Java 平台概論

特色

物件導向

完全支援網際網路

- Java 是因應 www 風潮所設計的，所以他當然要能支援網際網路的所有功能，Java 撰寫出的程式可以很容易地在瀏覽器中，像 IE、chrome、firefox 中呈現。

跨平台

- 由 Java 語言所撰寫的程式，可以在不修改 code 的情況下，就能在不同作業系統執行。

豐富函數庫

- Java 被豐富的函數庫支撐著，像繪圖函數庫，圖形使用者介面函數庫，網路設計函數庫等，這些對於開發者來說，都是很方便的資源。

特殊處理機制：多執行緒(multi-thread)、垃圾收集(garbage collection)、例外處理(exception handling)

- 它有許多特殊處理機制，例如「多執行緒」機制可以在同時間內執行不同的程序、「垃圾收集」可以將無用的變數所占用的記憶體釋放，才不會爆掉、「例外處理」機制則可在程式碰到非正常處理狀況時，依情況拋出例外，使程式不會因此中斷執行。

版本

版本	代碼名稱	年份
JDK Beta		1994
JDK 1.0		1996
JDK 1.1		1997
J2SE 1.2		1998
J2SE 1.3		2000
J2SE 1.4		2002
J2SE 5.0	Tiger	2005
Java SE 6	Mustang	2006
Java SE 7	Dolphin	2011
Java SE 8		2014

三大平台

Java SE(J2SE)

JVM

- Java 程式執行環境

JRE

- 運行 Java 程式

JDK

- 開發 Java 程式

Java 語言

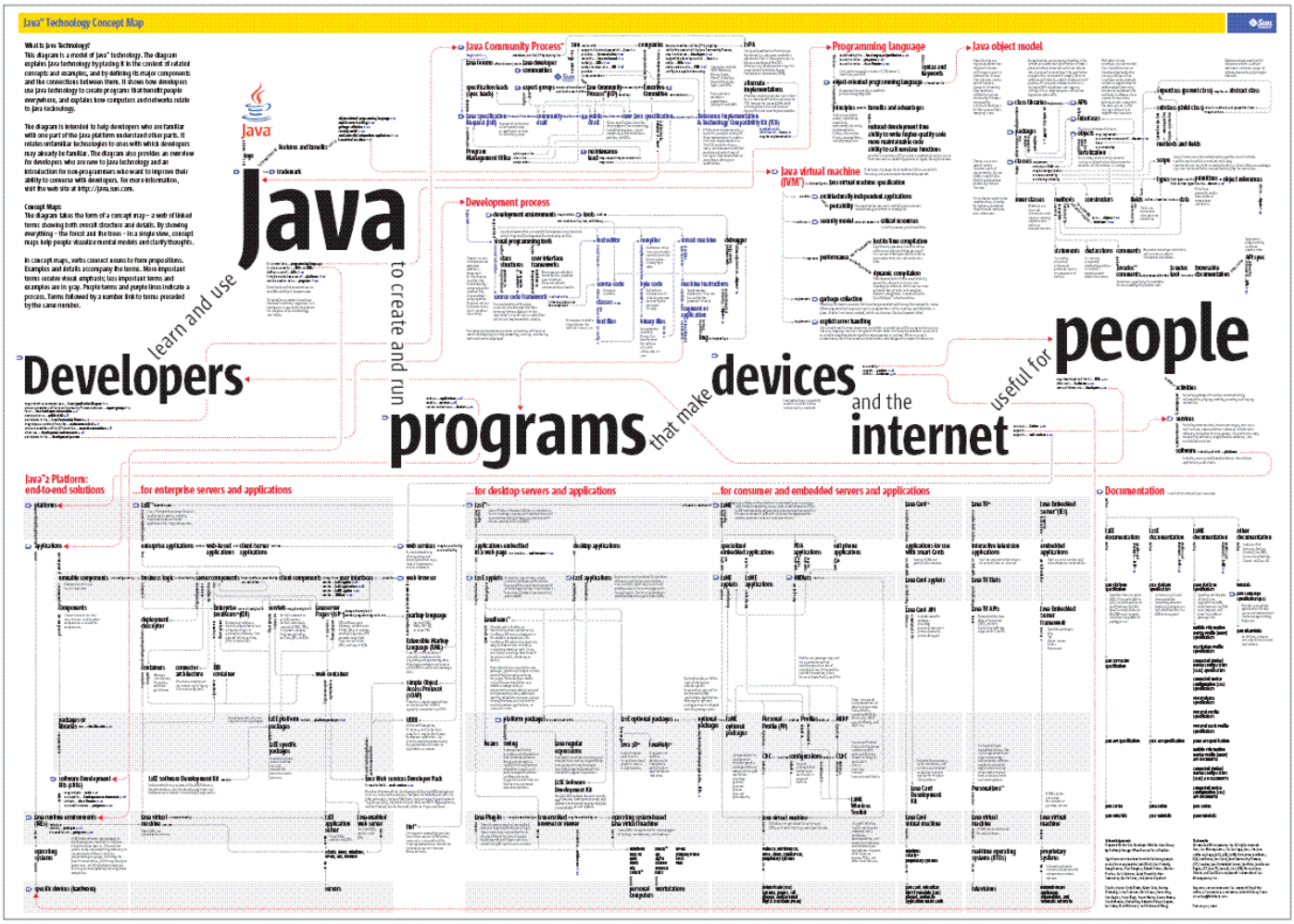
Java EE(J2EE)

- 以 Java SE 為基礎，定義了一系列的服務、API、協定等，適用於開發分散式、多層式、以元件為基礎、以 Web 為基礎的應用程式
- 技術：JSP、Servlet、JavaMail、EJB 等

Java ME(J2ME)

- 作為小型數位設備上開發及部署應用程式的平台

建議學習路徑

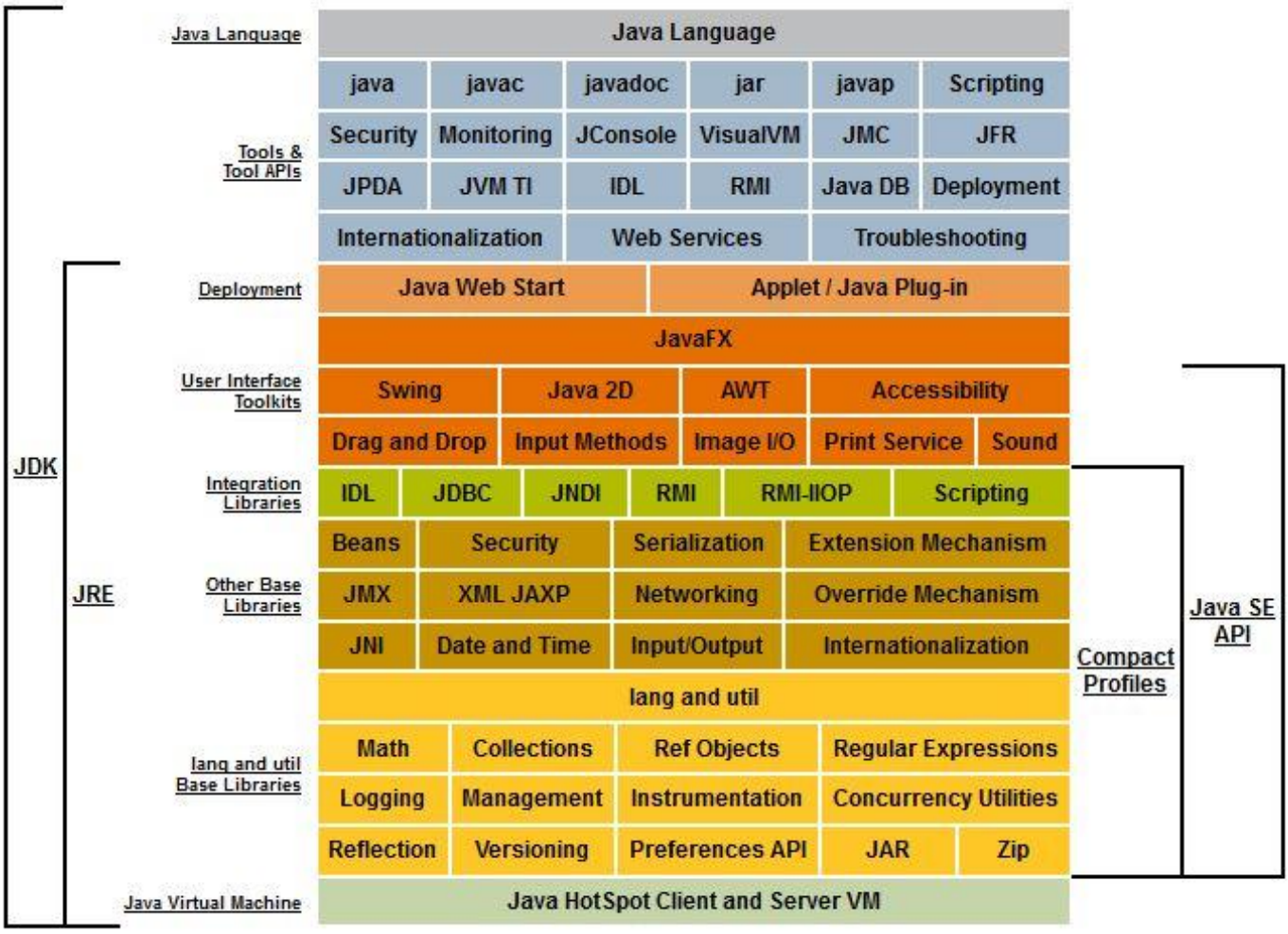


- 深入了解 JVM/JRE/JDK
- 理解封裝、繼承、多型
- 掌握常用 Java SE API 架構
- 學習容器觀念
- 研究開放原始碼專案
- 學習設計模式與重構
- 熟悉相關開發工具

查詢 Java API 文件

http://www.oracle.com/technetwork/java/api-141528.html

JVM/JRE/JDK



JVM

- Java Virtual Machine
- JVM 可以讓 Java 跨平台
- Java(.java)→Byte Code(.class)→各平台的 JVM 負責將 Byte Code 轉成機器碼

JRE 與 JDK

- JRE:Java Runtime Environment
  - 包括部署技術, Java SE API, JVM
- JDK:Java Development Kit
  - 提供編譯器
  - 包括 javac, appletviewer, javadoc 等工具程式
  - 包含 JRE

基礎語法

型態

基本型態(Primitive type)

- 每種型態佔有的記憶體長度不同，可儲存的數值範圍不同，超過範圍稱為溢位(Overflow)

整數

- short
  - 2bytes
  - -32,768~32,767
- int(預設)
  - 4bytes
  - -2,147,483,648~2,147,483,647
- long
  - 8bytes
  - -9,223,372,036,854~9,223,372,036,854,775,807

位元組

- byte
  - 可表示-128~127 的整數

浮點數

- float
  - 4bytes
  - 1.4E-45~3.402823e38
- double(預設)
  - 8bytes
  - 4.9E-324~1.79769313486232e308
  - 精準度較 float 高
- 不要對浮點數做相等性運算

[例] com.cathay.lesson03.FloatEquals

字元

- char
  - 2bytes
  - \u0000~\uFFFF

布林

- boolean
  - true, false

類別型態(Class type)

變數

- 暫存資料

基本規則

- 開頭不可為數字
- 不可使用特殊字元（e.g. *&^%）
- 不可使用 Java 關鍵字
- 不可使用 Java 保留字
- 以清楚易懂為主
- 命名慣例：小寫字母開始，每個單字的首字母大寫（駝峰式）

運算子

算數運算

- +, -, *, /, %

比較、條件運算

- >, >=, <, <=, ==, !=
- 對類別型態的變數來說，==是比較兩者是否參考同一物件
- 條件運算子：條件式 ? 成立回傳值 : 失敗回傳值

邏輯運算

- AND: &&
- OR: ||
- NOT: !

位元運算

- 逐位元運算
- AND: &
  - 兩個位元為 1 才是 1，其餘皆為 0
- OR: |
  - 兩個位元為 0 才是 0，其餘皆為 1
- XOR: ^
  - 兩個位元不同為 1，相同為 0
- NOT: ~
  - 0 變 1，1 變 0
- 左移(次方)運算: <<
  - 左邊捨棄，右邊補 0
- 右移運算: >>
  - 右邊捨棄，左邊補原來的位元
- >>>
  - 右邊捨棄，左邊補 0

遞增、遞減運算

- ++, --
- 變數前面：先執行，再回傳
- 變數後面：先回傳，再執行

指定運算

- +=, -=, *=, /=, %=, &=, |=, ^=, <<=, >>=

型態轉換

- 運算式包括不同型態數值，運算以長度最常的型態為主，其他數值自動提昇型態
- 如果運算元都是不大於 int 的整數，全部提昇為 int

流程控制

if..else..

```
if(條件式){
    陳述句;
}
else if(條件式){
    陳述句;
}
else{
    陳述句;
}
```

- else 如果沒有要做任何事，可以不寫
- 雖然只有一行陳述句時可以不寫{}，但是為了可讀性與可維護性，建議還是寫{}明確定義範圍

switch 條件式

```
switch(變數){
    case 比較對象:
        陳述句;
    default:
        陳述句;
}
```

- JDK7 前可比對整數、字元、Enum；JDK7 開始增加字串比對
- 遇到 break 會離開 switch 區塊

for 迴圈

```
for(初始式; 執行結果必須是 boolean 的重複式(預設為 true); 重複式){
    陳述句;
}
```

增強式 for 迴圈(JDK5↑)

```
for(多筆資料中的一筆:多筆資料){
    陳述句;
}
```

while 迴圈

```
while(條件式){
    陳述句;
}
```

```
do{
    陳述句;
}while(條件式);
```

break, continue

- break 可以離開目前 switch, for, while, do..while 區塊
- continue 與 break 類似，不過用於迴圈；會略過之後的陳述句，回到迴圈區塊開頭進行下一次迴圈

物件

類別與物件

- Java 撰寫程式幾乎都是在使用物件(Object)，要產生物件要先定義類別(Class)
- 類別是物件的設計圖，物件是類別的實體(Instance)

定義類別

```
class Clothes {
    String color;
    char size;
}
```

- 定義類別用 class
- 建立衣服實例要用 new >>> 建立一個物件

```
new Clothes();
```

- 可以宣告變數綁到這個物件上，這叫參考名稱(Reference name)、參考變數(Reference variable)、參考(Reference)
- 將變數綁到物件上要用=，Java 術語叫「參考到新建物件」
- 只要有一個 class，編譯器就會產生一個.class

物件指定與相等性

- 基本型態
  - =: 值複製給變數
  - ==: 比較兩個變數儲存的值是否相同
- 類別型態
  - =: 指定參考名稱參考某個物件
  - ==: 比較兩個參考名稱是否參考同一物件
  - equals: 比較兩個物件實質內容是否相同

基本型態包裹器

包裹基本型態

- Long, Integer, Double, Float, Boolean, Byte
- 將基本型態包裹在物件裡面，當作物件操作

自動裝箱、拆箱

- J2SE 5.0 之後提供自動裝箱(Auto boxing)與自動拆箱(Auto unboxing)功能

```
Integer number = 100;

// 反組譯
```

```
Integer number = Integer.valueOf(100);
```

[例] com.cathay.lesson04.IntegerEquals

陣列物件

陣列基礎

- 物件
- 具有索引(index)的資料結構
- 建議將[]放在型態後面
- length 屬性可以取得陣列長度（元素個數）
- 陣列建立之後，長度就固定了

二維陣列

```
型態[][]
```

- 三維以上的陣列則依此類推，但是不建議。因為不容易理解和閱讀，建議自訂類別來解決這類需求

操作陣列物件

- 不知道陣列內容，只知道個數 >>> new 型態[個數]

初始值

資料型態	初始值
byte	0
short	0
int	0
long	0L
float	0.0F
double	0.0D
char	\u0000
boolean	false
類別	null

陣列複製

- System.arraycopy()
- [JDK6↑] Arrays.copyOf()
- 以上都是淺層複製（複製參考），如果要深層複製（所有物件都複製一份）的話要自己實作

字串物件

- 是包裹字元陣列的物件，java.lang.String 的實例

字串特性

字串常量與字串池

- Java 中為了效率考量，以""包括的字串(字串常量)，只要內容相同，無論在程式碼出現幾次，JVM 都只會產生一個 String 實例，並在字串池(String pool)中維護
- 比較字串內容要用 equals

不可變動字串

- 字串物件一旦建立，就無法更改物件中任何內容
- 使用+串接字串實際上會產生新的物件
  - JDK5↑會用 StringBuilder 串接，但是在迴圈裡面的話，還是會一直產生物件
- 如果需要頻繁字串串接的話，可以用 StringBuilder 或 StringBuffer
  - StringBuffer 會處理同步問題，建議多執行緒環境下使用；StringBuilder 則相反，但是比 StringBuffer 有效率

[練習] com.cathay.practice.lesson04

- Q1:原練習 1-1
- Q2:原練習 1-2
- Q3:原練習 3-1

物件封裝

何謂封裝？

- 封裝的目的：隱藏物件細節，將物件當做黑箱進行操作

封裝物件初始流程

[例] com.cathay.lesson05.CashCard

[例] com.cathay.lesson05.CashCardDemo

- 不用重複撰寫物件初始化流程
- 使用者不用知道物件如何初始化，即便修改建構式的內容，使用類別的人也不需要修改程式

封裝物件操作流程

[例] com.cathay.lesson05.CashCard

[例] com.cathay.lesson05.CashCardDemo

- 封裝了操作流程，讓使用者不用知道流程也可以進行資料異動
- Java 命名規範中，取值方法的名稱為 get+首字大寫的單字

封裝物件內部資料

[例] com.cathay.lesson05.CashCard

[例] com.cathay.lesson05.CashCardDemo

- 封裝類別私有資料，讓使用者無法直接存取，必須透過類別提供的方法才有可能存取私有資料

類別語法細節

修飾子

修飾子	範圍
private	class
沒有宣告<default>	package
protected	package, 繼承類別的子類別
public	其他 package 也可使用

關於建構式

- 建構式是與類別同名的方法，不需要宣告回傳型態
- 沒有撰寫建構式，則 compiler 會加入一個無參數，內容為空的建構式（預設建構式）；如果自己有寫建構式的話，則不會加入

建構式與方法重載

- 可以定義多個建構式，只要參數型態或個數不同，稱之為重載建構式

- 方法也可以進行重載
  - 回傳型態不同不能當作重載依據
- JDK5↑使用重載時要注意 Auto boxing 及 Auto unboxing 的問題，如果要呼叫參數為物件版本時，參數需要明確指定
- compiler 處理重載方法會依以下順序處理
  1. Auto boxing 前可符合引數個數與型態的方法
  2. Auto boxing 後可符合引數個數與型態的方法
  3. 嘗試有不定長度引數並符合引數型態的方法
  4. 找不到合適的方法，compile error

使用 this

- 除了被宣告為 static 的地方以外，this 可以出現在類別任何地方
- 在物件建立後為「這個物件」的參考名稱
- this()代表另外一個建構式，必須在建構式的第一行

final

- 變數：設值後不能再變動
  - 未設值：延遲指定值，需在建構式指定

static 類別成員

- 不會讓個別物件擁有，而是屬於類別
- 將類別名稱作為名稱空間，static 方法也是
- 在 static 方法或區塊不能出現 this，因為 static 成員是屬於類別而非個別物件
- 如果希望在 byte code 載入後執行，可以定義 static 區塊

```
static {
    陳述句;
}
```

- [JDK5↑] 新增 import static，可以在使用靜態成員時少打一些字，但要注意名稱衝突。名稱衝突時 compiler 會以下列順序解析
  1. 區域變數覆蓋
  2. 成員覆蓋
  3. 重載方法比對
  4. 無法判斷則發生 compile error

不定長度引數

- JDK5↑
- 使用時，方法上宣告的不定長度參數必須是最後一個參數（只能有一個）

[例] com.cathay.lesson05.VariableLengthArg

內部類別

- 在類別中再定義類別

```
class Some{
    class Other{
    }
}
```

- 實務上較少看到在方法中定義具名的內部類別，比較常看到定義匿名內部類別並直接實例化

```
Object o = new Object(){
    public String toString(){
        return "";
    }
};
```

傳值呼叫

- Java 只有傳值呼叫（Call by value）

```
Object o1 = new Object();
Object o2 = o1;
show(o2);
void show(Object o){
}
```

- o2 只是將自己的值（位址）複製一份給 o，所以兩個還是參考到同一物件

繼承與多型

何謂繼承？

- 物件導向中，子類別繼承父類別，避免重複的行為與實作定義

繼承共同行為與實作

- 避免多個類別間重複定義了相同的行為與實作

[例] com.cathay.lesson06.game1

[例] com.cathay.lesson06.game2

多型與 is-a

- 子類別只能繼承一個父類別
- 子類別和父類別有一種 is-a 的關係

```
Role role1 = new SwordsMan();
```

- 從等號右邊往左讀：SwordsMan is a Role

```
Role role2 = new Magician();
SwordsMan swordsMan = (SwordsMan)role2;
```

- 執行時會出現 java.lang.ClassCastException
- 多型
  - 使用單一介面操作多種型態的物件
  - 具有更高的可維護性

重新定義實作

[例] com.cathay.lesson06.game3

- 繼承父類別之後，定義與父類別中相同名稱的方法 (大小寫需相同)，但是實作內容不同
- 在 JDK5 之後，可以用@Override 標註子類別中重新定義的方法，讓 compiler 檢查該方法是不是父類別中的方法

抽象方法、抽象類別

- 如果某方法區塊中真的沒有實作任何程式碼，可以使用 abstract 標示該方法為抽象方法，且不需要寫{}
- 如果類別中有方法沒有實作且標示 abstract，表示該類別不完整，無法產生實例。這種類別需要在 class 前面標示 abstract，即為抽象類別
- 子類別如果繼承抽象類別，對於抽象方法有兩種作法，一為繼續標示 abstarct，另外一個則是實作該方法

繼承語法細節

重新定義的細節

[例] com.cathay.lesson06.game4

- 可以使用 super 呼叫的父類別方法不能定義為 private
- 對於父類別中的方法權限只能擴大，不能縮小
- JDK5 之前，重新定義方法時除了可以定義權限較大的修飾子之外，其它部分必須與父類別的方法完全相同；JDK5 之後，回傳值可以改成父類別方法回傳型態的子類別
- static 方法屬於類別，如果子類別定義了相同的 static 成員，該成員屬於子類別，而不是重新定義
- static 方法也沒有多型

再看建構式

- 建構子類別實例時，會先呼叫父類別的建構式，再呼叫子類別的建構式
- this()與 super()只能擇一呼叫，且一定要在建構式第一行
- 如果定義了有參數的建構式，也可以加入無參數的建構式，內容可以為空，主要是為了之後使用上的彈性（e.g.使用 Reflection 機制生成物件的需求，或是繼承時呼叫父類別建構式）

再看 final 關鍵字

- 類別：無法被其它類別繼承
  - e.g. String: http://docs.oracle.com/javase/8/docs/api/java/lang/String.html
- 方法：子類別不可重新定義方法
  - e.g. Object.notify(): http://docs.oracle.com/javase/8/docs/api/java/lang/Object.html
- 在 Java SE API 中會宣告為 final 的類別或方法通常與 JVM 物件或作業系統資源管理有關，因此不希望使用者繼承或重新定義

java.lang.Object

- 子類別只能繼承一個父類別，如果繼承任何類別，則為繼承 java.lang.Object
- Java 中的所有物件一定「是一種」Object

instanceOf

- Object 可以透過 obj instanceOf 類別 來判斷是否為該類別

關於垃圾收集

- JVM 有垃圾收集（Garbage Collection, GC）機制，收集到的垃圾物件所佔據的記憶體空間會被釋放
- 執行流程中無法透過變數參考的物件就是垃圾物件
- GC 啟動時機由 GC 演算法決定，無法透過程式控制

[練習] com.cathay.practice.lesson06

介面與多型

何謂介面？

使用介面定義行為

[例] com.cathay.lesson07.oceanworld1

[例] com.cathay.lesson07.oceanworld2

- 定義「可被擁有的行為」，可以使用 interface
- 類別要實作介面必須使用 implements，實作介面時，對介面中定義的抽象方法有兩種處理方式
- 繼承會有「是一種(is a)」關係，實作介面則表示「擁有(has a)行為」，並不會有「是一種」的關係

行為的多型

```
Swimmer swimmer1 = new Shark();
Swimmer swimmer2 = new Human();
Swimmer swimmer3 = new Submarine();
```

- 右邊是不是擁有左邊的行為？

```
Swimmer swimmer1 = new Shark();
Shark shark = swimmer1;
```

- 有 Swimmer 行為的不一定是 Shark

[例] com.cathay.lesson07.oceanworld2

解決需求變化

[例] com.cathay.lesson07.oceanworld3

- 類別可以同時繼承某個類別，並實作某些介面
- 類別可以實作兩個以上的介面
- 介面可以繼承自另一個介面
  - 可以繼承兩個以上的介面

介面語法細節

介面的預設

- 介面中的方法沒有實作時，一定得是公開且抽象(public abstract)，所以 public abstract 可以省略
- 在 interface 中可以定義常數

```
public interface Action {
    public static final int STOP = 0;
    public static final int RIGHT = 1;
```

```
    public static final int LEFT = 2;
    public static final int UP = 3;
    public static final int DOWN = 4;
}
```

- 上述的常數稱為列舉常數
- 在 interface 中 public static final 可省略

匿名內部類別

- 在撰寫 Java 程式時，經常會有臨時繼承某個類別或實作某個介面並建立實例的需求，由於這類子類別或介面實作類別只使用一次，不需要為這些類別定義名稱，這時可以使用匿名內部類別

```
new 父類別()|介面(){
    // 類別本體實作
};
```

- JDK8 之前，匿名內部類別的區域變數必須是 final；但在 JDK8，如果變數在匿名類別中不會有重新指定的動作，就可以不用加上 final

使用 enum 列舉常數

- JDK5↑

```
public enum Action {
    STOP, RIGHT, LEFT, UP, DOWN
}
```

[練習] com.cathay.practice.lesson07

例外處理

語法與繼承架構

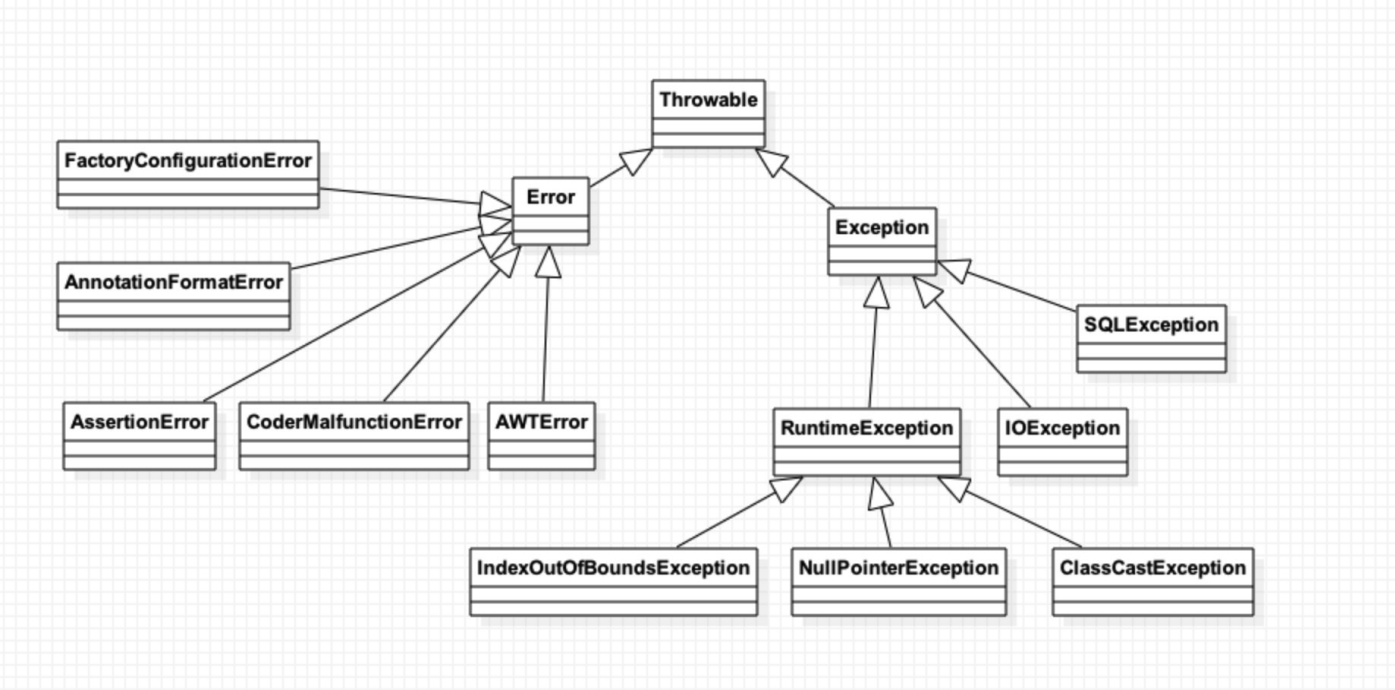
- Java 中的錯誤也以物件方式呈現，為 java.lang.Throwable 的子類別實例

使用 try..catch..

- Java 中所有的錯誤都會被包裹為物件，所以可以用 try 執行程式並 catch 代表錯誤的物件做處裡

[例] com.cathay.lesson08.trycatch.Average

例外繼承架構



- Error 與子類別代表嚴重系統錯誤，基本上不用處理
- 程式設計的錯誤建議使用 Exception 或其子類別實例，所以通常稱錯誤處理為例外處理
- Exception 或其子類別，但不屬於 RuntimeException 或其子類別稱為受檢(受 compiler 檢查)例外
- RuntimeException 衍生出來的類別實例，代表 API 設計者實作某方法時，某些條件成立會引發錯誤，compiler 不會強迫一定要處理這個錯誤
- 父類別例外需在子類別例外後面
- [JDK7↑] 可以使用多重捕捉

```
try{
    do something
}catch(IOException | InterruptedException | ClassCastException e){
    e.printStackTrace();
}
```

- 父類別某個方法宣告 throws 某些例外，子類別重新定義該方法時，
  - 可以

- 不宣告 throws 任何例外
- throws 父類別該方法中宣告的某些例外
- throws 父類別該方法中宣告例外之子類別

■ 不可以

- throws 父類別方法中未宣告的其它例外
- throws 父類別方法中宣告例外之父類別

認識堆疊追蹤

➢ 多重方法呼叫下，例外發生點可能是在某個方法之中，如果想要得知例外發生的根源，以及多重方法呼叫下例外的堆疊傳播，可以利用例外物件自動收集的堆疊追蹤(Stack Trace)

```
e.printStackTrace();
```

➢ 如果想要取得個別的堆疊追蹤元素進行處理，可以用 getStackTrace()，這會回傳 StackTraceElement 陣列，接下來就可以用 getClassName(), getFileName(), getLineNumber(), getMethodName()等方法取得對應的資訊

➢ fillInStackTrace()可以讓例外堆疊起點為重拋例外的地方

[例] com.cathay.lesson08.trycatch.StackTraceDemo

例外與資源管理

使用 finally

➢ 與 try 搭配使用

➢ 無論 try 區塊有無例外發生，finally 區塊一定會被執行

[例] com.cathay.lesson08.trycatch.Average2

➢ 如果 try 或 catch 區塊有寫 return，則 finally 區塊執行完之後才會回傳

自動嘗試關閉資源

➢ JDK7↑

➢ 使用嘗試關閉資源語法時，不要再自行撰寫關閉資源的程式碼

➢ 關閉的資源物件需實作 java.lang.AutoCloseable

■ AutoCloseable 為 interface，只定義了 close()方法

➢ 同時關閉兩個以上的資源中間用分號區隔

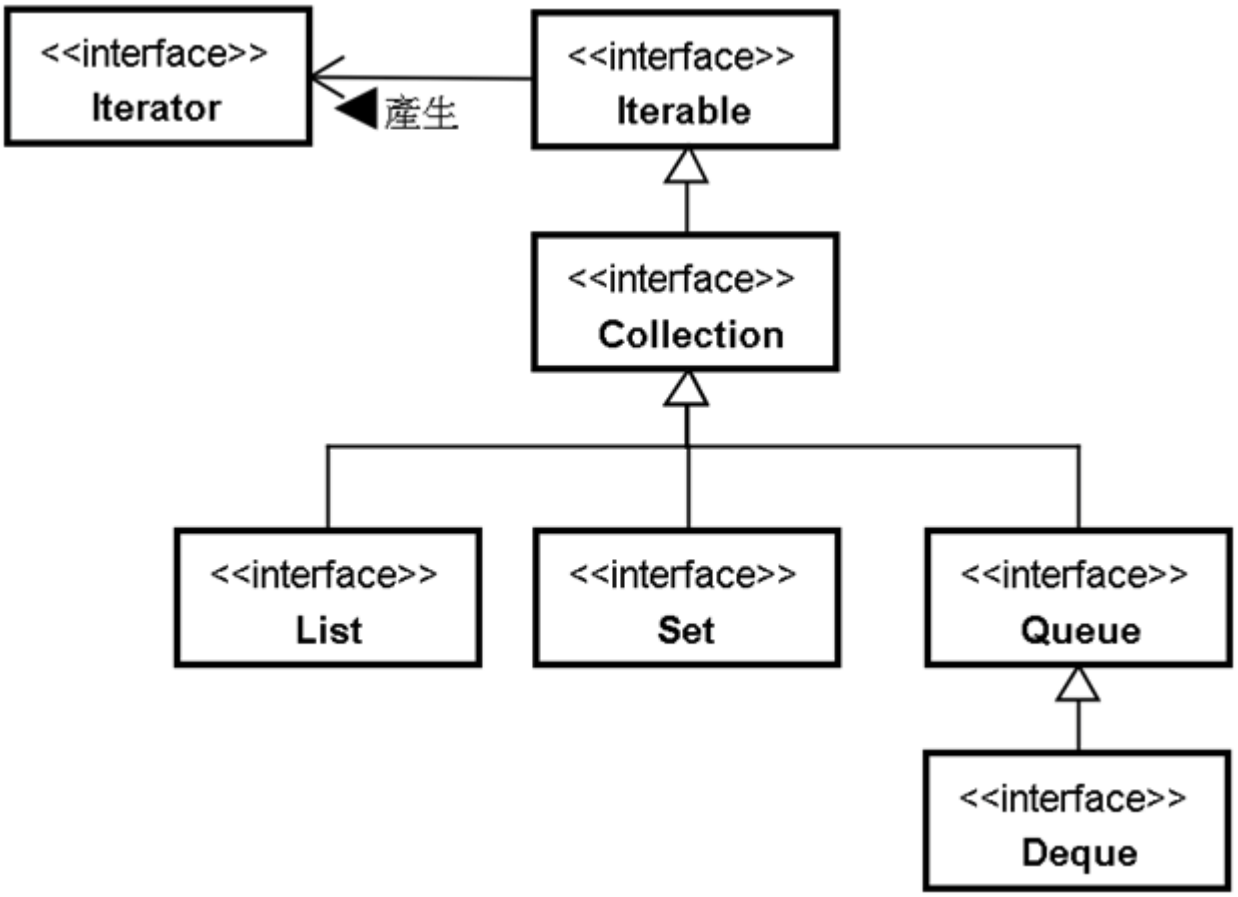
➢ 越後面撰寫的物件資源會越早關閉

[例] com.cathay.lesson08.trycatch.Average3

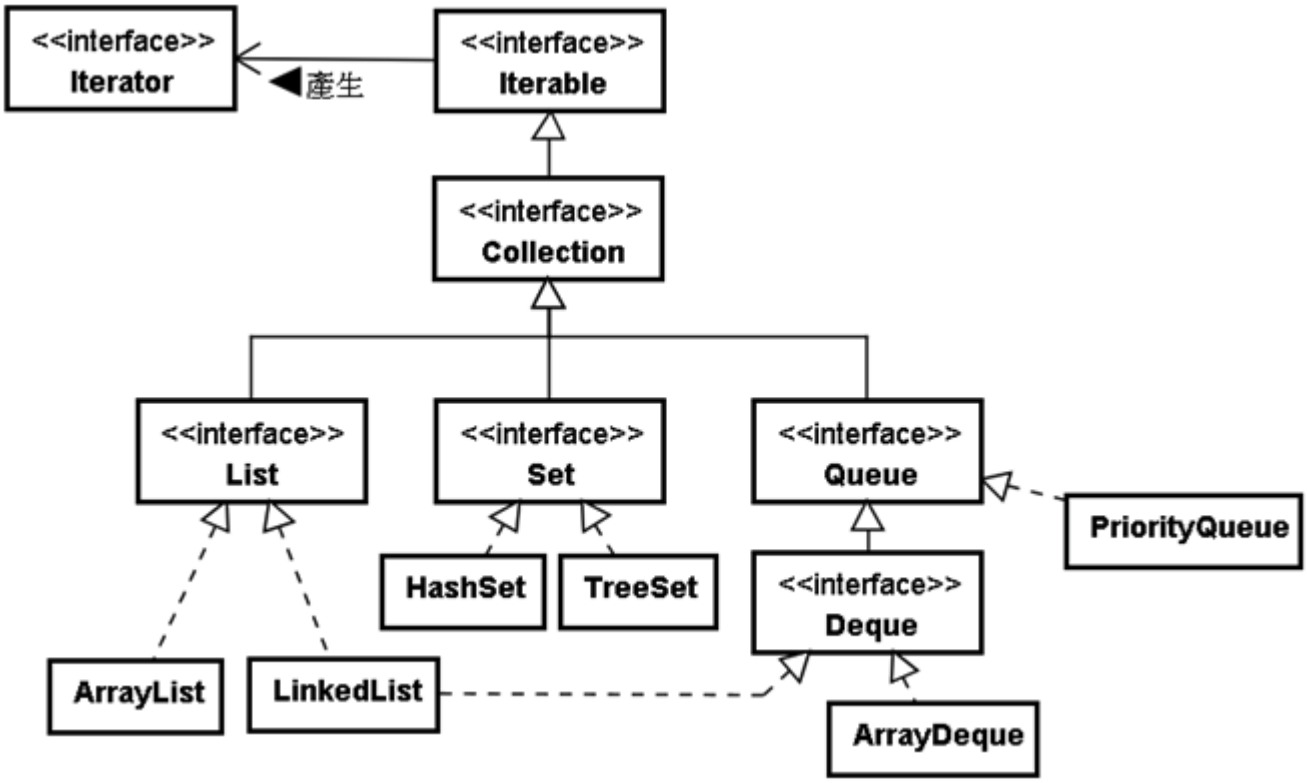
Collection 與 Map

使用 Collection 收集物件

認識 Collection 架構



- List: 希望收集時紀錄每個物件的索引順序，並可依索引取回物件
- Set: 收集的物件不重複，具有集合的行為
- Queue: 收集物件時可以佇列方式，收集的物件加入至尾端，取得物件時可以從前端
- Deque: 可以對 Queue 的兩端進行加入、移除等操作



具有索引的 List

ArrayList 特性

- 根據索引隨機存取速度快，因為陣列在記憶體是連續的線性空間
- 需要調整索引順序時的表現較差
- 陣列長度不夠時會建立新陣列，並將就陣列的參考指定給新陣列，需要耗費時間與記憶體
- 有個可以指定容量的建構式

LinkedList 特性

- 指定索引隨機存取效率較差
- 調整索引順序時表現較佳
- Add 時才建立新的 Node 保存物件，不會事先耗費記憶體

內容不重複的 Set

- 會依序使用 HashCode()與 equals()判斷物件是否相同
- Java 中許多要判斷物件是否重複時，都會呼叫 HashCode()與 equals()方法，所以建議兩個方法必須同時實作

[例] com.cathay.lesson09.collection.Students

支援佇列操作的 Queue

- 繼承 Collection，所以有 add(), remove(), element()。操作失敗會拋例外
- 自己定義了 offer(), poll(), peek()。操作失敗會回傳特定值
  - 建議使用

- offer(): 佇列後端加入物件，成功:true, 失敗:false
- poll(): 取出佇列前端物件，佇列為空回傳 null
- peek(): 取得(物件還是存在)佇列前端物件，佇列為空回傳 null

➢ LinkedList 也有實作

➢ Queue 的子介面 Deque 定義了前端與尾端加入或取出物件的行為

使用泛型

➢ 使用 Collection 收集物件時，由於事先不知道被收集物件之型態，因此實作都用 Object 來參考被收集的物件，取物件也是用 Object，所以執行時期被收集的物件會失去型態資訊

➢ [JDK5↑] 在設計 API 時可以指定類別或方法支援泛型，不但後續操作的語法會更簡潔，而且 compile 時會檢查

Iterable 與 Iterator

➢ List, Set, Queue 都有 iterator()

➢ [JDK5↓] Collection

➢ [JDK5↑] java.lang.Iterable

[例] com.cathay.lesson09.collection.ForEach

Comparable 與 Comparator

➢ Collections.sort()
- 接受 List 實作物件

[例] com.cathay.lesson09.collection.Sort

[例] com.cathay.lesson09.collection.Sort2

➢ Collections.sort()要求被排序物件需要實作 java.lang.Comparable

[例] com.cathay.lesson09.collection.Sort3

實作 Comparator

➢ 如果使用的類別已經有實作 Comparable 或拿不到原始碼或原始碼不可以修改

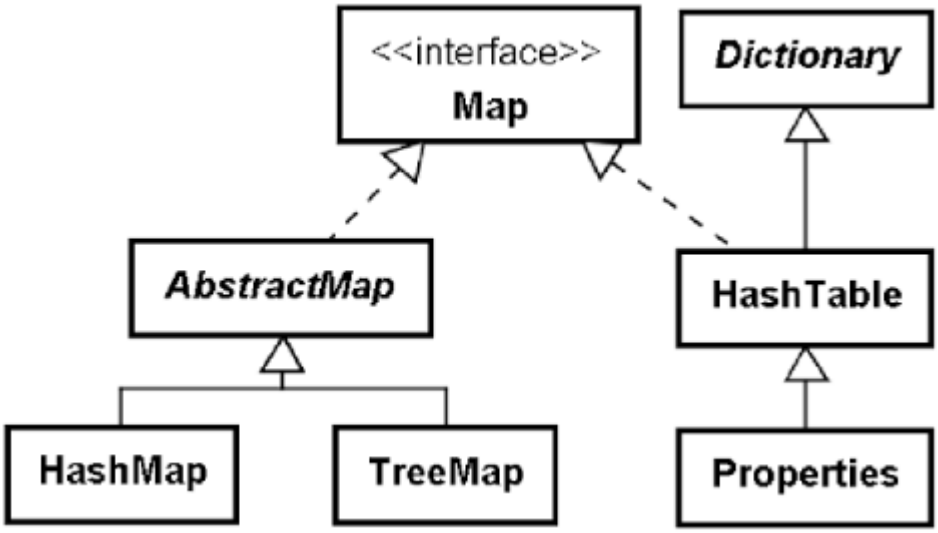
➢ Collections.sort()有一個重載版本，可接受實作 java.util.Comparator 的物件

[例] com.cathay.lesson09.collection.Sort4

➢ 跟順序有關的行為，不是物件有實作 Comparable，要不就是另外指定 Comparator 物件排序
- java.util.TreeSet
- java.util.PriorityQueue

鍵值對應的 Map

常用 Map 實作類別



- [JDK1.0↑] java.util.Dictionary, java.util.HashTable 不建議使用

使用 HashMap

- 判斷 key 是否重複是根據 hashCode()和 equals()，所以當作 key 的物件必須實作

使用 TreeMap

- Key 的部份會排序
  - Key 的物件需實作 Comparable 或建構 TreeMap 時指定實作 Comparator

使用 Properties

- 繼承 HashTable，HashTable 實作 Map
- 一般使用 setProperty()指定字串型態的 key
- getProperty()指定字串型態的 key，取得字串型態的值
- 可以從檔案讀屬性
- 除了 new Properties()之外，也可以用 System.getProperties()取得

走訪 Map 鍵值

- 取得所有的 key: keySet()
- 取得所有的 value: values()
- 同時取得 key 和 value: entrySet()
  - getKey()
  - getValue()

MultiKeyMap 與 MultiKey

- org.apache.commons.collections.map.MultiKeyMap
- org.apache.commons.collections.keyvalue.MultiKey
- Apache 提供的多 Key Map

[練習] com.cathay.practice.lesson09

- Q1:原練習 2-1
- Q2:原練習 2-2

輸入輸出

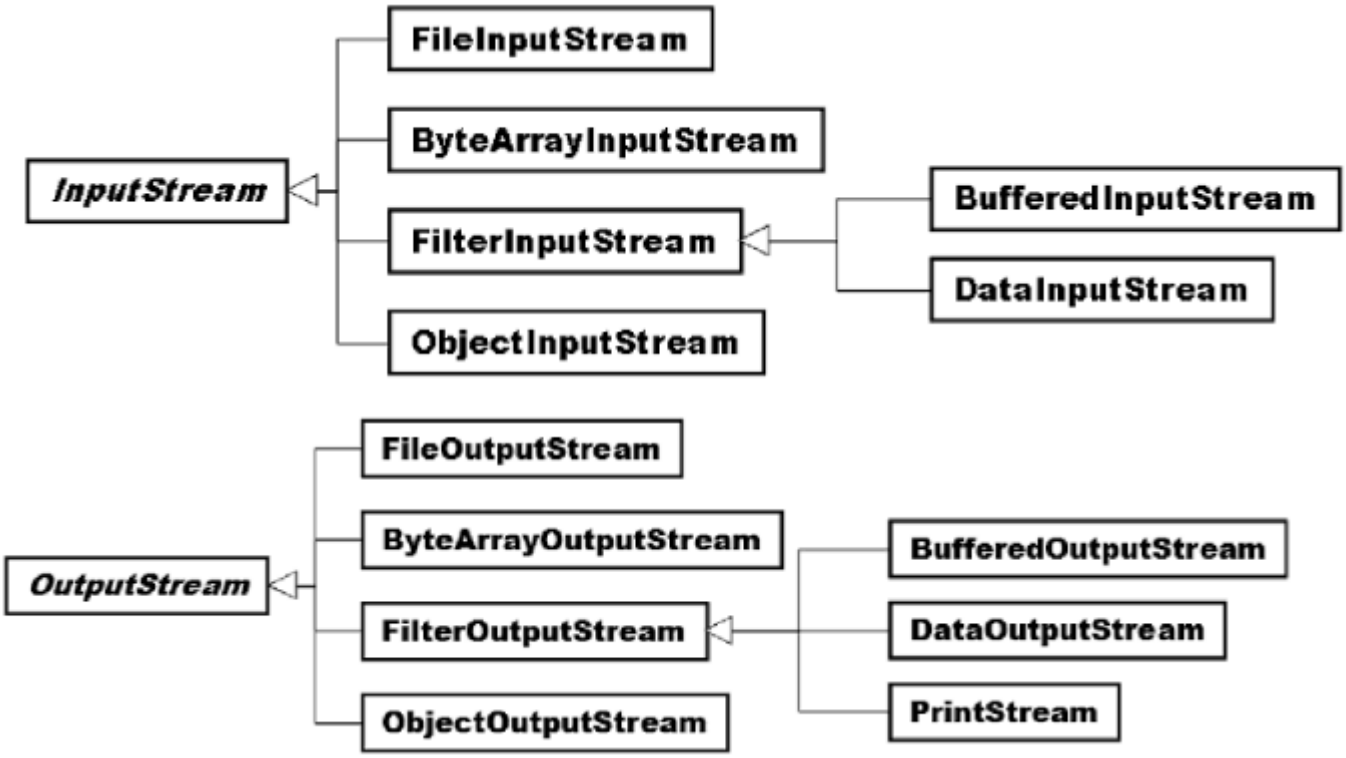
InputStream 與 OutputStream

串流設計觀念

- 輸入串流: java.io.InputStream
- 輸出串流: java.io.OutputStream
- 不管資料來源或目的地，只要取得 InputStream 和 OutputStream 的實例，接下來操作都相同
- 不使用時需用 close()關閉(寫在 finally 區塊)

[例] com.cathay.lesson10.stream.IO

串流繼承架構



標準輸入輸出

- System.in: InputStream 實例
- System.out: PrintStream 實例
- System.err: PrintStream 實例

FileInputStream 與 FileOutputStream

- FileInputStream: InputStream 的子類
  - 實作 InputStream 的 read()
- FileOutputStream: OutputStream 的子類
  - 實作 OutputStream 的 write()
- 讀寫檔案時，以位元為單位，通常會用一些高階類別包裹起來

[例] com.cathay.lesson10.stream.Copy

ByteArrayInputStream 與 ByteArrayOutputStream

- ByteArrayInputStream: InputStream 的子類
  - 實作 InputStream 的 read()
- ByteArrayOutputStream: OutputStream 的子類
  - 實作 OutputStream 的 write()

串流處理裝飾器

- 具備緩衝區作用: BufferedInputStream, BufferedOutputStream
- 具備資料轉換處理作用: DataInputStream, DataOutputStream
- 具備物件序列化能力: ObjectInputStream, ObjectOutputStream
- 本身沒有改變 InputStream 與 OutputStream 的行為，只是在取資料或輸出時作一些加工

BufferedInputStream 與 BufferedOutputStream

- BufferedInputStream: 先去緩衝區讀取資料，如果緩衝區沒有資料，再從資料來源讀資料近緩衝區
- BufferedOutputStream: 將資料寫入緩衝區，如果緩衝區滿了，才將資料寫入目的地
- 提供緩衝區功能，可以減少對資料來源或目的地的讀取及寫入次數

[例] com.cathay.lesson10.stream.BufferedIO

DataInputStream 與 DataOutputStream

- 提供讀取、寫入 Java 基本資料類型的方法，像是讀寫 int, double, boolean 等的方法。這些方法會自動在指定型態與位元組間轉換

[例] com.cathay.lesson10.stream.Member

ObjectInputStream 與 ObjectOutputStream

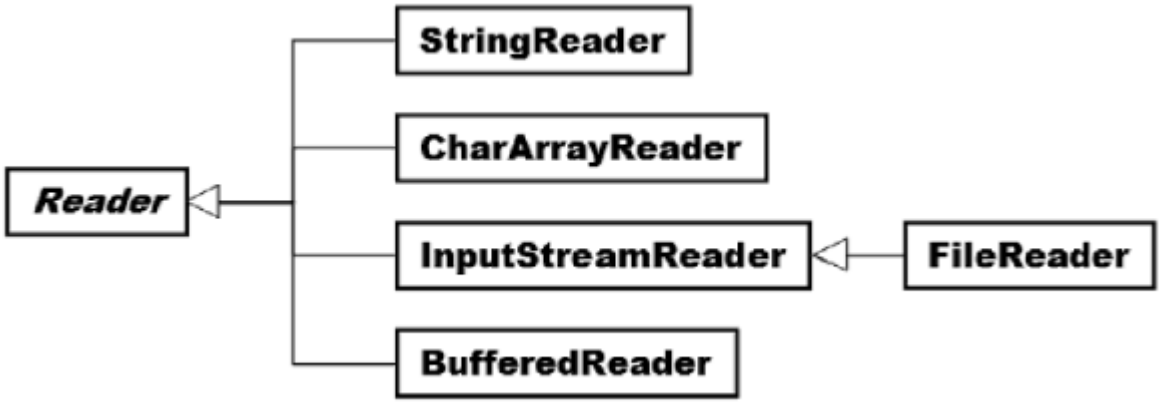
- ObjectInputStream: 提供 readObject()讀取資料轉為物件
- ObjectOutputStream: 提供 writeObject()將物件寫到目的地
- 可以被處理的物件需要實作 java.io.Serializable
  - 介面沒有定義方法，只是標示該物件可以序列化
- 物件序列化時，不希望被寫入的資料成員可以標上 transient

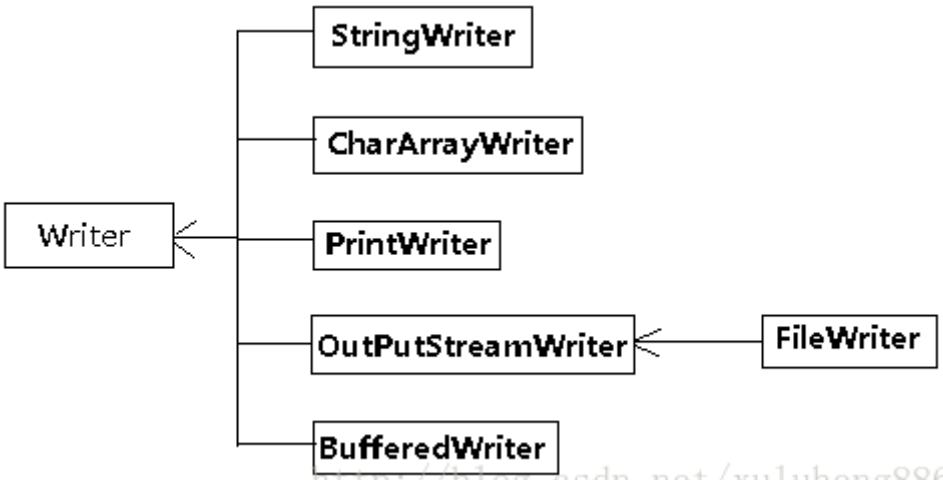
[例] com.cathay.lesson10.stream.Member2

字元處理類別

Reader 與 Writer 繼承架構

- 針對字元資料處理





- FileReader 和 FileWriter 預設使用作業系統預設編碼來作字元轉換
- JVM 可以透過設定-Dfile.encoding 指定使用的編碼
- 如果程式想要指定編碼需要使用 InputStreamReader 和 OutputStreamWriter

[例] com.cathay.lesson10.stream.CharUtil

字元處理器

InputStreamReader 與 OutputStreamWriter

- 串流處理的位元組資料
- 建立時可以指定編碼，沒有則用 JVM 啟動時的編碼

[例] com.cathay.lesson10.stream.CharUtil2

BufferedReader 與 BufferedWriter

- 提供 Reader, Writer 緩衝區作用

PrintWriter

- 與 PrintStream 類似

[練習] com.cathay.practice.lesson10

- Q1:原練習 4
- Q2:原練習 5

時間與日期

認識 Date 與 Calendar

時間軸上瞬間的 Date

- 系統時間: System.currentTimeMillis()
  - 1970-01-01 00:00:00.000 至今的毫秒數
- [JDK1.0↑] java.util.Date
- Date 實例基本上建議只用來當作時間軸上的某一瞬間
- 不建議用 toString()取得年月日資訊

格式化時間日期的 DateFormat

- 抽象類別，實作為 SimpleDateFormat
  - 建構式：可以使用字串自訂格式
  - DateFormat 的 getDateInstance(), getTimeInstance(), getDateTimeInstance()

處理時間日期的 Calendar

- 取得或操作某個時間日期資訊
- 抽象類別
- java.util.GregorianCalendar 是其子類別: 實作了 Julian calendar 和 Gregorian calendar 的混合曆

[例] com.cathay.lesson13.datecalendar.CalendarUtil

[練習] com.cathay.practice.lesson13

- Q1:原練習 3-2

API

常用格式控制符號

符號	說明
%%	顯示%
%d	以 10 進位整數格式輸出，可用於 byte, short, int, long, Byte, Short, Integer, Long, BigInteger
%f	以 10 進位浮點數格式輸出，可用於 float, double, Float, Double, BigDecimal
%e, %E	以科學記號浮點數格式輸出，可用於 float, double, Float, Double, BigDecimal
%o	以 8 進位整數格式輸出，可用於 byte, short, int, long, Byte, Short, Integer, Long, BigInteger
%x, %X	以 16 進位整數格式輸出，可用於 byte, short, int, long, Byte, Short, Integer, Long, BigInteger
%s, %S	字串
%c, %C	以字元符號輸出，可用於 byte, short, char, Byte, Short, Character, Integer
%b, %B	輸出 boolean 值
%h, %H	使用 Integer.toHexString(arg.hashCode())來得到輸出結果，也常用於想得到 16 進位格式輸出
%n	換行 Windosw:\r\n; Linux:\n; MacOS:\r

設定陣列初始值

java.util.Arrays.fill()

將字串轉為基本型態

方法	說明
Byte.parseByte	byte 整數
Short.parseShort	short 整數
Integer.parseInt	int 整數
Long.parseLong	long 整數
Float.parseFloat	float 浮點數
Double.parseDouble	double 浮點數