

# Java Fundamentals

## Object-Oriented

Alvin

fcaibi@gmail.com

# Content

2

- 類別與物件
- 封裝
  - ▣ 封裝的概念
  - ▣ Java語言實作封裝
  - ▣ 存取權限修飾字
- 建構子

# Class & Object

3

- 在物件導向的程式語言中，類別是一個不可或缺的機制，它擔負著架構物件的重責大任，它的主要用途是在規劃物件的內部構造，表示物件與物件間的關係，聯絡管道及運作方式。
- 物件是以類別來建立的。也就是類別定義了物件的架構，然後系統才能根據類別中所定義的架構產生實際物件。
  - 物件導向程式都是以類別為基本單位所組成的。
  - 物件導向程式在執行時的主體是物件，而非類別。

# Example : Class & Object(1)

4

```
1. class C {  
2.     int i;  
3. }  
  
4. public class D {  
5.     public static void main(String[] args) {  
6.         C c1;           // 宣告一個物件 c1  
7.         c1 = new C();    // 產生一個物件 c1  
8.     }  
9. }
```

# Example : Class & Object(1)

5

- `int i`
  - ▣ 類別 `C` 宣告了一整數變數 `i`。
  - ▣ 註：資料型態包含 `int`, `float`, `double`, `char` 以及 `boolean` 等。
- `C c1;`
  - ▣ 類別 `D` 以類別 `C` 為範本，宣告了一個物件，它的名字為 `c1`。
- `c1 = new C();`
  - ▣ 等號右邊，`new C()` 的意義是：以類別 `C` 為範本產生一個物件。
  - ▣ 經過指定運算 (等號) 後，這個以類別 `C` 為範本所產生的新物件，它的名字就叫做 `c1`。
- 兩者可合寫為 `C c1 = new C();`
- 在 `Java` 中，使用物件中的變數和方法方式是：
  - ▣ 物件名稱.變數名稱
  - ▣ 物件名稱.方法名稱

# Example : Class & Object(2)

6

```
1. class C {  
2.     int i;  
3. }  
  
4. public class D {  
5.     public static void main(String[] args) {  
6.         C c1 = new C();  
7.         c1.i = 10;  
8.         System.out.println("c1.i = "+c1.i);  
  
9.         c1.i = 20;  
10.        System.out.println("c1.i = "+c1.i);  
11.    }  
12. }
```

輸出結果：  
c1.i = 10  
c1.i = 20

# Example : Class & Object(3)

7

```
1. class Account {  
2.     private int balance;  
  
3.     void clearAccount() { balance = 0; }  
4.     void deposit(int m) { balance = balance + m; }  
5.     int getBalance() { return balance; }  
6. }
```

// 宣告一帳戶餘額

// 清空帳戶餘額的 "方法"

// 存錢的 "方法"

// 顯示目前餘額的 "方法"

```
7. public class E {  
8.     public static void main(String[] args) {  
9.         Account joe = new Account();  
10.        Account wason = new Account();  
11.        joe.clearAccount();  
12.        wason.clearAccount();  
13.        joe.deposit(300);  
14.        wason.deposit(500);
```

// 產生一個帳戶：joe

// 產生一個帳戶：wason

// 將 Joe 的帳戶餘額清空

// 將 Wason 的帳戶餘額清空

// Joe 存了 300 元

// Wason 存了 500 元

```
15.        System.out.println("Joe has " + joe.getBalance() + " dollars.");  
16.        System.out.println("Wason has " + wason.getBalance() + " dollars.");  
17.    }  
18. }
```

// 顯示 Joe 目前的帳戶餘額

// 顯示 Wason 目前的帳戶餘額

執行結果：

Joe has 300 dollars.

Wason has 500 dollars.

# Example : Class Array

8

- 陣列不僅可以儲存大量基本類型，也可以儲存大量物件。

範例 ObjectArray.java

```
1. class Book{
2.     String name;
3.     double price;
4.     String author;
5.     Book(String n, double p, String a){
6.         name = n;
7.         price = p;
8.         author = a;
9.     }
10.    void show(){
11.        System.out.println("書名：" + name);
12.        System.out.println("定價：" + price);
13.        System.out.println("作者：" + author);
14.    }
15. }
16.
17. class ObjectArray{
18.     public static void main(String[] args){
19.         Book[] books = new Book[2];
20.         books[0] = new Book("Java 程式設計", 580.0, "張振風");
21.         books[1] = new Book("JSP 程式設計", 650.0, "黃會紅");
22.         for(Book book : books)
23.             book.show();
24.     }
25. }
```

-----輸出-----

書名：Java 程式設計  
定價：580.0  
作者：張振風  
書名：JSP 程式設計  
定價：650.0  
作者：黃會紅



# Content

9

- 類別與物件
- 封裝
  - ▣ 封裝的概念
  - ▣ Java語言實作封裝
  - ▣ 存取權限修飾字
- 建構子

# Encapsulation

10

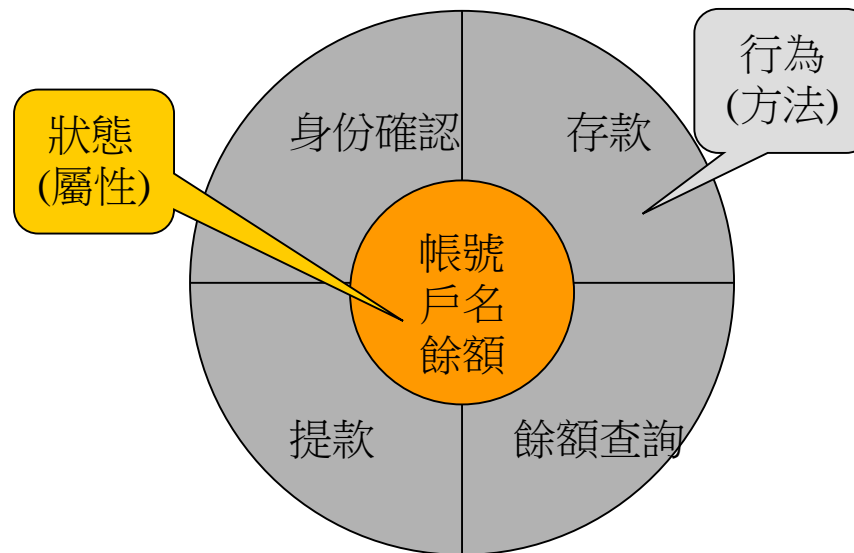
- 將資料及使用此資料的所有方法包裝成一個物件。
- 封裝之特性使物件導向的系統較容易維護。
- Java的封裝是將物件中資源(資料或方法)的存取分為幾個等級，以便來管理如何將物件中某些資源隱藏在物件中，某些資源應該開放給外界使用。

# 封裝 Encapsulation

11

## □ 封裝 encapsulation

- ▣ **保護**類別中的資料,不讓資料被誤用或破壞
- ▣ 隱藏實作的細節,**增加應用程式可維護性**



# Encapsulation

12

□ 資源(資料或方法)的存取分為四個等級：

□ **Public** (公開)



- 將類別內部的資源開放給外界使用。

□ **Private** (私有)



- 資源的所有權已完全屬於該類別所有，**只有在類別內部**才可對其作存取動作。
- 任何外部的存取均會導致錯誤發生。

□ **Default Access** (預設存取)



- 在所屬的 package 中是被視為 public 資源；
- 但是在其他的 package 中，**則會被視為是 private 資源而無法被使用**。

□ **Protected** (保護)



- 在所屬的 package 中是被視為 public 資源；
- 但是在其他的 package 中，**則只被繼承的子類別使用**。

# 封裝的方法

13

- 封裝的方法
  - ▣ 更改屬性為private
  - ▣ 提供存取屬性的方法
    - Accessor (getter & setter)
    - Setter提供保護資料的邏輯
  - ▣ 存取此類別之資料,需使用類別所提供的方法來存取

# Java 語言實作封裝

14

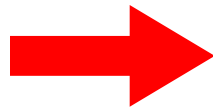
```
public class MyDate {  
    public int day;  
    public int month;  
    public int year;  
}
```

```
public class TestMyDate {  
    public static void main(String args[]) {  
        MyDate d = new MyDate();  
  
        d.day = 30;  
        d.month = 2;  
        d.year = 2003; } Invalid date  
  
        System.out.println(d.day + "/" + d.month + "/" + d.year);  
    }  
}
```

# Java 語言實作封裝

15

```
public class MyDate {  
    public int day;  
    public int month;  
    public int year;  
}
```



```
public class MyDate {  
    private int day;  
    private int month;  
    private int year;  
  
    public void setDate(int d, int m, int y) {  
        .....  
        .....  
    }  
  
    public String getDate() {  
        return day + "/" + month + "/" + year;  
    }  
}
```

setter method  
setXXX( )

getter method  
getXXX( )

# Java 語言實作封裝

16

```
public class MyDate {  
    private int day;  
    private int month;  
    private int year;  
  
    public void setDate(int d, int m, int y) {  
        .....  
        .....  
    }  
  
    public String getDate() {  
        return day + "/" + month + "/" + year;  
    }  
}
```

```
public class TestMyDate {  
    public static void main(String args[]) {  
        MyDate d = new MyDate();  
  
        d.day = 30;  
        d.month = 2;  
        d.year = 2003; } compile error!  
  
        System.out.println(d.day + "/" + d.month +  
                           "/" + d.year);  
  
        d.setDate(28,2,2003);  
  
        System.out.println("Date: " + d.getDate());  
    }  
}
```



# 封裝 Shirt 類別

17

```
01 public class Shirt {
02     private int shirtID = 0;
03     private char colorCode = 'G';
04     private String size = "XL" ;
05     private double price = 299.00;
06
07     public void setColorCode(char c) {
08         if(c=='R' || c=='G' || c=='Y')
09             colorCode = c;
10     }
11     public double getColorCode ( ) {
12         return colorCode;
13     }
14     public void setSize(String s) {
15         if(s.equals("S") || s.equals("M") ||
16            s.equals("L") || s.equals("XL") )
17             size = s;
18     }
19     public String getSize( ) {
20         return size;
21     }
22     public void setPrice(double p) {
23         if(p>=0.0)
24             price = p;
25     }
26     public double getPrice( ) {
27         return price;
28     }
29
30 }
```

# Example : Encapsulation

18

```
1. class C {  
2.     private int i;           // 私有資料  
3.     public int j;           // 公開資料  
4.     int k;                   // 預設存取資料  
5. }  
  
6. public class D {  
7.     public static void main(String[] args) {  
8.         C c = new C();  
9.         c.i = 5;              // Error! c.i 是 private 資料，禁止存取！  
10.        c.j = 10;             // OK!  
11.        c.k = 15;             // OK!  
12.    }  
13. }
```

# Example : Encapsulation

19

```
1. class Account {
2.     private int balance;                // 私有資料
3.     public void clearAccount() { balance = 0; }    // 公開方法
4.     void deposit(int m) { balance = balance + m; } // 預設存取方法
5.     private int getBalance() { return balance; }  // 私有方法
6. }

7. class D {
8.     public static void main(String[] args) {
9.         Account joe = new Account();
10.        Account wason = new Account();
11.
12.        joe.clearAccount();                // OK!
13.        wason.clearAccount();              // OK!
14.        joe.deposit(300);                  // OK!
15.        wason.deposit(500);                // OK!

16.        System.out.println("Joe has " + joe.getBalance() + "dollars. ");    // ERROR! 使用私有方法
17.        System.out.println("Wason has " + wason.getBalance() + "dollars. "); // ERROR! 使用私有方法
18.    }
19. }
```

# Content

20

- 類別與物件
- 封裝
- **建構子**
  - ▣ 物件屬性欄位初始化
  - ▣ 建構子
  - ▣ 預設建構子
  - ▣ 建構子多載

# Java 建構子

21

Shirt
+shirtID: int +colorCode: char +size: String +price: double +description : String
+Shirt (c: char, s: String, p: double, d: String)
+setPrice(p: double ) +getPrice ( ) : double +displayInformation ( )

```
01 public class Shirt {
02
03     public int shirtID = 0;
04     public char colorCode = 'G';
05     public String size = "XL" ;
06     public double price = 299.00;
07     public String description = "Polo Shirt";
08
09     public Shirt(char c, String s, double p, String d) {
10         colorCode = c;
11         size = s;
12         price = p;
13         description = d;
14     }
15
16     public void setPrice(double p) {
17         price = p;
18     }
19     public double getPrice ( ) {
20         return price;
21     }
22     public void displayInformation() {
23         System.out.println("Shirt ID:" + shirtID);
24         System.out.println("Color:" + colorCode);
25         System.out.println("Size:" + size);
26         System.out.println("Price:" + price);
27     }
28
29 }
30
```

建構子

# Constructor

22

- 在定義類別時，可以使用「建構式」(Constructor)來進行物件的初始化。
- 「建構式」就是將類別實體化建立成物件時，所執行的方法。會在物件產生之後自動被呼叫，建構物件初始的狀態，因此稱為建構式(建構方法)。
- 建構式名稱必須與類別名稱相同，建構式不得指定傳回值。
- 例如：

```
public class SafeArray {  
    // ..  
    public SafeArray() { // 建構方法  
        // ....  
    }  
    public SafeArray(參數列) { //  
        // ....  
    }  
}
```

如果沒有定義任何的建構方法，則編譯器會自動配置一個無參數且沒有陳述內容的建構式。

程式在運行時，會根據配置物件時所指定的引數資料型態等，來決定該使用哪一個建構式新建物件。

# 建構子 constructor 語法

23

```
[modifiers] class <class_name> {  
    [modifiers] constructor_name([arguments]) {  
        ↪ Accessibility ↪ Same as class_name ↪ Arguments list  
        code_blocks  
    }  
}
```

- 與類別名稱一樣
- 沒有回傳型態
- 可以多載(Overloading)
- 預設建構子

# 物件建構流程 – 宣告

24

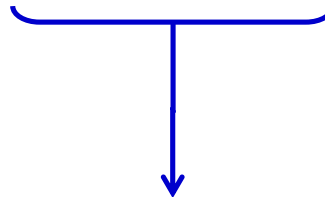
```
public class Shirt {  
    private int shirtID = 101;  
    private char colorCode = 'R';  
    private double price = 299.0;  
    private String description = "Polo Shirt";  
  
    public Shirt(char c, double p, String d){  
        colorCode = c;  
        price = p;  
        description = d;  
    }  
}
```

**Shirt myShirt;**

myShirt = new Shirt('G', 199.0 , "T-Shirt" );

myShirt

????



Stack memory



# 物件建構流程 – 實體化

## 記憶體配置

25

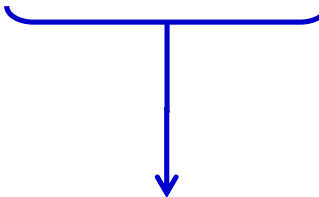
```
public class Shirt {  
    private int shirtID = 101;  
    private char colorCode = 'R';  
    private double price = 299.0;  
    private String description = "Polo Shirt";  
  
    public Shirt(char c, double p, String d){  
        colorCode = c;  
        price = p;  
        description = d;  
    }  
}
```

Shirt myShirt;

myShirt = new Shirt ('G', 199.0 , "T-Shirt" );

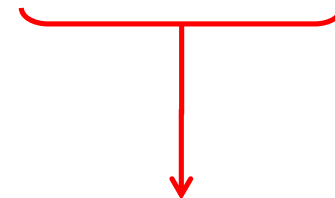
myShirt

????



Stack memory

0	shirtID
' '	colorCode
0.0	price
NULL	description



Heap Memory

# 物件建構流程 – 初始化

## 初始值賦值

26

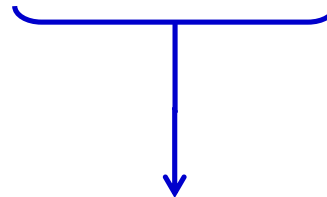
```
public class Shirt {  
    private int shirtID = 101;  
    private char colorCode = 'R';  
    private double price = 299.0;  
    private String description = "Polo Shirt";  
  
    public Shirt(char c, double p, String d){  
        colorCode = c;  
        price = p;  
        description = d;  
    }  
}
```

Shirt myShirt;

myShirt = new Shirt ('G', 199.0 , "T-Shirt" );

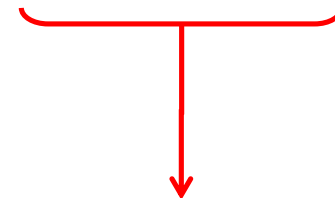
myShirt

????



Stack memory

101	shirtID
'R'	colorCode
299.0	price
"Polo Shirt"	description



Heap Memory

# 物件建構流程 – 初始化

## 執行建構式

27

```
public class Shirt {  
    private int shirtID = 101;  
    private char colorCode = 'R';  
    private double price = 299.0;  
    private String description = "Polo Shirt";  
  
    public Shirt(char c, double p, String d){  
        colorCode = c;  
        price = p;  
        description = d;  
    }  
}
```

Shirt myShirt;

myShirt = new **Shirt** ('G', 199.0 , "T-Shirt" );

myShirt

????

Stack memory

101	shirtID
'G'	colorCode
199.0	price
"T-Shirt"	description

Heap Memory

# 物件建構流程 – 儲存物件參考

28

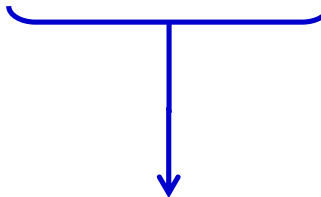
```
public class Shirt {  
    private int shirtID = 101;  
    private char colorCode = 'R';  
    private double price = 299.0;  
    private String description = "Polo Shirt";  
  
    public Shirt(char c, double p, String d){  
        colorCode = c;  
        price = p;  
        description = d;  
    }  
}
```

Shirt myShirt;

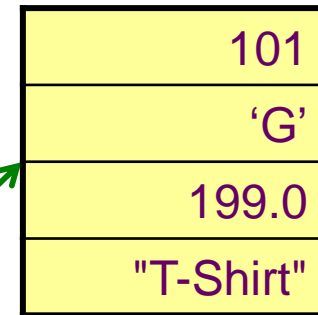
**myShirt** = new Shirt('G', 199.0 , "T-Shirt" );

myShirt

0x01234567



Stack memory



101

shirtID

'G'

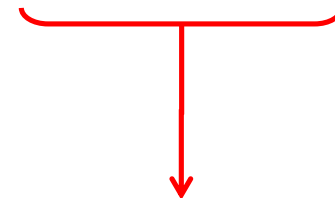
colorCode

199.0

price

"T-Shirt"

description



Heap Memory

# Constructor

29

- 建構式雖然很像方法，但是有3個不同點：
  1. 呼叫時機不同
  2. 建構式無回傳值
  3. 建構式名稱與類別相同
- 建構式可分成：
  1. 預設建構式
  2. 非預設建構式

# 預設建構子 Default Constructor

30

- 物件裡面一定要有建構子，所以在撰寫class時必須定義該物件的建構子
- 程式中若沒有定義建構子，在編譯時期會自動加入，所加入的就稱之為預設建構子；
  - 預設建構子沒有參數列(no arguments)；
  - 除了初始物件變數或繼承時super()的定義外，預設建構子沒有其他的程式敘述(no body statement)。
  - 自行建立後預設建構子即失效

```
01 public class Shirt {  
02     private int shirtID = 101;  
03     private char colorCode = 'R';  
04     private double price = 299.0;  
05     private String description = "Polo Shirt";  
06     public Shirt(){ }  
07  
08 }
```

```
01 public class TestShirt {  
02     public static void main(String[] args) {  
03         Shirt s = new Shirt();  
04     }  
05 }
```

javac Shirt.java

# Constructor

31

範例 Constructor.java

```
1. class Book{
2.     String name;
3.     double price;
4.     String author;
5.     Book(){
6.         name = "不詳";
7.         price = 0.0;
8.         author = "不詳";
9.     }
10.    Book(String n, double p, String a){ //非預設建構式
11.        name = n;
12.        price = p;
13.        author = a;
14.    }
15.    void show(){
16.        System.out.println("書名：" + name);
17.        System.out.println("定價：" + price);
18.        System.out.println("作者：" + author);
19.    }
20. }
21.
22. class Constructor{
23.     public static void main(String[] args){
24.         Book book1 = new Book("Java 程式設計", 580.0, "張振風");
25.         book1.show();
26.         Book book2 = new Book();
27.         book2.show();
28.     }
29. }
```

-----輸出-----

書名：Java程式設計  
定價：580.0  
作者：張振風  
書名：不詳  
定價：0.0  
作者：不詳

# 建構子多載 Constructors overloading

32

- 提供多組建構子為物件設定初值
  - ▣ 傳入參數數量或型態不同

```
01 public class Shirt {
02     private int shirtID = 101;
03     private char colorCode = 'R';
04     private double price = 299.0;
05     private String description = "Polo Shirt";
06
07     public Shirt(int id) {
08         shirtID = id;
09     }
10     public Shirt(char color, double newPrice) {
11         colorCode = color;
12         price = newPrice;
13     }
14     public Shirt(char color, double newPrice,
15                 String desc) {
16         colorCode = color;
17         price = newPrice;
18         description = desc;
19     }
20 }
```

```
01 public class TestShirt {
02     public static void main(String[] args) {
03
04         Shirt s1 = new Shirt();
05
06         Shirt s2 = new Shirt(101);
07
08         Shirt s3 = new Shirt('G', 600.0);
09
10         Shirt s4 = new Shirt('Y', 199.0, "T-Shirt");
11
12     }
13 }
14
```

The diagram illustrates constructor overloading. On the left, the `Shirt` class has three constructors: a no-argument constructor (line 07), a constructor with a `char` and a `double` (line 10), and a constructor with a `char`, a `double`, and a `String` (line 14). On the right, the `TestShirt` class has a `main` method (line 02) that creates four `Shirt` objects. The first line, `Shirt s1 = new Shirt();` (line 04), is crossed out with a red X. The second line, `Shirt s2 = new Shirt(101);` (line 06), is linked by a blue arrow to the no-argument constructor of `Shirt`. The third line, `Shirt s3 = new Shirt('G', 600.0);` (line 08), is linked by a red arrow to the constructor that takes a `char` and a `double`. The fourth line, `Shirt s4 = new Shirt('Y', 199.0, "T-Shirt");` (line 10), is linked by a purple arrow to the constructor that takes a `char`, a `double`, and a `String`.

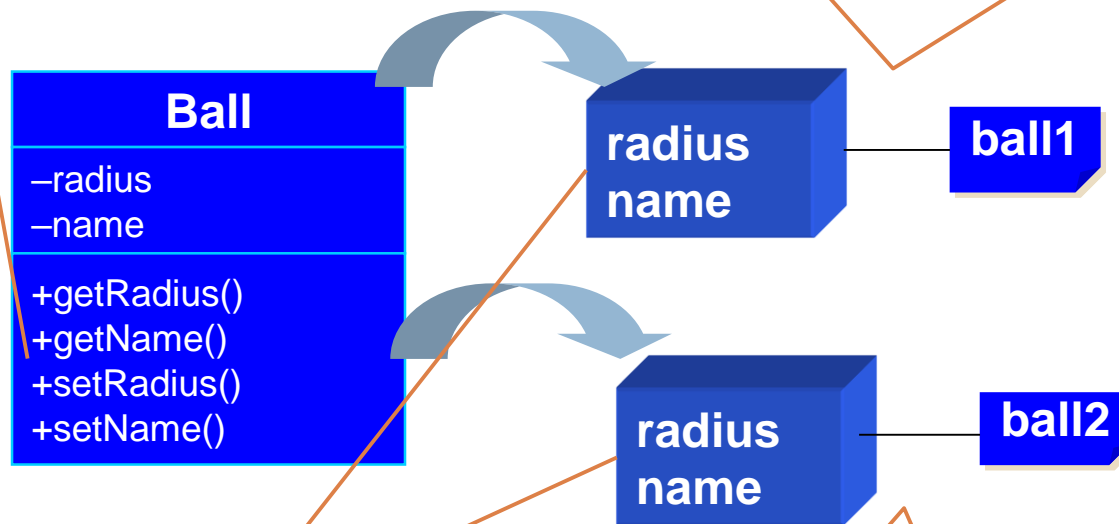


# About this

33

方法成員在記憶體中只有一份

新建實體，並以ball1名稱參考



各自擁有Field成員

新建實體，並以ball2名稱參考

# About this

34

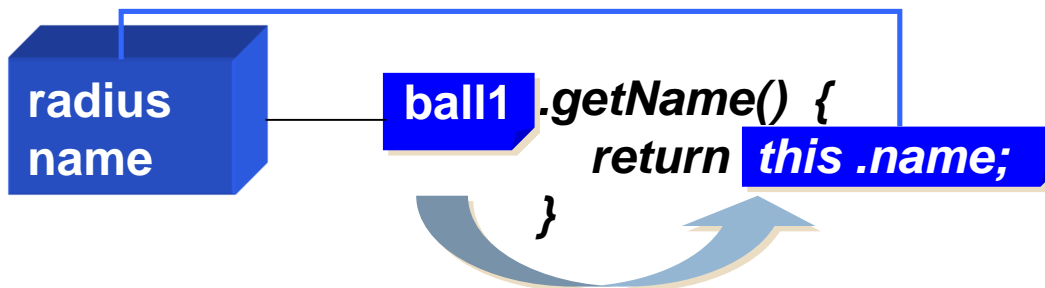
- 方法中所撰寫的每一個資料成員其實會隱含一個 **this** 參考名稱，這個 **this** 名稱參考至 **呼叫方法的物件**，當呼叫 `getName()` 方法時，其實相當於執行：

```
public double getName()  
{  
    return name;  
}
```



```
public double getName()  
{  
    return this.name;  
}
```

- 當使用 `ball1` 並呼叫 `getRadius()` 方法時，**this** 所參考的就是 `ball1` 所參考的物件：



# About this

35

- 當在方法中使用資料成員時，都會隱含的使用**this**名稱，當然也可以明確的指定，例如在方法定義時使用：

```
public Ball(double radius, String name) {  
  
    this.radius = radius;  
  
    this.name = name;  
  
}
```

- 參數名稱與資料成員名稱相同時，為了避免參數的作用範圍覆蓋了資料成員的作用範圍，必須明確的使用**this**名稱來指定，但如果參數名稱與資料成員名稱不相同則不用特別指定：

```
public Ball(double r, String n) {  
  
    radius = r;           // 實際等於this.radius = r;  
  
    name = n;             // 實際等於this.name = n;  
  
}
```

# Constructor & this

36

```
public class SafeArray {  
    private int[] arr;  
  
    public SafeArray() {  
        this(10); // 預設 10 個元素  
    }  
  
    public SafeArray(int length) {  
        arr = new int[length];  
    }  
  
    ....  
}
```

使用**this(10)**，  
這會呼叫另一個有參數的建構方法。

# Content

37

- 繼承(Inheritance)
  - ▣ 繼承觀念
  - ▣ 繼承實作
  - ▣ 方法覆寫
  - ▣ 繼承關係下的物件建構
- 多型(Polymorphism)

# Inheritance

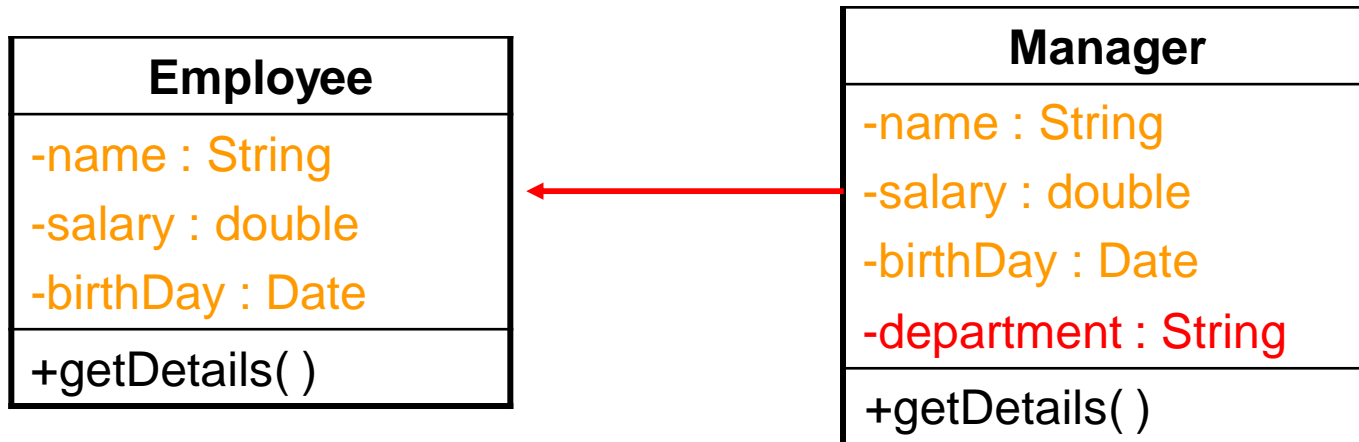
38

- 繼承是類別間之關係，在此關係中某類別之資料結構與行為可供其關係中之類別分享。
- 繼承者稱為子類別(Subclass)，被繼承者稱為父類別(Superclass)。
- Java使用extends來表示繼承的關係。
- 父類別定義的屬性與方法，子類別當然也適用，不過要成為子類別，必須透過繼承(使用extends關鍵字)。當然子類別也可以自行定義新的成員，以表現本身的特質，但是父類別物件無法存取子類別定義的成員。
- final類別不可被繼承
  - ▣ final class Book{}
  - ▣ class ComputerBook extends Book{} //編譯失敗，因為Book類別不可被繼承

# 類別的繼承

39

- 繼承讓類別的程式碼可以延伸及重複使用
  - ▣ 父類別 base class/superclass
  - ▣ 子類別 derived class/subclass
  - ▣ 類別延伸(extends class)的關係是 "is a" 的概念。



A Manager is a Employee

# Inheritance

40

- 下例中，類別 D 繼承類別 C：

```
class C {  
    ...  
}
```

```
class D extends C {  
    ...  
}
```

- 當類別 D 繼承了類別 C 後，類別 C 中一切可以被繼承的事物都將變成類別 D 中的一部分。



# Example : Inheritance

41

```
1.  class C {
2.      private int i;
3.      public void setInfo(int x) { i = x; }
4.      public int getInfo() { return i; }
5.  }
6.  class D extends C {} // 類別 D 繼承類別 C
7.  class E extends C {} // 類別 E 繼承類別 C

8.  public class F {
9.      public static void main(String[] args) {
10.
11.          D d = new D();
12.          E e = new E();
13.
14.          d.setInfo(5);
15.          e.setInfo(7);
16.
17.          System.out.println("The value of d is "+d.getInfo());
18.          System.out.println("The value of e is "+e.getInfo());
19.      }
20.  }
```

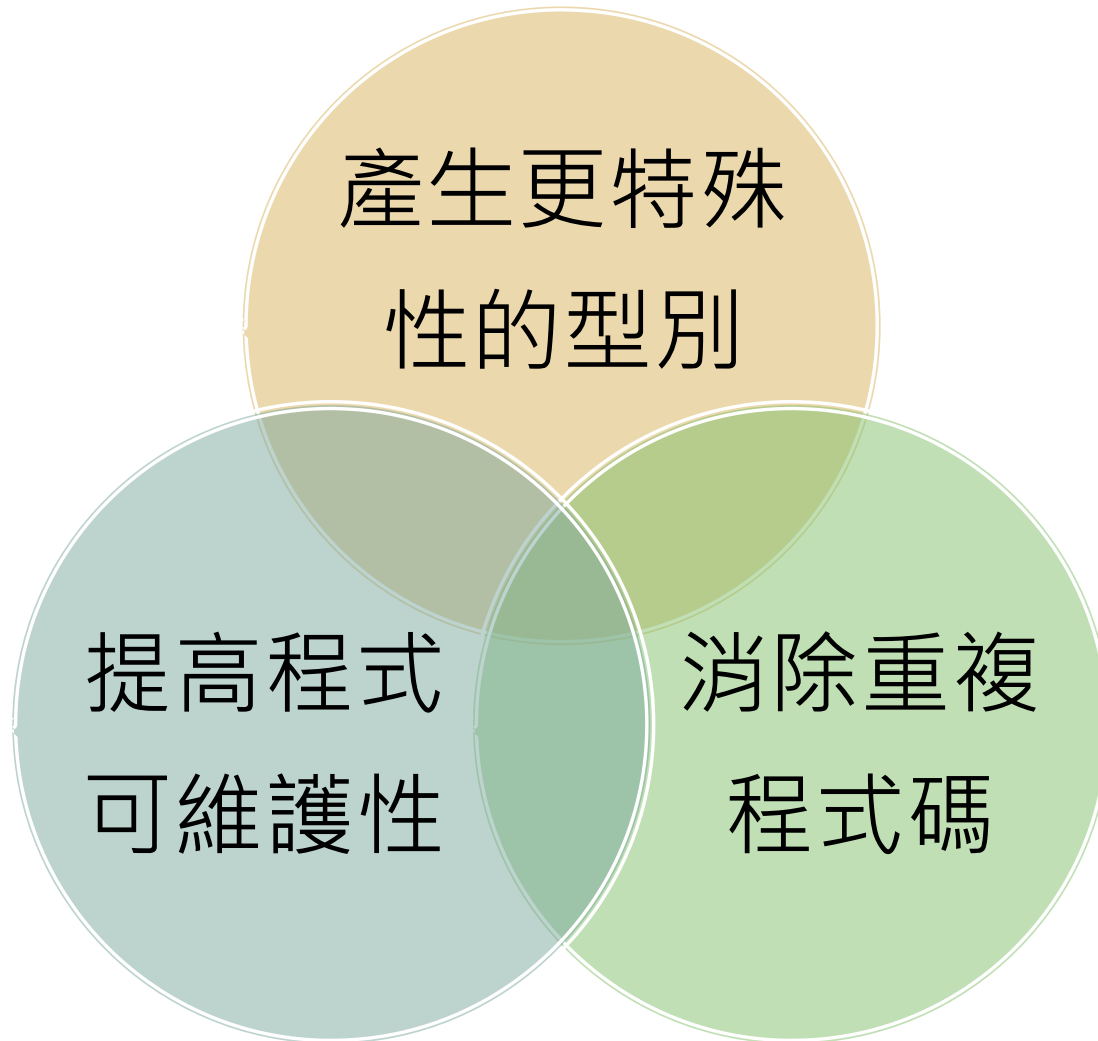
執行結果：

**The value of d is 5**

**The value of e is 7**

# 繼承機制的優點

42



# is-a 與 has-a

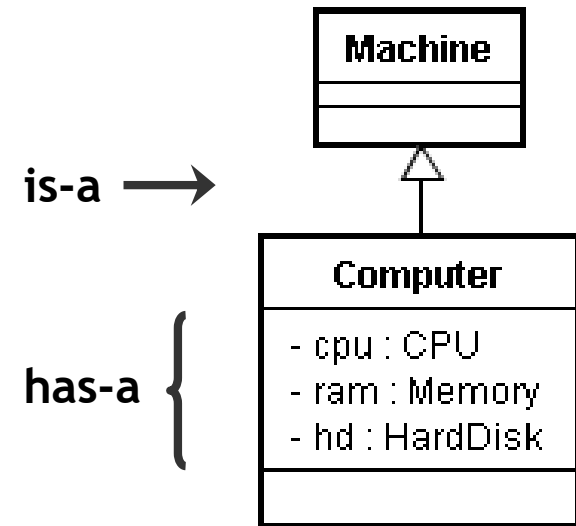
43

## ■ is a (是一個) :

- 延伸的關係。
- Java 語言利用 **extends** 關鍵字來實作
- EX : 電腦是一種電子機械產品
  - ✓ 電腦類別繼承了電子機械產品類別。
  - ✓ 電子機械產品是父類別，而電腦則是子類別。

## ■ has a (有一個) :

- 聚合的關係。
- 類別中的成員變數(member variable)來表示。
- EX : 電腦中有 CPU、256M RAM、40GB HD
  - ✓ CPU、RAM 與 HD 便成了電腦的成員變數。



# Java 技術實作繼承

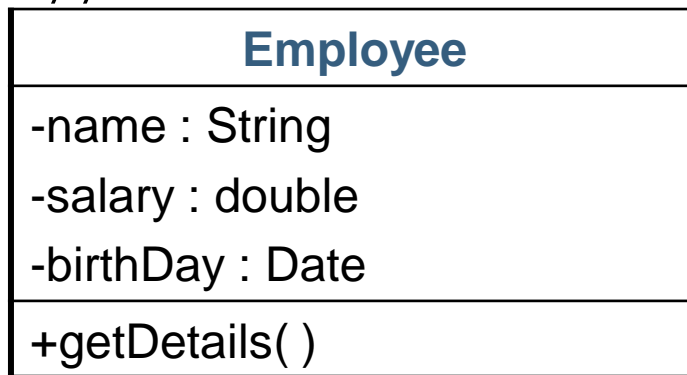
44

## ■ 建立類別語法

[modifier] class 子類別名稱 **extends** 父類別名稱 {

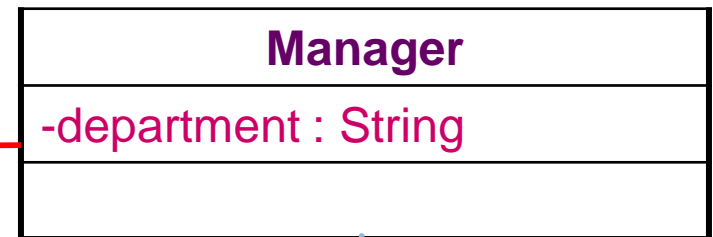
//類別內容

}



```
01 public class Employee {
02     private String name = "Sean";
03     private double salary = 10000;
04     public void getDetails( ) {
05         System.out.println("Name:" + name);
06         System.out.println("Salary:" + salary);
07     }
08 }
```

Manager 繼承了 Employee 所有成員;  
屬性:name, salary, birthDay  
方法:getDetails()方法



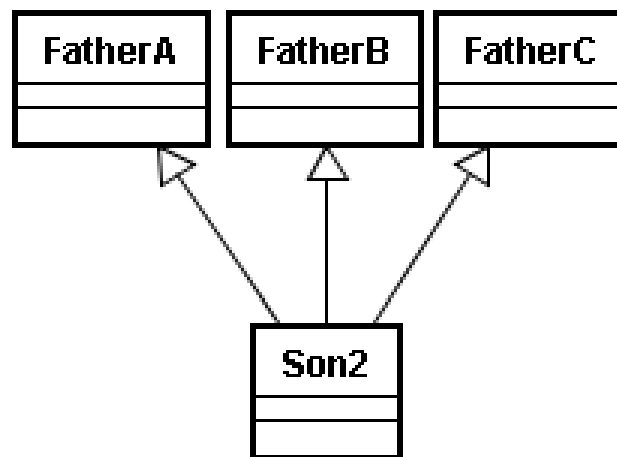
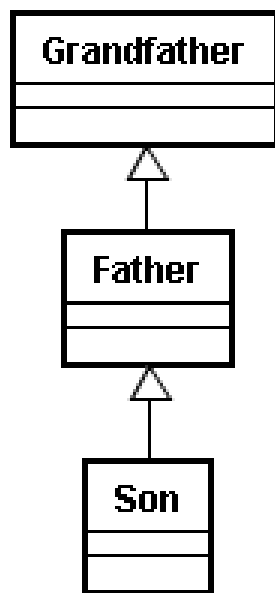
```
01 public class Manager extends Employee {
02     private String dept = "EDU";
03 }
04 }
05 }
```

# 單一繼承

45

- Java 語言在繼承上只允許單一繼承(Single Inheritance)關係。
  - ▣ 子類別在定義繼承的關係時，只能針對單一父類別做延伸，不能同時使用來自多個父類別的資源。

單一繼承(Java 支援)



多重繼承C++等：支援

# 繼承中的資源使用

46

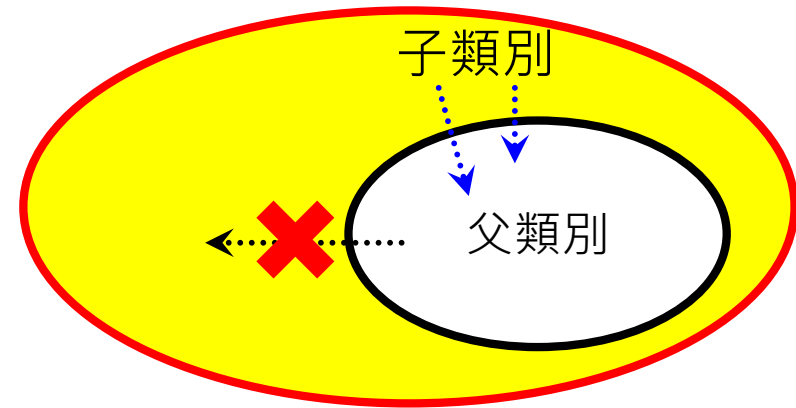
- 當子類別繼承父類別，擁有父類別中的資源

- ▣ 子類別繼承(擁有)父類別成員(屬性和方法)

- 實作：子類別中包含父類別

- ▣ 父類別不擁有子類別屬性和方法

- ▣ **建構子不會被繼承**



- 子類別存取父類別資源

- ▣ 存取父類別的屬性：直接使用屬性名稱;

- ▣ 呼叫父類別的方法：直接使用方法名稱(參數列);

- ▣ 但存取權限仍受限於父類別的存取修飾字

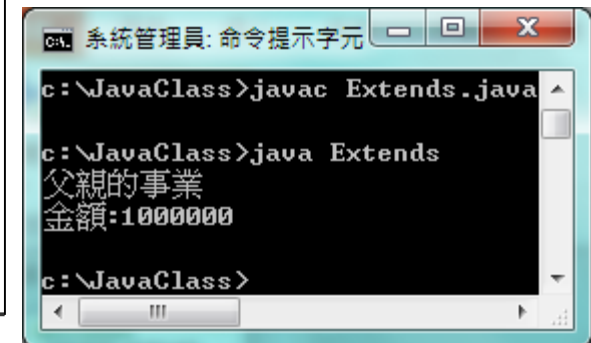
# 繼承範例

47

```
01 class Father {  
02     public int money = 1000000; ←  
03     public void undertaking() { ←  
04         System.out.println("父親的事業");  
05     }  
06 }
```

```
01 class Son extends Father{  
02  
03 }
```

```
01 public class Extends {  
02     public static void main(String[] args) {  
03         Son son = new Son();  
04         son.undertaking(); ←  
05         System.out.println("金額:" + son.money); ←  
06     }  
07 }
```



```
系統管理員: 命令提示字元  
c:\JavaClass>javac Extends.java  
c:\JavaClass>java Extends  
父親的事業  
金額:1000000  
c:\JavaClass>
```

# Inheritance

48

- 子類別將由父類別繼承來的屬性(變數、資料結構)或方法(行為)重新定義的動作稱為覆寫(overriding)。
- 方法覆寫
  - ▣ 將繼承自父類別的方法遮蓋起來，使得以子類別所產生的物件不能再使用已經被覆蓋的方法。
  - ▣ 這種設計主要是在繼承類別時，可以將不符合需求的方法加以改寫，如此一來，子類別的使用者就不會也不能看到及使用父類別中的方法，達到重新設計的目的。
- 變數覆寫
  - ▣ 將繼承自父類別的變數遮蓋起來。事實上，當變數產生覆蓋問題時，在父類別中的變數宣告並不會因此而消失，它只是隱藏起來而已。



# Example : Inheritance

49

## □ 變數覆寫

```
1. class C {  
2.     int i = 10;  
3. }  
  
4. class D extends C {  
5.     int i = 5;  
6. }  
  
7. public class E {  
8.     public static void main(String[] args) {  
9.         D d = new D();  
10.        System.out.println("d = "+d.i);  
11.    }  
12. }
```

執行結果：

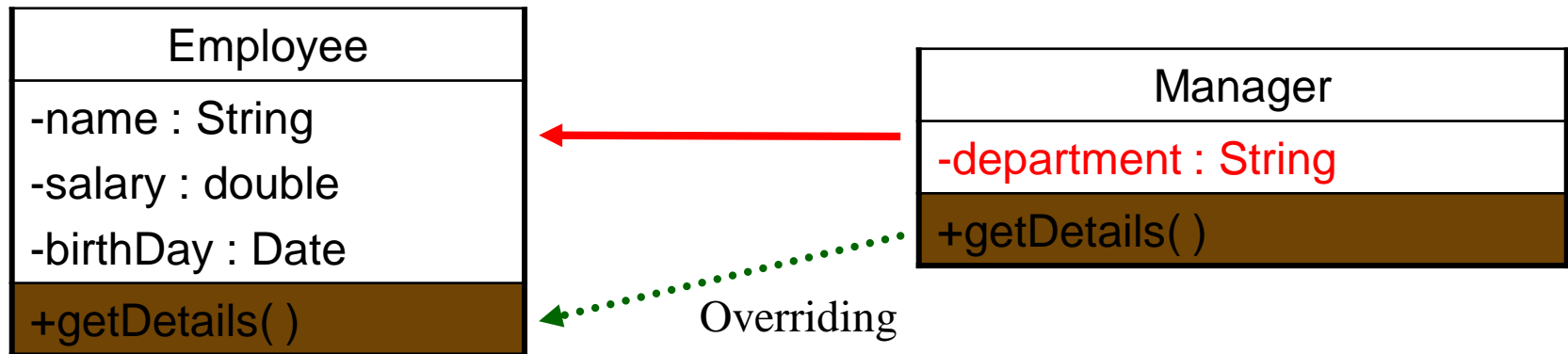
d = 5

# 方法覆寫 Method Override

50

## □ 方法覆寫 Method Override

- ▣ 子類別改寫父類別中相同的名稱及參數列的方法



# 方法覆寫 Method Override

51

```
01 public class Employee {
02     private String name = "Sean";
03     private double salary = 10000;
04     public void getDetails( ) {
05         System.out.println("Name:" + name);
06         System.out.println("Salary:" + salary);
07     }
08 }
```

```
01 public class Manager extends Employee {
02     private String dept = "EDU";
03     @Override
04     public void getDetails( ) {
05         System.out.println("Name:" + name);
06         System.out.println("Salary:" + salary);
07         System.out.println("Department:" + dept);
08     }
09 }
```

```
01 public class Test {
02     public static void main(String [ ] args) {
03         Employee e = new Employee();
04         e.getDetails( );
05         Manager m = new Manager ( );
06         m.getDetails( );
07     }
08 }
```

# 呼叫被覆寫的方法 - super

52

## □ super 關鍵字

- 子類別物件中欲參考父類別物件的屬性、方法及建構子
  - super.屬性
  - super.方法(參數列)
  - super(參數列)
- super 關鍵字必須在繼承關係的運作下才有意義。

```
public class Employee {  
    private String name = "Sean";  
    private double salary = 10000;  
    public void getDetails( ) {  
        System.out.println("Name:" + name);  
        System.out.println("Salary:" + salary);  
    }  
}
```

```
public class Manager extends Employee {  
    private String dept = "EDU";  
    public void getDetails( ) {  
        super.getDetails( );  
  
        System.out.println("Department:" + dept);  
    }  
}
```

# 存取被遮蔽的屬性 - this

54

```
public class MyDate {  
    private int year = 2000;  
    private int month = 1;  
    private int day = 1;  
  
    public MyDate(int day, int month, int year) {  
        this.day = day;  
        this.month = month;  
        this.year = year;  
    }  
}
```

## □ this 關鍵字

- 編譯時期自動加入,代表本身物件的參考(Reference)
- 方法或建構子中,欲參考被遮蔽的物件屬性及方法
  - this.屬性
  - this.方法(參數列)

# 繼承關係下的建構子

55

- 子類別產生物件時,需先建構父類別
  - ▣ 先初始化父類別的屬性及方法後,才執行本身初始化
  - ▣ 追溯到 `java.lang.Object` 為止
- `super()`和`this()`
  - ▣ `super()`：呼叫父類別建構子的方法
  - ▣ `this()`：呼叫類別自己其他建構子的方法
  - ▣ 注意事項
    - 只能用在建構子程式碼第一行,一次只能使用一種
    - 只能用在建構子之中,不能在程式其他位置出現
    - 建構子之中,若沒有使用`super`或`this`,JVM自動在第一行加上`super()`

# Default Constructor

56

```
public class Employee {  
    private String name = "Sean";  
    private double salary = 10000;  
    public Employee() {  
        super();  
    }  
    public void getDetail() {  
        System.out.println("Name:" + name);  
        System.out.println("Salary:" + salary);  
    }  
}
```

Manager m = new Manager();

```
public class Manager extends Employee {  
    private String dept = "EDU";  
    public Manager() {  
        super();  
    }  
    public void getDetail() {  
        super.getDetail();  
        System.out.println("Department:" + dept);  
    }  
}
```

# Constructors & super()

57

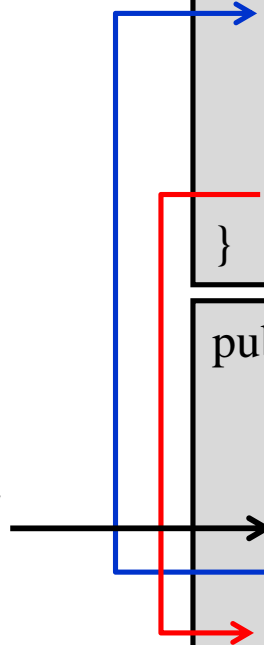
```
public class Employee {  
    private String name = "Sean";  
    private double salary = 10000;
```

```
    public Employee(String n, double s) {  
        name = n;  
        salary = s;  
    }  
}
```

```
public class Manager extends Employee {  
    private String dept = "EDU";
```

```
    public Manager(String n, double s, String d) {  
        super(n, s);  
        dept = d;  
    }  
}
```

**Manager m = new Manager  
("Sean", 50000.0, "EDU");**





# Inheritance & Constructor

58

- 若使用**super**或**this**呼叫其他建構式，該呼叫式必須放在建構式區塊內的第1行

```
ComputerBook(String n, double p, String a, boolean h){  
    super(n, p, a); //正確，放在建構式內的第1行  
    hasDisk = h;  
}
```

```
ComputerBook(String n, double p, String a, boolean h){  
    hasDisk = h;  
    super(n, p, a); //錯誤，必須放在建構式內的第1行  
}
```

- 建構式進階觀念

1. 任何類別的建構式內都必須存在呼叫父類別建構式的呼叫式，如果沒有則編譯器會自動加上呼叫父類別預設建構式的呼叫式 - 「**super();**」。
2. 如果類別內沒有任何建構式，則會加上1個含有「**super();**」的預設建構式。如果該類別沒有父類別，則「**super();**」代表**呼叫Object類別的預設建構式**。

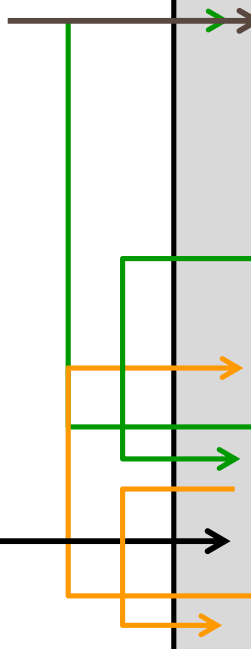
# Constructors overloading & this()

59

**MyDate d = new MyDate (27, 6, 2011);**

**MyDate d = new MyDate (27);**

```
public class MyDate {  
    private int year = 2000;  
    private int month = 1;  
    private int day = 1;  
  
    public MyDate(int d, int m, int y ) {  
        super();  
        year = y;  
        month = m;  
        day = d;  
    }  
    public MyDate(int d, int m) {  
        this(d, m, 2013);  
    }  
    public MyDate(int d) {  
        this(d, 5);  
    }  
}
```



# Object類別

60

- Object類別是所有Java類別的父類別，也是類別階層架構中的根類別

## Object類別常用方法

```
public String toString()
```

回傳一個可以代表這個物件的字串，可改寫這個方法以更精準地描述該物件。

### 範例 ObjectEx.java

```
1. class Book{
2.     String name;
3.     double price;
4.     String author;
5.     Book(String n, double p, String a){
6.         name = n;
7.         price = p;
8.         author = a;
9.     }
10.    public String toString(){
11.        return name;
12.    }
13. }
14.
15. class ObjectEx{
16.     public static void main(String[] args){
17.         Book book1 = new Book("Java 程式設計", 580.0, "張掘風");
18.         System.out.print(book1);
19.     }
20. }
```

-----輸出-----  
Java程式設計

# Content

61

- 繼承(Inheritance)
- 多型(Polymorphism)
  - ▣ 多型的特性
  - ▣ 型別轉型

# polymorphism

62

- 同名異式，簡稱為多型。
  - ▣ 指的是一個方法可以有許多型式，也就是相同的方法名稱，定義以不同的實作(implementation)。
  - ▣ 目的是希望簡化系統發展的複雜性並增加其彈性。
- 多型在程式執行時，呼叫方法是以動態連結(Dynamic Binding)方式，判斷當時被呼叫物件所屬的類別來決定執行那一實作，所以又稱為動態多型。
- 動態多型是建立在繼承的架構上。

# Example : polymorphism

63

```
1.  abstract class Shape {
2.      public abstract void f();
3.  }
4.  class Triangle extends Shape {
5.      public void f() {
6.          System.out.println("Triangle!");
7.      }
8.  }

9.  class Rectangle extends Shape {
10.     public void f() {
11.         System.out.println("Rectangle!");
12.     }
13. }

14. class Circle extends Shape {
15.     public void f() {
16.         System.out.println("Circle!");
17.     }
18. }
```

```
19. public class E {
20.     public static void main (String[] args) {
21.         Shape[] s = new Shape[] { new Triangle(),
22.                                     new Rectangle(), new Circle() };
23.         for (int i=0; i<s.length; i++) {
24.             s[i].f();
25.         }
26.     }
```

執行結果：  
Triangle!  
Rectangle!  
Circle!

# polymorphism

64

- 當呼叫方法時，依參數的數目與類型來決定執行那個實作，此稱為多重定義、過荷或超載(Overloading)。
- 超載又稱為靜態多型。
- 因此，在設計方法過荷時需要特別小心，千萬不能有任何方法的參數個數及型態是完全一致的。

# polymorphism

65

□ 例如：

- `int add(int i, int j) { return i + j; }`
- `float add(float i, float j) { return i + j; }`
- `double add(double i, double j) { return i + j; }`
- `int add(int i, float f) { return i + (int) f; }`
- `int add(int i, int j, int k) { return i + j + k; }`
  
- `add(1, 2)`
- `add(1, 2, 3)`



# 多型 Polymorphism

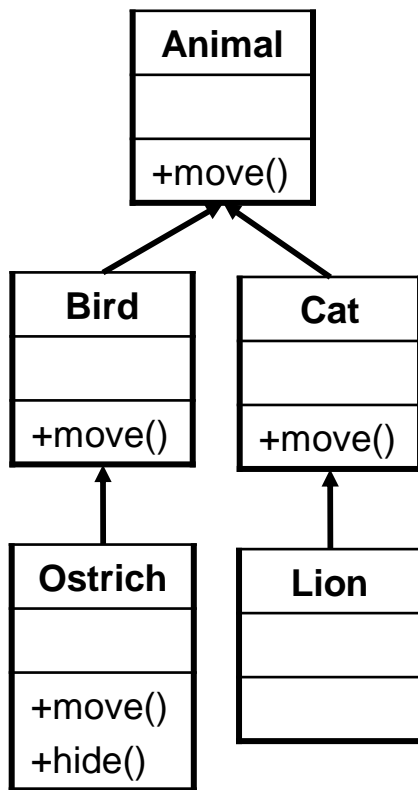
66

- 多型的意義
  - ▣ 一個物件可以用多種形態來看待
  - ▣ 型態間需有繼承關係：子類別可以被看待為父類別
- Java技術實作多型
  - ▣ 具有繼承關係的架構下,物件實體可以被視為多種型別。
  - ▣ 將子類別物件參考指定給父類別變數  
父類別 變數名稱 = new 子類別建構子();

# 物件多型

67

- 獅子是貓科動物，所有貓科動物皆是動物。
- 鴝鳥是鳥類，所有鳥類皆是一種動物



```
class Animal {
    void move() {...}
}
class Bird extends Animal {
    void move() {...}
}
class Cat extends Animal {
    void move() {...}
}
class Ostrich extends Bird {
    void move() {...}
    void hide() {...}
}
class Lion extends Cat {
}
```

Lion l = new Lion();  
用 **Lion** 獅子的眼光來看 **Lion**

Cat c = new Lion();  
用 **Cat** 貓科動物的眼光來看 **Lion**

Animal a = new Lion();  
用 **Animal** 動物的眼光來看 **Lion**

~~Lion l1 = new Cat();~~  
用 **Lion** 獅子的眼光來看所有的 **Cat** 貓科動物

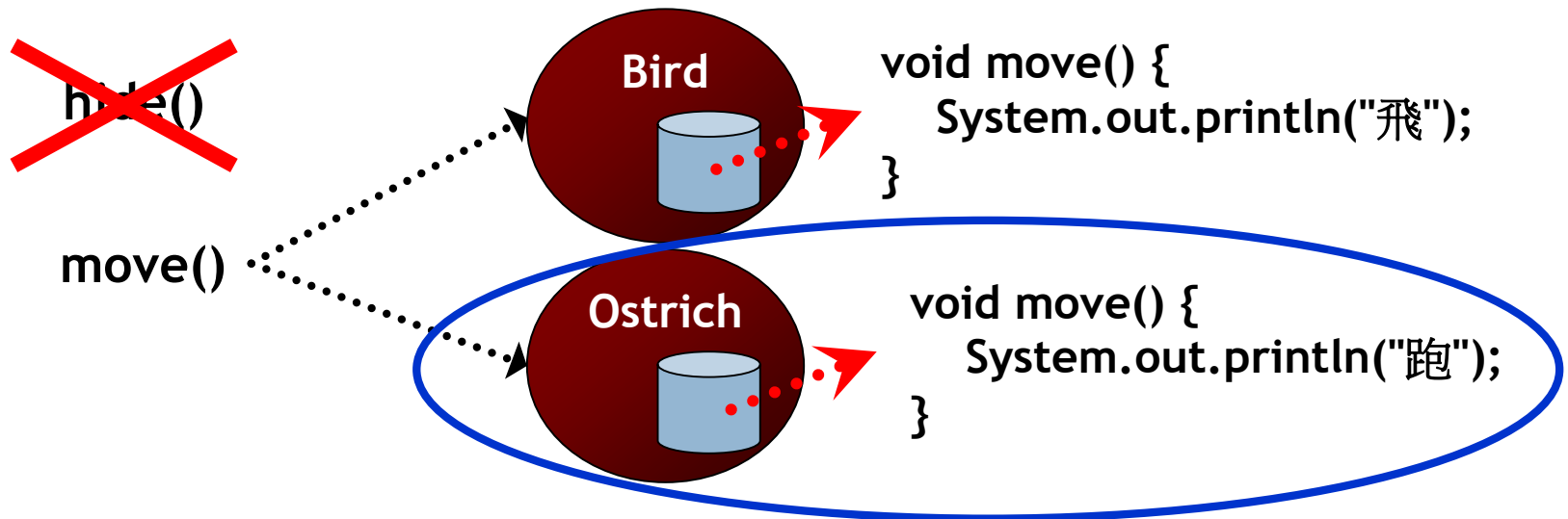
編譯錯誤

# 多型的特性

68

- 多型的特性
  - ▣ 不同型態表示並不會改變原來的實體
  - ▣ 將物件視為父類別,只能用父類別有定義之屬性及方法
  - ▣ 若父類別方法被子類別覆寫,多型時,用父類別的觀點呼叫,仍會執行子類別的方法

```
Bird bird = new Ostrich();
```



# 範例 - 多型的特性

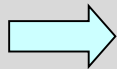
69

```
class Animal {  
    void move() {  
        System.out.println("動");  
    }  
}
```

```
class Bird extends Animal {  
    void move() {  
        System.out.println("飛");  
    }  
}
```

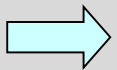
```
class Ostrich extends Bird {  
    void move() {  
        System.out.println("跑");  
    }  
    void hide() {  
        System.out.println("頭埋在土裡");  
    }  
}
```

```
Ostrich ostrich = new Ostrich();  
ostrich.move();  
ostrich.hide();
```

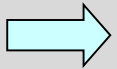
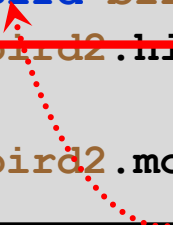


跑  
頭埋在土裡

```
Bird bird1 = new Bird();  
bird1.move();  
  
Bird bird2 = new Ostrich();  
bird2.hide();  
  
bird2.move();
```



飛



跑

在 **Bird** 型別中並不知道有 **hide()** 方法

# 型別檢查與虛擬方法調用

70

## □ Java型別檢查

- ▣ 編譯時期：compiler會以**宣告的型別**作型別檢查
  - 確保物件被視為父類別, 只能用父類別定義之屬性及方法
- ▣ 執行時期：JVM會以**實際的型別**作型別檢查
  - 確保多型時, 用父類別的觀點呼叫, 仍會執行子類別的方法

## □ 虛擬方法使用(呼叫) Virtual Method Invocation

- ▣ Java程式會使用(呼叫)變數在執行時期所參考之物件的行為, 而不是在編譯時期宣告類別的行為

# Example

71

```
public class Employee {  
    private String name = "Sean";  
    private double salary = 10000;  
    public void getDetails() {  
        System.out.println("Name:" + name);  
        System.out.println("Salary:" + salary);  
    }  
}
```

```
public class Manager extends Employee {  
    private String dept = "EDU";  
    public void getDetails() {  
        super.getDetails();  
        System.out.println("Department:" + dept);  
    }  
    public void getDepartment() {  
        System.out.println("Department:" + dept);  
    }  
}
```

```
public class Test {  
    public static void main(String [] args) {  
        Employee e = new Employee();  
        e.getDetails();  
        Manager m = new Manager();  
        m.getDetails();  
  
        Employee p = new Manager();  
  
        p.getDetails();  
        p.getDepartment();  
    }  
}
```



Employee type  
Manager instance

# Example

72

```
public class Employee {  
    private String name = "Sean";  
    private double salary = 10000;  
    public void getDetails() {  
        System.out.println("Name:" + name);  
        System.out.println("Salary:" + salary);  
    }  
}
```

```
public class Manager extends Employee {  
    private String dept = "EDU";  
    public void getDetails() {  
        super.getDetails();  
        System.out.println("Department:" + dept);  
    }  
    public void getDepartment() {  
        System.out.println("Department:" + dept);  
    }  
}
```

```
public class Test {  
    public static void main(String [] args) {  
        Employee e = new Employee();  
        e.getDetails();  
        Manager m = new Manager();  
        m.getDetails();  
  
        Employee p = new Manager();  
  
        { p.getDetails();  
        p.getDepartment();  
        }  
    }  
}
```

Compile-Time Type  
Employee

# Example

73

```
public class Employee {  
    private String name = "Sean";  
    private double salary = 10000;  
    public void getDetails() {  
        System.out.println("Name:" + name);  
        System.out.println("Salary:" + salary);  
    }  
}
```

```
public class Manager extends Employee {  
    private String dept = "EDU";  
    public void getDetails() {  
        super.getDetails();  
        System.out.println("Department:" + dept);  
    }  
    public void getDepartment() {  
        System.out.println("Department:" + dept);  
    }  
}
```

```
public class Test {  
    public static void main(String [] args) {  
        Employee e = new Employee();  
        e.getDetails();  
        Manager m = new Manager();  
        m.getDetails();  
  
        Employee p = new Manager();  
  
        p.getDetails();  
        // p.getDepartment();  
    }  
}
```

Run-Time Type  
Manager



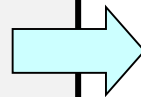


# 強制轉型

74

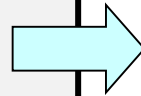
- 強制轉型
  - 將被宣告為父類別的子類別物件轉型回子類別
  - (目標類別名稱)物件名稱;
  - 可先用 <物件名稱> instanceof <類別名稱> 檢查
- 藉由轉型來解決呼叫hide()方法的問題。

```
Bird bird1 = new Bird();  
bird1.move();
```



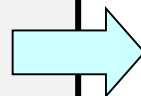
飛

```
Bird bird2 = new Ostrich();  
bird2.move();
```



跑

```
((Ostrich) bird2).hide();
```



頭埋在土裡

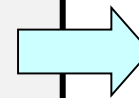
# instanceof 運算子

75

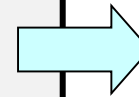
- instanceof 運算子
  - ▣ 確認物件是否為某種類別型態
  - ▣ <物件名稱> instanceof <類別名稱>
  - ▣ 回傳布林值
    - **true**：該變數所參考的物件可以轉換成特定類別。
    - **false**：反之則否。

```
Bird bird1 = new Bird();  
bird1.move();
```

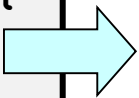
```
Bird bird2 = new Ostrich();  
bird2.move();  
if(bird2 instanceof Ostrich) {  
    ((Ostrich) bird2).hide();  
}
```



飛



跑



頭埋在土裡

# Abstract

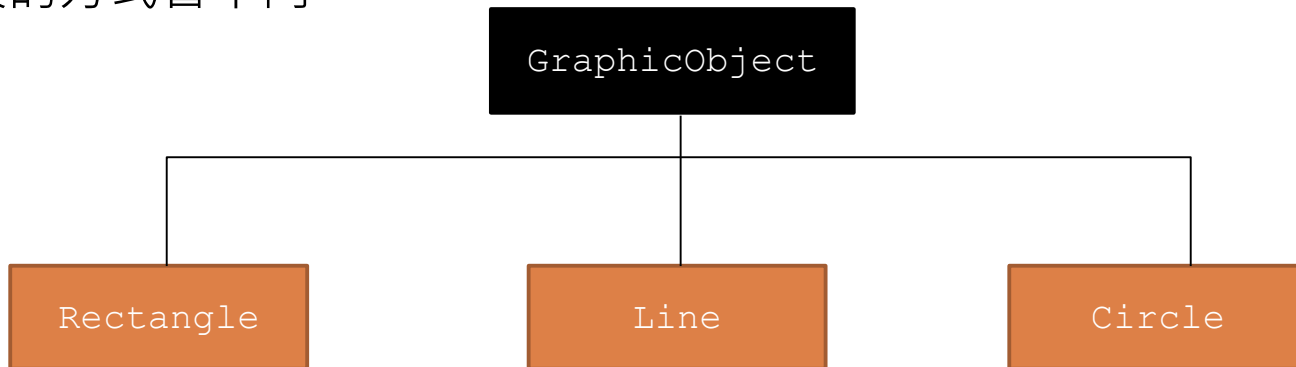
76

- 真實世界的抽象觀念與實體
  - ▣ 例如：食物 vs. 蘋果
    - 食物代表一種抽象的觀念
    - 蘋果代表一種實體
- 物件導向世界中的抽象觀念
  - ▣ 例如：我們會想模擬一種抽象觀念而不建立其實體，如`java.lang.Number`類別代表一種數字的抽象類別，但我們所用的則是`Integer`、`Float`等有實際意義的類別
  - ▣ 何謂抽象類別？一種代表抽象觀念而不能被實體化的類別。抽象類別只能被繼承。

# Abstract

77

- 假設在一繪圖程式中，你可以繪製圓形、矩形、線條等等。
- 這些圖形物件都有一些共同的狀態(位置、外框)與行為(移動、改變大小、繪製)。因此可以利用這些共同特性，而將其全部宣告成繼承自同一父類別 **GraphicObject**。
- 不過，圖形物件也有許多不同點：畫圓形與畫矩形是十分不同的。而圖形物件無法共享這些狀態與行為。
- 另一方面，所有的 **GraphicObject** 也必須知道如何繪製其本身，唯一差別在繪製的方式會不同。



# 抽象方法(abstract method)

78

## □ 抽象方法

- ▣ 只宣告方法定義,而沒有撰寫方法的內容
- ▣ 語法:

```
abstract <return_type> <name> ([arguments]);
```

## □ 用途

- ▣ 抽象方法強迫有繼承關係的子類別去覆寫此方法
- ▣ 同樣要有某個方法,但實作方法卻大不相同
- ▣ 使多型可以使用

# 抽象類別(abstract class)

79

## □ 抽象類別

- 類別**只要有一個**以上的抽象方法,就必須為抽象類別

## □ 語法:

```
abstract class <Name> {  
    ....  
}
```

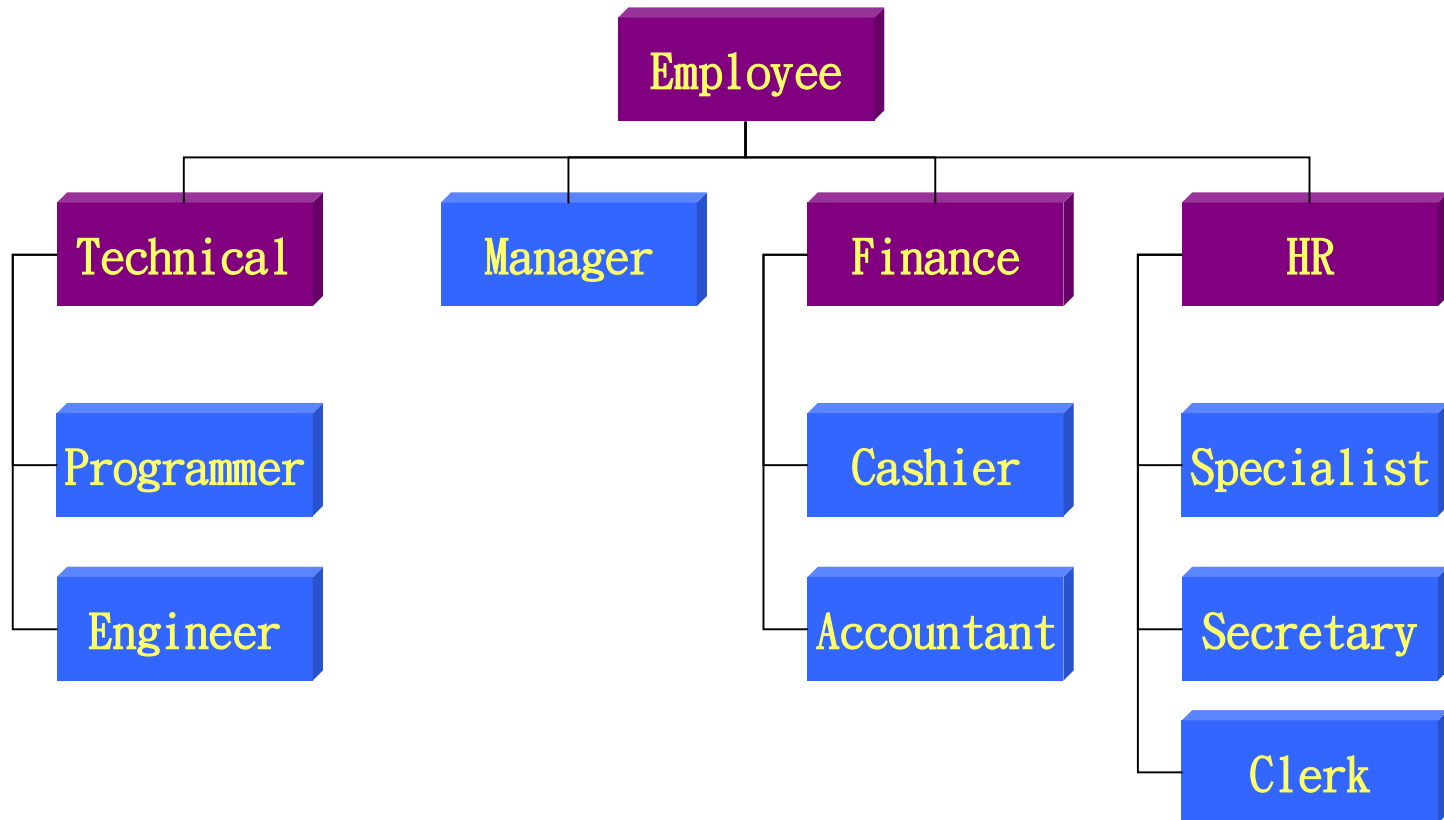
- 抽象類別不能建構物件實體
- 抽象類別仍可以有建構子, 屬性與具體方法
  - 建構子用來初始化實體成員,只能由子類別用`super()`呼叫
- **繼承自抽象類別的子類別一定要實作抽象方法, 除非繼續宣告為抽象類別**
- 注意 :
  - `abstract`與`final`修飾詞不可共存。
  - `abstract`與`private`修飾詞不可共存。

# 抽象類別(abstract class)

80

## □ 用途

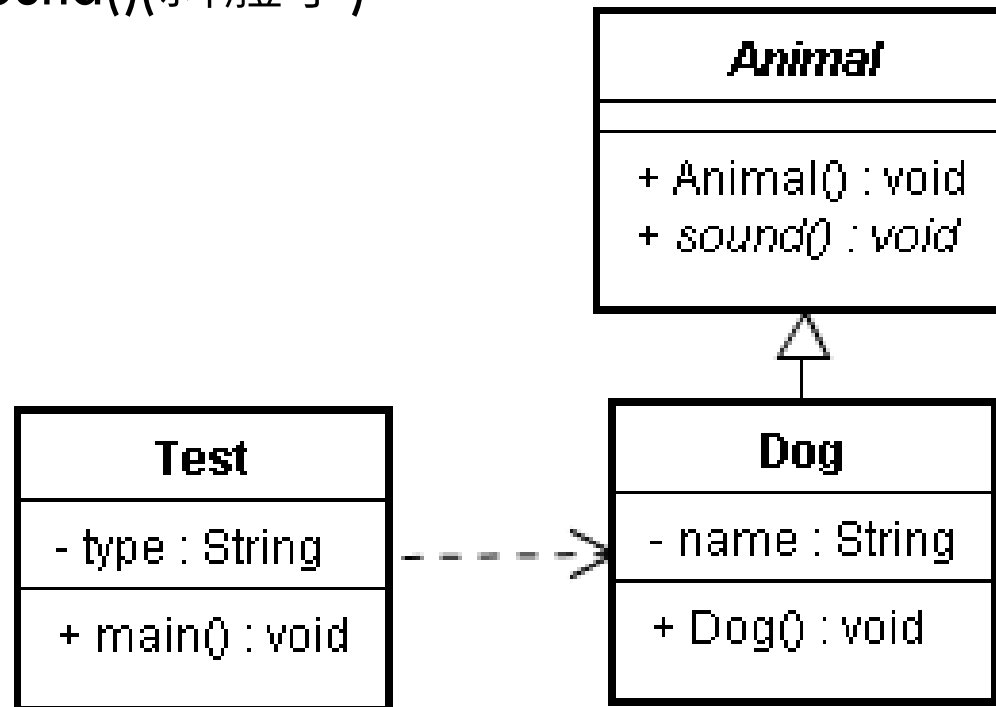
- 某些類別只是定義一些抽象的分類概念
- 員工是抽象概念沒有實體
- 員工的實體必須負責特定工作,Ex: 程式設計,經理,秘書等



# Abstract Class & UML

81

- 使用斜體的就是表示抽象，抽象類別為*Animal*(斜體字)，抽象方法為*sound()*(斜體字)。

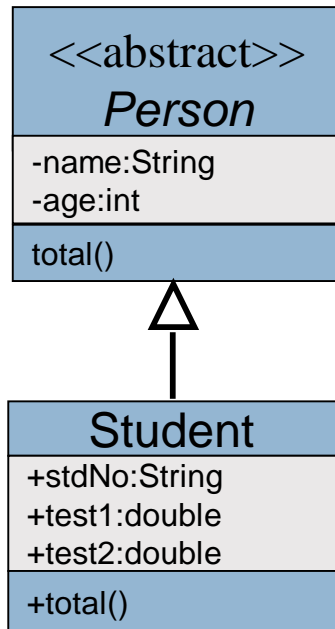




# 抽象類別的定義與使用

82

## □ UML類別圖與Example

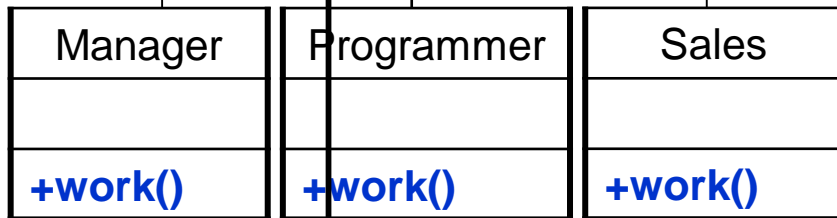
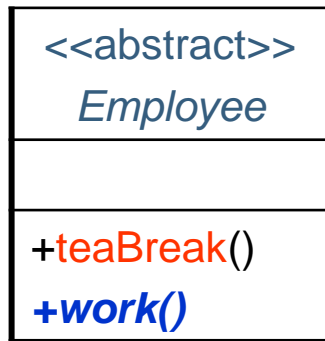


```
abstract class Person
{
    public String name;
    public int age;
    public abstract void total();
}
```

```
class Student extends Person
{
    public String stdNo;
    public double test1, test2;
    public Student(String no, String name, int age,
                    double t1, double t2)
    { stdNo = no; this.name = name; this.age=age;
      test1 = t1; test2 = t2; }
    public void total()
    { System.out.println("總分:" + (test1+test2)); }
}
```

# 抽象類別(abstract class)

83



```
01 abstract class Employee {
02     public void teaBreak() {
03         System.out.println("喝咖啡");
04     }
05     abstract void work();
06 }
```

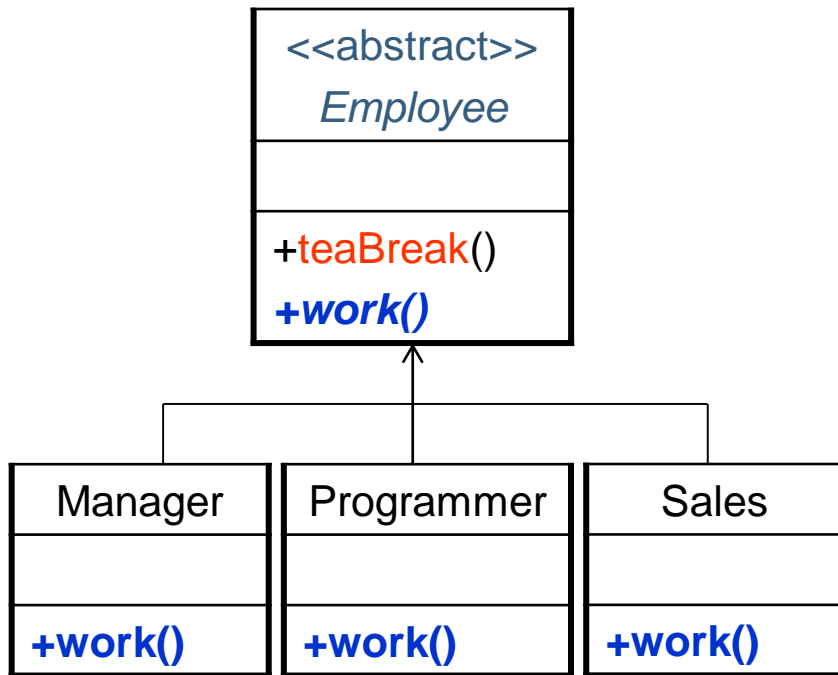
```
01 class Sales extends Employee {
02     public void work() {
03         System.out.println("推銷");
04     }
05 }
```

```
01 class Manager extends Employee {
02     public void work() {
03         System.out.println("開會");
04     }
05 }
```

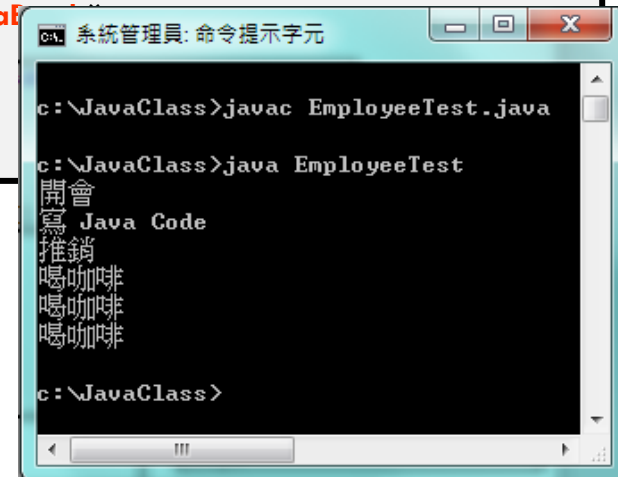
```
01 class Programmer extends Employee {
02     public void work() {
03         System.out.println("寫 Java Code");
04     }
05 }
```

# 抽象類別多型

84



```
01 public class EmployeeTest{
02     public static void main(String[] args) {
03
04         Employee manager = new Manager();
05         Employee programmer = new Programmer();
06         Employee sales= new Sales();
07
08         manager.work();
09         programmer.work();
10         sales.work();
11
12         manager.teaBreak();
13         programmer.teaBreak();
14         sales.teaBreak();
15
16     }
17 }
```



# 抽象類別的物件變數

85

- 抽象類別不能建立物件, 但是可以當做資料型態, 做為由子類別實例所產生之物件的參考, 例如:

```
Person s2 = new Student("s002", "陳曉明", 30, 67, 23);
```

- 子類別物件也屬於父類別物件, 所以Student物件也是一種Person物件。因此, Person物件變數也可以參考Student物件變數, 呼叫Student物件變數所實作的total()方法。例如:

```
s2.total();
```

- 物件變數s2只能存取Student物件中原本在抽象類別中就有的屬性和方法, **若要存取子類別新增的屬性和方法, 必須強制轉型為Student物件才行:**

```
Student s;  
s = (Student) s2;
```

s2 →

```
Person抽象類別的物件  
String name;  
int age;  
abstract void total();
```

s →

```
Student 物件  
String stdNo = "s002";  
String name = "陳曉明";  
int age = 30;  
double test1 = 67;  
double test2 = 23;  
void total() {...}
```

# Example：抽象類別的物件變數

86

```
1. abstract class Person      // Person類別宣告
2. { // 成員資料
3.     public String name;      // 姓名
4.     public int age;          // 年齡
5.     // 抽象方法: 計算總分或總價
6.     public abstract void total();
7. }
8. // Student類別宣告
9. class Student extends Person
10. {
11.     // 成員資料
12.     public String stdNo;
13.     public double test1, test2;
14.     // 建構式
15.     public Student(String no, String name,
16.         int age, double t1, double t2)
17.     { stdNo = no;
18.       this.name = name;
19.       this.age = age;
20.       test1 = t1;
21.       test2 = t2;
22.     }
23.     // 成員方法: 實作抽象方法total()
24.     public void total()
25.     {
26.         System.out.println("總分:"+(test1+test2));
27.     }
28. }
```

```
29. // 主程式類別
30. public class AbstractTest
31. {
32.     // 主程式
33.     public static void main(String[] args)
34.     {
35.         Student s; // 類別的物件變數
36.         // 宣告Student類別型態的變數, 並且建立物件
37.         Student s1 = new Student("s001","陳會安",35,56,78);
38.         Person s2 = new Student("s002","江小魚",30,67,23);
39.         // 顯示學生s1的資料
40.         System.out.println("學生s1的資料 =====");
41.         System.out.println("編號: " + s1.stdNo);
42.         System.out.println("姓名: " + s1.name);
43.         System.out.println("年齡: " + s1.age);
44.         // 呼叫物件的副本方法
45.         s1.total();
46.         // 顯示學生s2的資料, 檢查是否為Student的副本
47.         if (s2 instanceof Student)
48.             System.out.println("->s2是Student類別副本");
49.         System.out.println("學生s2的資料 =====");
50.         s = (Student) s2; // 型別轉換
51.         System.out.println("編號: " + s.stdNo);
52.         System.out.println("姓名: " + s.name);
53.         System.out.println("年齡: " + s.age);
54.         // 呼叫物件的副本方法
55.         s2.total();
56.     }
57. }
```

# 子類別物件也是父類別物件

87

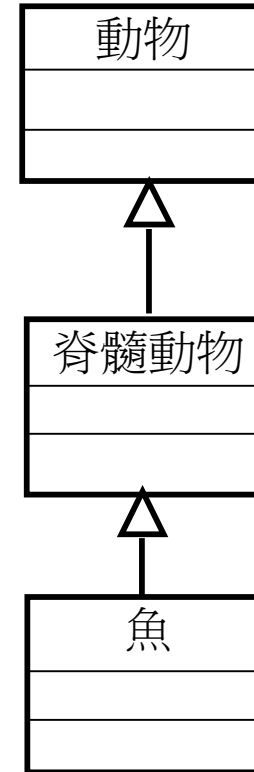
```
1. public class Test
2. {
3.     public static void main(String[] args)
4.     {
5.         魚 a = new 魚();
6.         脊髓動物 b = new 魚();
7.         動物 c = new 魚();
8.         // 魚 d = new 脊髓動物(); // 錯誤
9.         // 脊髓動物 e = new 動物(); // 錯誤

10.        System.out.println("a=" + a);
11.        System.out.println("b=" + b);
12.        System.out.println("c=" + c);
13.    }
14. }

15. class 動物 { }

16. class 脊髓動物 extends 動物 { }

17. class 魚 extends 脊髓動物 { }
```



```
a= 魚@14f8dab
b= 魚@1ddebc3
c= 魚@a18aa2
```

# 修飾子使用對象

88

	static	final	abstract
class		V	V
method	V	V	V
attribute	V	V	
local variable		V	

# Java Fundamentals

## Exception Handling

Alvin

fcaibi@gmail.com



# Content

90

- 例外處理機制
  - ▣ 錯誤回報
  - ▣ 例外分類
- 捕捉例外 `try-catch` 敘述句
- 例外傳遞

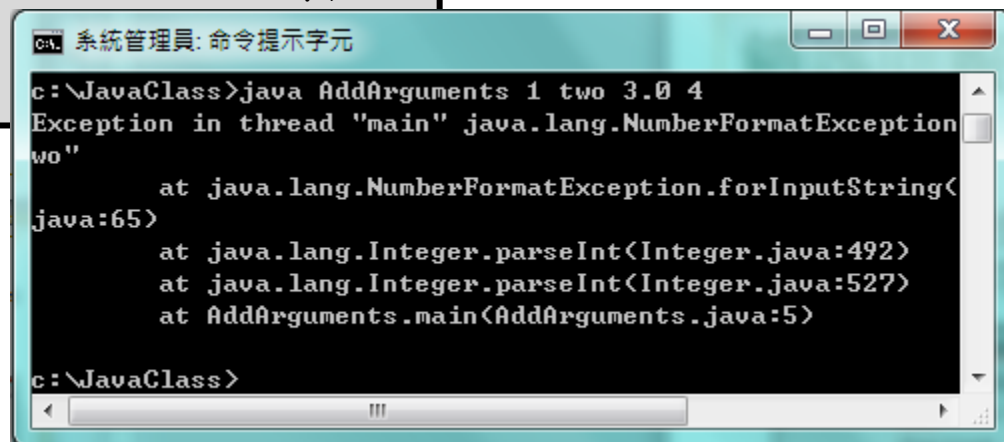
# 錯誤回報

91

## ❑ 錯誤回報

- ❑ 當程式發生無法執行的狀態，系統停止執行，並於命令提示字元顯示錯誤訊息

```
public class AddArguments{  
    public static void main(String[] args) {  
        int sum = 0;  
        for(int i=0; i<args.length; i++){  
            sum += Integer.parseInt(args[i]);  
        }  
        System.out.println("sum = " + sum);  
    }  
}
```



The screenshot shows a Windows Command Prompt window titled "系統管理員: 命令提示字元". The command executed is `c:\JavaClass>java AddArguments 1 two 3.0 4`. The output shows a `java.lang.NumberFormatException` exception in the main thread, with the message "two". The stack trace indicates the error occurred at `java.lang.NumberFormatException.forInputString` (line 65), `java.lang.Integer.parseInt` (line 492), `java.lang.Integer.parseInt` (line 527), and `AddArguments.main` (line 5).

```
系統管理員: 命令提示字元  
c:\JavaClass>java AddArguments 1 two 3.0 4  
Exception in thread "main" java.lang.NumberFormatException  
two"  
    at java.lang.NumberFormatException.forInputString(  
java:65)  
    at java.lang.Integer.parseInt(Integer.java:492)  
    at java.lang.Integer.parseInt(Integer.java:527)  
    at AddArguments.main(AddArguments.java:5)  
c:\JavaClass>
```

# 例外處理機制

92

- 例外處理機制
  - ▣ 定義程式執行時，發生非預期狀況時應如何處理
    - Ex: 網路連線失敗、欲開啟檔案不存在、傳入參數值錯誤
  - ▣ 確保系統在非預期情況發生時，仍能運行不會中斷
- 例外分類
  - ▣ 不須檢查的例外 (Unchecked Exception)
    - 無法處理的例外
    - 程式的臭蟲 Bug
  - ▣ 必須檢查的例外 (Checked Exception)
    - 開發人員預期可能發生，並應加以處理

# Exception

93

## □ Java 中有兩種不同型態的例外(Exceptions)

### ▣ 執行時期的例外(runtime exceptions)

#### ■ 發生在 Java 執行系統內部的例外，如：

- 算數例外(分母為 0)
- reference 例外(透過 null 來存取資料)
- index 例外(超出陣列範圍)

#### ■ 這一類的例外，不一定要處理

### ▣ 非執行時期的例外(nonruntime exceptions)

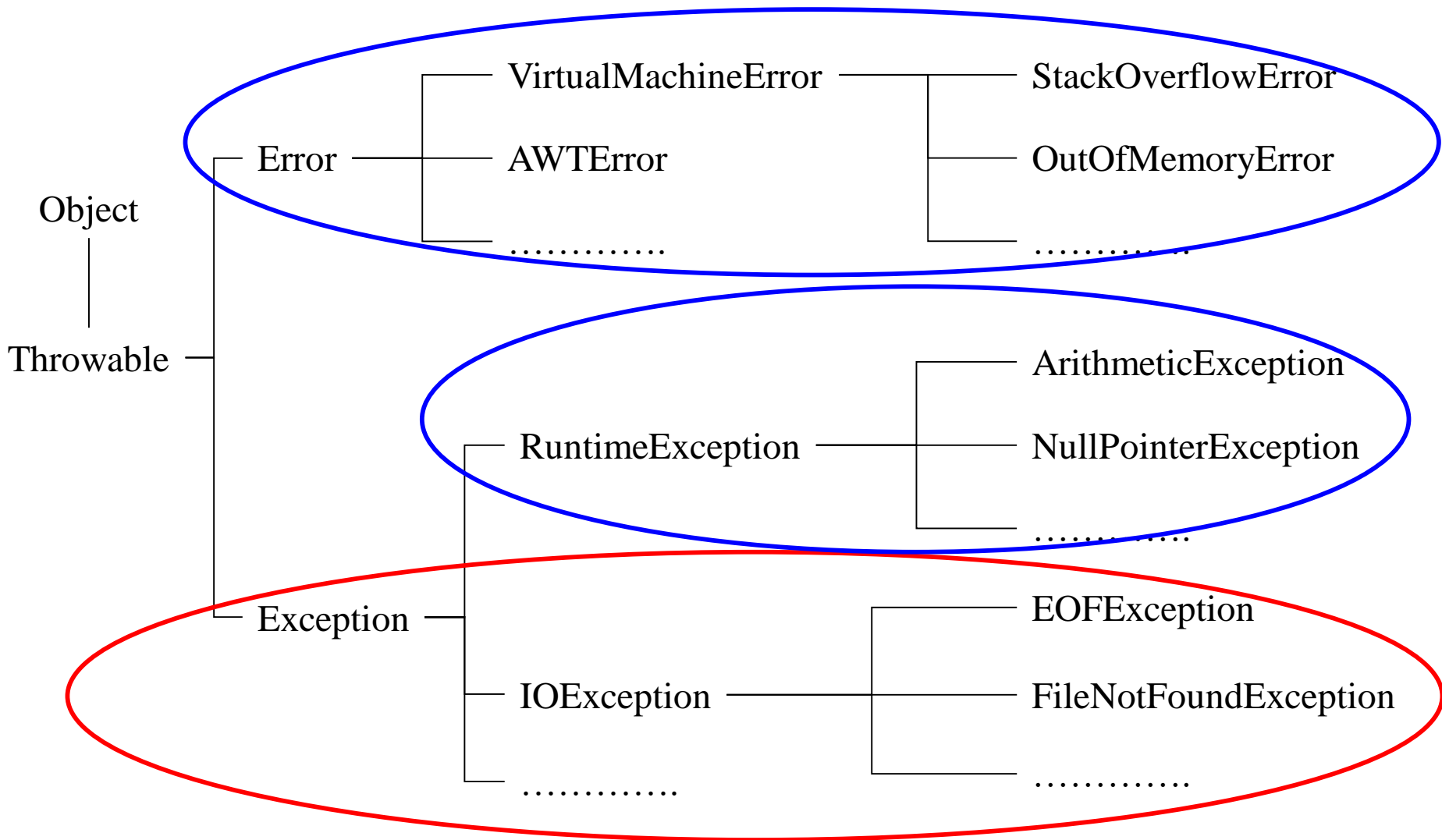
#### ■ 發生在 Java 執行系統之外的例外，如：

- I/O 例外(檔案不存在)，

#### ■ 這一類的例外，一定要處理，也稱為必須檢查的例外(checked exceptions)

# Exception Categories

94



# Exception Categories

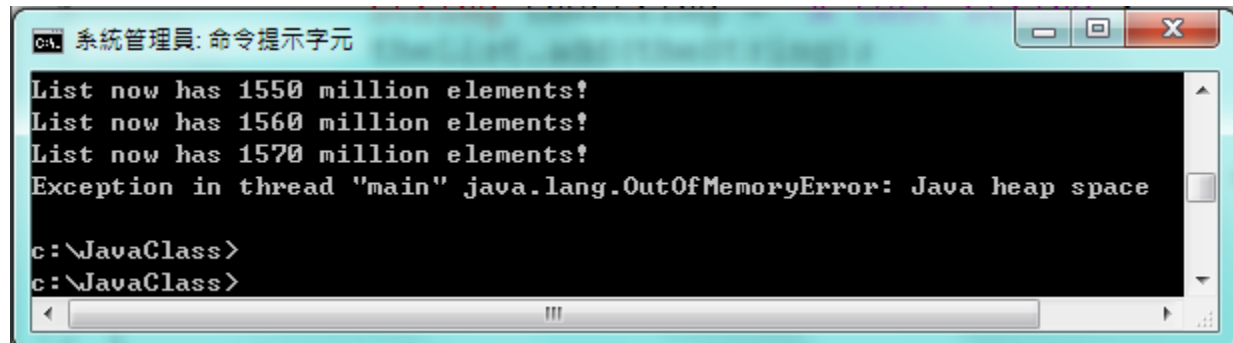
95

- 不須檢查的例外(Unchecked exception)
  - ▣ java.lang.Error
    - 系統難以回復的例外
      - 記憶體不足→OutOfMemoryError
      - 執行緒死結→ThreadDeath
  - ▣ java.lang.RuntimeException
    - 程式設計上疏忽造成,應該要修正程式
      - 除數為0→ArithmeticException
      - 忘了將物件reference指到物件實體→NullPointerException
      - 陣列索引值超出範圍→ArrayIndexOutOfBoundsException
      - 數值格式不符→NumberFormatException

# Error Example

96

```
public class ThrowError {  
    public static void main (String args[]) {  
        java.util.ArrayList theList = new java.util.ArrayList();  
        while(true) {  
            String theString = "A test String";  
            theList.add(theString);  
  
            if (theList.size()% 1000000 == 0) {  
                System.out.println("List now has " + theList.size()/100000 + " million elements!");  
            }  
        }  
    }  
}
```



```
系統管理員: 命令提示字元  
List now has 1550 million elements!  
List now has 1560 million elements!  
List now has 1570 million elements!  
Exception in thread "main" java.lang.OutOfMemoryError: Java heap space  
c:\JavaClass>  
c:\JavaClass>
```

# 常見的例外(Runtime Exception)

97

執行期例外	說明
ArithmeticException	數學運算時的例外。 例如：某數除以0。
ArrayIndexOutOfBoundsException	陣列索引值超出範圍。
NegativeArraySizeException	陣列的大小為負數。
NullPointerException	物件參照為null， 並使用物件成員時所產生的例外。
NumberFormatException	數值格式不符所產生的例外。



# Exception Categories

98

## □ Checked exception

- ▣ java.lang.Exception的子類別,但不是 RuntimeException的子類別
- ▣ 可預期的外部因素造成的例外
  - 檔案不存在錯誤→FileNotFoundException
  - 輸出入處理錯誤→IOException
  - 資料庫處理錯誤→SQLException
- ▣ 系統強程式中一定要處理,否則編譯失敗

# Java API常見 Checked Exception

99

## Constructor Detail

### File

```
public File(String pathname)
```

Creates a new `File` instance by converting the given pathname string into an abstract pathname. If the given string is the empty string, then the result is the empty abstract pathname.

#### Parameters:

pathname - A pathname string

#### Throws:

`NullPointerException` - If the pathname argument is null

## Method Detail

### createNewFile

```
public boolean createNewFile()  
    throws IOException
```

Atomically creates a new, empty file named by this abstract pathname if and only if a file with this name does not yet exist. The check for the existence of the file and the creation of the file if it does not exist are a single operation that is atomic with respect to all other filesystem activities that might affect the file.

Note: this method should *not* be used for file-locking, as the resulting protocol cannot be made to work reliably. The `FileLock` facility should be used instead.

#### Returns:

true if the named file does not exist and was successfully created; false if the named file already exists

#### Throws:

`IOException` - If an I/O error occurred

`SecurityException` - If a security manager exists and its `SecurityManager.checkWrite(java.lang.String)` method denies write access to the file

#### Since:

1.2

# Java API常見 Checked Exception

100

## Constructor Detail

### FileReader

```
public FileReader(String fileName)
    throws FileNotFoundException
```

Creates a new `FileReader`, given the name of the file to read from.

#### Parameters:

`fileName` - the name of the file to read from

#### Throws:

`FileNotFoundException` - if the named file does not exist, is a directory rather than a regular file, or for some other reason cannot be opened for reading.

## Method Detail

### read

```
public int read()
    throws IOException
```

Reads a single character. This method will block until a character is available, an I/O error occurs, or the end of the stream is reached.

Subclasses that intend to support efficient single-character input should override this method.

#### Returns:

The character read, as an integer in the range 0 to 65535 (`0x00-0xffff`), or -1 if the end of the stream has been reached


#### Throws:

`IOException` - If an I/O error occurs

# Checked Exception Example

101

```
public class CheckedException1 {  
    public static void main(String[] args) {  
        java.io.File testFile = new java.io.File("test.txt");  
        testFile.createNewFile();  
  
        System.out.println("File exists: " + testFile.exists());  
        testFile.delete();  
        System.out.println("File exists: " + testFile.exists());  
    }  
}
```



```

c:\JavaClass>javac CheckedException1.java
CheckedException1.java:4: error: unreported exception IOException; must be caught
or declared to be thrown
    testFile.createNewFile();
                        ^
1 error

c:\JavaClass>
```

# Content

102

- 例外處理機制
- 捕捉例外 **try-catch** 敘述句
- 例外傳遞

# 捕捉例外 try-catch 敘述

103

```
try {  
    // 保護區塊  
} catch (Specialize_Exc e) {  
    // 錯誤處理  
} catch (Normalize_Exc e) {  
    // 錯誤處理  
} finally {  
    // 一定要執行的動作  
}
```

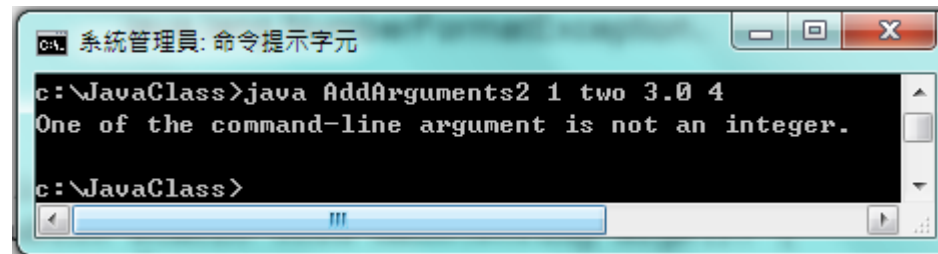
可以有一個以上的 catch blocks  
但是要注意 catch 的順序  
例外也符合自動轉型  
特定性的例外在前  
一般性的例外在後

不論是否有例外產生都  
一定會執行的區塊  
除非在保護區塊中遇到  
System.exit() 方法

# try-catch 敘述範例

104

```
public class AddArguments2{  
    public static void main(String[] args) {  
        try {  
            int sum = 0;  
            for(int i=0; i<args.length; i++){  
                sum += Integer.parseInt(args[i]);  
            }  
            System.out.println("sum = " + sum);  
        } catch (NumberFormatException nfe) {  
            System.err.println("One of the command-line "  
                               + "argument is not an integer.");  
        }  
    }  
}
```

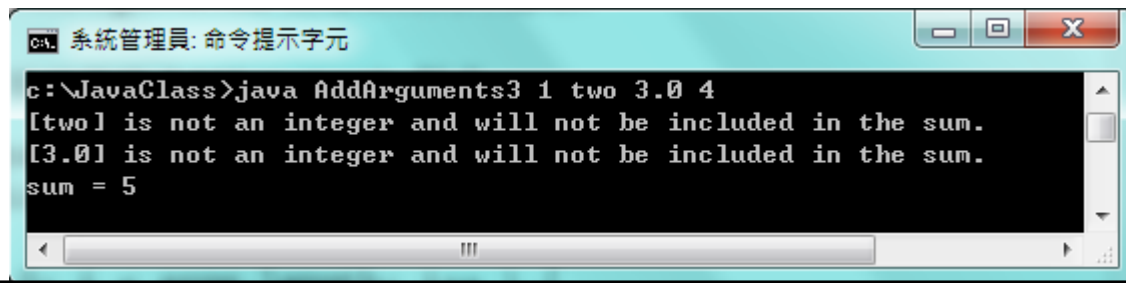


The screenshot shows a Windows Command Prompt window titled "系統管理員: 命令提示字元". The command prompt shows the following sequence of events:  
1. The user enters the command: `c:\JavaClass>java AddArguments2 1 two 3.0 4`  
2. The program outputs the error message: `One of the command-line argument is not an integer.`  
3. The prompt returns to `c:\JavaClass>`

# try-catch 敘述範例

105

```
public class AddArguments3{  
    public static void main(String[] args) {  
        int sum = 0;  
        for(int i=0; i<args.length; i++){  
            try {  
                sum += Integer.parseInt(args[i]);  
            } catch (NumberFormatException nfe) {  
                System.err.println "["+args[i]+" is not an integer"  
                    + " and will not be included in the sum.");  
            }  
        }  
        System.out.println("sum = " + sum);  
    }  
}
```



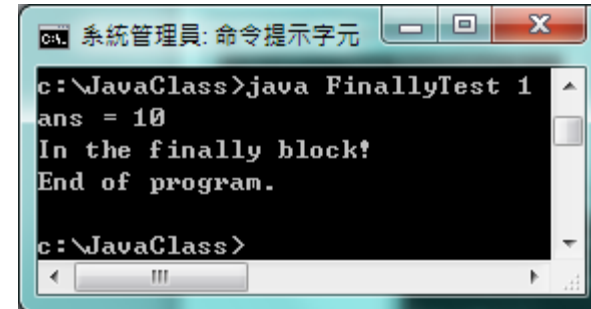
```
系統管理員: 命令提示字元  
c:\JavaClass>java AddArguments3 1 two 3.0 4  
[two] is not an integer and will not be included in the sum.  
[3.0] is not an integer and will not be included in the sum.  
sum = 5
```



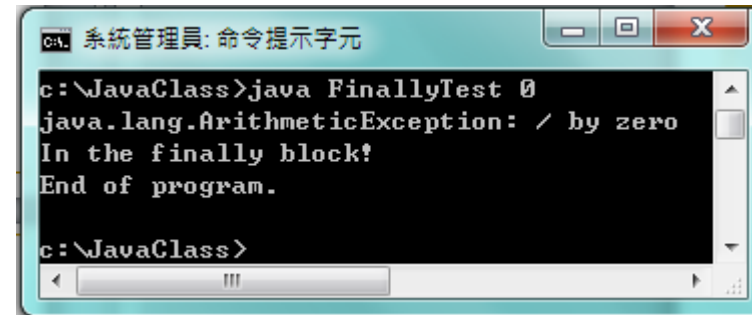
# finally 區段

106

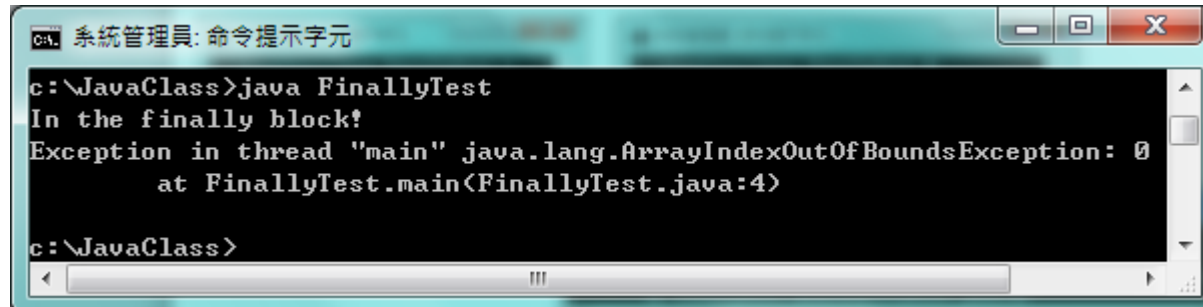
```
public class FinallyTest {  
    public static void main(String argv[]) {  
        try {  
            int i = Integer.parseInt(argv[0]);  
            int ans = 10 / i;  
            System.out.println("ans = " + ans);  
        } catch (ArithmeticException e) {  
            System.err.println(e);  
        } finally {  
            System.out.println("In the finally block!");  
        }  
  
        System.out.println("End of program.");  
    }  
}
```



```
C:\JavaClass>java FinallyTest 1  
ans = 10  
In the finally block!  
End of program.  
C:\JavaClass>
```



```
C:\JavaClass>java FinallyTest 0  
java.lang.ArithmeticException: / by zero  
In the finally block!  
End of program.  
C:\JavaClass>
```

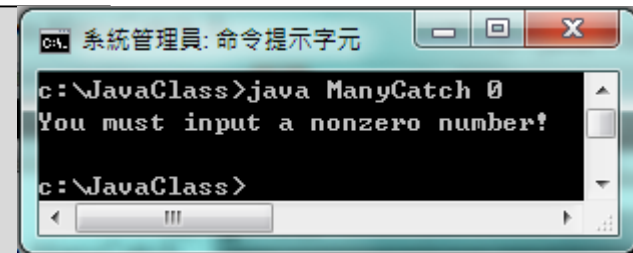


```
C:\JavaClass>java FinallyTest  
In the finally block!  
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 0  
    at FinallyTest.main(FinallyTest.java:4)  
C:\JavaClass>
```

# 多重 catch 區段

107

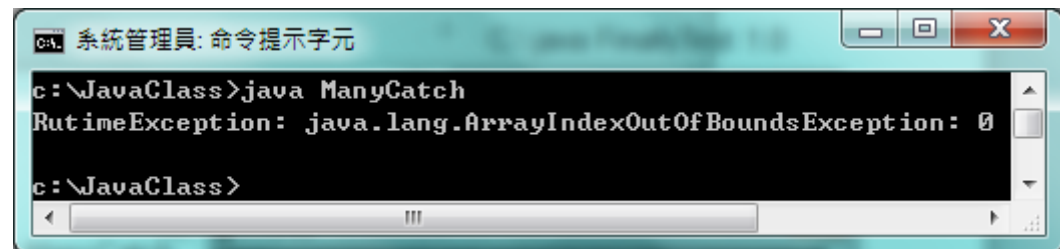
```
public class ManyCatch {  
    public static void main(String argv[]) {  
        try {  
            int i = Integer.parseInt(argv[0]);  
            int ans = 10 / i;  
        } catch (ArithmeticException ae) {  
            System.err.println("You must input a nonzero number!");  
        } catch (NumberFormatException ne) {  
            System.err.println("You must input a integer number!");  
        } catch (RuntimeException re) {  
            System.err.println("RutimeException: "+re);  
        } catch (Exception e) {  
            System.err.println("Exception: "+e);  
        }  
    }  
}
```



```
系統管理員: 命令提示字元  
c:\JavaClass>java ManyCatch 0  
You must input a nonzero number!  
c:\JavaClass>
```



```
系統管理員: 命令提示字元  
c:\JavaClass>java ManyCatch 1.0  
You must input a integer number!  
c:\JavaClass>
```



```
系統管理員: 命令提示字元  
c:\JavaClass>java ManyCatch  
RutimeException: java.lang.ArrayIndexOutOfBoundsException: 0  
c:\JavaClass>
```

# Multiple catch

108

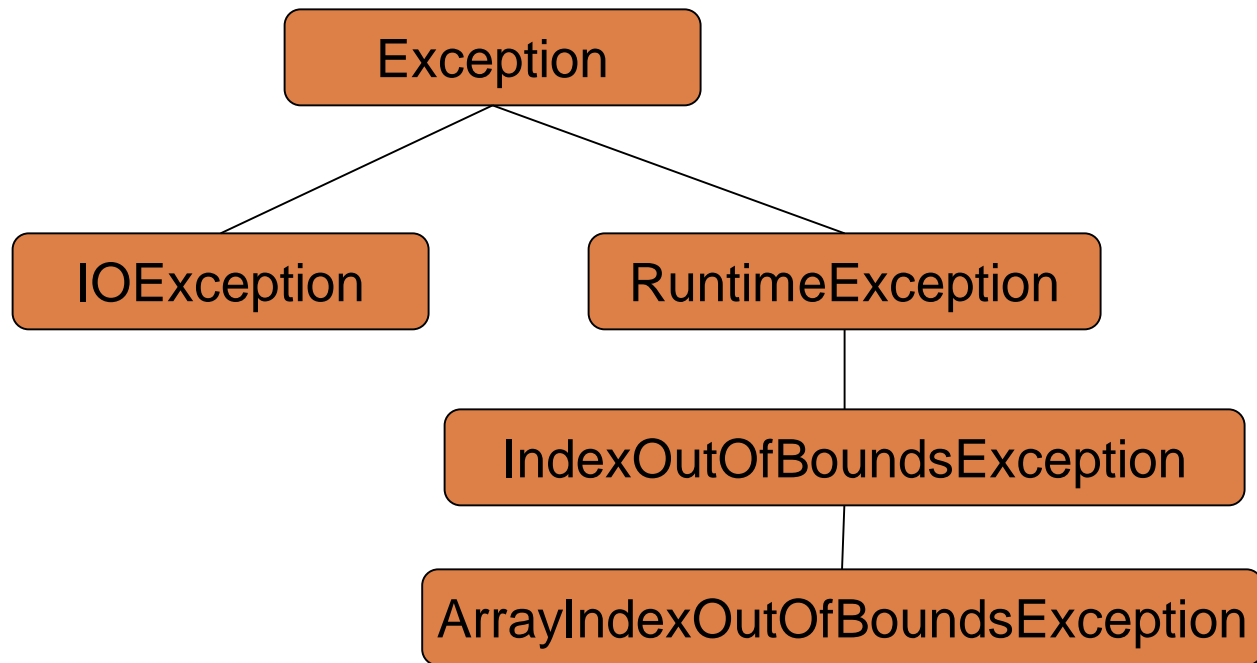
- **catch** 敘述可以有幾個。這些 **catch** 敘述是否可以隨意亂排，沒有一定順序？

```
1. try {  
2.     ....  
3. } catch (ArrayIndexOutOfBoundsException e) {  
4.     System.err.println("Caught ArrayIndexOutOfBoundsException: " + e.getMessage());  
5. } catch (IOException e) {  
6.     System.err.println("Caught IOException: " + e.getMessage());  
7. }
```

# Multiple catch

109

## □ Exception 繼承樹：



# Multiple catch

110

```
1. Try {  
2. } catch (Exception e) {  
3. } catch (IOException e) {  
4. } catch (RuntimeException e) {  
5. } catch (ArrayIndexOutOfBoundsException e) {  
6. } finally {  
7. }
```

- 如果程式這麼寫，會發生所有的例外，包括 `IOException`、`ArrayIndexOutOfBoundsException`，都會被第一個 `catch` 收走。因為所有 `Exception` 都是 `Exception` 的子類別。

# Multiple catch

111

## □ 正確寫法

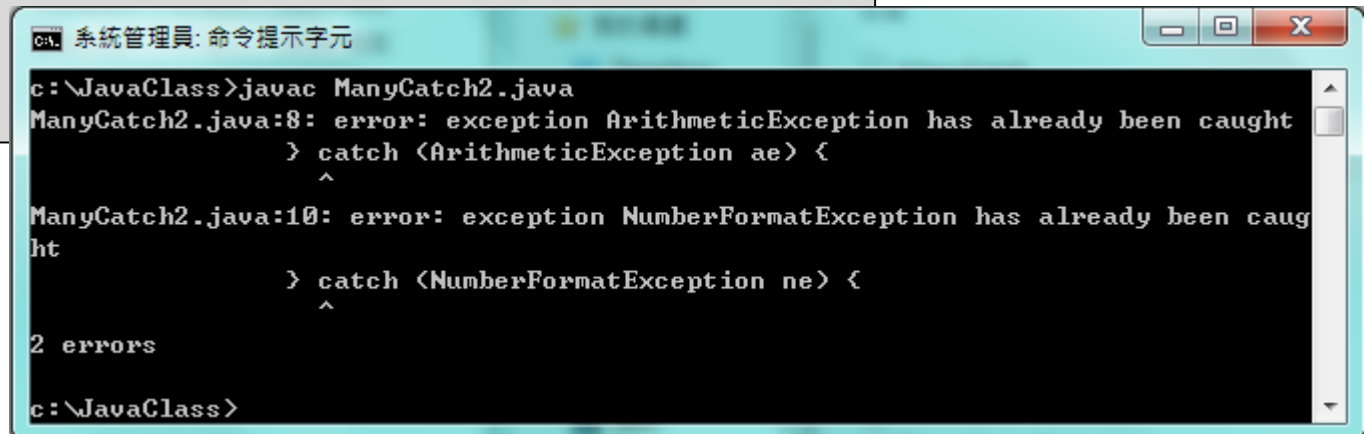
```
1. try {  
2. } catch (ArrayIndexOutOfBoundsException e) {  
3. } catch (RuntimeException e) {  
4. } catch (IOException e) {  
5. } catch (Exception e) {  
6. } finally {  
7. }
```

- 要由繼承樹底端物件往上寫，才能讓各例外"掉入"正確的錯誤處理程式中。正如"水果篩選器"，也是把洞小的滾筒擺前方，洞大的滾筒擺後方，如此才能分出水果的大小。

# catch 區段順序

112

```
public class ManyCatch2 {  
    public static void main(String argv[]) {  
        try {  
            int i = Integer.parseInt(argv[0]);  
            int ans = 10 / i;  
        } catch (RuntimeException re) {  
            System.err.println("Runtime Exception: "+re);  
        } catch (ArithmeticException ae) {  
            System.err.println("You must input a nonzero number!");  
        } catch (NumberFormatException ne) {  
            System.err.println("You must input a integer number!");  
        } catch (Exception e) {  
            System.err.println("Exception: "+e);  
        }  
    }  
}
```



The screenshot shows a Windows command prompt window titled "系統管理員: 命令提示字元". The command prompt shows the following text:

```
c:\JavaClass>javac ManyCatch2.java  
ManyCatch2.java:8: error: exception ArithmeticException has already been caught  
        } catch (ArithmeticException ae) {  
            ^  
ManyCatch2.java:10: error: exception NumberFormatException has already been caught  
        } catch (NumberFormatException ne) {  
            ^  
2 errors  
c:\JavaClass>
```

# 處理例外相關方法

113

- `java.lang.Throwable`
  - ▣ `public String toString()`
  - ▣ `public String getMessage()`
  - ▣ `public String getLocalizedMessage()`
  - ▣ `public void printStackTrace()`



# 處理例外相關方法

114

```
public class ThrowableTest {  
    public static void main(String argv[]) {  
        try {  
            java.io.FileReader f = new java.io.FileReader("test.txt");  
        } catch (FileNotFoundException e) {  
            System.out.println("=== toString() ===");  
            System.err.println(e);  
  
            System.out.println("=== getLocalizedMessage() ===");  
            System.err.println(e.getLocalizedMessage());  
  
            System.out.println("=== getMessage() ===");  
            System.err.println(e.getMessage());  
  
            System.out.println("=== printStackTrace() ===");  
            e.printStackTrace();  
        }  
    }  
}
```



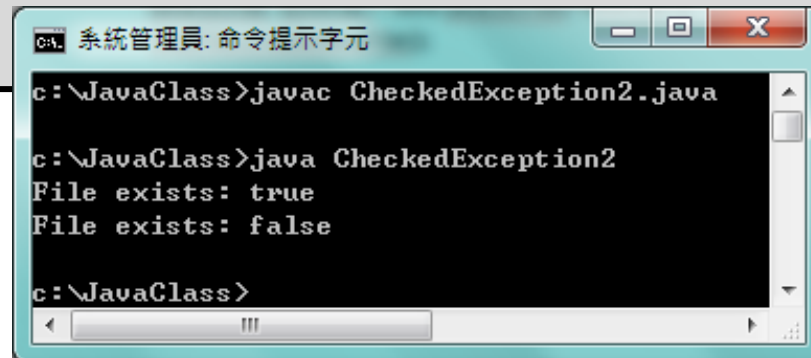
The screenshot shows a Windows command prompt window titled "C:\Windows\system32\cmd.exe". The output of the Java program is displayed as follows:

```
=== toString() ===  
java.io.FileNotFoundException: test.txt <系統找不到指定的檔案。>  
=== getLocalizedMessage() ===  
test.txt <系統找不到指定的檔案。>  
=== getMessage() ===  
test.txt <系統找不到指定的檔案。>  
=== printStackTrace() ===  
java.io.FileNotFoundException: test.txt <系統找不到指定的檔案。>  
    at java.io.FileInputStream.open(Native Method)  
    at java.io.FileInputStream.<init>(FileInputStream.java:120)  
    at java.io.FileInputStream.<init>(FileInputStream.java:79)  
    at ThrowableTest.main(ThrowableTest.java:6)
```

# Checked Exception Example

115

```
public class CheckedException2 {  
    public static void main(String[] args) {  
        try{  
            java.io.File testFile = new java.io.File("test.txt");  
            testFile.createNewFile();  
  
            System.out.println("File exists: " + testFile.exists());  
            testFile.delete();  
            System.out.println("File exists: " + testFile.exists());  
        } catch (java.io.IOException ioe){  
            System.err.println(ioe);  
        }  
    }  
}
```



```
C:\JavaClass>javac CheckedException2.java  
  
C:\JavaClass>java CheckedException2  
File exists: true  
File exists: false  
  
C:\JavaClass>
```

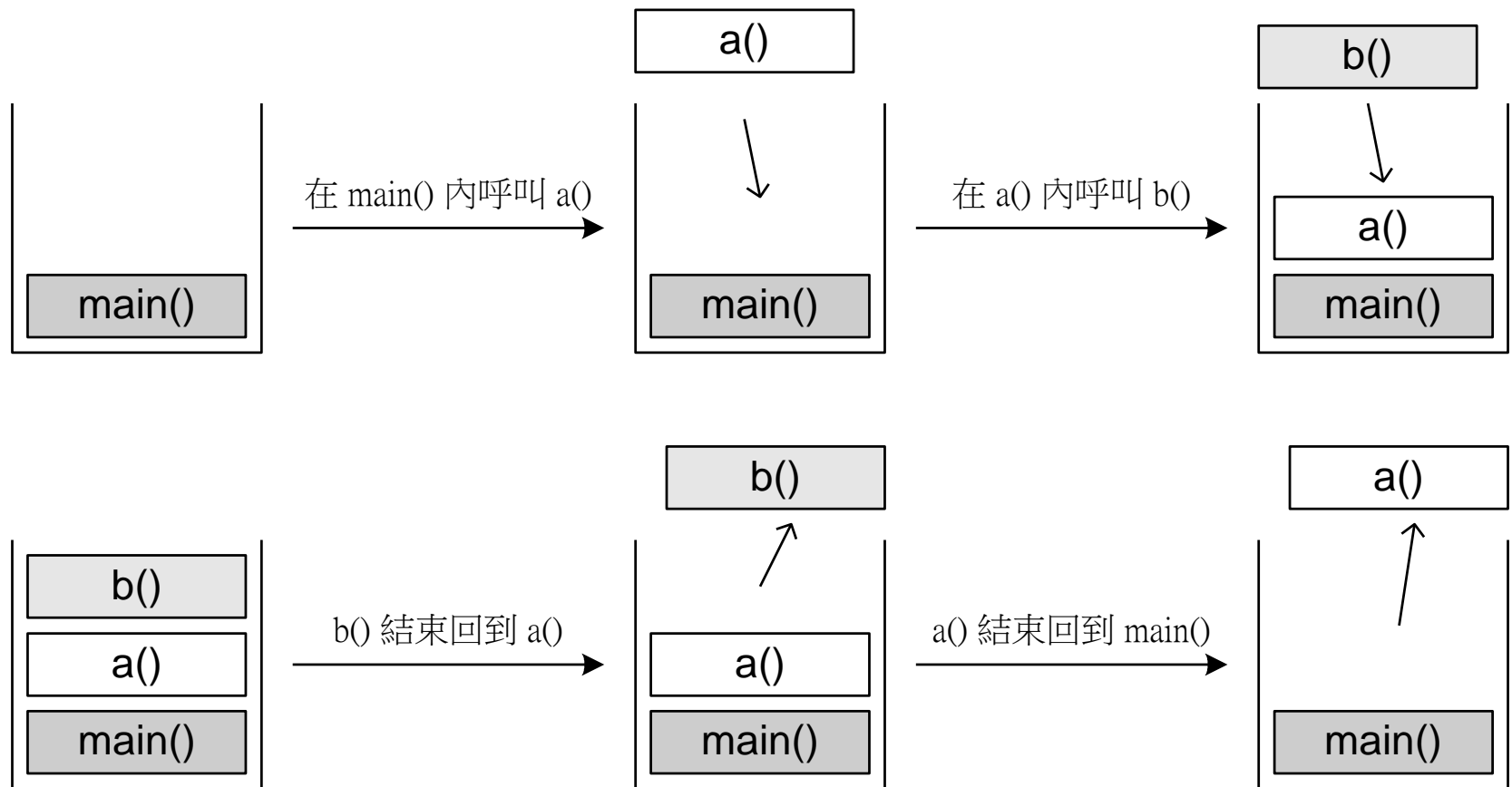
# Content

116

- 例外處理機制
- 捕捉例外 `try-catch` 敘述句
- 例外傳遞

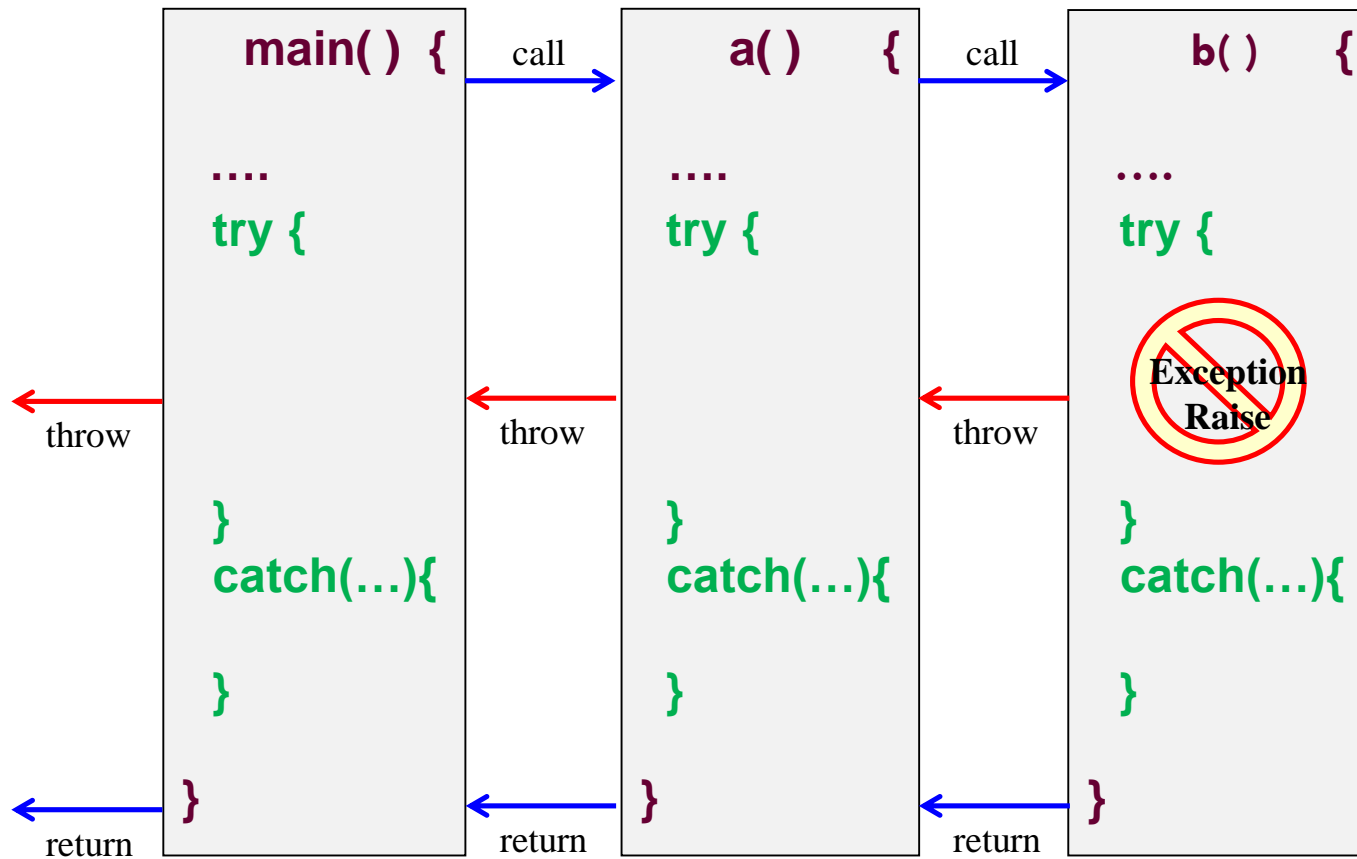
# 方法呼叫堆疊

117



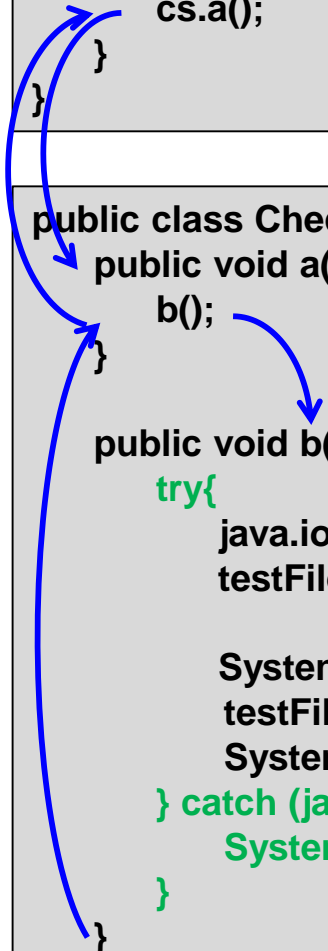
# 堆疊與例外處理

118



```
public class TestCheckedCallStack {  
    public static void main(String[] args) {  
        CheckedCallStack cs = new CheckedCallStack();  
        cs.a();  
    }  
}
```

```
public class CheckedCallStack {  
    public void a(){  
        b();  
    }  
  
    public void b() {  
        try{  
            java.io.File testFile = new java.io.File("test.txt");  
            testFile.createNewFile();  
  
            System.out.println("File exists: " + testFile.exists());  
            testFile.delete();  
            System.out.println("File exists: " + testFile.exists());  
        } catch (java.io.IOException ioe){  
            System.err.println(ioe);  
        }  
    }  
}
```



# throws

120

- 當例外發生時,可以不立即處理,而將例外丟給呼叫的方法處理
  - ▣ 被呼叫的方法內不作try/catch
  - ▣ 在方法宣告之後加上throws宣告
  - ▣ 宣告可能丟出CheckedException：系統強制方法呼叫者一定要作例外處理,否則編譯失敗
  - ▣ 宣告可能丟出UncheckedException,則無強制機制
- 例外處理 Handle or Declare Rule
  - ▣ try-catch-finally敘述處理
  - ▣ throws宣告丟出。

# throws

121

## □ 語法

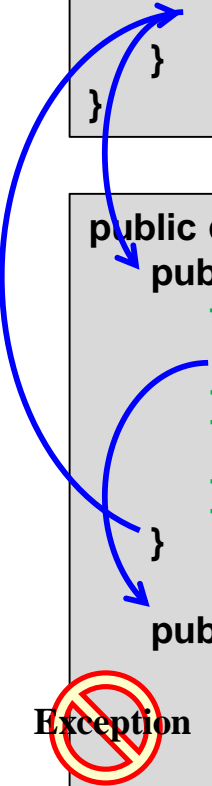
```
public void methodA(String str) throws 例外類別,... {  
    //方法敘述  
}
```

- ▣ **throws**之後可以接多個例外型別，表示方法執行過程中可能丟出屬於這些例外型別的物件。



```
public class TestCheckedCallStack2 {  
    public static void main(String[] args) {  
        CheckedCallStack2 cs = new CheckedCallStack2();  
        cs.a();  
    }  
}
```


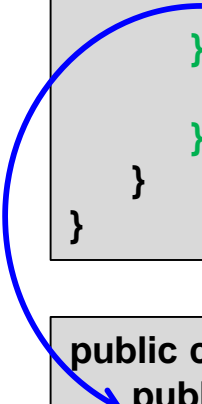

```
public class CheckedCallStack2 {  
    public void a(){  
        try{  
            b();  
        } catch (java.io.IOException ioe){  
            System.err.println(ioe);  
        }  
    }  
  
    public void b() throws java.io.IOException {  
        java.io.File testFile = new java.io.File("test.txt");  
        testFile.createNewFile();  
  
        System.out.println("File exists: " + testFile.exists());  
        testFile.delete();  
        System.out.println("File exists: " + testFile.exists());  
    }  
}
```



Exception

```
public class TestCheckedCallStack3 {  
    public static void main(String[] args) {  
        CheckedCallStack3 cs = new CheckedCallStack3();  
        try{  
            cs.a();  
        } catch (java.io.IOException ioe){  
            System.err.println(ioe);  
        }  
    }  
}
```

```
public class CheckedCallStack3 {  
    public void a() throws java.io.IOException{  
        b();  
    }  
    public void b() throws java.io.IOException {  
        java.io.File testFile = new java.io.File("test.txt");  
        Exception testFile.createNewFile();  
  
        System.out.println("File exists: " + testFile.exists());  
        testFile.delete();  
        System.out.println("File exists: " + testFile.exists());  
    }  
}
```



# throw

124

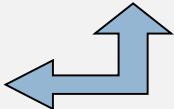
- 程式內部自行產生要丟出的例外物件
  - ▣ `throw` 所丟出的例外物件同樣可以使用`try-catch`敘述處理。
- 語法

```
throw new Exception("錯誤訊息");
```

# throw vs. throws

125

```
public void method() throws XXXException {  
    throw new XXXException();  
}
```



## □ Checked vs. Unchecked Exception

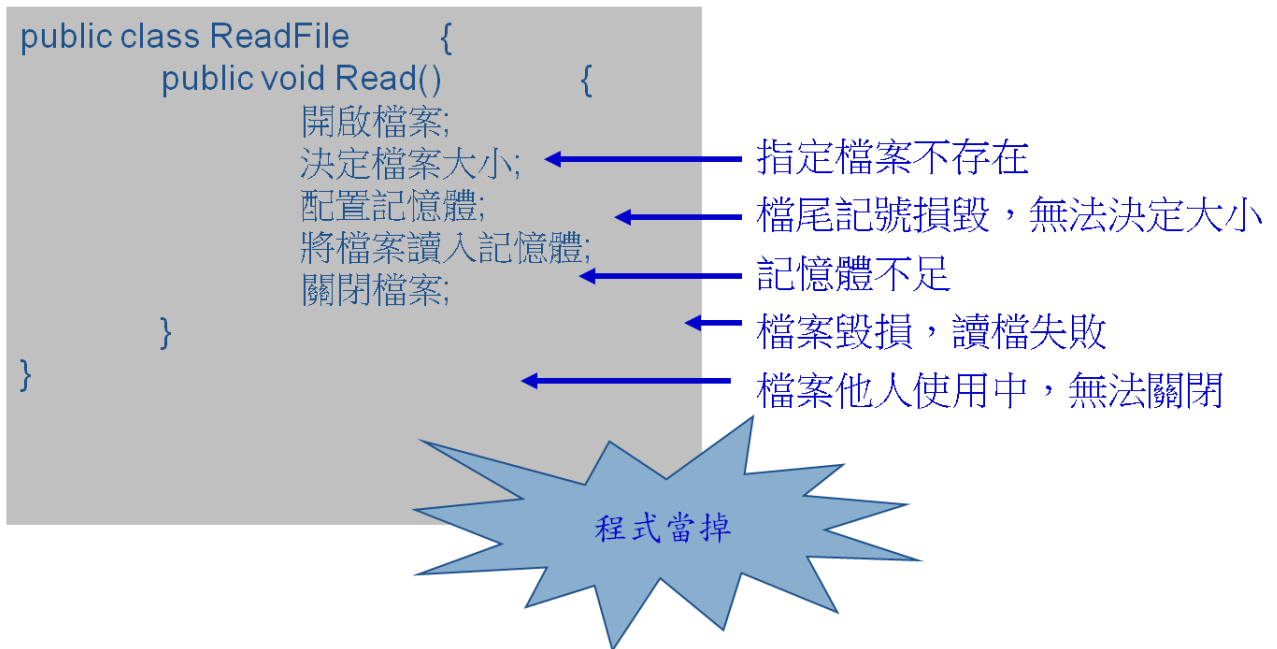
- 方法內丟出(throw) CheckedException, 必須在方法簽章宣告 throws 此例外型別
- 方法內丟出(throw) Unchecked Exception (Error/RuntimeException), 可以不在方法簽章宣告 throws 此例外型別

# Advantages of Exceptions

126

## Advantages

- ▣ 將錯誤處理程式與一般程式完全分開
- ▣ 程式錯誤會往 **call stack** 上方傳遞
- ▣ 可將錯誤型態分門別類



# Advantages of Exceptions

127

## □ 沒有例外處理時的處理方法

```
public class ReadFile {
    public int Read() {
        int ErrCode = 0;
        開啟檔案;
        if (檔案存在) {
            決定檔案大小;
            if (檔案大小已定) {
                配置記憶體;
                if (記憶體足夠) {
                    將檔案讀入記憶體;
                    if (讀入成功) {
                        開始操作檔案內容;
                    } else {
                        ErrCode = -1;
                    } else {
                        ErrCode = -2;
                    }
                }
            } else {
                ErrCode = -3;
            }
            關閉檔案;
            if (檔案無法關閉 && 無其它錯誤)
                ErrCode = -4;
            else
                ErrCode = -5;
        }
        else
            ErrCode = -6;
        return ErrCode;
    }
}
```

→ 雜亂而且原始程式邏輯被 **if ~ else** 淹沒了  
→ 更糟糕的是，程式原本的邏輯不見了！

# Advantages of Exceptions

128

## 有例外處理時的處理方法

```
Public class ReadFile {  
    public int Read() {  
        try {  
            開啟檔案;  
            決定檔案大小;  
            配置記憶體;  
            將檔案讀入記憶體;  
            關閉檔案;  
        } catch (檔案打不開) {  
            return -1;  
        } catch (無法決定大小) {  
            return -2;  
        } catch (記憶體不夠) {  
            return -3;  
        } catch (檔案讀不進來) {  
            return -4;  
        } catch (檔案關不掉) {  
            return -5;  
        }  
    }  
}
```



簡單又整齊

129

# TQC+ Java

物件導向程式設計與例外處理





# 銀行理財帳戶

130

## □ 題目一

- ▣ 題目說明：請開啟JPD06\_1.java，設計「銀行理財帳戶」程式。銀行共設有四種帳戶，需計算目前存款，請依下列題意完成作答。將JPD06\_1.java內的class JPD06\_1修改為class JPA06\_1，將檔案另存為JPA06\_1.java後編譯為JPA06\_1.class，所有題目中有使用到的類別也請編譯後一併儲存。

# 銀行理財帳戶

131

## □ 題目一設計說明

- 銀行共設有四種帳戶，分別是「定期存款」帳戶、「活期存款」帳戶、「優惠存款」帳戶及「基金存款」帳戶。前三種帳戶都有開戶人、年利率及帳戶餘額的資料。
- 每個帳戶都可以存款(deposit)、提款(withdraw)、查詢餘額(balance)、加計利息(addInterest)。
- 定期存款帳戶(DepositAccount)的年利率分1、2、3年期，各為3%、4%、5%。
- 活期存款帳戶(FreeAccount)的年利率為2%。
- 優惠存款帳戶(SpecialAccount)的利率與活期存款同為2%，但優惠存款帳戶餘額若保持在10000元以上則買賣基金可免手續費(回布林值代表可/不可)。

# 銀行理財帳戶

132

## □ 題目一設計說明

- 基金存款帳戶(FundAccount)的必要資料有：開戶人、基金名稱、單位數、一活期存款戶、一優惠存款戶。買入的手續費為買入金額的2%，賣出的手續費為賣出金額的2%。基金帳戶可以買、賣、查詢餘額，這幾個功能均需傳入基金目前的價格。買賣基金由活期存款戶轉帳。
- 請先為peter開設一定期存款帳戶(2年期，存入5000元)、活期存款帳戶(存入20000元)、優惠存款帳戶(存入10000元)。並加總一年的利息後，再查詢各帳戶的餘額，依序列出「定期存款」、「活期存款」、「優惠存款」目前的餘額。
- 接著再為peter新開設一基金存款帳戶，並買入A基金15000元，價格為每單位500元。於三天後該基金跌價為300元，請計算目前基金的餘額，及peter活期存款戶的餘額，請接續7.列出「基金現額」以及「活期餘額」

# 銀行理財帳戶

133

## □ 題目一設計說明

```
1 class Account{
2     String name;//開戶人
3     double rate;//年利率
4     int balance;//帳戶餘額
5     //建構子，設定名字和利率
6     Account(String s,double d){name = s;rate = d;}
7     //設定利率
8     void setRate(double d){rate = d;}
9     //存款
10    void deposit(int i){balance+=i;}
11    //提款
12    void withdraw(int i){balance-=i;}
13    //餘額查詢
14    int balance(){return balance;}
15    //加計利息，將利息利率整體+1
16    void addInterest(){balance*=rate+1;}
17 }
```

```
19 //定期存款戶方法，繼承Account的所有成員及方法
20 class DepositAccount extends Account{
21     DepositAccount(String s,int i)
22     {
23         super(s,0.0);//使用父親的建構子，初始化姓名和年利率
24         double d =0.0;
25         switch(i)
26         {
27             case 1:
28                 d=0.03;break;
29             case 2:
30                 d=0.04;break;
31             case 3:
32                 d=0.05;break;
33         }
34         setRate(d);//設定年利率
35     }
36 }
37
```

# 銀行理財帳戶

134

## □ 題目一設計說明

```
38 //活期存款戶方法，繼承Account的所有成員及方法
39 class FreeAccount extends Account
40 {
41     FreeAccount(String s)
42     {super(s,0.02);/*使用父親的建構子，初始化姓名和年利率*/}
43 }
44
45 //優惠存款戶方法，繼承Account的所有成員及方法
46 class SpecialAccount extends Account
47 {
48     SpecialAccount(String s){super(s,0.02);}
49     //判斷買基金是否免手續費方法，大於10000則回傳true
50     boolean isEmpty()
51     {return balance>10000;}
52 }
```

```
54 //基金存款戶方法，繼承Account的所有成員及方法
55 class FundAccount extends Account{
56     String fundName;//基金名稱
57     FreeAccount freeAccount;//活期存款戶
58     SpecialAccount specialaccount;//優惠存款戶
59     double unit;//購買基金單位(基金儲存方式是以股為單位，而不是以金額，單位金額隨時都有可能漲跌)
60     //建構子，初始化參數，從外部傳入"開戶人"、"基金名稱"、"活期存款戶物件"、"優惠存款戶物件"，四個參數
61     FundAccount(String s,String s1,FreeAccount f,SpecialAccount sa){
62         super(s,0.0);
63         fundName = s1;
64         freeAccount = f;
65         specialaccount =sa;
66     }
67     //購買基金方法
68     void buy(int i,int j)//i購買金額，j單位金額
69     {
70         //利用優惠存款檢查是否餘額大於10000再給予優惠
71         if(specialaccount.isEmpty())
72             //直接提款
73             freeAccount.withdraw(i);
74         else
75             //多提出手續費2%扣除
76             freeAccount.withdraw((int)(i*1.02));
77         //購買單位=購買金額/每單位金額
78         unit+=(double)i/(double)j;
79     }
80
81     void sell(double d,int i)//d基金單位，每單位金額
82     {
83         //利用優惠存款檢查是否餘額大於10000再給予優惠
84         if(specialaccount.isEmpty())
85             //直接存款
86             freeAccount.deposit((int)(d*i));
87         else
88             //扣除手續費2%在進行存款
89             freeAccount.deposit((int)(d*i*0.98));
90         unit-=d;
91     }
92     //將單位基金金額傳入，乘上持有單位數，回傳總基金現額
93     int balance(int i){return (int)(unit*i);}
94     //取得持有多少單位基金
95     double getUnit(){return unit;}
96 }
```

# 銀行理財帳戶

135

## □ 題目一設計說明

```
98 public class JPD06_1 {
99     public static void main(String args[]) {
100         //為Peter開個定期帳戶，兩年期的
101         DepositAccount deposit = new DepositAccount("peter", 2);
102         //存款5000元
103         deposit.deposit(5000);
104         //為Peter開個活期帳戶
105         FreeAccount free = new FreeAccount("peter");
106         //存款20000
107         free.deposit(20000);
108         //為Peter開個優惠帳戶
109         SpecialAccount special = new SpecialAccount("peter");
110         //存款10000
111         special.deposit(10000);
112         //利用加計利息增加帳戶餘額
113         deposit.addInterest();
114         free.addInterest();
115         special.addInterest();
116         //顯示個帳戶的餘額
117         System.out.println("定期存款：" + deposit.balance());
118         System.out.println("活期存款：" + free.balance());
119         System.out.println("優惠存款：" + special.balance());
120         //為Peter建立一個基金帳戶，名稱為"A"
121         FundAccount fund = new FundAccount("peter", "A", free, special);
122         //購入15000元的基金，且每單位為500元
123         fund.buy(15000, 500);
124         System.out.println("基金現額：" + fund.balance(300));
125         System.out.println("活期餘額：" + fund.freeAccount.balance());
126     }
127 }
```

# 銀行理財帳戶

136

- 執行結果參考畫面：

定期存款：5200

活期存款：20400

優惠存款：10200

基金現額：9000

活期餘額：5400

# 銀行理財帳戶

137

## □ 題目二

- ▣ 題目說明：請開啟JPD06\_2.java，Peter想要再次購買基金，請依下列題意完成作答。將JPD06\_2.java內的class JPD06\_2修改為class JPA06\_2，將檔案另存為JPA06\_2.java後編譯為JPA06\_2.class，所有題目中有使用到的類別也請編譯後一併儲存。



# 銀行理財帳戶

138

## □ 題目二設計說明

- Peter由優惠存款帳戶提款5000元並再次購買A基金2000元，價格為每單位300元
- 請列出該【基金餘額】，及Peter之活期存款帳戶【售出前活期餘額】
- 接著Peter將其基金全數賣出，賣價為每單位400元
- 請再次查詢Peter之活期存款戶的餘額

# 銀行理財帳戶

139

## □ 題目二設計說明

```
1 class Account{
2     String name;
3     double rate;
4     int balance;
5
6     Account(String s,double d){name = s;rate = d;}
7     void setRate(double d){rate = d;}
8     void deposit(int i){balance+=i;}
9     void withdraw(int i){balance-=i;}
10    int balance(){return balance;}
11    void addInterest(){balance*=rate+1;}
12 }
13
14 class DepositAccount extends Account{
15
16     DepositAccount(String s,int i)
17     {
18         super(s,0.0);
19         double d =0.0;
20         switch(i)
21         {
22             case 1:
23                 d=0.03;break;
24             case 2:
25                 d=0.04;break;
26             case 3:
27                 d=0.05;break;
28         }
29         super.setRate(d);
30     }
31 }
```

```
33 class FreeAccount extends Account
34 {
35     FreeAccount(String s)
36     {super(s,0.02);}
37 }
38
39 class SpecialAccount extends Account
40 {
41     SpecialAccount(String s)
42     {super(s,0.02);}
43     boolean isEmpty(){return balance>10000;}
44 }
```

# 銀行理財帳戶

140

## □ 題目二設計說明

```
46 class FundAccount extends Account{
47     String fundName;
48     FreeAccount freeAccount;
49     SpecialAccount specialaccount;
50     double unit;
51 FundAccount(String s,String s1,FreeAccount f,SpecialAccount sa){
52     super(s,0.0);
53     fundName = s1;
54     freeAccount = f;
55     specialaccount =sa;
56 }
57
58 void buy(int i,int j)
59 {
60     if(specialaccount.isEmpt())
61         freeAccount.withdraw(i);
62     else
63         freeAccount.withdraw((int)(i*1.02));
64     unit+=(double)i/(double)j;
65 }
66 void sell(double d,int i)
67 {
68     if(specialaccount.isEmpt())
69         freeAccount.deposit((int)(d*i));
70     else
71         freeAccount.deposit((int)(d*i*0.98));
72     unit-=d;
73 }
74 int balance(int i){return (int)(unit*i);}
75 double getUnit(){return unit;}
76 }
```

# 銀行理財帳戶

141

## □ 題目二設計說明

```
78 public class JPD06_2 {
79     public static void main(String args[]) {
80         DepositAccount deposit = new DepositAccount("peter", 2);
81         deposit.deposit(5000);
82         FreeAccount free = new FreeAccount("peter");
83         free.deposit(20000);
84         SpecialAccount special = new SpecialAccount("peter");
85         special.deposit(10000);
86         deposit.addInterest();
87         free.addInterest();
88         special.addInterest();
89         FundAccount fund = new FundAccount("peter", "A", free, special);
90         fund.buy(15000, 500);
91         //從優惠帳戶中提款5000元
92         special.withdraw(5000);
93         //再買入2000元的基金，以每單位300元購入
94         fund.buy(2000, 300);
95         System.out.println("基金餘額：" + fund.balance(300));
96         System.out.println("售出前活期餘額：" + fund.freeAccount.balance());
97         //賣出全部的股，以每單位400元賣出
98         fund.sell(fund.getUnit(), 400);
99         //這邊的fund.freeAccount.balance()，是透過fund裡面的freeAccount的balance來取出餘額的，因為fund本身的balance沒有儲存金額進去
100        System.out.println("售出後活期餘額：" + fund.freeAccount.balance());
101    }
102 }
```

# 銀行理財帳戶

142

- 執行結果參考畫面：

基金餘額：11000

售出前活期餘額：3360

售出後活期餘額：17733

# 銀行理財帳戶

143

## □ 題目三

- ▣ 題目說明：請開啟 JPD06\_3.java，該銀行為服務客戶增設網路銀行帳戶，請依下列題意完成作答。將 JPD06\_3.java 內的 class JPD06\_3 修改為 class JPA06\_3，將檔案另存為 JPA06\_3.java 後編譯為 JPA06\_3.class，所有題目中有使用到的類別也請編譯後一併儲存。

# 銀行理財帳戶

144

## □ 題目三設計說明

- ▣ 該銀行為服務客戶增設網路銀行帳戶，網路銀行帳戶含有某客戶之所有帳戶的資料。
- ▣ 請為Peter開立一網路銀行帳戶，並撰寫能夠計算Peter所有存款帳戶(不含基金)的總餘額的方法。

# 銀行理財帳戶

145

## □ 題目三設計說明

```
1 class Account{
2     String name;
3     double rate;
4     int balance;
5
6     Account(String s,double d){name = s;rate = d;}
7     void setRate(double d){rate = d;}
8     void deposit(int i){balance+=i;}
9     void withdraw(int i){balance-=i;}
10    int balance(){return balance;}
11    void addInterest(){balance*=rate+1;}
12 }
13
14 class DepositAccount extends Account{
15
16     DepositAccount(String s,int i)
17     {
18         super(s,0.0);
19         double d =0.0;
20         switch(i)
21         {
22             case 1:
23                 d=0.03;break;
24             case 2:
25                 d=0.04;break;
26             case 3:
27                 d=0.05;break;
28         }
29         super.setRate(d);
30     }
31 }
```

```
33 class FreeAccount extends Account
34 {FreeAccount(String s){super(s,0.02);}}
35
36 class SpecialAccount extends Account
37 {
38     SpecialAccount(String s)
39     {super(s,0.02);}
40     boolean isEmpty(){return balance>10000;}
41 }
```



# 銀行理財帳戶

146

## □ 題目三設計說明

```
43 class FundAccount extends Account{
44     String fundName;
45     FreeAccount freeAccount;
46     SpecialAccount specialaccount;
47     double unit;
48     FundAccount(String s,String s1,FreeAccount f,SpecialAccount sa){
49         super(s,0.0);
50         fundName = s1;
51         freeAccount = f;
52         specialaccount =sa;
53     }
54
55     void buy(int i,int j)
56     {
57         if(specialaccount.isEmpty())
58             freeAccount.withdraw(i);
59         else
60             freeAccount.withdraw((int)(i*1.02));
61         unit+=(double)i/(double)j;
62     }
63     void sell(double d,int i)
64     {
65         if(specialaccount.isEmpty())
66             freeAccount.deposit((int)(d*i));
67         else
68             freeAccount.deposit((int)(d*i*0.98));
69         unit-=d;
70     }
71     int balance(int i){return (int)(unit*i);}
72     double getUnit(){return unit;}
73 }
```

# 銀行理財帳戶

147

## □ 題目三設計說明

```
74 //建立網路帳戶方法
75 class InternetAccount{
76     DepositAccount deposit;
77     FreeAccount free;
78     SpecialAccount specisl;
79     FundAccount fund;
80     InternetAccount(){}
81     //從外部傳入定期存款戶物件
82     void setDeposit(DepositAccount d){deposit = d;}
83     //從外部傳入活期存款戶物件
84     void setFree(FreeAccount f){free=f;}
85     //從外部傳入優惠存款戶物件
86     void setSpecial(SpecialAccount s){specisl=s;}
87     void setFund(FundAccount ff){fund=ff;}
88     int getTotalBalance(){return deposit.balance+free.balance+specisl.balance;}
89 }
```

# 銀行理財帳戶

148

## □ 題目三設計說明

```
91 public class JPD06_3 {
92     public static void main(String args[]) {
93         DepositAccount deposit = new DepositAccount("peter", 2);
94         deposit.deposit(5000);
95         FreeAccount free = new FreeAccount("peter");
96         free.deposit(20000);
97         SpecialAccount special = new SpecialAccount("peter");
98         special.deposit(10000);
99         deposit.addInterest();
100        free.addInterest();
101        special.addInterest();
102        FundAccount fund = new FundAccount("peter", "A", free, special);
103        fund.buy(15000, 500);
104        special.withdraw(5000);
105        fund.buy(2000, 300);
106        fund.sell(fund.getUnit(), 400);
107        //產生一個網路銀行的物件
108        InternetAccount internet = new InternetAccount();
109        //設定定期帳戶
110        internet.setDeposit(deposit);
111        //設定活期帳戶
112        internet.setFree(free);
113        //設定優惠帳戶
114        internet.setSpecial(special);
115        //設定基金帳戶
116        internet.setFund(fund);
117        System.out.println("存款總額：" + internet.getTotalBalance());
118    }
119 }
```

# 銀行理財帳戶

149

- 執行結果參考畫面：

存款總額： 28133

# 銀行理財帳戶

150

## □ 題目四

- ▣ 題目說明：請開啟JPD06\_4.java，Peter想要購買多筆基金，請依下列題意完成作答。將JPD06\_4.java內的class JPD06\_4修改為class JPA06\_4，將檔案另存為JPA06\_4.java後編譯為JPA06\_4.class，所有題目中有使用到的類別也請編譯後一併儲存。

# 銀行理財帳戶

151

## □ 題目四設計說明

- Peter決定買入B基金2000元，買價每單位50元；C基金5000元，買價每單位30元。
- 請查詢Peter之活期存款帳戶的餘額及A、B、C三個基金的單位數。
- 基金B的價格現在漲到100，Peter想要查詢基金B的餘額。請使用HashMap為Peter建立一MultiFund(多筆基金)類別，HashMap的key值為基金名稱(String)，value值為FundAccount，使其代入B基金後，可輸出目前B基金的金額。

# 銀行理財帳戶

152

## □ 題目四設計說明

```
1 import java.util.HashMap;
2 import java.util.Iterator;
3
4 class Account{
5     String name;
6     double rate;
7     int balance;
8
9     Account(String s,double d)
10    {name = s;rate = d;}
11
12    void setRate(double d)
13    {rate = d;}
14
15    void deposit(int i)
16    {balance+=i;}
17
18    void withdraw(int i)
19    {balance-=i;}
20
21    int balance()
22    {return balance;}
23
24    void addInterest()
25    {balance*=rate+1;}
26 }
```

```
28 class DepositAccount extends Account{
29
30     DepositAccount(String s,int i)
31     {
32         super(s,0.0);
33         double d =0.0;
34         switch(i)
35         {
36             case 1:
37                 d=0.03;break;
38             case 2:
39                 d=0.04;break;
40             case 3:
41                 d=0.05;break;
42         }
43         super.setRate(d);
44     }
45 }
46
47 class FreeAccount extends Account
48 {
49     FreeAccount(String s)
50     {super(s,0.02);}
51 }
52
53 class SpecialAccount extends Account
54 {
55     SpecialAccount(String s)
56     {
57         super(s,0.02);
58     }
59     boolean isEmpty()
60     {
61         return balance>10000;
62     }
63 }
64 }
```

# 銀行理財帳戶

153

## □ 題目四設計說明

```
66 class FundAccount extends Account{
67     String fundName;
68     FreeAccount freeAccount;
69     SpecialAccount specialaccount;
70     double unit;
71
72     FundAccount(String s,String s1,FreeAccount f,SpecialAccount sa){
73         super(s,0.0);
74         fundName = s1;
75         freeAccount = f;
76         specialaccount =sa;
77     }
78
79
80     void buy(int i,int j)
81     {
82         if(specialaccount.isEmpt())
83             freeAccount.withdraw(i);
84         else
85             freeAccount.withdraw((int)(i*1.02));
86         unit+=(double)i/(double)j;
87     }
88     void sell(double d,int i)
89     {
90         if(specialaccount.isEmpt())
91             freeAccount.deposit((int)(d*i));
92         else
93             freeAccount.deposit((int)(d*i*0.98));
94         unit-=d;
95     }
96     int balance(int i)
97     {
98         return (int)(unit*i);
99     }
100     double getUnit()
101     {
102         return unit;
103     }
104 }
```



# 銀行理財帳戶

154

## □ 題目四設計說明

```
106 class InternetAccount{
107     DepositAccount deposit;
108     FreeAccount free;
109     SpecialAccount specisl;
110     FundAccount fund;
111     InternetAccount(){}
112     void setDeposit(DepositAccount d){deposit = d;}
113     void setFree(FreeAccount f){free=f;}
114     void setSpecial(SpecialAccount s){specisl=s;}
115     void setFund(FundAccount ff){fund=ff;}
116     int getTotalBalance()
117     {return deposit.balance+free.balance+specisl.balance;}
118 }
119
120 //建立一個多基金的方法
121 class MultiFund
122 {
123     HashMap funds;
124     //初始化建構子，使其產生一個HashMap
125     MultiFund(){funds = new HashMap();}
126     //增加新的基金方法
127     void addFund(String s, FundAccount fundaccount)
128     {funds.put(s, fundaccount);}
129     //列出所有基金和擁有單位數
130     void printEachUnit()
131     {
132         FundAccount fundaccount ;
133         for(Iterator iterator = funds.values().iterator();iterator.hasNext();)
134         {
135             fundaccount = (FundAccount)iterator.next();
136             System.out.println(fundaccount.fundName+" : "+fundaccount.getUnit());
137         }
138     }
139     //建立一個方法，傳入基金名稱、傳入單位金額
140     int getFundBalance(String s ,int i)
141     {
142         return ((FundAccount)funds.get(s)).balance(i);
143     }
144 }
```

# 銀行理財帳戶

155

## □ 題目四設計說明

```
146 public class JPD06_4 {
147     public static void main(String args[]) {
148         DepositAccount deposit = new DepositAccount("peter", 2);
149         deposit.deposit(5000);
150         FreeAccount free = new FreeAccount("peter");
151         free.deposit(20000);
152         SpecialAccount special = new SpecialAccount("peter");
153         special.deposit(10000);
154         deposit.addInterest();
155         free.addInterest();
156         special.addInterest();
157         FundAccount fund = new FundAccount("peter", "A", free, special);
158         fund.buy(15000, 500);
159         special.withdraw(5000);
160         fund.buy(2000, 300);
161         fund.sell(fund.getUnit(), 400);
162         InternetAccount internet = new InternetAccount();
163         internet.setDeposit(deposit);
164         internet.setFree(free);
165         internet.setSpecial(special);
166         internet.setFund(fund);
167         //建立一個多帳戶的物件
168         MultiFund multi = new MultiFund();
169         //增加A基金(因A基金本身就存在,不需再另外建立)
170         multi.addFund("A", fund);
171         //建立一個B基金
172         FundAccount fundB = new FundAccount("peter", "B", free, special);
173         //購買每單位50元的基金, 2000元
174         fundB.buy(2000, 50);
175         //增加B基金
176         multi.addFund("B", fundB);
177         //建立一個C基金
178         FundAccount fundC = new FundAccount("peter", "C", free, special);
179         //購買每單位30元的基金, 5000元
180         fundC.buy(5000, 30);
181         multi.addFund("C", fundC);
182         System.out.println("活期餘額: " + free.balance());
183         //顯示特定基金的總現值,須傳入基金名稱和單位金額
184         multi.printEachUnit();
185         System.out.println("B 基金餘額: " + multi.getFundBalance("B", 100));
186     }
187 }
```

# 銀行理財帳戶

156

□ 執行結果參考畫面：

活期餘額：10593

A:0.0

B:40.0

C:166.66666666666666

B 基金餘額：4000

# 銀行理財帳戶

157

## □ 題目五

- ▣ 題目說明：請開啟JPD06\_5.java，接續題目四的基金購買，請依下列題意完成作答。將JPD06\_5.java內的class JPD06\_5修改為class JPA06\_5，將檔案另存為JPA06\_5.java後編譯為JPA06\_5.class，所有題目中有使用到的類別也請編譯後一併儲存。

# 銀行理財帳戶

158

## □ 題目五設計說明

- 請使用Exception撰寫一功能：當任何一個帳戶之餘額低於0時，不可提款，且印出警告信息。
- Peter再次購買A基金14000元, 價格為每單位300元。

# 銀行理財帳戶

159

## □ 題目五設計說明

```
1 import java.util.HashMap;
2 import java.util.Iterator;
3
4 class Account{
5     String name;
6     double rate;
7     int balance;
8
9     Account(String s,double d){name = s;rate = d;}
10
11     void setRate(double d){rate = d;}
12
13     void deposit(int i){balance+=i;}
14
15     void withdraw(int i) throws Exception
16     {
17         //設定當提款金額大於存款金額時，丟出錯誤訊息
18         if(balance<i)
19             {throw new Exception(name+"提款金額:"+i+"大於存款金額:"+balance);}
20         else
21             {balance-=i;
22              return;
23             }
24     }
25
26     int balance()
27     {return balance;}
28
29     void addInterest()
30     {balance*=rate+1;}
31 }
```

# 銀行理財帳戶

160

## ■ 題目五設計說明

```
32 class DepositAccount extends Account{
33
34     DepositAccount(String s,int i)
35     {
36         super(s,0.0);
37         double d =0.0;
38         switch(i)
39         {
40             case 1:
41                 d=0.03;break;
42             case 2:
43                 d=0.04;break;
44             case 3:
45                 d=0.05;break;
46             }
47         super.setRate(d);
48     }
49 }
50
51 class FreeAccount extends Account
52 {
53     FreeAccount(String s)
54     {super(s,0.02);}
55 }
56
57 class SpecialAccount extends Account
58 {
59     SpecialAccount(String s)
60     {
61         super(s,0.02);
62     }
63     boolean isEmpty()
64     {
65         return balance>10000;
66     }
67 }
```

```
69 class FundAccount extends Account{
70     String fundName;
71     FreeAccount freeAccount;
72     SpecialAccount specialaccount;
73     double unit;
74
75     FundAccount(String s,String s1,FreeAccount f,SpecialAccount sa){
76         super(s,0.0);
77         fundName = s1;
78         freeAccount = f;
79         specialaccount =sa;
80     }
81
82     void buy(int i,int j)
83     {
84         //此處也有使用到提款功能，也須將try catch嵌入
85         try{
86             if(specialaccount.isEmpty())
87                 freeAccount.withdraw(i);
88             else
89                 freeAccount.withdraw((int)(i*1.02));
90             unit+=(double)i/(double)j;
91         }
92         catch(Exception ex)
93         {System.out.println(ex.getMessage());}
94     }
95     void sell(double d,int i)
96     {
97         if(specialaccount.isEmpty())
98             freeAccount.deposit((int)(d*i));
99         else
100             freeAccount.deposit((int)(d*i*0.98));
101         unit-=d;
102     }
103     int balance(int i)
104     {
105         return (int)(unit*i);
106     }
107     double getUnit()
108     {
109         return unit;
110     }
111 }
```

# 銀行理財帳戶

161

## □ 題目五設計說明

```
113 class InternetAccount{
114     DepositAccount deposit;
115     FreeAccount free;
116     SpecialAccount specisl;
117     FundAccount fund;
118     InternetAccount(){ }
119     void setDeposit(DepositAccount d){deposit = d;}
120     void setFree(FreeAccount f){free=f;}
121     void setSpecial(SpecialAccount s){specisl=s;}
122     void setFund(FundAccount ff){fund=ff;}
123     int getTotalBalance()
124     {return deposit.balance+free.balance+specisl.balance;}
125 }
126
127 class MultiFund
128 {
129
130     HashMap funds;
131     MultiFund(){funds = new HashMap();}
132     void addFund(String s, FundAccount fundaccount)
133     {funds.put(s, fundaccount);}
134
135     void printEachUnit()
136     {
137         FundAccount fundaccount ;
138         for(Iterator iterator = funds.values().iterator();iterator.hasNext();System.out.println(fundaccount.fundName+" : "+fundaccount.getUnit()))
139         {
140             fundaccount = (FundAccount)iterator.next();
141         }
142     }
143
144     int getFundBalance(String s ,int i)
145     {
146         return ((FundAccount)funds.get(s)).balance(i);
147     }
148 }
149 }
```



# 銀行理財帳戶

162

## □ 題目五設計說明

```
150 public class JPD06_5 {
151     public static void main(String args[]) {
152
153         DepositAccount deposit = new DepositAccount("peter", 2);
154         deposit.deposit(5000);
155         FreeAccount free = new FreeAccount("peter");
156         free.deposit(20000);
157         SpecialAccount special = new SpecialAccount("peter");
158         special.deposit(10000);
159         deposit.addInterest();
160         free.addInterest();
161         special.addInterest();
162         FundAccount fund = new FundAccount("peter", "A", free, special);
163         //加入try catch，確保可以抓取錯誤提款訊息
164         try {
165             fund.buy(15000, 500);
166             special.withdraw(5000);
167             fund.buy(2000, 300);
168             fund.sell(fund.getUnit(), 400);
169             InternetAccount internet = new InternetAccount();
170             internet.setDeposit(deposit);
171             internet.setFree(free);
172             internet.setSpecial(special);
173             internet.setFund(fund);
174             MultiFund multi = new MultiFund();
175             multi.addFund("A", fund);
176             FundAccount fundB = new FundAccount("peter", "B", free, special);
177             fundB.buy(2000, 50);
178             multi.addFund("B", fundB);
179             FundAccount fundC = new FundAccount("peter", "C", free, special);
180             fundC.buy(5000, 30);
181             multi.addFund("C", fundC);
182             fund.buy(14000, 300);
183         } catch (Exception e) {
184             System.out.println(e.getMessage());
185         }
186     }
187 }
```

# 銀行理財帳戶

163

- 執行結果參考畫面：

```
peter:提款金額: 14280 大於存款餘額: 10593
```

# 員工薪資制度

164

## □ 題目一

- ▣ 題目說明：請開啟JPD06\_1.java，設計「員工薪資制度」程式，請依下列題意完成作答。將JPD06\_1.java內的class JPD06\_1修改為class JPA06\_1，將檔案另存為JPA06\_1.java後編譯為JPA06\_1.class，所有題目中有使用到的類別也請編譯後一併儲存。

# 員工薪資制度

165

## □ 題目一設計說明

- 該工廠有三種類別的員工。
- `SalaryWorker` 以「年薪」（月薪為年薪除以12）計算薪水。
- `HourlyWorker` 以「時薪」計算薪水。
- `Manager` 必須是 `SalaryWorker`，但是除了年薪之外，每月還有紅利。
- 每名員工都有編號。
- 請宣告一個 `SalaryWorker` 類別。其員工編號為 96001，年薪為180000，撰寫 `monthPay` 方法以計算這名員工該月的收入。

# 員工薪資制度

166

## □ 題目一設計說明

- 請宣告一個 `HourlyWorker` 類別。其員工編號為 `96002`，每月工時160小時，時薪100，撰寫`monthPay`方法計算這名員工該月的收入。
- 請宣告一個`Manager`類別。其員工編號為 `97001`，年薪為 `240000`，每月的紅利為 `5000`，撰寫 `monthPay`方法計算這名員工該月的收入。
- 請分別顯示三位員工的薪水，顯示結果如下頁所示：

# 員工薪資制度

167

## □ 題目一設計說明

```
1 //建立抽象類別供子類別繼承
2 abstract class work{
3     String wno;
4     work(String s)
5     {wno=s;}
6
7     //建立一個每月收入抽象的方法
8     abstract double monthPay();
9
10 }
11 //建立一個SalaryWorker類別，繼承work，在此方法中有寫入紅利變數，但設定成0
12 class SalaryWorker extends work{
13     int mp;//年薪
14     int redp=0;//紅利
15     //建構子初始化員工編號、年薪
16     SalaryWorker(String s ,int i)
17     {super(s);mp=i;}
18     double monthPay(){return (mp/12.0+redp);}
19 }
20
21 //建立一個HourlyWorker類別，繼承work
22 class HourlyWorker extends work{
23     int hr, hp;
24     //建構子初始化員工編號、工作時數、時薪
25     HourlyWorker(String s, int i1, int i2)
26     {super(s); hr=i2; hp=i1;}
27     double monthPay()
28     {return hr*hp;}
29 }
30 //建立一個主管類別繼承SalaryWorker，該處將紅利的變數重新寫入
31 class Manager extends SalaryWorker{
32     Manager(String s, int i ,int i1)
33     {
34         super(s,i);
35         redp = i1;
36     }
37 }
```

# 員工薪資制度

168

## □ 題目一設計說明

```
39 public class JPD06_1 {
40     public static void main(String argv[]) {
41         //建立一個銷售員的物件
42         SalaryWorker sw1 = new SalaryWorker("96001",180000);
43         //取的該銷售員的月薪
44         System.out.println("SalaryWorker:" + sw1.monthPay());
45         //建立一個時薪人員的物件
46         HourlyWorker hw1 = new HourlyWorker("96002", 100, 160);
47         //取的該時薪人員的月薪
48         System.out.println("HourlyWorker:" + hw1.monthPay());
49         //建立一個主管的物件
50         Manager ma1 = new Manager("97001", 240000, 5000);
51         //取得該主管的月薪
52         System.out.println("Manager:" + ma1.monthPay());
53     }
54 }
```

# 員工薪資制度

169

- 執行結果參考畫面：

```
SalaryWorker: 15000.0
```

```
HourlyWorker: 16000.0
```

```
Manager: 25000.0
```



# 員工薪資制度

170

## □ 題目二

- ▣ 題目說明：請開啟JPD06\_2.java，比較兩個員工的收入何者較高，請依下列題意完成作答。將JPD06\_2.java內的class JPD06\_2修改為class JPA06\_2，將檔案另存為JPA06\_2.java後編譯為JPA06\_2.class，所有題目中有使用到的類別也請編譯後一併儲存。

# 員工薪資制度

171

## □ 題目二設計說明

- 根據上題說明，依 `SalaryWorker`、`HourlyWorker`、`Manager` 順序先印出此三位的該月收入。
- 請寫一方法，首先比較`SalaryWorker` 與 `HourlyWorker` 兩者的該月收入，何者較高；再比較`HourlyWorker`與`Manager`的月收入何者較高，並接續列印薪資較高的員工代號，顯示【員工代號+較高】。
- 再請計算每位員工該月的應繳稅額，稅率均為薪水的15%，分別列出`SalaryWorker`、`HourlyWorker`、`Manager` 的稅額。

# 員工薪資制度

172

## □ 題目二設計說明

```
1  abstract class work{
2      String wno;
3      work(String s){
4          wno=s;
5      }
6      abstract double monthPay();
7      //建立一個比較薪水高低的方法
8      void isheight(work k){
9          if(monthPay()>k.monthPay())
10             System.out.println(wno+"較高");
11         else
12             System.out.println(k.wno+"較高");
13     }
14     //計算出每個人的應繳稅額
15     double monthTaxes(){
16         return monthPay()*0.15;
17     }
18 }
19
20 class SalaryWorker extends work{
21     int mp;
22     int redp=0;
23     SalaryWorker(String s ,int i)
24     {super(s);mp=i;}
25     double monthPay()
26     {return (mp/12.0+redp);}
27 }
28
29 class HourlyWorker extends work{
30     int hr, hp;
31     HourlyWorker(String s,int i1,int i2)
32     {super(s);hr=i2;hp=i1;}
33     double monthPay()
34     {return hr*hp;}
35 }
```

# 員工薪資制度

173

## □ 題目二設計說明

```
37 class Manager extends SalaryWorker{
38
39     Manager(String s,int i ,int i1)
40     {
41         super(s,i);
42         redp = i1;
43     }
44 }
45
46 public class JPD06_2 {
47     public static void main(String argv[]) {
48
49         SalaryWorker sw1 = new SalaryWorker("96001",180000);
50         System.out.println("SalaryWorker:" + sw1.monthPay());
51         HourlyWorker hw1 = new HourlyWorker("96002", 100, 160);
52         System.out.println("HourlyWorker:" + hw1.monthPay());
53         Manager ma1 = new Manager("97001", 240000, 5000);
54         System.out.println("Manager:" + ma1.monthPay());
55
56         //比較(物件)sw1和(物件)hw1誰的薪水高
57         sw1.ishight(hw1);
58         //比較(物件)hw1和(物件)ma1誰的薪水高
59         hw1.ishight(ma1);
60         //取得每個人的應繳稅額
61         System.out.println("SalaryWorker稅額:" + sw1.monthTaxes());
62         System.out.println("HourlyWorker稅額:" + hw1.monthTaxes());
63         System.out.println("Manager稅額:" + ma1.monthTaxes());
64     }
65 }
```

# 員工薪資制度

174

- 執行結果參考畫面：

SalaryWorker: 15000.0

HourlyWorker: 16000.0

Manager: 25000.0

96002較高

97001較高

SalaryWorker稅額: 2250.0

HourlyWorker稅額: 2400.0

Manager稅額: 3750.0

# 員工薪資制度

175

## □ 題目三

- ▣ 題目說明：請開啟JPD06\_3.java，計算員工的總月繳稅額，請依下列題意完成作答。將JPD06\_3.java內的class JPD06\_3修改為class JPA06\_3，將檔案另存為JPA06\_3.java後編譯為JPA06\_3.class，所有題目中有使用到的類別也請編譯後一併儲存。

# 員工薪資制度

176

## □ 題目三設計說明

- ▣ 請以類別變數累計員工人數，並在每次計算某一員工的月繳稅額時，以類別變數累計總月繳稅額。
- ▣ 撰寫一個能夠傳回所有員工的平均繳稅金額的類別方法。

# 員工薪資制度

177

## □ 題目三設計說明

```
1  abstract class work{
2      String wno;
3      //新增員工人數變數
4      static int wt=0;
5      //新增總應繳稅額變數
6      static double tottax= 0.0;
7      work(String s){
8          wno=s;
9          wt++;
10     }
11     abstract double monthPay();
12     //建立取得平均應繳稅金額
13     static double getAverageTax(){
14         return ((double)tottax/wt);
15     }
16
17     void ishigh(work k){
18         if(monthPay()>k.monthPay())
19             System.out.println(wno+"較高");
20         else
21             System.out.println(k.wno+"較高");
22     }
23
24     double monthTaxes(){
25         double sssss=monthPay()*0.15;
26         tottax = tottax+sssss;
27         return sssss;
28     }
29 }
30
31 class SalaryWorker extends work{
32     int mp;
33     int redp=0;
34     SalaryWorker(String s ,int i)
35     {super(s);mp=i;}
36     double monthPay()
37     {return (mp/12.0+redp);}
38 }
```



# 員工薪資制度

178

## □ 題目三設計說明

```
40 class HourlyWorker extends work{
41     int hr, hp;
42     HourlyWorker(String s, int i1, int i2)
43     {super(s); hr=i2; hp=i1;}
44     double monthPay()
45     {return hr*hp;}
46 }
47
48 class Manager extends SalaryWorker{
49     Manager(String s, int i, int i1)
50     {
51         super(s, i);
52         redp = i1;
53     }
54 }
55
56 public class JPD06_3 {
57     public static void main(String argv[]) {
58         SalaryWorker sw1 = new SalaryWorker("96001", 180000);
59         HourlyWorker hw1 = new HourlyWorker("96002", 100, 160);
60         Manager ma1 = new Manager("97001", 240000, 5000);
61
62         System.out.println("SalaryWorker稅額：" + sw1.monthTaxes());
63         System.out.println("HourlyWorker稅額：" + hw1.monthTaxes());
64         System.out.println("Manager稅額：" + ma1.monthTaxes());
65         //計算出平均稅額
66         System.out.println("平均稅額：" + work.getAverageTax());
67     }
68 }
```

# 員工薪資制度

179

- 執行結果參考畫面：

SalaryWorker稅額: 2250.0

HourlyWorker稅額: 2400.0

Manager稅額: 3750.0

平均稅額: 2800.0

# 員工薪資制度

180

## □ 題目四

- ▣ 題目說明：請開啟JPD06\_4.java，計算員工稅後薪資，請依下列題意完成作答。  
將 JPD06\_4.java 內的 class JPD06\_4 修改為 class JPA06\_4，將檔案另存為 JPA06\_4.java後編譯為JPA06\_4.class，所有題目中有使用到的類別也請編譯後一併儲存。

# 員工薪資制度

181

## □ 題目四設計說明

- ▣ 請使用 `HashMap` 寫一個 `Management` 人事管理類別，`Management` 提供一個能夠傳入員工編號，然後傳回該員工稅後薪資的方法 `afterTax`。
- ▣ 請查詢 `97001` 的稅後薪資。

# 員工薪資制度

182

## □ 題目四設計說明

```
1 import java.util.HashMap;
2
3 abstract class work{
4     String wno;
5     static int wt=0;
6     static double tottax= 0.0;
7     work(String s){
8         wno=s;
9         wt++;
10    }
11    abstract double monthPay();
12
13    static double getAverageTax(){
14
15        return ((double)tottax/wt);
16    }
17
18    void ishigh(work k){
19        if(monthPay()>k.monthPay())
20            System.out.println(wno+"較高");
21        else
22            System.out.println(k.wno+"較高");
23    }
24
25    double monthTaxes(){
26        double sssss=monthPay()*0.15;
27        tottax = tottax+sssss;
28        return sssss;
29    }
30 }
31
32 class SalaryWorker extends work{
33     int mp;
34     int redp=0;
35     SalaryWorker(String s ,int i)
36     {super(s);mp=i;}
37     double monthPay()
38     {return (mp/12.0+redp);}
39 }
```

```
41 class HourlyWorker extends work{
42     int hr, hp;
43     HourlyWorker(String s, int i1, int i2)
44     {super(s); hr=i2; hp=i1;}
45     double monthPay()
46     {return hr*hp;}
47 }
48
49 class Manager extends SalaryWorker{
50     Manager(String s, int i ,int i1)
51     {
52         super(s, i);
53         redp = i1;
54     }
55 }
56
57 //建立一個管理的類別
58 class Management{
59     HashMap worker;
60     //建構子初始化物件為HashMap
61     Management(){worker=new HashMap();}
62     //建立方法將物件放入HashMap中
63     void put(String s ,work ww)
64     {worker.put(s, ww);}
65     //建立方法取得扣除稅後的薪資
66     double afterTax(String s )
67     {
68         work w = (work)worker.get(s);
69         return w.monthPay()-w.monthTaxes();
70     }
71 }
```

# 員工薪資制度

183

## □ 題目四設計說明

```
73 public class JPD06_4 {  
74     public static void main(String argv[]) {  
75         SalaryWorker sw1 = new SalaryWorker("96001",180000);  
76         HourlyWorker hw1 = new HourlyWorker("96002", 100, 160);  
77         Manager ma1 = new Manager("97001", 240000, 5000);  
78         //建立一個管理的物件  
79         Management m = new Management();  
80         //將資料放素HashMap中  
81         m.put("96001", sw1);  
82         m.put("96002", hw1);  
83         m.put("97001", ma1);  
84         System.out.println("97001 的稅後薪資： " + m.afterTax("97001"));  
85     }  
86 }
```

# 員工薪資制度

184

- 執行結果參考畫面：

97001 的稅後薪資： 21250.0

# 員工薪資制度

185

## □ 題目五

- ▣ 題目說明：請開啟JPD06\_5.java，請為題目四增加例外處理的功能，請依下列題意完成作答。將JPD06\_5.java內的class JPD06\_5修改為class JPA06\_5，將檔案另存為JPA06\_5.java後編譯為JPA06\_5.class，所有題目中有使用到的類別也請編譯後一併儲存。



# 員工薪資制度

186

## □ 題目五設計說明

- ▣ 請為 `Management` 寫一個計算員工總月薪的方法。如果總月薪超過50000，則以`exception`印出警告信息。

# 員工薪資制度

187

## □ 題目五設計說明

```
1 import java.util.HashMap;
2 import java.util.Iterator;
3
4 abstract class work{
5     String wno;
6     static int wt=0;
7     static double tottax= 0.0;
8     work(String s){
9         wno=s;
10        wt++;
11    }
12    abstract double monthPay();
13
14    static double getAverageTax(){
15        return ((double)tottax/wt);
16    }
17
18    void ishigh(work k){
19        if(monthPay()>k.monthPay())
20            System.out.println(wno+"較高");
21        else
22            System.out.println(k.wno+"較高");
23    }
24
25    double monthTaxes(){
26        double sssss=monthPay()*0.15;
27        tottax = tottax+sssss;
28        return sssss;
29    }
30 }
```

# 員工薪資制度

188

## □ 題目五設計說明

```
32 class SalaryWorker extends work{
33     int mp;
34     int redp=0;
35     SalaryWorker(String s ,int i)
36     {super(s);mp=i;}
37     double monthPay()
38     {return (mp/12.0+redp);}
39 }
40
41 class HourlyWorker extends work{
42     int hr, hp;
43     HourlyWorker(String s, int i1, int i2)
44     {super(s); hr=i2; hp=i1;}
45     double monthPay()
46     {return hr*hp;}
47 }
48
49 class Manager extends SalaryWorker{
50     Manager(String s, int i ,int i1)
51     {
52         super(s, i);
53         redp = i1;
54     }
55 }
56
57 class Management{
58     HashMap worker;
59     double total =0;
60     Management(){worker=new HashMap();}
61
62     void put(String s ,work ww)
63     {
64         worker.put(s, ww);
65     }
66
67     double totalSalary() throws limex{
68         for(Iterator iterator = worker.values().iterator(); iterator.hasNext();){
69             work www =(work)iterator.next();
70             total = www.monthPay()+total;
71             if(total>50000)//當總月薪超過50000時，則會拋出錯誤訊息
72                 throw new limex(total);
73         }
74         return total;
75     }
76 }
```

# 員工薪資制度

189

## □ 題目五設計說明

```
78 //新增一個exception的類別
79 class limex extends Exception{
80     double dd;
81     limex(double d)
82     {dd=d;}
83     double getAm(){return dd;}
84 }
85 public class JPD06_5 {
86     public static void main(String argv[]) {
87         SalaryWorker sw1 = new SalaryWorker("96001",180000);
88         HourlyWorker hw1 = new HourlyWorker("96002", 100, 160);
89         Manager ma1 = new Manager("97001", 240000, 5000);
90         Management m = new Management();
91         m.put("96001", sw1);
92         m.put("96002", hw1);
93         m.put("97001", ma1);
94         //抓取錯誤訊息
95         try {m.totalSalary();
96         } catch (limex e) {System.out.println("Total salary exceed limit:"+e.getAm());
97         }
98     }
99 }
```

# 員工薪資制度

190

- 執行結果參考畫面：

```
Total salary exceed limit: 56000.0
```