“父類定義許多“空的”方法名稱(abstract)，因為方法要處理的指令步驟在各子類別有不同，因此，方法細節由各子類別中各自實作!”

抽象類別(Abstract class)、抽象方法(Abstract method)

簡介

一個抽象類別(abstract class)裡，有一個方法只有名字，沒有實作(implement)，這個方法稱為抽象方法(abstract method)。你要繼承這個類別，你就必須自己定義這個抽象方法，也就是實作出方法內容。

Abstract: 抽象 摘要
➔抽象這個詞就代表著不是很具體易懂!你必須親自透過具體的動手做去觀察體會。

重點摘要:

- ✓ 抽象方法 abstract method，只有給定名稱，沒有內容。
- ✓ 一個類別若是擁有 abstract method，就必須要冠上 abstract class 字眼，變成抽象類別。
- ✓ 一個抽象類別可以被繼承，抽象類別被繼承是其存在的主要目的。
- ✓ 一個抽象類別不能被初始化(被 new)，但是可以當作資料型別使用，讓一個陣列可以放各式各樣的子類別物件。
- ✓ this("標準貓"); //呼叫本身這個類別的另外一個建構子(前述有一個參數的)

父類別成員的修飾語 private、不寫、protected、public

請問父類別的 private 成員可以被子類別看到或使用嗎?

請問父類別的無修飾語成員可以被子類別看到或使用嗎?
請問父類別的 public 成員可以被子類別看到或使用嗎?
請問父類別的 protected 成員可以被子類別看到或使用嗎?

試一下就知道答案了!

更深的問題:為何要有這些修飾語?有何不同?

目的: 簡單一句話:對成員做不同等級程度的開放，確保安全，這是資料封裝(Encapsulation)的特色!

- 欄位的存取修飾語、方法的存取修飾語，影響該類別被使用的限制(Scope)

存取修飾語	同一類別	同一套件	子類別	全域 包含別的套件或其他專案的程式
private	可使用			
(default)(不寫)	可使用	可使用		
protected	可使用	可使用	可使用	
public	可使用	可使用	可使用	可使用(要先 import 才可)

有趣又簡單的實例

貓、狗等寵物的例子

抽象類別 Pet

```
1  /**
2  *抽象類別 抽象方法
3  move()在各貓狗類別中 有不同的移動方式
4  */
5  public abstract class Pet {
```

```
6
7       private String name;
8       //private int age;
9
10      public Pet(   String   name    )
11      {
12          this.name = name; //this 是指 "這個類別"
13          //System.out.printf("產生新物件:%s\n", name);
14      }
15      public abstract void move(); //只有方法名稱，暫時不要實作 因為每個子類別實作方式不一樣
16
17       public void showMe()
18      {
19          System.out.printf("我是%s，大家好!\n", name);
20      }
21      public static void main(String[] args) {
22       //產生 Pet 物件 並執行此物件所擁有的方法 showMe()
23          Pet p1 = new Pet("小黑"); //抽象類別不可以被 new 來用
24          p1.showMe();
25
26          //用這種奇怪的方式使用(先實作再 new)
27          Pet p1= new Pet("小花"){ //大括號內等於是一個匿名類別的內部，此匿名類別
28  繼承了 Pet,因此要實作抽象類別 move()
29
30              @Override
31              public void move() {
                    System.out.println("在內部定義一個匿名類別，與繼承自 Pet 得到相
    同的結果");
                }
            };  //這裡的分號不可少!
        }
    }
```

子類別 Cat

```
1   public class Cat extends Pet {
```

```
2       public Cat(String name) {
3           super(name);   //呼叫父類別的建構子
4       }
5       public Cat() //定義一個沒有參數的建構子
6       {
7           this("標準貓"); //呼叫本身這個類別的另外一個建構子(前述有一個參數的)
8       }
9
10      @Override
11      public void showMe() {
12          super.showMe();   //呼叫父類別的 showMe()
13          System.out.println("我是一隻貓!");
14      }
15      public void meow() {
16          System.out.println("喵喵!");
17      }
18
19      @Override
20      public void move() {
21          System.out.println("輕聲走兩步");
22      }
23
24      public static void main(String[] args) {
25          Cat cat1 = new Cat("小貓");
26          cat1.showMe();
        }
    }
```

子類別 Dog

```
1
2   public class Dog extends Pet {
3
4       public Dog(String name) {
5           super(name);   //呼叫父類別的建構子
6       }
```

```
7
8       @Override      //取代父類別的 showMe()，重新定義這個方法所執行的指令
9       public void showMe() {
10          super.showMe();   //呼叫父類別的 showMe()
11          System.out.println("我是一隻狗!");
12      }
13
14      public void bark() //定義屬於 Dog 類別專屬的方法 bark()
15      {
16          System.out.println("汪汪!");
17      }
18
19      //實作 move()抽象方法
20      @Override
21      public void move() {
22          System.out.println("我大跳 5 公尺");
23      }
24
25      public static void main(String[] args) {
26          Dog d1 = new Dog("小黑");
27          d1.showMe(); //呼叫的 showMe()是在 Dog 類別中重新定義的 showMe()喔!
28      }
29  }
```

類別 PetZoo

```
1   package oodemo_02_inheri_abstract;
2
3   public class PetZoo{
4
5       public static void main(String[] args) {
6
7   //存放 寵物物件的 陣列 可以放 Dog 物件 也可以放 Cat 物件
8           Pet[] pets = new Pet[4];
9
```

```
10              pets[0] = new Cat("小花");
11              pets[1] = new Cat("Kitty");
12              pets[2] = new Dog("小黑");
13              pets[3] = new Dog("小白");
14
15            //寵物物件各自呼叫 showMe()
16              for (int i=0; i< pets.length; i++)
17              {
18                    pets[i].showMe();
19                    //呼叫父類別中定義名稱的 move()，為甚麼直接可以呼叫？
20                    //怎麼知道要大跳還是輕聲走?
21                    pets[i].move();
22
23
24                    //貓叫 狗叫 是 子類別 特有的方法  因此需要 向下型轉之後才可呼叫
25                    if (pets[i] instanceof Cat)
26                    {
27                          Cat kitty = (Cat)pets[i];
28                          kitty.meow();
29                    } else if (pets[i] instanceof Dog)
30                    {
31                          Dog dd = (Dog)pets[i];
                            dd.bark();
                      }
              }
      }
}
```

作業 1

請仿照上述實例，自行設計並實作另外一個抽象類別(Abstract class)、抽象方法(Abstract method) 的應用案例。

作業 2: 研究所入學考題(國立中山大學)

八、本程式例應用繼承與多形概念設計而成數支 Java 程式: A, B, D 以及 Demo；其中 D 為 final class。回答下列的問題：

1. (6%) 何謂多形(Polymorphism)(寫出並解釋多形的兩項重要特色)？並依這兩項特色分別指出程式中展現多形設計之處（以程式行碼作答並指出該行特定針對多形的程式內容)
2. (4%) 指出程式中違反封裝(Encapsulation)原則之處（以程式行碼作答即可)？如何改正？
3. (4%) 指出 Class B 內錯誤之處(以程式行碼作答)並改正之
4. (5%) 完成 Class D 的內容
5. (4%) 指出這程式例中的 method overriding 的所有程式行碼

6. (2%) 指出這程式例中的 method overloading 的所有程式行碼

```
C:\2014 grad exam\case 3>java Demo
第一行列印：printing t
第二行列印：printing f
C:\2014 grad exam\case 3>
```

```
01  public abstract class A{
02     public int VA=1;
03     public abstract String ma();
04     public String mb(){ return "printing r"; }
05  }
06
07  public class B extends A{
08        public B(){}
09        public String mb(){ return "printing t"; }
10        public String mf(){ return "printing f"; }
11  }
12
13  public class D extends B{
14        public String mh(){ return "printing h"; }
15        public String mh(int VA){return "printing hh"; }
16        public String mx(){ return "printing x"; }
17        //Class D is incomplete
18  }
19
20  public class Demo{
21     public static void main(String [] args){
22     B b = new D();
23           System.out.println("第一行列印：" + b);
24           System.out.println("第二行列印：" + b.mf());
25     }
26  }
```