

# Java程式設計進階

## 抽象類別及介面

鄭安翔

ansel\_cheng@hotmail.com

---

# 課程大綱

## 1) 抽象方法與抽象類別

- 抽象方法
- 抽象類別
- **Java API** 常見抽象類別

## 2) 介面

# 抽象方法(abstract method)

## ■ 抽象方法

- 只宣告方法定義,而沒有撰寫方法的內容
- 語法:

```
abstract <return_type> <name> ([arguments]);
```

## ■ 用途

- 同樣要有某個方法,但實作方法卻大不相同
- 使多型可以使用
- 抽象方法強迫有繼承關係的子類別去覆寫此方法

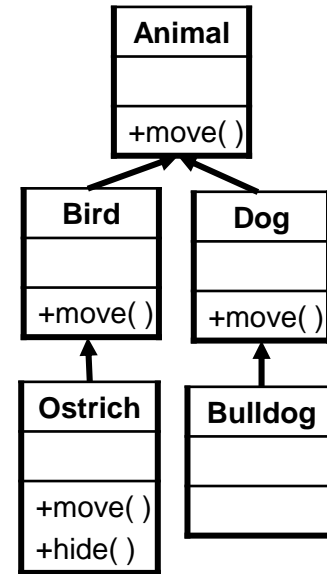
# 抽象方法

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

```
class Dog extends Animal {  
    void move( ) {  
        System.out.println("跑");  
    }  
}
```

```
class Bird extends Animal {  
    void move( ) {  
        System.out.println("飛");  
    }  
}
```

```
class Ostrich extends Bird {  
    void move( ) {  
        System.out.println("跑");  
    }  
    void hide( ) {  
        System.out.println("頭埋在土裡");  
    }  
}
```



```
class Zoo {  
    public static void main(String[ ] args) {  
        Animal a1 = new Dog( );  
        Animal a2 = new Bird( );  
        Animal a3 = new Ostrich( );  
        a1.move( );  
        a2.move( );  
        a3.move( );  
    }  
}
```

# 抽象類別(abstract class)

## ■ 抽象類別

- 類別只要有一個以上的抽象方法,就必須為抽象類別
- 語法:

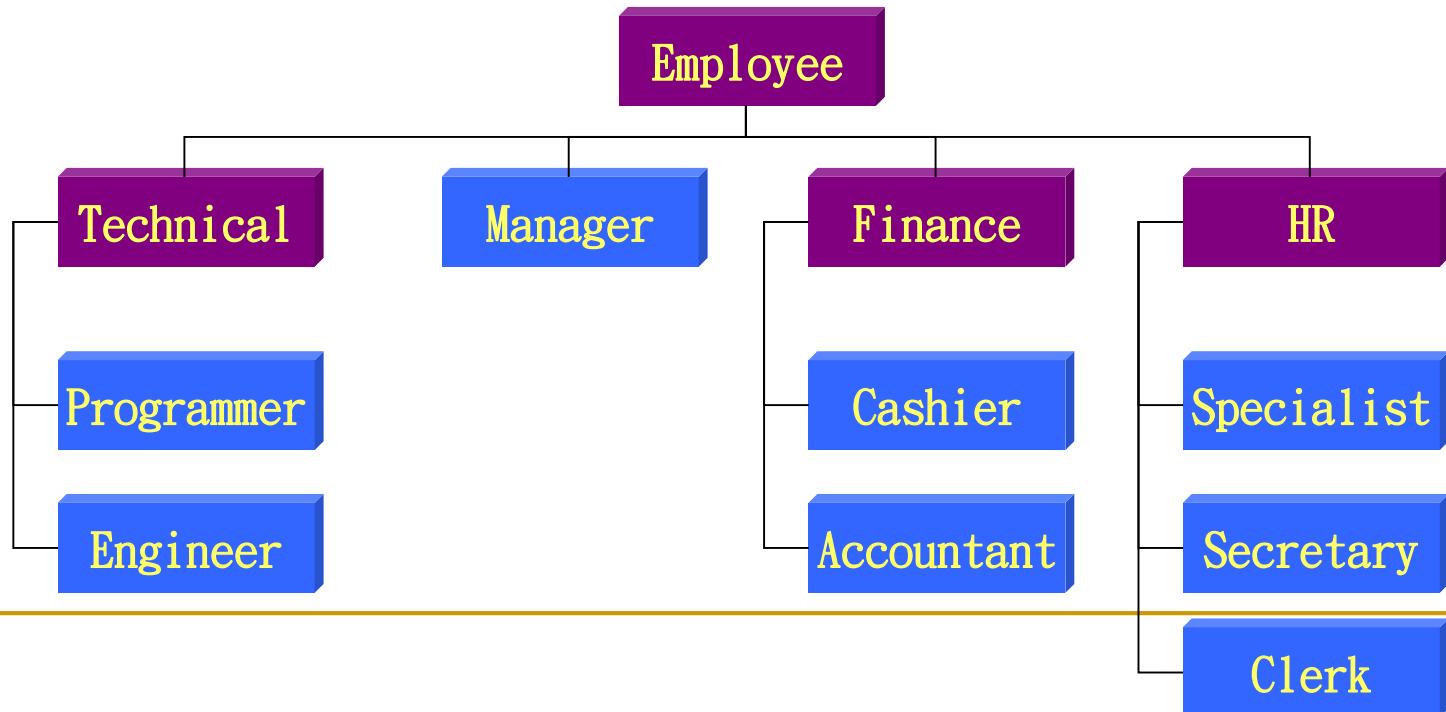
```
abstract class <Name> {  
    .....  
}
```

- 抽象類別不能建構物件實體
- 抽象類別仍可以有屬性，具體方法與建構子
- 建構子用來初始化實體成員，由子類別用 **super( )** 呼叫
- 繼承自抽象類別的子類別一定要實作抽象方法，除非繼續宣告為抽象類別

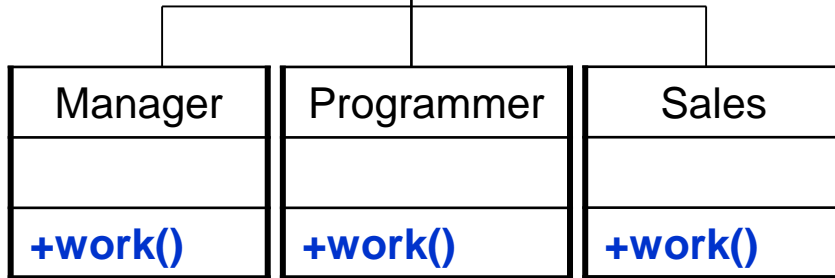
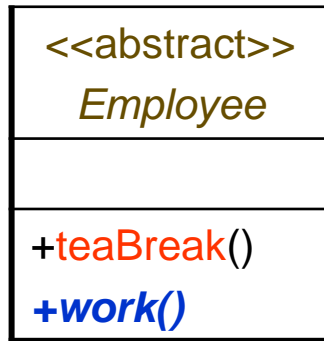
# 抽象類別(abstract class)

## ■ 用途

- 某些類別只是定義一些抽象的分類概念
  - 員工是抽象概念沒有實體
  - 員工的實體必須負責特定工作, **Ex:** 程式設計, 經理, 秘書等



# 抽象類別 - abstract class



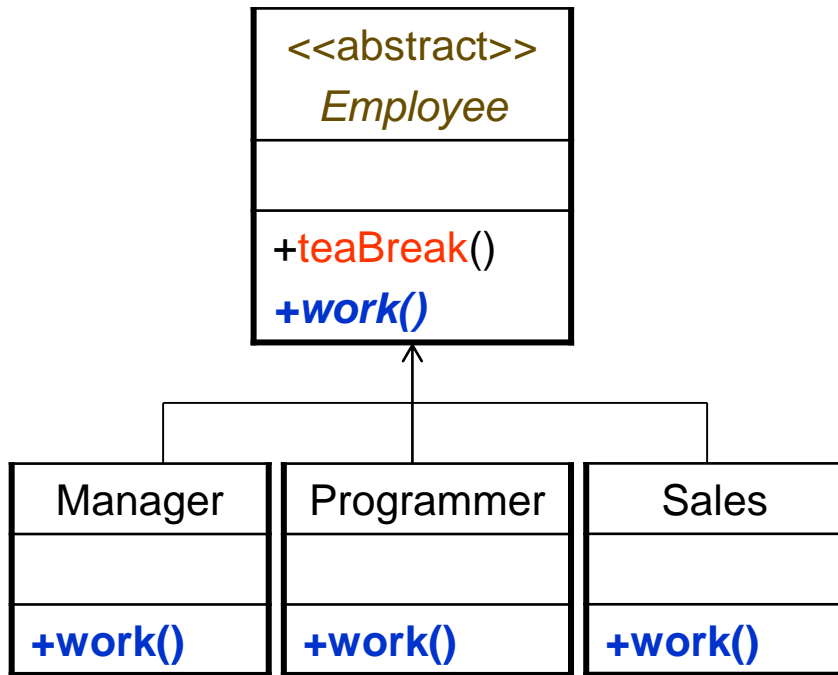
```
01 abstract class Employee {
02     public void teaBreak() {
03         System.out.println("喝咖啡");
04     }
05     abstract void work();
06 }
```

```
01 class Sales extends Employee {
02     public void work() {
03         System.out.println("推銷");
04     }
05 }
```

```
01 class Manager extends Employee {
02     public void work() {
03         System.out.println("開會");
04     }
05 }
```

```
01 class Programmer extends Employee {
02     public void work() {
03         System.out.println("寫 Java Code");
04     }
05 }
```

# 抽象類別多型



```
01 public class EmployeeTest{
02     public static void main(String[] args) {
03
04         Employee manager = new Manager();
05         Employee programmer = new Programmer();
06         Employee sales= new Sales();
07
08         manager.work();
09         programmer.work();
10         sales.work();
11
12         manager.teaBreak();
13         programmer.teaBreak();
14         sales.teaBreak();
15
16     }
17 }
```

The screenshot shows a Windows command prompt window titled "系統管理員: 命令提示字元". The user has entered the following commands:

```
c:\JavaClass>javac EmployeeTest.java
c:\JavaClass>java EmployeeTest
```

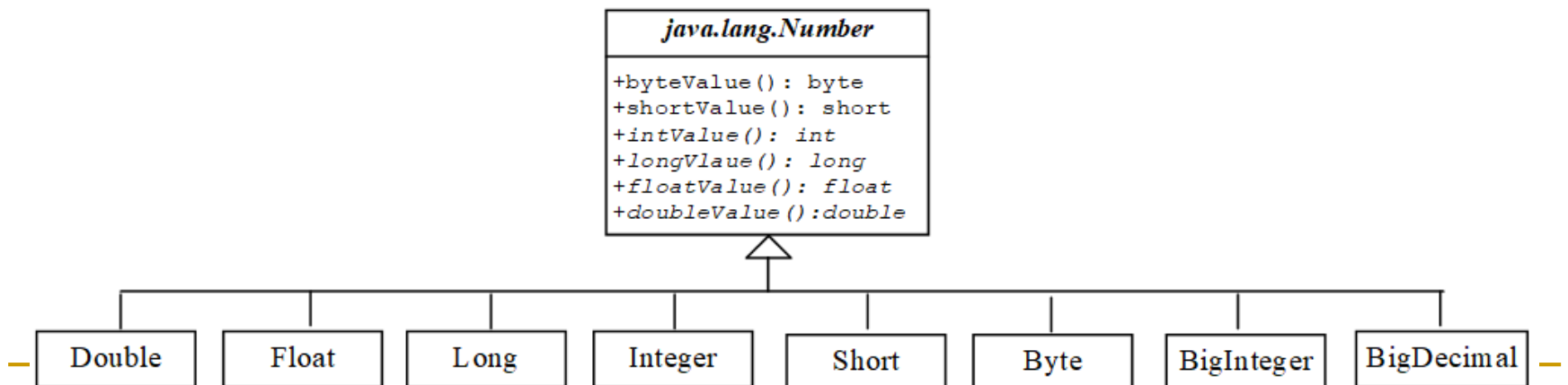
The output of the program is displayed as follows:

```
開會
寫 Java Code
推銷
喝咖啡
喝咖啡
喝咖啡
```



# Java API 常見抽象類別

- 抽象類別 `java.lang.Number`
  - `Byte`、`Short`、`Integer`、`Long`、`Float`、`Double` 等類別的父類別
  - 子類別需提供將數值轉換為 `byte`、`double`、`float`、`int`、`long` 和 `short` 的方法。



# java.lang.Number

java.lang

## 類別 Number

[java.lang.Object](#)

└ [java.lang.Number](#)

所有已實作的介面：

[Serializable](#)

直接已知子類別：

[AtomicInteger](#), [AtomicLong](#), [BigDecimal](#), [BigInteger](#),  
[Byte](#), [Double](#), [Float](#), [Integer](#), [Long](#), [Short](#)

```
public abstract class Number  
extends Object  
implements Serializable
```

## 建構子摘要

[Number](#) ()

## 方法摘要

byte	<a href="#">byteValue</a> () 以 byte 形式返回指定的數值。
abstract double	<a href="#">doubleValue</a> () 以 double 形式返回指定的數值。
abstract float	<a href="#">floatValue</a> () 以 float 形式返回指定的數值。
abstract int	<a href="#">intValue</a> () 以 int 形式返回指定的數值。
abstract long	<a href="#">longValue</a> () 以 long 形式返回指定的數值。
short	<a href="#">shortValue</a> () 以 short 形式返回指定的數值。

# 實作類別

java.lang

## 類別 Integer

```
java.lang.Object
├ java.lang.Number
└ java.lang.Integer
```

所有已實作的介面：

[Serializable](#), [Comparable](#)<[Integer](#)>

```
public final class Integer
extends Number
implements Comparable<Integer>
```

### 方法摘要

byte	<a href="#">byteValue()</a> 以 byte 型別返回該 Integer 的值。
double	<a href="#">doubleValue()</a> 以 double 型別返回該 Integer 的值。
float	<a href="#">floatValue()</a> 以 float 型別返回該 Integer 的值。
int	<a href="#">intValue()</a> 以 int 型別返回該 Integer 的值。
long	<a href="#">longValue()</a> 以 long 型別返回該 Integer 的值。
short	<a href="#">shortValue()</a> 以 short 型別返回該 Integer 的值。

java.lang

## 類別 Double

```
java.lang.Object
├ java.lang.Number
└ java.lang.Double
```

所有已實作的介面：

[Serializable](#), [Comparable](#)<[Double](#)>

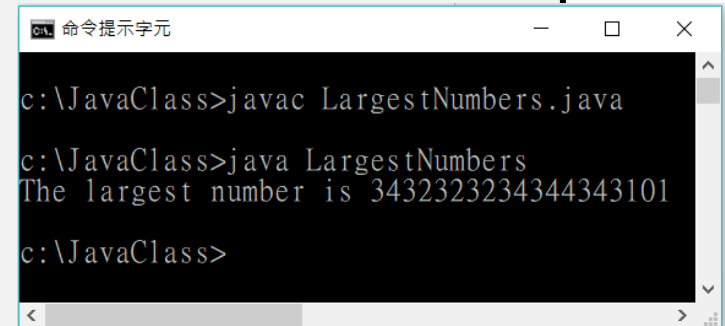
```
public final class Double
extends Number
implements Comparable<Double>
```

### 方法摘要

byte	<a href="#">byteValue()</a> 以 byte 形式返回此 Double 的值（通過強制轉換為 byte）。
double	<a href="#">doubleValue()</a> 返回此 Double 物件的 double 值。
float	<a href="#">floatValue()</a> 返回此 Double 物件的 float 值。
int	<a href="#">intValue()</a> 以 int 形式返回此 Double 的值（通過強制轉換為 int 型別）。
long	<a href="#">longValue()</a> 以 long 形式返回此 Double 的值（通過強制轉換為 long 型別）。
short	<a href="#">shortValue()</a> 以 short 形式返回此 Double 的值（通過強制轉換為 short）。

# Number 抽象類別範例

```
01 import java.math.*;
02 public class LargestNumbers {
03     public static void main(String[ ] args) {
04         Number[ ] nums = new Number[4];
05         nums[0] = new Integer(45); // Add an integer
06         nums[1] = new Double(3445.53); // Add a double
07         nums[2] = new BigInteger("3432323234344343101"); // Add a BigInteger
08         nums[3] = new BigDecimal("2.0909090989091343433344343"); // Add a BigDecimal
09         System.out.println("The largest number is " + getLargestNumber(nums));
10     }
11
12     public static Number getLargestNumber(Number[ ] list) {
13         if (list == null || list.length == 0)
14             return null;
15         Number largestNum = list[0];
16         for (int i=1; i < list.length; i++)
17             if (largestNum.doubleValue( ) < list[i].doubleValue( ))
18                 largestNum = list[i];
19         return largestNum;
20     }
21 }
```



The screenshot shows a command prompt window titled "命令提示字元" (Command Prompt). It displays the following commands and their output:

```
c:\JavaClass>javac LargestNumbers.java
c:\JavaClass>java LargestNumbers
The largest number is 3432323234344343101
c:\JavaClass>
```

# BigDecimal範例

```
01 import java.math.BigDecimal;
02 public class BigDecimalDemo {
03     public static void main(String[] args) {
04         var a = 0.1;
05         var b = 0.1;
06         var c = 0.1;
07         if((a+b+c) == 0.3)
08             System.out.println("a+b+c == 0.3");
09         else
10             System.out.println("a+b+c != 0.3");
11
12         var x = new BigDecimal("0.1");
13         var y = new BigDecimal("0.1");
14         var z = new BigDecimal("0.1");
15         var r = new BigDecimal("0.3");
16         if(x.add(y).add(z).equals(r))
17             System.out.println("x+y+z 等於 "+r.doubleValue());
18         else
19             System.out.println("x+y+z 不等於 "+r.doubleValue());
20     }
21 }
```

命令提示字元

D:\JavaClass\Examples\Ch12>javac BigDecimalDemo.java

D:\JavaClass\Examples\Ch12>java BigDecimalDemo

a+b+c != 0.3

x+y+z 等於 0.3

D:\JavaClass\Examples\Ch12>

# Java API 常見抽象類別 - 日期與日曆

## ■ java.util.Date

- 實例代表某一個特定的時間點，以毫秒為單位表示

## ■ java.util.Calendar

- 抽象類別
- 用來取得詳細的日曆資訊，諸如年份、月份、日期、小時、分鐘數及秒數
- **Calendar**的子類別實作特定的日曆系統，**ex**：陽曆、陰曆及猶太年曆等

## ■ java.util.GregorianCalendar

- 支援陽曆的實作類別

# Calendar 與 GregorianCalendar

## ■ java.util.Calendar 類別

**Calendar** calendar = **Calendar**.getInstance();

static <b>Calendar</b>	<b>getInstance()</b> 使用預設時區和語言環境獲得一個日曆。
static <b>Calendar</b>	<b>getInstance(Locale aLocale)</b> 使用預設時區和指定語言環境獲得一個日曆。
static <b>Calendar</b>	<b>getInstance(TimeZone zone)</b> 使用指定時區和預設語言環境獲得一個日曆。
static <b>Calendar</b>	<b>getInstance(TimeZone zone, Locale aLocale)</b> 使用指定時區和語言環境獲得一個日曆。

## ■ java.util.GregorianCalendar 類別

**Calendar** calendar = new **GregorianCalendar()**;

*java.util.Calendar*

```
#Calendar()
+get(field: int): int
+set(field: int, value: int): void
+set(year: int, month: int,
    dayOfMonth: int): void
+getActualMaximum(field: int): int
+add(field: int, amount: int): void
+getTime(): java.util.Date
+setTime(date: java.util.Date): void
```



*java.util.GregorianCalendar*

```
+GregorianCalendar()
+GregorianCalendar(year: int,
    month: int, dayOfMonth: int)
+GregorianCalendar(year: int,
    month: int, dayOfMonth: int,
    hour: int, minute: int, second: int)
```

# GregorianCalendar 類別

## ■ java.util.GregorianCalendar

### 建構子摘要

#### GregorianCalendar()

在具有預設語言環境的預設時區內使用當前時間建構一個預設的 `GregorianCalendar`。

#### GregorianCalendar(int year, int month, int dayOfMonth)

在具有預設語言環境的預設時區內建構一個帶有給定日期設置的 `GregorianCalendar`。

#### GregorianCalendar(int year, int month, int dayOfMonth, int hourOfDay, int minute, int second)

為具有預設語言環境的預設時區建構一個具有給定日期和時間設置的 `GregorianCalendar`。

**Calendar** calendar = new **GregorianCalendar()**;

- 以目前時間來建構預設的 `GregorianCalendar`

**Calendar** calendar = new **GregorianCalendar(2018, 2, 21)**;

- 以指定年份、月份，以及日期來建構 `GregorianCalendar`。
- `Month` 參數以 0 為基底，也就是說，0 代表一月。



# Calendar 類別中的 get 方法

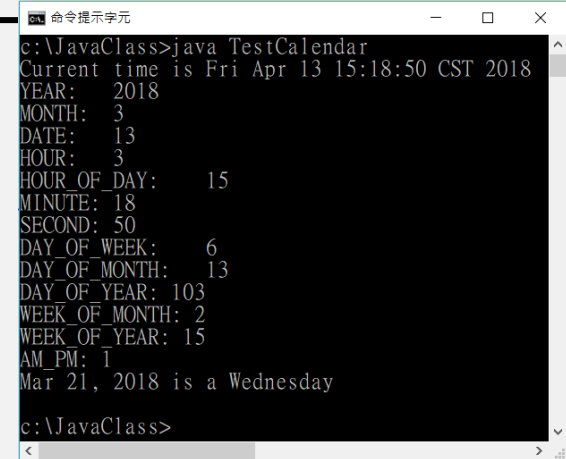
- `get(field: int): int` 方法
  - 從 **Calendar** 物件取出資料與時間資訊
  - **field** 欄位常數，如下表：

常數	說明
YEAR	日曆裡的年份
MONTH	日曆裡的月份， <b>0</b> 代表一月
DATE	日曆裡的日子
HOUR	日曆裡的小時 ( <b>12</b> 小時表示法)
HOUR_OF_DAY	日曆裡的小時 ( <b>24</b> 小時表示法)
MINUTE	日曆裡的分鐘數
SECOND	日曆裡的秒數
DAY_OF_WEEK	一周裡的星期幾， <b>1</b> 代表星期天
DAY_OF_MONTH	跟 <b>DATE</b> 一樣
DAY_OF_YEAR	一年裡的日子數， <b>1</b> 代表一年裡的第一天
WEEK_OF_MONTH	一個月裡的哪一周， <b>1</b> 代表第一周
WEEK_OF_YEAR	一年裡的哪一周， <b>1</b> 代表第一周
AM_PM	AM或PM的指示器 ( <b>0</b> 代表AM， <b>1</b> 代表PM)

# Calendar 抽象類別範例

```
01 import java.util.*;
02 public class TestCalendar {
03     public static void main(String[] args) {
04         // Construct a Gregorian calendar for the current date and time
05         Calendar calendar = new GregorianCalendar();
06         System.out.println("Current time is " + new Date());
07         System.out.println("YEAR:\t" + calendar.get(Calendar.YEAR));
08         System.out.println("MONTH:\t" + calendar.get(Calendar.MONTH));
09         System.out.println("DATE:\t" + calendar.get(Calendar.DATE));
10         System.out.println("HOUR:\t" + calendar.get(Calendar.HOUR));
11         System.out.println("HOUR_OF_DAY:\t" + calendar.get(Calendar.HOUR_OF_DAY));
12         System.out.println("MINUTE:\t" + calendar.get(Calendar.MINUTE));
13         System.out.println("SECOND:\t" + calendar.get(Calendar.SECOND));
14         System.out.println("DAY_OF_WEEK:\t" + calendar.get(Calendar.DAY_OF_WEEK));
15         System.out.println("DAY_OF_MONTH:\t" + calendar.get(Calendar.DAY_OF_MONTH));
16         System.out.println("DAY_OF_YEAR: " + calendar.get(Calendar.DAY_OF_YEAR));
17         System.out.println("WEEK_OF_MONTH: " + calendar.get(Calendar.WEEK_OF_MONTH));
18         System.out.println("WEEK_OF_YEAR: " + calendar.get(Calendar.WEEK_OF_YEAR));
19         System.out.println("AM_PM: " + calendar.get(Calendar.AM_PM));
20         // Construct a calendar for Mar 21, 2018
21         Calendar calendar1 = new GregorianCalendar(2018, 2, 21);
22         System.out.println("March 21, 2018 is a " + dayNameOfWeek(calendar1.get(Calendar.DAY_OF_WEEK)));
23     }
```

```
24     public static String dayNameOfWeek(int dayOfWeek) {
25         switch (dayOfWeek) {
26             case 1: return "Sunday";
27             case 2: return "Monday";
28             case 3: return "Tuesday";
29             case 4: return "Wednesday";
30             case 5: return "Thursday";
31             case 6: return "Friday";
32             case 7: return "Saturday";
33             default: return null;
34         }
35     }
36 }
```



```
c:\JavaClass>java TestCalendar
Current time is Fri Apr 13 15:18:50 CST 2018
YEAR: 2018
MONTH: 3
DATE: 13
HOUR: 3
HOUR_OF_DAY: 15
MINUTE: 18
SECOND: 50
DAY_OF_WEEK: 6
DAY_OF_MONTH: 13
DAY_OF_YEAR: 103
WEEK_OF_MONTH: 2
WEEK_OF_YEAR: 15
AM_PM: 1
Mar 21, 2018 is a Wednesday
c:\JavaClass>
```

---

# 課程大綱

## 1) 抽象方法與抽象類別

## 2) 介面

- 介面宣告
  - 介面實作
  - 介面進階
-

# Interface

## ■ Interface 介面

- ❑ 定義沒有繼承關係的類別有共同的行為或操作方式
- ❑ 溝通的標準或規範：使用介面規範的訊息，可與所有實作介面的物件溝通
- ❑ 介面宣告語法：

```
<modifier> interface <name> extends [interface,]* {  
    <常數宣告>;  
    <方法宣告>;  
}
```

# Interface性質

- **Interface**中所有的方法均為抽象方法
  - **abstract** (一般省略)
  - **public**
- **Interface**中屬性均為類別常數
  - **public**
  - **static**
  - **final** (可省略)
  - 一定要設定初始值

# Interface性質

- Interface沒有建構子
  - 多重繼承
- Interface可以多重繼承其他Interface
- Interface的內容(抽象方法)由Class提供實作

# 介面實作(implements)

## ■ 以類別實作介面

### □ 語法：

```
<modifier> class <name> implements [interface,]* {  
    .....  
}
```

### □ 多重實作

- 類別只能繼承一個父類別，卻可以實作多個介面

### □ 類別如果同時有繼承與實作介面的行為，**extends** 放在 **implements** 之前

### □ 類別必須實作所有介面宣告的方法，否則此類別需宣告為 **abstract** 抽象類別

# 介面多型

## ■ 介面多型

- 將物件以介面的角度來看

父介面 <變數> = new 實作類別( );

- 只能使用該介面定義的方法

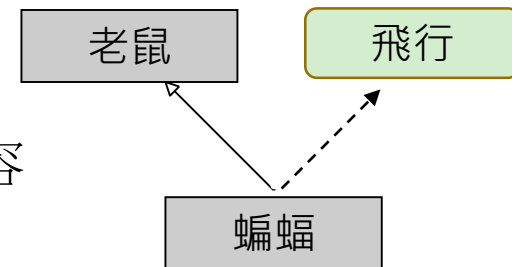
## ■ 介面實作檢查

instanceof

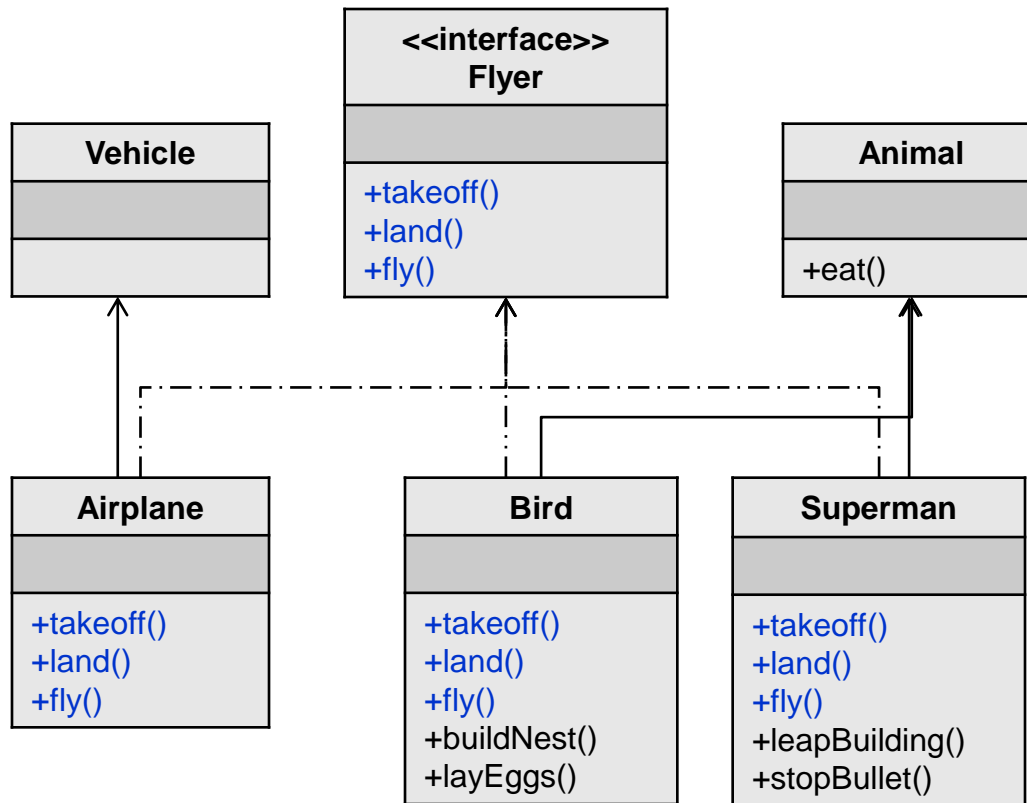


# 介面實作 (implements)

- Java 語言在繼承上只允許單一繼承 (Single Inheritance) 關係
  - 繼承類別表示 "是一種(is-a)" 的關係
- 使用實作介面的方式，達到多型的效果
  - 介面實作表示 "擁有行為" 的關係
  - 避免類別多重繼承時內容衝突的問題
    - 介面沒有建構子、實體屬性及方法實際內容



# 範例 - 介面實作



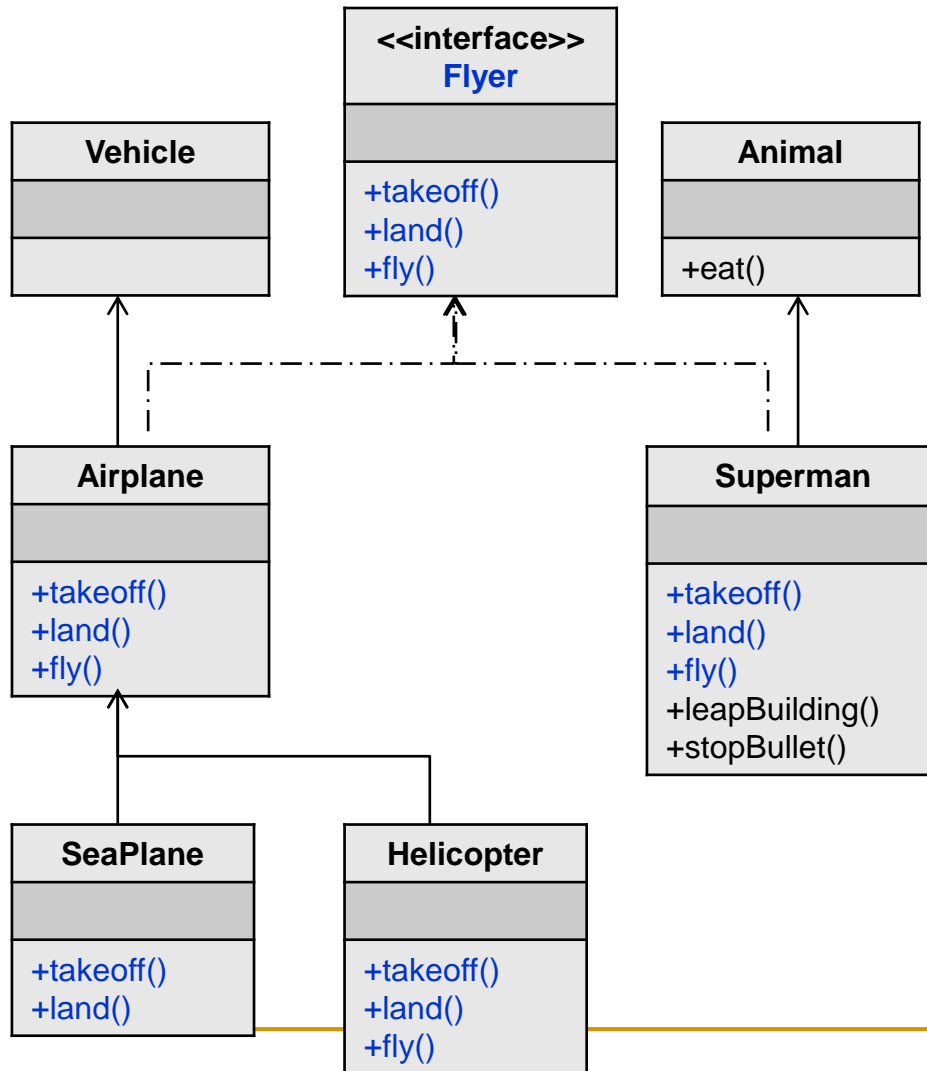
```
01 public interface Flyer {
02     public void takeoff();
03     public void land();
04     public void fly();
05 }
```

```
01 public class Superman extends Animal
02     implements Flyer {
03     public void takeoff() {...}
04     public void land() {...}
05     public void fly() {...}
06     public void leapBuilding() {...}
07     public void stopBullet() {...}
08 }
```

```
01 public class Airplane extends Vehicle
02     implements Flyer {
03     public void takeoff() {...}
04     public void land() {...}
05     public void fly() {...}
06 }
```

```
01 public class Bird extends Animal
02     implements Flyer {
03     public void takeoff() {...}
04     public void land() {...}
05     public void fly() {...}
06     public void buildNest() {...}
07     public void layEggs() {...}
08 }
```

# 範例 - 介面多型

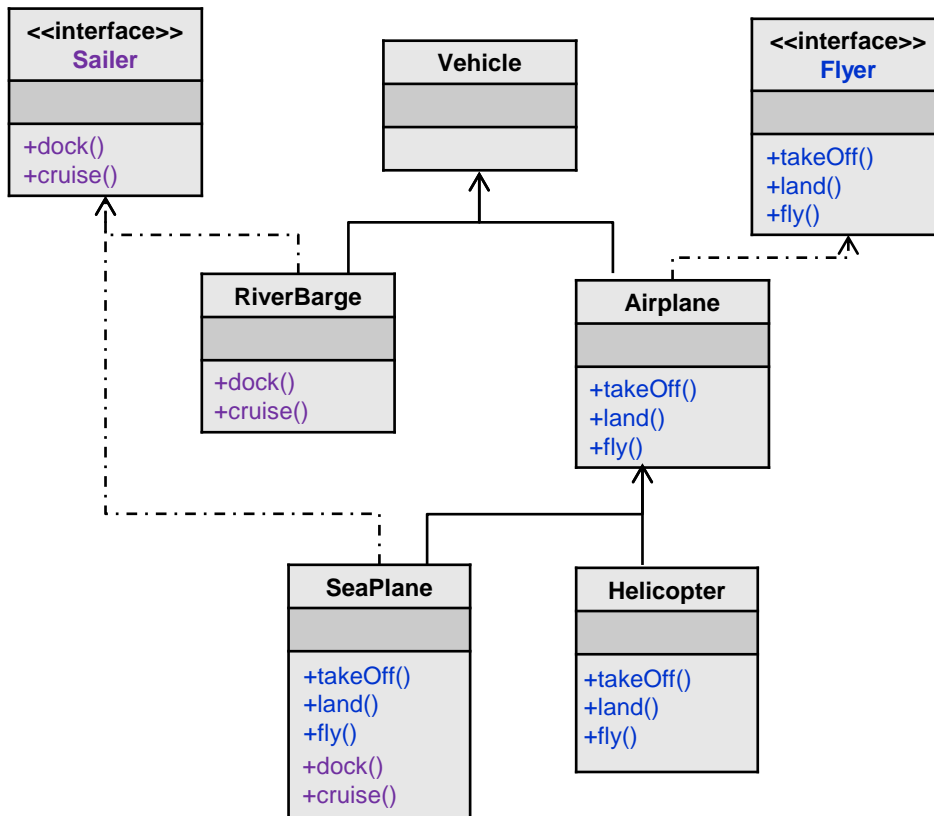


```
01 public class Airport {
02     public static void main(String[] args) {
03         Flyer helicopter = new Helicopter();
04         Flyer seaplane = new SeaPlane();
05         Flyer superman = new Superman();
06
07         helicopter.takeoff();
08         helicopter.fly();
09         helicopter.land();
10
11         seaplane.takeoff();
12         seaplane.fly();
13         seaplane.land();
14
15         superman.takeoff();
16         superman.fly();
17         superman.land();
18     }
19 }
20 }
```

The screenshot shows a Windows command prompt window titled "系統管理員: 命令提...". The command `c:\JavaClass>java Airport` has been executed. The output of the program is displayed in Chinese characters, showing the actions of the helicopter, seaplane, and superman. The output is as follows:

```
c:\JavaClass>java Airport
直昇機起飛!
直昇機飛行!
直昇機降落!
水上飛機起飛!
飛機飛行!
水上飛機降落!
超人起飛!
超人飛行!
超人降落!
```

# 範例 - 多重介面



```
01 public class Harbor {
02     public static void main(String[] args) {
03         Harbor Taichung = new Harbor();
04
05         RiverBarge riverBarge = new RiverBarge();
06         SeaPlane seaplane = new SeaPlane();
07
08         Taichung.givePermissionToDock(riverBarge);
09         Taichung.givePermissionToDcok(seaplane);
10     }
11
12     private void givePermissionToDock(Sailer s) {
13         s.dock();
14     }
15 }
```

SeaPlane可由桃園機場起飛停泊在台中港

# 常用 Abstract Class & 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

```
public class ArrayList<E>  
extends AbstractList<E>  
implements List<E>, RandomAccess, Cloneable, Serializable
```

Resizable-array implementation of the List interface. Implements all optional list operations, and permits all elements, including null. In addition to implementing the List interface, this class provides methods to manipulate the size of the array that is used internally to store the list. (This class is roughly equivalent to Vector, except that it is unsynchronized.)

java.util

## Class AbstractList<E>

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

### All Implemented Interfaces:

Iterable<E>, Collection<E>, List<E>

### Direct Known Subclasses:

AbstractSequentialList, ArrayList, Vector

```
public abstract class AbstractList<E>  
extends AbstractCollection<E>  
implements List<E>
```

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:

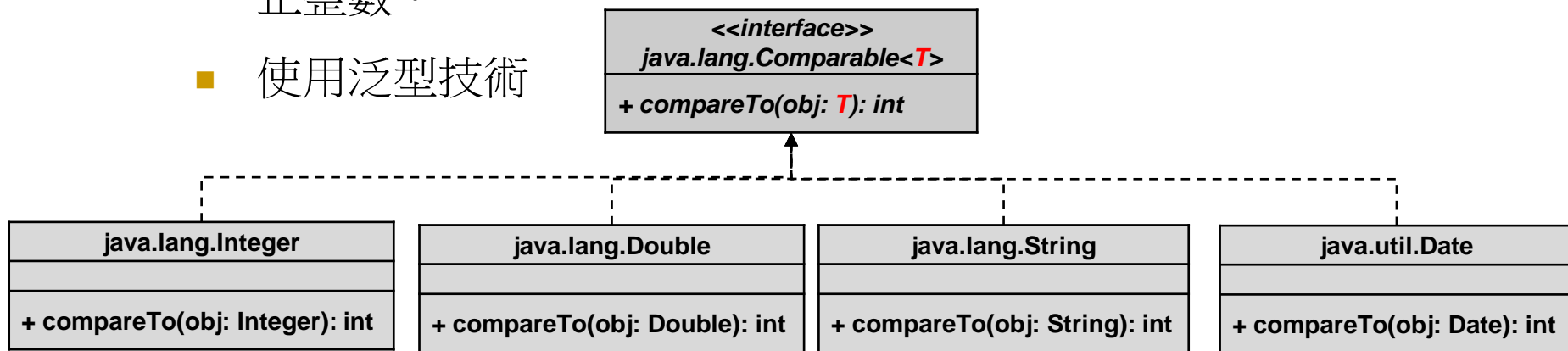
AbstractList, AbstractSequentialList, ArrayList, AttributeList, CopyOnWriteArrayList, LinkedList, RoleList, RoleUnresolvedList, Stack, Vector

```
public interface List<E>  
extends Collection<E>
```

# Java API 常見介面

## ■ java.lang.Comparable 介面

- 實作此介面的陣列，可使用 Arrays 或 Collections 的 sort() 方法自動排序
- compareTo(obj: T): int 提供物件排序的方法
  - 本身物件小於、等於或大於傳入物件，分別傳回負整數、零或正整數。
  - 使用泛型技術



# java.util.Arrays

java.util

## 類別 Arrays

[java.lang.Object](#)

└ java.util.Arrays

```
public class Arrays  
extends Object
```

此類別包含用來操作陣列（比如排序和搜尋）的各種方法。此類別還包含一個允許將陣列作為列表來查看的靜態處理器。

```
static void sort(Object[] a)
```

根據元素的[自然順序](#)對指定物件陣列按升序進行排序。

# java.lang.Comparable

java.lang

## 介面 Comparable<T>

型別參數：

T - 可以與此物件進行比較的那些物件的型別

所有已知子介面：

[Delayed](#), [Name](#), [RunnableScheduledFuture<V>](#), [ScheduledFuture<V>](#)

所有已知實作類別：

[Authenticator.RequestorType](#), [BigDecimal](#), [BigInteger](#), [Boolean](#), [Byte](#), [ByteBuffer](#), [Calendar](#), [Character](#), [CharBuffer](#), [Charset](#), [ClientInfoStatus](#), [CollationKey](#), [Component.BaselineResizeBehavior](#), [CompositeName](#), [CompoundName](#), [Date](#), [Date](#), [Desktop.Action](#), [Diagnostic.Kind](#), [Dialog.ModalExclusionType](#), [Dialog.ModalityType](#), [Double](#), [DoubleBuffer](#), [DropMode](#), [ElementKind](#), [ElementType](#), [Enum](#), [File](#), [Float](#), [FloatBuffer](#), [Formatter.BigDecimalLayoutForm](#), [FormSubmitEvent.MethodType](#), [GregorianCalendar](#), [GroupLayout.Alignment](#), [IntBuffer](#), [Integer](#), [JavaFileObject.Kind](#), [JTable.PrintMode](#), [KeyRep.Type](#), [LayoutStyle.ComponentPlacement](#), [LdapName](#), [Long](#), [LongBuffer](#), [MappedByteBuffer](#), [MemoryType](#), [MessageContext.Scope](#), [Modifier](#), [MultipleGradientPaint.ColorSpaceType](#), [MultipleGradientPaint.CycleMethod](#), [NestingKind](#), [Normalizer.Form](#), [ObjectName](#), [ObjectStreamField](#), [Proxy.Type](#), [Rdn](#), [Resource.AuthenticationType](#), [RetentionPolicy](#), [RoundingMode](#), [RowFilter.ComparisonType](#), [RowIdLifetime](#), [RowSorterEvent.Type](#), [Service.Mode](#), [Short](#), [ShortBuffer](#), [SOAPBinding.ParameterStyle](#), [SOAPBinding.Style](#), [SOAPBinding.Use](#), [SortOrder](#), [SourceVersion](#), [SSLEngineResult.HandshakeStatus](#), [SSLEngineResult.Status](#), [StandardLocation](#), [String](#), [SwingWorker.StateValue](#), [Thread.State](#), [Time](#), [Timestamp](#), [TimeUnit](#), [TrayIcon.MessageType](#), [TypeKind](#), [URI](#), [UUID](#), [WebParam.Mode](#), [XmlAccessOrder](#), [XmlAccessType](#), [XmlNsForm](#)

## 方法摘要

int	<a href="#">compareTo</a> (T o) 比較此物件與指定物件的順序。
-----	---

```
public interface Comparable<T>
```



# 實作類別

java.lang

## 類別 Integer

[java.lang.Object](#)

└ [java.lang.Number](#)

└ **java.lang.Integer**

所有已實作的介面：

[Serializable](#), [Comparable](#)<[Integer](#)>

---

```
public final class Integer
extends Number
implements Comparable<Integer>
```

### 方法摘要

int	<a href="#">compareTo</a> ( <a href="#">Integer</a> anotherInteger) 在數字上比較兩個 Integer 物件。
-----	---

java.lang

## 類別 Double

[java.lang.Object](#)

└ [java.lang.Number](#)

└ **java.lang.Double**

所有已實作的介面：

[Serializable](#), [Comparable](#)<[Double](#)>

---

```
public final class Double
extends Number
implements Comparable<Double>
```

### 方法摘要

int	<a href="#">compareTo</a> ( <a href="#">Double</a> anotherDouble) 對兩個 Double 物件所表示的數值進行比較。
-----	---

# 實作類別

java.lang

## 類別 String

[java.lang.Object](#)

└ [java.lang.String](#)

所有已實作的介面：

[Serializable](#), [CharSequence](#), [Comparable<String>](#)

```
public final class String
extends Object
implements Serializable, Comparable<String>, CharSequence
```

### 方法摘要

int	<a href="#">compareTo</a> ( <a href="#">String</a> anotherString) 按字典順序比較兩個字元串。
-----	--

java.util

## 類別 Date

[java.lang.Object](#)

└ [java.util.Date](#)

所有已實作的介面：

[Serializable](#), [Cloneable](#), [Comparable<Date>](#)

直接已知子類別：

[Date](#), [Time](#), [Timestamp](#)

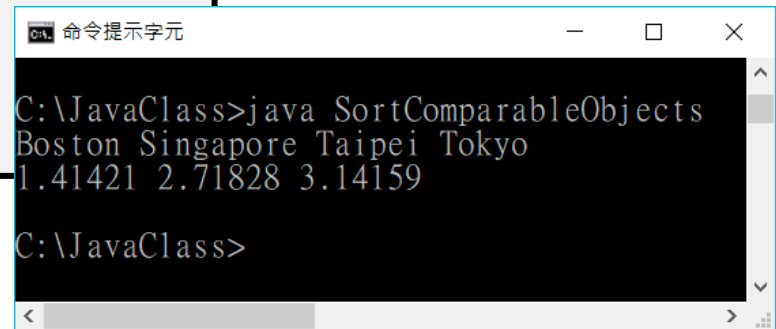
```
public class Date
extends Object
implements Serializable, Cloneable, Comparable<Date>
```

### 方法摘要

int	<a href="#">compareTo</a> ( <a href="#">Date</a> anotherDate) 比較兩個日期的順序。
-----	---

# Comparable 介面範例

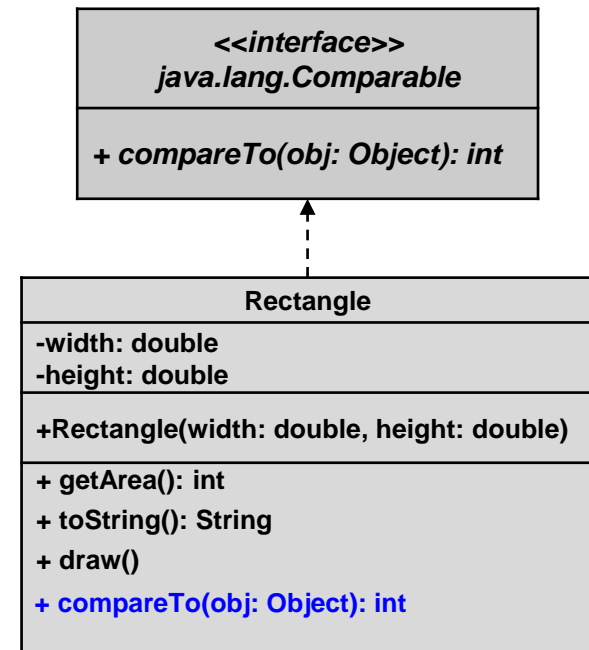
```
01 public class SortComparableObjects {
02     public static void main(String[] args) {
03         String[] cities = {"Taipei", "Boston", "Tokyo", "Singapore"};
04         java.util.Arrays.sort(cities);
05         for (String city: cities)
06             System.out.print(city + " ");
07         System.out.println();
08
09         Double[] doubleNumbers = {new Double("3.14159"),
10             new Double("1.41421"), new Double("2.71828")};
11         java.util.Arrays.sort(doubleNumbers);
12         for (Double number: doubleNumbers)
13             System.out.print(number + " ");
14         System.out.println();
15     }
16 }
```



```
命令提示字元
C:\JavaClass>java SortComparableObjects
Boston Singapore Taipei Tokyo
1.41421 2.71828 3.14159
C:\JavaClass>
```

# 實作 Comparable 介面

```
01 public class Rectangle {
02     private double width;
03     private double height;
04     public Rectangle(double width, double height) {
05         this.width = width;
06         this.height = height;
07     }
08     public double getArea() {
09         return width * height;
10     }
11     @Override
12     public String toString() {
13         return "Width: " + width + " Height: " + height
14             + " Area: " + getArea();
15     }
16 }
```



```

01 public class Rectangle implements Comparable {
02     private double width;
03     private double height;
04     public Rectangle(double width, double height) {
05         this.width = width;
06         this.height = height;
07     }
08     public double getArea() {
09         return width * height;
10     }
11     @Override
12     public String toString() {
13         return "Width: " + width + " Height: " + height
14             + " Area: " + getArea();
15     }
16     @Override
17     public int compareTo(Object obj) {
18         Rectangle o = (Rectangle)obj;
19         if (getArea() > o.getArea())
20             return 1;
21         else if (getArea() < o.getArea())
22             return -1;
23         else
24             return 0;
25     }

```

```

01 public class SortRectangles {
02     public static void main(String[] args) {
03         Rectangle[] rectangles = {
04             new Rectangle(3.4, 5.4),
05             new Rectangle(13.24, 55.4),
06             new Rectangle(7.4, 35.4),
07             new Rectangle(1.4, 25.4));
08         java.util.Arrays.sort(rectangles);
09         for (Rectangle rectangle: rectangles) {
10             System.out.print(rectangle + " ");
11             System.out.println();
12         }
13     }
14 }

```

```

選取 命令提示字元
C:\JavaClass>javac SortRectangles.java
C:\JavaClass>java SortRectangles
Width: 3.4 Height: 5.4 Area: 18.36
Width: 1.4 Height: 25.4 Area: 35.559999
Width: 7.4 Height: 35.4 Area: 261.96
Width: 13.24 Height: 55.4 Area: 733.496
C:\JavaClass>

```

# Abstract Class vs. Interface

	Abstract Class	Interface
用途	當子類別的行為有極大的差異性，無法於父類別定義，由子類別去覆寫共同的方法	讓沒有繼承關係的類別，擁有相同的存取方式
抽象方法	不一定	全部
屬性	沒有限制	public static final
建構子	有	無
繼承關係	類別繼承類別	介面繼承介面
	單一繼承	多重繼承
實作關係	無	類別實作介面

# 用 interface 解決問題

## ■ 用 interface 解決問題

- 一家五金公司販售差異性極高的幾種商品
  - 碎石 (Rocks, 以磅計重)
  - 塗料 (Paint, 以加侖計算容積)
  - 小零件 (Widgets, 以個數計算)
- 希望在財務報表上有一致的呈現方式
  - 售價 (Sales Price)
  - 成本 (Cost)
  - 利潤 (Profit)

# 範例 – 用interface 解決問題

Rock
- name:String - unitPrice:double - unitCost:double - weight:double
+ Rock(p:double, c:double, w: double)

Paint
- name:String - unitPrice:double - unitCost:double - volume:double
+ Paint(p:double, c:double, v: double)

Widget
- name:String - unitPrice:double - unitCost:double - quantity:int
+ Paint(p:double, c:double, q: int)

```
public class Rock {  
    private String name = "Rock";  
    private double unitPrice;  
    private double unitCost;  
    private double weight;  
  
    public Rock(double p, double c, double w) {  
        unitPrice = p;  
        unitCost = c;  
        weight = w;  
    }  
}
```

```
public class Paint {  
    private String name = "Paint";  
    private double unitPrice;  
    private double unitCost;  
    private double volume;  
  
    public Paint(double p, double c, double v) {  
        unitPrice = p;  
        unitCost = c;  
        volume = v;  
    }  
}
```

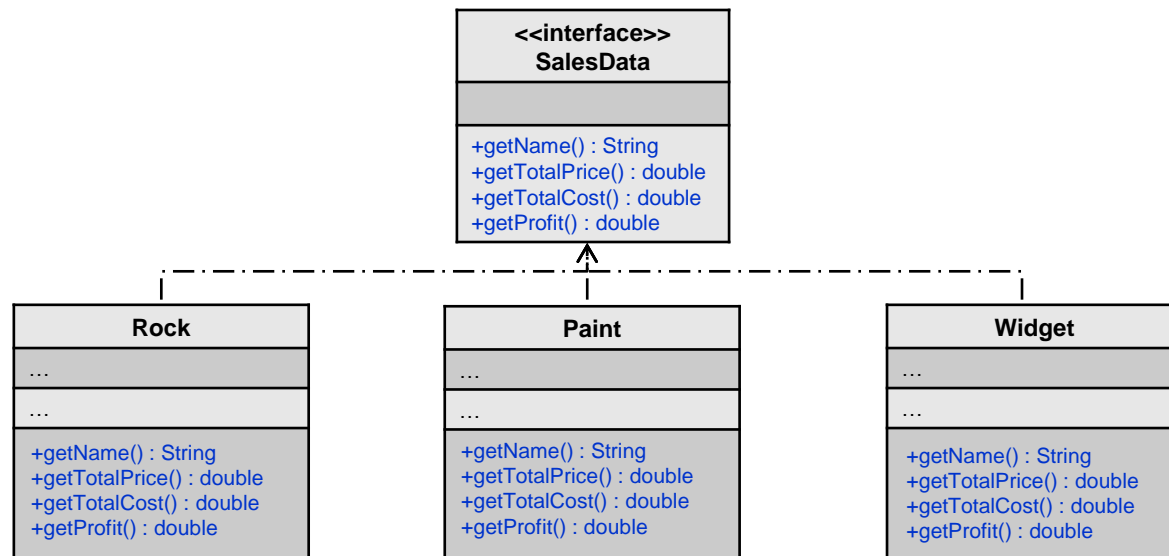
```
public class Widget {  
    private String name = "Widget";  
    private double unitPrice;  
    private double unitCost;  
    private int quantity;  
  
    public Widget(double p, double c, int q) {  
        unitPrice = p;  
        unitCost = c;  
        quantity = q;  
    }  
}
```



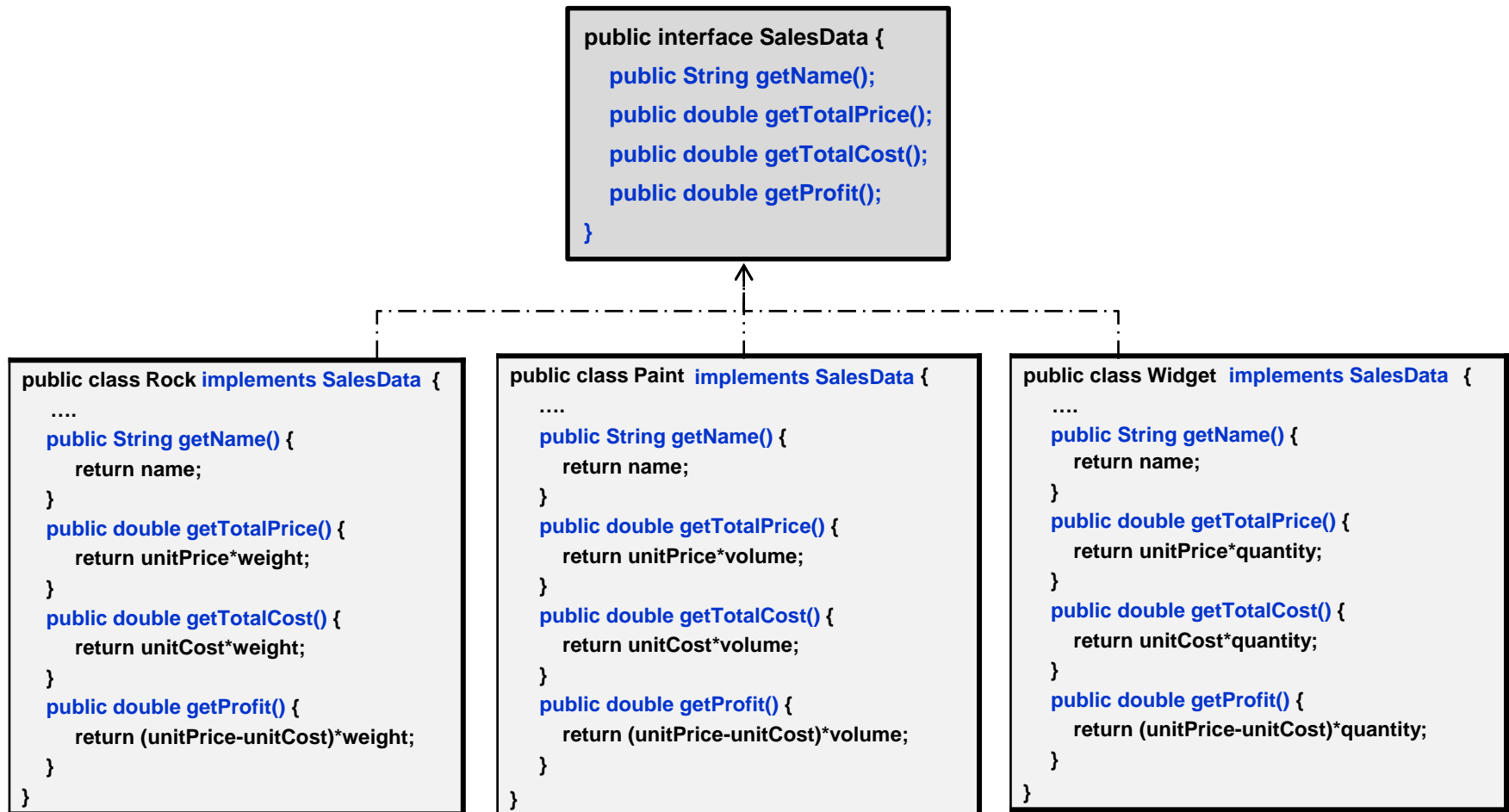
# 範例 – 用interface 解決問題

## ■ SalesData interface

- 定義在財務表報上需要出現的資訊
- 所有商品均需要實作SalesData介面以提供資訊

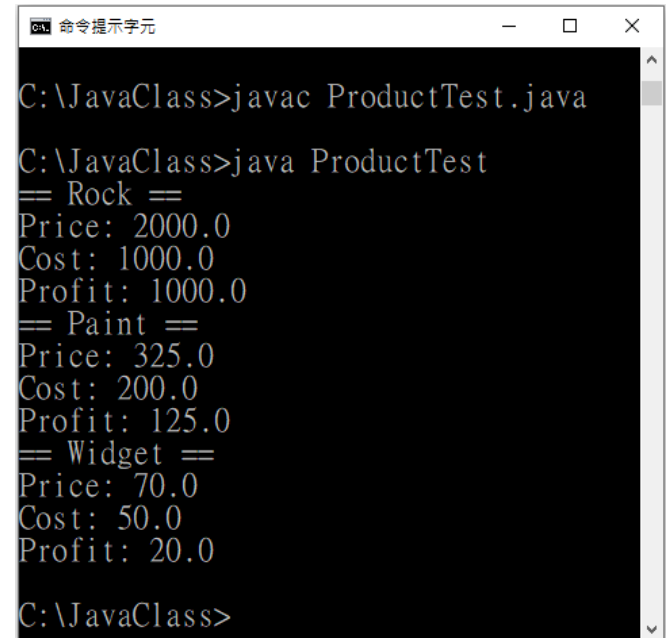


# 範例 – 用interface 解決問題



# 範例 – 用interface 解決問題

```
public class ProductTest {  
    public static void main(String args[]) {  
        Rock rock = new Rock(20, 10, 100);  
        Paint paint = new Paint(13.0, 8.0, 25.0);  
        Widget widget = new Widget(7.0, 5.0, 10);  
  
        System.out.println("== " + rock.getName() + " ==");  
        System.out.println("Price: " + rock.getTotalPrice());  
        System.out.println("Cost: " + rock.getTotalCost());  
        System.out.println("Profit: " + rock.getProfit());  
  
        System.out.println("== " + paint.getName() + " ==");  
        System.out.println("Price: " + paint.getTotalPrice());  
        System.out.println("Cost: " + paint.getTotalCost());  
        System.out.println("Profit: " + paint.getProfit());  
  
        System.out.println("== " + widget.getName() + " ==");  
        System.out.println("Price: " + widget.getTotalPrice());  
        System.out.println("Cost: " + widget.getTotalCost());  
        System.out.println("Profit: " + widget.getProfit());  
    }  
}
```



A screenshot of a Windows command prompt window titled "命令提示字元". The window shows the following commands and output:

```
C:\JavaClass>javac ProductTest.java  
  
C:\JavaClass>java ProductTest  
== Rock ==  
Price: 2000.0  
Cost: 1000.0  
Profit: 1000.0  
== Paint ==  
Price: 325.0  
Cost: 200.0  
Profit: 125.0  
== Widget ==  
Price: 70.0  
Cost: 50.0  
Profit: 20.0  
  
C:\JavaClass>
```

# 工具類別提供一致的呈現方式

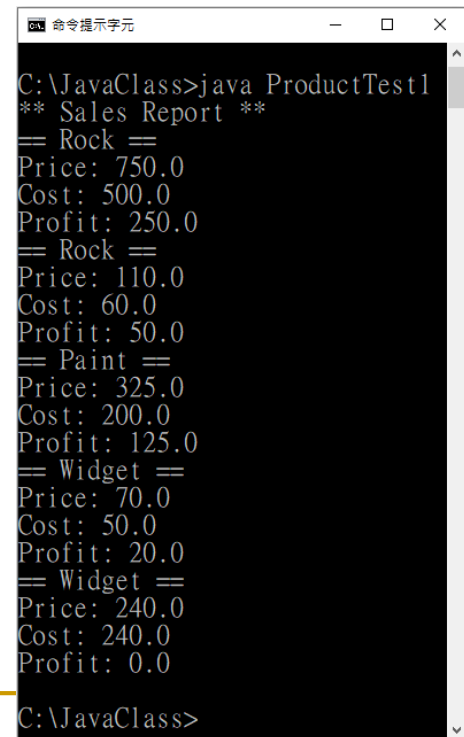
- 工具類別utility class :
  - 產出商品報表
    - 將多個操作整合成一個標準顯示格式
  - show()傳入SalesData介面的實作

```
public class SalesTool {  
    public void show (SalesData item) {  
        System.out.println("== " + item.getName() + " ==");  
        System.out.println("Price: " + item.getTotalPrice());  
        System.out.println("Cost: " + item.getTotalCost());  
        System.out.println("Profit: " + item.getProfit());  
    }  
}
```

# 工具類別提供一致的呈現方式

- 各式商品均實作 **SalesData** 介面
- 使用 **SalesData** 作為各式商品的參考型別

```
public class ProductTest1 {  
    public static void main(String args[]) {  
        SalesData[] itemList = new SalesData[5];  
        SalesTool tool = new SalesTool();  
        itemList[0] = new Rock(15.0, 10.0, 50.0);  
        itemList[1] = new Rock(11.0, 6.0, 10.0);  
        itemList[2] = new Paint(13.0, 8.0, 25.0);  
        itemList[3] = new Widget(7.0, 5.0, 10);  
        itemList[4] = new Widget(12.0, 12.0, 20);  
        System.out.println("** Sales Report **");  
        for (SalesData item : itemList) {  
            tool.show(item);  
        }  
    }  
}
```



```
C:\JavaClass>java ProductTest1  
** Sales Report **  
== Rock ==  
Price: 750.0  
Cost: 500.0  
Profit: 250.0  
== Rock ==  
Price: 110.0  
Cost: 60.0  
Profit: 50.0  
== Paint ==  
Price: 325.0  
Cost: 200.0  
Profit: 125.0  
== Widget ==  
Price: 70.0  
Cost: 50.0  
Profit: 20.0  
== Widget ==  
Price: 240.0  
Cost: 240.0  
Profit: 0.0  
C:\JavaClass>
```

# 工具類別提供一致的呈現方式

- SalesTool 工具類別
  - 只有一個方法
  - 只用來處理實作 SalesData 的商品
- 將方法移至 SalesData interface 中
  - default 方法
  - static 方法

# 介面 default 方法

- Java 8 介面中可定義 default 方法
  - 使用 default 關鍵字修飾方法。
    - 擁有實作內容
  - default 方法可被實作子類別的物件使用
  - 事後加入不影響子類別
    - default 方法已經有內容，不會強制子類別必須實作該方法
  - 非 default 方法不能有 {}

# default 方法

```
public interface SalesData {  
    public String getName();  
    public double getTotalPrice();  
    public double getTotalCost();  
    public double getProfit();  
    public default void show() {  
        System.out.println("== " + this.getName() + " ==");  
        System.out.println("Price: " + this.getTotalPrice());  
        System.out.println("Cost: " + this.getTotalCost());  
        System.out.println("Profit: " + this.getProfit());  
    }  
}
```

```
public class ProductTest2 {  
    public static void main(String args[]) {  
        SalesData[] itemList = new SalesData[5];  
        //SalesTool tool = new SalesTool();  
        itemList[0] = new Rock(15.0, 10.0, 50.0);  
        itemList[1] = new Rock(11.0, 6.0, 10.0);  
        itemList[2] = new Paint(13.0, 8.0, 25.0);  
        itemList[3] = new Widget(7.0, 5.0, 10);  
        itemList[4] = new Widget(12.0, 12.0, 20);  
        System.out.println("*** Sales Report ***");  
        for (SalesData item : itemList)  
            item.show();  
    }  
}
```



# static 方法

## ■ Java 8 介面中可定義 static 方法

- 以介面名稱呼叫類別方法
- 允許有實作內容
- 傳入要處理的物件

```
public interface SalesData {  
    ....  
  
    public static void show(SalesData item) {  
        System.out.println("== " + item.getName() + " ==");  
        System.out.println("Price: " + item.getTotalPrice());  
        System.out.println("Cost: " + item.getTotalCost());  
        System.out.println("Profit: " + item.getProfit());  
    }  
}
```

```
public class ProductTest3 {  
    public static void main(String args[]) {  
        SalesData[] itemList = new SalesData[5];  
        itemList[0] = new Rock(15.0, 10.0, 50.0);  
        itemList[1] = new Rock(11.0, 6.0, 10.0);  
        itemList[2] = new Paint(13.0, 8.0, 25.0);  
        itemList[3] = new Widget(7.0, 5.0, 10);  
        itemList[4] = new Widget(12.0, 12.0, 20);  
        System.out.println("*** Sales Report ***");  
        for (SalesData item : itemList)  
            SalesData.show(item);  
    }  
}
```