ้าหากเขียน constructor แบบใดก็ตามไว้ default constructor จะไม่ถูกสร้างอัตโนมัติ นักพัฒนาจะต้องเขียน default constructor เอง

TestConstructor.java

ชื่อ:	อนันต์
แผนก:	พนักงานทั่วไป
เงินเดือน:	25000
OT:	2500
รวม:	25000 + 2500 = 27500
ผลลัพธ์:	ชื่อ:อนันต์ แผนก: พนักงานทั่วไป เงินเดือน: 25000 OT: 2500 รวม: 27500

```
package com.payroll;

public class TestConstructor {
    public static void main(String[] args) {

        Employee obj1 = new Employee "อนันต์","พนักงานทั่วไป",25000,2500);
   System.out.println(obj1.display());

    }
}
```

Constructor

Object

ชื่อ:	อนันต์
แผนก:	พนักงานทั่วไป
เงินเดือน:	25000
OT:	2500
รวม:	25000 + 2500 = 27500
ผลลัพธ์:	ชื่อ:อนันต์ แผนก: พนักงานทั่วไป เงินเดือน: 25000 OT: 2500 รวม: 27500

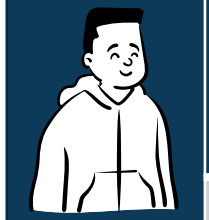
ชื่อ:	นิมมาน
แผนก:	พนักงานทั่วไป
เงินเดือน:	30000
OT:	0
รวม:	30000 + 0 = 30000
ผลลัพธ์:	ชื่อ:นิมมาน แผนก: พนักงานทั่วไป เงินเดือน: 30000 OT: 0 รวม: 30000

ชื่อ:	จอมขวัญ
แผนก:	พนักงานทั่วไป
เงินเดือน:	45000
OT:	0
รวม:	45000 + 0 = 45000
ผลลัพธ์:	ชื่อ:จอมขวัญ แผนก: พนักงานทั่วไป เงินเดือน: 45000 OT: 0 รวม: 45000

Constructor

Object

ชื่อ:	อนันต์
แผนก:	พนักงานทั่วไป
เงินเดือน:	25000
OT:	2500
รวม:	25000 + 2500 = 27500
ผลลัพธ์:	ชื่อ:อนันต์ แผนก: พนักงานทั่วไป เงินเดือน: 25000 OT: 2500 รวม: 27500



ชื่อ:	นิมมาน
แผนก:	พนักงานทั่วไป
เงินเดือน:	30000
OT:	0
รวม:	30000 + 0 = 30000
ผลลัพธ์:	ชื่อ:นิมมาน แผนก: พนักงานทั่วไป เงินเดือน: 30000 OT: 0 รวม: 30000

ชื่อ:	จอมขวัญ
แผนก:	พนักงานทั่วไป
เงินเดือน:	45000
OT:	0
รวม:	45000 + 0 = 45000
ผลลัพธ์:	ชื่อ:จอมขวัญ แผนก: พนักงานทั่วไป เงินเดือน: 45000 OT: 0 รวม: 45000

ถ้าหากไม่ต้องการกำหนดค่าให้กับตัวแปร OT หรือไม่รับค่า ให้มีค่าเริ่มต้นเป็น 0 จะเขียน Constructor อย่างไร

Constructor

การใช้ Constructor และไม่ใช้ Constructor

```
package com.payroll;

public class TestObj {

public static void main(String[] args) {

        Employee obj1 = new Employee();
        obj1.name = "อนันต์";
        obj1.department = "พนักงานทั่วไป";
        obj1.salary = 25000;
        obj1.ot = 2500;
        System.out.println(obj1.display());
}
}
```

```
package com.payroll;

public class TestConstructor {
    public static void main(String[] args) {

        Employee obj1 = new Employee "อนันต์","พนักงานทั่วไป",25000,2500);
        System.out.println(obj1.display());

    }
}
```

Encapsulation

- เป็นความสามารถของการเขียนแบบ OOP ในการซ่อนข้อมูล (Information hiding) ที่อยู่ในคลาสเพื่อปกป้องการเข้าถึงจากคลาสอื่น ๆ
- การห่อหุ้มเกิดขึ้นเมื่อมีการใช้คำสั่งควบคุมระดับการเข้าถึง (Access Modifier)
- Information hiding และ Encapsulation จะต้องทำควบคู่กันเสมอ

private

- จะถูกเรียกใช้ได้เฉพาะ ภายในคลาสเท่านั้น
- มักใช้กับ attribute
- มักใช้กับเมธอดที่ใช้เฉพาะภายในคลาส ไม่ต้องการให้คลาสอื่นเรียกได้

public

- ถูกเรียกใช้ได้ทั้งจากภายนอกหรือภายในคลาสได้
- มักใช้กับเมธอดของคลาส เช่น เมธอดกำหนดค่าและขอค่า attribute

ทำไมต้องห่อหุ้ม

- ลดความความผิดพลาดจากการทำงาน ที่อาจเกิดจากการกำหนดค่าด้วยผู้ใช้เอง
- เพื่อให้ข้อมูลถูกจัดการ โดยเมธอดภายในคลาสเท่านั้น
- สามารถตรวจสอบความถูกต้องของข้อมูล (validate data) ได้
- นักพัฒนาเรียกใช้โดยไม่จำเป็นต้องรู้โครงสร้างภายในคลาส เพราะสิ่งที่ต้องการคือผลลัพธ์เท่านั้น
- สามารถควบคุมทิศทางของข้อมูลได้ เช่น อ่านอย่างเดียว (read-only), กำหนดค่าได้อย่างเดียว (write-only)

Employee.java

```
public class Employee {
//ประกาศตัวแปร
   private String name;
     private String department;
     private int salary;
     private int ot;

// ...      }
```

```
1  package jw;
2
3  public class TestObj {
4
5      public static void main(String[] args) {
6
7          Employee obj1 = new Employee();
8          obj1.name = "สมศักดิ์";
9          obj1.department = "พนักงานทั่วไป";
10         obj1.salary = -25000;
11         obj1.ot = 2500;
12         System.out.println(obj1.display());
13 }
14 }
```

Getter & Setter

เมธอดอ่านค่า (Getter)

- เมื่อ attribute ถูกห่อหุ้มด้วย private แล้ว หากต้องการให้ผู้ใช้อ่านค่าได้จะต้องสร้างเมธอดสำหรับใช้ในการอ่าน/ดึงค่าจาก attribute ของคลาส

- รูปแบบ

```
public String get...() {
        return ...;
```

- ตัวอย่าง

```
public String getName() {
        return name;
```

Getter & Setter

เมธอดกำหนดค่า (Setter)

- เมื่อ attribute ถูกห่อหุ้มด้วย private แล้ว หากต้องการให้ผู้ใช้กำหนดค่าให้กับ attribute จะต้องสร้างเมธอดสำหรับใช้ในการกำหนดค่า

- รูปแบบ

```
public void set...(String ตัวแปรรับค่า) {
ชื่อ attribute = ตัวแปรรับค่า;
}
```

- ตัวอย่าง

```
public void setName(String name) {
this.name = name;
}
```

Getter & Setter

```
public class Employee {

private String name;
private int salary;

public String getName() {
return name;
}
public void setName(String name) {
this.name = name;
}
public int getSalary() {
return salary;
}
public void setSalary(int salary) {
this.salary = salary;
}
}
```

การใช้ this ในกรณีที่ชื่อตัวแปรพารามิเตอร์ชื่อเดียวกับตัวแปรในคลาส จะใช้ this เพื่อบอกว่านี่คือตัวแปรของส่วนคลาส

Getter & Setter

Employee.java

```
public void setSalary(int salary) {
if (salary <= 0 )
{
System.out.println("เงินเดือนต้องมีค่าไม่ต่ำกว่า 0");
System.exit(0);
}
else {
this.salary = salary;
}
}
```

สร้างเงื่อนไข if else ในเมธอด setter เพื่อกำหนดค่าที่ให้ตรงกับเงื่อนไขที่ต้องการ

Getter & Setter

TestEncapsulation.java

```
public static void main(String[] args) {
Employee obj1 = new Employee();
obj1.setName("สมศักดิ์");
obj1.setDepartment("พนักงานทั่วไป");
obj1.setSalary(-25000);
obj1.setOt(2500);
System.out.println(obj1.display());
}
```

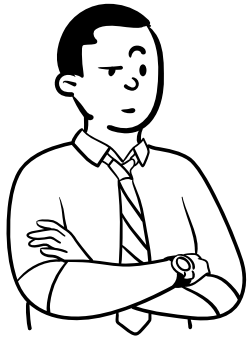
Output:

เงินเดือนต้องมีค่าไม่ต่ำกว่า 0

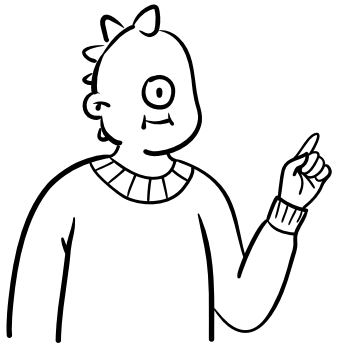
Inheritance

- การสืบทอดคลาส คือ การส่งต่อ attribute และ method จากคลาสหนึ่งไปยังคลาสใหม่
- คลาสลูก (sub class) จะได้รับ attribute และ method จากคลาสสืบทอดหรือ คลาสแม่ (Super class) มาทุกอย่าง
- การสืบทอดทำเพิ่มส่วน attribute และ method เพื่อให้ทำงานเฉพาะเจาะจงกว่าคลาสเดิมที่มีอยู่
- Reuse โค้ดให้นำกลับมาใช้ใหม่ โดยไม่ต้อง copy โค้ดเดิมไปยังคลาสใหม่

Inheritance

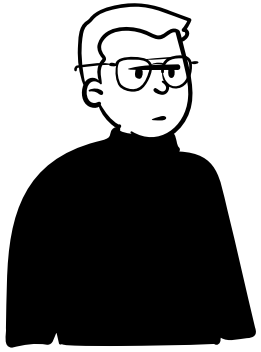


Employee
name department salary ot
methods

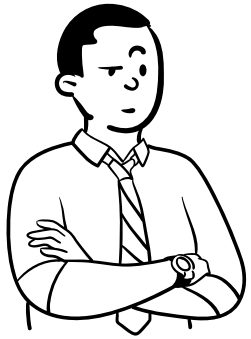


Programmer
-name = “ ” -department = “ ” -salary = 32000 -ot = 12000 -skill = 2000 (ค่าทักษะ)
methods

Project Manager
-name = “ ” -department = “ ” -salary = 55000 -ot = 0 -position = 5000 (ค่าตำแหน่ง)
methods



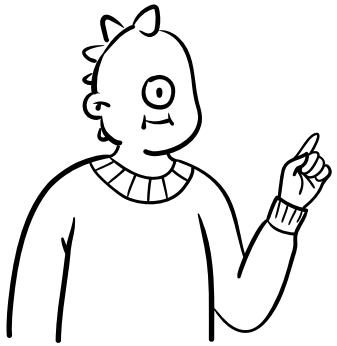
Inheritance



Employee

#name
#department
#salary
#Ot

methods



Programmer

-skill = 2000 (ค่าทักษะ)

methods

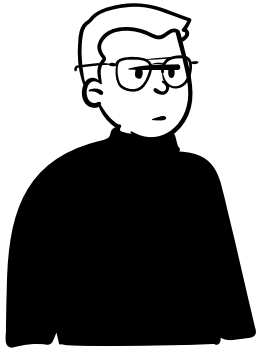
extends

extends

Project Manager

-position = 5000 (ค่าตำแหน่ง)

methods



Inheritance

รูปแบบคลาสแม่ (super class) และคลาสลูก (sub class)

Super class

```
public class ชื่อคลาสแม่ {
//ประกาศตัวแปร
        protected ชนิดตัวแปร ชื่อตัวแปร1;
        protected ชนิดตัวแปร ชื่อตัวแปร2;
//Getter Setter method
}
```

Sub class

```
public class  ชื่อคลาสลูก extends ชื่อคลาสแม่ {
        private ชนิดตัวแปร ชื่อตัวแปร1;
        private ชนิดตัวแปร ชื่อตัวแปร2;
        }
```

Inheritance

Super เป็น keyword ที่ใช้ในการเรียกใช้งานตัวแปรและเมธอดต่างๆใน super class

เช่น

```
super.name;          //เรียกใช้ตัวแปรในคลาสแม่
super(...);          // เรียกใช้คอนสตรัคเตอร์ในคลาสแม่
super.display();     //เรียกใช้เมธอดในคลาสแม่
```

Inheritance

Employee.java (superclass)

```java
public class Employee {
//ประกาศตัวแปร
 protected String name;
    protected String department;
    protected int salary;
    protected int ot;

// method...
// method คำนวณเงินเดือน
 public int calculateSalary() {
        return salary + ot;
    }

}
```

เปลี่ยนจาก private เป็น protected เพื่อให้คลาสลูกเข้าถึงได้โดยไม่ต้องเรียกผ่านเมธอด getter/setter

Programmer.java (subclass)

```java
public class Programmer extends Employee {
    private int skill;
}
```

เพิ่ม constructor จาก superclass

Programmer.java (subclass)

```java
    public Programmer(String name, String
department, int salary, int ot, int skill) {
        super(name, department, salary, ot);
        this.skill = skill;
    }
```

เรียกใช้ calculateSalary จาก superclass

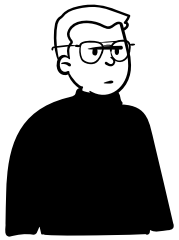
Programmer.java (subclass)

```java
    public int calculateSalary() {
        return super.calculateSalary() + skill;
    }
```

Override

- Override คือการเขียนทับเมธอดที่อยู่ในคลาสแม่ (superclass)
- คลาสลูกที่สืบทอดมาจากคลาสแม่ หากไม่ต้องการใช้สิ่งที่คลาสแม่มี และต้องการเขียนใหม่เอง สามารถทำได้โดยประกาศชื่อให้ตรงกับคลาสแม่
- การบดบังเมธอดหรือ attribute คลาสแม่เรียกว่า Override
- การ Override จะเกิดในคลาสลูกเท่านั้น
- เมธอดที่จะถูก Override ต้องมีรายการ parameter ตรงกับเมธอดแม่ด้วยจึงจะถูก Override

Override



Project Manager
-position = 5000 (ค่าตำแหน่ง)
methods

- ประกาศตัวแปร position เพื่อเก็บค่าตำแหน่ง
- สร้าง constructor จากคลาสแม่

ProjectManager.java

```java
public class ProjectManager extends Employee {
    private int position;
    public ProjectManager(String name, String department, int salary,
int ot, int position) {
        super(name, department, salary, ot);
        this.position = position;
    }
    public int calculateSalary() {
        return salary + ot + position;
    }
}
```

Copy method มาจากคลาสแม่

TestProjectManager.java

```java
public class TestProjectManager {

    public static void main(String[] args) {
        ProjectManager obj1 = new ProjectManager("ธนาธร","Project Manager",55000,0
,5000);
        System.out.println(obj1.calculateSalary());
    }
}
```

Override vs Super

Override

Super class

```
public int [illegible]() {
    retu[illegible]
}
```

Sub class

ProjectManager.java

```
public int calculateSalary() {
return salary + ot + position;
}
```

Super

Super class

Employee.java

```
public int calculateSalary() {
    return salary + ot;
}
```

Sub class

Programmer.java (subclass)

```
public int calculateSalary() {
    return super.calculateSalary() + skill;
}
```

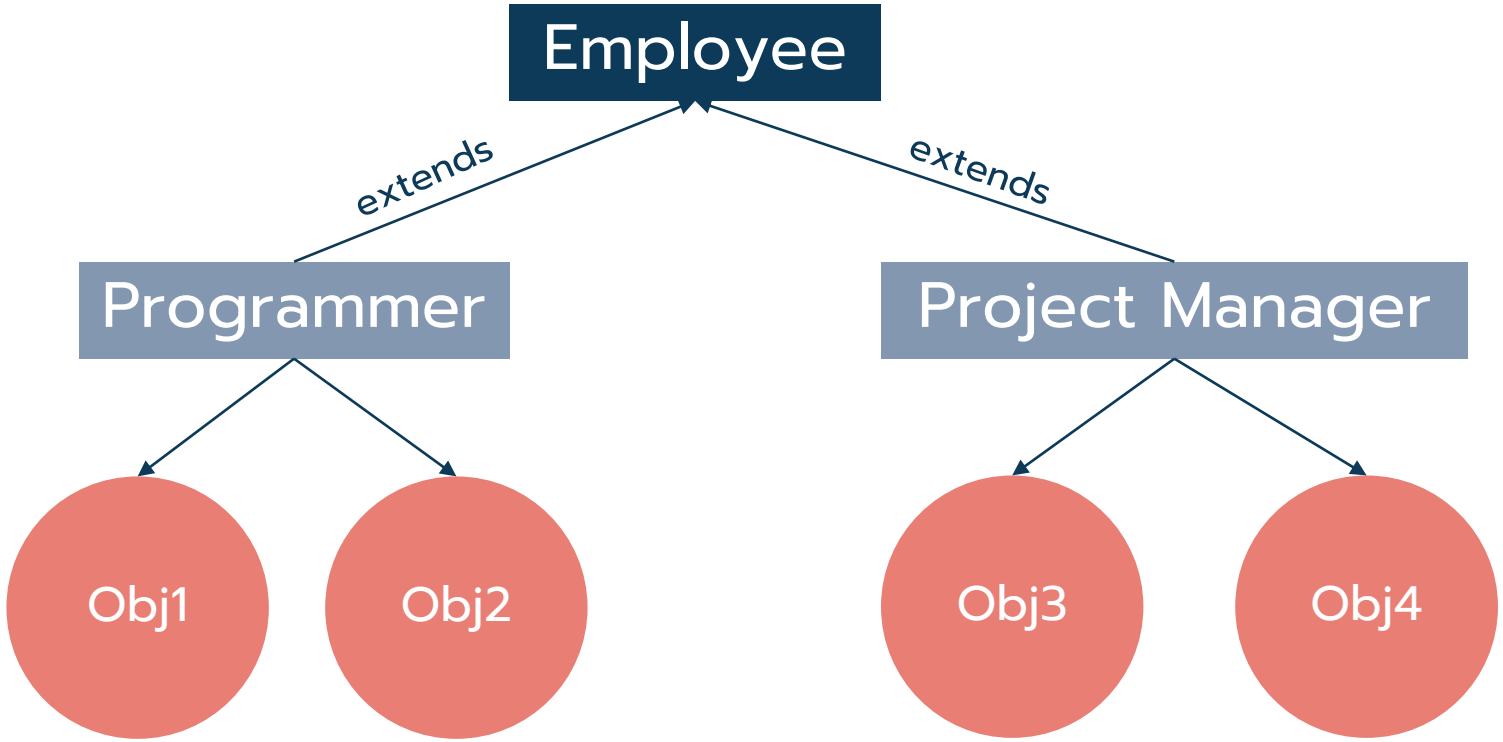
Override vs Super

- **Override** คือการเขียนทับเมธอดที่อยู่ในคลาสแม่
- **Super** คือ keyword ที่ใช้ในการเรียกใช้ส่วนต่าง ๆ ในคลาสแม่
- สามารถใช้ Override และ Super ร่วมกันได้

Polymorphism

- ความหลากหลาย (Polymorphism) เป็นคุณสมบัติที่ต่อเนื่องมาจากเรื่องการสืบทอด (Inheritance)
- ในทาง OOP รูปแบบของ object ที่สามารถมีการทำงานที่หลากหลาย ซึ่งเกิดจากการสืบทอดคลาส โดยยังคงคุณสมบัติแม่เอาไว้
- ช่วยในการรับและส่งข้อมูลไม่เฉพาะเจาะจงเฉพาะ object ของคลาสเดียวเท่านั้น อาจมีทั้งคลาสลูกหรือคลาสหลานก็ได้

Polymorphism



Polymorphism

สร้างไฟล์ TestPolymorphism และสร้าง Object ขึ้นมาใหม่ 4 objects

TestPolymorphism.java

```
public class TestPolymorphism {

    public static void main(String[] args) {
        Programmer obj1 = new Programmer("สุขใจ", "Programmer", 32000, 1200, 2000);
        Programmer obj2 = new Programmer("สถาพร", "Programmer", 42000, 1200, 3000);

        ProjectManager obj3 = new ProjectManager("ธนาธร", "Project Manager", 55000, 0, 5000);
        ProjectManager obj4 = new ProjectManager("สมหญิง", "Project Manager", 55000, 0, 5000);
    }

}
```