介面是最簡略的規格，只有方法名稱、常數，其細節都是空白!

介面 interface

簡介

一個介面裡有規定好的方法名稱，不過沒有實作，你要用它(為了某個功能任務)，就必須用它定義好的方法名稱，並實作出方法。

介面就如同外掛程式一樣，給有需要用這個功能的 其他類別 去添加其他功能

介面就是先定義好一些抽象公開的方法，或是常數 (具有初值的變數)，讓人“外掛”使用。

外掛的方式是用 implements，與繼承的觀念類似

介面與抽象類別不一樣，不過兩者也具有一些相似的用途。

繼承：只能繼承一個父類別

介面：可以實作多個介面

簡單說介面

```
//介面就如同外掛程式一樣，給有需要用這個功能的 其他類別 去添加其他功能
//介面就是先定義好一些抽象公開的方法，或是常數 (具有初值的變數)，讓人外掛
//外掛的方式是用 implements，與繼承的觀念類似
//介面與抽象類別具有相似的用途，不過兩者使用在不同的時機
```

重點摘要:

1. interface 的 method 都一定是抽象的，只有定義名稱，沒有內容。
2. 介面常被用來做“外掛”功能來使用，例如: class MyHandler implements ActionListener{}
3. 實作 interface 要使用 implements 這個關鍵字

4. 一個類別可以 implements 多個不同的 interface (extends 只能繼承一個類別)
5. 一個 interface 不能被初始化(被 new)，只能被 implements。但是，interface 可以當作共同型別使用，讓一個陣列可以放各式各樣的子類別物件。
6. 一個 interface 可以被另一個介面繼承 (extends)，也是遵循一般類別的繼承方式。

有趣又簡單的實例

貓、狗等寵物的例子

定義 Flyable 介面

```
1  *定義 Flyable 介面
2
3  public interface Flyable{    //介面的定義
4
5      public abstract void fly();          //public abstract 可以省略不寫
   }
```

豬 Pig 實作 Flyable 介面

```
1
2  *豬 Pig 實作 Flyable 介面
3  //豬類別"外掛"一個 Flyable 介面，變成一隻"飛天豬"
4  public class Pig implements Flyable {
5      @Override
6      public void fly() {
7          System.out.println("飛天豬逍遙飛 20 公尺");
8      }
9
10     public void showMe() {
11         System.out.println("我是豬!");
12     }
```

13	}

豬 Pig 的操作

```
public static void main(String[] args) {

        Pig p1 = new Pig();
        p1.fly();
        p1.showMe();

        Flyable p2 = new Pig();   //介面可以當作型別
        p2.fly();
        //p2.showMe();    //由於 p2 不是 Pig  不可以操作 showMe()
        Pig p2a = (Pig) p2;   //向下型轉之後才可以操作 showMe()
        p2a.showMe();
        //這樣向下型轉也可以
        ((Pig) p2).showMe();

        //Flyable p3 = new Flyable();   //介面可以當作型別，但不能被 new !!!
        //可以用這種方式 new，等於宣告一個匿名類別，此匿名類別用 implements 的方式來"繼承" Flyable
        //在此匿名類別中只能 override 或是實作抽象方法，不能定義新的方法
        //p4 是一個按照此匿名類別產生的一個物件
        Flyable p4 = new Flyable() {
            @Override
            public void fly() {
                System.out.println("在內部定義一個匿名類別，此類別 implements 自 ICanFly");
            }
        }; //注意這裡有一個分號喔!
        p4.fly();

        //前述 p4 的寫法，等同於以下方式使用，但是這種方法除了可以 override 抽象方法之外，也可以定義額外新增的方法
        class NoName implements Flyable {

            @Override
```

```
            public void fly() {
                System.out.println("NoName 類別 implements 自 ICanFly");
            }
        }

        NoName p5 = new NoName();
        p5.fly();
    }
}
```

Pet 類別

```
父類別
public class Pet {
    private String name;    //instance variable, field  實體變數或欄位
    //private int age;

    public Pet(    String    name     )
    {
        this.name = name; //this  是指  "這個類別"
        System.out.printf("產生新物件:%s\n", name);
    }
    public void showMe()
    {
        System.out.printf("我是%s，大家好!\n", name);
    }
```

子類別 Cat

```
1  *子類別 Cat
2  同時繼承 與 實作介面
3  public class Cat extends Pet implements Flyable {
4
5      public Cat(String name) {
6          super(name);    //呼叫父類別的建構子
```

```
7       }
8
9       @Override
10      public void showMe() {
11          super.showMe();   //呼叫父類別的 showMe()
12          System.out.println("我是一隻貓!");
13      }
14
15      public void meow() {
16          System.out.println("喵喵!");
17      }
18
19      @Override
        public void move() {
            System.out.println("輕聲走兩步");
        }

        @Override
        public void fly() {
            System.out.println("貓飛 100 公尺");
        }

        public static void main(String[] args) {

            //(1)以 Cat 當作型別
            Cat cat = new Cat("小貓");
            cat.showMe();
            cat.move();
            cat.fly();

            //(2)以 Pet 當作型別
            Pet pet = new Cat("Kitty");
            pet.showMe();

            //向下型轉
            ((Cat) pet).fly();
            //向下型轉
```

```
        Cat pp = (Cat) pet;
        pp.showMe();

        //(3)以介面當作型別
        Flyable ff = new Cat("Kitty");
        ff.fly();

        //向下型轉
        ((Cat) ff).showMe();

        //向下型轉
        Cat cc = (Cat) ff;
        cc.showMe();
    }
}
```

子類別 Dog

```
1  *子類別 Dog
2  同時繼承 與 實作介面
3  public class Dog extends Pet implements Flyable {
4
5      public Dog(String name) {
6          super(name);   //呼叫父類別的建構子
7      }
8
9      @Override      //取代父類別的 showMe()，重新定義這個方法所執行的指令
10     public void showMe() {
11         super.showMe();   //呼叫父類別的 showMe()
12         System.out.println("我是一隻狗!");
13     }
14
15     public void bark() //定義屬於 Dog 類別專屬的方法 bark()
16     {
17         System.out.println("汪汪!");
```

```
18      }
19
        //實作 move()抽象方法
        @Override
        public void move() {
            System.out.println("我大跳 5 公尺");
        }

        @Override
        public void fly() {
            System.out.println("狗飛 100 公尺");
        }

        public static void main(String[] args) {
            Dog dog = new Dog("小黑");
            dog.showMe(); //呼叫的 showMe()是在 Dog 類別中重新定義的 showMe()喔!
        }
    }
```

寫一個可以飛天的動物園測試類別

```
1   *寫一個可以飛天的動物園測試類別
2   public class FlyablePetZoo {
3
4       public static void main(String[] args) {
5
6           Flyable[] pets = new Flyable[4];//存放 可飛 寵物物件的 陣列 可以放 Dog 物
7   件 也可以放 Cat 物件
8
9           pets[0] = new Cat("小花");
10          pets[1] = new Cat("Kitty");
11          pets[2] = new Dog("小黑");
12          pets[3] = new Dog("小白");
13
```

```
14
15              for (int i=0; i< pets.length; i++)
16              {
17
18                  //可飛物件可以呼叫 fly()
19                  pets[i].fly();
20
21                  //以下者兩行不能用?為甚麼?
22                  //pets[i].showMe();
23                  //pets[i].move();
24
25                  //可飛物件向下型轉為寵物 Pet 物件
26                  Pet p = (Pet)pets[i];
27                 //寵物物件各自呼叫 showMe()   move()
28                  p.move();
29                  p.showMe();

                    //也可以這樣向下型轉 注意：小括弧有兩個、還有括弧位置
                    ((Pet)pets[i]).move();
                    ((Pet)pets[i]).showMe();



                    //貓叫   狗叫   因為有各自的 meow()與 bark()方法所以必須向下型轉
                    if (pets[i] instanceof Cat)
                    {
                        Cat c = (Cat)pets[i];
                        c.meow();
                    } else if (pets[i] instanceof Dog)
                    {
                        Dog d = (Dog)pets[i];
                        d.bark();
                    }
                }
            }
        }
```

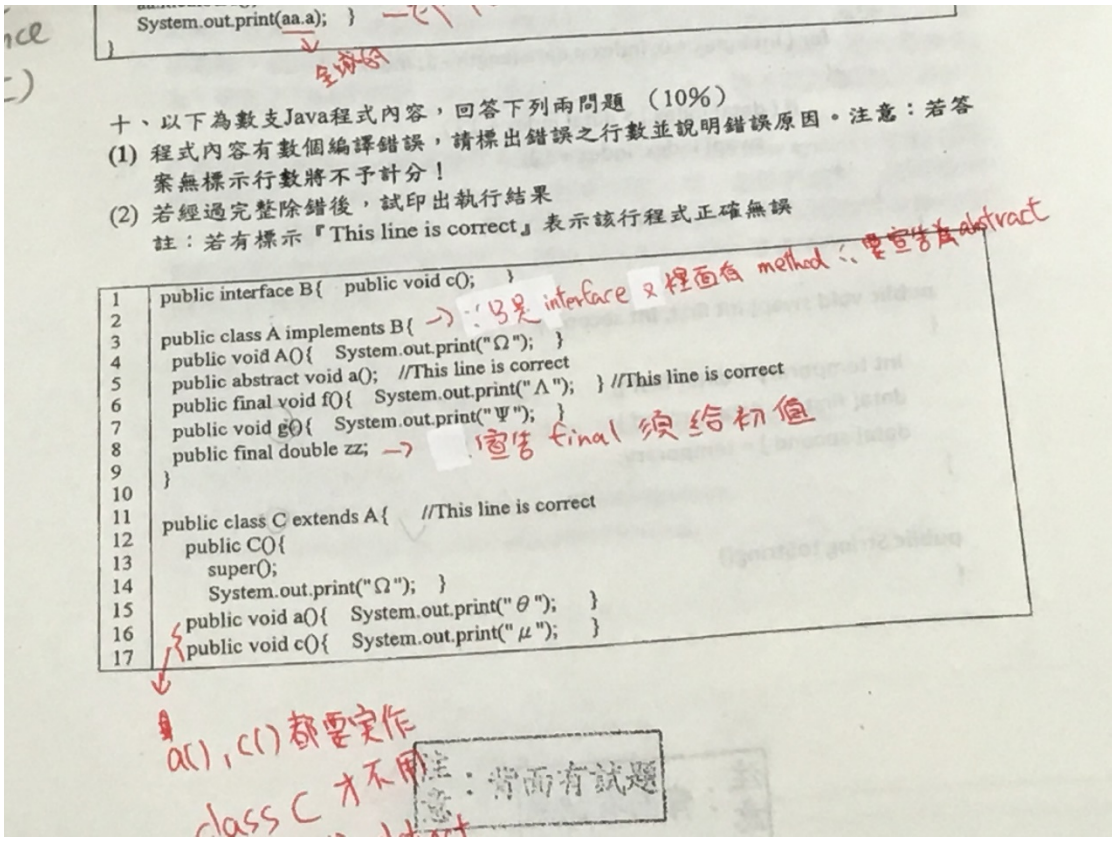
怎樣?搞得你昏頭轉向的，別氣餒，多寫程式去測試，你就會熟練。研究所入學

考試最喜歡考這種複雜觀念的題目!!

習題作業

研究所入學考題

(國立中央大學資訊管理研究所)



System.out.print(aa.a); }

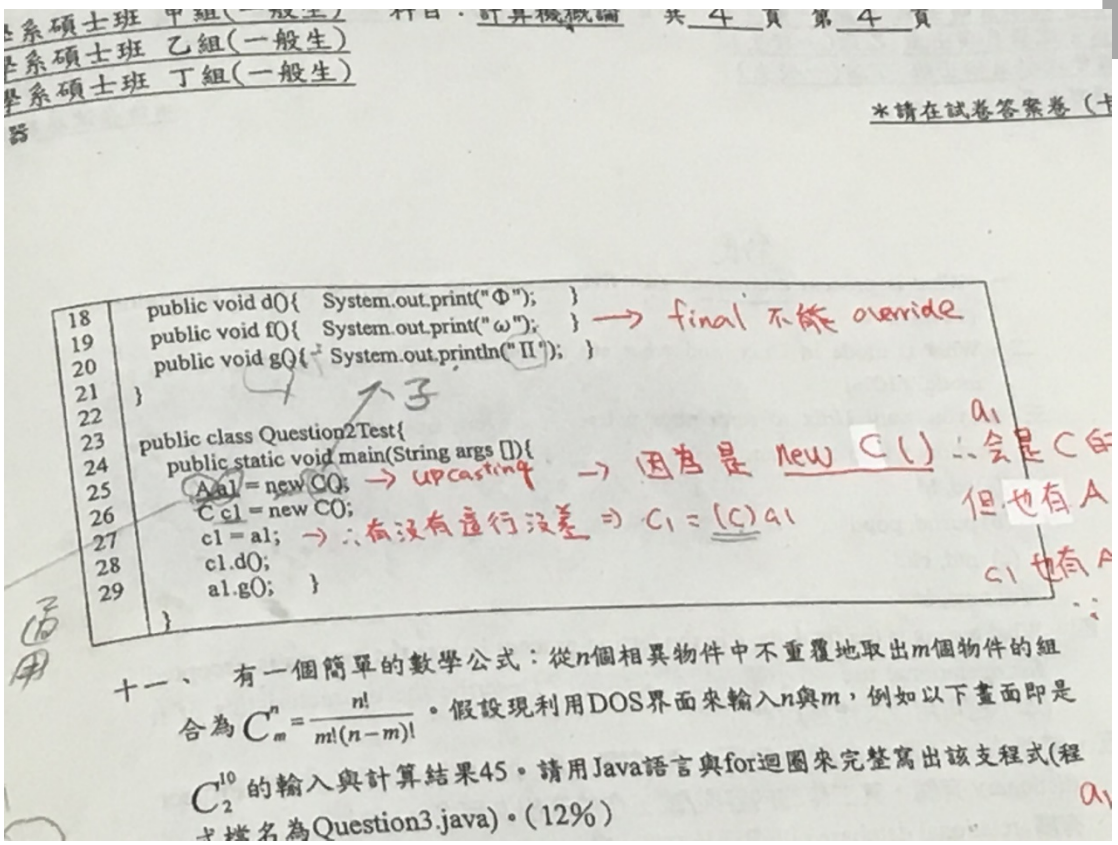
十、以下為數支Java程式內容，回答下列兩問題 (10%)

(1) 程式內容有數個編譯錯誤，請標出錯誤之行數並說明錯誤原因。注意：若答案無標示行數將不予計分！

(2) 若經過完整除錯後，試印出執行結果

註：若有標示『This line is correct』表示該行程式正確無誤

```
1  public interface B{   public void c();   }
2
3  public class A implements B{
4    public void A(){   System.out.print("Ω");  }
5    public abstract void a();   //This line is correct
6    public final void f(){   System.out.print("Λ");  } //This line is correct
7    public void g(){   System.out.print("Ψ");  }
8    public final double zz;
9  }
10
11 public class C extends A{     //This line is correct
12   public C(){
13     super();
14     System.out.print("Ω");  }
15   public void a(){   System.out.print("θ");   }
16   public void c(){   System.out.print("μ");   }
17 }
```



系碩士班 乙組(一般生)
系碩士班 丁組(一般生)
共 4 頁 第 4 頁
*請在試卷答案卷(卡

```
18  public void d(){  System.out.print("Φ");  }
19  public void f(){  System.out.print("ω");  }
20  public void g(){  System.out.println("Π");  }
21  }
22
23  public class Question2Test{
24    public static void main(String args []){
25      A a1 = new C();
26      C c1 = new C();
27      c1 = a1;
28      c1.d();
29      a1.g();   }
    }
```

final 不能 override

upcasting → 因為是 new C() ∴會是C的

但也有A

c1 也有A

十一、 有一個簡單的數學公式：從n個相異物件中不重覆地取出m個物件的組合為 $C_m^n = \frac{n!}{m!(n-m)!}$。假設現利用DOS界面來輸入$n$與$m$，例如以下畫面即是 C_2^{10} 的輸入與計算結果45。請用Java語言與for迴圈來完整寫出該支程式(程式檔名為Question3.java)。(12%)

問答題 (40%)

1. Consider the following JAVA code:

```
public interface Moveable{ void move();}

public abstract class Animal implements Moveable{
  private String name;
  public abstract void eat();
  public void move(){System.out.println("animal moves");}
  public void sleep(){..}
}

public class Mammal extends Animal{
  public void eat(){System.out.println("gimme meat!");}
  public void regulateTemperature(){..}
}

public class Human extends Mammal{
  public void think(){..}
  public void move(){System.out.println("human walks");}
}
```

A. What is the result of executing the following code? (which eat() method will invoke or compiler error)

```
Movable m = new Mammal();
m.eat();
```

B. What is the result of executing the following code? (which move() method will invoke or compiler error)

```
Animal a = new Human();
Movable m = a;
m.move();
```