




繼承與多型與抽象類別

學習目標

- 瞭解繼承目的
 - 瞭解繼承與多型的關係
 - 瞭解抽象類別
- 

繼承 (Inheritance)

- 繼承(**inheritance**)的觀念指的是物件可自其他物件延續使用其成員變數及成員方法，物件導向程式設計的繼承觀念可方便資料及方法的重覆使用。
- 繼承是一種由已有的類別創建新類別的機制。利用繼承，我們可以先創建一個共有屬性的一般類別，根據該一般類別再創建具有特殊屬性的新類別，新類別繼承一般類的狀態和行為，並根據需要增加它自己的新的狀態和行為。由繼承而得到的類別稱為子類別，被繼承的類別稱為父類別。

子類別與父類別

- 父類別可以是自己編寫的類別也可以是JAVA類庫中的類別。
- 利用繼承有利於實現代碼的重複使用，子類別只需要添加新的功能代碼即可。JAVA不支援多重繼承，即子類別只能有一個父類別。

```
class 子類別名 extends 父類別名別{  
    ...  
}
```

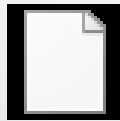
類別的宣告中沒有使用關鍵字extends時，被系統預設為是Object的子類別，Object是包java.lang中的類別。

class A{		class A extends Object{
...
}	=	}



CMath.java

```
package ch6_1;
public class CMath {
    public void getMax(int a, int b) {
        int bigNum;
        if (a > b)
            bigNum = a;
        else
            bigNum = b;
        System.out.println(a + " 與 " + b + " 的最大數為 " + bigNum);
    }
}
```



SonCMath.java

```
package ch6_1;
public class SonCMath extends CMath {
    public void getFactorial(int a) {
        int ans = 1, i;
        System.out.print(a + "! = ");
        for (i = 1; i < a; i++) {
            System.out.print(i + " * ");
            ans *= i;
        }
        ans *= a;
        System.out.println(a + " = " + ans);
    }
}
```

```
ch6_1
├── JRE 系統程式庫 [jdk-17.0.4.1]
└── src
    └── ch6_1
        ├── CMath.java
        ├── ExtendDemo.java
        └── SonCMath.java
```



ExtendDemo.java

```
package ch6_1;

public class ExtendDemo {
    public static void main(String[] args) {
        SonCMath math1 = new SonCMath();
        math1.getMax(15, 43);
        // 呼叫子類別繼承父類別的方法
        math1.getFactorial(6);
        // 呼叫子類別自己的方法
    }
}
```

結果:

15 與 43 的最大數為 43

$6! = 1 * 2 * 3 * 4 * 5 * 6 = 720$

子類別的繼承性

繼承的定義

- 子類別的成員中有一部分是子類別自己宣告的定義，另一部分是從它的父類別繼承的。
- 子類別繼承父類別的成員變數作為自己的一個成員變數，就好像它們是在子類別中直接宣告一樣，可以被子類別中自己宣告的任何實例方法操作。
- 子類別繼承父類別的方法作為子類別中的一個方法，就像它們是在子類別中直接宣告一樣，可以被子類別中自己宣告的任何實例方法呼叫。

存取限制相同類別庫

子類別和父類別在相同類別庫中

- 子類別繼承了其父類別`private`以外的成員變數作為自己的成員變數。
- 繼承了父類中`private`以外的方法作為自己的方法。
- 繼承的成員變數以及方法的存取權限保持不變。

存取限制相同類別庫

	修飾字	可否繼承	可否存取
在相同類別庫中	public	可	可
	private	否	否
	protected	可	可
	無修飾字	可	可

存取限制相同類別庫

```
package ch6_2;
public class Father {
    private int moneyDollar=300;
    int moneyNTD=200;
    int add(int x,int y){
        return x+y;
    }
}
```

```
public class Son extends Father{
    int moneyUSD=800;
    public void changMoneyNTD(int x){
        moneyNTD=x;
    }
    public void changMoneyUSD(int x){
        moneyUSD=x;
    }
    int subs(int x,int y){
        return x-y;
    }
}
```

```
package ch6_2;
public class GrandSon extends Son{
    int multi(int x,int y){
        return x*y;
    }
}
```

存取限制相同類別庫

```
package ch6_2;
public class ch6_2 {
    public static void main(String[] args) {
        int a=5,b=3;
        Son s=new Son();
        GrandSon sund=new GrandSon();
        s.changMoneyNTD(666);
        s.changMoneyUSD(5000);
        System.out.println("兒子的台幣是繼承的屬性, 當前的值是:"+s.moneyNTD);
        System.out.println("兒子的美金是新增的屬性, 當前的值是:"+s.moneyUSD);
        System.out.printf("減法是兒子新增的功能,%d-%d等於%d\n",a,b,s.subs(a,b));
        System.out.printf("加法是兒子繼承的功能,%d+%d等於%d\n",a,b,s.add(a,b));
        System.out.println("孫子的台幣和美金都是繼承的屬性, 當前的值是:");
        System.out.println("台幣:"+sund.moneyNTD+" 美金:"+sund.moneyUSD);
        System.out.printf("乘法是孫子新增的功能,%d*%d等於%d\n",a,b,sund.multi(a,b));
        System.out.printf("加法是孫子繼承的功能,%d+%d等於%d\n",a,b,sund.add(a,b));
        System.out.printf("減法是孫子繼承的功能,%d-%d等於%d\n",a,b,sund.subs(a,b));
    }
}
```

存取限制相同類別庫

結果:

兒子的台幣是繼承的屬性,當前的值是:666

兒子的美金是新增的屬性,當前的值是:5000

減法是兒子新增的功能, $5-3$ 等於2

加法是兒子繼承的功能, $5+3$ 等於8

孫子的台幣和美金都是繼承的屬性,當前的值是:

台幣:200 美金:800

乘法是孫子新增的功能, $5*3$ 等於15

加法是孫子繼承的功能, $5+3$ 等於8

減法是孫子繼承的功能, $5-3$ 等於2

存取限制在不同類別庫

子類別和父類別在不同類別庫中

- 子類別只能繼承父類別的protected、public成員變數和方法
- 繼承的成員或方法的存取權限不變。
- 子類別不能繼承父類別的友好變數和友好方法。

存取限制在不同類別庫

	修飾字	可否繼承	可否存取
在不同 類別庫中	public	可	可
	private	否	否
	protected	可	否
	無修飾字	否	否

繼承 (Inheritance)

- 語法： `class ClassName extends BaseClass`
 - 例如： `class Line extends GraphicsObject`
 - Base Class (`SuperClass`)：基底類別、父類別
 - Derived Class (`Subclass`)：衍生類別、子類別
 - Java 理論上不支援多重繼承，也就是說，一個子類別只能有一個父類別。
 - 子類別將會繼承到父類別中所有可以存取的成員，包括：變數以及方法
 - 注意： `constructor`:建構式;建構元;建構子，無法被繼承
 - Java 中每個物件的總祖先： `Object` 類別 (`java.lang.Object`)

繼承 (Inheritance)

- 可繼承成員
 - Superclass 中宣告為 **public** 或 **protected** 的成員。
 - 如果 Subclass 與 Superclass 在同一個 package 中，會繼承未做任何存取控制宣告的成員。
- 不可繼承成員
 - 如果 Subclass 與 Superclass 在不同 package，所有未宣告有效範圍的成員全部不繼承。（因為預設式 package access）
 - Superclass 中宣告成 **private** 的成員

繼承 (Inheritance)

- `super()`可以呼叫父類別中沒有參數的建構式方法。
- `this ();`敘述可以呼叫類別中沒有參數的建構式方法。
- `super()`及`this ();`敘述也可以加入參數以呼叫需要傳入參數的建構方法。

Object類別

- 是所有類別的父類別，而Object類別中沒有參數的建構方法中不包含任何敘述。

繼承 (Inheritance)

```
package ch6_3;
public class Account {
    public Account(String st) {
        System.out.println(st);
    }
}

package ch6_3;
public class SavingsAccount extends Account {
    String name;
    public SavingsAccount(String name) {
        super("SavingsAccount");
        this.name = name;
    }
}

package ch6_3;
public class ch6_3 {
    public static void main(String[] args) {
        SavingsAccount sav = new SavingsAccount("Hi");
        System.out.println(sav.name);
    }
}
```

結果:

SavingsAccount
Hi

繼承 (Inheritance)

- 繼承可以使用 **extends** 和 **implements** 這兩個關鍵字來實現繼承，而且所有的類別都是繼承於 **java.lang.Object**，當一個類別沒有繼承的兩個關鍵字，則默認繼承 **Object**（這個類在 **java.lang** 包中，所以不需要 **import**）祖先類。
- **extends** 關鍵字在 **Java** 中，類的繼承是單一繼承，也就是說，一個子類只能擁有一個父類別，所以 **extends** 只能繼承一個類別。
- **implements** 關鍵字可以具有多繼承的特性，使用範圍為類繼承介面的情況，可以同時繼承多個介面（介面跟介面之間採用逗號分隔）。

多型 (Polymorphism)

- 多型表示一個方法可以有許多型式，以呼叫方法時所傳入的參數的個數與型別來決定。
- 在子類別中重新定義繼承自父類別方法的動作稱為重新定義(**overriding**)。
- 擁有不同的型式的方法我們稱為重載 (**overloaded**)

多型 (Polymorphism)

- 重載(Overloading) 是一個類別中定義了多個方法名相同,而他們的參數的數量不同或數量相同而類型和次序不同。
- 重新定義(Overriding)是在子類別存在方法與父類別的方法的名字相同,而且參數的個數與類型一樣,返回值也一樣的方法。
- 方法重載是一個類的多態性表現,而方法重新定義是子類與父類的一種多態性表現。

多型 (Polymorphism)

- 重新定義 **Overriding**
 - 若繼承下來後，不滿意祖先定義的方法，子孫可以在繼承以後重新改寫，稱為 **Overriding**。
 - 可重新定義成員
 - 任何與 **Superclass** 同名的成員
 - 必重新定義成員
 - **Subclass** 一定要覆蓋 **superclass** 中宣告為 **abstract** 的 **methods**，除非 **subclass** 本身也是 **abstract** 類別
 - 不可重新定義成員
 - **Subclass** 不可覆蓋 **superclass** 的 **final methods**

重新定義Overriding

```
package ch6_4;
public class Animal {
    public void sound() {
        System.out.println("動物叫");
    }
}
```

```
package ch6_4;
public class Dog extends Animal {
    public void sound() {
        System.out.println("狗叫");
    }
}
```

```
package ch6_4;
public class ch6_4 {
    public static void main(String[] args) {
        Animal a = new Animal(); // Animal 物件
        Dog b = new Dog(); // Dog 物件
        a.sound(); // 執行 Animal 類別的方法
        b.sound(); // 執行 Dog 類別的方法
    }
}
```

結果:

動物叫
狗叫

重載 Overloading

- 同一份函式，準備多種定義，以供各種場合呼叫，稱為**Overloading**。
- 建構元也可以利用**參數的不同**，來達成 **overloading**。

重載 Overloading

```
package ch6_5;
public class Animal {
    public void sound() {
        System.out.println("動物叫");
    }
}
```

```
package ch6_5;
public class bird extends Animal {
    public void sound(int a) {
        System.out.println("鳥叫");
    }
}
```

```
package ch6_5;
public class dog extends Animal {
    //以下參數類型順序不同
    public String sound(int a,String s){
        System.out.println(s);
        return "狗叫";
    }
}
```

```
package ch6_5;
public class monkey extends Animal {
    public String sound(String s,int a){
        System.out.println(s);
        return "猴叫";
    }
}
```

```
package ch6_5;
public class ch6_5 {
    public static void main(String[] args) {
        Animal a = new Animal(); // Animal 物件
        bird b = new bird(); // Animal 物件
        dog c = new dog(); // Animal 物件
        monkey d = new monkey(); // Animal 物件
        a.sound(); // 執行 Animal 類別的方法
        b.sound(1); // 執行 Animal 類別的方法
        c.sound(1,"狗叫"); // 執行 Animal 類別的方法
        d.sound("猴叫",1); // 執行 Animal 類別的方法
    }
}
```

結果:

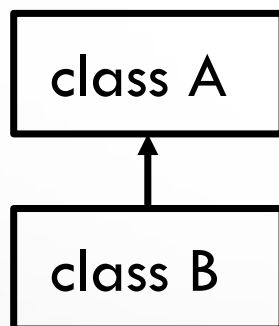
- 動物叫
- 鳥叫
- 狗叫
- 猴叫

重新定義寫與重載的比較

區別點	重載方法	重新定義方法
參數列表	必須修改	一定不能修改
返回類型	可以修改	一定不能修改
異常	可以修改	可以減少或刪除，一定不能拋出新的或者更廣的異常
訪問	可以修改	一定不能做更嚴格的限制（可以降低限制）

繼承類型

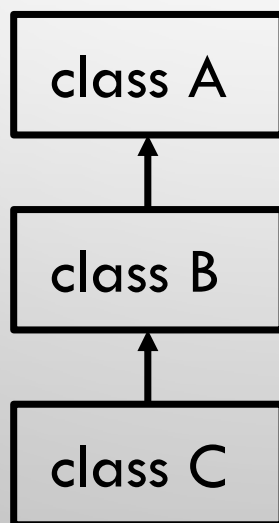
單繼承



```
public class A {  
    ...  
}
```

```
public class B extend A {  
    ...  
}
```

多重繼承



```
public class A {  
    ...  
}
```

```
public class B extend A {  
    ...  
}
```

```
public class C extend B {  
    ...  
}
```

多重繼承

```
package ch6_6;
public class Animal {
    void color(){}
}

package ch6_6;
public class Dog extends Animal{
    void color(){
        System.out.println("黃色");
    }
}

package ch6_6;
public class Hair extends Dog{
    void color(){
        System.out.println("灰色");
    }
}
```

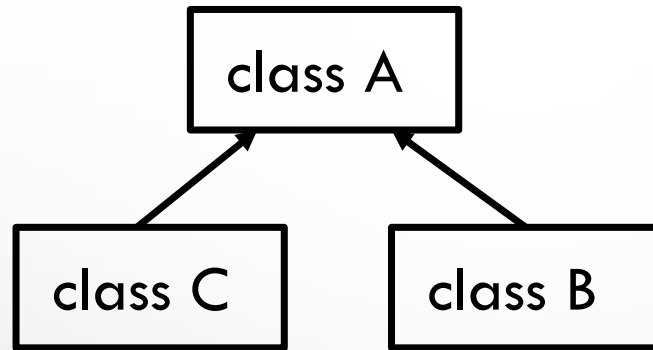
```
package ch6_6;
public class ch6_6 {
    public static void main(String[] args) {
        Dog d=new Dog();
        d.color();
        Hair h=new Hair();
        h.color();
    }
}
```

結果:

黃色
灰色

繼承類型

不同類別繼承



```
public class A {  
    ...  
}  
  
public class B extend A {  
    ...  
}  
  
public class C extend A {  
    ...  
}
```

不同類別繼承

```
package ch6_7;
public class Animal {
    void cry(){}
}

package ch6_7;
public class Dog extends Animal{
    void cry(){
        System.out.println("汪汪");
    }
}
```

```
package ch6_7;
public class Cat extends Animal{
    void cry(){
        System.out.println("喵喵");
    }
}
```

```
package ch6_7;
public class ch6_7 {
    public static void main(String[] args) {
        Dog d=new Dog();
        d.cry();
        Cat c=new Cat();
        c.cry();
    }
}
```

結果:

汪汪
喵喵

Super 關鍵字

Super關鍵字有兩種用法：一種用法是子類使用super調用父類的建構式方法，另一種用法是子類使用super呼叫被子類別隱藏的成員變數和方法。

使用super呼叫父類的建構式方法

- 通過**super**關鍵字來實現對父類別成員的訪問，用來引用當前物件的父類別。
- **this**關鍵字：指向自己的呼叫

Super 關鍵字

```
package ch6_8;
public class Animal {
void eat() {
    System.out.println("animal 父類別");
    System.out.println("-----");
}
}

package ch6_8;
public class Dog extends Animal {
    void eat() {
        System.out.println("dog 類別 ");
        System.out.println("-----");
    }

    void eatTest1() {
        super.eat(); // super呼叫父類別方法
    }

    void eatTest2() {
        this.eat(); // this 呼叫自己的方法
    }
}
```

```
package ch6_8;
public class ch6_8 {
    public static void main(String[] args) {
        Animal a = new Animal();
        a.eat();
        Dog d = new Dog();
        d.eat(); // dog子類別
        System.out.println("super呼叫父類別方法 ");
        d.eatTest1(); // animal 父類別
        System.out.println("this 呼叫自己的方法 ");
        d.eatTest2(); // dog 子類別
    }
}
```

結果:

```
animal 父類別
-----
dog 類別
-----
super呼叫父類別方法
animal 父類別
-----
this 呼叫自己的方法
dog 類別
-----
```

Super 關鍵字

使用super操作被隱藏的成員變數和方法

- 當子類中定義了一個方法，並且這個方法的名字、返回類型、參數個數和類型和父類別的某個方法完全相同時，子類別從父類別繼承的這個方法將被隱藏。如果我們在子類別中想使用被子類別隱藏的成員變數或方法就可以使用關鍵字super。
- 子類別使用super呼叫被隱藏的成員變數和方法。

Super 關鍵字

```
package ch6_9;
public class A {
    int x,y;
    A(){
        x=100;
        y=200;
    }
    A(int x,int y){
        this.x=x;
        this.y=y;
    }
}
```

```
package ch6_9;
public class B extends A{
    int z;
    B(int x,int y){
        super(x,y);
        z=300;
    }
    B(){
        super();
        z=800;
    }
    public void f(){
        System.out.printf("x=%d,y=%d,z=%d\n",x,y,z);
    }
}
```

Super 關鍵字

```
package ch6_9;  
public class ch6_8 {  
    public static void main(String[] args) {  
        B b1=new B(10,20);  
        b1.f();  
        B b2=new B();  
        b2.f();  
    }  
}
```

結果:

x=10,y=20,z=300
x=100,y=200,z=800

final 類別和final方法

final類別不能被繼承，即不能有子類別，如

```
final class A{  
    ... ..  
}
```

- final類別對於編譯器和解譯器的正常運行有很重要的作用，對它不能輕易改變。
- final方法，不能被重寫，即不允許子類重寫隱藏繼承的final方法，final方法的行為不允許子類別修改。

```
package ch6_10;
public class Ccar {
    private final int SPEED = 120 ;
    public final void showBigSpeed(String s) {
        System.out.println(s + " 最大速度是 " + SPEED + " 公里!");
    }
}
```

```
package ch6_10;
class PiliCcar extends Ccar {
    public void showBigSpeed(String s, int n) {
        System.out.println(s + " 加強後最大速度是 " + n + " 公里!");
    }
}
```

```
package ch6_10;
public class ch6_10 {
    public static void main(String[] args) {
        Ccar car1 = new Ccar();
        car1.showBigSpeed("car1");           //呼叫Ccar父類別的showBigShow()
        PiliCcar car2 = new PiliCcar();
        car2.showBigSpeed("car2");           //呼叫Ccar父類別的showBigShow()
        car2.showBigSpeed("car2", 180);      //呼叫PiliCcar子類別的showBigShow()
    }
}
```

結果:

car1 最大速度是 120 公里！

car2 最大速度是 120 公里！

car2 加強後最大速度是 180 公里！

抽象類別abstract class

```
abstract class A{  
    ...  
}
```

abstract類別有如下特點：

1. abstract類別中可以有abstract方法，也可以沒有abstract方法
abstract方法，只允許宣告，不允許實現，而且不允許使用final和abstract同時修飾一個方法。

2. abstract類不能用new運算創建物件

對於abstract類別，我們不能使用new運算元創建該類別的物件。

3. 做上轉型物件

儘管abstract類別不能創建物件，但它的非abstract子類別必須要重寫abstract方法，這樣一來，就可以讓abstract類別宣告的物件成為其子類別物件的上轉型物件，並呼叫子類別重新定義的方法。

注：如果一個abstract類別是abstract類的子類別，它可以重寫父類別的abstract方法，也可以繼承這個abstract方法。

抽象方法

- **abstract**修飾字的方法稱為抽象方法，抽象方法是僅具有宣告部分而沒有定義區塊的方法
- 例如：

```
public abstract int 抽象方法(char 字元, int 整數);
```
- 抽象方法可以被繼承，一個抽象方法的定義區塊是在被繼承至子類別之後才加上的。

抽象類別和抽象方法

- 抽象類別不能被產生物件。
- 抽象類別不一定要包含抽象方法。
- 若類別中包含了抽象方法，則該類別必須被定義為抽象類別。
- 抽象方法只需宣告，無需實現。
- 抽象類別需要被繼承，抽象方法需要被重寫。


```
package ch6_11;
public abstract class Shape {
    // 屬性開始
    public double x;
    public double y;
    // 屬性結尾
    // 方法開始
    public abstract void area();
    // 方法結尾
}
```

```
package ch6_11;
public class Circle extends Shape {
    // 屬性開始
    public double r;
    // 屬性結尾
    public Circle(double x, double y, double r) {
        this.x = x;
        this.y = y;
        this.r = r;
    }
    // 方法開始
    public void area() {
        System.out.println("圓面積: " + 3.1416*r*r);
    }
    // 方法結尾
}
```

```
package ch6_11;
public class Rectangle extends Shape {
    // 屬性開始
    private double x1;
    private double y1;
    // 屬性結尾
    public Rectangle(double x, double y, double x1, double y1) {
        this.x = x;
        this.y = y;
        this.x1 = x1;
        this.y1 = y1;
    }
    // 方法開始
    public void area() {
        System.out.println("長方形面積: "+(y1-y)*(x1-x));
    }
    // 方法結尾
}
```

```
package ch6_11;
public class ch6_11 {
public static void main(String[] args) {
    // 抽象類別的物件變數
    Shape s;
    // 宣告類別型態的變數，並且建立物件
    Circle c=new Circle(5.0,10.0,4.0);
    Rectangle r=new Rectangle(10.0,10.0,20.0,20.0);
    // 呼叫抽象類型物件的抽象方法area()
    for ( int i = 1; i <= 2; i++ ) {
        if ( i == 1 )      s = c;    // 圓形
        else                s = r;    // 長方形
        s.area();
    }
}
```

結果:

圓面積： 50.2656

長方形面積： 100.0

protected成員

關鍵字	類別內部	相同套件類別	不同套件類別
public	可存取	可存取	可存取
protected	可存取	可存取	子類別可存取
無	可存取	可存取	不可存取
private	可存取	不可存取	不可存取