



Inheritance

การสืบทอด

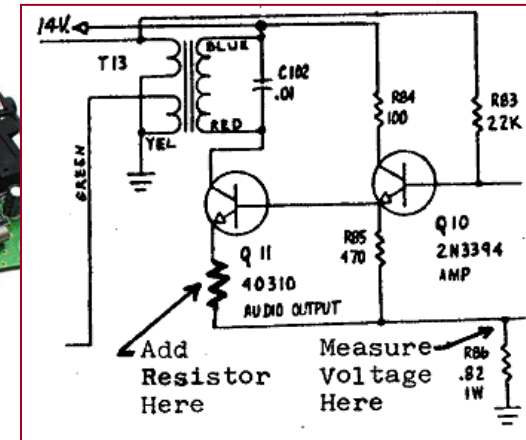
Edited by Kanjana Eiamsaard, 2022.01.04

Agenda

- Encapsulation & Class members accessibility
- UML Class Diagram notation
- การสืบทอด
- การสืบทอดโดยปริยาย

Encapsulation

- **encapsulation:** Hiding implementation details from clients.
 - Encapsulation forces *abstraction*.
 - separates external view (behavior) from internal view (state)
 - protects the integrity of an object's data



ตัวอย่างเหตุการณ์ที่เกิด Encapsulation

A field that cannot be accessed from outside the class

private type name;

– Examples:

```
private int id;  
private String name;
```

- Client code won't compile if it accesses private fields:

```
PointMain.java:11: x has private access in Point  
System.out.println(p1.x) ;  
                        ^
```

Accessing Private fields with getter/setter methods

// A "read-only" access to the x field ("accessor")

```
public int getX() {  
    return x;  
}
```

// Allows clients to change the x field ("mutator")

```
public void setX(int newX) {  
    x = newX;  
}
```

– Client code will look more like this:

```
System.out.println(p1.getX());
```

```
p1.setX(14);
```

The new better version of Point class

```
// A Point object represents an (x, y) location.
public class Point {
    private int x;
    private int y;

    public Point(int initialX, int initialY) {
        x = initialX;
        y = initialY;
    }

    public int getX() {
        return x;
    }

    public int getY() {
        return y;
    }

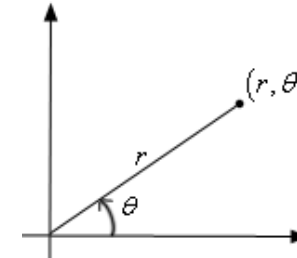
    public double distanceFromOrigin() {
        return Math.sqrt(x * x + y * y);
    }

    public void setLocation(int newX, int newY) {
        x = newX;
        y = newY;
    }

    public void translate(int dx, int dy) {
        setLocation(x + dx, y + dy);
    }
}
```

Benefits of Encapsulation

- Abstraction between object and clients
- Can change the class implementation later
 - Example: Point could be rewritten in polar coordinates (r, θ) with the same methods.
- Can constrain objects' state (**invariants**)
 - Example: Only allow Accounts with non-negative balance.
 - Example: Only allow Dates with a month from 1-12.



Data and method visibility

- Besides the **private** keyword, java also provides three other types of keywords for controlling data and method visibility:
 - **public, protected, default(package)**
- สิ่งที่ทำให้ผู้ศึกษาสับสนได้คือ visibility ถูกนำไปใช้ในหลายระดับ เช่น คลาส ตัวแปรของคลาส เมทอด

Valid Application of Visibility Modifiers

Modifier	Class	Constructor	Method	Data	block
(default)*	○	○	○	○	○
public	○	○	○	○	X
protected	X	○	○	○	X
private	X	○	○	○	X

* **default** access has no modifier associated with it



UML

Class Diagram

Object Diagram



แผนภาพคลาส

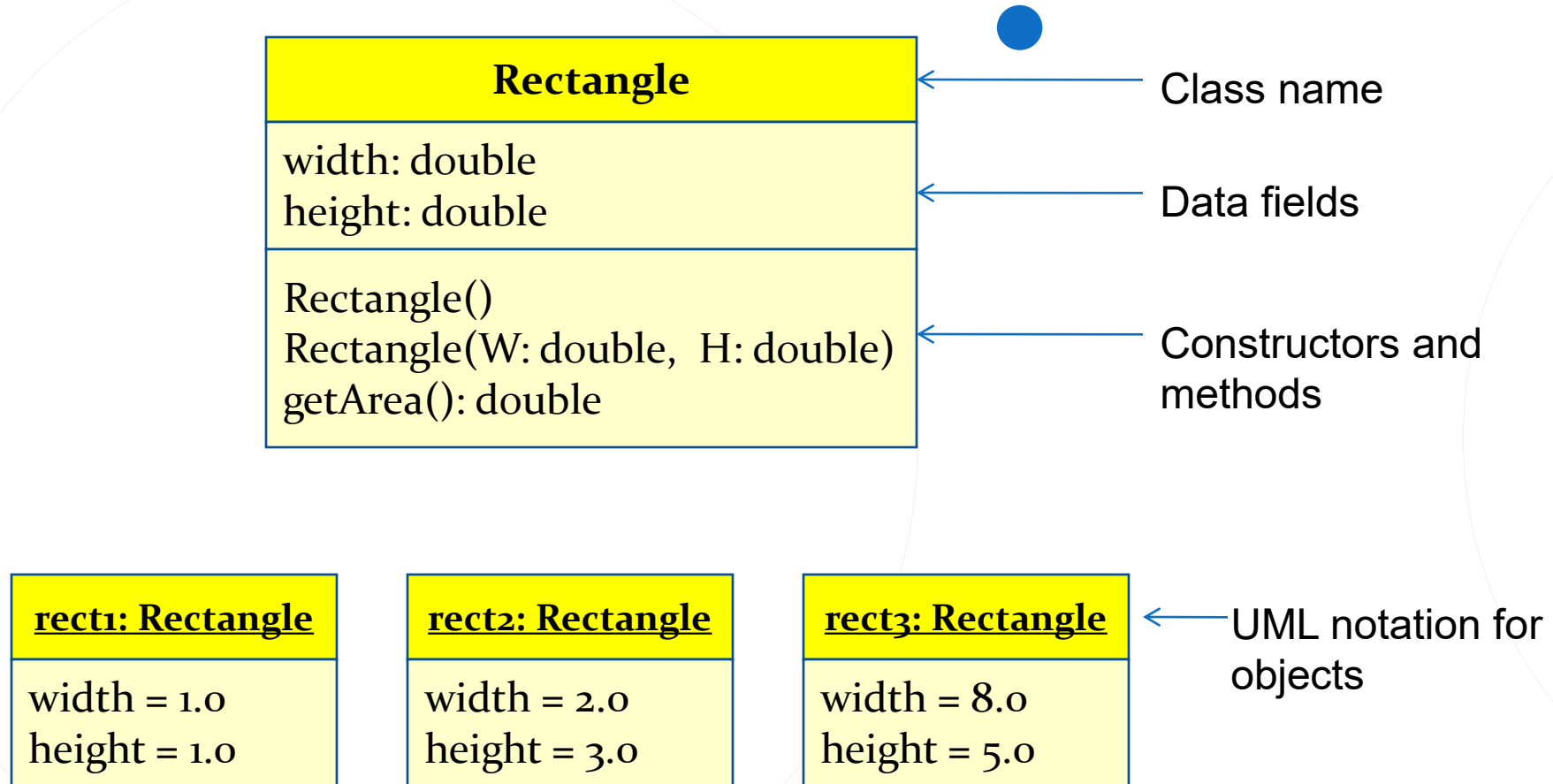
- UML (Unified Modeling Language) เป็นแผนภาพแสดงโครงสร้างและการทำงานของซอฟต์แวร์ นิยมใช้เพื่อช่วยในการออกแบบและสื่อสารแบบ
- ประกอบด้วยแผนภาพหลายชนิด ขึ้นอยู่กับความเหมาะสมในการใช้งาน
- Class Diagram คือแผนภาพชนิดหนึ่งใน UML ที่นำมาใช้เพื่อสื่อสารโครงสร้างของโปรแกรม ที่พัฒนาด้วยแนวคิดเชิงวัตถุ
- Object Diagram คือแผนภาพที่แสดงถึงรายละเอียดของวัตถุที่เกิดขึ้นในระบบคอมพิวเตอร์

เปรียบเทียบแผนภาพคลาสและโค้ด

Example
+ id : int = 10 + name : String - hiddenVar : double
+ Example() + setHiddenVar(double value) - replace(String newName) : String

```
public class Example {  
    public int id = 10;  
    public String name;  
    private double hiddenVar;  
  
    public Example() {  
        // ... code omitted  
    }  
  
    public void setHiddenVar(double value) {  
        // ... code omitted  
    }  
  
    private String replace(String newName) {  
        // ... code omitted  
    }  
}
```

UML Class & Diagram



Example: class Rectangle

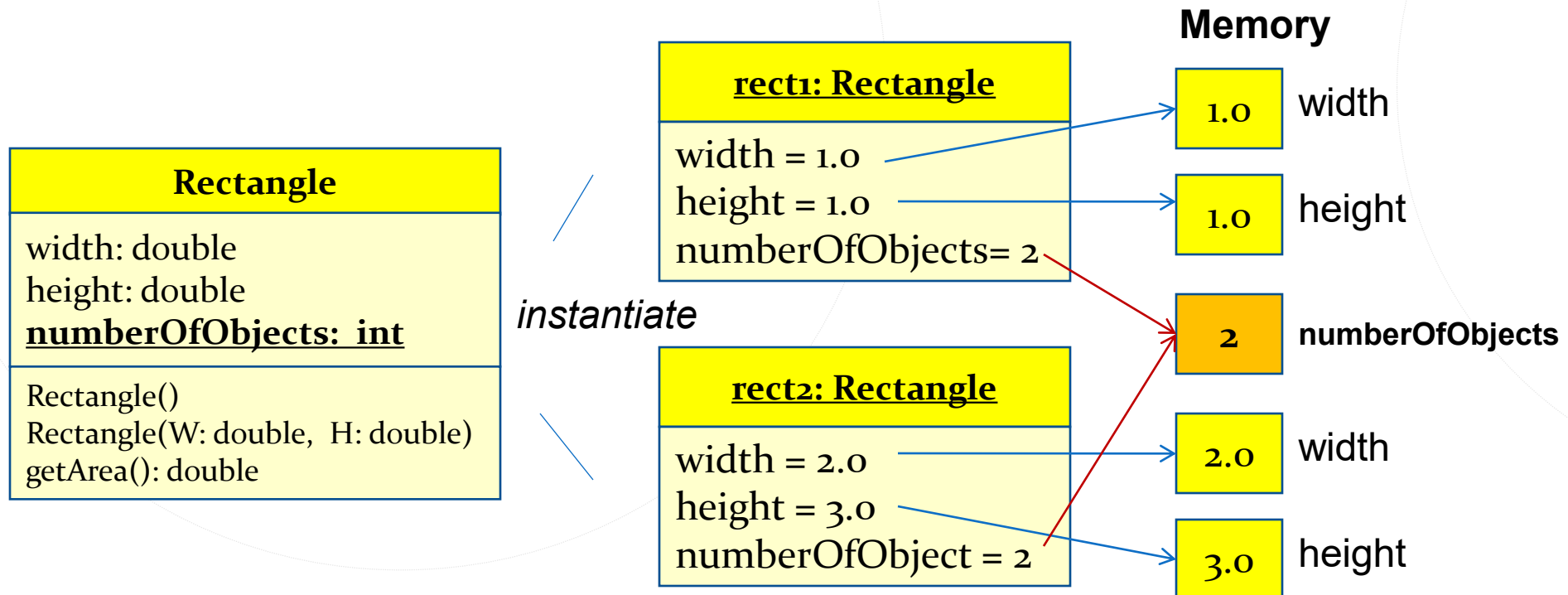
```
class Rectangle {  
    /* data fields */  
    double width = 1.0;  
    double height = 1.0;  
    /* constructors */  
    Rectangle() {  
    }  
    Rectangle( double W, double H) {  
        width = W; height = H;  
    }  
    /* methods */  
    double getArea() {  
        return width*height;  
    }  
}
```

Adding a static field and method to our class

```
class Rectangle {  
    /* data fields */  
    double width = 1.0;  
    double height = 1.0;  
    static int numberOfObjects;  
    /* constructors */  
    Rectangle() { }  
    Rectangle( double W, double H) {  
        width = W; height = H;  
    }  
    /* methods */  
    double getArea() { return width*height; }  
    static int getNumberOfObjects() {  
        return numberOfObjects;  
    }  
}
```

Instance vs. Class Fields (or Methods)

- An **instance field or method** belongs to an instance of a class.
- A **static field or method** is shared by all instances of the same class, and can be invoked without using an instance





Inheritance

การสืบทอด

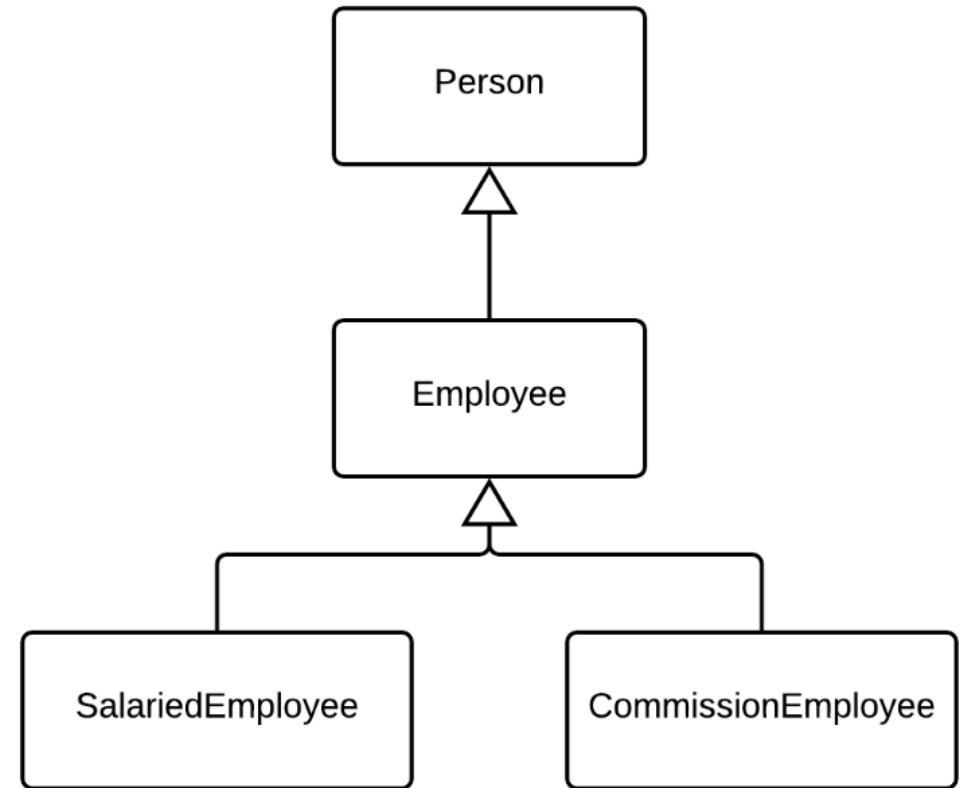


การสืบทอด

- การสร้างคลาสใหม่บนฐานของคลาสที่มีอยู่เดิม
- รับเอาคุณลักษณะและความสามารถมาจากคลาสเดิม
- อาจจะเพิ่มความสามารถใหม่หรือเปลี่ยนแปลงความสามารถเดิมที่ระบุไว้ในคลาสเดิม
- คลาสที่เป็นต้นแบบของการสืบทอด เรียก “ซูเปอร์คลาส (superclass) / คลาสพื้นฐาน (base class) หรือ คลาสแม่ (parent class) ”
- คลาสที่สืบทอดมา เรียก “ซับคลาส (subclass) / คลาสย่อย คลาสสืบท่อ (derived class) หรือ คลาสลูก (child class)”

การสืบทอด (ต่อ)

- การสืบทอดทำให้เกิดความสัมพันธ์แบบลำดับชั้นของคลาส (class hierarchy)
 - หัวลูกศรสามเหลี่ยมโป่งและเส้นเชื่อมโยงทึบ แสดงความสัมพันธ์ของการสืบทอด
 - การสืบทอดโดยตรง เช่น Person-Employee
 - การสืบทอดโดยอ้อม เช่น Person-SalariedEmployee



การสืบทอด (ต่อ)

- Constructors เป็นเมทอดพิเศษที่ไม่ถ่ายทอดไปยัง subclass
 - แต่คำสั่ง super ถูกใช้เมื่อ subclass ต้องการเรียกใช้ constructor ของ super class
- สัญลักษณ์จุดเด่นของ OOP ในการ reuse code
- เมื่อวิเคราะห์ได้ว่า วัตถุหนึ่งมีลักษณะคล้ายกับวัตถุหนึ่งแต่มีความเฉพาะเจาะจงมากกว่า “จึงใช้เทคนิคการสืบทอด” หรือ ใช้คำว่า “...is a...” ในการดูความสัมพันธ์ระหว่างวัตถุที่คิด จะใช้การสืบทอด
- ไม่ควรใช้การ reuse code กับคลาสที่ไม่มีความสัมพันธ์กันจริง ทำให้เกิดปัญหา คือ
 - พัวพันโดยไร้เหตุ
 - อุบัติเหตุจากการเรียกใช้ความสามารถ

วากยสัมพันธ์ของการสืบทอด

```
<class-modifiers> class SubclassName extends SuperclassName {  
    ...  
}
```

```
public class Counter {  
    public int value = 0;  
  
    public void increase() {  
        value++;  
    }  
}
```

```
public class UpDownCounter extends Counter {  
    public void decrease() {  
        value--;  
    }  
}
```

```
Counter c1 = new Counter();  
UpDownCounter c2 = new UpDownCounter();  
  
c1.increase();  
c2.increase();  
c2.decrease();  
  
System.out.println("c1: " + c1.value);  
System.out.println("c2: " + c2.value);
```

```
c1: 1  
c2: 0
```

การสืบทอด (ต่อ)

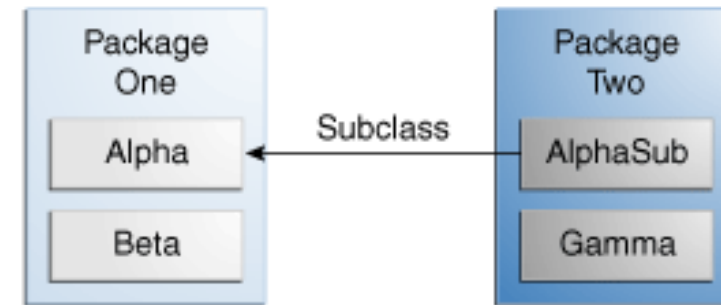
- หากต้องการให้ Subclass สามารถเข้าถึงข้อมูลของคลาสได้ ต้องกำหนดการเข้าถึงแบบ protected

Modifier on members in a class	Accessed from the same class	Accessed from the same package	Accessed from a subclass	Accessed from a different package
public	○	○	○	○
protected	○	○	○	×
(default) *	○	○	×	×
private	○	×	×	×

* **default** access has no modifier associated with it

ตัวกำหนดระดับการเข้าถึง
ให้กับสมาชิกของคลาส (attributes, methods)

Modifier	Alpha	Beta	Alphasub	Gamma
Public	Y	Y	Y	Y
Protected	Y	Y	Y	N
No modifier (package- private)	Y	Y	N	N
private	Y	N	N	N





Package

- Organizing your code
- Avoid class name collision



Packages

- Packages are used to organize classes
- All standard Java packages are inside the **java** and **javax** package hierarchies
- Uses packages to guarantee the uniqueness of class names
- To put a class into a package add
package packageName;
as the first non-comment and non-blank statement
in the program

Using Public Class from other packages

- add the full package name in front of *every class name*

```
java.util.Date aday = new java.util.Date();
```

- use the *import statement*

```
import java.util.*;
```

```
Date aday = new Date();
```

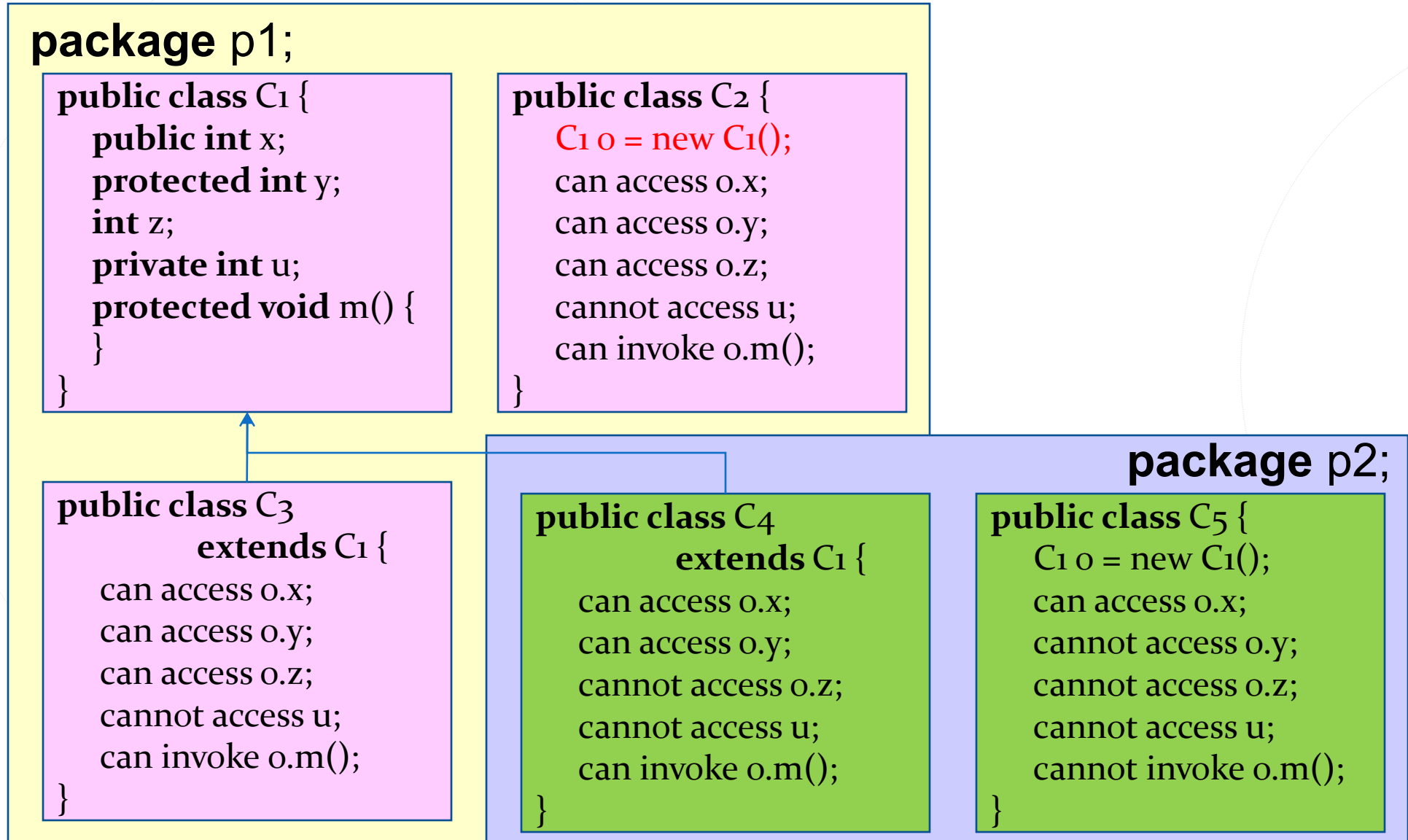
Or

```
import java.util.Date;
```

```
Date aday = new Date();
```



Data and Methods Visibility



Data and Methods Visibility (Code demo)

EXPLORER

- > OPEN EDITORS
 - ✓ FUND2
 - > labtest
 - > midterm
 - > my
 - > mydriver
 - ✓ package_demo
 - bin\package_demo...
 - C1.class
 - C2.class
 - C3.class
 - ✓ p1
 - C1.java
 - C2.java
 - C3.java
 - ✓ p2
 - C4.java
 - C5.java

package_demo > p1 > C1.java > C1

```
1 package package_demo.p1;
2
3 public class C1 {
4     public int x = 1;
5     protected int y = 2;
6     int z = 3;
7     private int u = 4;
8     protected void m() {
9         System.out.println("Method inside C1.");
10    }
11    public int getU(){
12        return u;
13    }
14 }
15
```

package_demo > p1 > C2.java > ...

```
1 package package_demo.p1;
2 public class C2 {
3     Run | Debug
4     public static void main(String[] args) {
5         C1 obj = new C1();
6         System.out.println("Access x from C2 = "+obj.x);
7         System.out.println("Access y from C2 = "+obj.y);
8         System.out.println("Access z from C2 = "+obj.z);
9         //System.out.println("Access u from C2 = "+obj.u);
10        obj.m();
11    }
12 }
```

```
E:\playground\fund2\package_demo\p1>javac -d "../bin" *.java
```

```
E:\playground\fund2\package_demo\p1>java -classpath "../bin" package_demo.p1.C2
```

```
Access x from C2 = 1
```

```
Access y from C2 = 2
```

```
Access z from C2 = 3
```

```
Method inside C1.
```

ตัวอย่างการใช้ระดับการเข้าถึงแบบคุ้มครอง (protected)

```
package my.util;

public class Pair {
    protected int first;
    protected int second;


    public Pair(int first, int second) {
        this.first = first;
        this.second = second;
        System.out.println("Pair constructor called");
    }

    public int getFirst() {
        return first;
    }

    public int getSecond() {
        return second;
    }

    public void setPair(int first, int second) {
        this.first = first;
        this.second = second;
    }

    public void print() {
        System.out.println("(" + first + ", " + second + ")");
    }
}
```




```
import my.util.Pair;

public class SwappablePair extends Pair {
    public SwappablePair(int first, int second) {
        super(first, second);
        System.out.println("SwappablePair constructor called");
    }

    public void swap() {
        int temp = first;
        first = second;
        second = temp;
    }
}
```

ตัวอย่างการใช้ระดับการเข้าถึงแบบคุ้มครอง (ต่อ)

```
import my.util.Pair;   
  
class PairUser {  
    public static void main(String[] args) {  
        Pair a = new Pair(1, 2);  
        SwappablePair b = new SwappablePair(3, 4);  
  
        System.out.println(a.first + a.second); // OK if in the same package, ERROR otherwise  
        System.out.println(b.first + b.second); // OK if in the same package, ERROR otherwise  
  
        b.swap(); // OK  
        b.print(); // OK  
    }  
}
```

ผลลัพธ์ของการรัน PairUser จะเป็นเช่นไร ?

```
E:\playground\fund2>javac PairUser.java
PairUser.java:8: error: first has protected access in Pair
    System.out.println(a.first + a.second); // OK if in the same package, ERROR otherwise
                        ^
PairUser.java:8: error: second has protected access in Pair
    System.out.println(a.first + a.second); // OK if in the same package, ERROR otherwise
                        ^
PairUser.java:9: error: first has protected access in Pair
    System.out.println(b.first + b.second); // OK if in the same package, ERROR otherwise
                        ^
PairUser.java:9: error: second has protected access in Pair
    System.out.println(b.first + b.second); // OK if in the same package, ERROR otherwise
                        ^
4 errors
```

คอนสตรัคเตอร์ในซัพคลาส

- Subclass เรียกใช้ constructor ของ super class ผ่านคำสั่ง super
- กรณี subclass ไม่ได้เรียกใช้ super เลย Java จะทำการเรียก default constructor ของ super class โดยปริยาย (ตัวที่ไม่มีพารามิเตอร์)
 - ถ้า super class ไม่มี constructor , compiler จะแจ้งข้อผิดพลาด

```
// Version 1
public class ResettablePair extends Pair {
    public void reset() {
        first = 0;
        second = 0;
    }
}
```

Compile Error

```
// Version 2
public class ResettablePair extends Pair {
    public ResettablePair() {
        System.out.println("ResettablePair constructor called");
    }

    public void reset() {
        first = 0;
        second = 0;
    }
}
```

Compile Error

คอนสตรัคเตอร์ในชั้นคลาส (ต่อ)

- Solution 1 ให้ default constructor ของ subclass เรียกใช้ constructor ของ super class ที่มีอยู่จริง (ไม่ต้องรอให้ Java เลือก)

```
// Version 3
public class ResettablePair extends Pair {
    public ResettablePair() {
        super(0, 0); // super must be called before other statements
        System.out.println("ResettablePair constructor called");
    }

    public void reset() {
        first = 0;
        second = 0;
    }
}
```

- Solution 2 สร้าง default constructor ให้ super class

```
public class Pair {

    // ... code omitted

    public Pair() {
        this(0, 0);
    }

    public Pair(int first, int second) {
        this.first = first;
        this.second = second;
        System.out.println("Pair constructor called");
    }

    // ... code omitted

}
```



Overriding

Rewrite super class's method



Overriding Method (ต่อ)

- Overriding เป็นการเปลี่ยนแปลงการทำงานของ Method ใน Subclass ที่สืบทอดมาจาก Superclass
- Method ที่ override ใน Subclass จะมีชื่อ, ชนิดข้อมูลที่คืนค่า, จำนวนและชนิดข้อมูลของ Argument ที่เหมือนกับ Superclass
- สามารถพัฒนา Method ให้มีการทำงานในเรื่องเดียวกัน แต่แตกต่างกันในรายละเอียดของการทำงาน
- มีการใช้ annotation เพื่อเป็นการย้ำเตือนว่า method ที่เขียนคือการ override method
 - จะระบุหรือไม่ระบุก็ได้ หากระบุจะช่วยลดความผิดพลาดโดยไม่ตั้งใจ

`@Override`

Ex. Overriding Method

```
public class Employee {
    private String name;
    private double salary;
    private final int id;
    private static int lastId = 1000;

    public static final double SALARY_STEP_SIZE = 10.0;

    public Employee(String name, double salary) {
        this.name = name;
        this.salary = computeNextSalaryStep(salary);
        id = ++lastId;
    }

    public String getName() {
        return name;
    }

    public int getId() {
        return id;
    }
}
```

```
private static double computeNextSalaryStep(double salary) {
    // Round to the next salary step
    double steps = Math.ceil(salary / SALARY_STEP_SIZE);
    return steps * SALARY_STEP_SIZE;
}

public void raiseSalary(double percent) {
    double raise = salary * percent / 100.0;
    salary = computeNextSalaryStep(salary + raise);
}

public double getEarnings() {
    return salary;
}

public void printProfile() {
    System.out.printf("Name: %s\n", getName());
    System.out.printf("ID: %d\n", getId());
    System.out.printf("Salary: %, .2f\n", getEarnings());
}
```

Ex. Overriding Method (๓๑)

```
public class SalesEmployee extends Employee {  
    private double grossSales;  
    private double commissionRate;  
  
    public SalesEmployee(String name, double salary, double grossSales, double commissionRate) {  
        super(name, salary);  
        this.grossSales = grossSales;  
        this.commissionRate = commissionRate;  
    }  
  
    public void setGrossSales(double grossSales) {  
        this.grossSales = grossSales;  
    }  
  
    public double getCommission() {  
        return grossSales * commissionRate;  
    }  
  
    @Override  
    public double getEarnings() {  
        return super.getEarnings() + getCommission();  
    }  
  
    @Override  
    public void printProfile() {  
        System.out.printf("Name: %s\n", getName());  
        System.out.printf("ID: %d\n", getId());  
        System.out.printf("Current Earnings: %, .2f\n", getEarnings());  
    }  
}
```

Ex. Overriding Method (๓๑)

```
Employee george = new Employee("George", 15_000.00);  
SalesEmployee sarah = new SalesEmployee("Sarah", 12_000.00, 120_000.00, 2.5);  
george.printProfile();  
sarah.printProfile();
```

Name: George

ID: 1001

Salary: 15,000.00

Name: Sarah

ID: 1002

Current Earnings: 312,000.00



คลาส(ที่ชื่อ) Object

ทำให้เกิดการสืบทอดปริยาย



Class Object

- คลาสทุกคลาสมีคลาส Object เป็นซูเปอร์คลาสไม่ว่าจะโดยตรงหรือโดยอ้อม
- คลาส Object อยู่ใน package java.lang
- คลาสที่อยู่บนสุดของลำดับชั้นของคลาส
- ในคลาส Object จะมีเมทอดอยู่จำนวนหนึ่งซึ่งจะสืบทอดไปยังทุก ๆ คลาส
 - clone
 - hashCode
 - Equals
 - notify, notifyAll, wait
 - Finalize
 - toString
 - getClass

Class Object's method

- clone สร้างอ็อบเจกต์ใหม่ที่มีลักษณะเหมือนกับอ็อบเจกต์ปัจจุบัน
- equals เปรียบเทียบว่า 2 อ็อบเจกต์มีค่าเหมือนกัน โดยใช้เครื่องหมาย ==
- finalize เมทอดนี้จะถูกเรียกโดยตัวเก็บขยะ (garbage collector) เมื่อกำลังจะคืนพื้นที่ในหน่วยความจำของอ็อบเจกต์ที่ไม่ได้ใช้งานแล้ว
- getClass คืนค่าเป็นข้อมูลของคลาสของอ็อบเจกต์นั้น
- hashCode คืนค่าเป็นจำนวนเต็มที่ใช้สำหรับการอ้างอิงตำแหน่งในโครงสร้างข้อมูล ที่เรียกว่าตารางแฮช (hash table) โดยหลักการแล้ว อ็อบเจกต์ที่ต่างกันแต่ละตัว ควรจะมีค่ารหัสแฮชที่ไม่ซ้ำกัน

Class Object's method (ต่อ)

- notify, notifyAll และ wait ใช้ในงานด้านการโปรแกรมแบบหลายเธรด (multithreading)
- toString คื้้นค่าเป็นสตริงที่แทนตัวอ็อบเจกต์นั้น โดยคื้้นค่าเป็นชื่อแพกเกจ ตามด้วยชื่อคลาส และค่ารหัสแฮชที่ได้จากเมทอด hashCode การโอเวอร์ไรด์เมทอดนี้จะทำให้เราสามารถกำหนดได้ว่าจะให้แสดงค่าอ็อบเจกต์ในรูปแบบใดเมื่อมีการสั่งพิมพ์ค่าของอ็อบเจกต์บนจอภาพ เช่น ผ่านคำสั่ง `System.out.println`

ตัวอย่างการ overriding method toString

```
public class Employee {  
  
    // ... code omitted  
  
    public void printProfile() {  
        System.out.printf("Name: %s\n", getName());  
        System.out.printf("ID: %d\n", getId());  
        System.out.printf("Salary: %, .2f\n", getEarnings());  
    }  
  
    @Override  
    public String toString() {  
        return String.format("Name: %s, ID: %d, Salary: %, .2f",  
                               getName(), getId(), getEarnings());  
    }  
}
```

```
Employee george = new Employee("George", 15_000.00);  
System.out.println(george);  
george.printProfile();
```

Name: George, ID: 1001, Salary: 15,000.00

Name: George

ID: 1001

Salary: 15,000.00

ข้อดีของการสืบทอด

- การนำกลับมาใช้ใหม่

ถ้าต้องการสร้าง Class ที่มีความสัมพันธ์แบบ “...is a...” กับ Class ที่มีอยู่แล้ว สามารถใช้การสืบทอดแทนที่จะเขียนขึ้นมาใหม่หมด

- ความเป็นมาตรฐานเดียวกัน

Class พื้นฐานเป็นการกำหนดโครงสร้างแบบในการระบุความสามารถของ Object ใน Subclass ไม่เกิดความซ้ำซ้อน (ที่แอบมีข้อแตกต่างโดยไม่ตั้งใจ)

- ปรับปรุงและทดสอบโปรแกรมง่ายขึ้น

ข้อเสียของการสืบทอด

- โปรแกรมทำงานช้าลง
มี (overhead) ในการค้นหาและเรียกใช้ Method ของ Subclass แต่ overhead ดังกล่าวถือว่าน้อยมากเมื่อเทียบกับประโยชน์ที่ได้จากการสืบทอด
- ความซับซ้อนเพิ่มขึ้น
ผู้ใช้ต้องดูการ implement ใน Class ต่าง ๆ ที่อยู่ในผังการสืบทอด จนกว่าจะพบ Class ที่implement Method นั้น ปัญหาที่กล่าวถึงนี้มีชื่อว่า ปัญหาลูกดึง (Yo-yo problem)

ข้อควรระวังในการสืบทอดกรรมนำมาใช้เพราะ reuse code โดยไว้ ความสัมพันธ์ที่แท้จริง

```
public class Point2D {  
    protected double x;  
    protected double y;  
  
    public Point2D(double x, double y) {  
        this.x = x;  
        this.y = y;  
    }  
  
    public double getX() {  
        return x;  
    }  
  
    public double getY() {  
        return y;  
    }  
  
    public double distance(Point2D p) {  
        System.out.println("distance() called on Point2D.");  
        double dx = p.x - x;  
        double dy = p.y - y;  
        return Math.sqrt(dx*dx + dy*dy);  
    }  
}
```

```
public class Point3D extends Point2D {  
    protected double z;  
  
    public Point3D(double x, double y, double z) {  
        super(x, y);  
        this.z = z;  
    }  
  
    public double getZ() {  
        return z;  
    }  
  
    public double distance(Point3D p) {  
        System.out.println("distance() called on Point3D.");  
        double dx = p.x - x;  
        double dy = p.y - y;  
        double dz = p.z - z;  
        return Math.sqrt(dx*dx + dy*dy + dz*dz);  
    }  
}
```

ข้อควรระวังในการสืบทอดฯ (ต่อ)

```
Point3D p1 = new Point3D(1.0, 2.5, 4.5);  
Point3D p2 = new Point3D(8.0, 2.0, 0.0);  
Point2D p3 = new Point2D(8.0, 2.0);  
System.out.printf("p1->p2: %.2f%n", p1.distance(p2));  
System.out.printf("p1->p3: %.2f%n", p1.distance(p3));
```

distance() called on Point3D.

p1->p2: 8.34

distance() called on Point2D.

p1->p3: 7.02

ผลการทำงานผิดพลาด

P1 ได้รับการสืบทอดมาจาก Point2D ทำให้สามารถเรียกใช้

method distance ที่รับ parameter เป็น object Point2D ได้

ซึ่งในความเป็นจริงเราไม่สามารถหารระยะห่างระหว่าง จุดในปริภูมิสองมิติ กับ จุดที่อยู่ในปริภูมิสามมิติได้

ข้อควรระวังในการสืบทอดกรณีทำให้คลาสขึ้นต่อกันโดยไม่จำเป็น

```
public class Point2D {  
    protected double position[] = new double[2];  
  
    public Point2D(double x, double y) {  
        position[0] = x;  
        position[1] = y;  
    }  
  
    public double getX() {  
        return position[0];  
    }  
  
    public double getY() {  
        return position[1];  
    }  
  
    public double distance(Point2D p) {  
        System.out.println("distance() called on Point2D.");  
        double dx = p.position[0] - position[0];  
        double dy = p.position[1] - position[1];  
        return Math.sqrt(dx*dx + dy*dy);  
    }  
}
```

- การเปลี่ยนแปลง Point2D ทำให้ Point3D compile ไม่ผ่าน เรียกปัญหาลักษณะนี้ว่า ซูเปอร์คลาสเปราะบาง (fragile superclass หรือ brittle superclass)

แก้ไขด้วยการเปลี่ยน protected เป็น private และนำ getter/setter มาใช้งานแทน

อ้างอิง

- <https://nbviewer.jupyter.org/github/Poonna/java-book/>