

# 類別與物件

TYIC 桃高資訊社

# 物件導向

物件導向程式設計(Object-oriented programming，簡稱 OOP)

是指使用物件(object)的程式設計模式

而物件就是類別(class)的實例(instance)

類別定義了欄位(field)和方法(method)

物件則真正擁有這些東西

且每個物件都是獨立的，互不相干

物件導向的三大特性：封裝、繼承、多型

# 類別

類別定義方式如右，名稱建議使用大駝峰命名法  
也可以在前方加上  
存取修飾子(**Access Modifier**)中的 **public**  
表示公開的

```
class 類別名稱 {  
    欄位...  
    方法...  
}  
java
```

一個檔案中  
可以有多個**頂級(top level)**類別  
但只能有一個**公開頂級類別**  
且**公開頂級類別**的名稱要和檔名一致

```
public class 類別名稱 {  
    欄位...  
    方法...  
}  
java
```

# 類別

```
class 類別名稱 {  
    資料型別 欄位名稱;  
    資料型別 欄位名稱 = 值;  
  
    存取修飾子 final static 資料型別 欄位名稱;  
    存取修飾子 final static 資料型別 欄位名稱 = 值;  
  
    返回值型別 方法名稱(參數型別1 參數名稱1, 參數型別2 參數名稱2, ...) {  
        陳述式...  
    }  
  
    存取修飾子 static 返回值型別 方法名稱(參數型別1 參數名稱1, 參數型別2 參數名稱2, ...) {  
        陳述式...  
    }  
}
```

java

# 動態與靜態

沒有 `static` 表示是動態的，有 `static` 表示是靜態的

而兩者的區別在於：

動態的在被使用時才會分配記憶體(`memory`)

而靜態的則是在類別被載入(`load`)時就分配記憶體

動態成員需要透過物件來存取

而靜態成員則須透過類別來存取

# 靜態方法與靜態欄位

要存取類別或物件的成員

須使用 `"."`（存取運算子，`access operator`）

呼叫靜態方法：`類別名稱.靜態方法名稱(引數1, 引數2, ...)` java

存取靜態欄位：`類別名稱.靜態欄位名稱` java

若存取的靜態方法或靜態欄位與當前屬同類別

且作用域中沒有其他的同名方法或變數，則可省略類別名稱

# 靜態方法與靜態欄位

```
import java.util.Scanner;

public class Main {
    static void printIsPrime(int number) {
        if (Util.isPrime(number)) {
            System.out.printf("%d is prime%n",
                              number);
            return;
        }
        System.out.printf("%d is not prime%n",
                          number);
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int base = scanner.nextInt();
        int power = scanner.nextInt();
        printIsPrime(base);
        printIsPrime(power);
        System.out.printf("%d ^ %d = %d", base,
                          power, Util.pow(base, power));
    }
}
```

```
class Util {
    static boolean isPrime(int number) {
        for (int i = 2; i * i <= number; i++) {
            if (number % i == 0) return false;
        }
        return true;
    }

    static int pow(int base, int power) {
        int result = 1;
        if (power >= 0) {
            for (int i = 0; i < power; i++) {
                result *= base;
            }
            return result;
        }
        for (int i = -power; i > 0; i--) {
            result *= base;
        }
        return 1 / result;
    }
}
```



java

```
2 4
2 is prime
4 is not prime
2 ^ 4 = 16 console
```

# 靜態區塊

若想在類別被載入時執行某些程式碼，可以使用靜態區塊：

```
public class Main {  
    public static void main(String[] args) {  
        System.out.println("載入 SayHello 類別前");  
        SayHello.wow(); // 載入 SayHello 類別  
        System.out.println("載入 SayHello 類別後");  
    }  
}  
  
class SayHello {  
    static {  
        System.out.println("Hello");  
    }  
  
    static void wow() {  
    }  
}
```

```
載入 SayHello 類別前  
Hello  
載入 SayHello 類別後  output
```



# 物件

可以使用 **new** 運算子創建特定類別的實例(物件)

**new** 運算可為表達式或表達陳述式

**new** 類別名稱(**args**)

java

要存取物件的成員，須使用 **"."**(存取運算子，**access operator**)

物件.成員

java

```
public class Main {  
    public static void main(String[] args) {  
        int i = 1;  
        final String PREFIX = "喜歡你的第";  
        final String SUFFIX = new String("年，我還是沒告白");  
        System.out.println(new StringBuilder().append(PREFIX).append(i++).append("年，我還沒有告白"));  
        System.out.println(new StringBuilder().append(PREFIX).append(i++).append(SUFFIX));  
        System.out.println(new StringBuilder().append(PREFIX).append(i++).append(SUFFIX));  
        System.out.println(new StringBuilder().append(PREFIX).append(i++).append(SUFFIX));  
        System.out.println(new StringBuilder().append(PREFIX).append(i++).append(SUFFIX));  
        System.out.println(new StringBuilder().append(PREFIX).append(i++).append("年，我終於告白了"));  
    }  
}
```

創建物件

呼叫方法



output

java

# 物件

呼叫動態方法：`物件.動態方法名稱(引數1, 引數2, ...)` java

存取動態欄位：`物件.動態欄位名稱` java

若存取的動態成員與當前屬同類別

且作用域中沒有其他的同名方法或變數，則可省略物件

```
public class Main {  
    public static void main(String[] args) {  
        Person person1 = new Person();  
        person1.age = 60;  
        person1.name = "任嫌齊";  
        System.out.println("person1 創建完成");  
        person1.printInfo();  
        Person person2 = new Person();  
        person2.age = 65;  
        person2.name = "李宗聖";  
        System.out.println("person2 創建完成");  
        person1.printInfo();  
        person2.printInfo();  
    }  
}
```

```
class Person {  
    int age = 0;  
    String name;  
  
    void printInfo() {  
        System.out.printf("姓名 : %s 年齡 : %d %n",  
            name, age);  
    }  
}
```

```
person1 創建完成  
姓名 : 任嫌齊 年齡 : 60  
person2 創建完成  
姓名 : 任嫌齊 年齡 : 60  
姓名 : 李宗聖 年齡 : 65
```

output

java



# this

若存取的動態欄位與當前屬同類別，但作用域中有其他同名變數，則必須使用以下方式來指定存取動態欄位，否則會存取同名變數。

`this.動態欄位名稱`

java

其中 "this" 所代表的就是當前這個物件

```
public class Main {  
    public static void main(String[] args) {  
        Person person1 = new Person();  
        person1.setAge(60);  
        person1.name = "任嫌齊";  
        person1.printInfo();  
        Person person2 = new Person();  
        person2.setAge(-65);  
        person2.name = "李宗聖";  
        person2.printInfo();  
    }  
}
```

姓名：任嫌齊 年齡：60  
姓名：李宗聖 年齡：0

output

```
class Person {  
    int age = 0;  
    String name;  
  
    void printInfo() {  
        System.out.printf("姓名：%s 年齡：%d %n",  
            name, age);  
    }  
  
    void setAge(int age) {  
        if (age < 0) age = 0;  
        this.age = age;  
    }  
}
```



java

# 建構子

建構子(**constructor**)是一種特殊的動態方法

方法名稱與類別名稱完全相同，而且不需要返回型別及返回值


會在創建物件時被呼叫

若沒有定義建構子

則編譯器會補上無參數建構子

```
public class Main {  
    public static void main(String[] args) {  
        Person person1 = new Person(60, "任嫌齊");  
        person1.printInfo();  
        Person person2 = new Person(-65, "李宗聖");  
        person2.printInfo();  
    }  
}
```

姓名：任嫌齊	年齡：60	output
姓名：李宗聖	年齡：0	



```
class Person {  
    int age = 0;  
    String name;  
  
    Person(int age, String name) {  
        setAge(age);  
        this.name = name;  
    }  
  
    void printInfo() {  
        System.out.printf("姓名：%s 年齡：%d %n",  
            name, age);  
    }  
  
    void setAge(int age) {  
        if (age < 0) age = 0;  
        this.age = age;  
    }  
}
```

java

# 補充：解構子

因為 Java 有垃圾回收(Garbage Collection)機制  
而且 Java 不允許手動更改記憶體  
所以 Java 中其實並沒有解構子(destructor)

# 建構子重載

建構子也可以重載

而重載的建構子內部

可以呼叫其他重載的建構子

但一定要在建構子內部的第一行

且須使用以下格式呼叫：

`this(args)`

java

```
public class Main {  
    public static void main(String[] args) {  
        Person person1 = new Person(60, "任嫌齊");  
        person1.printInfo();  
        Person person2 = new Person(-65, "李宗聖");  
        person2.printInfo();  
        Person person3 = new Person(30, "蔡秦", true);  
        person3.printInfo();  
    }  
}
```

```
class Person {  
    int age = 0;  
    String name;  
    boolean pregnant = false;
```

```
    Person(int age, String name) {  
        setAge(age);  
        this.name = name;  
    }
```

姓名：任嫌齊 年齡：60  
姓名：李宗聖 年齡：0  
姓名：蔡秦 年齡：30 懷孕 output

```
    Person(int age, String name, boolean pregnant) {  
        this(age, name);  
        this.pregnant = pregnant;  
    }
```

```
    void printInfo() {  
        System.out.printf("姓名：%s 年齡：%d ", name, age);  
        System.out.println(pregnant ? "懷孕" : "");  
    }
```

```
    void setAge(int age) {  
        if (age < 0) age = 0;  
        this.age = age;  
    }
```

java



# 存取修飾子 - private

存取修飾子用來進行存取權限的管理  
避免外界隨意存取提高穩定與安全性  
這稱為物件封裝(encapsulation)  
存取修飾子有 public、protected  
、(無存取修飾子)、private  
在此只先介紹 private  
表示私有的，外界完全無法存取


```
public class Main {  
    public static void main(String[] args) {  
        Person person1 = new Person(35, "蔡秦", true);  
        person1.printInfo();  
        person1.age = -35;  
        System.out.println(person1.age);  
        //build failed: age has private access in Person  
    }  
}
```

```
class Person {  
    private int age = 0;  
    String name;  
    boolean pregnant = false;  
  
    Person(int age, String name) {  
        setAge(age);  
        this.name = name;  
    }  
  
    Person(int age, String name, boolean pregnant) {  
        this(age, name);  
        this.pregnant = pregnant;  
    }  
  
    void printInfo() {  
        System.out.printf("姓名:%s 年齡:%d ", name, age);  
        System.out.println(pregnant ? "懷孕" : "");  
    }  
  
    void setAge(int age) {  
        if (age < 0) age = 0;  
        this.age = age;  
    }  
}
```

# getter 與 setter

將 **Person** 的 **age** 設為 **private**  
使得外界無法存，但同時也無法取  
若要解決此問題，就需要通過  
非 **private** 的方法來存取 **age**  
而這個方法的名稱  
通常叫做 "**getXxx**" 或 "**setXxx**"  
其中 "**Xxx**" 為欄位名稱

```
public class Main {  
    public static void main(String[] args) {  
        Person person1 = new Person(35, "蔡秦", true);  
        person1.printInfo();  
        person1.setAge(-35);  
        System.out.println(person1.getAge());  
    }  
}
```



```
class Person {  
    private int age = 0;  
    String name;  
    boolean pregnant = false;  
  
    Person(int age, String name) {  
        setAge(age);  
        this.name = name;  
    }  
  
    Person(int age, String name, boolean pregnant) {  
        this(age, name);  
        this.pregnant = pregnant;  
    }  
  
    void printInfo() {  
        System.out.printf("姓名:%s 年齡:%d ", name, age);  
        System.out.println(pregnant ? "懷孕" : "");  
    }  
  
    void setAge(int age) {  
        if (age < 0) age = 0;  
        this.age = age;  
    }  
  
    int getAge() {  
        return age;  
    }  
}
```

姓名：蔡秦 年齡：35 懷孕 0	output
---------------------	--------

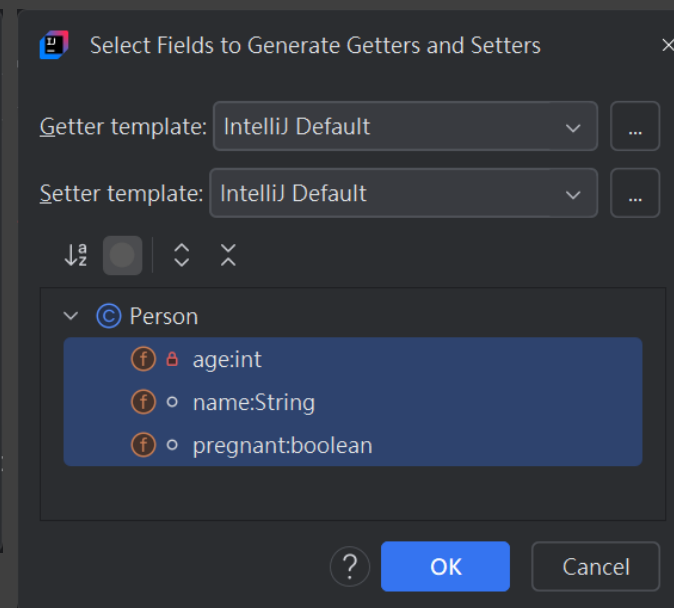
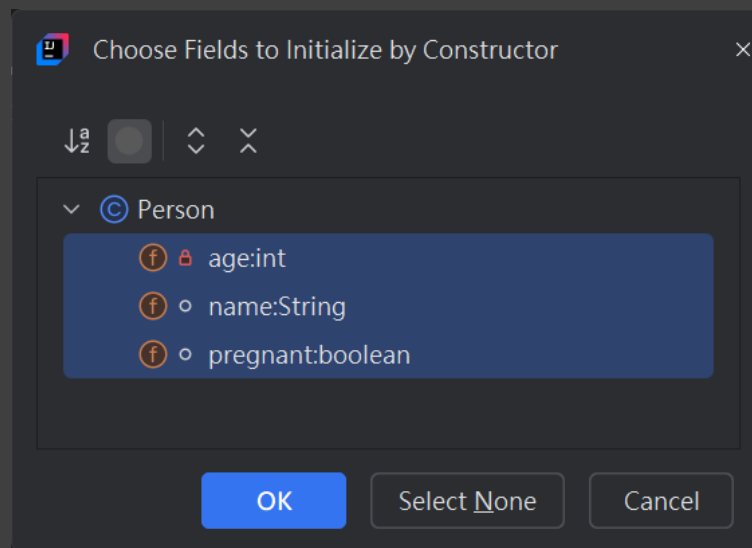
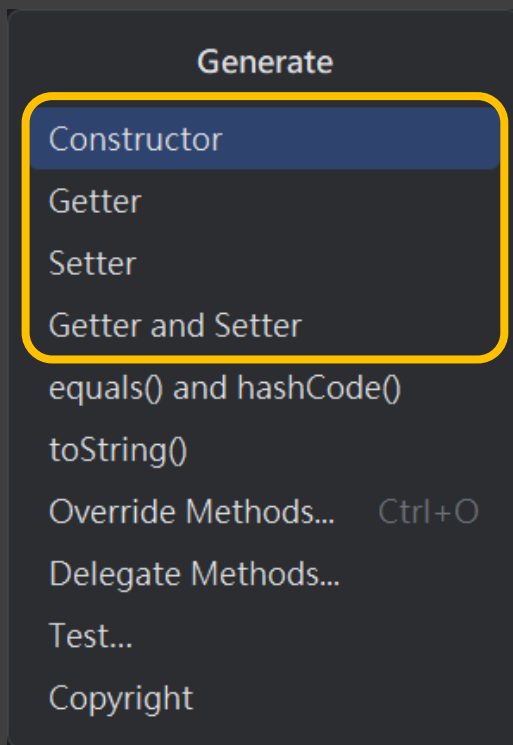
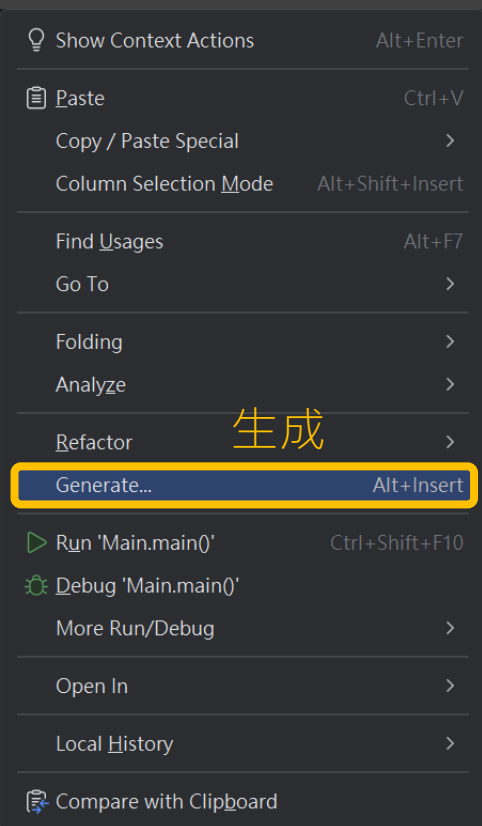
java



# IntelliJ IDEA

## 生成建構子、getter 和 setter

右鍵 -> 生成 或是 **Alt + Insert**  
便會顯示生成選單，可以選擇要生成  
建構子、getter、setter  
之後選擇要生成的欄位



# 抽象類別

在類別定義前方加上 **abstract**  
表示抽象(**abstract**)類別

```
public abstract class 類別名稱 {  
    欄位...  
    方法...  
}
```

java

```
abstract class 類別名稱 {  
    欄位...  
    方法...  
}
```

java

抽象類別不可被實例化

換言之就是不能創建物件

常常用在只有靜態方法的類別

(工具類別, **Utility Class**)

或是需要被繼承(**inherit**)的類別

```
abstract class Util {  
    static boolean isPrime(int number) {  
        for (int i = 2; i * i <= number; i++) {  
            if (number % i == 0) return false;  
        }  
        return true;  
    }  
  
    static int pow(int base, int power) {  
        int result = 1;  
        if (power >= 0) {  
            for (int i = 0; i < power; i++) {  
                result *= base;  
            }  
            return result;  
        }  
        for (int i = -power; i > 0; i--) {  
            result *= base;  
        }  
        return 1 / result;  
    }  
}
```

# 繼承

繼承(**inherit**)是指從另一個類別獲得同樣的成員  
被繼承的叫做父類別(**super class**)，繼承的叫做子類別(**subclass**)  
若子類別要繼承父類別，須使用 **extends** 關鍵字：

```
class 子類別名稱 extends 父類別名稱 {  
    欄位...  
    方法...  
}
```

java

在 **Java** 中，一個類別只能直接繼承另一個類別，稱為單一繼承  
一個類別的父(子)類別的父(子)類別，也是該類別的父(子)類別  
如果沒有繼承其他類別，則編譯器會自動繼承 **"Object"** 類別  
換言之，**Java** 中的所有類別皆為 **"Object"** 類別的子類別

# 繼承

若父類別沒有無參數建構子  
則子類別必須在建構子中  
使用以下格式呼叫父類別建構子：

```
public class Main {  
    public static void main(String[] args) {  
        Person person = new Person(35, "蔡秦");  
        person.printInfo();  
        Worker worker = new Worker(25, "周節倫", "歌手");  
        worker.printInfo();  
        Student student = new Student(16, "白氦", 10);  
        student.printInfo();  
    }  
}
```

```
class Person {  
    private int age = 0;  
    String name;  
  
    Person(int age, String name) {  
        setAge(age);  
        this.name = name;  
    }  
  
    void printInfo() {  
        System.out.printf("姓名:%s 年齡:%d %n", name, age);  
    }  
  
    void setAge(int age) {  
        if (age < 0) age = 0;  
        this.age = age;  
    }  
}
```

姓名：蔡秦 年齡：35  
姓名：周節倫 年齡：25  
姓名：白氦 年齡：16

output

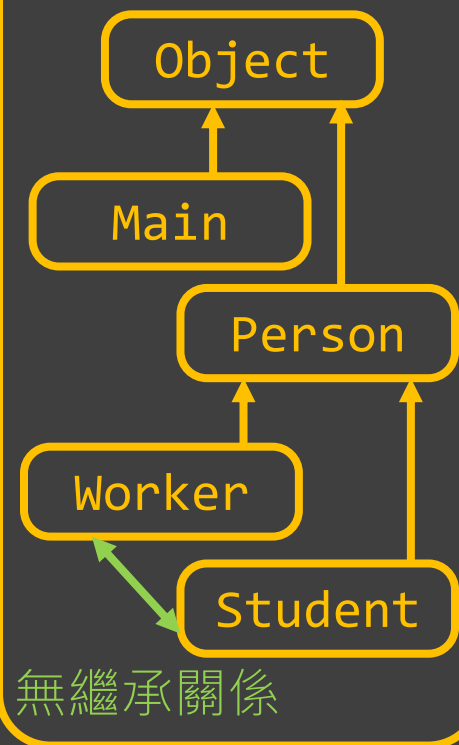
`super(args)`

java

```
class Worker extends Person {  
    String occupation;  
  
    Worker(int age, String name) {  
        super(age, name);  
    }  
  
    Worker(int age, String name, String occupation) {  
        this(age, name);  
        this.occupation = occupation;  
    }  
}  
  
class Student extends Person {  
    int grade;  
  
    Student(int age, String name) {  
        super(age, name);  
    }  
  
    Student(int age, String name, int grade) {  
        this(age, name);  
        this.grade = grade;  
    }  
}
```



繼承關係圖



# 覆寫

覆寫(**override**)是指將父類別的動態方法覆蓋掉  
若呼叫子類別的該方法時，會執行覆寫過的方法，覆寫方法的格式如下：

```
class 父類別名稱 {  
    相同返回值型別 方法名稱(相同參數型別1 參數名稱1, 相同參數型別2 參數名稱2, ...) {  
        陳述式...  
    }  
}  
  
class 子類別名稱 extends 父類別名稱{  
    @Override  
    相同返回值型別 方法名稱(相同參數型別1 參數名稱1, 相同參數型別2 參數名稱2, ...) {  
        陳述式...  
    }  
}
```

java

在 Java 中，"@" 開頭的是一種特殊的註解，能讓編譯器檢查，建議加上  
"@Override" 表示該動態方法覆寫了父類別的動態方法

# 覆寫



```
public class Main {
    public static void main(String[] args) {
        Person person = new Person(35, "蔡秦");
        person.printInfo();
        Worker worker = new Worker(25, "周節倫", "歌手");
        worker.printInfo();
        Student student = new Student(16, "白氫", 10);
        student.printInfo();
    }
}

class Person {
    private int age = 0;
    String name;

    Person(int age, String name) {
        setAge(age);
        this.name = name;
    }

    void printInfo() {
        System.out.printf("姓名 : %s 年齡 : %d %n", name, age);
    }

    void setAge(int age) {
        if (age < 0) age = 0;
        this.age = age;
    }

    int getAge() {
        return age;
    }
}
```

```
class Worker extends Person {
    String occupation;

    Worker(int age, String name) {
        super(age, name);
    }

    Worker(int age, String name, String occupation) {
        this(age, name);
        this.occupation = occupation;
    }

    @Override
    void printInfo() {
        System.out.printf("姓名 : %s 年齡 : %d 職業 : %s %n", name, getAge(), occupation);
    }
}

class Student extends Person {
    int grade;

    Student(int age, String name) {
        super(age, name);
    }

    Student(int age, String name, int grade) {
        this(age, name);
        this.grade = grade;
    }

    @Override
    void printInfo() {
        System.out.printf("姓名 : %s 年齡 : %d 年級 : %d %n", name, getAge(), grade);
    }
}
```

姓名 : 蔡秦 年齡 : 35  
姓名 : 周節倫 年齡 : 25 職業 : 歌手  
姓名 : 白氫 年齡 : 16 年級 : 10      output

# 存取修飾子 - protected

從上個範例中可以看到，子類別也不能存取私有成員

而要讓子類別也可以存取成員，就必須使用 **protected**

```
public class Main {  
    public static void main(String[] args) {  
        Person person = new Person(35, "蔡秦");  
        person.printInfo();  
        Worker worker = new Worker(25, "周節倫", "歌手");  
        worker.printInfo();  
        Student student = new Student(16, "白氦", 10);  
        student.printInfo();  
    }  
}  
  
class Person {  
    protected int age = 0;  
    String name;  
  
    Person(int age, String name) {  
        setAge(age);  
        this.name = name;  
    }  
  
    void printInfo() {  
        System.out.printf("姓名 : %s 年齡 : %d %n", name, age);  
    }  
  
    void setAge(int age) {  
        if (age < 0) age = 0;  
        this.age = age;  
    }  
}
```

姓名：蔡秦 年齡：35  
姓名：周節倫 年齡：25 職業：歌手  
姓名：白氦 年齡：16 年級：10 output

```
class Worker extends Person {  
    String occupation;  
  
    Worker(int age, String name) {  
        super(age, name);  
    }  
  
    Worker(int age, String name, String occupation) {  
        this(age, name);  
        this.occupation = occupation;  
    }  
  
    @Override  
    void printInfo() {  
        System.out.printf("姓名 : %s 年齡 : %d 職業 : %s %n", name, age, occupation);  
    }  
}  
  
class Student extends Person {  
    int grade;  
  
    Student(int age, String name) {  
        super(age, name);  
    }  
  
    Student(int age, String name, int grade) {  
        this(age, name);  
        this.grade = grade;  
    }  
  
    @Override  
    void printInfo() {  
        System.out.printf("姓名 : %s 年齡 : %d 年級 : %d %n", name, age, grade);  
    }  
}
```



java

# super

要存取父類別未覆寫的方法，須使用以下格式：

**super.動態方法名稱**

java

```
public class Main {
    public static void main(String[] args) {
        Person person = new Person(10, "蔡秦");
        person.printInfo();
        Worker worker = new Worker(10, "周節倫", "歌手");
        worker.printInfo();
    }
}

class Person {
    private int age = 0;
    String name;

    Person(int age, String name) {
        setAge(age);
        this.name = name;
    }

    void printInfo() {
        System.out.printf("姓名 : %s 年齡 : %d %n", name, age);
    }

    void setAge(int age) {
        if (age < 0) age = 0;
        this.age = age;
    }

    public int getAge() {
        return age;
    }
}
```

```
class Worker extends Person {
    String occupation;

    Worker(int age, String name) {
        super(age, name);
    }

    Worker(int age, String name, String occupation) {
        this(age, name);
        this.occupation = occupation;
    }

    @Override
    void printInfo() {
        System.out.printf("姓名 : %s 年齡 : %d 職業 : %s %n",
            name, getAge(), occupation);
    }

    @Override
    void setAge(int age) {
        if (age < 15) age = 15;
        super.setAge(age);
    }
}
```



姓名：蔡秦 年齡：10  
姓名：周節倫 年齡：15 職業：歌手    output

java



# 抽象方法

在抽象類別中，動態方法  
定義前方加上 **abstract**  
表示抽象方法

```
abstract class 類別名稱 {  
    abstract 返回值型別 方法名稱(...);  
}
```

java

抽象方法在該抽象類別中  
不可以定義方法的執行內容  
以分號結尾  
且抽象類別的子類別  
一定要覆寫該抽象方法

```
public class Main {  
    public static void main(String[] args) {  
        Person person = new Person(10, "蔡秦");  
        person.printInfo();  
        Whale whale = new Whale(10, 5);  
        whale.printInfo();  
    }  
}
```

姓名：蔡秦 年齡：10  
長度：10 寬度：5      output

```
abstract class Animal {  
    abstract void printInfo();  
}
```

```
class Whale extends Animal {  
    private int length = 0;  
    private int width = 0;  
  
    Whale(int length, int width) {  
        this.length = length;  
        this.width = width;  
    }  
  
    @Override  
    void printInfo() {  
        System.out.printf(  
            "長度：%d 寬度：%d %n",  
            length, width);  
    }  
}
```

```
class Person extends Animal {  
    private int age = 0;  
    String name;  
  
    Person(int age, String name) {  
        this.age = age;  
        this.name = name;  
    }  
  
    @Override  
    void printInfo() {  
        System.out.printf(  
            "姓名：%s 年齡：%d %n",  
            name, age);  
    }  
}
```



# 多型

多型(polymorphism)是指  
通過同一套方式操作不同種類的物件



```
public class Main {
    public static void main(String[] args) {
        Person person1 = new Person(35, "蔡秦");
        person1.printInfo();
        Person person2 = new Worker(25, "周節倫", "歌手");
        person2.printInfo();
        Person person3 = new Student(16, "白氫", 10);
        person3.printInfo();
    }
}

class Person {
    private int age = 0;
    String name;

    Person(int age, String name) {
        setAge(age);
        this.name = name;
    }

    void printInfo() {
        System.out.printf("姓名:%s 年齡:%d %n", name, age);
    }

    void setAge(int age) {
        if (age < 0) age = 0;
        this.age = age;
    }

    int getAge() {
        return age;
    }
}
```

Worker  
和 Student  
繼承 Person  
所以前二者的實例  
一定是後者的實例

```
class Worker extends Person {
    String occupation;

    Worker(int age, String name) {
        super(age, name);
    }

    Worker(int age, String name, String occupation) {
        this(age, name);
        this.occupation = occupation;
    }

    @Override
    void printInfo() {
        System.out.printf("姓名:%s 年齡:%d 職業:%s %n", name, getAge(), occupation);
    }
}

class Student extends Person {
    int grade;

    Student(int age, String name) {
        super(age, name);
    }

    Student(int age, String name, int grade) {
        this(age, name);
        this.grade = grade;
    }

    @Override
    void printInfo() {
        System.out.printf("姓名:%s 年齡:%d 年級:%d %n", name, getAge(), grade);
    }
}
```

姓名：蔡秦 年齡：35  
姓名：周節倫 年齡：25 職業：歌手  
姓名：白氫 年齡：16 年級：10      output

# 多型

下方程式碼定義了  
靜態方法 `printInfo`  
讓傳入的物件呼叫  
該物件的 `printInfo` 方法

```
public class Main {
    public static void main(String[] args) {
        Animal.printInfo(new Person(10, "蔡秦"));
        Animal.printInfo(new Whale(10, 5));
    }
}

abstract class Animal {
    abstract void printInfo();

    static void printInfo(Person person) {
        person.printInfo();
    }

    static void printInfo(Whale whale) {
        whale.printInfo();
    }
}
```

```
class Whale extends Animal {
    private int length = 0;
    private int width = 0;

    Whale(int length, int width) {
        this.length = length;
        this.width = width;
    }

    @Override
    void printInfo() {
        System.out.printf(
            "長度:%d 寬度:%d %n",
            length, width);
    }
}

class Person extends Animal {
    private int age = 0;
    String name;

    Person(int age, String name) {
        this.age = age;
        this.name = name;
    }

    @Override
    void printInfo() {
        System.out.printf(
            "姓名:%s 年齡:%d %n",
            name, age);
    }
}
```

java

程式碼中為  
每個 `Animal` 的子類別  
重載 `printInfo` 方法  
考慮 `Animal`  
有更多的子類別  
則重載方法的做法  
不切實際  
可使用多型解決此問題  
修改後的程式碼如下：

```
abstract class Animal {
    abstract void printInfo();

    static void printInfo(Animal animal) {
        animal.printInfo();
    }
}
```

java

姓名：蔡秦 年齡：10  
長度：10 寬度：5

output

# instanceof

`instanceof` 是用來判斷物件是否是某個類別的實例  
會返回一個 `boolean` 值，經常與 `if` 搭配，用法如下：

```
物件 instanceof 比較類別
```

java

```
物件 instanceof 比較類別 變數
```

java

第二種寫法中

若物件的類別是比較類別的父類，且物件是類別的實例  
則變數會初始化成轉型成比較類別的物件，否則變數無法使用

要特別注意的是，使用 `instanceof`

需要物件的類別和比較類別有繼承關係，否則會報錯

# 物件轉型

物件轉型(Cast)是將物件的型別強制轉為其他型別

只能在確保前者的實際型別和後者相同的情況下才能轉型，否則會報錯

(欲轉換型別) 物件

java

```
public class Main {  
    public static void main(String[] args) {  
        Animal animal1 = new Cat("小貓", 2);  
        Animal.printInfo(animal1);  
        Animal animal2 = new Dog("小狗", 5);  
        Animal.printInfo(animal2);  
    }  
}
```

```
class Dog extends Animal {  
  
    Dog(String name, int age) {  
        super(name, age);  
    }  
  
    void bark() {  
        System.out.println("汪!");  
    }  
}
```

```
class Cat extends Animal {  
  
    Cat(String name, int age) {  
        super(name, age);  
    }  
  
    void meow() {  
        System.out.println("喵!");  
    }  
}
```

名稱：小貓 年齡：2  
喵！  
名稱：小狗 年齡：5  
汪！  
output

```
abstract class Animal {  
    String name;  
    int age;  
  
    Animal(String name, int age) {  
        this.name = name;  
        this.age = age;  
    }  
  
    void printInfo() {  
        System.out.printf("名稱：%s 年齡：%d %n", name, age);  
    }  
  
    static void printInfo(Animal animal) {  
        animal.printInfo();  
        makeSound(animal);  
    }  
  
    static void makeSound(Animal animal) {  
        if (animal instanceof Cat) {  
            ((Cat) animal).meow();  
        } else if (animal instanceof Dog dog) {  
            dog.bark();  
        }  
    }  
}
```



java

# Object 類別

下方為 **Object** 類別的部分程式碼

其中 **native** 表示  
該方法是由其他語言實現

**"equals"** 方法

是在比較物件時會呼叫

**"hashCode"** 方法

是在某些特殊的集合呼叫

**"toString"** 方法是在

字串串接或輸出時呼叫

```
public class Object {  
    ...  
  
    @IntrinsicCandidate  
    public Object() {}  
  
    @IntrinsicCandidate  
    public final native Class<?> getClass();  
  
    @IntrinsicCandidate  
    public native int hashCode();  
  
    public boolean equals(java.lang.Object obj) {  
        return (this == obj);  
    }  
  
    public String toString() {  
        return getClass().getName() + "@" + Integer.toHexString(hashCode());  
    }  
  
    ...  
}
```

# 物件比較

若要比較兩個物件是否相等

一定不可以使用比較運算子(==)來進行比較

而是要呼叫物件的 "equals" 方法

因為比較運算子用在物件上時，是比較兩個物件是否為同一個實例

但顯然的，我們要比較的是物件的內容是否相同

所以必須覆寫 "equals" 方法，然後在要比較物件時呼叫


特別注意，覆寫 "equals" 方法時也要覆寫 "hashCode" 方法

# 物件比較

`equals` 和 `hashCode` 覆寫  
可以通過 IDEA 自動生成

其中 "`getClass`" 方法  
會返回 "`Class`" 類別實例  
且同類別物件返回同個實例

```
public class Main {  
    public static void main(String[] args) {  
        Person person1 = new Person(35, "蔡秦");  
        System.out.println(person1);  
        Person person2 = new Person(35, "蔡秦");  
        System.out.println(person2);  
        System.out.println(person1 == person2);  
        System.out.println(person1.equals(person2));  
    }  
}
```



```
class Person {  
    protected int age = 0;  
    String name;  
  
    Person(int age, String name) {  
        this.age = age;  
        this.name = name;  
    }  
  
    @Override  
    public String toString() {  
        return "姓名 : %s 年齡 : %d %n".formatted(name, age);  
    }  
  
    @Override  
    public boolean equals(Object o) {  
        if (this == o) return true;  
        if (o == null || getClass() != o.getClass()) return false;  
        Person person = (Person) o;  
        return age == person.age && Objects.equals(name, person.name);  
    }  
  
    @Override  
    public int hashCode() {  
        return Objects.hash(age, name);  
    }  
}
```

姓名 : 蔡秦 年齡 : 35  
姓名 : 蔡秦 年齡 : 35  
false  
true

output

java



# 內部類別

內部類別(`inner class`)有三種：

成員內部類別(`member inner class`)、

區域內部類別(`local inner class`)、

匿名內部類別(`anonymous inner class`)

成員內部類別就是在類別中定義類別

用法與其他成員完全相同，可加 `static` 或存取修飾子

區域內部類別就是在方法中定義類別

匿名內部類別就是在創建實例時才定義類別

是個全新的類別，但沒有名稱，並且會繼承已存在類別

```
new 已存在類別(引數...) {  
    匿名內部類別定義...  
}
```

java

# 內部類別


```
public class Main {
    public static void main(String[] args) {
        class Person extends Animal {
            int height;

            Person(String name, int age, int height) {
                super(name, age);
                this.height = height;
            }

            void printInfo() {
                System.out.printf("名稱 : %s 年齡 : %d 身高 : %d %n", name, age, height);
            }
        }

        Animal animal1 = new Animal.Cat("小貓", 2);
        Animal.printInfo(animal1);
        Animal animal2 = new Animal.Dog("小狗", 5);
        Animal.printInfo(animal2);
        Animal animal3 = new Person("小人", 10, 140);
        Animal.printInfo(animal3);
        Animal animal4 = new Animal("小鯨", 7) {
            @Override
            void printInfo() {
                System.out.printf("名稱 : %s 年齡 : 我的年齡是祕密 ! %n", name);
            }
        };
        Animal.printInfo(animal4);
    }
}
```

```
名稱：小貓 年齡：2
喵！
名稱：小狗 年齡：5
汪！
名稱：小人 年齡：10 身高：140
名稱：小鯨 年齡：我的年齡是祕密！ output
```



```
abstract class Animal {
    String name;
    int age;

    Animal(String name, int age) {
        this.name = name;
        this.age = age;
    }

    void printInfo() {
        System.out.printf("名稱 : %s 年齡 : %d %n", name, age);
    }

    static void printInfo(Animal animal) {
        animal.printInfo();
        makeSound(animal);
    }

    static void makeSound(Animal animal) {
        if (animal instanceof Cat) {
            ((Cat) animal).meow();
        } else if (animal instanceof Dog dog) {
            dog.bark();
        }
    }

    static class Cat extends Animal {

        Cat(String name, int age) {
            super(name, age);
        }

        void meow() {
            System.out.println("喵！");
        }
    }

    static class Dog extends Animal {

        Dog(String name, int age) {
            super(name, age);
        }

        void bark() {
            System.out.println("汪！");
        }
    }
}
```

# 不可繼承類別

在類別定義前方加上 **final**  
表示該類別是不可繼承類別  
也就是該類別不能被繼承  
這通常在

```
修飾子 final class 類別名稱 {  
    欄位...  
    方法...  
    建構子...  
    類別...  
}
```

java

**API(Application Programming Interface**，應用程式介面)  
中才會出現

用途是防止使用該 **API** 的人亂繼承

常見的 **final** 類別有：**java.lang.String**、包裝類別

# JavaBeans

**JavaBean** 是指一個類別：

1. 具有所有欄位的公開 **getter & setter**
2. 具有公開無參數建構子
3. 可序列化(可以變成位元組串流，儲存在各種能存東西的地方)

**JavaBean** 常常用來作為資料載體，傳遞資料

但也因為上面的三點限制，導致 **JavaBean** 常常定義過長

# JavaBeans

```
public class Main {
    public static void main(String[] args) {
        Person person =
            new Person(new PersonData("徐懷豫", 30, 160, 40));
        System.out.println(person);
    }
}

class Person {
    PersonData data;

    Person(PersonData data) {
        this.data = data;
    }

    @Override
    public String toString() {
        return "Person{" +
            "name=" + data.getName() +
            ",age=" + data.getAge() +
            ",height=" + data.getHeight() +
            ",weight=" + data.getWeight() +
            '}';
    }
}
```



```
class PersonData {
    private String name;
    private int age;
    private int height;
    private int weight;

    public PersonData() {
    }

    public PersonData(String name, int age, int height, int weight) {
        this.name = name;
        this.age = age;
        this.height = height;
        this.weight = weight;
    }

    public String getName() {
        return name;
    }

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

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }

    public int getHeight() {
        return height;
    }

    public void setHeight(int height) {
        this.height = height;
    }

    public int getWeight() {
        return weight;
    }

    public void setWeight(int weight) {
        this.weight = weight;
    }
}
```

# 資料載體類別

從上個範例中可以看到，我們為了要存取幾個資料，而定義超級長的類別  
此時便可以考慮資料載體類別，能減少很多不必要的程式碼

```
修飾子 record 資料載體類別名稱(資料型別1 資料名稱1, 資料型別2 資料名稱2, ..., 資料型別n 資料名稱n) {  
    靜態欄位...  
    方法...  
    建構子...  
    類別...  
}
```

java

資料載體類別是個特殊的類別：

1. 資料為私有不可變動態欄位，且有與資料名稱同名公開方法供讀取資料
2. 資料載體類別不可定義額外的動態欄位
3. 資料載體類別為不可繼承類別，也不可以繼承類別，但可實作介面
4. 資料載體類別必定帶有一個建構子，且該建構子的參數即為全部資料
5. 資料載體類別的多載建構子，必須呼叫參數為全部資料的建構子

# 資料載體類別

若要定義參數為全部資料的建構子，不須寫參數

且該建構子後方會被編譯器加上資料賦值的程式碼

資料載體類別預設覆寫了 `equals`、`toString`、`hashCode` 等方法

```
public class Main {
    public static void main(String[] args) {
        Person person1 =
            new Person(new PersonData("徐懷豫", 30, 160, 40));
        Person person2 = new Person(new PersonData("芳大同"));
        System.out.println(person1);
        person2.data.print();
    }
}

class Person {
    PersonData data;

    Person(PersonData data) {
        this.data = data;
    }

    @Override
    public String toString() {
        return "Person{" +
            "name=" + data.name() + ",age=" + data.age() +
            ",height=" + data.height() + ",weight=" + data.weight() +
            '}';
    }
}
```

```
record PersonData(String name, int age, int height, int weight) {
    PersonData {
        age = Math.max(age, 0);
        height = Math.max(height, 0);
        weight = Math.max(weight, 0);
        // 編譯器會自動補上：
        // this.name = name;
        // this.age = age;
        // this.height = height;
        // this.weight = weight;
    }

    PersonData(String name) {
        this(name, 0, 0, 0);
    }

    void print() {
        System.out.println(this);
    }
}
```

```
Person{name=徐懷豫,age=30,height=160,weight=40}
PersonData[name=芳大同, age=0, height=0, weight=0]
```

output



java