# [從抽象類別(Abstract Class)到介面(Interface)](http://tianchyi1955.pixnet.net/blog/post/186152356)

## 抽象類別(Abstract Class)

在前面章節已說明過，不描述處理內容，只定義呼叫方式的方法，稱為抽象方法(abstract method)。而擁有抽象方法的類別就是抽象類別(abstract class)。抽象類別不能建立物件，需經繼承並覆寫抽象方法，才可建立物件。類別如包含多個方法，只要其中一個為抽象方法，此類別就成為抽象類別。類別被定義為抽象類別，但其方法可全部不為抽象方法。

抽象類別可宣告欄位。

### 只含抽象方法的抽象類別

<例>含抽象方法卻不定義抽象類別-錯誤

含抽象方法之類別必須為抽象類別。

class Abstract01{//class無abstract修飾子

int a = 0, b = 10;

abstract void absMethod01(int x);//抽象方法

}

編譯錯誤：C:\js>javac Abstract01.java

Abstract01.java:1: error: Abstract01 is not abstract and does not override abstr

act method absMethod01(int)in Abstract01

class Abstract01{//class無abstract修飾子

^

1 error

<例>抽象方法內有敘述句(處理內容)-錯誤

抽象方法內不可含處理內容。只有{}無內容亦不可。

abstract class Abstract02{

int a = 0, b = 10;

abstract void absMethod02(int x){//抽象方法含處理內容

a = x;

if (a == b) System.out.println("absMethod02：a等於b");

else System.out.println("absMethod02：a不等於b");

}

}

編譯錯誤：C:\js>javac Abstract02.java

Abstract02.java:3: error: abstract methods cannot have a body

abstract void absMethod02(int x){//抽象方法含處理內容

^

1 error

<例>一個抽象方法

abstract class Abstract03{

int a = 0, b = 10;

abstract void absMethod03(int x);

}

編譯正常。

<例>多個抽象方法

abstract class Abstract04{

int a = 0, b = 10;

abstract void absMethod04a(int x);

abstract void absMethod04b();

}

編譯正常。

### 含抽象方法及非抽象方法的抽象類別

<例>含抽象方法及非抽象方法

abstract class Abstract05{

int a = 0, b = 10;

abstract void absMethod05a(int x);

void absMethod05b(){

System.out.println("absMethod05b：非abstract方法");

}

}

編譯正常。

### 不含抽象方法的抽象類別

<例>抽象類別可不含抽象方法，但和一般抽象類別一樣，都不可建立物件。

abstract class Abstract06{

int a = 0, b = 10;

void absMethod06(){

System.out.println("absMethod06：非abstract方法");

}

}

編譯正常。

### 抽象類別的繼承及建立物件

抽象類別不可建立物件，須經繼承並覆寫抽象方法後才可建立物件。

<例>抽象類別直接建立物件-錯誤

abstract class Abstract07{

int a = 0, b = 10;

abstract void absMethod07(int x);

}

class AbsMain07{

public static void main(String[] args){

Abstract07 A07 = new Abstract07();

}

}

編譯錯誤：C:\js>javac AbsMain07.java

AbsMain07.java:7: error: Abstract07 is abstract; cannot be instantiated

Abstract07 A07 = new Abstract07();

^

1 error

<例>繼承抽象類別但未覆寫抽象方法，由繼承之子類別建立物件-錯誤

abstract class Abstract08{

int a = 0, b = 10;

abstract void absMethod08(int x);

}

class AbsSon08 extends Abstract08{};

class AbsMain08{

public static void main(String[] args){

AbsSon08 A08 = new AbsSon08();

}

}

編譯錯誤：C:\js>javac AbsMain08.java

AbsMain08.java:8: error: AbsSon08 is abstract; cannot be instantiated

AbsSon08 A08 = new AbsSon08();

^

1 error

<例>繼承抽象類別並覆寫抽象方法，由繼承之子類別建立物件

abstract class Abstract09{

int a = 0, b = 10;

abstract void absMethod09(int x);

}

class AbsSon09 extends Abstract09{

void absMethod09(int x){

a = x;

if (a == b) System.out.println("absMethod09：a等於b");

else System.out.println("absMethod09：a不等於b");

}

}

class AbsMain09{

public static void main(String[] args){

AbsSon09 A09 = new AbsSon09();

A09. absMethod09(10);

A09. absMethod09(9);

}

}

執行結果：C:\js>java AbsMain09

absMethod09：a等於b

absMethod09：a不等於b

<例>繼承沒有抽象方法的抽象類別，由繼承之子類別建立物件

abstract class Abstract10{

int a = 0, b = 10;

void absMethod10(){

System.out.println("absMethod10：非abstract方法");

}

}

class AbsSon10 extends Abstract10{ }

class AbsMain10{

public static void main(String[] args){

AbsSon10 A10 = new AbsSon10();

A10.absMethod10();

 }

}

執行結果：C:\js>java AbsMain10

absMethod10：非abstract方法

### 抽象類別的功用

抽象類別和抽象方法界定了類別的運用方式，或說是運用架構，也就是說其子類別都只能用和其抽象方法相同名稱的方法，但可有不同的行為(處理內容程式碼)。抽象類別不能被建立物件，必須經繼承之後的子類別才能建立物件，所以不想被建立物件的類別，即使沒有抽象方法，只要設定為抽象類別就可以達到目的了。

抽象類別內可含抽象方法及非抽象方法，如果全部都為沒有實體的抽象method時，便可以使用下述之介面(Interface)了。

## 介面(Interface)

介面是定義類別的範本，由介面實作出的類別都長得一樣的外表。

介面是純粹的抽象類別(abstract class)，其下之method都只能宣告方法名稱、參數列、回傳型別，不能有方法實體。介面不像抽象類別，是不可包含一般method的。

介面關鍵字為interface，置於介面名稱前。

介面定義之欄位具static及final屬性，即具唯一性且不可改變其值。

介面不得用於建立物件。

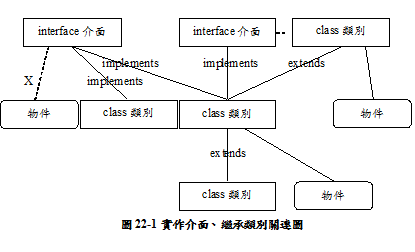
介面可使用implements關鍵字實作類別，如同類別被子類別繼承。

類別可同時實作(implements)多個介面，並繼承一個類別。

介面內方法(method)具有abstract屬性，不得有內容程式碼，但在方法之外可宣告欄位(變數)。介面被實作(implements)為類別時，須置入介面方法(method)之內容程式碼。

介面內方法具public屬性，不需宣告，但實作類別覆寫方法時須指定為

public。



### 實作介面

介面被視為抽象的，不可直接建立物件，須實作為類別後，方可建立物件。

<例>

interface Iffat {//定義介面

int x = 100;

void ifmd ();

}

public class Ifmain {

public static void main(String args[]){

Iffat aa = new Iffat();//建立物件，錯誤

System.out.println("x=" + aa.x);

}

}

編譯結果：C:\js>javac Ifmain.java

Ifmain.java:8: error: Iffat is abstract; cannot be instantiated

Iffat aa = new Iffat();//建立物件，錯誤

 ^

1 error

 類別可實作(implements)多個介面，並可同時繼承(extends)一個類別。

### 實作單一介面

interface Iffat1 {//定義介面

int x = 100;//宣告欄位，具有final屬性

void ifmd1();//宣告無內容之方法，具有public屬性

}

class Ifson1 implements Iffat1 {//實作介面

 public void ifmd1(){//覆蓋方法並置入內容，並須指定public

System.out.println("這是實作單一介面的例子");

}

}

public class Ifmain1 {

public static void main(String args[]){

Ifson1 aa = new Ifson1();

System.out.println("x=" + aa.x);

aa.ifmd1();

}

}

執行結果：C:\js>java Ifmain1

x=100

這是實作單一介面的例子

### 實作複數介面

interface Iffat2a {//定義介面

int x = 100;//宣告欄位

void ifmd2a();//宣告無內容之方法

}

interface Iffat2b {//定義介面

int y = 200;//宣告欄位

void ifmd2b();//宣告無內容之方法

}

class Ifson2 implements Iffat2a, Iffat2b {//實作多個介面

 public void ifmd2a(){//覆蓋方法並置入內容

System.out.println("這是實作複數介面的例子Iffat2a");

}

public void ifmd2b(){//覆蓋方法並置入內容

System.out.println("這是實作複數介面的例子Iffat2b");

}

}

public class Ifmain2 {

public static void main(String args[]){

Ifson2a aa = new Ifson2a();

System.out.println("x=" + aa.x);

System.out.println("y=" + aa.y);

aa.ifmd2a();

aa.ifmd2b();

}

}

執行結果：C:\js>java ifmain2

x=100

y=200

這是實作複數介面的例子Iffat2a

這是實作複數介面的例子Iffat2b

### 實作介面及繼承類別

interface Iffat3 {//定義介面

int x = 100;//宣告欄位

void ifmd3();//宣告無內容之方法

}

class Clfat3 {//定義類別

int z = 300;//宣告欄位

public void clmd3(){

System.out.println("實作介面及繼承類別的例子Clfat3");

}//類別可宣告有內容之方法

}

class Ifson3 extends Clfat3 implements Iffat3 {//實作介面並繼承類別

public void ifmd3(){//覆蓋方法並置入內容

System.out.println("實作介面及繼承類別的例子Iffat3");

}

}

public class Ifmain3 {

public static void main(String args[]){

Ifson3 aa = new Ifson3();

//aa.x = aa.x + 100;介面定義之欄位具final屬性，不可改變其值

System.out.println("x=" + aa.x);

aa.z = aa.z + 200;//類別定義之欄位可改變其值

System.out.println("z=" + aa.z);

aa.ifmd3();

aa.clmd3();

}

}

執行結果：C:\js>java ifmain3

x=100

z=500

實作介面及繼承類別的例子Iffat3

實作介面及繼承類別的例子Clfat3

## 錯誤範例

### 介面方法具abstract屬性

介面之方法具有abstract屬性，雖不必明示，但卻不能有執行敘述碼內容。

<錯誤例>

interface Iffat4 {

int x = 100;

void ifmd4(){

System.out.println("介面方法具abstract屬性，不能有執行碼");

}

}

編譯結果：C:\js>javac Iffat4.java

Iffat4.java:3: error: interface abstract methods cannot have body

void ifmd4(){

^

1 error

### 介面方法具public屬性

介面內方法具public屬性，不需宣告，但實作類別覆寫方法時須指定為

public。

<錯誤例>

interface Iffat5 {

int x = 500;

void ifmd5();//方法具有public屬性，不須宣告

}

class Ifson5 implements Iffat5 {//實作介面

 void ifmd5(){//覆蓋方法並置入內容，但無指定public

System.out.println("沒指定public之錯誤範例");

}

}

編譯結果：C:\js>javac Ifson5.java

Ifson5.java:7: error: ifmd5()in Ifson5 cannot implement ifmd5()in Iffat5

void ifmd5(){//覆蓋方法並置入內容，但無指定public

 ^

attempting to assign weaker access privileges; was public

1 error

### 介面欄位具final屬性

<錯誤範例>

interface Iffat6 {

int x = 600;//宣告欄位，不須明示即具有final屬性

void ifmd6();

}

class Ifson6 implements Iffat6 {

public void ifmd1(){

x = x+400;//不可變更final欄位值

System.out.println("變更final欄位值，x = " + x);

}

}

編譯結果：C:\js>javac Ifson6.java

Ifson6.java:6: error: Ifson6 is not abstract and does not override abstract meth

od ifmd6()in Iffat6

class Ifson6 implements Iffat6 {

^

Ifson6.java:8: error: cannot assign a value to final variable x

x = x+400;//不可變更final欄位值

^

2 errors

### 介面欄位具static屬性

介面宣告之欄位具static及final屬性，即具唯一性且不可改變其值。因不可改變其值，則非static屬性亦無意義，且難以一般程式證明此點。

非static屬性欄位建立多個物件，各物件都擁有各自位址的欄位，都可各自變更其值而不互相影響。static屬性欄位為當建立多個物件時，都使用同一位址之物件變數，任何一個物件static欄位更改時，都是變更各物件同一變數之值，可參見前面章節關於靜態static欄位之說明。