



J01-12

進階物件導向程式設計

曾瑞君 (Jim_Tzeng)

學習目標

1. 了解繼承 (inheritance)
2. 繼承和建構子(Constructor)的關係
3. 使用父類別(super-class) 和子類別(sub-class)
4. 了解多型(Polymorphism)和覆寫(Override)方法
5. 了解介面(Interface)
6. 物件始祖 【Object】





1/6

了解繼承 (Inheritance)

Edited by Ruei-Jiun Tzeng



何謂繼承？

- 繼承在法律層面，談的多半是被繼承人身故後，財產方面的轉移。
- 繼承在精神方面，有「繼承遺志」的說法，談的是完成前人未完成的事情。
- 繼承在行為方面，則關係到父母親和子女間在習慣上的相似，所以有人說「龍生龍、鳳生鳳，老鼠生的兒子會打洞」。
- 即便繼承，也可以走自己的路，所以「青出於藍而勝於藍」。

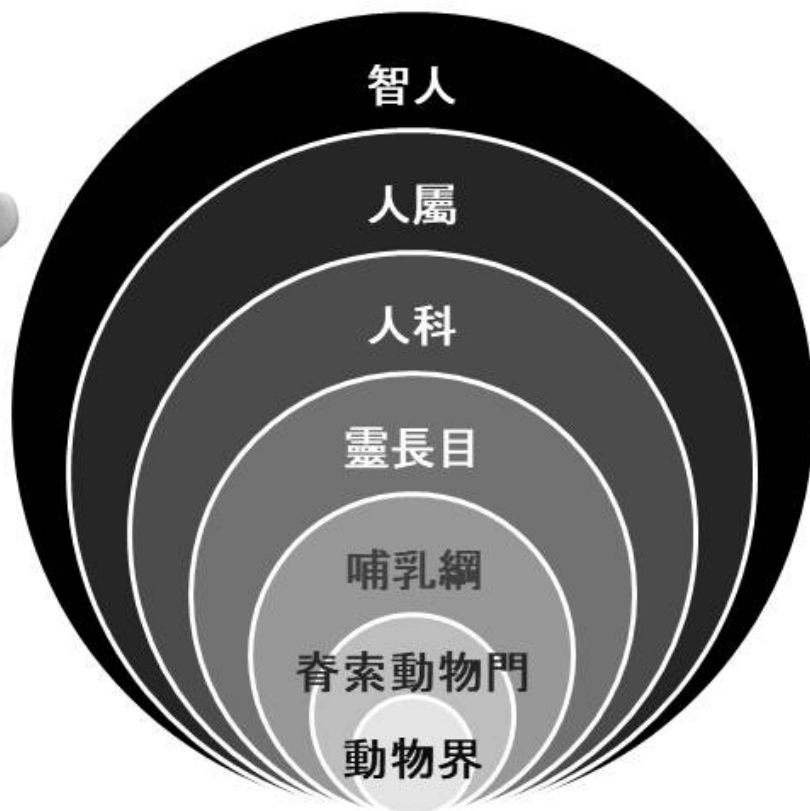
自然界生命的 分類(class) & 繼承(Inheritance)

自然界的生命種類很多，觀察每個「物種」的屬性和行為方式的不同，用「界門綱目科屬種」等七種階層來「分類(class)」。

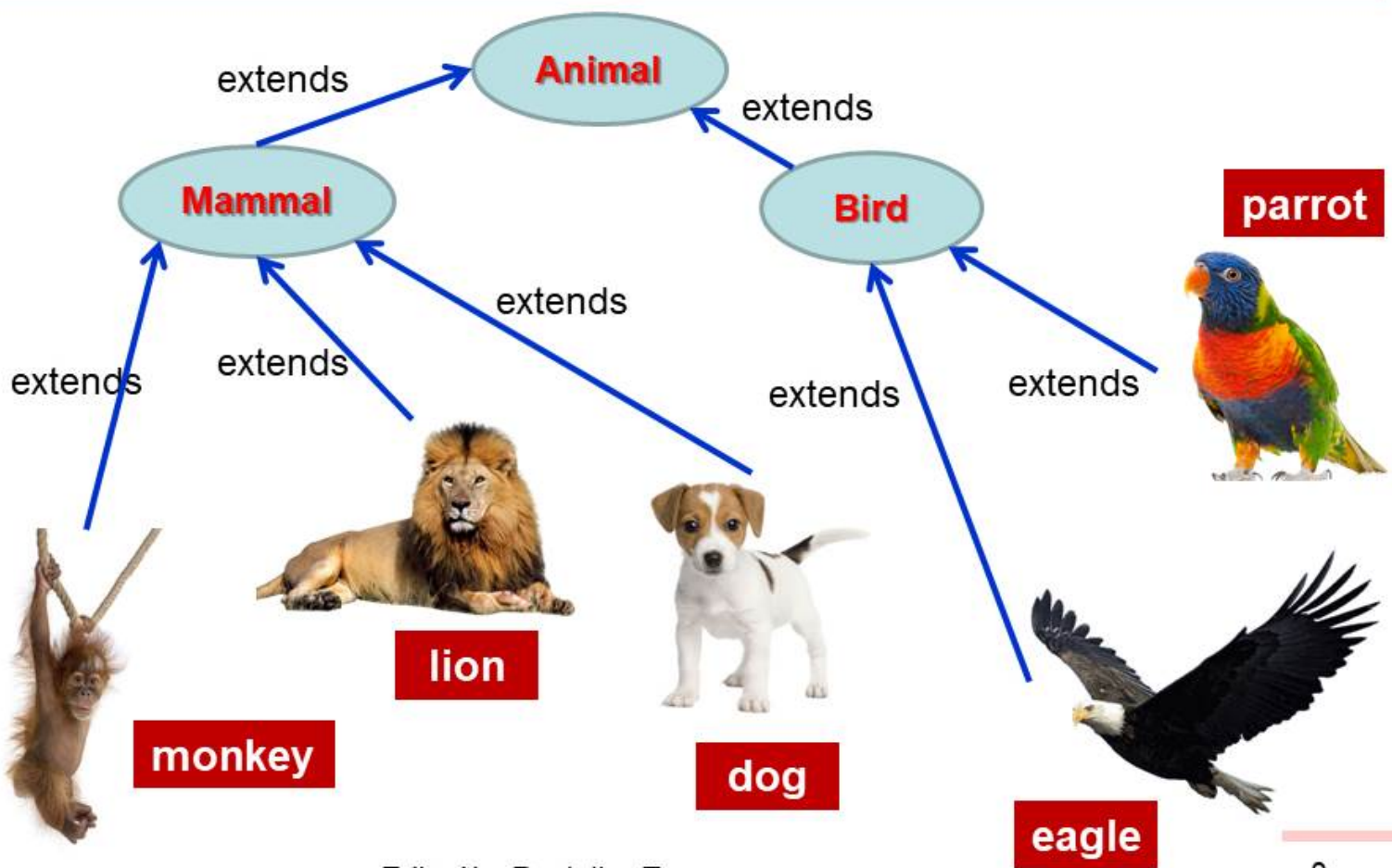


自然界生命的 分類(class) & 繼承(Inheritance)

- 不同階層的關係，可以用「繼承」的概念來思考。脊索動物門隸屬於動物界，是因為牠們具有動物界的特質(至少能“動”)，所以可以說脊索動物門，繼承了動物界，所以取得動物界的屬性和行為。
- 以人類來說，由分類下層的「智人」到上層的「動物界」，我們一脈相承，「繼承」了每個階層的屬性和能力；所以到最後「智人」的出現，等於是集大成。



Java 的 類別(class) & 繼承(Inheritance)



Edited by Ruei-Jiun Tzeng



Java 的 類別(class) & 繼承(Inheritance)

Java 的類別和繼承，概念與自然界的分類和繼承如出一轍。所以透過繼承的做法，可以：

1. 取得上一層的屬性和方法
2. 青出於藍而勝於藍(使用覆寫)
3. 繼承遺志，完成上一代的遺珠之憾(使用抽象類別和方法)

善用繼承減輕自己負擔。但Java只允許單一繼承，亦即每個類別只能繼承單一類別。機會只有一次，慎選繼承對象。



2/6

繼承和建構子(Constructor) 的關係

繼承的宣告與目的

- 繼承的宣告語法：

```
class sub-class extends super-class {  
    // content of sub-class  
}  
  
// sub-class: 子類別  
// super-class: 父類別
```

- 繼承的宣告目的：
 - 子類別繼承父類別後，可以取得父類別屬性和方法，所以程式碼可以重複使用。
 - 多個類別有相似程式碼，可以將相同部份抽出建立父類別，再以繼承父類別的方式共用程式碼。

```

class SuperXParent {
    public SuperXParent() {
        System.out.println("動物界");
    }
}
class Super4Parent extends SuperXParent {
    public Super4Parent() {
        System.out.println("脊索動物門");
    }
}
class Super3Parent extends Super4Parent {
    public Super3Parent() {
        System.out.println("哺乳綱");
    }
}
class Super2Parent extends Super3Parent {
    public Super2Parent() {
        System.out.println("靈長目");
    }
}

```

```

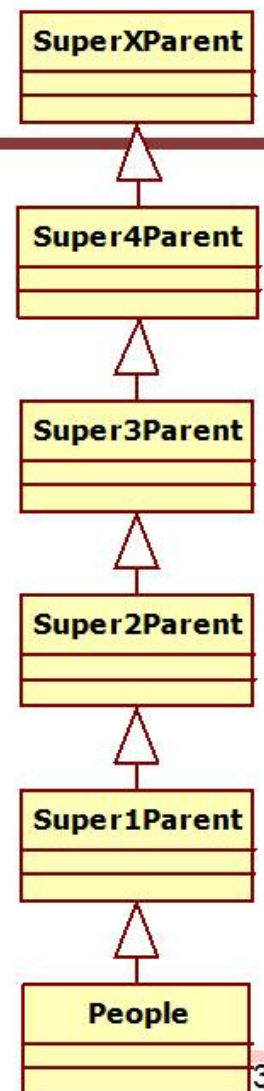
class Super1Parent extends Super2Parent {
    public Super1Parent() {
        System.out.println("人科");
    }
}
class Parent extends Super1Parent {
    public Parent() {
        System.out.println("人屬");
    }
}
class People extends Parent {
    public People() {
        System.out.println("智人");
    }
}
public class TestConstructors {
    public static void main(String[] args) {
        People c = new People();
    }
}

```

UML 表示方式 & 執行結果

- 執行結果：

動物界
脊索動物門
哺乳綱
靈長目
人科
人屬
智人





Constructor並未繼承

- Constructor非物件成員，無法繼承。但有 2 種方法取得：
 - 建立自己 Constructor
 - 使用Default Constructor (無參數)
 - 如果未建立建構子，java 將提供 Default Constructor。
 - 反之，若開發者已建立類別的新建構子，將不再提供 Default Constructor。若有需要則自己建立。
- 存取自己的Constructor，使用**this()**
- 存取父類別的Constructor，使用**super()**
- **this** 可以用來存取目前類別的fields或methods。 **super** 可用來存取父類別的fields或methods。

子類別實例化和父類別的關係 -1

- 1) 物件實例化的過程裡，必須先進行在父類別的建構子的內容，再進行子類別的建構子的內容。
- 2) 為達到這個目的，在每個子類別的建構子，都必須在第一行程式碼呼叫一個父類別的建構子。
- 3) Java 可以幫忙將 `super()` 偷偷放到子類別的建構子的第一行。但
 - 前提是`super()`：“父類別的default 建構子”必須 **存在**。
 - **若不存在**，則 developer 必須幫 **每個子類別的建構子**都 **各**找一個合適的父類別的建構子，加在第一行。

```

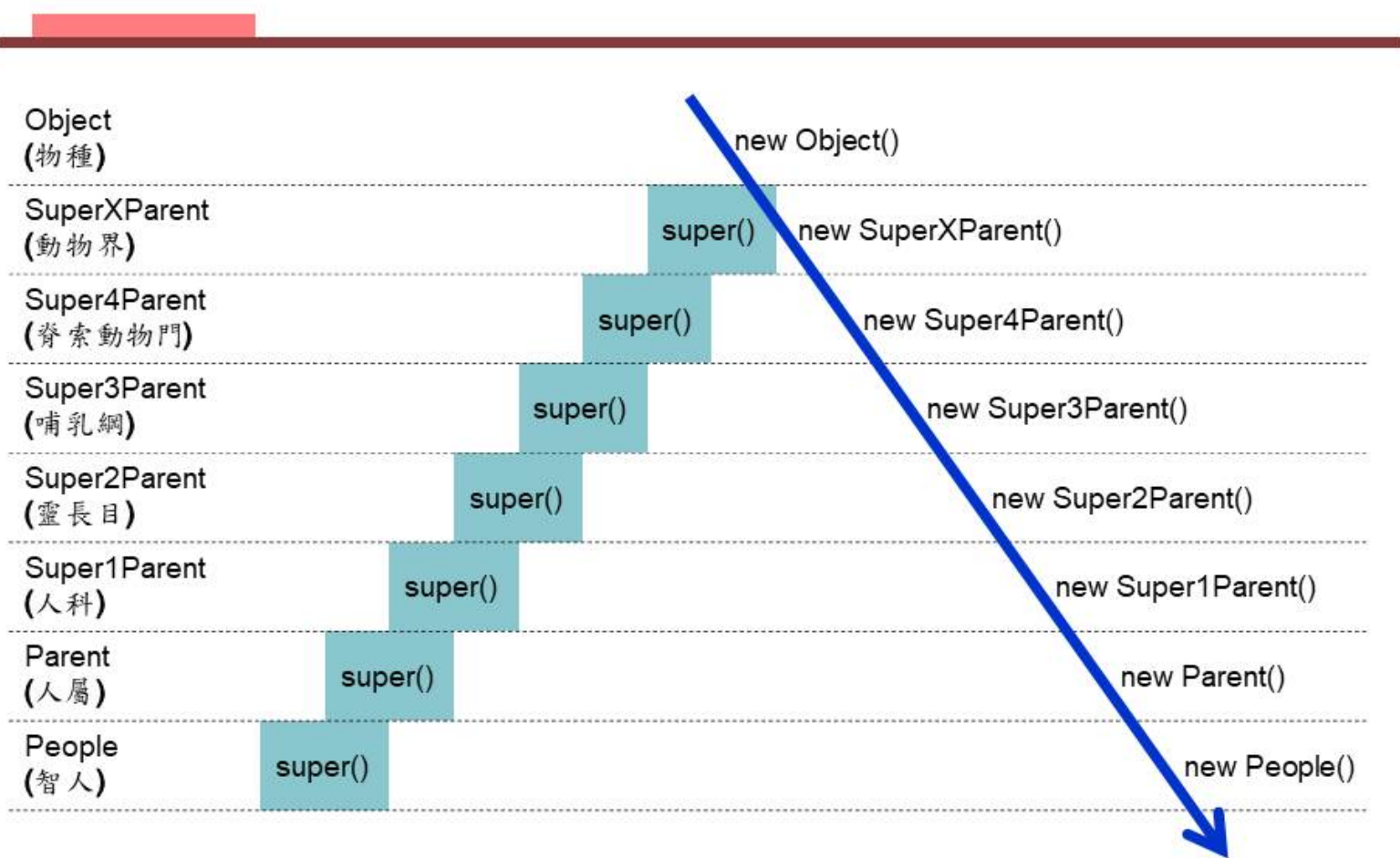
class SuperXParent extends Object {
    public SuperXParent() {
        super();
        System.out.println("動物界");
    }
}
class Super4Parent extends SuperXParent {
    public Super4Parent() {
        super();
        System.out.println("脊索動物門");
    }
}
class Super3Parent extends Super4Parent {
    public Super3Parent() {
        super();
        System.out.println("哺乳綱");
    }
}
class Super2Parent extends Super3Parent {
    public Super2Parent() {
        super();
        System.out.println("靈長目");
    }
}

```

```

class Super1Parent extends Super2Parent {
    public Super1Parent() {
        super();
        System.out.println("人科");
    }
}
class Parent extends Super1Parent {
    public Parent() {
        super();
        System.out.println("人屬");
    }
}
class People extends Parent {
    public People() {
        super();
        System.out.println("智人");
    }
}
public class TestConstructors1 {
    public static void main(String[] args) {
        People c = new People();
    }
}

```



子類別實例化和父類別的關係 -2

- 4) 「子類別建構子必須在**第一時間**呼叫父類別建構子」的過程不一定要『直接』，只要保證子類別的每個建構子的**第一件事**就是呼叫父類別的建構子即可。

```

class Super {
    Super(String s) {
        System.out.println("Super");
    }
}

class Sub extends Super {
    Sub(String s) {
        super(s);
        System.out.println("Sub");
    }
    Sub(String s1, String s2) {
        this(s1); // will compile?
        System.out.println("Sub");
    }
}

```

```


public class TestConstructors2 {
    public static void main(String[] args) {
        new Sub("Jim");
        System.out.println("-----");
        new Sub("Hi", "Jim");
    }
}

```

```

Super
Sub
-----
Super
Sub
Sub

```

子類別實例化和父類別的關係 -3

5) 因為在子類別的建構子中

- 呼叫父類別建構子 `super(...)`
- 呼叫自己的其他建構子 `this(...)`

都有機會達成「踩階梯」的目的，因此這兩種建構子的呼叫都必須在**第一行**程式碼。

6) 為了保證前述原則，建構子內只能有一次機會呼叫父類別或自己的其他建構子。兩種情況不能同時存在，也必須在第一行。


```
class Vehicle {
    int x;
    Vehicle() {
        this(10);
    }
    Vehicle(int x) {
        this.x = x;
    }
}
class Car extends Vehicle {
    int y;
    Car() {
        super();
        this(20); // compile error
    }
    Car(int y) {
        this.y = y;
    }
}
public class TestConstructors3 {}
```



總結

- 建構子第一行必須是`super()`或`this()`。
- `super()`或`this()`也只能在建構子第一行。



3/6

使用

父類別(**super-class**) 和 子類別(**sub-class**)

Edited by Ruei-Jiun Tzeng

分析類別成員差異

購衣網站除了襯衫(Shirt)，準備再開賣其他衣物類產品，如褲子(Trousers)和襪子(Sock)。

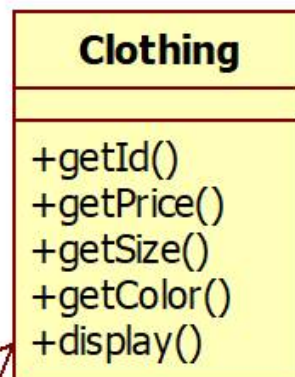
但分析物件成員後，發現大部分都相同...

Shirt	Trousers	Sock
getId()	getId()	getId()
getPrice()	getPrice()	getPrice()
getSize()	getSize()	getSize()
getColor()	getColor()	getColor()
getFit()		
	getGender()	
setId()	setId()	setId()
setPrice()	setPrice()	setPrice()
setSize()	setSize()	setSize()
setColor()	setColor()	setColor()
setFit()		
	setGender()	
display()	display()	display()

使用繼承

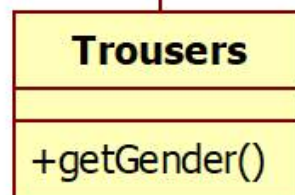
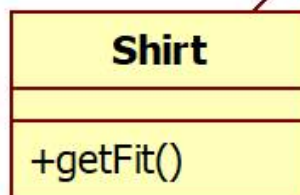
Super class:

父類別：



Sub class:

子類別：





Overriding Superclass Methods

一旦將相同的methods抽至父類別後，

1. 若子類別有相同簽名的 method：
 - 此時稱子類別方法覆寫 (override) 了父類別方法
 - 執行時將使用該子類別的 method。
2. 若子類別無相同簽名的 method：
 - 執行時將使用父類別的 method

建立父類別 Clothing

```
public class Clothing {  
  
    private int itemID = 0;  
    private String description = "-description required-";  
    private char colorCode = 'U';  
    private double price = 0.0;  
  
    public String getDescription() {return description;}  
    public double getPrice() {return price;}  
    public int getItemID() {return itemID;}  
    public char getColorCode() {return colorCode;}  
  
    public void setItemID(int itemID) {this.itemID = itemID;}  
    public void setDescription(String description) {this.description = description;}  
    public void setColorCode(char colorCode) {this.colorCode = colorCode;}  
    public void setPrice(double price) {this.price = price;}  
}
```

建立父類別 Clothing

```
public class Clothing {  
  
    // ...other codes  
  
    public Clothing(int itemID, String description, char colorCode, double price) {  
        this.itemID = itemID;  
        this.description = description;  
        this.colorCode = colorCode;  
        this.price = price;  
    }  
  
    public void display() {  
        System.out.println("Item ID: " + getItemID());  
        System.out.println("Item description: " + getDescription());  
        System.out.println("Item price: " + getPrice());  
        System.out.println("Color code: " + getColorCode());  
    }  
}
```

Edited by Ruei-Jiun Tzeng

建立子類別 Shirt

- 使用 `extends`, `super`, `this`

```
public class Shirt extends Clothing {  
    private char fit = 'U'; //S=Small,M=Medium,L=Large, U=Unset  
    public Shirt(int itemID, String description, char colorCode, double price, char fit) {  
        super (itemID, description, colorCode, price);  
        this.fit = fit;  
    }  
    public char getFit() {  
        return this.fit;  
    }  
    public void setFit(char fit) {  
        this.fit = fit;  
    }  
}
```

建立子類別 Shirt

- Override 父類別的方法

```
public class Shirt extends Clothing {  
  
    // ...other codes  
  
    @Override  
    public void display() {  
        super.display();  
        System.out.println("Fit: " + this.getFit());  
    }  
  
}
```

繼承和封裝

Modifier (keyword)	同一class	同一package	不同package but 子類別	公開
private (最嚴格)	Y			
default (次嚴格)	Y	Y		
protected	Y	Y	Y, 透過super取	
public (最寬鬆)	Y	Y	Y	Y

Protected Access Control: Example

```
package course.c12.accessCtl.demo;
public class Foo {
    protected int result = 20;
    int other = 25;           //default, 不能跨package
}
```

```
package course.c12.accessCtl.test;
import course.c12.accessCtl.demo.Foo;
public class Bar extends Foo {
    private int sum = 10;
    public void reportSum () {
        sum += result;       //OK
        sum += other;        //NG
    }
}
```




```
class Deeper {  
    public Number getDepth( Number n) {  
        return 10;  
    }  
}
```

```
class DeepA extends Deeper {  
    @Override  
    protected Integer getDepth( Number n) {  
        return 5;  
    }  
}
```

```
class DeepB extends Deeper {  
    @Override  
    public Double getDepth( Number n) {  
        return 5d;  
    }  
}
```

```
class DeepC extends Deeper {  
    @Override  
    public String getDepth( Number n) {  
        return "";  
    }  
}
```

```
class DeepD extends Deeper {  
    @Override  
    public Long getDepth( int d) {  
        return 5L;  
    }  
}
```

```
class DeepE extends Deeper {  
    @Override  
    public Short getDepth( Integer n) {  
        return 5;  
    }  
}
```

```
class DeepF extends Deeper {  
    @Override  
    public Object getDepth( Object n) {  
        return 5;  
    }  
}
```



Overridden Methods 注意事項

- 簽名相同
- Access Modifier
- Throws ExceptionType
- Return Type



Overload vs. Override

- 子類別繼承父類別後，擁有父類別部分方法(依 modifier 而定)。
- 單一類別內的方法，方法簽名不可以相同；若子類別繼承父類別後有相同簽名的方法，父類別方法將被子類別 override。
- 子類別方法(含繼承自父類別)，若簽名不同但名稱相同，稱為 overload。

Abstract Classes (抽象類別)

- 已知子類別 Shirt、Trousers、Sock，都繼承自父類別 Clothing。
- 子類別 Shirt、Trousers、Sock 有具體樣式：



- 父類別 Clothing 應該像甚麼？

抽象(Abstract)的原因

```
public abstract class Clothing {...}
```

- class前加上【abstract】宣告，表示：
 - 該類別屬於抽象的概念，不該被實例化(new)成為實體 object。
 - 該類別有無內容的method，導致無法被實例化。如：

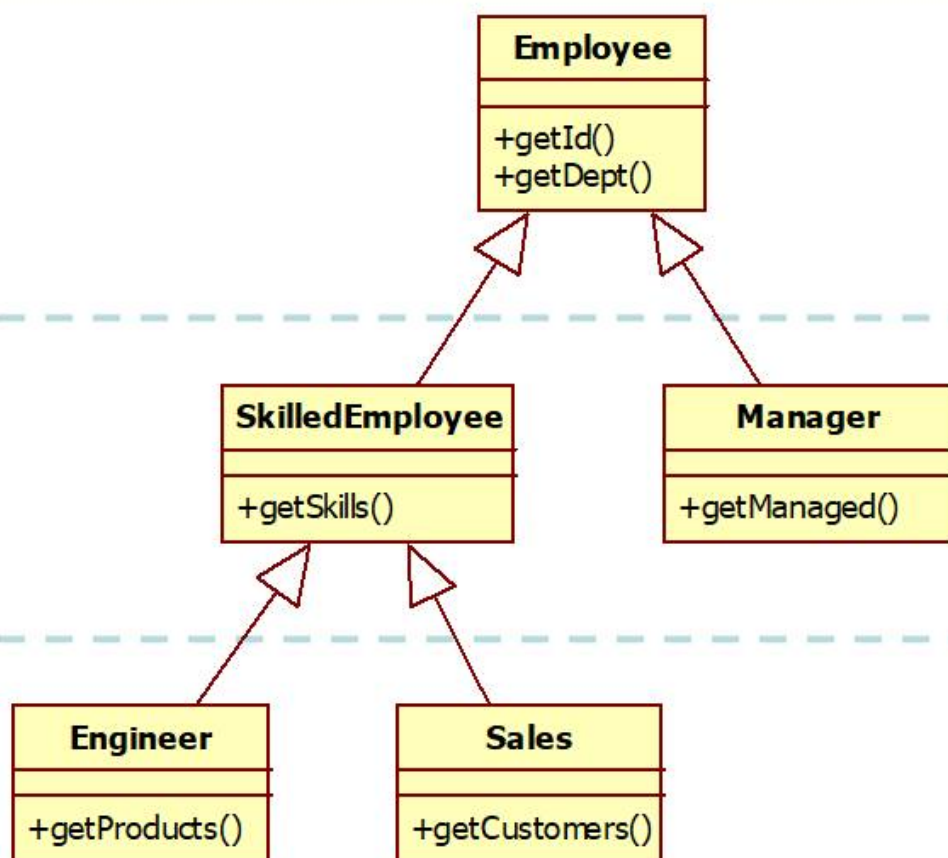
```
public abstract char getColorCode();  
public abstract void setColorCode(char colorCode);
```

- 子類別若繼承abstract class，且將被實例化，則必須實作abstract method.
- abstract class因無法實例化，多扮演父類別角色。

父子類別關係的釐清

- 【繼承】的使用，最常被誤以為只要能共用 method，避免 duplicated code，就符合條件!!其實要 reuse code，在 OO 的程式裡還有其他方法。
- 避免誤用的方法：用【is a】的語法來檢查。如：
 - Shirt is a Clothing => OK
 - Shirt is a Sock => NG
- Object 之間的關聯性，則用【has a】，如：
 - Car has a Wheel

其他繼承樹案例





4/6

多型 (Polymorphism)

Edited by Ruei-Jiun Tzeng

使用 Superclass 做 Reference Types

- 過去都是這樣寫：

```
Shirt s = new Shirt();
```

- 其實這樣也可以：

```
Clothing s = new Shirt();
```

```
Clothing t = new Trousers();
```

多型

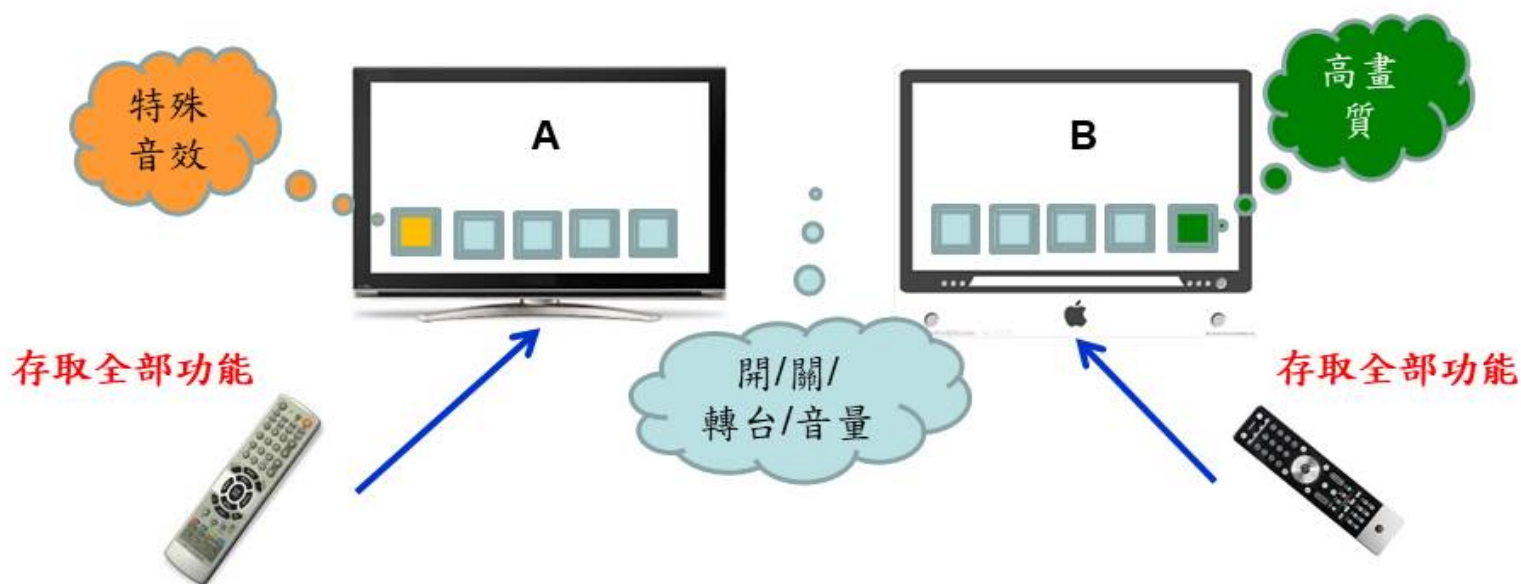
多型

可以使用「父類別」或是「本身」的參考型別去參照同一個物件實例，稱為【多型】。

也就是「一」個物件實例，可以有「多」個宣告型態(型別)。

過去...

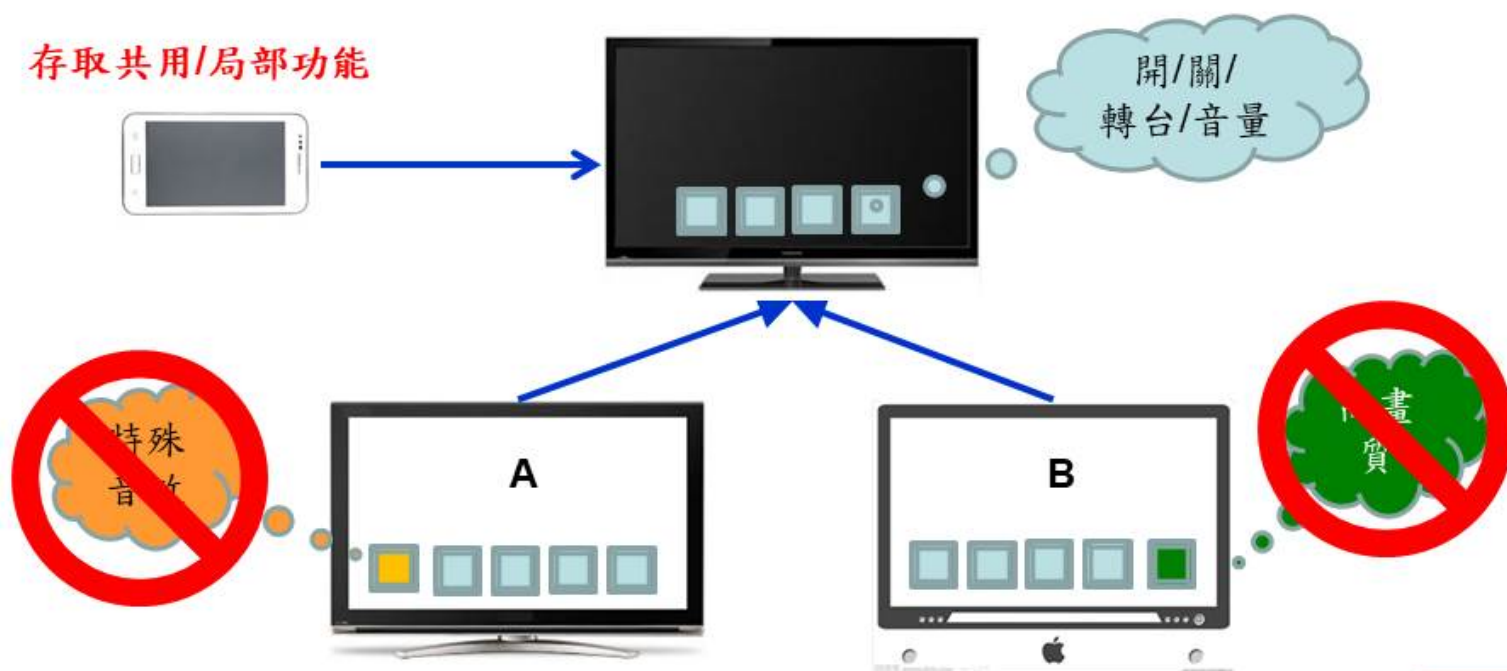
- 使用多型前，每台電視都必須使用專用的遙控器存取，可以使用獨特功能。



現在...

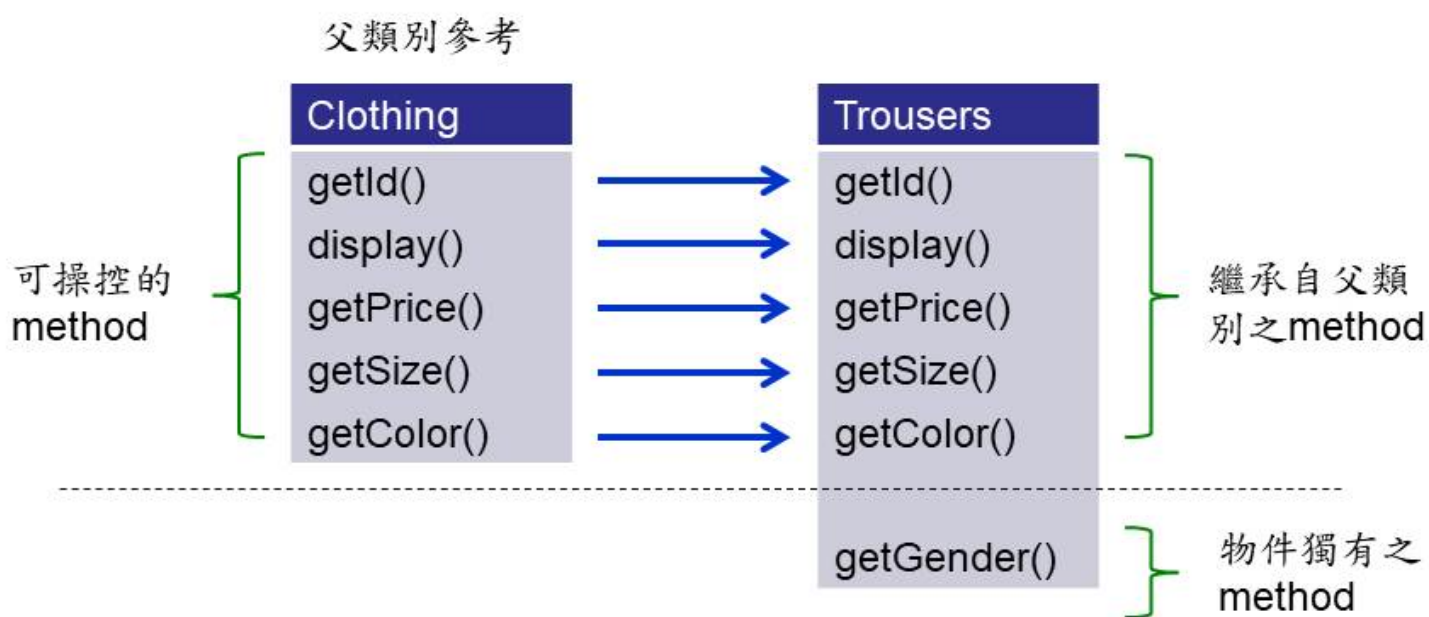
- 使用多型後，改用萬用遙控器存取所有電視，但只能使用基本功能：

存取共用/局部功能



Edited by Ruei-Jiun Tzeng

由父類別參考 控制 子類別方法



使用 casting 取得全部控制

- 透過 【(目標型別)物件參考】 的語法轉型
- 不當轉型在runtime時將會出錯

```
Clothing cl = new Trousers (1, "Dress Trousers", 'B', 1200.5, 6, 'M');  
cl.display();
```

```
char gender = cl.getGender(); // 無法編譯!!
```

```
char gender = ( (Trousers) cl ).getGender(); // 可編譯
```

Upward (向上) Casting

- 在基本型別時，若宣告型態較大，在assign value時會自動升級 (Automatic Promotion)。

```
int i = 10;  
long l = i;
```

- 如同基本型別，宣告的參考型別若為物件父類別，稱為 upward casting，將自動完成。

```
Shirt s = new Shirt();  
Clothing c = s;
```

Downward (向下) Casting

- 向下轉型時，compiler 只檢查目標是否是子類別。但runtime 時可能遇到 `java.lang.ClassCastException`。
- 假如Casting的對象超出類別繼承架構，將compile 失敗



轉型(Casting)原則 -1

- 「向上轉型」成為父類別：讓可以使用的成員變少，因為沒有風險，所以Java在運算式不對等的情況下會自動發動。
- 「向下轉型」成為子類別：讓可以使用的成員變多，有風險，必須自己發動。
- 「平行轉型」讓自己轉成自己：沒有風險，通常也沒有意義，必須自己發動，



轉型(Casting)原則 -2

- 編譯(compile)時期
 - 看「宣告型別」。允許「向上轉型」或「平行轉型」或「**向下轉型**」。
- 執行(run)時期
 - 看「實例型別」，允許「向上轉型」或「平行轉型」。
- 因為採用「多型宣告」會讓宣告型別大於實例型別，所以編譯時期相較執行時期可以多接受「向下轉型」。

//大指向小，轉型為小

```
long l1 = 10;  
int i1 = (int)l1;
```

//大指向大，轉型為小

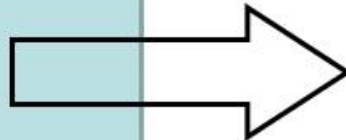
```
long l2 = 1000000000000L;  
int i2 = (int)l2;    //overflow  
System.out.println(i2);
```

//大指向小，轉型為小

```
Clothing c1 = new Shirt();  
Shirt s1 = (Shirt)c1;
```

//大指向大，轉型為小

```
Clothing c2 = new Clothing();  
Shirt s2 = (Shirt)c2;
```



編譯時期：向下轉型
執行時期：平行轉型



編譯時期：向下轉型
執行時期：向下轉型



何謂 多型？

- 多型操作指的是使用【同一個】操作介面(super-class reference type)，以操作【不同】的物件實例(class instance)
- 在設計並不依賴於具體類別，而是依賴於【介面/抽象/父類別】。

```

public static void displayShirt (Shirt r) {
    r.display();
}
public static void displayTrouser (Trouser t) {
    t.display();
}
public static void displaySock (Sock s) {
    s.display();
}

```



```

public static void displayClothing (Clothing c) {
    c.display();
}

```

```

public static void main(String[] args) {
    Shirt r = new Shirt();
    Trouser t = new Trouser();
    Sock s = new Sock();
    displayShirt(r);
    displayTrouser(t);
    displaySock(s);
}

```



```

public static void main(String[] args) {
    Clothing r = new Shirt();
    Clothing t = new Trouser();
    Sock s = new Sock();
    displayClothing(r);
    displayClothing(t);
    displayClothing(s);
}

```



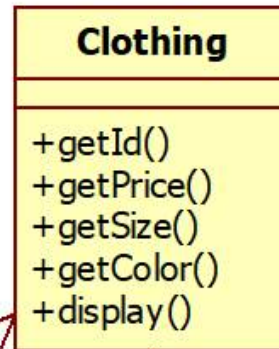
5/6

了解介面(interface)

怎麼退貨？

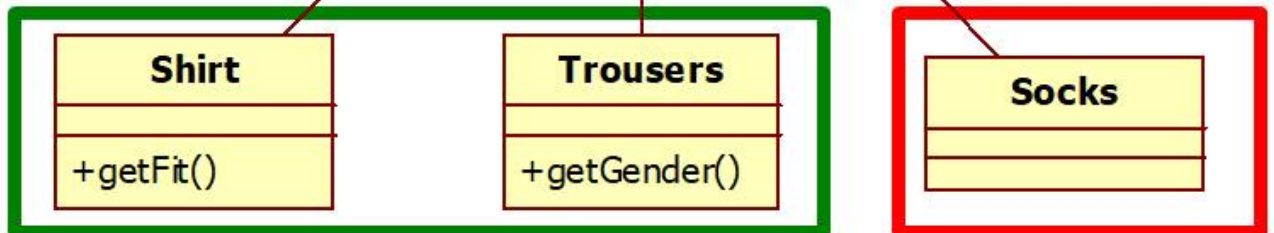
Super class:

父類別：



Sub class:

子類別：



可退貨

無法退貨

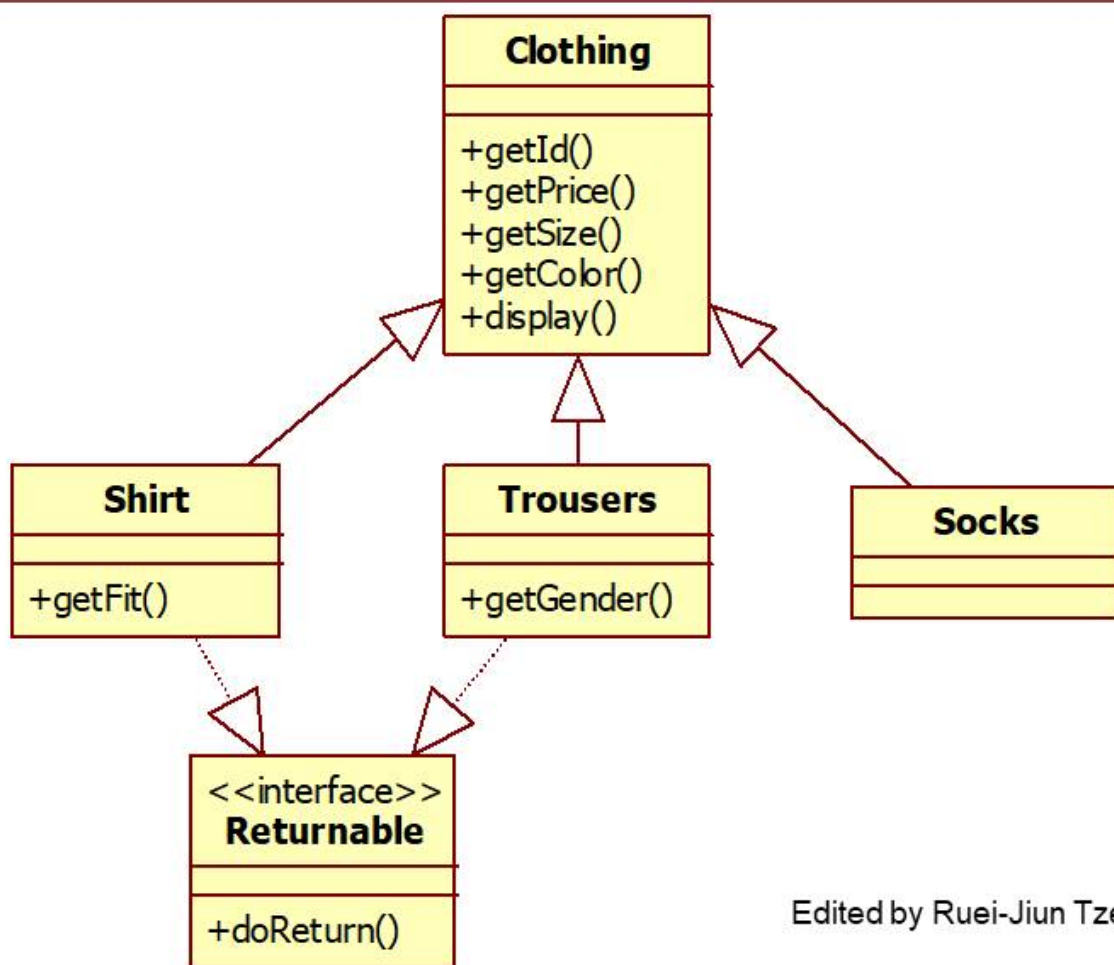
使用“多”繼承來處理退貨？

- 知道多型的好處後，想建立一個類別「Returnable」，有一個method「doReturn()」，讓可以退貨的Clothing都繼承，來處理退貨問題：

```
public static void returnClothing(Returnable r) {  
    r.doReturn();  
}
```

- 但，Java只允許單一繼承。1個class只能extends一個super-class，不支援多繼承。

使用 interface



介面(interface)介紹

```
[public] [abstract] interface interface_identifier {  
    [public] [static] [final] field declarations;  
    [public] [abstract] method declarations; // java 7: no body  
    [public] default | static method declarations; // java 8: has body  
    private [static] method declarations; // java 9: has body  
}
```

支援版本	存取修飾詞	非存取修飾詞	方法內容區塊{}
Java 8 之前	public (可省略)	abstract (可省略)	N
Java 8 開始	public (可省略)	default	Y
Java 8 開始	public (可省略)	static	
Java 9 開始	private	留空	
Java 9 開始	private	static	



介面(interface)介紹

1. 使用關鍵字「interface」。
2. 子類別只可以extends單一父類別，但可以同時「implements」多個介面，有「實作內容」的概念。
3. 介面可以具備欄位，修飾詞只能是**public**、**static**、**final**或留空，未標明時這些就是預設值。
4. 介面可以具備方法，存取修飾詞可以是**public**、**private**或留空，非存取修飾詞可以是**abstract**、**default**、**static**、**strictfp** (不在本書介紹範圍)或留空。要注意2種修飾詞的搭配組合，如後。



介面(interface)介紹

- 在Java 8之前，唯一允許 **public abstract** 的抽象方法，兩種修飾詞都可以省略。
- 由Java 8開始，
 - 新增 **public default** 的物件方法，要實作方法內容；**public** 修飾詞可以省略，但性質不變。
 - 新增 **public static** 的靜態方法，要實作方法內容；**public** 可以省略，但性質不變。



介面(interface)介紹

— 由Java 9開始，

- 新增 **private** 的物件方法，要實作方法內容；修飾詞不可省略，可以被 public default 方法使用。
- 新增 **private static** 的靜態方法，要實作方法內容；修飾詞皆不可省略，可以被 public static/default 的方法使用。

1	public interface InterfaceModifierLab {
2	<i>/* before Java 8, method has no body */</i>
3	public abstract void abstractMethod1();
4	abstract void abstractMethod2(); //隱含 public
5	void abstractMethod3(); // 隱含public abstract
6	
7	<i>/* after Java 8, default method has body */</i>
8	public default void defaultMethod1() {}
9	default void defaultMethod2() {} //隱含 public
10	
11	<i>/* after Java 8, static method has body */</i>
12	public static void staticMethod1() {}
13	static void staticMethod2() {} //隱含 public
14	
15	<i>/* after Java 9, method with body could be private */</i>
16	private static void privateMethod1() {}
17	private void privateMethod2() {}
18	// private default void privateMethod3() {} //編譯失敗：private與default衝突
19	}

使用介面

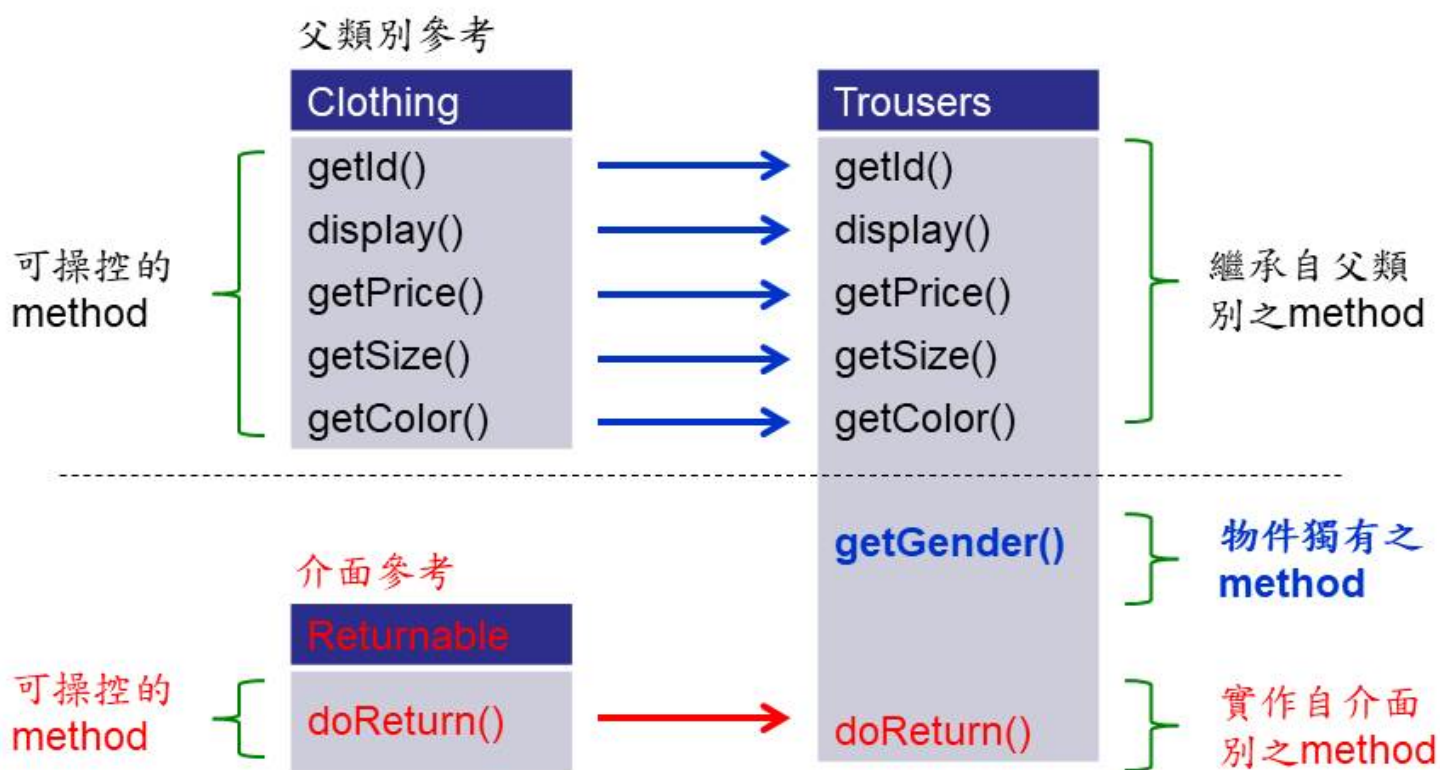
```
public class Shirt extends Clothing implements Returnable {  
    public Shirt(...) {  
        super(...);  
        ...  
    }  
    public void doReturn() {  
        System.out.println("Could be returned within 7 days");  
    }  
    ... //other code  
}
```




```
public static void dealReturn (Returnable r) {  
    r.doReturn();  
}
```

```
public static void main(String[] args) {  
    Shirt r = new Shirt();  
    Returnable t = new Trouser();  
    Sock s = new Sock();  
    dealReturn(r);  
    dealReturn(t);  
    dealReturn(s);  
}
```

由interface參考控制子類別方法



Java 裡的 interface

java.util

Class ArrayList<E>

java.lang.Object
java.util.AbstractCollection<E>
java.util.AbstractList<E>
java.util.ArrayList<E>

All Implemented Interfaces:

Serializable, Cloneable, Iterable<E>, Collection<E>, **List<E>**, RandomAccess

Direct Known Subclasses:

AttributeList, RoleList, RoleUnresolvedList

java.util

Interface List<E>

Type Parameters:

E - the type of elements in this list

All Superinterfaces:

Collection<E>, Iterable<E>

All Known Implementing Classes:

AbstractList, AbstractSequentialList, ArrayList, AttributeList, CopyOnWriteArrayList, LinkedList, RoleList, RoleUnresolvedList, Stack, Vector

Edited by Ruei-Jiun 



6/6

物件始祖【Object】

Edited by Ruei-Jiun Tzeng

物件的始祖

java.util

Class ArrayList<E>

java.lang.Object

java.util.AbstractCollection<E>

java.util.AbstractList<E>

java.util.ArrayList<E>

All Implemented Interfaces:

Serializable, Cloneable, Iterable<E>, Collection<E>, List<E>, RandomAccess

Direct Known Subclasses:

AttributeList, RoleList, RoleUnresolvedList

java.lang

Class Object

java.lang.Object

```
public class Object
```

Class `Object` is the root of the class hierarchy. Every class has `Object` as a superclass. All objects, including arrays, implement the methods of this class.

Since:

JDK1.0

Edited by Ruei-Jiun Tzeng


Object 的地位

- 任何class只要未extends其他類別，java會預設extends Object：

```
public class Clothing {  
    public Clothing (...) {  
        //...  
    }  
    //...  
}
```



```
public class Clothing extends Object {  
    public Clothing (...) {  
        //...  
    }  
    //...  
}
```

toString() method

- 身為類別始祖，Object 類別只提供基本的方法。
- Object 類別的 toString()方法提供對物件的簡單描述：

```
public String toString() {  
    return getClass().getName() + "@" + Integer.toHexString(hashCode());  
}
```

- 通常類別都會改寫此一方法，已提供客製化的個別class訊息。

toString() 範例

```
class First {}

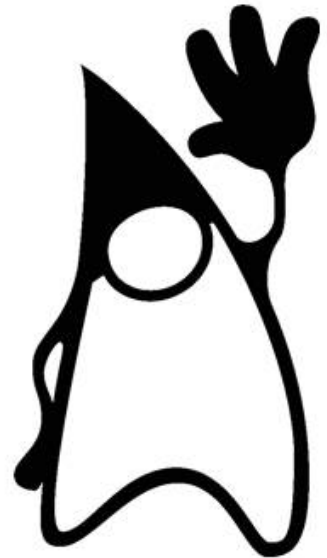
class Second {
    @Override
    public String toString() {
        return "I am Second";
    }
}

public class Test {
    public static void main(String[] args) {
        System.out.println( new Object() );
        System.out.println( new First() );
        System.out.println( new Second() );
        System.out.println( new Second().toString() );
    }
}
```



END ~~

Thank you!!



Edited by Ruei-Jiun Tzeng