繼承 Inheritance

簡介

透過繼承(inheritance)可以讓衍生類別“享用”父類別的所有功能(public, protected 的變數屬性值與方法)。

父類別 Parent class, Superclass
子類別、衍生類別 Derived class, Subclass

這些都是繼承:
(1)Java
class Cat extends Pet{}

(2)C++, C#
class Cat:Pet{}

(3)Python
class Cat(Pet):
    pass

重點摘要:

- ✓ 繼承是 OOP 物件導向程式設計的主要核心精隨之一。
- ✓ 透過繼承可以擴充一個類別的功能,通常父類別是由各個子類別所需要的共通功能所組成。
- ✓ Java 的子類別只能繼承一個父類別，可以讓複雜度降低。
- ✓ 一個子類別只會有一個父類別，也只會有一個父父類別,父父父類別,...可以有很多層
- ✓ 子類別的 showMe()方法，若不打算完全 override 掉父類別的 showMe()，可以使用 super.showMe()，先呼叫父類別的 showMe()，再額外定義其餘的指令。

- ✓ 父類別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。
- ✓ super()或 super(參數)是用來呼叫父類別的建構子，且 super()必須放在第一行，表示父類別的初始化必須早於子類別的初始化。
- ✓ 如果使用者自己沒有呼叫 super(),則編譯器會自己幫我們加上 super()的呼叫,編譯器只會自動幫我們呼叫沒有參數版本的建構子 super()。
- ✓ this("標準貓")，呼叫本身這個類別的另外一個建構子(通常是要執行相同步驟，可以把共同步驟的指令放在這個建構子中)，同樣也必須放在第一行。
- ✓ 每個 constructor 都可以呼叫 this()或 super(),但 this()或 super()可以同時使用嗎? this()必須放在第一行 super()必須放在第一行可以同時滿足?

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

Pet --Cat
--Dog

Pet 類別

```
1   父類別
2   public class Pet {
3
4       private String name;    //instance variable, field  實體變數或欄位
5       //private int age;
6
7       public Pet(    String    name      )
8       {
9           this.name = name; //this  是指  "這個類別"
10          System.out.printf("產生新物件:%s\n", name);
11      }
12      public void showMe()
13      {
14          System.out.printf("我是%s，大家好!\n", name);
15      }
16
17      public static void main(String[] args) {
18       產生 Pet 物件  並執行此物件所擁有的方法 showMe()
19          Pet p1 = new Pet("小黑");
20          p1.showMe();
21          new Pet("小花");
22      }
23  }
24
```

25	

Cat 類別

```
1   子類別 Cat
2
3   public class Cat    extends Pet {
4
5   public Cat( String    name    )
6   {
7         super(name);    //呼叫父類別的建構子
8   }
9   public Cat() //定義一個沒有參數的建構子
10  {
11     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)
12  }
13
14  public void meow()//定義屬於 Cat 類別專屬的方法 meow()
15  {
16        System.out.println("喵喵!");
17  }
18
19  public static void main(String[] args) {
                Cat    cat1 = new Cat("小貓");
                cat1.showMe();
          }


    }
```

Dog 類別

```
1   //子類別 Dog
2   public class Dog    extends Pet {
3
4   public Dog( String    name    )
5   {
6         super(name);    //呼叫父類別的建構子
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  {
11      super.showMe();   //呼叫父類別的 showMe()
12      System.out.println("我是一隻狗!");
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  public static void main(String[] args) {
20          Dog   d1 = new Dog("小黑");
21          d1.showMe(); //呼叫的  showMe()是在 Dog 類別中重新定義的 showMe()喔!
22          d1.bark();
23      }
24  }
25
```

PetZoo 類別

```
1   public class PetZoo {
2
3       public static void main(String[] args) {
4
5           //存放 寵物物件的 陣列 可以放 Dog 物件 也可以放 Cat 物件
6           //這就是  多型 polymorphism 的觀念
7           Pet[] pets = new Pet[4];
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          //寵物物件各自呼叫 showMe()
15          for (int i=0; i< pets.length; i++)
16          {
```

```
17                  pets[i].showMe();
18
19                  //貓叫 狗叫
20                  if (pets[i] instanceof Cat)
21                  {
22                      Cat kitty = (Cat)pets[i];
23                      //強制型別轉換: 向下型轉 down cast 為子類別物件
24                      kitty.meow();
25                  } else if (pets[i] instanceof Dog)
                    {
                        Dog dd = (Dog)pets[i];
                        dd.bark();
                    }
                }
            }

        }
```