

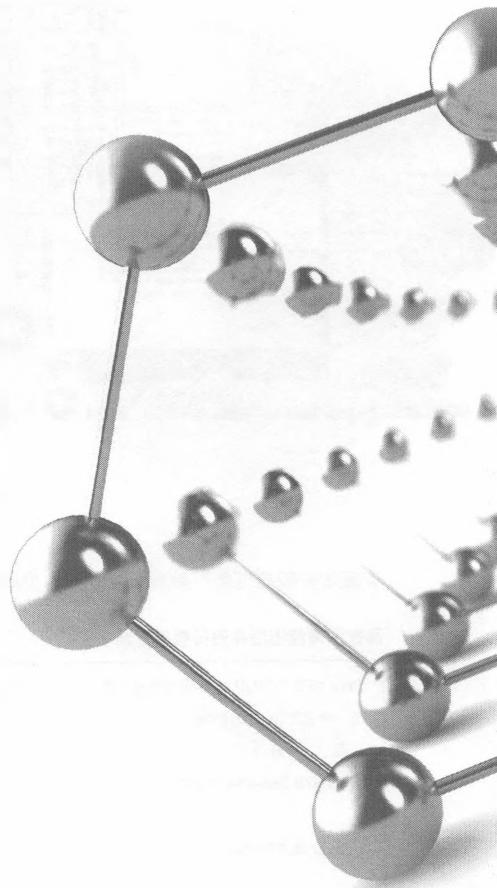
Java SE8 OCPJP

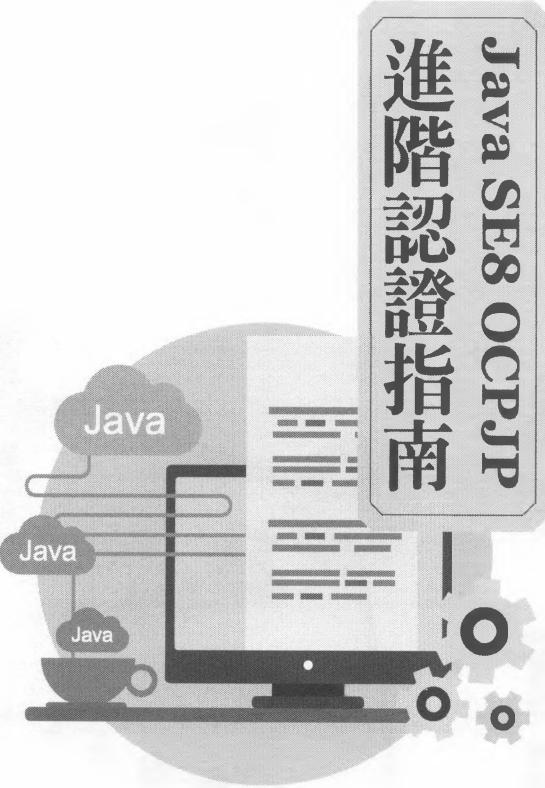
進階認證指南

曾瑞君 著

由OCAJP進階OCPJP

- » 解析原廠文件，切合認證範圍！
- » 對照範例程式，迅速了解內容！
- » 彙整教學經驗，重點一次掌握！
- » 圖解複雜觀念，學習輕鬆上手！
- » 演練擬真試題，掌握考試精髓！





Java SE8 OCPJP 進階認證指南

作 者：曾瑞君

責任編輯：曾婉玲

董事長：蔡金嵐

總編輯：陳錦輝

出 版：博碩文化股份有限公司

地 址：221 新北市汐止區新台五路一段 112 號 10 樓 A 棟

電話 (02) 2696-2869 傳真 (02) 2696-2867

郵撥帳號：17484299 戶名：博碩文化股份有限公司

博碩網站：<http://www.drmaster.com.tw>

讀者服務信箱：dr26962869@gmail.com

訂購服務專線：(02) 2696-2869 分機 238、519

(週一至週五 09:30 ~ 12:00；13:30 ~ 17:00)

版 次：2019 年 6 月初版一刷

2022 年 6 月初版二刷

建議零售價：新台幣 720 元

I S B N : 978-986-434-399-7 (平裝)

律師顧問：鳴權法律事務所 陳曉鳴 律師

本書如有破損或裝訂錯誤，請寄回本公司更換

國家圖書館出版品預行編目資料

Java SE8 OCPJP 進階認證指南 / 曾瑞君著 . -- 新北

市：博碩文化，2019.06

面： 公分

ISBN 978-986-434-399-7(平裝)

1. Java(電腦程式語言)

312.32J3

108007831

Printed in Taiwan



歡迎團體訂購，另有優惠，請洽服務專線
博 碩 粉 絲 團 (02) 2696-2869 分機 238、519

商標聲明

本書中所引用之商標、產品名稱分屬各公司所有，本書引用純屬介紹之用，並無任何侵害之意。

有限擔保責任聲明

雖然作者與出版社已全力編輯與製作本書，唯不擔保本書及其所附媒體無任何瑕疵；亦不為使用本書而引起之衍生利益損失或意外損毀之損失擔保責任。即使本公司先前已被告知前述損毀之發生。本公司依本書所負之責任，僅限於台端對本書所付之實際價款。

著作權聲明

本書著作權為作者所有，並受國際著作權法保護，未經授權任意拷貝、引用、翻印，均屬違法。

推薦序

金融科技（FinTech）創新正如一架即將起飛的太空梭，未來將徹底改變傳統金融業的型態，金融業若未即早因應，將無法抓住發展趨勢，提升競爭力，甚至逐漸退出市場。金融科技發展趨勢包括：無現金的社會、區塊鏈發展、數位貨幣發展、生物辨識及機器人應用。在這波「破壞式創新」趨勢中，金融從業人員應儘快熟悉相關數位資訊，如雲端科技、大數據分析等，以增加職場競爭力。

Java 是一種電腦程式語言，也是大多數科技應用的基礎，取得 Java 專業證照，除了使工作選擇更加多元，亦可以提升個人職場競爭優勢。

瑞君擁有 Java、資料庫管理、資訊安全等多種跨領域證照與專長，也曾於金融業服務，了解科技創新的重要性，目前也是 Oracle 原廠授權講師。

三年前，本人推薦他關於 Java 專業證照的考試書籍，因受市場肯定而於日前再刷，並修訂內容與書名為《Java SE8 OCAJP 專業認證指南》與《Java SE8 OCPJP 進階認證指南》；這次同步推出的新書《Java RWD Web 企業網站開發指南：使用 Spring MVC 與 Bootstrap》，則提供了讀者進入企業後的進階工作技能，向金融科技創新邁進一大步。

本人樂意為其三本著作推薦。希望透過這三本書，讀者能順利考取證照並習得進階技能，注入創新活水，進而為台灣金融科技產業創造競爭優勢！

立法院立法委員 曾銘宗 謹識

推薦序

曾經有過一份研究，訪問了一百多名臨終前的老人，要他們回想人生中最大的遺憾是什麼？幾乎全部的人的答案，都不是關於他們做過的事，而是關於他們沒做過什麼、沒冒過的險、沒追過的夢，所以最終人生留下遺憾。

瑞君曾在 HTC 的資訊部門擔任帶領廠商進行系統架構設計與維運的任務，公餘時間則投入 Java 教學、著作，實現自己的另一個志趣。離開 HTC 之後持續耕耘，很高興他的書籍《Java SE8 OCAJP 專業認證指南》與《Java SE8 OCPJP 進階認證指南》能夠獲得市場肯定，進入二版；並且再接再厲推出新書《Java RWD Web 企業網站開發指南：使用 Spring MVC 與 Bootstrap》。以他的專業能力與教學經驗，相信其系列著作必能為更多有志 Java 程式開發領域的人提供基礎與進階助力，本人樂為其推薦。

htc IT 副總經理 謝曼帆 謹識

推薦序

時間消逝之快，真如白駒過隙！

Jim 在 2005~2009 年間服務於華碩，當時工作內容是 Oracle DBA 與效能調教，表現突出。後來因為個人生涯規劃想多接觸一些軟體工程與程式設計領域，因此在公餘學習 Java，取得證照後轉職到軟體公司歷練。

十多年來，我們一直保持連繫，也時常對一些資訊科技交流與討論。Jim 由原本在資料庫管理與效能調教的專長，又涵蓋了 Java 程式開發與資訊安全，這過程中除了執著，更投入了可觀的心力。如今他將自己的心路歷程與經驗點滴彙整為《Java SE8 OCAJP 專業認證指南》、《Java SE8 OCPJP 進階認證指南》與《Java RWD Web 企業網站開發指南：使用 Spring MVC 與 Bootstrap》等三本書，對於想和他一樣邁入 Java 程式設計領域的人，無疑是捷徑與福音！本人樂為其書推薦。

 華碩企業智能數據中心 處長 郭海威 謹識

推薦序

和瑞君結緣，是在他就讀研究所的空檔時間裡，當時他在我們的中國台商發展協會中處理電腦相關事務。他畢業後，我們一直保持著密切的連繫；在成立中華兩岸頤養促進會時，他也協助了網站建置，並討論了許多資訊相關流程。

一路走來，我看著瑞君在工作裡都能有卓越的績效，因此得到前長官如華碩、宏達電等高階主管的肯定；在教學與著作方面，能夠努力不懈與推陳出新，因此擁有愈來愈多的學生與讀者。

如今，他有兩本為程式新鮮人撰寫的證照書籍《Java SE8 OCAJP 專業認證指南》與《Java SE8 OCPJP 進階認證指南》即將再刷；而新書《Java RWD Web 企業網站開發指南：使用 Spring MVC 與 Bootstrap》則鎖定企業應用領域，相信三管齊下，必能為他的讀者提供更多助益！本人樂為其推薦。

中華兩岸頤養促進會 理事長
中國台商發展促進協會 秘書長

詹清池 謹識

序言

感謝各位書友、版友的支持，也誠摯感謝 前金管會主委暨現任立法委員 曾銘宗委員、宏達電 謝昱帆副總經理、華碩電腦 郭海威處長、中華兩岸頤養促進會 詹清池理事長等對本書的肯定與推薦，《Java SE7/8 OCAJP 專業認證指南：擬真試題實戰》與《Java SE7/8 OCPJP 進階認證指南：擬真試題實戰》進入二刷了！這次再刷除了修正些許筆誤外，書名也調整為《Java SE8 OCAJP 專業認證指南》與《Java SE8 OCPJP 進階認證指南》。

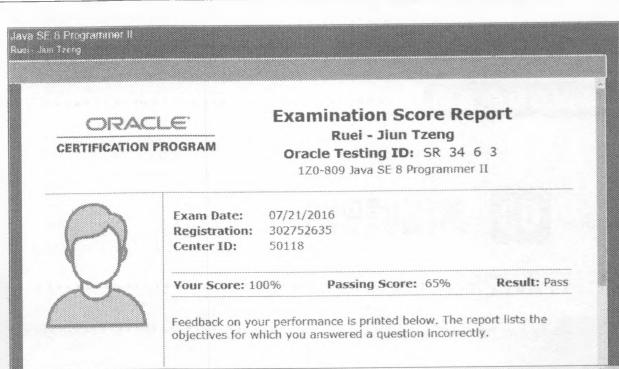
約莫三年前首刷時，為了能與書友們更直接互動，於 FB 成立「Java 技術與認證交流平台」社團。交流過程中，發現部分書友對擬真試題的重視，甚至遠超過書籍正文！這樣的認知很容易理解，但相信是不太恰當的！

對於一個程式設計師而言，進入職場後還是以解決問題的實力為重；證照的目的在證明自己 Java 觀念的熟練度，而非考試能力。本書內容多依據原廠的教育訓練文件編寫，建議研讀時應以觀念、範例為首要，擬真試題只是用來複習，並檢驗學習內容是否符合原廠認證考試的測驗範圍。也因此，本次再刷時取消原本書名「擬真試題實戰」字樣，避免過度強調，而忽略了該培養的基礎與實力。

此外，因為 Oracle 宣布自 2019/01/01 起將 Java SE7 認證考試全面由 Java SE8 取代，故本系列書籍配合調整部分內容，並將書名由 SE7/8 修改為 SE8，讓讀者更明確專注於 SE8 教材與考試內容。預祝各位書友都能在本書的協助下，瞭解 Java SE8 的精髓，或考取證照，或了解您想知道的東西！

曾瑞君 謹誌於桃園

以下 是 <https://brm-certview.oracle.com> 查詢個人 1Z0-809 考試成績單結果。Testing ID 與大頭照做了變更，其餘不變，給立志考取證照的您參考：



目錄

Chapter 01 Java類別與基本語法複習

1.1 建立 Java 類別	01-02
1.2 使用 package、import 敘述	01-09
1.3 使用選擇結構和重複結構	01-11
1.4 參數傳遞機制	01-13
1.5 記憶體使用機制	01-16

Chapter 02 了解封裝、繼承和多型

2.1 使用封裝	02-02
2.2 建立子類別和使用繼承關係	02-04
2.2.1 繼承的目的和建立子類別	02-04
2.2.2 建立子類別的建構子	02-05
2.3 多載方法與使用可變動參數個數的方法	02-08
2.3.1 多載方法	02-08
2.3.2 可變動參數個數的方法	02-09
2.4 使用多型	02-10
2.4.1 多型的意義	02-10
2.4.2 Java 只能單一繼承	02-15
2.5 認證考試命題範圍	02-18
本章擬真試題實戰	02-19

Chapter 03 類別設計

3.1 存取控制	03-02
3.1.1 存取控制層級	03-02

3.1.2 欄位遮蔽效應	03-03
3.2 覆寫方法	03-05
3.2.1 覆寫方法的規則	03-05
3.2.2 只有物件成員的方法可以覆寫	03-09
3.2.3 善用多型 (Polymorphism)	03-12
3.2.4 instanceof 運算子	03-13
3.3 轉型	03-14
3.4 覆寫 Object 類別的方法	03-17
3.5 認證考試命題範圍	03-21
本章擬真試題實戰	03-22

Chapter 04 進階類別設計

4.1 使用抽象類別	04-02
4.2 使用 static 關鍵字	04-02
4.2.1 static 的方法與欄位	04-03
4.2.2 static import	04-05
4.3 使用 final 關鍵字	04-05
4.4 實作獨體設計模式	04-08
4.5 列舉型別 (enum)	04-09
4.5.1 列舉型別初體驗	04-09
4.5.2 列舉型別的使用方式	04-11
4.5.3 進階型列舉型別	04-12
4.6 使用巢狀類別	04-16
4.6.1 巢狀類別的目的與分類	04-16
4.6.2 匿名巢狀類別	04-18
4.6.3 巢狀類別綜合範例	04-21
4.7 認證考試命題範圍	04-22
本章擬真試題實戰	04-24

Chapter 05 使用interface

5.1 使用 Interface	05-02
5.1.1 實作「取代機制」	05-02
5.1.2 Interface 設計要點	05-02
5.1.3 Marker interfaces	05-05
5.2 使用設計模式	05-06
5.2.1 設計模式和 interface	05-06
5.2.2 DAO 設計模式	05-06
5.2.3 工廠設計模式的使用契機	05-07
5.3 使用複合	05-10
5.3.1 重複使用程式碼的技巧	05-10
5.3.2 解決複合設計的難題	05-10
5.3.3 方法的委派和轉交	05-14
5.4 認證考試命題範圍	05-14
本章擬真試題實戰	05-15

Chapter 06 泛型和集合物件

6.1 泛型 (Generics)	06-02
6.2 集合物件 (Collections)	06-04
6.2.1 集合物件的定義和種類	06-04
6.2.2 List	06-05
6.2.3 自動裝箱 (Boxing) 和開箱 (Unboxing)	06-07
6.2.4 Set	06-08
6.2.5 Deque	06-10
6.3 Map	06-11
6.4 集合物件成員的排序	06-14
6.4.1 排序的做法	06-14
6.4.2 使用 Comparable 介面排序	06-15

6.4.3 使用 Comparator 介面排序	06-18
6.5 認證考試命題範圍	06-21
本章擬真試題實戰	06-22

Chapter 07 Exceptions和Assertions

7.1 Exceptions.....	07-02
7.1.1 使用 try-catch 程式碼區塊	07-03
7.1.2 使用 try-with-resources 程式碼區塊	07-06
7.1.3 Suppressed Exceptions	07-08
7.1.4 使用 multi-catch 語法	07-10
7.1.5 使用 throws 宣告	07-12
7.1.6 Exception 物件的捕捉再拋出	07-14
7.1.7 建立客製的 Exception 類別	07-15
7.2 Assertions.....	07-17
7.2.1 Assertions 的簡介和語法	07-17
7.2.2 Assertions 的使用	07-18
7.3 認證考試命題範圍	07-21
本章擬真試題實戰	07-22

Chapter 08 Java I/O基礎

8.1 基礎 I/O	08-02
8.1.1 何謂 I/O ?	08-02
8.1.2 處理串流的類別	08-03
8.1.3 串流類別的串接	08-06
8.2 由主控台讀寫資料	08-10
8.2.1 主控台的 I/O	08-10
8.2.2 使用標準輸出方法	08-10
8.2.3 使用標準輸入由主控台取得輸入資料	08-10

8.2.4 java.io.Console 類別介紹	08-12
8.3 Channel I/O	08-13
8.4 使用序列化技術讀寫物件	08-14
8.4.1 了解序列化技術	08-14
8.4.2 定義物件保存的版本號碼	08-15
8.4.3 序列化和反序列化範例	08-16
8.5 認證考試命題範圍	08-21
本章擬真試題實戰	08-22

Chapter 09 NIO.2

9.1 NIO.2 基礎	09-02
9.1.1 java.io.File 的限制	09-02
9.1.2 Java I/O 套件發展歷史	09-02
9.1.3 檔案系統、路徑和檔案	09-03
9.1.4 Symbolic Links	09-03
9.1.5 NIO.2 的基本架構	09-05
9.2 使用 Path 介面定義檔案 / 目錄	09-05
9.2.1 Path 介面和其主要功能	09-05
9.2.2 移除 Path 的多餘組成	09-07
9.2.3 建立子路徑	09-08
9.2.4 結合 2 個路徑	09-08
9.2.5 建立連接 2 個路徑的路徑	09-09
9.2.6 連結檔案	09-09
9.3 使用 Files 類別操作檔案 / 目錄	09-10
9.3.1 檔案 / 目錄的基本處理	09-10
9.3.2 複製和移動檔案 / 目錄	09-13
9.3.3 Stream 和 Path 互相複製	09-15
9.3.4 列出目錄內容	09-15
9.3.5 讀取和寫入檔案	09-16

9.4 使用 Files 類別操作 channel 和 stream I/O	09-16
9.5 讀寫檔案 / 目錄的屬性	09-19
9.5.1 使用 Files 管理屬性資料	09-19
9.5.2 讀取檔案屬性	09-20
9.5.3 修改檔案屬性	09-21
9.6 遞迴存取目錄結構	09-25
9.6.1 對檔案目錄進行遞迴操作	09-25
9.7 使用 PathMatcher 類別找尋檔案 / 目錄	09-28
9.7.1 搜尋檔案	09-28
9.7.2 glob 樣式語法介紹	09-29
9.8 其他	09-31
9.8.1 使用 FileStore 類別	09-31
9.8.2 使用 WatchService	09-33
9.8.3 由基礎 I/O 轉換至 NIO.2	09-34
9.9 認證考試命題範圍	09-34
本章擬真試題實戰	09-35

Chapter 10 執行緒

10.1 執行緒介紹	10-02
10.1.1 名詞說明	10-02
10.1.2 常見的效能瓶頸	10-02
10.1.3 執行緒類別	10-03
10.2 執行緒常見的問題	10-04
10.2.1 使用 Shared Data 可能造成的問題	10-04
10.2.2 使用 Non-Atomic Operations 可能造成的問題	10-06
10.2.3 使用 Cached Data 可能造成的問題	10-07
10.3 執行緒的 synchronized 與等待	10-08
10.3.1 使用 synchronized 關鍵字	10-08

10.3.2 使用 synchronized 的時機	10-10
10.3.3 縮小 synchronized 的程式區塊	10-13
10.3.4 其他執行等待的情況	10-13
10.4 其他執行緒方法介紹	10-16
10.4.1 使用 interrupt() 方法	10-16
10.4.2 使用 sleep () 方法	10-16
10.4.3 使用其他方法	10-17
10.4.4 不建議使用的方法	10-20
10.5 認證考試命題範圍	10-21
本章擬真試題實戰	10-22

Chapter 11 執行緒與並行API

11.1 使用並行 API	11-02
11.1.1 並行 API 介紹	11-02
11.1.2 AtomicInteger 類別	11-02
11.1.3 ReentrantReadWriteLock 類別	11-03
11.1.4 執行緒安全的集合物件	11-06
11.1.5 常用的同步器工具類別	11-07
11.2 使用 ExecutorService 介面同時執行多樣工作	11-09
11.2.1 使用更高階的多執行緒執行方案	11-09
11.2.2 ExecutorService 概觀	11-09
11.2.3 使用 java.util.concurrent.Callable	11-10
11.2.4 使用 java.util.concurrent.Future	11-11
11.2.5 關閉 ExecutorService	11-13
11.2.6 ExecutorService 完整範例	11-13
11.2.7 ExecutorService 進階範例	11-14
11.3 使用 Fork-Join 框架	11-20
11.3.1 平行處理的策略	11-20

11.3.2 套用 Fork-Join 框架	11-21
11.3.3 Fork-Join 框架的使用建議	11-28
11.4 認證考試命題範圍	11-28
本章擬真試題實戰	11-30

Chapter **12** 使用 JDBC 建立資料庫連線

12.1 了解 Database、DBMS 和 SQL	12-02
12.1.1 基本名詞介紹	12-02
12.1.2 使用 SQL 存取資料庫	12-03
12.1.3 Derby 資料庫介紹	12-05
12.1.4 操作 Derby 資料庫	12-07
12.2 使用 Eclipse 連線並存取資料庫	12-09
12.2.1 連線資料庫	12-09
12.2.2 存取資料表	12-16
12.2.3 建立預存程序	12-18
12.3 使用 JDBC	12-24
12.3.1 JDBC API 概觀	12-24
12.3.2 取得 JDBC 驅動程式	12-25
12.3.3 開發 JDBC 程式	12-31
12.3.4 結束 JDBC 相關物件的使用	12-34
12.3.5 開發可攜式的 JDBC 程式碼	12-35
12.3.6 使用 <code>java.sql.SQLException</code> 類別	12-36
12.3.7 Statement 介面與 SQL 敘述的執行	12-37
12.3.8 使用 <code>ResultSetMetaData</code> 介面	12-37
12.3.9 取得查詢結果的資料筆數	12-38
12.3.10 控制 <code>ResultSet</code> 每次由資料庫取回的筆數	12-39
12.3.11 使用 <code>PreparedStatement</code> 介面	12-40
12.3.12 使用 <code>CallableStatement</code> 介面	12-42

12.4 使用 JDBC 進行交易	12-43
12.4.1 何謂資料庫交易 ?	12-43
12.4.2 使用 JDBC 的交易	12-45
12.5 使用 JDBC 4.1 的 RowSetProvider 和 RowSetFactory	12-46
12.6 回顧 DAO 設計模式	12-47
12.7 認證考試命題範圍	12-47
本章擬真試題實戰	12-49

Chapter 13 Java的區域化(Localization)

13.1 了解 Java 的軟體區域化做法	13-02
13.1.1 使用 Locale 類別	13-02
13.1.2 建立多國語系文字檔	13-03
13.1.3 使用 ResourceBundle 類別	13-04
13.2 使用 DateFormat 類別	13-06
13.3 使用 NumberFormat 類別	13-08
13.4 認證考試命題範圍	13-09
本章擬真試題實戰	13-10

Chapter 14 Interfaces與lambda表示式的應用

14.1 使用 interface 的 static 和 default 方法	14-02
14.2 使用 lambda 表示式	14-10
14.3 使用基礎的內建 functional interfaces	14-15
14.4 在泛型內使用萬用字元	14-19
14.5 使用進階的內建 functional interfaces	14-23
14.6 使用方法參照	14-30
14.7 認證考試命題範圍	14-34
本章擬真試題實戰	14-35

Chapter 15 使用 Stream API

15.1 建構者模式和方法鏈結	15-02
15.2 使用 Optional 類別	15-05
15.3 Stream API 介紹	15-09
15.3.1 介面 Iterable 和 Collection 的擴充	15-09
15.3.2 Stream API	15-11
15.4 Stream API 操作	15-14
15.4.1 中間作業	15-15
15.4.2 終端作業	15-23
15.4.3 短路型終端作業	15-32
15.5 Stream API 和 NIO.2	15-36
15.6 Stream API 操作平行化	15-40
15.6.1 平行化的前提	15-40
15.6.2 平行化的做法	15-42
15.6.3 Reduction 操作	15-45
15.7 認證考試命題範圍	15-49
本章擬真試題實戰	15-51

Chapter 16 Date/Time API

16.1 Java 8 在 Date 和 Time 相關類別的進步	16-02
16.2 當地日期與時間	16-03
16.3 時區和日光節約時間	16-07
16.3.1 時區和日光節約時間的簡介	16-07
16.3.2 Java 8 在時區和日光節約時間的應用	16-10
16.4 描述日期與時間的數量	16-19
16.5 認證考試命題範圍	16-23
本章擬真試題實戰	16-24

Java類別與 基本語法複習

-
- | 1.1 建立Java 類別
 - | 1.2 使用package、import 敘述
 - | 1.3 使用選擇結構和重複結構
 - | 1.4 參數傳遞機制
 - | 1.5 記憶體使用機制

1.1 建立 Java 類別

本書提供 Java 學習的進階內容，在協助讀者有效率取得 OCPJP 證照。第一章僅將 Java 的類別定義和基本語法做一個快速導覽，若讀者有需要做更清楚的了解，可以參閱本書前冊《Java SE8 OCAJP 專業認證指南》。

類別架構

撰寫一個 Java 類別時，基本組成如下：

<code>package <package_name>;</code>	相似功能類別放在同一 package，非必要。
<code>import <other_packages>;</code>	匯入其他 package 的類別，非必要。
<code>class ClassName {</code>	定義類別起始，必要。
<code> <variables / fields>;</code>	定義屬性 (variables / fields)，非必要。
<code> <constructor method(s)>;</code>	定義建構子，會提供預設，非必要。
<code> <other methods>;</code>	定義行為，非必要。
<code>}</code>	定義類別結束，必要。

依照前述定義，可以了解一個最簡單的類別是這樣：

範例

```
1 class Simple {
2 }
```

為了執行類別，可以加上 main 方法：

範例

```
1 public class Simple {
2     public static void main (String args[]){
3         //...
4     }
5 }
```

其中 main 方法是 Java SE 程式執行時的進入點。

程式碼區塊

Java 的程式碼都定義在「程式碼區塊」，也就是 {} 內。可以分成二種：

- 類別宣告後的 {} 內定義類別所有欄位 (fields) 和方法 (methods)，稱為「class scope」。該範圍內所有成員皆可互相存取。欄位又稱為「實例變數 (instance variables)」。

2. 方法的內容定義在 {} 內，稱為「method scope」。方法內宣告的變數的有效範圍就在該 {} 內，稱為「區域變數 (local variables)」。

定義變數

使用變數前必須先宣告型別。Java 的型別有二種：

1. 基本型別。
2. 參考型別。

其中基本型別有以下八類：

◆表 1-1 Java 基本型別分類

類型 (types)	型別	位元組 (bytes)	位元數 (bits)	最小值	最大值
整數 Integral	byte	1	8	-128	127
	short	2	16	-2^{15}	$2^{15}-1$
	int	4	32	-2^{31}	$2^{31}-1$
	long	8	64	-2^{63}	$2^{63}-1$
浮點數 Floating point	float	單精度, 32-bit 浮點數		依 IEEE 754 標準	
	double	雙精度, 64-bit 浮點數		依 IEEE 754 標準	
字元 Textual	char	2	16	'\u0000' - '\uffff'	
布林值 Logical	boolean	1	8	true, false	

若要知道某個整數或浮點數類型的基本型別的最大 / 最小值，可以藉助其 wrapper 類別。如：

- int 的最大值 / 最小值為： Integer.MAX_VALUE / Integer.MIN_VALUE 。
- double 的最大值 / 最小值為： Double.MAX_VALUE / Double.MIN_VALUE 。

變數初始值

Java 的原則是「使用前一定要有值，亦即需要初始化」。

- 實例變數 (instance variables)：即便沒有給值，因為 Java 會偷偷給預設值，所以沒問題。
- 區域變數 (local variables)：沒有預設值，所以一定要明確給值，否則會出現編譯失敗訊息：「The local variable ~ may not have been initialized」。

新增二進位常量的表示方式

Java 7 之後支援二進位常量 (Binary Literals) 的表示方式，只要在二進位數字之前加上「0b」或「0B」。如範例「/OCP/src/course/c01/BinaryLiterals.java」：

範例

```

1 public static void main(String[] args) {
2     // 8-bit 'byte' value:
3     byte b = 0b0001_0001;
4     // 16-bit 'short' value:
5     short s = (short) 0b1001_0001_0100_0101;
6     // 32-bit 'int' values:
7     int i1 = 0b1001_0001_0100_0101_1010_0001_0100_0101;
8 }
```

改善數字常量的可讀性

Java 7 之後可以在數字常量(numeric literals)內使用「_」提高數字常量的可讀性(readability)。如：

範例

```

1 public static void main(String[] args) {
2     long creditCardNumber = 1234_5678_9012_3456L;
3     long socialSecurityNumber = 999_99_9999L;
4     long hexBytes = 0xFF_EC_DE_5E;
5     long hexWords = 0xCAFE_BABE;
6     long maxLong = 0x7fff_ffff_ffff_ffffL;
7     byte nybbles = 0b0010_0101;
8     long bytes = 0b11010010_01101001_10010100_10010010;
9 }
```

但須注意不能放在以下地方：

- 數字前後。
- 鄰近小數點。
- 在 F 或 L 前。
- 預期數字該出現的地方。

如範例「/OCP/src/course/c01/NumericLiterals.java」：

範例

```

1 public static void main(String[] args) {
2     float pi1 = 3_.1415F;           // 編譯失敗!
3     float pi2 = 3._1415F;          // 編譯失敗!
4     long l = 999_99_9999_L;        // 編譯失敗!
5     int x1 = _52;                 // 編譯失敗!
```

```

6   int x2 = 5_2;           // OK
7   int x3 = 52_;          // 編譯失敗！
8   int x4 = 5_____2;     // OK
9   int x5 = 0_x52;        // 編譯失敗！
10  int x6 = 0x_52;       // 編譯失敗！
11  int x7 = 0x5_2;       // OK
12  int x8 = 0x52_;       // 編譯失敗！
13  int x9 = 0_52;        // OK
14  int x10 = 05_2;       // OK
15  int x11 = 052_;       // 編譯失敗！
16 }

```

課堂小祕訣

OCP 考試喜歡找一些特殊的狀況來測驗是否可以通過編譯。狀況百百種，有一個簡單口訣幫讀者記憶：「不能前後，不能左右，不能中斷」。

- 「不能前後」指不能在數字前後。
- 「不能左右」指不能在特殊符號或字母左右，如小數點、F、L 等。
- 「不能中斷」指不能插入特殊定義間，如 16 進位字首「0x」和「0X」，和 2 進位字首「0b」和「0B」。這個新功能允許插入「_」讓長數字的可讀性提高，就好像習慣在代表金錢的數字上，每千分位就使用「,」區隔，因此原本就該出現在數字之間。

常用運算子

Java 也是計算機語言，常用運算子為：

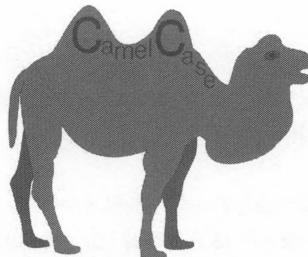
◆ 表 1-2 常用運算子列表

指定符號 (Assignment)	
=	Assign
數學運算子 (Arithmetic operators)	
+	加
-	減
*	乘
/	除
%	餘數
一元運算子 (Unary operators)	
+	正
-	負
++	加 1
--	減 1
!	否定

Java 命名習慣

Java 對於類別、變數、方法的命名習慣為：

1. 類別名稱：大寫開頭。
2. 變數、方法名稱：小寫開頭。
3. 常數：全部大寫，必要時以底線分隔複合字。
4. 名稱為複合字：使用「camel case」，亦即「駝峰型」命名原則，由第二個單字開始，每個字首都是大寫。因為很像駱駝背上的駝峰，故名：



❖ 圖 1-1 駝峰型命名原則示意圖（來源：維基百科）

建立類別

定義類別 Employee，如範例「/OCP/src/course/c01/Employee.java」：

範例

```
1 public class Employee {           // 類別
2     public final int COMPANY_ID = 1234567890; // 常數
3     private int empId;                  // 欄位
4     private String name;                // 欄位
5     private String ssn;                 // 欄位
6     private double salary;              // 欄位
7     public Employee() {                // 建構子
8     }
9     public int getEmpId() {            // getter 方法
10        return this.empId;
11    }
12    public void setEmpId (int empId) { // setter 方法
13        this.empId = empId;
14    }
15 }
```

若類別內有欄位，習慣上會將欄位宣告為 private，並增設對應的方法：

- 儲存資料：setter 方法，如 setEmpId()。

- 取得資料：getter 方法，如 getEmpId()。

方法內的關鍵字「this」，指物件自己。在變數前面加上「this」，可區隔方法內的變數是方法傳入的參數 (parameters)，或是類別欄位\ 實例變數。

定義建構子來建立物件

沒有參數的建構子稱「no-arg constructor」。如：

範例

```
1 public class Employee {
2     public Employee() {}
3 }
```

若程式設計人員未建立任何建構子時，如：

範例

```
1 public class Employee {
2 }
```

此時 Java 會自動提供「預設建構子 (default constructor)」，但程式碼裡看不到，該建構子也不帶參數。

建構子可以傳入參數。此稱為建構子的多載 (overloading)，如：

範例

```
1 public class Employee {
2     public Employee(int empId, String name) {
3         //...
4     }
5 }
```

使用「new」關鍵字呼叫 Employee 類別的任一建構子來建構 Employee 物件實例。建構完成後，以參考型別的變數 emp 指向該物件實例。因為變數 emp 位在記憶體 stack 裡，而 Employee 物件實例在記憶體 heap 上，因此形成使用 emp 變數遠端操控 Employee 物件的方式。可以比擬為控制 3C 產品如電視的「遙控器」，所以變數 emp 又稱為「物件參考 (object reference)」。

我們在前冊《Java SE8 OCAJP 專業認證指南》中使用「遙控器」的概念貫穿整個物件導向程式語言，若這部分觀念不清楚，讀者可以自行參閱。

取得物件參考 emp 變數後，可以使用來呼叫物件方法，就和遙控器一樣：

```
Employee emp = new Employee();
emp.setEmpId(101); // use a method instead
emp.setName("Jim Tzeng");
emp.setSsn("011-22-3467");
emp.setSalary(120345.27);
```

字串類別

Java 裡經常使用到字串 (String) 物件，為了避免相同字串不斷被建立而浪費記憶體空間，Java 使用「字串池 (String Pool)」的概念，相同字串預設將重複使用。因此必須知道建立字串物件不建議使用「new」關鍵字，因為：

- 將強制生成新字串。
- 無法重複使用字串。

如範例「/OCP/src/course/c01/StringTest.java」：

範例

```
1 public static void testEquality () {
2     String s1 = "jim";
3     String s2 = "jim";
4     String s3 = new String("jim"); // 非必要，勿使用！
5     System.out.println(s1 == s2); // true
6     System.out.println(s2 == s3); // false
7     System.out.println(s1 + s2); // 字串相連
8 }
```

因為運算子「==」用來判斷物件參考是否指向同一記憶體位址，所以結果為：

結果

```
true
false
jimjim
```

使用字串其他方法：

範例

```
1 public static void testFun() {
2     String s1 = "World";
3     String s2 = "";
```

```

4     s2 = "Hello".concat(s1);           // 字串連接
5     System.out.println("String2: " + s2);
6     System.out.println("Length: " + s2.length());    // 取得長度
7     System.out.println("Sub: " + s2.substring(0, 4)); // 取得子字串
8     System.out.println("Upper: " + s2.toLowerCase()); // 轉換為大寫
9 }

```

結果

```

String2: HelloWorld
Length: 10
Sub: Hell
Upper: helloworld

```

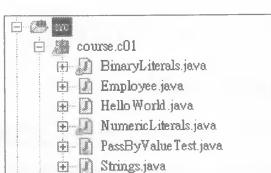
1.2 使用 package、import 敘述

使用 package 敘述

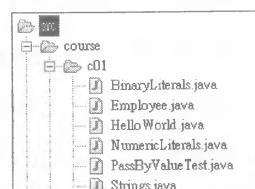
Java 中使用「套件 (package)」關鍵字將性質接近的類別「群組」起來。有二個主要意涵：

- 相同 package 的類別表示程式碼位在作業系統裡的同一資料夾內。
- package 提供「名稱空間 (namespace)」的概念給類別。讓相同名稱的類別，只要 package 不同，透過「package.class」的名稱，依然可以辨識。

比較如下圖：



◆ 圖 1-2 以 Java 套件顯示



◆ 圖 1-3 以檔案系統目錄結構顯示

使用 import 敘述

使用 import 敘述可以協助辨識其他套件的類別。如設計類別的時候，需要使用到日期類別 Date，就必須以下選項二擇一：

- 使用 `java.util.Date` 完整名稱。
- 先 `import java.util.Date`，接下來程式碼只要使用到 `Date` 類別，將自動認定為 `java.util.Date`。

類別 `Date` 在 Java 標準函式庫內有二個同名類別，如下。因此明確指定類別全名有其必要：

- `java.util.Date`
- `java.sql.Date`

「`java.util`」是 `Date` 類別所屬套件。可以同時 `import` 多個相同套件下的類別。如：

範例

```
1 import java.util.Date;  
2 import java.util.Calendar;
```

也可以使用萬用字元「*」一次 `import` 整個套件，但應減少使用，避免一些可能衝突，如：

範例

```
1 import java.util.*;      // 含 Date  
2 import java.sql.Date;
```

因為 `import java.util.*` 時包含 `java.util.Date`，又同時 `import java.sql.Date`，在接下來的程式碼中若使用 `Date` 類別，就不容易清楚究竟是來自 `java.util.Date` 或是 `java.sql.Date`。

其他使用 `import` 敘述必須注意的事項：

- 不一定需要。但若有則必須在 `package` 告知之後，在 `class` 告知之前。
- Java 會自動 `import java.lang.*`。因為該套件是 Java 語言基礎。
- 同一 `package` 內的類別毋須 `import`。這觀念有時被解讀為預設已經被 Java 自動 `import`，如 `java.lang.*`。OCP 考試時可以藉由題目要求的選項數目來判定該如何作答。

1.3 使用選擇結構和重複結構

選擇結構

選擇結構使用「if」敘述，和「邏輯運算子」的使用息息相關，主要有以下三類：

❖ 表 1-3 常用邏輯運算子列表

關係運算子 (Relational operators)	
==	是否相等
!=	是否不相等
>	是否大於
>=	是否大於或等於
<	是否小於
<=	是否小於或等於
條件運算子 (Conditional operators)	
&&	且
	或
?:	三元運算子，if-then-else 的簡式
型態比較運算子 (Type comparison operator)	
instanceof	比較物件是否屬於某種型態

選擇結構搭配邏輯運算子判斷條件是否成立，是否執行某程式區塊。範例如下：

範例

```

1 public static void main(String args[]) {
2     long x = 10;
3     long y = 20;
4     if (x == y) {
5         System.out.println("True");
6     } else {
7         System.out.println("False");
8     }
9 }
```

除了 if 敘述外，選擇結構也可以使用 switch 敘述。Java 7 之前，switch 敘述除支援：「byte、short、char、int」等基本資料型別外，也可以使用「enum」。

Java 7 之後，開始支援「String」。範例如下：

範例

```

1 public static void main(String args[]) {
2     String test = "Blue";
3     String color = null;
4     switch (test) {
5         case "Blue":      // 若 "Blue" 和 test 變數串內容相同
6             color = "Blue";
7             break;
8         case "Red":       // 若 "Red" 和 test 變數串內容相同
9             color = "Red";
10            break;
11        default:        // 若都不相同
12            color = "White";
13        }
14     System.out.println("Color: " + color);
15 }
```

結果

Color: Blue

陣列和重複結構

Java 重複結構使用迴圈 (loop)，在滿足特定條件 (expression) 時即重複某些行為 (code block)。可以分成三種主要型態：

1. while 迴圈：若滿足 expression = true 時將持續進行。
2. do/while 迴圈：執行一次後，若滿足 expression = true 時將持續進行。
3. for 迴圈：重複特定次數。

其中，for 迴圈推出進階型 (enhanced) 可用於存取 Java 的集合物件 (Collections) 和陣列 (Array)，如以下範例：

範例

```

1 public static void main(String args[]) {
2     String[] names = new String[3];
3     names[0] = "Jim";
4     names[1] = "John";
5     names[2] = "Joseph";
6     for (String name : names) {
7         System.out.println("Name: " + name);
8     }
```

```

9      int[] numbers = { 150, 250, 350 };
10     for (int number : numbers) {
11         System.out.println("Integer: " + number);
12     }
13 }
```

也可以使用傳統 while 迴圈：

範例

```

1 public static void main(String args[]) {
2     for (int i = 0; i < 9; i++) {
3         System.out.println("i: " + i);
4     }
5     int[] numbers = { 100, 200, 300 };
6     int index = 0;
7     while (index < numbers.length) {
8         System.out.println("Number: " + numbers[index]);
9         index++;
10    }
11 }
```

1.4 參數傳遞機制

一般而言，Java 在二種情況時需要傳遞 (pass) 變數 / 參數：

- 由指定運算子「=」的右側，將值 (value) 傳遞給左側的變數。
- 透過方法宣告的參數，將值由呼叫者 (Caller) 方法傳遞進入工作者 (Worker) 方法中。

由範例程式「/OCP/src/course/c01/PassByValueTest.java」可以了解參數傳遞機制在「基本型別」和「參考型別」的不同：

範例

```

1 public class PassByValueTest {
2     public static void main(String[] args) {
3         testPrimitive();
4         testReference();
5     }
6     private static void testPrimitive() {
7         int x = 10;
8         int y = x;
```

```

9      x = 5;
10     System.out.println(y);
11 }
12 private static void testReference() {
13     Employee x = new Employee(1, "Jim");
14     Employee y = x;
15     x.setEmpId(2);
16     System.out.println(y);
17     modifyEmployee(x);
18     System.out.println(x);
19 }
20 private static void modifyEmployee(Employee e) {
21     e = new Employee(3, "Bill");
22     e.setEmpId(4);
23 }
24 }
```

結果

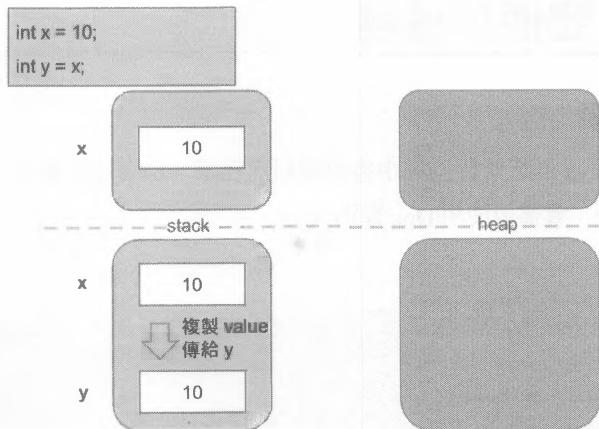
```

10
2-Jim
2-Jim
```

Java 不管對於「基本型別」或是「參考型別」，在傳遞變數時，都是「pass by value」；也就是把變數的「值(value)」，「複製(copy)」一份後再傳遞副本出去。為了方便記憶，也可以想成「pass by copy」。但「基本型別」和「參考型別」對「值」的定義不同：

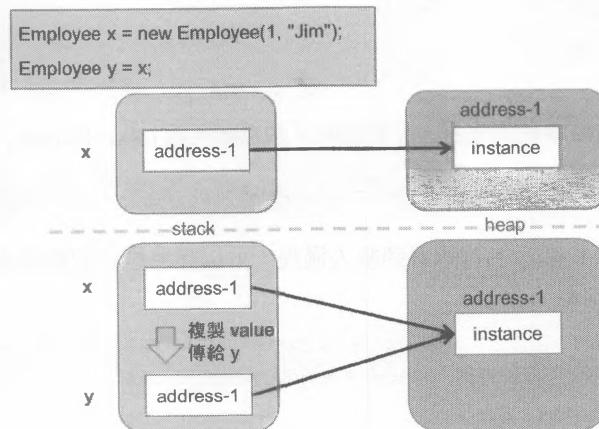
1. 參考型別：複製物件參考(遙控器)後進行傳遞，非複製物件實例。複製前後雖然遙控器不同，但指向同一物件實例。
2. 基本型別：因為沒有遙控器的概念，因此直接複製變數值，如同影印機複製「原稿」後產生「副本」，兩者各自獨立。

基本型別的變數傳遞概念如下：



◆ 圖 1-4 傳遞基本型別變數的記憶體配置示意圖

參考型別的變數傳遞概念如下：



◆ 圖 1-5 傳遞參考型別變數的記憶體配置示意圖

最後，本例方法 `modifyEmployee()` 將傳入的物件參考變數 `e` 重新指向到新建構的實例，故方法內外的物件參考變數根本指向不同實例，所以修改內容將互不影響。簡單的比喻是複製一份遙控器候傳入方法中，複製前和複製後原本都指向同一台電視 A，但傳入方法後將遙控器改指向電視 B，從此二個遙控器分別控制二台不同電視，彼此互不影響。

1.5 記憶體使用機制

Java 類別載入器

執行程式時，Java 會將執行過程中需要的類別在第一次使用時載入。如以下範例，執行 TestDog 類別時，會需要使用 Dog 類別：

範例

```

1  class Dog {
2  }
3  public class TestDog {
4      public void run() {
5          new Dog();
6          System.out.println("done!");
7      }
8      public static void main(String[] args) {
9          new TestDog().run();
10     }
11 }
```

則執行時期，Java 會使用 TestDog 類別的「類別載入器(class loader)」載入 Dog 類別：

```
TestDog.class.getClassLoader().loadClass("Dog");
```

這個過程預設不會顯示。若想看到載入過程，可以在主控台的程式命令列執行 java 時使用「-verbose」選項：

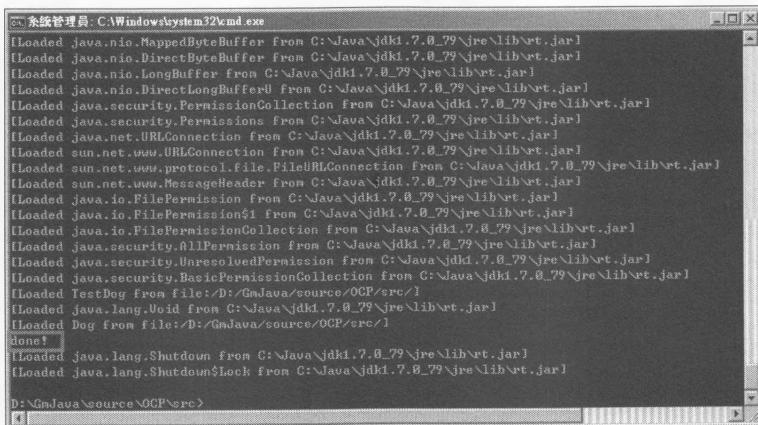
```
java -verbose TestDog
```

如下圖：

The screenshot shows a Windows Command Prompt window titled '系統管理員: C:\Windows\system32\cmd.exe'. The command line shows the path 'D:\GeJava\source\OCP\src' and the execution of 'java -verbose TestDog'. The output displays the class loading process, including the loading of 'java.lang.Object', 'java.util.ArrayList', and 'TestDog' itself, along with the message 'done!'. The window title bar also shows the path 'D:\GeJava\source\OCP\src>java -verbose TestDog'.

◆ 圖 1-6 執行 Java 時使用 -verbose 選項

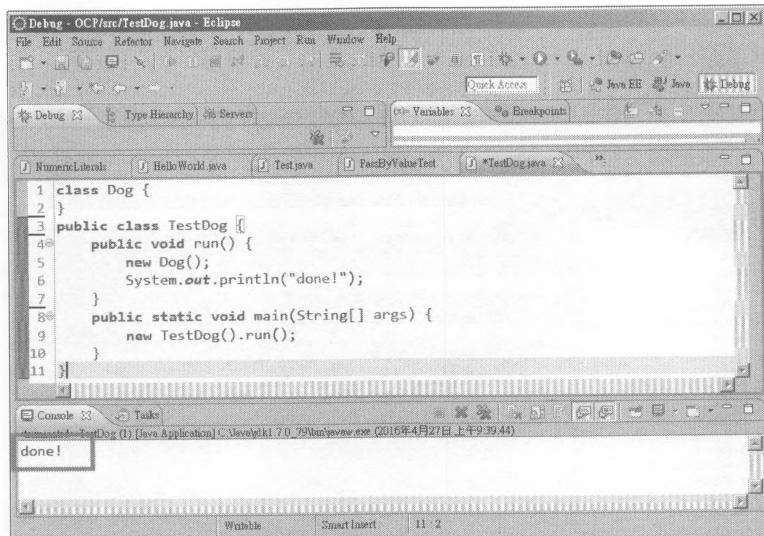
除了執行結果外，還可以看到為了執行 TestDog 所需要載入的所有相關類別：



```
[Loaded java.nio.MappedByteBuffer from C:\Java\jdk1.7.0_79\jre\lib\rt.jar]
[Loaded java.nio.DirectByteBuffer from C:\Java\jdk1.7.0_79\jre\lib\rt.jar]
[Loaded java.nio.DirectLongBufferFactory from C:\Java\jdk1.7.0_79\jre\lib\rt.jar]
[Loaded java.security.PermissionCollection from C:\Java\jdk1.7.0_79\jre\lib\rt.jar]
[Loaded java.net.URLConnection from C:\Java\jdk1.7.0_79\jre\lib\rt.jar]
[Loaded sun.net.www.HttpURLConnection from C:\Java\jdk1.7.0_79\jre\lib\rt.jar]
[Loaded sun.net.www.protocol.FileURLConnection from C:\Java\jdk1.7.0_79\jre\lib\rt.jar]
[Loaded sun.net.www.MessageHeader from C:\Java\jdk1.7.0_79\jre\lib\rt.jar]
[Loaded java.io.FilePermission from C:\Java\jdk1.7.0_79\jre\lib\rt.jar]
[Loaded java.io.FilePermission$1 from C:\Java\jdk1.7.0_79\jre\lib\rt.jar]
[Loaded java.io.FilePermissionCollection from C:\Java\jdk1.7.0_79\jre\lib\rt.jar]
[Loaded java.security.AllPermission from C:\Java\jdk1.7.0_79\jre\lib\rt.jar]
[Loaded java.security.UnresolvedPermission from C:\Java\jdk1.7.0_79\jre\lib\rt.jar]
[Loaded java.security.BasicPermissionCollection from C:\Java\jdk1.7.0_79\jre\lib\rt.jar]
[Loaded TestDog from file:/D:/GmJava/source/OCP/src/]
[Loaded java.lang.Void from C:\Java\jdk1.7.0_79\jre\lib\rt.jar]
[Loaded Dog from file:/D:/GmJava/source/OCP/src/]
done!
[Loaded java.lang.Shutdown from C:\Java\jdk1.7.0_79\jre\lib\rt.jar]
[Loaded java.lang.Shutdown$Lock from C:\Java\jdk1.7.0_79\jre\lib\rt.jar]
```

❖ 圖 1-7 執行 Java 時載入相關類別

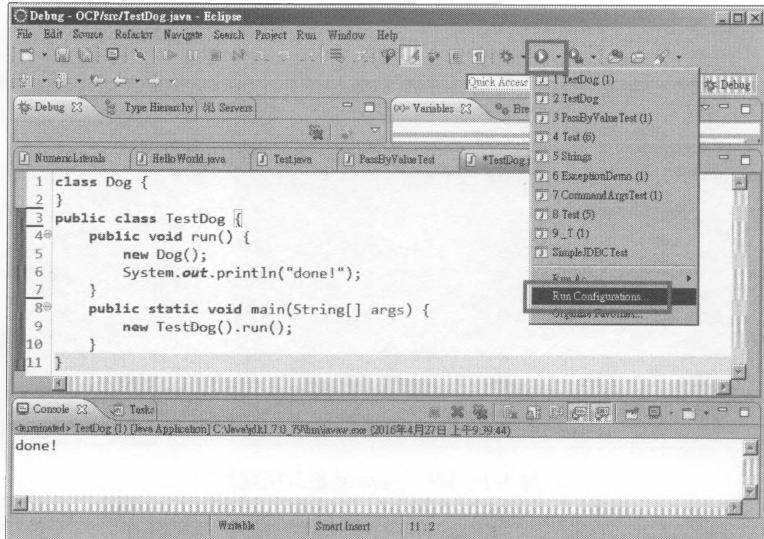
也可以使用 Eclipse。若使用一般的程式執行，會得到結果如下圖：



❖ 圖 1-8 一般程式執行與結果

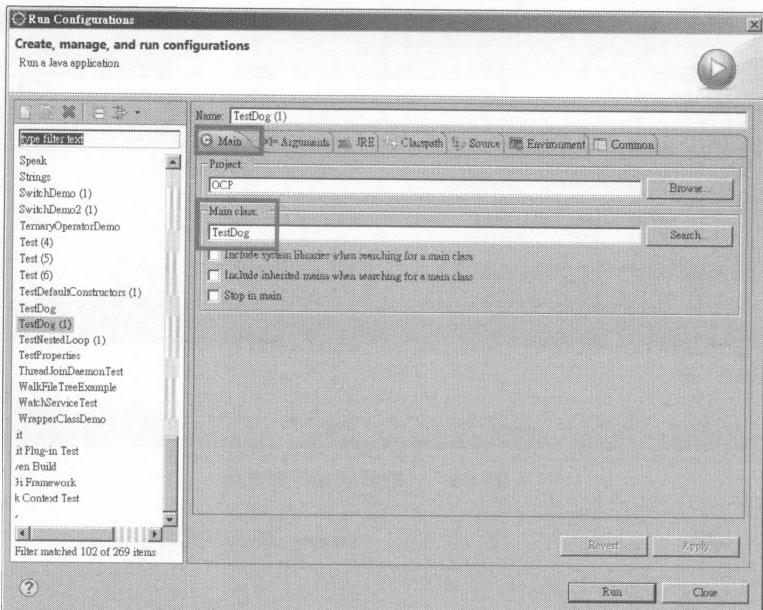
依照以下步驟，可以得到和主控台上使用 -verbose 選項一樣的結果：

STEP01 執行程式時選擇「Run Configurations」：



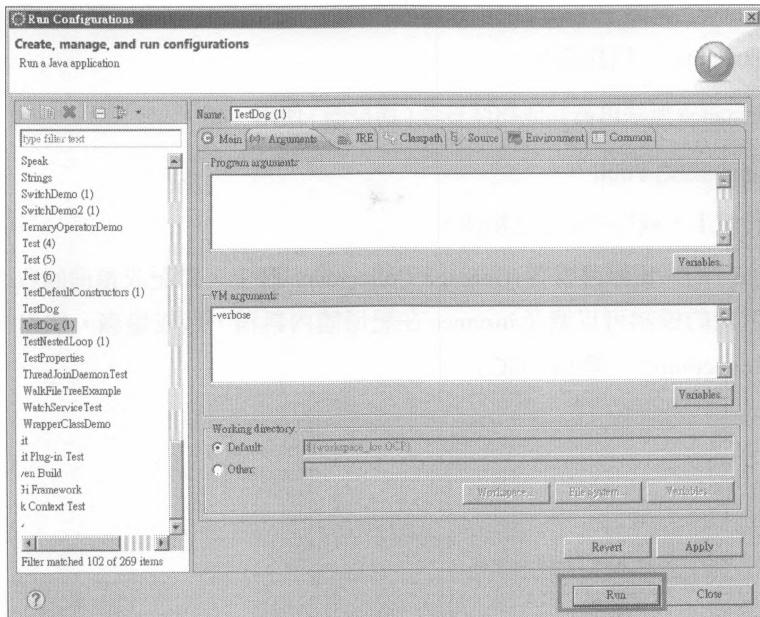
❖ 圖 1-9 修改執行設定

STEP02 選擇要執行的類別：



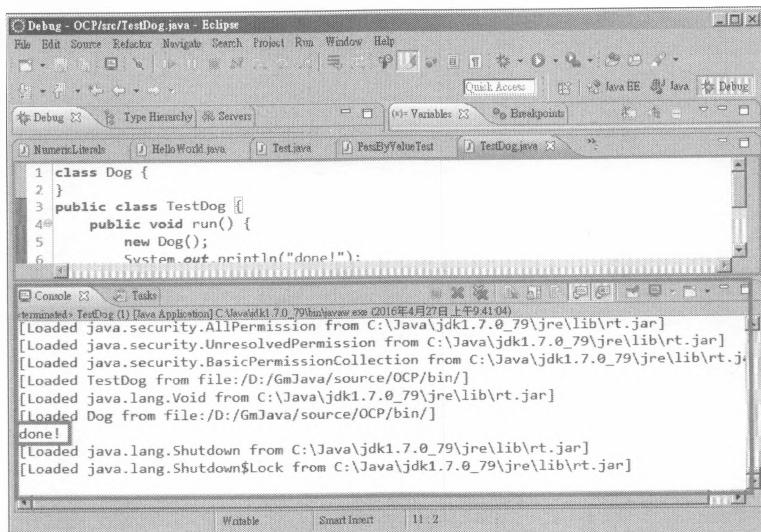
❖ 圖 1-10 選擇要執行的類別

STEP03 在「VM arguments」內輸入「-verbose」後，點選「Run」。



❖ 圖 1-11 填入 -verbose

STEP04 程式執行時，除了印出結果，也會一併顯示載入其他類別的過程。



❖ 圖 1-12 顯示 -verbose 結果

資源回收機制

當使用「new」關鍵字呼叫建構子以建構物件時，記憶體中就會切割出一部分，稱「物件實例 (instance)」，供其使用。

一旦該 instance 無法再被其他物件參考 (遙控器) 使用，如：

1. 原參考變數被設為 null。
2. 方法執行結束，執行中產生之物件。

就會被 Java 的「資源回收者 (Garbage Collector)」盯上，標記成預備回收，再等待機會回收。這樣的機制可以避免 instance 在記憶體內累積，造成爆滿，稱為「資源回收 (Garbage Collection)」，簡稱「GC」。

了解封裝、繼承 和多型

-
- | 2.1 使用封裝
 - | 2.2 建立子類別和使用繼承關係
 - | 2.3 多載方法與使用可變動參數個數的方法
 - | 2.4 使用多型
 - | 2.5 認證考試命題範圍

2.1 使用封裝

何謂封裝

封裝使用的英文是「*Encapsulation*」，字面的解釋是：

- to enclose (封填) in a capsule (膠囊)
- To wrap (隱藏) something around an object to cover (遮蓋) it

用於物件導向程式開發時，就是用來隱藏 Java 物件內部的欄位狀態和商業邏輯，主要用在欄位和方法。可以使用修飾詞 (modifiers) 來決定開放範圍：

- public：表示所有類別均可使用。
- private：限制在自己類別內部使用。

「欄位」封裝後，外界無法知道物件狀態；視需要再提供公開的方法讓其他類別可以間接存取。「方法」隱藏實作細節後，視需要再提供公開介面方法讓封裝的方法可以被轉呼叫。因此只要介面不修改，將不會影響調用方法的類別。

封裝三部曲

以第一章的 Employee 類別「/OCP/src/course/c01/Employee.java」為範例，開始進行「封裝」流程。

1. 首先，將所有欄位變成 private。有需要給外界存取的，再提供 public 方法，亦即「Private Data，Public Methods」。
2. 其次，重新檢視所有方法的設計：
 - 和商業邏輯有關的方法，儘可能封裝成 private，再提供公開介面方法轉呼叫。
 - 方法的命名應該讓使用者對於使用方式和功能意義可以一目了然。方法輸入的參數名稱也是一種輔助說明。
3. 最後，讓類別的整體設計儘可能做到「immutable」。亦即一旦物件生成，除非必要，物件狀態不允許改變：
 - 移除所有 setter 方法，使用建構子設定欄位的初始值。
 - 移除不帶參數 (no-args) 的建構子。如此物件一旦生成，將無法修改資料狀態。

依照這三個原則改寫 Employee 類別後，可以得到範例「/OCP/src/course/c02/Employee.java」的結果：

範例

```

1  public class Employee {
2      public final int COMPANY_ID = 1234567890;
3      private int empId;
4      private String name;
5      private String ssn;
6      private double salary;
7      public Employee(int empId, String name, String ssn, double salary) {
8          this.empId = empId;
9          this.name = name;
10         this.ssn = ssn;
11         this.salary = salary;
12     }
13     public void changeName(String newName) {
14         if (newName != null) {
15             this.name = newName;
16         }
17     }
18     public void raiseSalary(double increase) {
19         this.salary += increase;
20     }
21     public int getEmpId() {
22         return empId;
23     }
24     public String getName() {
25         return name;
26     }
27     public String getSSN() {
28         return ssn;
29     }
30     public double getSalary() {
31         return salary;
32     }
33 }
```

其中，因為商業邏輯需要，有二個方法可以改變物件狀態：

- `changeName (String newName)`
- `raiseSalary (double increase)`

搭配方法名稱和參數名稱，讓人清楚方法的用意。比方說，參數「`newName`」加上方法「`changeName`」，很清楚目的是「改變名稱」，而非「設定名稱」。參數「`increase`」加上方法「`raiseSalary`」，很清楚目的是「增加薪資」，和「設定薪資」不同。

2.2 建立子類別和使用繼承關係

2.2.1 繼承的目的和建立子類別

繼承的目的

Java 使用「繼承」的目的：

- 物件導向程式語言的一大特色。
- 藉由繼承，子類別可以直接使用父類別方法，達成程式碼重複使用 (code reuse) 的目的。
- 將方法設計為依賴父類別 (super class) 或介面 (interface)，讓程式碼可以以一般化 (generalization) 的原則處理更多的子類別，不會因為頻繁新增子類別而改動程式。

因此，我們應該儘可能使用最一般化的參考型別，亦即「coding for generalization」。

建立子類別

建立子類別是一種「特別化、專業化 (specialization)」的過程。繼承父類別後的子類別，除具備父類別的屬性方法外，也會有自己獨特且更專業化的特質。

承前一範例，Employee 類別是基礎類別，公司內的人員都是員工。特殊的角色如 Manager 類別，繼承 Employee 類別後，因為需要管理部門，會再擁有 managedDept 的屬性和相關 getter 方法：

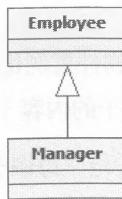
範例

```

1  public class Manager extends Employee {
2      private String managedDept;
3      public Manager(int empId, String name, String ssn, double salary,
4                      String managedDept) {
5          super (empId, name, ssn, salary);
6          this.managedDept = managedDept;
7      }
8      public String getManagedDept () {
9          return managedDept;
10     }

```

以 UML 來繪製二個類別的繼承關係，會像這樣：



◆ 圖 2-1 Manager 繼承 Employee 的 UML 類別圖

使用 Manager 類別來建構物件：

範例

```

1 Manager mgr = new Manager (102, "Barbara Jones", "107-99-9078", 109345.67,
                           "Marketing");
2 mgr.raiseSalary (10000.00);           // 來自繼承
3 String dept = mgr.getManagedDept (); // 自己特化
  
```

2.2.2 建立子類別的建構子

建構子的建立方式

建構子 (constructor) 不是物件成員，非屬性或方法，所以無法繼承取得。但因為需要用來生成物件，所以是必要的類別組成，有以下方法可以建立：

1. 若程式設計人員「沒有」建立自己的建構子，則 Java 主動提供「無參數」的建構子，稱為「預設建構子 (default constructor)」。
2. 若程式設計人員「有」建立自己的建構子，則 Java 不再提供預設建構子。若仍需要「無參數建構子 (no-args constructor)」，就必須自己建立。
3. 建構子可以使用「多載 (overloading)」的技巧建立多個。因為必須和類別同名，所以只能藉由不同的參數組合，區分不同建構子，也代表不同的物件建構方式。

使用 this 和 super 關鍵字

在類別內呼叫自己的建構子或是父類別的建構子時，不是使用建構子名稱，而是：

- 存取自己的建構子，使用「this()」加上需要傳入的參數。
- 存取父類別的建構子，使用「super()」加上需要傳入的參數。

關鍵字「this」和「super」加上運算子「.」也可以用來分別存取自己類別和父類別的物件成員，也就是欄位和方法。可以想成指向自己和父類別的另類遙控器。

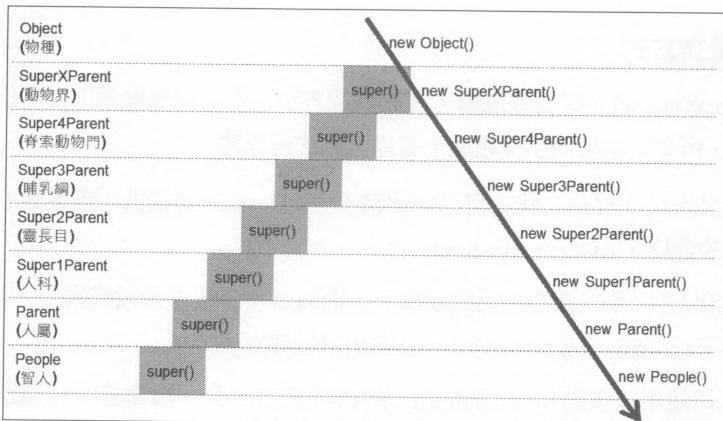
建構子的運作方式

建構子在繼承部分的基本原則是：「物件實例化的過程裡，為了讓子類別可以取得父類別屬性和方法，必須先執行父類別建構子的內容，再執行子類別建構子的內容」。

為了達成這個原則，Java 採取的手段是「每個子類別的建構子執行內容前，都必須先呼叫一個父類別的建構子」。可以分成二種情況：

1. 若父類別的「無參數的建構子」存在，且程式設計人員沒有主動呼叫父類別的建構子，則不管該子類別的建構子是預設還是自己建立，Java 都會自動在每個子類別的建構子的第一行程式碼主動呼叫 super()。
2. 若父類別的「無參數的建構子」不存在，則程式設計人員必須幫「每個」子類別的建構子都各找一個合適的父類別的建構子，且在「第一個動作」呼叫。

執行建構子時第一件事必須呼叫父類別建構子，可以讓子類別在生成時像是「踩階梯」逐步往上爬到最高層 Object 類別，再順勢「溜滑梯」執行繼承結構裡的每一階建構子，示意圖如下。完整觀念可參考本書前冊第十二章：



❖ 圖 2-2 建構子執行示意圖 (節錄自本書前冊第十二章)

範例「/OCP/src/course/c02/TestConstructors.java」是考試常見的小陷阱，可以用來釐清觀念：

❶ 範例

```

1  class Super {
2      Super(String s) {
3          System.out.println("Super");
4      }

```

```

5 }
6 class Sub extends Super {
7     Sub(String s) {
8         super(s);
9         System.out.println("Sub");
10    }
11    Sub(String s1, String s2) {
12        this(s1);
13        System.out.println("Sub");
14    }
15 }
16 public class TestConstructors {
17     public static void main(String[] args) {
18         new Sub("Jim");
19         System.out.println("-----");
20         new Sub("Hi", "Jim");
21     }
22 }
```

結果

```

Super
Sub
-----
Super
Sub
Sub
```

說明

12 這個範例的關鍵是「行 12 的程式碼是否可以通過編譯？」因為看似沒有馬上呼叫父類別建構子。但這裡呼叫 `this(s1)` 後，程式碼會跳到第 8 行，然後馬上執行 `super()`。子類別建構子在執行自己工作之前，還是很盡責地先呼叫父類別的建構子，因此可以通過編譯。

由這個範例，我們可以知道「**子類別建構子可以藉由呼叫其他多載 (overloaded) 的子類別建構子，再呼叫父類別建構子**」。為了避免這種彈性設計演變成「子類別建構子一次跳級多階父類別建構子」的失控局面：

範例

```

1 class SuperClass {
2 }
3 class ChildClass {
4     ChildClass(int x) {
```

```

5      super();      // 踏階梯到父類別
6      this();       // 再踏一次階梯到父類別的父類別！
7  }
8  ChildClass() {
9      super();
10 }
11 }
```

所以，Java 的限制是：

- 每個子類別建構子只能存在一個「自己類別多載的建構子」或是「父類別建構子」，而且必須是在第一行程式碼。亦即「Constructor call must be the first statement in a constructor」。
- 若未做到上述規則，且父類別存在無參數建構子的情況下，Java 預設自動加入父類別的無參數建構子。

2.3 多載方法與使用可變動參數個數的方法

2.3.1 多載方法

- 方法宣告的「名稱 + 參數」，合稱「簽名(signature)」。簽名用來識別方法，好比人的名字有時相同，但簽名字跡不同，所以拿來辨認。因此類別內的每個方法必須有不同的簽名，用以識別。
- 方法在簽名不同時，若名稱相同，稱為「多載方法(overloading)」，用於相似功能的方法。

常用的 `System.out.print()` 就是最好範例。我們這樣設計：

```
void print (int i)
void print (float f)
void print (String s)
```

不會這樣設計：

```
void printInt (int i)
void printFloat (float f)
void printString (String s)
```

因為對呼叫方法的人而言，後者使用時必須熟知不同的方法名稱，不若前者只要一個方法名稱 print，就可以處理所有不同參數。

2.3.2 可變動參數個數的方法

雖然 overloading 很方便，還是會有不足的時候。比如說，需求是設計一個計算平均的方法，如範例「/OCP/src/course/c02/Statistics1.java」。在考慮各種可能的參數的個數和種類組合後，預期必須設計很多方法：

範例

```

1  public class Statistics1 {
2      public float average (int x1, int x2) {
3          return (x1 + x2) / 2;
4      }
5      public float average (int x1, int x2, int x3) {
6          return (x1 + x2 + x3) / 3;
7      }
8      public float average (int x1, int x2, int x3, int x4) {
9          return (x1 + x2 + x3 + x4) / 4;
10     }
11     // 還要更多方法...
12 }
```

為了解決這種情況，可以使用「可變動參數個數的方法」，亦即方法的參數的個數可以自動調整，只要型別一致即可。宣告方式是將方法的參數型別後面加上三個點，如：

```
public float average (int... x1) { }
```

所以原範例升級為「/OCP/src/course/c02/Statistics2.java」：

範例

```

1  public class Statistics2 {
2      public float average(int... nums) {
3          int sum = 0;
4          float result = 0;
5          if (nums.length > 0) {
6              for (int x : nums)
7                  // iterate int array nums
8                  sum += x;
9              result = (float) sum / nums.length;
10         }
11 }
```

```

11         return (result);
12     }
13     public static void main(String args[]) {
14         Statistics2 s = new Statistics2();
15         System.out.println(s.average());
16         System.out.println(s.average(1,2,3,4));
17         System.out.println(s.average(1,2,3,4,5,6));
18         System.out.println(s.average(1,2));
19         int[ ] arr = {1,2,3,4};
20         System.out.println(s.average(arr));
21     }
22 }
```

程式碼 15~20 行顯示只用一個「可變動參數個數的方法」卻可以支援不同數目的輸入參數，甚至可以沒有參數（行 15），或是放入陣列（行 20），是不是很方便呢？

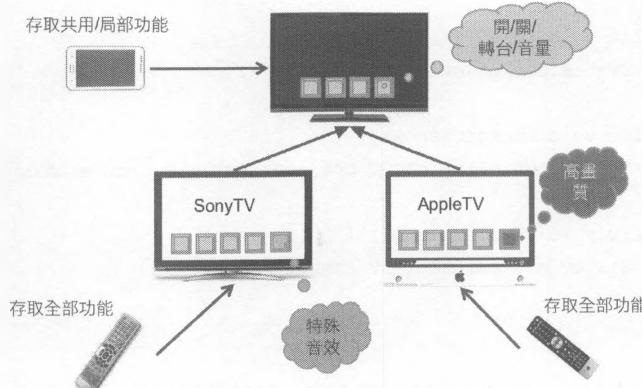
2.4 使用多型

2.4.1 多型的意義

「多型」的英文是「*Polymorphism*」，字面的解釋是「many forms」，所以中文的翻譯為「多型」，可以解釋為「多種型別」。

在 Java 等物件導向的程式語言裡，要參照一個物件實例，除了使用原本的型別外，也可以使用父類別或介面的型別；也就是一個物件實例，可以多種多種型別宣告，所以稱為「多型」。

在 OCA 時我們曾經用過很大的篇幅介紹多型的概念。使用多型的好處在於程式碼的彈性，但必須注意使用不同的宣告後，可以存取的物件成員將不同。簡單的比喻就是「原廠遙控器」vs.「萬用遙控器」，如下圖：



◆ 圖 2-3 多型範例示意圖

假設我們有A電視屬於A品牌，和B電視屬於B品牌，A、B兩台電視都有基本的「開、關、調整音量、切換頻道」功能。而A電視有「特殊音效」的強項，B電視則為「高畫質」。使用「原廠遙控器」的好處，在於遙控器可以百分之百控制對應的電視。

「原廠遙控器」用來對比以「子類別」宣告的物件參考變數。

另外，有一種「萬用遙控器」，萬一原廠遙控器遺失，可以去購買這樣的遙控器；只要做一些設定工作，就可以和大部分的電視連接。有些手機若支援紅外線裝置，通常也有類似功能，所以使用手機的圖樣來表示「萬用遙控器」。這類遙控器可以連接不同的電視，美中不足的地方在於通常不能完全控制該電視，尤其是電視的特殊功能；但基本的「開、關、調整音量、切換頻道」沒有問題。

「萬用遙控器」用來對比以「父類別」宣告的物件參考變數。

所以，雖然父類別物件參考和子類別物件參考都指向同一物件實例，但父類別物件參考只能使用父類別定義的方法，子類別物件參考則使用子類別定義的方法；因為子類別的方法來自父類別加上自己的新增，因此可用的方法比較多，就像「原廠遙控器」能使用的功能多於「萬用遙控器」一樣。將由路徑「/OCP/src/course/c02/polymorphism」下的類別展示多型觀念的使用，步驟如下。

首先建立電視類別 TV。本例設計為抽象類別，因此不適用於直接生產製作，必須有品
牌廠繼承後才能建立實例：

範例

```

1  public abstract class TV {
2      public void turnOn() {
3          System.out.println("turnOn TV");

```

```

4      }
5      public void turnOff() {
6          System.out.println("turnOff TV");
7      }
8      public void changeChannel() {
9          System.out.println("TV changeChannel(), once a channel.");
10     }
11     public void changeVolume() {
12         System.out.println("TV changeVolume()");
13     }
14 }
```

建立 DVDable 介面。一旦 TV 有實作該介面，就具備播放 DVD 的功能：

範例

```

1  public interface DVDable {
2      void playDVD();
3  }
```

建立 SonyTV 類別，繼承 TV 類別：

範例

```

1  public class SonyTV extends TV {
2      public void showSpecialSounds() {
3          System.out.println("SonyTV showSpecialSounds()");
4      }
5  }
```

建立 AppleTV 類別，繼承 TV 類別，並實作 DVDable 介面：

範例

```

1  public class AppleTV extends TV implements DVDable {
2      public void showHD() {
3          System.out.println("AppleTV showHD()");
4      }
5      @Override
6      public void changeChannel() {
7          System.out.println("AppleTV jumps channels.");
8      }
9      @Override
10     public void playDVD() {
11         System.out.println("AppleTV playDVD()");
12     }
13 }
```

接下來，首先測試覆寫 (override) 的影響：

範例

```

1  private static void testOverride() {
2      AppleTV x1 = new AppleTV();
3      TV x2 = x1;
4      DVDable x3 = x1;
5      x1.changeChannel();
6      x2.changeChannel();
7      x1.playDVD();
8      x3.playDVD();
9  }

```

說明

2~4	使用多型的作法，讓同一個 AppleTV 的物件實例，用三種型別宣告。
5,6	分別使用不同型態的物件參考，呼叫相同方法。結果證明，雖然使用不同種類遙控器，但只要遙控器
7,8	上具備此一功能（編譯時期檢查），因為都是指向同一台電視，執行結果都相同（執行時期）。

結果

```

AppleTV jumps channels.
AppleTV jumps channels.
AppleTV playDVD()
AppleTV playDVD()

```

若使用父類別或介面作為宣告型態，子類別或實作類別就可以抽換 / 更換。所以，以下都是多型概念的呈現：

- 以多種宣告型別指向一個物件實例。
- 以一種宣告型別指向多個物件實例。

範例

```

1  private static void showChangeImpl() {
4      TV tv;
5      tv = new AppleTV();
6      tv = new SonyTV();
7      DVDable dvd;
8      dvd = new AppleTV();
9  }

```

在繼續測試之前，分別建立執行 AppleTV 和 SonyTV 功能的方法：

範例

```

1  public static void showAppleTvFuns(AppleTV apple) {
2      System.out.println("---- 顯示所有 AppleTV 功能 -----");
3      apple.turnOn();           // 繼承自父類別
4      apple.turnOff();          // 繼承自父類別
5      apple.changeChannel();    // 繼承自父類別
6      apple.changeVolume();     // 繼承自父類別
7      apple.showHD();           // 自己特化
8      apple.playDVD();          // 外掛 (實作介面)
9  }
10 public static void showSonyTvFuns(SonyTV sony) {
11     System.out.println("---- 顯示所有 SonyTV 功能 -----");
12     sony.turnOn();           // 繼承自父類別
13     sony.turnOff();          // 繼承自父類別
14     sony.changeChannel();    // 繼承自父類別
15     sony.changeVolume();     // 繼承自父類別
16     sony.showSpecialSounds(); // 自己特化
17 }
```

一個衍生的問題是目前只有二家品牌電視繼承 TV 類別，因此我們建立了二個方法。未來若繼承 TV 類別的品牌電視如雨後春筍冒個不停，是不是這類方法也跟著一直新增呢？

使用程式碼測試前述方法：

範例

```

1  private static void withoutPolymorphism() {
2      AppleTV apple = new AppleTV();
3      SonyTV sony = new SonyTV();
4      showAppleTvFuns (apple);
5      showSonyTvFuns (sony);
6  }
```

新增品牌電視繼承 TV 類別，這是商業邏輯的需求無可避免；但其他程式碼，就應該檢討是否可以「以不變，應萬變」。因此方法傳入的參數摒棄特化的子類別，改用父類別或介面：

範例

```

1  public static void showBasicTvFunctions (TV tv) {
2      System.out.println("----- 顯示所有 TV 功能 -----");
3      tv.turnOn();
4      tv.turnOff();
```

```

5      tv.changeChannel();
6      tv.changeVolume();
7  }
8  public static void playDvd (DVDable dvd) {
9      System.out.println("-----顯示 DVDable 功能 -----");
10     dvd.playDVD();
11 }
```

使用程式碼測試前述方法：

範例

```

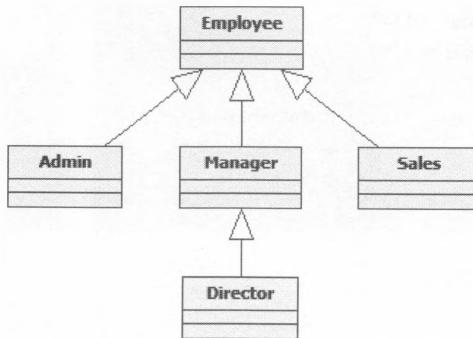
1  private static void withPolymorphism() {
2      AppleTV apple = new AppleTV();
3      SonyTV sony = new SonyTV();
4      showBasicTvFunctions (apple);
5      showBasicTvFunctions (sony);
6      playDvd (apple);
7  }
```

可以發現，如果方法傳入的參數使用父類別或介面宣告，則可以接受所有子類別和實作類別。這樣是不是以逸待勞呢？寫了一個依賴父類別的程式，抵得過千千萬萬個子類別的程式！這就是多型設計強大的地方。

倘若有一天您發現一定得用子類別來寫程式，這時或許就是要重新思考類別繼承架構，或是使用其他設計技巧的時候。「永遠使用父類別、抽象、介面等概念寫程式」，一直都是物件導向程式設計的習慣。

2.4.2 Java 只能單一繼承

Java 不同於 C++ 語言，在繼承上只允許單一繼承，亦即每個類別只能繼承一個父類別，因此必須妥善使用這唯一的機會。除了先前的 Employee 類別和 Manager 類別外，再新增 Amin、Sales、Director 類別，善用一次繼承的機會，架構設計如下：



❖ 圖 2-4 Employee 家族 UML 繼承圖

至於 Java 拒絕多重繼承，主要考量是若多個父類別有相同方法時可以避免子類別繼承的衝突。因為介面在設計時不能提供方法內容，因此沒有衝突的困擾，所以可以實作多個介面。如範例「/OCP/src/course/c02>ShowMultiInterfaces.java」：

④ 範例

```

1  interface Skill_1 {
2      void makeMoney();
3  }
4  interface Skill_2 {
5      void makeMoney();
6  }
7  class Father {
8      void makeMoney() {
9          System.out.println("from Father")
10     }
11 }
12 class Mother {
13     public void makeMoney() {
14         System.out.println("from Father")
15     }
16 }
  
```

以上一共定義了二個類別、二個介面，大家都有相同的方法 `makeMoney()`。但類別的方法有內容，介面的方法則沒有內容。

以下類別分別繼承前述類別，或實作前述介面：

④ 範例

```

1  class Child implements Skill_1, Skill_2 {
2      @Override
  
```

```

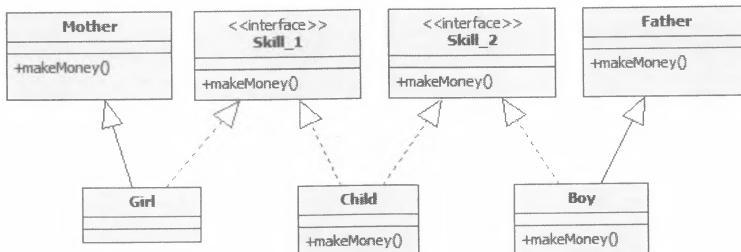
3   public void makeMoney() {
4       System.out.println("make money by java and database");
5   }
6 }
7 class Boy extends Father implements Skill_2 {
8     @Override
9     public void makeMoney() {
10        super.makeMoney();
11    }
12 }
13 class Girl extends Mother implements Skill_1 {
14     // 父類別的方法必須是public
15 }
```

Java 不允許多重繼承的理由在這裡就看得很清楚了。假設 Child 類別同時繼承了 Father 類別和 Mother 類別，請問他該得到誰的 makeMoney() 方法呢？因為有衝突，因此多重繼承無法通過編譯。

但實作多個介面就不同。因為 Skill_1 和 Skill_2 介面雖然定義了相同方法 makeMoney()，但本身沒有提供內容，而且都是要求子類別必須實作方法內容。雖然二個介面各自要求了一次，但實際上 Child 類別只要實作一次就可以應付，因此沒有多重繼承的衝突問題。

至於 Boy 類別和 Girl 類別都由父類別繼承了 makeMoney() 方法，也都需要實作 Skill_1 和 Skill_2 介面要求的 makeMoney() 方法，但 Girl 類別可以直接拿 Mother 類別的 makeMoney() 方法回應介面實作的要求，但 Boy 類別繼承的 Father 類別卻不行。何故？

若要拿父類別既有的方法回應介面要求的實作方法，前提是父類別的方法必須是 public。因為介面的方法預設是 public，覆寫時也必須宣告為 public，即便來自父類別。



❖ 圖 2-5 單一繼承與多實作範例的 UML 類別圖

2.5 認證考試命題範圍

依據甲骨文公布的考試範圍 (https://education.oracle.com/java-se-8-programmer-ii/pexam_1Z0-809)，和本章相關的主題有：

Java Class Design

1. Create and use **singleton** classes and **immutable** classes.

本章擬真試題實戰

考題 1

Given:

```
interface HorseLike {
    String ride();
}

class Horse implements HorseLike {
    String ride() {
        return "cantering";
    }
}

class Colt extends Horse {
    String ride() {
        return "tolting";
    }
}
```

And:

```
public static void main(String[] args) {
    HorseLike r1 = new Colt();
    HorseLike r2 = new Horse();
    Horse h1 = new Colt();
}
```

What is the result?

- A. tolting cantering tolting
- B. cantering cantering cantering
- C. compilation fails
- D. an exception is thrown at runtime

答案 C

說明 interface 的方法，預設是 public 且 abstract。覆寫的子類別必須是 public。

考題 2

Given:

```
class Word {  
    private Word(int length) {  
    }  
    protected Word(String w) {  
    }  
}  
class SubWord extends Word {  
    public SubWord() {  
        // insert code here  
    }  
    public SubWord(String w) {  
        super(w);  
    }  
}
```

Which two code snippets, added independently at // insert code here, can make the SubWord class compile?

- A. `this();`
- B. `this(100);`
- C. `this("SubWord ");`
- D. `super();`
- E. `super(100);`
- F. `super("SubWord ");`

答案 CF

說明 參照本章「2.2.2 建立子類別的建構子」，或本書前冊《Java SE8 OCAJP 專業認證指南》。

考題 3

Given:

```
interface Glommer {}  
interface Plinkable {}  
class Flimmer implements Plinkable {
```

```

        List<Target> t = new ArrayList<>();
    }
    class Flommer extends Flimmer {
        String s = "Test";
    }
    class Target implements Glommer {
        void doStuff() {
            String s = "OCP Test";
        }
    }
}

```

Which two statements concerning the OO concepts "IS-A" and "HAS-A" are true?

- A. Flimmer is-a Glommer.
- B. Flommer has-a String.
- C. Target has-a Glommer.
- D. Flimmer is-a ArrayList.
- E. Target has-a doStuff()
- F. Target is-a Glommer.

答案 BF

考題 4

Given:

```

interface Sports {
    String getCatalog();
}

class CueSports {
    public String getCatalog() {
        return "Cue Sports";
    }
}

class MySports extends CueSports implements Sports { // line1
    public static void main(String[] args) {
        Sports obj1 = new MySports(); // line2
        CueSports obj2 = new MySports(); // line3
        System.out.print(obj1.getCatalog() + "," + obj2.getCatalog()); // line4
    }
}

```

What is the result?

- A. Cue Sports,Cue Sports
- B. Compilation fails at //line1
- C. Compilation fails at //line2
- D. Compilation fails at //line3
- E. Compilation fails at //line4

答案 A

說明 參考本章「2.4.2 Java 只能單一繼承」觀念。MySports 類別 implements Sports 後，原本該提供的方法實作 getCatalog()，可以由繼承 CueSports 類別取得。因此程式可以正常編譯。

考題 5

Given:

```
interface Vehicle {  
    void start();  
    void stop();  
}  
  
interface Motorized {  
    void stop();  
    void speed();  
}  
  
class MyCar implements Vehicle, Motorized {  
    public void start() { };  
    public void stop() { };  
    public void speed() { };  
}
```

What is the result of invoking MyCar's stop method?

- A. Both Vehicle's and Motorized's stop methods are invoked.
- B. Vehicle's stop method is invoked.
- C. Motorized's stop method is invoked
- D. The implementation of the MyCar's stop determines the behavior.
- E. Compilation fails.

答案 D

考題 6

Given:

```
interface Runnable {
    public String ride() {return "riding";}
}

class Horse implements Runnable {
    public String ride() {
        return "cantering";
    }
}

class Icelandic extends Horse implements Runnable {
    public String ride() {
        return "tolting";
    }
}
```

And:

```
public static void main(String[] args) {
    Runnable r1 = new Icelandic();
    Runnable r2 = new Horse();
    Horse h1 = new Icelandic();
    System.out.println(r1.ride() + r2.ride() + h1.ride());
}
```

What is the result?

- A. riding riding tolting
- B. riding riding cantering
- C. tolting cantering tolting
- D. tolting cantering cantering
- E. Compilation fails.
- F. An exception is thrown at runtime.

答案 E

說明 Interface 的方法不能有內容。

類別設計

-
- | 3.1 存取控制
 - | 3.2 覆寫方法
 - | 3.3 轉型
 - | 3.4 覆寫Object 類別的方法
 - | 3.5 認證考試命題範圍

3.1 存取控制

3.1.1 存取控制層級

Java 成員的「存取控制層級 (access level)」一共分成四種。除了先前封裝時介紹的「public」、「private」，還有「protected」和「default/package」兩種層級。作用範圍分別是：

- **protected** 層級：嚴格程度高於 public。除了同一類別內、同一套件下，具有繼承關係的子類別也可以存取父類別宣告為 protected 的欄位或方法。
- **default** 層級：若欄位或方法建立時未使用存取控制層級的修飾詞（亦即未載明使用 public、protected 或 private），Java 將採取預設 (default) 層級的存取控制。因為 Java 重視安全性，該層級僅次於 private。此時只有在同一套件 (package) 下的類別能夠存取該類別的欄位或方法，故又名「package」層級。

四種層級的彙整如下。其中 Y 表示欄位或方法可被存取：

◆ 表 3-1 各存取控制層級比較表

	同一 class	同一 package	不同 package 但為子類別	不設限
private	Y			
default	Y	Y		
protected	Y	Y	Y	
public	Y	Y	Y	Y

以下範例顯示存取控制層級的用法：

④ 範例

```

1 package demo;
2 public class Foo {
3     protected int result = 20;
4     int other = 25;      //default, 不能跨 package
5 }
```

和其子類別：

④ 範例

```

1 package test;
2 import demo.Foo;
```

```

3  public class Bar extends Foo {
4      private int sum = 10;
5      public void reportSum () {
6          sum += result;        //OK
7          sum += other;        //NG
8      }
9  }

```

Bar 和 Foo 兩個類別位於不同 package，所以 Foo 類別的欄位 other，存取層級為預設 (default\package)，不能被不同 package 的 Bar 類別存取。

因為 Bar 類別繼承了 Foo 類別，且 Foo 類別的 result 欄位其存取層級為 protected，所以可以被子類別 Bar 存取，不受 package 限制。

3.1.2 欄位遮蔽效應

設計類別成員的存取控制層級時候，容易因為相同的欄位 (field) 名稱出現在父類別和子類別，卻沒有使用 private 告知有效區隔兩者，導致「欄位遮蔽 (field shadowing)」效應而造成程式邏輯混淆。如範例「/OCP/src/course/c03/FieldShadowDemo1.java」：

範例

```

1  class Source1 {
2      protected int result = 20;
3  }
4  class Test1 extends Source1 {
5      protected int result = 30;      // 子類別欄位將遮蔽父類別欄位
6      public int reportSum() {
7          return 10 + result;
8      }
9  }
10 public class FieldShadowDemo1 {
11     public static void main(String args[]) {
12         System.out.println(new Test1().reportSum());
13     }
14 }

```

結果

40

說明

5	子類別欄位已有欄位 result 。 父類別也定義欄位 result ，加上使用 protected 壓告，子類別也可以存取。
	二個欄位子類別都可以存取。但子類別自己定義的欄位效力高於父類別，因而遮蔽父類別同名欄位，稱為「欄位遮蔽」。
7	因為欄位遮蔽效應，這裡的 result 欄位為子類別欄位，值為 30。

設計類別欄位時，應養成好習慣，避免「欄位遮蔽」讓程式邏輯混淆：

- 父類別欄位改為 **private**，並建立 **getter()** 方法供子類別呼叫。
- 因為子類別看不見父類別欄位，就不會有遮蔽的問題。

修改後如範例「/OCP/src/course/c03/FieldShadowDemo2.java」：

範例

```

1  class Source2 {
2      private int result = 20;
3      protected int getResult() {
4          return result;
5      }
6  }
7  class Test2 extends Source2 {
8      public int reportSum() {
9          return 10 + getResult();
10     }
11 }
12 public class FieldShadowDemo2 {
13     public static void main(String args[]) {
14         System.out.println(new Test2().reportSum());
15     }
16 }
```

結果

30

3.2 覆寫方法

3.2.1 覆寫方法的規則

當子類別和父類別同時具備簽名(signature)相同的方法時，執行時子類別方法的效力將高於父類別，稱為子類別方法「覆寫 override」父類別方法。此處的「方法簽名(method signature)」指方法名稱和方法參數。

換個角度看，類別的方法是以簽名作為識別。當子類別繼承父類別後，在存取控制層級允許的情況下，將具備使用父類別方法的能力。若父子類別存在相同簽名的名稱，將會造成衝突。因此子類別方法可以「覆寫 override」父類別方法，避免衝突。

首先建立範例「/OCP/src/course/c03/Employee.java」：

範例

```

1  public class Employee {
2      private int empId;
3      private String name;
4      private String ssn;
5      private double salary;
6      public Employee(int empId, String name, String ssn, double salary) {
7          this.empId = empId;
8          this.name = name;
9          this.ssn = ssn;
10         this.salary = salary;
11     }
12     public String profile () {
13         return "id:" + empId + " ,name:" + name;
14     }
15 }
```

建立前述Employee類別的子類別Manager，如範例「/OCP/src/course/c03/Manager.java」：

範例

```

1  public class Manager extends Employee {
2      private String deptName;
3      public Manager(int empId, String name, String ssn, double salary,
4                      String deptName) {
5          super(empId, name, ssn, salary);
6      }
7 }
```

```

5         this.deptName = deptName;
6     }
7     @Override
8     public String profile() {
9         return super.profile() + " ,Department:" + this.deptName;
10    }
11 }
```

其中，方法 profile() 因為父子類別都有，因此發生子類別覆寫父類別的狀況。對兩個類別做測試：

Q 範例

```

1 public static void main(String args[]) {
2     Employee e = new Employee (1, "Jim", "11111", 100_000.00);
3     System.out.println (e.profile());
4     Manager m = new Manager (2, "Tom", "22222", 200_000.00, "Marketing");
5     System.out.println (m.profile());
6 }
```

可以得到結果：

Q 結果

```

id:1 ,name:Jim
id:2 ,name:Tom ,Department:Marketing
```

雖然父類別 Employee 也有 profile() 方法，呼叫 Manager 時，還是會使用子類別覆寫後的方法。

前例中宣告和實際建立的物件其型別都一致。但若不一致時會如何呢？如：

Q 範例

```

1 public static void main(String args[]) {
2     Employee em = new Manager (2, "Tom", "22222", 200_000.00, "Marketing");
3     System.out.println (em.profile());
4 }
```

Q 結果

```

id:2 ,name:Tom ,Department:Marketing
```

由編譯到執行，過程是：

1. 編譯器檢查 Employee 型別是否具備 profile() 方法。
2. 執行時由 Manager 物件執行 profile() 方法。

處理這類問題時，必須分清楚「編譯時期」和「執行時期」兩段行為的不同。物件在「執行時期」表現出來的行為，就是在 heap 裡的實際的物件表現出來的行為。

在 C++ 裡，若要使用這種機制，必須在方法前加上「virtual」的宣告詞，故又稱為「virtual method invocation」，簡稱「VMI」。

因為一開始宣告為父類別，執行時的行為則交由實際物件「動態決定」，故又稱為「延遲繫結 (late binding)」或「動態繫結 (dynamic binding)」。

OCA 及 OCP 考試很喜歡測試考生對於「覆寫 (override)」的熟練度。如範例「/OCA/src/course/c11/TestOverride.java」，請問哪些類別無法通過編譯？

範例

```

1  class Deeper throws IOException {
2      public Number getDepth(Number n) {
3          return 10;
4      }
5  }
6  class DeepA extends Deeper {
7      @Override
8      protected Integer getDepth(Number n) {
9          return 5;
10     }
11 }
12 class DeepB extends Deeper {
13     @Override
14     public Double getDepth(Number n) {
15         return 5d;
16     }
17 }
18 class DeepC extends Deeper {
19     @Override
20     public String getDepth(Number n) {
21         return "";
22     }
23 }
24 class DeepD extends Deeper {
25     @Override

```

```
26     public Long getDepth(int d) {
27         return 5L;
28     }
29 }
30 class DeepE extends Deeper {
31     @Override
32     public Short getDepth(Integer n) {
33         return 5;
34     }
35 }
36 class DeepF extends Deeper {
37     @Override
38     public Object getDepth(Object n) {
39         return 5;
40     }
41 }
42 class DeepG extends Deeper {
43     @Override
44     public Number getDepth(Number n) throws Exception {
45         return 5;
46     }
47 }
48 class DeepH extends Deeper {
49     @Override
50     public Number getDepth(Number n) throws FileNotFoundException {
51         return 5;
52     }
53 }
54 class DeepI extends Deeper {
55     @Override
56     public Number getDepth(Number n)
57             throws IOException, FileNotFoundException {
58         return 5;
59     }
60 }
61 class DeepJ extends Deeper {
62     @Override
63     public Number getDepth(Number n) throws IOException, SQLException {
64         return 5;
65     }
66 }
```

要解答這類問題，必須掌握四個原則：

1. 覆寫的前提是父子類別的方法的「簽名(名稱 + 參數)完全相同」。
2. 存取層級(access modifier)必須相同或更高。
3. 回傳型態(return type)必須相同或是子類別。
4. 括出的例外類別(exception type)必須相同，或是子類別。而且數量可以更少。



課堂小秘訣

1. 關於存取層級，若是繼承後每次覆寫的存取層級都變小，由 public 被覆寫為 protected，再被覆寫為 default，再被覆寫為 private，是不是繼承 3 代後就不能再繼續？等於是絕後了！
2. 關於回傳型態和拋出的例外類別，因為繼承目的是為了更精進，所以覆寫之後可以允許回傳型態是子類別，拋出的例外也可以是子類別。
3. 關於拋出的例外類別，若原先拋出 2 個例外，覆寫後例外都被處理了，不需丟出例外，是不是也是一種精進呢？

所以，上述範例編譯結果整理如下：

◆表 3-2 範例 TestOverride 結果列表

類別	結果	原因
DeepA	NG	覆寫後存取層級變小。
DeepB	OK	可以不拋出任何例外。
DeepC	NG	覆寫後回傳型態不對。
DeepD	NG	方法簽名未完全相同，所以不是覆寫，和 @Override 的註釋不符。
DeepE	NG	方法簽名未完全相同，所以不是覆寫，和 @Override 的註釋不符。
DeepF	NG	方法簽名未完全相同，所以不是覆寫，和 @Override 的註釋不符。
DeepG	NG	拋出的例外類別 FileNotFoundException 非 IOException 一族(子類別)。
DeepH	OK	FileNotFoundException 為 IOException 一族，為其子類別。
DeepI	OK	FileNotFoundException 為 IOException 一族，為其子類別。
DeepJ	NG	SQLException 非 IOException 一族。

3.2.2 只有物件成員的方法可以覆寫

必須注意的是，Java 裡只有「物件成員」的「方法」可以覆寫。欄位無法覆寫，靜態成員也無法覆寫。亦即：

❖ 表 3-3 各類成員覆寫機制比較表

分類 / 是否提供覆寫	物件成員	靜態成員
欄位	否	否
方法	是	否

範例「/OCP/src/course/c03/TestOverride1.java」測試當「宣告型別(reference type)」和「物件型別(object type)」不同時，使用「物件方法」和使用「物件欄位」的差異：

🔍 範例

```

1  class Super1 {
2      protected int num = 20;
3      public int reportSum() {
4          return num;
5      }
6  }
7  class Sub1 extends Super1 {
8      protected int num = 30;
9      public int reportSum() {
10         return 10 + num;
11     }
12 }
13 public class TestOverride1 {
14     public static void main(String args[]) {
15         Super1 s = new Sub1();
16         System.out.println(s.reportSum());
17         System.out.println(s.num);
18     }
19 }
```

⌚ 結果

```

40
20
```

🔊 說明

10	因為欄位遮蔽的效應，此處 num=30。
15	宣告型別和實際物件型別不同時。
16	呼叫物件方法時，因為有覆寫功能，執行時以「物件型別」的方法為主。
17	呼叫物件欄位時，無覆寫功能，執行時以「宣告型別」的欄位為主。

範例「/OCP/src/course/c03/TestOverride2.java」顯示呼叫「類別成員」和「物件成員」範例的差異。本例中所有成員均宣告為 static：

範例

```

1  class Super2 {
2      protected static int num = 20;
3      public static int reportSum() {
4          return num;
5      }
6  }
7  class Sub2 extends Super2 {
8      protected static int num = 30;
9      public static int reportSum() {
10         return 10 + num;
11     }
12 }
13 public class TestOverride2 {
14     public static void main(String args[]) {
15         Super2 s = new Sub2();
16         System.out.println( s.reportSum() );
17         System.out.println( s.num );
18         System.out.println( Super2.reportSum() ); // 應該這樣呼叫，就不會混淆
19         System.out.println( Super2.num );           // 應該這樣呼叫，就不會混淆
20     }
21 }
```

結果

```

20
20
20
20
```

說明

15	宣告型別和物件型別不同。
16	因為是 static 方法，無覆寫功能，執行時以「宣告型別」的方法為主。
17	因為是 static 欄位，無覆寫功能，執行時以「宣告型別」的欄位為主。
18	在《Java SE8 OCAJP 專業認證指南》一書講解 static 方法時曾經說明，使用類別 / 靜態成員時，應該使用類別名稱呼叫其方法和屬性，雖然和使用物件參考呼叫結果相同，但使用類別名稱呼叫，就可以「避免宣告型別和物件型別不同時的困擾」，本例是很好的驗證。
19	

3.2.3 善用多型 (Polymorphism)

使用多型可以讓程式碼更具彈性。以範例「/OCP/src/course/c03/polymorphism/Before.java」而言，目前架構是若公司增加一種新的職位，如 Engineer，除了增加必需的 Engineer 類別外，EmployeeStockPlan 類別也必須增加一個對應的方法 grantStock(Engineer a)，以處理員工的配股數量：

範例

```

1  class Employee {}
2  class Manager extends Employee {}
3  class Director extends Manager {}
4  class EmployeeStockPlan {
5      public int grantStock(Manager m) {
6          return 30;
7      }
8      public int grantStock(Director a) {
9          return 40;
10     }
11     // ... 針對新增員工型態必須增加對應方法
12 }
13 public class Before {
14     public static void main(String[] args) {
15         EmployeeStockPlan plan = new EmployeeStockPlan();
16         Manager m = new Manager();
17         System.out.println(plan.grantStock(m));
18     }
19 }
```

範例「/OCP/src/course/c03/polymorphism/After.java」導入多型設計改善程式架構。此時 EmployeeStockPlan 類別的方法只剩下一個，參數改為輸入父類別 Employee，因此可以將處理的職位對象放至最寬，不再因為新增職位而需要新增方法。但是方法內容改成轉呼叫員工的 calculateStock() 方法，因此每個職位類別必須處理自己的配股數量：

1. 改版後 Employee 類別變為 abstract，同時建立 abstract 方法 calculateStock()，因此子類別必須實作該方法。
2. 改版後 EmployeeStockPlan 類別只餘下一個 grantStock() 方法，可接受所有繼承 Employee 後的新職位類別。方法內容只是轉呼叫傳入職位類別的 calculateStock() 方法。如下：

範例

```

1 abstract class Employee {
2     abstract protected int calculateStock();
3 }
4 class Manager extends Employee {
5     protected int calculateStock() {
6         return 30;
7     }
8 }
9 class Director extends Manager {
10    protected int calculateStock() {
11        return 40;
12    }
13 }
14 class EmployeeStockPlan {
15     public int grantStock(Employee e) {
16         return e.calculateStock();
17     }
18 }
19 public class After {
20     public static void main(String[] args) {
21         EmployeeStockPlan plan = new EmployeeStockPlan();
22         Manager m = new Manager();
23         System.out.println(plan.grantStock(m));
24     }
25 }
```

比較前後兩個範例，可以發現使用改版後的多型設計架構，當公司有新增職位時（增加新類別是無法避免的事情），因為把計算配股數量的邏輯也放在職位類別裡，因此不再需要修改 EmployeeStockPlan 類別以增加新方法，避免了既有程式的異動，也減少了 bug 產生的機會。這就實踐了我們在「物件導向程式設計(Object Oriented Programming，簡稱 OOP)」裡的「開閉法則(Open Closed Principle，簡稱 OCP)」：「Open for extension, closed for modification」，中文解釋為「對程式擴充開放(歡迎)，但對程式修改關閉(拒絕)」。

3.2.4 instanceof 運算子

使用 instanceof 運算子可以確認物件的型態，如以下範例。傳入的物件都符合 Employee 的子類別，但若需要確認是否是 Manager 類別的實例，可以使用 instanceof 運算子：

範例

```

1 public boolean isManager (Employee e) {
2     if (e instanceof Manager) {
3         return true;
4     }
5     return false;
6 }
```

又多型的優勢在於讓傳入的型別有較大空間；但有些時候必須轉型 (casting) 回物件真正的型態才能使用特化的子類別方法。此時使用 instanceof 運算子在轉型前先判斷，可避免 java.lang.ClassCastException 發生，如下例：

範例

```

1 public void modifyManagedDept (Employee e, String dept) {
2     if (e instanceof Manager) {
3         Manager m = (Manager) e;      // 使用轉型 (casting)
4         m.setDeptName (dept);
5     }
6 }
```

3.3 轉型

參考型別的轉型有以下三種可能：

- 「向上轉型」：目標是父類別，將讓可以使用的物件成員變少。因為沒有風險，所以在運算式不對等的情況下會自動發動。
- 「向下轉型」：目標是子類別，將讓可以使用的物件成員變多。因為有風險，必須自己發動。
- 「平行轉型」：目標是自己，沒有風險也沒有特殊意義，必須自己發動。

轉型的成功與否，則分成：

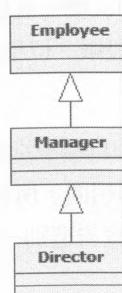
- 編譯 (compile) 時期：關鍵在於變數的「宣告型別」，可以是該型別的「向上轉型」、「平行轉型」或「向下轉型」。
- 執行 (run) 時期：關鍵在於變數的「物件型別」，只能是該型別的「向上轉型」、「平行轉型」，不可以是「向下轉型」。

其中，既然執行時期不允許「向下轉型」，為何編譯時期會放行？原因在於物件導向程式語言允許「多型宣告」，因此宣告型別可能大於實例型別，所以在編譯時期看似「向下轉型」，在執行時期可能只是「平行轉型」。因此編譯器必須可以接受「向下轉型」，如範例「/OCP/src/course/c03/casting/CastTest.java」：

範例

```

1  class Employee {
2  }
3  class Manager extends Employee {
4  }
5  class Director extends Manager {
6  }
7  interface Quit {
8  }
9  public class CastTest {
10     public static void main(String[] args) {
11         Manager m = new Manager();
12         testCastClass (m);
13         testCastInterface(m);
14     }
15     private static void testCastClass (Manager m) {
16
17         Employee e = (Employee) m;      // 向上轉型
18
19         Manager mg = (Manager) m;      // 平行轉型
20
21         Director d = (Director) m;    // 向下轉型
22     }
23     private static void testCastInterface (Manager m) {
24         Quit q = (Quit) m;           // 轉型成不相關的介面
25     }
26 }
```



◆ 圖 3-1 Employee 家族 UML 類別圖

本範例可以通過編譯，因為編譯時期可以接受向上、平行、向下轉型。但在執行時期，因為：

範例

```
11     Manager m = new Manager();
```

則程式碼在執行至行 21 時：

範例

```
21     Director d = (Director) m;
```

因為執行時期不允許向下轉型，將會拋出 ClassCastException 的例外。但若改為：

範例

```
11     Manager mg = new Director();
```

則程式碼在執行至行 21 時：

範例

```
21     Director d = (Director) m;
```

不會出錯，因為執行時期可以平行轉型。整理對應關係如下表：

表 3-4 編譯 / 執行時期轉型關係對照表

程式碼	編譯時期 (以宣告型態為主)	執行時期 (以物件實例型態為主)	
	Manager m;	m = new Manager();	m = new Director();
Employee e =(Employee) m;	向上轉型	向上轉型 (OK)	向上轉型 (OK)
Manager mg =(Manager) m;	平行轉型	平行轉型 (OK)	向上轉型 (OK)
Director d =(Director) m;	向下轉型	向下轉型 (NG)	平行轉型 (OK)

就因為執行時期可能因為物件實例的改變而導致正確或出錯，所以編譯器採取開放的態度，無論向上、平行、向下轉型都不設限。但程式設計師必須知道若執行時期發生「向下轉型」將導致錯誤。

此外，行 21 在編譯時期只檢查轉型的目標型別是否屬於宣告型別的子類別，執行時期就必須確保轉型目標和實際的物件實例相同，所以這類向下轉型，說穿了就是一種還原轉型，還原物件原貌，讓物件實例能和宣告型別一致。

最後，Employee 一族的類別均未實作介面 Quit，為何行 24 可以通過編譯？但執行時期卻出錯？如同先前的說明，編譯器必須預知執行時期可能的狀況並予以放行；對編譯器來說，因為 Java 只允許單一繼承，但可以實作其他 interface，所以「Employee 的子類別們」不可能再去繼承其他 class，卻有可能「實作其他 interface，如 Quit」，因此在編譯時期姑且放行。而在執行時期因為可以確定子類別身分，就是見真章的時候了；若該類別未實作轉型的目標的 interface，還是會失敗。

3.4 覆寫Object類別的方法

類別「java.lang.Object」是 Java 裡的第一號人物，也是所有物件的始祖。類別若未繼承其他類別，將於類別定義自動加上「extends Object」。如：

範例

```
1  class Employee {  
2  }
```

其實編譯後是：

範例

```
1  class Employee extends Object {  
2  }
```

Object 類別裡有許多「non-final」的方法可以覆寫，如：

- `toString()`
- `equals()`
- `hashCode()`
- `clone()`
- `finalize()`

比較需要覆寫的方法是：

- `toString()`
- `equals()`
- `hashCode()`

覆寫 `toString()` 方法

Object 類別的 `toString()` 方法提供對物件簡單描述：

範例

```
1 public String toString() {
2     return getClass().getName() + "@" + Integer.toHexString(hashCode());
3 }
```

通常類別都會改寫此一方法，已提供客製化的個別類別訊息。如 Employee 類別：

範例

```
1 public String toString () {
2     return "Employee id: " + empId + "\n" "Employee name:" + name;
3 }
```

`System.out.println()` 方法若傳入物件參考，就會呼叫該物件的 `toString()` 方法。

覆寫 `equals()` 方法

Object 類別的 `equals()` 方法預設使用「`==`」運算子比較物件在記憶體中的位置是否相同：

範例

```
1 public boolean equals(Object obj) {
2     return (this == obj);
3 }
```

然而我們在意的通常是物件的「內容」或「特徵」是否相同，因此必須覆寫原始的 `equals()` 方法。以 Employee 類別為例，若我們認定員工編號 `empId` 欄位可以代表一個員工，就可以將 `equals()` 方法覆寫為：

範例

```
1 public boolean equals(Object obj) {
2     if (this == obj)
3         return true;
4     if (obj == null)
5         return false;
6     if (getClass() != obj.getClass())
7         return false;
8     Employee other = (Employee) obj;
9     if (empId != other.empId)
```

```

10     return false;
11     return true;
12 }

```

比較 String 物件時，因為 equals() 方法已被覆寫，只看字串內容(組成字元)是否一致。

覆寫 hashCode() 方法

依據 Java API 文件對於 Object 類別的 hashCode() 方法描述：

- 在同一個應用程式執行期間，對同一物件呼叫 hashCode() 方法，必須回傳相同結果。
- 如果兩個物件使用 equals() 方法測試結果為「相等」，則這兩個物件使用 hashCode() 方法時，也「必須」獲得相同的結果。
- 如果兩個物件使用 equals() 方法測試結果為「不相等」，則這兩個物件使用 hashCode() 方法時，「不一定」要獲得不同的結果。

因此，一旦覆寫 equals() 方法，就應該同時覆寫 hashCode() 方法，使其結果一致。以 Employee 類別為例，因為我們使用員工編號 empId 作為 equals() 方法比較的關鍵，所以應該也將 empId 以一樣的設計理念融入於覆寫 hashCode() 方法中：

範例

```

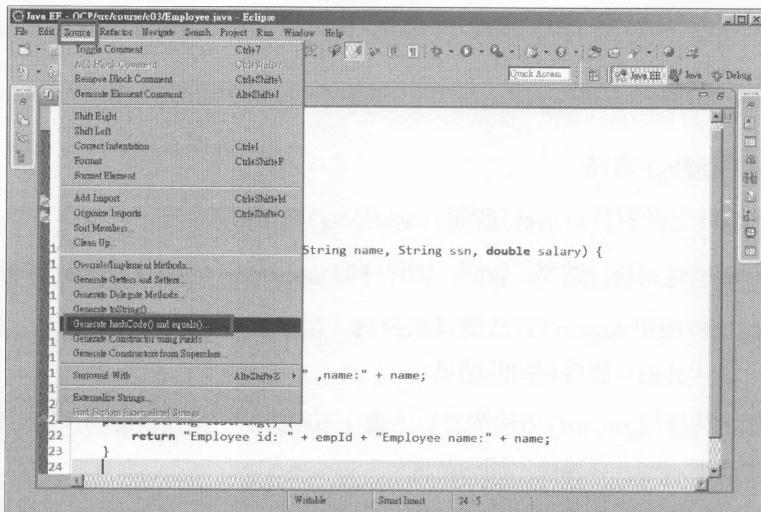
1 public int hashCode() {
2     final int prime = 31;
3     int result = 1;
4     result = prime * result + empId;
5     return result;
6 }

```

一些和雜湊函數(hash function)有關的集合物件，如 HashMap、HashSet 和 Hashtable 等，hashCode() 和 equals() 兩個方法會合併用來判斷集合裡的物件是否相同，因此建議一併覆寫。

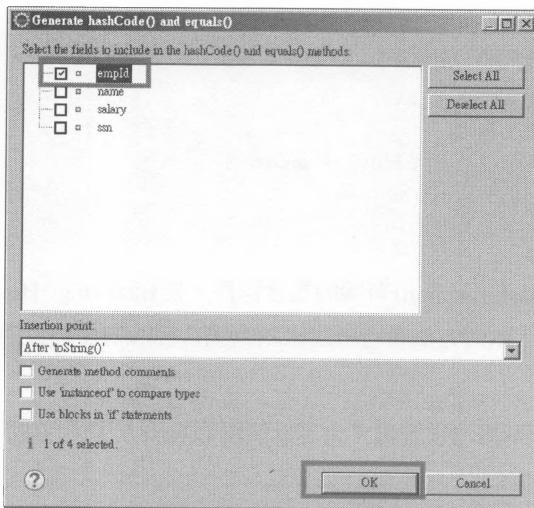
如果覺得覆寫這兩個方法很麻煩，也可以使用 Eclipse 的功能協助：

STEP01 選擇下拉選單的「Source」，再選擇「Generate hashCode() and equals()...」。



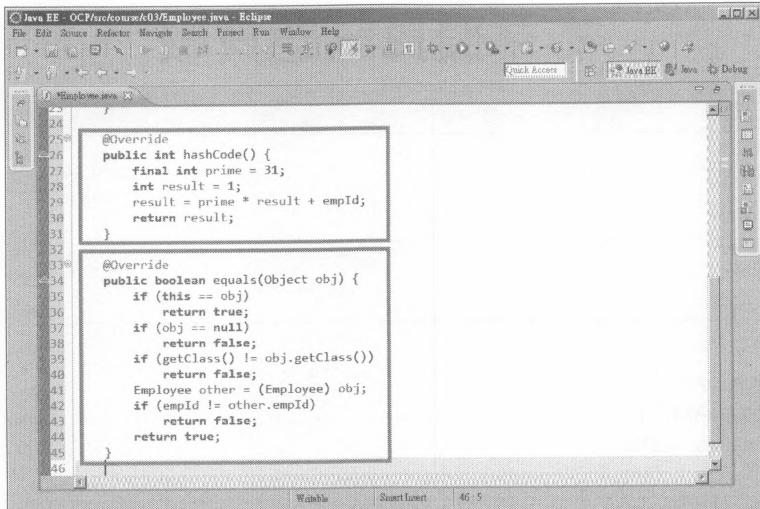
❖ 圖 3-2 進入功能表單

STEP02 決定要以類別裡的那些欄位作為覆寫方法時的使用要件。



❖ 圖 3-3 選擇欄位

STEP03 產生完畢。



```

25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46

```

```

@Override
public int hashCode() {
    final int prime = 31;
    int result = 1;
    result = prime * result + empId;
    return result;
}

@Override
public boolean equals(Object obj) {
    if (this == obj)
        return true;
    if (obj == null)
        return false;
    if (getClass() != obj.getClass())
        return false;
    Employee other = (Employee) obj;
    if (empId != other.empId)
        return false;
    return true;
}

```

◆ 圖 3-4 自動產出程式碼

3.5 認證考試命題範圍

依據甲骨文公布的考試範圍 (https://education.oracle.com/java-se-8-programmer-ii/pexam_1Z0-809)，和本章相關的主題有：

Java Class Design

1. Implement **encapsulation**.
2. Implement **inheritance** including **visibility** modifiers and **composition**.
3. Implement **polymorphism**.
4. Override **hashCode**, **equals**, and **toString** methods from **Object** class.

本章擬真試題實戰

考題 1

Given:

```
class A {  
    int a = 5;  
    String doA1() {  
        return "a1";  
    }  
    static String doA2() {  
        return "a2";  
    }  
}  
  
class B extends A {  
    int a = 7;  
    String doA1() {  
        return "b1";  
    }  
    public static String doA2() {  
        return "b2";  
    }  
    void go() {  
        A a = new B();  
        System.out.print(a.doA1() + "\n" + a.doA2() + "\n" + a.a);  
    }  
}
```

And:

```
public static void main(String[] args) {  
    new B().go();  
}
```

Which three values will appear in the output?

A. 5

B. 7

C. a1

D. a2

E. b1

F. b2

答案 ADE**說明** 參閱本章「3.2.2 只有物件成員的方法可以覆寫」。

考題 2

Given:

```
class Test {
    private int id;
    public Test(int id) {
        this.id = id;
    }
    public int hashCode() {
        return id + 42;
    }
    public boolean equals(Object obj) {
        return (this == obj) ? true : super.equals(obj);
    }
}
```

And:

```
public static void main(String[] args) {
    Test t1 = new Test(10);
    Test t2 = new Test(10);
    Test t3 = new Test(20);
    System.out.print(t1.equals(t2) + " ");
    System.out.print(t1.equals(t3));
}
```

What is the result?

A. false false

B. true false

C. true true

D. Compilation fails

E. An exception is thrown at runtime

答案 A

說明 覆寫後的 equals() 方法是以物件參考是否指向相同記憶體位址來判定。因此 t1、t2、t3 都不相等。

考題 3

Given:

```
class Plant {  
    abstract String grow();  
}  
class Vegetable extends Plant {  
    String grow() {  
        return "Up";  
    }  
}  
class Carrot extends Vegetable {  
    String grow() {  
        return "Down";  
    }  
}
```

And:

```
public static void main(String[] args) {  
    Vegetable e = new Vegetable();  
    Vegetable c = new Carrot();  
    System.out.print(e.grow() + c.grow());  
}
```

What is the result?

- A. Up Down
- B. Up Up
- C. Up null
- D. Compilation fails
- E. An exception is thrown at runtime

答案 D

說明 類別 Plant 編譯失敗。因為具備 abstract 方法，類別也必須是 abstract。

考題 4

Given:

```
class Student {
    String name = "Unknown";
    public String getName() {
        return name;
    }
}
class Jim extends Student {
    String name = "Jim";
}

class Harry extends Student {
    String name = "Harry";
    public String getName() {
        return name;
    }
}
```

And:

```
public static void main(String[] args) {
    Student s1 = new Jim();
    Student s2 = new Harry();
    System.out.print(s1.getName() + " ");
    System.out.print(s2.getName());
}
```

What is the result?

- A. Jim Harry
- B. Unknown Harry
- C. Jim Unknown
- D. Unknown Unknown
- E. Compilation fails.
- F. An exception is thrown at runtime.

答案 B

考題 5

Given:

```
class Dog {  
    protected String bark() {  
        return "oooo";  
    }  
}  
  
class Poodle extends Dog {  
    private String bark() {  
        return "zzzz";  
    }  
}
```

And:

```
public static void main(String[] args) {  
    Dog[] dogs = { new Dog(), new Poodle() };  
    for (Dog d : dogs) {  
        System.out.print(d.bark());  
    }  
}
```

What is the result?

- A. oooo zzzz
- B. oooo oooo
- C. zzzz zzzz
- D. A RuntimeException is generated
- E. The code fails to compile

答案 E

說明 方法 String bark() 覆寫後，必須是 protected 或 public。

考題 6

Given:

```
1. abstract class Shape {  
2.     Shape() { }  
3.     protected void area() { System.out.println("Shape"); }  
4. }  
5. class Square extends Shape {  
6.     int a;  
7.     Square(int a) {  
8.         /* insert code here */  
9.         this.a = a;  
10.    }  
11.    public void area() { System.out.println("Square"); }  
12. }  
13. class Rectangle extends Square {  
14.     int b, c;  
15.     Rectangle(int b, int c) {  
16.         /* insert code here */  
17.         this.b = b; this.c = c;  
18.     }  
19.     void area() { System.out.println("Rectangle"); }  
20. }
```

Which two modifications enable the code to compile?

- A. At line 1, remove abstract
- B. At line 8, insert super();
- C. At line 11, remove public
- D. At line 16, insert super(b);
- E. At line 16, insert super(); super.a = b;
- F. At line 19, use public void area() {

答案 DF

進階類別設計

-
- | 4.1 使用抽象類別
 - | 4.2 使用static 關鍵字
 - | 4.3 使用final 關鍵字
 - | 4.4 實作獨體設計模式
 - | 4.5 列舉型別
 - | 4.6 使用巢狀類別
 - | 4.7 認證考試命題範圍

4.1 使用抽象類別

繼承的目的之一是讓子類別可以使用父類別的方法。然而，部分情況父類別會希望子類別在繼承之後，必須覆寫自己的某些方法，此時，就可以使用「`abstract`」關鍵字宣告「抽象方法」和「抽象類別」。使用「`abstract`」的原則：

1. 「類別」加上 `abstract` 宣告後：

- 可以被繼承。
- 不可以被實例化。

2. 「方法」加上 `abstract` 宣告後：

- 不可以有內容，亦即不可以有 {}。
- 類別也必須也加上 `abstract` 宣告。
- 子類別若非 `abstract`，就必須覆寫該 `abstract` 方法。

3. 抽象類別可以包含「任何數目」的抽象方法，沒有抽象方法也可以。若抽象類別內沒有抽象方法，表示該類別為抽象意念，不適合被實例化，如類別 `Clothing`、`Animal` 等。

4. 繼承抽象類別只有二個結果：

- 留有未覆寫的抽象方法，子類別必須還是抽象類別。
- 實作所有抽象方法。

5. 使用 `abstract` 宣告的「方法」目的在強制子類別必須實作，所以不可以是 `private`。

6. 使用 `abstract` 宣告的「類別」不能實例化只能繼承，所以不可以有 `final` 宣告。

4.2 使用 `static` 關鍵字

`static` 關鍵字用來宣告欄位或方法屬於「類別成員」：

1. 可以直接以類別名稱呼叫，不需要產生物件，也不需要物件參考。
2. 用於解決不須物件的狀況，如常數、公式。
3. 使用於共享資料。
4. 不符合物件導向的精神，除非有好理由，不該過度使用。

4.2.1 static 的方法與欄位

即便類別未被實例、初始化，無參考變數，以 static 壓告的方法依然可被使用：

1. 稱類別方法 (class methods)。
2. 使用於非物件導向的 API 內，如 java.lang.Math。
3. 可使用於「靜態工廠方法設計模式 (static factory method design pattern)」中，以取代使用「new ~ 語法」的物件初始化流程。如：

```
NumberFormat nf = NumberFormat.getInstance();
```

4. 同一個類別內，只能被同樣是 static 的方法調用。
5. 也可以在子類別中有相同簽名的方法，但沒有覆寫效果。

以下範例顯示 static 方法的建立方式：

範例

```
1 public class StaticErrorClass {
2     private int instanceField;
3     public void instanceMethod() {
4         instanceField = 2;
5     }
6     public static void staticMethod() {
7         instanceField = 1; // 編譯失敗，static 方法不能呼叫物件欄位
8         instanceMethod(); // 編譯失敗，static 方法不能呼叫物件方法
9     }
10 }
```

呼叫 static 方法時，因為不需要物件參考，所以直接使用「類別名稱.方法名稱」，避免使用物件參考來呼叫。如：

範例

```
1 double d = Math.random(); // 使用類別名稱呼叫 static 方法
2 StaticErrorClass.instanceMethod(); // 使用類別名稱呼叫 static 方法
3 StaticErrorClass test= new StaticErrorClass();
4 test.printMessage(); // 可以編譯，但容易混淆，應避免
```

使用 static 欄位的原則同 static 方法，如下例：

範例

```

1  public class StaticCounter {
2      private static int counter = 0;
3      public StaticCounter() {
4          counter++;
5      }
6      public static int getCount() {
7          return counter;
8      }
9      public static void main(String args[]) {
10         new StaticCounter();
11         new StaticCounter();
12         System.out.println("count: " + StaticCounter.getCount());
13     }
14 }
```

以下範例顯示「靜態工廠方法設計模式 (static factory method design pattern)」的使用：

範例

```

1  class TV {
2      public static TV getInstance(String brand) {
3          if (brand.equals("APPLE")) {
4              return new AppleTV();
5          } else if (brand.equals("SONY")) {
6              return new SonyTV();
7          } else {
8              return null;
9          }
10     }
11     public static void main(String args[]) {
12         // 使用建構子生成物件
13         TV sony = new SonyTV();
14         sony.play();
15         // 使用靜態工廠方法取得物件
16         TV apple = TV.getInstance("APPLE");
17         apple.play();
18     }
19 }
```

使用靜態工廠方法生產物件，來取代使用建構子生成物件，好處在於可以多一點周旋空間，比方說在回傳物件實例前可以再做一些設定，或是選擇不同實例回傳等，就像工廠生產貨物。若希望 JVM 內只有一個 AppleTV 的 instance，也可在這裡實現，就是套用更進階的「獨體設計模式 (singleton design pattern)」，會在稍後章節說明。

4.2.2 static import

因為呼叫 static 類別成員時使用「類別名稱.靜態欄位/方法」，為了減少加上「類別名稱」的麻煩，可以使用「static import」，如範例「/OCP/src/course/c04/StaticImportTest.java」：

範例

```

1  import java.io.File;
2  import static java.io.File.separator;
3  import static java.io.File.*;
4  public class StaticImportTest {
5      public static void main(String[] args) {
6          System.out.println( File.separator );
7          System.out.println( separator );
8          System.out.println( pathSeparator );
9      }
10 }
```

說明

- | | |
|---|-------------------------------------|
| 6 | 本行要編譯通過，需要行 1 的 import。 |
| 7 | 本行要編譯通過，需要行 2 或行 3 的 import static。 |
| 8 | 本行要編譯通過，需要行 3 的模糊 import static。 |

4.3 使用final關鍵字

以 final 告示方法

以 final 關鍵字宣告的方法：

- 不能被子類別覆寫。
- 幾乎沒有效能好處。使用唯一考量就是不被子類別覆寫。

如下範例行 7 將編譯失敗：

範例

```

1  public class Super{
2      public final void finalMethod () {
3          System.out.println("This is a final method");
4      }
5  }
```

```

6  public class Sub extends Super {
7      public void finalMethod () { // 無法編譯
8          System.out.println("Cannot override method");
9      }
10 }

```

以 final 宣告類別

以 final 關鍵字宣告的類別：

1. 不能被繼承。
2. 類別前的修飾詞，不可以同時有 abstract 和 final 關鍵字，因為兩者用法衝突。

如以下範例行 3 將編譯失敗：

範例

```

1  public final class FinalClass {
2  }
3  public class ChildClass extends FinalClass {
4  }

```

以 final 宣告變數

以 final 關鍵字宣告的變數表示「初始化(給值)後」不能被修改。可以是：

- 類別欄位 (class fields)。若再加上 public 和 static 告訴，表示記憶體中只有一份，永遠公開可得，無法改變，常用於系統常數。
- 方法參數 (method parameters)。
- 區域變數 (local variables)。

final 若使用於物件參照，只保證所參照的對象不變，但對象的狀態還是可以修改。如下範例行 3 可以通過編譯，但行 5 不行：

範例

```

1  public static void main(String[] args) {
2      final int[] arr = new int[2];
3      arr[0] = 1;           // 編譯通過
4      final int i = 1;
5      i = 2;               // 編譯失敗
6  }

```

以 final 宣告類別欄位

以 final 關鍵字宣告類別欄位時，初始化只有二種選擇：

1. 宣告時同時給值。
2. 在每個建構子中都要給值。

要表現「常數」可使用 static + final 的宣告，命名原則為：

- 全部大寫。
- 有複合字則以底線分隔單字。

範例「/OCP/src/course/c04/FinalFieldClass.java」顯示 final 變數的使用注意事項：

範例

```

1  public class FinalFieldClass {
2      private final int field;
3      private final int forgottenField;
4      private final Date now = new Date();
5      public static final int SOME_CONSTANT = 10;
6      public FinalFieldClass() {
7          field = 100;
8          forgottenField = 200;
9      }
10     public FinalFieldClass(Object o) {
11         field = 300;
12         // compile-time error - forgottenField not initialized
13     }
14     public void changeValues(final int param) {
15         param = 1;           // 編譯失敗
16         now.setTime(0);    // 可以改變物件狀態
17         now = new Date();   // 不能改變物件參考指向
18         final int localVar;
19         localVar = 42;       // 宣告和初始化可以分開
20         localVar = 43;       // 編譯失敗
21     }
22 }
```

說明

5	public + static + final 表示系統常數，命名方式習慣全部大寫字母。
10	本建構子無法通過編譯，因為行 2, 3 所宣告的二個 final 欄位並未初始化。「每個」建構子都必須初始化這二個欄位，避免選擇某建構子初始化物件時，有 final 的欄位未完成初始化。
15	編譯失敗，不能改變 final 的方法輸入參數。

16	final 的物件參考可以改變指向的物件實例的狀態。
17	final 的物件參考不能改指向新的物件實例。
19	final 的區域變數可以分開宣告及初始化。
20	編譯失敗，不能改變 final 的區域變數值。

4.4 實作獨體設計模式

設計模式

「設計模式」是一些特殊的程式設計經驗，可以套用在類似的情境，簡化問題的解決。類似建築裡的工法。比如說，我們有一個理想，想要設計一個符合「綠建築設計」的大樓。但「綠建築」只是概念，就好比先前我們說的「物件導向程式設計(OOP)」或是「物件導向程式分析與設計(OOAD)」；還需要一些更實際的「施做工法」來實現這些概念，「設計模式」就扮演這個工法的角色。對應關係是：「綠建築設計 vs. 施做工法」和「物件導向程式設計 vs. 設計模式」。有一本大師級的經典之作《Design Patterns: Elements of Reusable Object-Oriented Software》由「Erich Gamma et al. (the "Gang of Four")」所撰寫，有興趣的讀者可以參閱。

「設計模式」也有助於程式設計時的溝通，用簡單的情境來說明設計的理念，就像UML一樣。若讀者對UML不清楚，可參閱本書前冊《Java SE8 OCAJP 專業認證指南》的第三章。

獨體設計模式

「獨體設計模式 (Singleton Design Pattern)」是設計模式的一種，用於「確保執行環境或是JVM裡只有一個物件實例」，避免浪費系統資源。比方說：

- 一個公司裡只要有一個執行長。
- 一個教室裡只要有一台印表機。
- 一個網站只需要有一個計數器，統計登入人數，多個計數器反而壞事。

這些情境，就可以套用獨體設計模式如下：

範例

```

1  public class SingletonClass {
2      private SingletonClass() {}
3      private static final SingletonClass instance = new SingletonClass();

```

```

4     public static SingletonClass getInstance() {
5         return instance;
6     }
7 }
```

獨體模式的設計有三個關鍵：

說明

2	建構子必須是 private。獨體模式必須封鎖類別外部使用 new 關鍵字建構物件。因為一旦建構子對外開放，就無法限制其他程式碼呼叫 new SingletonClass() 建立物件，也就無法保證只有單一物件。
3	承上，因為類別外部無法建構物件，就必須由類別內部建構物件，否則物件就根本不存在了。畢竟我們是需要一份，不是不需要。加上 static 關鍵字，更確保 JVM 裡只有一份。加上 final 關鍵字則要求變數永遠指向該物件。使用 private 告訴，加強封裝的概念；但需要搭配另一個公開的 getter 方法。
4~6	承上，提供一個 public getter 方法。因為回傳 static 欄位，所以方法也必須是 static。慣例上使用 getInstance() 名稱，這也是一個「靜態工廠方法」設計模式的實作。

所以，可以使用以下程式碼取得獨體物件，而且保證 JVM 裡只有一份：

```
SingletonClass ref = SingletonClass.getInstance();
```

4.5 列舉型別(enum)

4.5.1 列舉型別初體驗

範例「/OCP/src/course/c04/MachineTest.java」使用了三個整數的常數代表機器的三種電源狀態。這種寫法相當常見，您認為如何？

範例

```

1 class Machine {
2     public static final int POWER_OFF = 0;
3     public static final int POWER_ON = 1;
4     public static final int POWER_SUSPEND = 2;
5     private int state;
6     public void setState(int state) {
7         if (state == 0 || state == 1 || state == 2) {
8             this.state = state;
9         }
10    }
11    public int getState() {
12        return state;
13    }
14 }
```

```

13     }
14 }
15 public class MachineTest {
16     public static void main(String[] args) {
17         Machine c = new Machine();
18         c.setState(Machine.POWER_ON);
19         c.setState(Machine.POWER_SUSPEND);
20         c.setState(Machine.POWER_OFF);
21         System.out.println("before:" + c.getState());
22         c.setState(30);
23         System.out.println("after: " + c.getState());
24     }
25 }
```

結果

```
before:0
after:0
```

這個範例程式嚴謹使用 final 欄位，在設定電源狀態時也相當小心，先「檢查」是否是預期的三個整數。美中不足的地方在於「執行時期的檢查工作」，畢竟是一種效能的消耗。

為了解決這種效能浪費，Java 在版本 5 的時候推出一個新的列舉 (Enumeration) 型別。可以在原先宣告 class、interface 的地方，改使用「enum」關鍵字。這種新的列舉型別可以在編譯時期進行檢查是否合格，不需要執行時期的動態檢查，相較之下可以減少效能的浪費，也可以增進安全性，是一種「型態安全 (type-safe)」的概念，這和泛型 (generic) 的設計理念雷同。

所以升級上例成為「/OCP/src/course/c04/MachineTest2.java」：

範例

```

1 enum PowerState {
2     OFF, ON, SUSPEND;
3 }
4 class Machine2 {
5     private PowerState state;
6     public void setState( PowerState state) {
7         this.state = state;
8     }
9     public PowerState getState() {
10        return state;
11    }
12 }
```

```

13 public class MachineTest2 {
14     public static void main(String[] args) {
15         Machine2 c = new Machine2();
16         c.setState(PowerState.ON);
17         c.setState(PowerState.SUSPEND);
18         c.setState(PowerState.OFF);
19         System.out.println(c.getState());
20         System.out.println(c.getState().ordinal());
21         //c.setState(30); //compile error
22     }
23 }
```

結果

OFF

0

說明

1	使用 enum 關鍵字宣告 PowerState 列舉型別。
2	PowerState 列舉了三種項目。
5	enum 的使用方式，和一般類別相似，也可以作為參考型別。
16	Machine 物件要設定 PowerState，選擇 ON 狀態。
19	印出 Machine 物件目前的 PowerState。
20	enum 裡每個項目都可以呼叫 ordinal() 方法，取得自己的順序編號，由 0 開始。如：OFF=0, ON=1, SUSPEND=2，類似 index。
21	因為 Machine 的 setState() 方法改要求傳入 PowerState 列舉型別，且只能傳入三種定義的列舉項目。傳入 int 將無法通過編譯。

4.5.2 列舉型別的使用方式

列舉型別可以作為參考型別使用。因為列舉型別裡的列舉項目具備 static 特性，因此可以支援「static import」；也可以用於「switch 選擇結構」中。如範例「/OCP/src/course/c04/MachineTest3.java」：

範例

```

1 import static course.c04.PowerState.*;
2 public class MachineTest3 {
3     public static void main(String[] args) {
4         Machine2 c = new Machine2();
5         c.setState(ON);
6         c.setState(OFF);
```

```

7         System.out.println(getDescription(SUSPEND));
8     }
9     static String getDescription(PowerState state) {
10        switch (state) {
11            case OFF:
12                return "The power is off";
13            case ON:
14                return "The power is high";
15            case SUSPEND:
16                return "The power is low";
17            default:
18                return "unknown state";
19        }
20    }
21 }
```

4.5.3 進階型列舉型別

Java 的列舉型別 enum 實際上是可以更複雜的。首先讓我們由先前範例來了解 enum 的真相：

- 列舉型別的使用方式為「列舉型別名稱.列舉項目」，如「PowerState.ON」。其中「列舉項目」其實是物件實例。如下圖用 Eclipse 的快捷功能顯示「列舉項目」可以使用的方法，可以發現具備一般物件實例該具備的所有功能，如 equals()、hashCode()、toString() 等。一個列舉型別裡可以有數個列舉項目，等同於一個 enum 將產生數個物件實例：

```

1 package course.c04;
2
3 import static course.c04.PowerState.*;
4
5 public class MachineTest4 {
6     public static void main(String[] args) {
7         System.out.println(ON);
8         System.out.println(SUSPEND);
9     }
10 }
11 
```

● compareTo(PowerState o) : int -> Enum
● equals(Object other) : boolean -> Enum
● hashCode() : int -> Enum
● name() : String -> Enum
● ordinal() : int -> Enum
● toString() : String -> Enum
● getClass() : Class<T> -> Object
● getDeclaringClass() : Class<PowerState> -> Enum
U OFF : PowerState
U ON : PowerState
U SUSPEND : PowerState
U PowerState : PowerState
● valueOf(String arg0) : PowerState -> PowerState
● valueOf(Class<T> enumType, String name) : T -> Enum
● values() : PowerState[] -> PowerState
● notify() : void -> Object
● notifyAll() : void -> Object
● wait() : void -> Object
● wait(long timeout) : void -> Object
● wait(long timeout, int nanos) : void -> Object

✿ 圖 4-1 enum 項目具備的方法

2. 列舉型別的「列舉項目」支援「static import」，表示是 static 的物件成員。因為 static 成員使用類別名稱來呼叫，所以把「列舉型別名稱」當成類別名稱，使用「列舉型別名稱.列舉項目」的方式呼叫列舉項目，如「PowerState.ON」。
3. 經過簡單的存取限制的測試後，可以發現列舉項目到處可以使用，沒有 package 及是否繼承的限制，因此具備 public 的特質。最後，這類實例一旦生成，也無法修改，所以是 final。結合了 public + static + final，因此可以用來取代範例「MachineTest.java」中的常數。

既然每個「列舉項目」等同於類別的物件欄位，自然也具備建構子、屬性、和方法。因為所有「列舉項目」都內含在同一個「列舉型別」下，因此「列舉型別」所定義的建構子、屬性和方法，就設計成讓每個「列舉項目」共用。如範例「/OCP/src/course/c04/ComplexPowerState.java」：

範例

```

1  public enum ComplexPowerState {
2      OFF("The power is off"),
3      ON("The power is high"),
4      SUSPEND("The power is low");
5      private String description;
6      private ComplexPowerState(String d) {
7          description = d;
8      }
9      public String getDescription() {
10         return description;
11     }
12     public void setDescription(String d) {
13         description = d;
14     }
15 }
```

說明

6 定義了 enum 的建構子，讓三個列舉項目實例 OFF、ON、SUSPEND 使用。因為這個建構子必須輸入字串，因此行 2、3、4 的列舉項目，就必須使用符合的建構子的形式：

```

OFF ("The power is off"),
ON ("The power is high"),
SUSPEND ("The power is low");
```

可以這樣「想像」：

```

new OFF ("The power is off"),
new ON ("The power is high"),
new SUSPEND ("The power is low");
```

也因為列舉項目的實例建立方式特別，又必須符合 public + static + final，因此由 Java 自己操控，所以建構子宣告必須是 private，未註明時預設也是 private，不讓我們自己建立。

如果以類別 class 來模擬 enum 列舉型別的特色，可以將前範例改寫為「/OCP/src/course/c04/SimulatePowerState.java」。前後範例對照，會對 enum 更清楚：

範例

```

1  public class SimulatePowerState {
2      public static final SimulatePowerState OFF
3          = new SimulatePowerState("The power is off");
4      public static final SimulatePowerState ON
5          = new SimulatePowerState("The power is high");
6      public static final SimulatePowerState SUSPEND
7          = new SimulatePowerState("The power is low");
8      private String description;
9      private SimulatePowerState (String d) {
10         description = d;
11     }
12     public String getDescription() {
13         return description;
14     }
15     public void setDescription(String d) {
16         description = d;
17     }
18 }
```

列舉型別 (enum) 其實就是類別 (class) 的變形，只不過很多細節由編譯器暗地裡做了調整，所以某些程度上列舉型別依然具備類別的特性：

1. 列舉型別 (enum) 繼承自 java.lang.Enum 類別，而每個被列舉的項目都是列舉型別的欄位物件實例。預設是「final」，所以無法改變值；也是「public」且「static」的成員，所以可以透過列舉型別的名稱來使用它們，就像類別名稱一樣。
2. 因為列舉型別的建構子為 private，且每個列舉實例都是 public、static、final，因此列舉型別 (enum) 也是獨體模式 (Singleton Design Pattern) 的實作之一。
3. Java 支援單一繼承，既然 enum 已經繼承 java.lang.Enum 類別，就不能再繼承其他類別，但可以實作其他介面，如範例「/OCP/src/course/c04/enums/AlertAblePowerState.java」：

範例

```

1  interface AlertAble {
2      void alert();
3  }
4  public enum AlertAblePowerState implements AlertAble {
5      OFF("The power is off") {
```

```

6     @Override
7     public void alert() {
8         System.out.println("OFF alert");
9     }
10    },
11    ON("The power is high"),
12    SUSPEND("The power is low");
13    @Override
14    public void alert() {
15        System.out.println("OFF alert");
16    }
17    private String description;
18    private AlertAblePowerState(String d) {
19        description = d;
20    }
21    public String getDescription() {
22        return description;
23    }
24    public void changeDesc(String d) {
25        description = d;
26    }
27 }

```

本例中宣告列舉型別 AlertAblePowerState 實作了介面 AlertAble，因此 enum 內的所有列舉項目，含 OFF、ON、SUSPEND，都該提供方法 alert() 的實作內容。作法有兩種：

1. 在列舉型別的程式碼區塊裡，如行 13~16，提供 alert() 方法的實作。如此所有列舉項目，含 OFF、ON、SUSPEND，都可以使用該方法。
2. 若個別列舉項目，想提供專屬自己的 alert() 方法的實作，也可以如行 6~9 的方式進行實作。如此，列舉項目 ON 和 SUSPEND 的 alert() 實作為行 13~16，列舉項目 OFF 則為行 6~9。

最後，以範例「/OCP/src/course/c04/enums/AlertAblePowerStateTest.java」進行測試：

範例

```

1  public class AlertAblePowerStateTest {
2      import static java.lang.System.out;
3      public static void main(String[] args) {
4          // 測試列舉項目順序
5          out.println(AlertAblePowerState.OFF.ordinal());
6          out.println(AlertAblePowerState.ON.ordinal());
7          out.println(AlertAblePowerState.SUSPEND.ordinal());

```

```

8     // 測試列舉項目方法
9     out.println(AlertAblePowerState.OFF.getDescription());
10    out.println(AlertAblePowerState.ON.getDescription());
11    out.println(AlertAblePowerState.SUSPEND.getDescription());
12    AlertAblePowerState.OFF.changeDesc("the power is shutdown");
13    out.println(AlertAblePowerState.OFF.getDescription());
14    // 測試實作介面的方法
15    AlertAblePowerState.OFF.alert();
16    AlertAblePowerState.ON.alert();
17    AlertAblePowerState.SUSPEND.alert();
18 }
19 }

```

結果

```

0
1
2
The power is off
The power is high
The power is low
the power is shutdown
OFF alert
default alert
default alert

```

4.6 使用巢狀類別

4.6.1 巢狀類別的目的與分類

巢狀類別的目的

巢狀類別 (Nested Classes)，簡單地說，就是「在類別內宣告其他的類別」，和過去每個屬於平行存在的類別宣告，有一些不同。對比於一般類別，有幾種常見的稱呼方式，如：

1. 「內部類別 (Inner Class)」，對比於「外部類別 (Outer Class)」。
2. 「巢狀類別 (Nested Class)」，對比於「頂層類別 (Top-Level Class)」。
3. 「被包覆類別 (Enclosed Class)」，對比於「包覆類別 (Enclosing Class)」。

使用巢狀類別的目的通常是：

- 將邏輯上依存度高的類別，放在一起：

一個類別存在的目的主要是協助另一類別時，稱為「幫手類別 (Helper Class)」。邏輯上將幫手類別放在主要類別內相當合理。

- 提高封裝性：

巢狀類別可以直接存取外部類別內的私有 (private) 成員，不需要放寬存取層級，對於類別的封裝性比較好。

- 建立可讀性高，易維護的程式碼：

關係緊密的類別放在一起，一目瞭然，較好維護。

- 常使用於 Java 的「圖形化使用者介面 (Graphic User Interface)，簡稱 GUI」的程式設計，如 SWING。

巢狀類別的分類

巢狀類別宣告於外部類別內，可分為二大類：

1. 內部類別 (Inner Classes)，又可分為三種：

- 成員類別 (Member Classes)：位階同於物件成員。
- 區域類別 (Local Classes)：宣告於方法內，作用範圍和區域變數一致。
- 匿名類別 (Anonymous Classes)：沒有名稱的巢狀類別。

2. 靜態巢狀類別 (Static Nested Classes)：位階同於類別成員。

課堂小秘訣

這樣的分類方式，其實就和我們撰寫類別時，將成員(欄位和方法)以是否有 static 告知做的分類相同：

- 無 static 告知：物件成員。
- 有 static 告知：類別成員。

所以：

- 內部類別 (Inner Classes) 就是「物件成員」的一種。
- 靜態巢狀類別 (Static Nested Classes) 就是「類別成員」的一種。

使用的習慣也相同：

- 建立內部類別 (Inner Classes) 時必須先有外部類別的物件參考 (遙控器)，如同使用其他物件成員。
- 建立靜態巢狀類別 (Static Nested Classes) 時可以直接使用外部類別名稱。

以下範例為 Inner Class 的簡單示範：

範例

```

1  public class Car {
2      private boolean running = false;
3      private InnerEngine engine = new InnerEngine();
4      private class InnerEngine {
5          public void start() {
6              running = true;
7          }
8      }
9      public void start() {
10         engine.start();
11     }
12 }
```

在前述範例中，行 4~8 就是我們定義的內部 `InnerEngine` 類別，這樣的設計至少有二個好處：

1. 每台汽車都有一個引擎，該引擎只協助該台汽車，故設計於 `Car` 類別裡，屬於幫手類別。這樣設計邏輯清楚，讓人容易了解兩者的依存關係。
2. 內部類別可以直接存取外部類別的 `private` 成員，因此提高外部類別的封裝性。若 `InnerEngine` 類別沒有放在 `Car` 類別內部，是否表示 `private` 的欄位 `running` 就必須至少開放至 `default` 層級了呢？

4.6.2 匿名巢狀類別

巢狀類別多數並不困難，只是將類別宣告在另一個類別內，或是加上 `static` 宣告。只有「匿名類別(Anonymous Class)」涉及語法的改變，因此必須特別注意。

「匿名類別(Anonymous Class)」顧名思義，就是沒有名字的類別。有些類別宣告只是暫時用途，只用於生成物件一次，因此沒有必要正式宣告一個類別；但畢竟還是需要使用 `new` 關鍵字生成物件一次，那沒有類別名稱的物件如何建構實例？答案是使用「父類別或介面」來建構所需要的匿名子類別實例。如範例「/OCP/src/course/c04/AnonymousSamples.java」：

範例

```

1  class MySuper {
2      void doIt() {}
3  }
4  interface MyInterface {
```

```

5      void doIt();
6  }
7 class MySub extends MySuper {
8     void doIt() {
9         System.out.println("This is MySub");
10    }
11 }
12 class MyImpl implements MyInterface {
13     public void doIt() {
14         System.out.println("This is MyImpl");
15     }
16 }
17 public class AnonymousSamples {
18     MySuper c1 = new MySub();
19     MySuper c2 = new MySuper() {
20         void doIt() {
21             System.out.println("This is Anonymous Sub class");
22         }
23     };
24     MyInterface i1 = new MyImpl();
25     MyInterface i2 = new MyInterface() {
26         public void doIt() {
27             System.out.println("This is Anonymous Impl class");
28         }
29     };
30     public static void main(String args[]) {
31         AnonymousSamples c = new AnonymousSamples ();
32         c.c1.doIt();
33         c.c2.doIt();
34         c.i1.doIt();
35         c.i2.doIt();
36     }
37 }

```

說明

1~3	定義父類別 MySuper。
4~6	定義介面 MyInterface。
7~11	建立一般子類別 MySub。
12~16	建立一般實作 MyImpl。
18	使用一般子類別建立物件實例。
19~23	以匿名巢狀類別建立物件實例，使用父類別。
24	使用一般實作建立物件實例。
25~29	以匿名巢狀類別建立物件實例，使用介面。

使用一般子類別建立物件實例的過程為：

1. 定義子類別

範例

```
7  class MySub extends MySuper {
8      void doIt() {
9          System.out.println("This is MySub");
10     }
11 }
```

2. 使用子類別建立物件實例

範例

```
18 MySuper c1 = new MySub();
```

巢狀類別就是要將上述兩者合一，減少定義子類別的程序：

範例

```
19 MySuper c2 = new MySuper() {
20     void doIt() {
21         System.out.println("This is Anonymous Sub class");
22     }
23 };
```

因此，以巢狀類別建立物件實例的語法分為五個區段，以「new」關鍵字開頭，以「;」結尾：

表 4-1 巢狀類別語法分析

語法區段	1	2	3	4	5
區段內容	new	父類別\介面名稱	()	{ 覆寫覆類別\介面 程式碼區塊 }	；

此外，匿名巢狀類別雖然沒有明確的類別宣告，但編譯時還是會產生以 class 為附檔名的編譯檔。以上例而言，會產生二個巢狀類別的編譯檔：

- AnonymousSamples\$1.class
- AnonymousSamples\$2.class

其中，AnonymousSamples 是外部類別名稱。因為匿名巢狀類別沒有名稱，因此「\$」後面接上編號，以不同編號代表不同匿名類別；編號由 1 開始，依次 2、3、4…。

4.6.3 巢狀類別綜合範例

範例「/OCP/src/course/c04/EnclosingClass.java」顯示所有巢狀類別的宣告方式與位置：

範例

```

1  public class EnclosingClass {
2      private int privateField = 101;
3      // Anonymous member classes
4      public Object o = new Object() {
5          @Override
6          public String toString() {
7              return "Anonymous class as field (object member)";
8          }
9      };
10     // Member classes
11     class MemberInner {
12         public void run() {
13             System.out.println("Member class: " + privateField);
14         }
15     }
16     // Static nested classes
17     static class StaticNestedClass {
18         public void run() {
19             System.out.println("Static nested class");
20         }
21     }
22     public void test1() {
23         // Anonymous local classes
24         Object o = new Object() {
25             @Override
26             public String toString() {
27                 return "Anonymous class as local variable";
28             }
29         };
30         System.out.println(o);
31         System.out.println(this.o);
32     }
33     public void test2() {
34         // Local classes
35         class LocalInner {
36             public void run(String s) {
37                 System.out.println(s);
38             }
39         }

```

```

40         new LocalInner().run("Local classes: " + privateField);
41     }
42
43     public static void main(String[] args) {
44         EnclosingClass outer = new EnclosingClass();
45         outer.test1();
46         outer.test2();
47         // 初始化 Inner classes
48         MemberInner inner = outer.new MemberInner();
49         inner.run();
50         // 初始化 Static nested classes
51         StaticNestedClass staticNested =
52             new EnclosingClass.StaticNestedClass();
53         staticNested.run();
54     }
}

```

上述範例完整呈現所有巢狀類別的定義方式。必須注意的是，巢狀類別不只在外部類別的「內部」使用，只要不是宣告為 `private`，也可以在外部類別的「外部」使用，建立方式分二種：

1. Inner Class

使用方式和「物件成員」相同，必須先有外部類別的參考，才可以建立內部類別。如行 48。

2. Static Nested Class

使用方式和「類別成員」相同。需在內部類別名稱的前面，加上「外部類別名稱.」，作為完整類別名稱，然後使用 `new` 關鍵字建立物件實例。如行 51。

4.7 認證考試命題範圍

依據甲骨文公布的考試範圍 (https://education.oracle.com/java-se-8-programmer-ii/pexam_1Z0-809)，和本章相關的主題有：

Java Class Design

1. Create and use **singleton** classes and **immutable** classes.
2. Develop code that uses **static** keyword on initialize blocks, variables, methods, and classes.

Advanced Java Class Design

1. Develop code that uses **abstract** classes and methods.
2. Develop code that uses the **final** keyword.
3. Create inner classes including **static inner** class, **local** class, **nested** class, and **anonymous** inner class.
4. Use **enumerated** types including methods, and constructors in an enum type.

本章擬真試題實戰

考題 1

Given:

```
abstract class Account {  
    abstract void deposit(double amt);  
    public abstract boolean withdraw(double amt);  
}  
  
class SubAccount extends Account {  
}
```

What two changes, made independently, will enable the code to compile?

- A. Change the signature of Account to: public class Account.
- B. Change the signature of SubAccount to: public abstract class SubAccount
- C. Implement private methods for deposit and withdraw in SubAccount.
- D. Implement public methods for deposit and withdraw in SubAccount.
- E. Change Signature of SubAccount to: SubAccount implements Account.
- F. Make Account an interface.

答案 BD

考題 2

Given:

```
enum DirectionType {  
    EAST, WEST, SOUTH, NORTH;  
}
```

Which statement will iterate through Direction?

- A. for (DirectionType d : DirectionType.values()) { //...}
- B. for (DirectionType d : DirectionType.asList()) { //...}
- C. for (DirectionType d : DirectionType.iterator()) { //...}
- D. for (DirectionType d : DirectionType.toArray()) { //...}

答案 A

考題 3

Given:

```
public class Test {
    public static String name = "Nothing";
    public void begin() {
        System.out.println(name);
    }
    public static void main(String[] args) {
        name = "Jim";
        begin ();
    }
}
```

What is the result?

- A. Jim
- B. Nothing
- C. It may print "Nothing" or "Jim" depending on the JVM implementation.
- D. Compilation fails.
- E. An exception is thrown at runtime.

答案 D

說明 必須將方法 begin () 改宣告為 static。

考題 4

Which four are true about enums?

- A. An enum is typesafe.

- B. An enum cannot have public methods or fields.
- C. An enum can declare a private constructor.
- D. All enums implicitly implement Comparable.
- E. An enum can subclass another enum.
- F. An enum can implement an interface.

答案 ACDF

說明 選項B：enums 可以有 public 的 methods 和 fields。選項E、F：enums 已經暗中繼承 `java.lang.Enum`，因此不能再繼承其他類別；但可以實作其他介面。

考題 5

Given:

```
class Test {  
    String field;  
    static class Counter {  
        int counter;  
        void counting() {  
            counter++;  
        }  
    }  
    public static void main(String[] args) {  
        // insert code here  
    }  
}
```

Which statement, inserted at //insert code here, enables the code to compile?

- A. new Test().new Counter().counting();
- B. new Test().Counter().counting();
- C. new Test.Counter().counting();
- D. Test.Counter().counting();
- E. Test.Counter.counting();

答案 C

說明 請參閱範例「/OCP/src/course/c04/EnclosingClass.java」及其說明。

考題 6

Which two are true about Singletons?

- A. A Singleton must implement Serializable.
- B. A Singleton has only the default constructor.
- C. A Singleton implements a factory method.
- D. A Singleton improves a class's cohesion.
- E. Singletons can be designed to be thread-safe.

答案 CE

說明 選項A：和「是否可以序列化」無關。選項B：要將 constructor 改寫為 private，故沒有 default constructor。選項C：使用 getInstance() 方法。選項D：使用多型才會改善 class's cohesion(內聚性)。選項E：使用 synchronized 方法即可。

考題 7

Which two properly implement a Singleton pattern?

A.

```
class SingletonA {
    private static SingletonA instance;
    private SingletonA() { }
    public static synchronized SingletonA getInstance() {
        if (instance == null) {
            instance = new SingletonA();
        }
        return instance;
    }
}
```

B.

```
class SingletonB {
    private static SingletonB instance = new SingletonB();
    protected SingletonB() { }
    public static SingletonB getInstance() {
        return instance;
    }
}
```

C.

```
class SingletonC {  
    SingletonC() { }  
    private static class SingletonHolder {  
        private static final SingletonC INSTANCE = new SingletonC();  
    }  
    public static SingletonC getInstance() {  
        return SingletonHolder.INSTANCE;  
    }  
}
```

D.

```
enum SingletonD {  
    INSTANCE;  
}
```

答案 AD

說明

選項 A：無法使用 `SingletonA a = new SingletonA();`，必須使用

`SingletonA a = SingletonA.getInstance();`。

選項 B：可以使用 `SingletonB b = new SingletonB();`，證明不是獨體模式。

選項 C：可以使用 `SingletonC c = new SingletonC();`，證明不是獨體模式。

選項 D：無法使用 `SingletonD d = new SingletonD();`，必須使用

`SingletonD d = SingletonD.INSTANCE;`。

考題 8

Given:

```
1. class A {  
2.     private void a() {}  
3.     class B {  
4.         private void b() {  
5.             a();  
6.         }  
7.     }  
8.     public static void main(String[] args) {  
9.         B bb = new A().new B();  
10.        b();  
11.    }  
12. }
```

What is the result?

- A. Compilation fails at line 9
- B. Compilation fails at line 10
- C. Compilation fails at line 5
- D. Compilation fails at line 3
- E. Compilation succeeds

答案 B

考題 9

Given:

```

1. final class FinalTest {
2.     final String location;
3.     FinalTest(final String loc) {
4.         location = loc;
5.     }
6.     FinalTest(String loc, String s) {
7.         location = loc;
8.         loc = "unknown";
9.     }
10.}

```

What is the result?

- A. Compilation succeeds.
- B. Compilation fails due to an error on line 1.
- C. Compilation fails due to an error on line 2.
- D. Compilation fails due to an error on line 3.
- E. Compilation fails due to an error on line 4.
- F. Compilation fails due to an error on line 8.

答案 A

說明

1. final 的欄位最晚必須在建構子裡完成初始化。
2. final 的方法參數不能修改。

考題 10

A valid reason to declare a class as abstract is to:

- A. define methods within a parent class, which may not be overridden in a child class
- B. define common method signatures in a class, while forcing child classes to contain unique method implementations
- C. prevent instance variables from being accessed
- D. prevent a class from being extended
- E. define a class that prevents variable state from being stored when object instances are serialized
- F. define a class with methods that cannot be concurrently called by multiple threads

答案 B

說明 抽象類別的抽象方法可以迫使子類別必須提供自己的實作 (forcing child classes to contain unique method implementations)，故選 B。

考題 11

Given:

```
abstract class Boat {  
    String row() {  
        return "rowing";  
    }  
    abstract void dock();  
}  
  
class FishBoat extends Boat {  
    public static void main(String[] args) {  
        Boat b = new FishBoat();           // line1  
        Boat b2 = new Boat();             // line2  
    }  
    String row() {return "slow row";}   // line3  
    void dock() {}                   // line4  
}
```

Which two are true about the lines labeled A through D?

- A. The code compiles and runs as is.
- B. If only //line1 is removed, the code will compile and run.

- C. If only //line2 is removed, the code will compile and run.
- D. If only //line4 is removed, the code will compile and run.
- E. //line3 is optional to allow the code to compile and run.
- F. //line3 is mandatory to allow the code to compile and run.

答案 CE

說明 // line2 導致程式碼無法通過編譯，必須移除。

考題 12

Given:

```
final class IceCream {
    public void cook() {
    }
}

class Cake {
    public final void bake(int time, int heat) {
    }
    public void mix() {
    }
}

class Degree85 {
    private Cake c = new Cake();
    public void create() {
        c.bake(15, 110);
    }
}

class BerryCake extends Cake {
    public void bake(int minutes, int temperature) {
    }
    public void addBerry() {
    }
}
```

Which statement is true?

- A. Compilation error occurs in IceCream.
- B. Compilation error occurs in Cake.
- C. Compilation error occurs in Degree85.

D. Compilation error occurs in BerryCake

E. All classes compile successfully.

答案 D

說明 嘗試宣告為 final 的方法不能在子類別覆寫 (override)。

考題 13

Given:

```
interface Paintable {
    public abstract void paint();
}

class Canvas implements Paintable {
    public void paint() {
    }
}

abstract class WhiteBoard extends Canvas {
}

class Paper extends Canvas {
    protected void paint(char color) {
    }
}

class Frame extends Canvas implements Paintable {
    public void changeSize() {
    }
}
```

Which statement is true?

A. WhiteBoard compiles failed.

B. Paper compiles failed.

C. Frame compiles failed.

D. Paintable compiles failed.

E. All classes compile successfully.

答案 E

考題 14

Which two reasons should you use interfaces instead of abstract classes?

- A. You expect that classes that implement your interfaces have many common methods or fields, or require access modifiers other than public.
- B. You expect that unrelated classes would implement your interfaces.
- C. You want to share code among several closely related classes.
- D. You want to declare non-static or non-final fields.
- E. You want to take advantage of multiple inheritance of type.

答案 BE

說明

由原廠文件的說明文件：<https://docs.oracle.com/javase/tutorial/java/IandI/abstract.html>

Consider using abstract classes if any of these statements apply to your situation:

- You want to share code among several closely related classes.
- You expect that classes that extend your abstract class have many common methods or fields, or require access modifiers other than public (such as protected and private).
- You want to declare non-static or non-final fields. This enables you to define methods that can access and modify the state of the object to which they belong.

Consider using interfaces if any of these statements apply to your situation:

- You expect that unrelated classes would implement your interface. For example, the interfaces Comparable and Cloneable are implemented by many unrelated classes.
- You want to specify the behavior of a particular data type, but not concerned about who implements its behavior.
- You want to take advantage of multiple inheritance of type.

考題 15

Given:

```
interface Actable {
    public void doThis(String s);
}
```

Which two class definitions compile?

A.

```
abstract class Task implements Actable {  
    public void doThat(String s) { }  
}
```

B.

```
public abstract class Work implements Actable {  
    public abstract void doThis(String s) { }  
    public void doYourThing(Boolean b) { }  
}
```

C.

```
class Job implements Actable {  
    public void doThis(Integer i) { }  
}
```

D.

```
class Action implements Actable {  
    public void doThis(Integer i) { }  
    public String doThis(Integer j) { }  
}
```

E.

```
class Do implements Actable {  
    public void doThis(Integer i) { }  
    public void doThis(String s) { }  
    public void doThat(String s) { }  
}
```

答案 AE

說明 選項B：抽象方法不能有{}。選項C：方法參數改為(String s)。選項D：改為

```
class Action implements Actable {  
    public void doThis(String i) { }  
    public String doThis(Integer j) {  
        return null;  
    }  
}
```

考題 16

Given:

```
enum USMoney {
    PENNY(1), NICKLE(5), DIME(10);
    private int worth;
    public USMoney(int worth) {
        this.worth = worth;
    }
    public int getWorth() {
        return worth;
    }
}
```

And:

```
class Test {
    public static void main(String[] args) {
        USMoney curr = new USMoney.DIME;
        System.out.println(curr.getWorth());
    }
}
```

Which two modifications enable the given code to compile?

- A. Nest the USMoney enumeration declaration within the Test class.
- B. Make the USMoney enumeration constructor private.
- C. Remove the new keyword from the instantiation of curr.
- D. Make the getter method of value as a static method.
- E. Add the final keyword in the declaration of value.

答案 BC

考題 17

Given:

```
class Automobile {
    int kilometer;      //line1
    Automobile(int x) {
        this.kilometer = x;
```

```
    }
    public void increaseSpeed(int hour) {    // line2
        int interval = hour;                // line3
        class Car {
            int velocity = 0;
            public void speedUp() {
                velocity = kilometer / interval;
                System.out.println("Velocity with new speed " + velocity + " kmph");
            }
        }
        new Car().speedUp();
    }
}
```

And:

```
public static void main(String[] args) {
    Automobile v = new Automobile(80);
    v.increaseSpeed (50);
}
```

What is the result?

- A. Velocity with new speed 1 kmph.
- B. Compilation error occurs at //line1.
- C. Compilation error occurs at //line2.
- D. Compilation error occurs at //line3.

答案 A

使用interface

-
- | 5.1 使用Interface
 - | 5.2 使用設計模式
 - | 5.3 使用複合
 - | 5.4 認證考試命題範圍

5.1 使用Interface

Interface 是物件導向程式設計裡的關鍵，唯有適當使用 interface，才能將物件導向程式語言的彈性發揮到最大。本章除了複習 interface 概念外，也將介紹以 interface 為核心的 DAO 設計模式。

此外，Java 只能單一繼承，因此有些時候我們也使用複合 (composition) 的設計理念，來取代繼承，達成程式碼重複使用的目的。

5.1.1 實作「取代機制」

使用「抽象的參考型別」是 Java 在物件導向裡的重要功能。可分成「abstract class」和「interface」等兩大類。藉由參照抽象型別，就不用綁定特定的實作類別，具體好處有：

- 方便系統維護：若實作類別有問題，或有新的實作類別，可以直接改變，不影響程式架構。
- 實作內容取代：如「java.sql.*」套件可以讓程式連線各種資料庫；但各家資料庫的處理，則由資料庫廠商自己處理；妥慎設計後即可互相取代。
- 方便分工：讓程式操作介面和商業邏輯的開發，可以分頭進行。

5.1.2 Interface 設計要點

Java interfaces 用來定義 abstract types (抽象型態)，有以下基本特色：

- 類似 abstract class，但只允許「public」而且「abstract」的方法。
- 子類別若非 abstract class，一定要實作所有方法。
- 可以包含常數。
- 可以作為宣告時的參考型別。
- 在設計模式 (design patterns) 中扮演重要角色。

設計 interface 時，須注意：

- 類別實作 interface 時，使用 implements，而非 extends。
- 方法只能是「public」且「abstract」，未標出時將預設使用。
- 欄位只允許常數，因此宣告必須是「public」且「static」且「final」，未標出時預設使用。
- 避免系統所有常數，都放在同一個 interface 裡。

如以下 interface 的宣告範例：

範例

```

1  public interface ElectronicDevice {
2      public static final String WARNING = " handle with care!";
3      int x = 10;          // is still public & static & final
4      public abstract void turnOn();
5      void turnOff();    // is still public & abstract
6  }

```

實作的類別範例：

範例

```

1  public class Television implements ElectronicDevice {
2      public void turnOn() { }
3      public void turnOff() { }
4      public void changeChannel(int channel) {}
5  }

```

物件導向程式設計的精神，應該使用最一般(未特化)的參考型態來宣告型別，如 interface。此時：

- 只能使用 interface 上有定義的欄位和方法。
- interface 隱含 java.lang.Object 的所有方法。

如接續前述範例，可以建立物件並使用：

範例

```

1  ElectronicDevice d= new Television();
2  d.turnOn();
3  d.turnOff();
4  d.changeChannel(58);      // 特化方法，無法編譯
5  String s = d.toString();  // 可使用 Object 類別的所有方法

```

好處是未來若需要抽換實作，只需要修改 new 關鍵字後面的子類別即可。如：

範例

```
1  ElectronicDevice d = new DvdPlayer();
```

可以使用 instanceof 運算子判斷是否有實作某 interface。如：

範例

```

1 class Super {}
2 interface MyInterface {}
3 public class Child extends Super implements MyInterface {
4     public static void main(String args[]) {
5         MyInterface i = new Child();
6         System.out.println(i instanceof Object);
7         System.out.println(i instanceof MyInterface);
8         System.out.println(i instanceof Super);
9         System.out.println(i instanceof Child);
10        Super s = new Child();
11        System.out.println(s instanceof Object);
12        System.out.println(s instanceof MyInterface);
13        System.out.println(s instanceof Super);
14        System.out.println(s instanceof Child);
15    }
16 }
```

結果

```
true
true
true
true
true
true
true
true
```

但是，這樣不是好的設計：

範例

```

1 public static void turnObjectOn( Object o ) {
2     if (o instanceof ElectronicDevice) {
3         ElectronicDevice e = (ElectronicDevice)o;
4         e.turnOn();
5     }
6 }
```

雖然方法的參數使用了最寬大的型別：Object，但未來可能會因為新增類別的出現，必須經常增加if else的程式碼以處理新增類別，反而違反了物件導向設計原則裡的「開閉法則(Open Close Principle)」。

此外，子類別可以繼承一個父類別，同時實作其他 interface，但繼承的語法必須放在前面：

範例

```
1 public class SuperCar extends BasicCar
2           implements Flyable { }
```

也可以同時實作多個 interface，以 "," 區隔：

範例

```
1 public class SuperCar extends BasicCar
2           implements Flyable, java.io.Serializable { }
```

interface 可以繼承其他 interface：

範例

```
1 public interface SpecialFunction { }
2 public interface Flyable extends SpecialFunction { }
```

5.1.3 Marker interfaces

「Marker interfaces」指沒有定義任何方法的 interface。顧名思義，這類 interface 的唯一用途只在幫類別做記號，方便日後使用 instanceof 運算子檢查型別，做特殊用途。如最常見的「java.io.Serializable」：

```
public class Car implements java.io.Serializable {
}
```

「java.io.Serializable」用來決定物件是否可以序列化 (serialize) 自己的欄位狀態，如：

範例

```
1 Car c = new Car();
2 if (c instanceof Serializable) {
3     // do something, 在本書第 8 章將介紹序列化的相關主題
4 }
```

5.2 使用設計模式

5.2.1 設計模式和 interface

「封裝、繼承、多型」是物件導向程式語言的三大特色。但要讓物件導向程式語言發揮它強大的威力，就必須仰賴「物件導向程式設計原則(OOAD)」，具體的做法就是「設計模式(Design Pattern)」。這概念好比「建築理念」與「實作工法」的關聯性，「設計原則」則太抽象，所以「設計模式」提供了較實際的做法。

有一段來自物件導向程式設計的名言：

「Program to an interface, not an implementation.」

因此，interface 的使用，成了絕大多數設計模式的特色。本章介紹二種常見的設計模式：

- DAO 設計模式。
- 工廠(Factory) 設計模式。

藉由這兩個模式的認識與使用，可以感受一下設計模式的應用。

5.2.2 DAO 設計模式

DAO 設計模式用於程式必須保存(persist)資料時。套用模式後可以：

1. 分離「商業邏輯」和「資料保存機制」，降低程式碼異動時的衝擊。
2. 使用 interface 來定義保存資料時會用到的方法。如此實作內容在未來就可視需要改變：
 - Memory-based DAOs：使用記憶體暫存。
 - File-based DAOs：使用檔案儲存。
 - JDBC-based DAOs：使用 JDBC 相關 API 將資料保存於資料庫。
 - Java Persistence API (JPA)-based DAOs：使用 JPA 相關 API 將資料保存於資料庫。

在使用 DAO 設計模式前，Employee 類別是這樣設計：

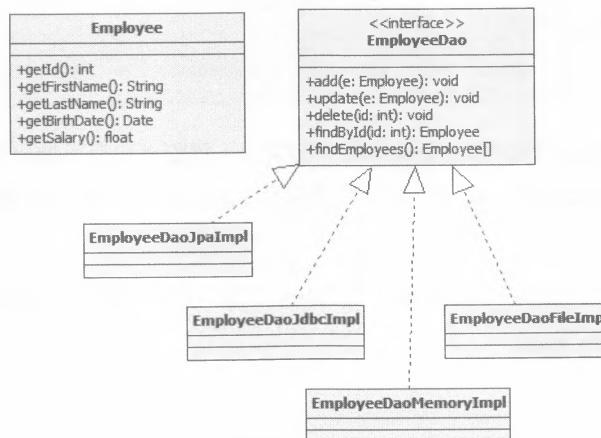
Employee
<pre>+getId(): int +getFirstName(): String +getLastName(): String +getBirthDate(): Date +getSalary(): float +save(): void +delete() +findById(id: int): Employee +findEmployees(): Employee[]</pre>

❖ 圖 5-1 Employee 類別圖，套用 DAO 模式前

這樣的設計方式，最大的問題是「商業邏輯」和「資料保存機制」並存。一旦類別新增商業邏輯欄位，或是修改資料儲存方式，都會導致二個部分同時必須重新測試與驗證的麻煩。

這也違反了物件導向設計原則裡的「單一責任制法則 (Single-Responsibility Principle)，簡稱 SRP」，亦即每個類別的設計都應該具備一個主要責任就好。一個類別肩負的責任太多，會充斥很多方法與變數，讓類別顯得雜亂無章，一旦修改就容易產生 bug。

套用 DAO 設計模式後，原先在 Employee 類別裡和資料儲存有關的方法，都被分離出來；因為這類機制未來有改變的可能，因此抽出 interface，並視需要建立相關實作：



◆ 圖 5-2 Employee 相關的類別圖，套用 DAO 模式後

DAO 設計模式的好處，在於將商業邏輯和資料保存兩大責任分開，也讓資料保存機制有了多樣的選擇。

5.2.3 工廠設計模式的使用契機

因為套用了 DAO 模式，所以我們程式碼裡會這樣宣告並建立資料保存的物件：

範例

```
1 EmployeeDAO dao = new EmployeeDAOMemoryImpl();
```

以 interface 作為宣告型別，好處是實作的類別在有需要時可以隨時抽換。但，若這樣的程式碼出現在程式裡很多地方，如範例「/OCP/src/course/c05/dao/DaoClient.java」，則大量抽換類別其實也是種麻煩：

範例

```

1 public class DaoClient {
2     public void addEmployee(Employee e) {
3         EmployeeDao dao = new EmployeeDaoMemoryImpl();
4         dao.add(e);
5     }
6     public void updateEmployee(Employee e) {
7         EmployeeDao dao = new EmployeeDaoMemoryImpl();
8         dao.update(e);
9     }
10    public void deleteEmployee(Employee e) {
11        EmployeeDao dao = new EmployeeDaoMemoryImpl();
12        dao.delete(e.getId());
13    }
14 }
```

本例只是模擬相同實作出現在不同方法中的情況；實際上也有可能出現在不同類別中。這時候，使用「工廠模式」就可以避免程式碼和 EmployeeDao 的某個實作綁在一起的情況發生。

首先，建立製造 EmployeeDao 物件實例的工廠，如範例「/OCP/src/course/c05/dao/EmployeeDaoFactory.java」：

範例

```

1 public class EmployeeDaoFactory {
2     public static EmployeeDao createEmployeeDao() {
3         return new EmployeeDaoMemoryImpl();
4         // return new EmployeeDaoFileImpl();
5     }
6 }
```

然後，將原本直接使用 new EmployeeDaoMemoryImpl() 的程式碼，如：

範例

```

2     public void addEmployee(Employee e) {
3         EmployeeDao dao = new EmployeeDaoMemoryImpl();
4         dao.add(e);
5     }
```

全部都換成 EmployeeDaoFactory.createEmployeeDao()，如：

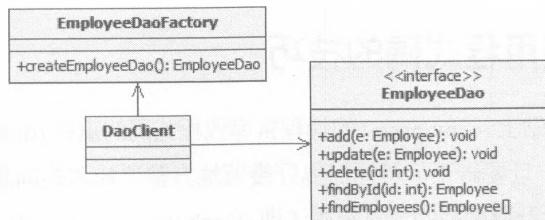
範例

```

2     public void addEmployee(Employee e) {
3         EmployeeDao dao = EmployeeDaoFactory.createEmployeeDao();
4         dao.add(e);
5     }

```

所以，未來若有需要抽換 EmployeeDao 的實作類別，只要修改 EmployeeDaoFactory 類別的 createEmployeeDao() 方法。由原本修改很多地方，縮減成只要修改一個地方即可，這就是工廠模式的不凡之處！UML 的類別圖如下。因為 DaoClient 類別只需要知道 EmployeeDao 界面和 EmployeeDaoFactory 類別，因此改變機率降低很多，程式碼相對穩定。



❖ 圖 5-3 工廠模式的 UML 類別圖

最後，有沒有機會都不用修改 Java 程式碼，就抽換整個系統的 EmployeeDao 實作呢？答案是肯定的。我們可以建立進階 DAO 工廠，使用「Java Reflection」技術，讓「字串 String」來決定選用哪個實作類別，如範例「/OCP/src/course/c05/dao/EmployeeDaoAdvancedFactory.java」：

範例

```

1  public class EmployeeDaoAdvancedFactory {
2      public static EmployeeDao createEmployeeDao() {
3          String name = "course.c05.dao.EmployeeDaoFileImpl";
4          try {
5              Class clazz = Class.forName(name);
6              EmployeeDao dao = (EmployeeDao) clazz.newInstance();
7              return dao;
8          } catch (Exception e) {
9              e.printStackTrace();
10         }
11         return null;
12     }
13 }

```

如果再將該字串改放到設定檔，如 *.properties，或是 *.xml 檔案中，並由 Java 程式讀取該設定檔，取得實作類別的字串，如「course.c05.dao.EmployeeDaoFileImpl」，就可以藉修改設定檔案內容來更改 EmployeeDao 實作類別，達到完全不用修改 Java 程式的境界。這也是目前廣泛應用的一些 Java 框架(Framework)，如 Spring，的主要機制之一，稱為「依賴注入(Dependency Injection)，簡稱 DI」。有興趣再深入這個主題的讀者可自行參閱相關資料。

5.3 使用複合

5.3.1 重複使用程式碼的技巧

程式碼經由複製 / 貼上(copy/paste)的過程常導致程式碼的重複(duplication)，因而造成維護的麻煩。因為一旦要修正，可能面臨好幾個地方都要修改的問題。這種情況，也違反了物件導向程式設計原則裡的「不重複法則(Don't Repeat Yourself)，簡稱 DRY」。

因此，程式碼撰寫的時候，應該儘量「重複使用程式碼(code reuse)」。有幾種常見技巧：

- 把常用的程式碼移到「Library(函式庫)」中。
- 使用繼承。把共用的方法移到父類別中，所有子類別即可共享。
- 使用複合(composition)。藉由引用物件來使用其方法。

其中複合的程式設計技巧，就是本章節說明的重點。

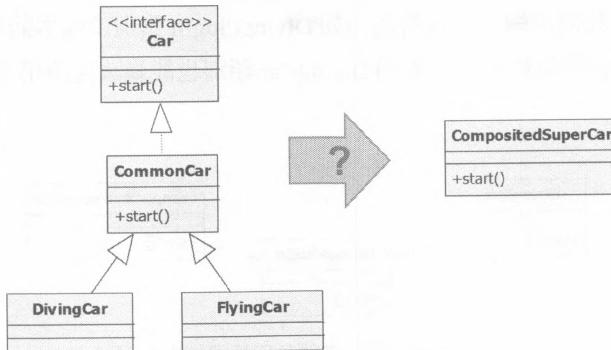
5.3.2 解決複合設計的難題

假設有一個 Car 的介面，實作之後的類別都可以成為車子。首先建立 CommonCar 類別去實作 Car 介面。爾後所有車輛只要繼承 CommonCar 類別，取得 start() 的方法，再加上自己特殊能力，就成了形形色色各式車款，如 DivingCar(潛水車) 和 FlyingCar(飛行車)。

但，如果我們想建立一輛複合功能的車款呢？希望由 CommonCar 中取得 start() 的能力，再同時由 DivingCar(潛水車) 和 FlyingCar(飛行車) 取得潛水和飛行的能力。

我們遇到的難題是：

1. 希望程式碼可以重複使用，而不是直接由相關類別做複製 / 貼上程式碼的動作。
2. Java 只能單一繼承。若複合車款繼承了 DivingCar 類別，就必須放棄繼承 FlyingCar 類別，所以能力只能取得一半。



◆ 圖 5-4 複合功能車款的設計難題

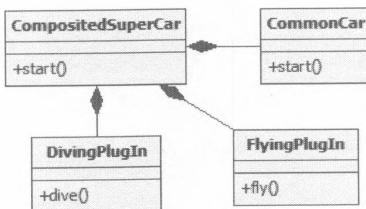
這個時候，我們就可以使用物件導向裡的「複合 (composition)」概念，協助產出複雜功能的物件。

複合的基本觀念是：「藉由納入不同物件，並使用其功能，來壯大自己」。

作法分成二階段：

1. 建立其他功能類別，並參照它們。
2. 建立和所參照物件的方法一樣簽名的方法，並將要實作的功能轉交 (forward) 給所參照的物件去執行。

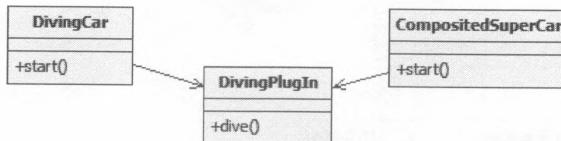
以本例來說，強調類別間複合關係的類別圖會是這樣：



◆ 圖 5-5 使用複合設計概念建立複合功能車款

有幾個要點：

1. 複合關係用實心菱形表示，代表兩者依存關係相當強烈，有唇亡齒寒的意味。菱形端點表示是關係的擁有者。
2. 在這裡，複合車款 ComposedSuperCar 類別所參照的物件來自 DivingPlugIn 類別和 FlyingPlugIn 類別，而非 DivingCar 類別和 FlyingCar 類別。複合車款需要納入特別能力，而非納入其他車輛；一旦將能力如 DivingPlugIn 類別由原本的 DivingCar 類別中抽出，不僅複合車輛能用，原先的 DivingCar 類別也能修改後使用，這也是共用程式碼的好案例：



◆ 圖 5-6 不同車款共用相同能力類別

程式範例為「/OCP/src/course/c05/CompositionDemo.java」：

Q 範例

```

1  interface Car {
2      public void start();
3  }
4  class CommonCar implements Car {
5      public void start() {
6          System.out.println("starting...");
7      }
8  }
9  class DivingPlugIn {
10     public void dive() {
11         System.out.println("diving...");
12     }
13 }
14 class FlyingPlugIn {
15     public void fly() {
16         System.out.println("flying...");
17     }
18 }
19 class ComposedSuperCar implements Car {
20     private CommonCar car = new CommonCar();
21     private DivingPlugIn divingPlugIn = new DivingPlugIn();
22     private FlyingPlugIn flyingPlugIn = new FlyingPlugIn();
  
```

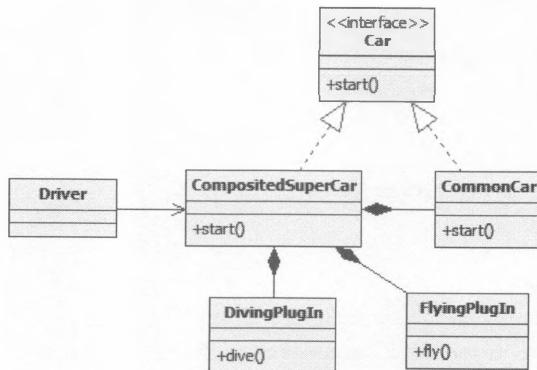
```

23     public void start() {
24         car.start();
25     }
26     public void fly() {
27         flyingPlugIn.fly();
28     }
29     public void dive() {
30         divingPlugIn.dive();
31     }
32 }
33 class Driver {
34     public void testCar(Car c) {
35         if (c instanceof CommonCar) {
36             ((CommonCar)c).start();
37         } else if (c instanceof CompositedSuperCar) {
38             CompositedSuperCar superCar = (CompositedSuperCar)c;
39             superCar.start();
40             superCar.dive();
41             superCar.fly();
42         }
43     }
44 }
45 public class CompositionDemo {
46     public static void main(String args[]) {
47         Car basicCar = new CommonCar();
48         Driver p1 = new Driver();
49         p1.testCar(basicCar);
50         Car superCar = new CompositedSuperCar();
51         Driver p2 = new Driver();
52         p2.testCar(superCar);
53     }
54 }
```

說明

19	定義複合車輛類別。
20	分別建立三個需要使用其功能的物件：
21	private CommonCar car = new CommonCar();
22	private DivingPlugIn divingPlugIn = new DivingPlugIn();
	private FlyingPlugIn flyingPlugIn = new FlyingPlugIn();
23	定義方法 start()，實作內容轉請 car.start() 提供。
24	定義方法 fly()，實作內容轉請 flyingPlugIn.fly() 提供。
25	定義方法 dive()，實作內容轉請 divingPlugIn.dive() 提供。

完成後 UML 類別圖如下：



◆ 圖 5-7 複合車款完整類別圖

5.3.3 方法的委派和轉交

在前述複合車輛的範例裡，我們使用了「複合(composition)」的概念，將類別自己要實作的方法內容，轉手給其他物件的方法代勞。這個轉手的過程，可以稱為「方法委派(method delegation)」或「方法轉交(method forwarding)」。

這兩種描述在多數情況下相通，經常可以交換使用。

若真要細分，「方法轉交(method forwarding)」指建立一個方法，實作內容就是將工作直接轉交給其他物件，自己卻什麼都沒做。

在某些情況，「方法委派(method delegation)」做的事會比較多些，Delegation 也是比較正式的名詞。

5.4 認證考試命題範圍

依據甲骨文公布的考試範圍 (https://education.oracle.com/java-se-8-programmer-ii/pexam_1Z0-809)，和本章相關的主題有：

Advanced Java Class Design

1. Develop code that declares, implements and/or extends **interfaces** and use the **@Override** annotation.
2. Implement inheritance including **visibility modifiers** and **composition**.

本章擬真試題實戰

考題 1

Given these facts about Java types in an application:

- Type x is a template for other types in the application.
- Type x implements doStuff().
- Type x declares, but does NOT implement doIt().
- Type y declares doOther() .

Which three are true?

- A. Type y must be an interface.
- B. Type x must be an abstract class.
- C. Type y must be an abstract class.
- D. Type x could implement or extend from Type y.
- E. Type x could be an abstract class or an interface.
- F. Type y could be an abstract class or an interface.

答案 BDF

說明

1. 由 Type x implements doStuff() , 推測 x 不是 interface 。
2. 由 Type x declares, but does NOT implement doIt() , 推測 x 是 abstract class 或是 interface 。綜合 1 和 2 , x 是 abstract class 。由 Type y declares doOther() , 推測 y 是 abstract class 或是 interface 。所以：選項 A 錯誤，可能是 abstract class 或是 interface 。選項 B 正確。選項 C 錯誤，可能是 abstract class 或是 interface 。選項 D 正確，abstract class 可以 extends 另一個 abstract class ，或是 implements interface 。選項 E 錯誤，x 確定是 abstract class 。選項 F 正確。

考題 2

Given:

```
interface Test{  
    //insert code here  
}
```

Which fragment, inserted in the Test interface, enables the code to compile?

- A. public abstract String typeA;
 public abstract String getTypeA();
- B. public static String typeB;
 public abstract String getTypeB();
- C. public String typeC = "ocjp";
 public static String getTypeC();
- D. public String typeD = "ocjp";
 public abstract String getTypeD();

答案 D

說明 interface 的欄位只能是 public、static 和 final，必須宣告時同時給值(初始化)。interface 的方法只能是 public 和 abstract。

考題 3

Given:

```
interface MyInterface {  
    String type = "MyInterface";  
    public void details();  
}  
class MySuper {  
    static String type = "MySuper ";  
}  
class MySub extends MySuper implements MyInterface {  
    public void details() {  
        System.out.print(type);  
    }  
    public static void main(String[] args) {  
        new MySub().details();  
    }  
}
```

```

        System.out.print(" " + type);
    }
}

```

What is the result?

- A. MyInterface MySuper
- B. MyInterface MyInterface
- C. MySuper MySuper
- D. MySuper MyInterface
- E. Compilation fails

答案 E

說明 介面 MyInterface 的欄位 type 預設是 static，會和類別 MySuper 的 static 欄位 type 混淆，所以程式碼使用時必須載明使用哪個 type 欄位。

考題 4

Which two forms of abstraction can a programmer use in Java?

- A. enums
- B. interfaces
- C. primitives
- D. abstract classes
- E. concrete classes
- F. primitive wrappers

答案 BD

考題 5

Given:

```

interface Writable {
    void doWrite(String w);
}

abstract class Writer implements Writable {
    //codes
}

```

Which two statements are true about the Writer class?

- A. It compiles without any changes.
- B. It compiles if the code `void doWrite(String w);` is added at //codes.
- C. It compiles if the code `void doWrite();` is added at //codes.
- D. It compiles if the code `void doWrite(String w) { }` is added at //codes.
- E. It compiles if the code `void doWrite() { }` is added at //codes.

答案 AE

說明 抽象類別可以不實作介面的方法。

考題 6

Given a description of a Person class:

```
public Person(int id)
public int getId()
public String getContactDetails()
public void setContactDetails(String contactDetails)
public String getName()
public void setName(String name)
public Person getPerson(int id) throws Exception
public void createPerson(Person p) throws Exception
public void deletePerson(int id) throws Exception
public void updatePerson(Person p) throws Exception
```

Which group of method is moved to a new class when implementing the DAO pattern?

A.

```
public int getId ()
public String getContractDetails ()
public void setContractDetails(String contractDetails)
public String getName ()
public void setName (String name)
```

B.

```
public int getId ()
public String getContractDetails()
public String getName()
public Person getPerson(int id) throws Exception
```

C.

```
public void setContractDetails(String contractDetails)
public void setName(String name)
```

D.

```
public Person getPerson(int id) throws Exception
public void createPerson(Person p) throws Exception
public void deletePerson(int id) throws Exception
public void updatePerson(Person p) throws Exception
```

答案 D**說明** 負責物件的 C (create), R (read), U (update), D (delete).

考題 7

Given:

```
interface Car {
    public void launch();
}

class BasicCar implements Car {
    public void launch() {
    }
}

class SuperCar {
    Car c = new BasicCar();
    public void launch() {
        c.launch();
    }
}
```

Which three are true?

- A. BasicCar uses composition.
- B. SuperCar uses composition.
- C. BasicCar is-a Car.
- D. SuperCar is-a Car.
- E. SuperCar takes advantage of polymorphism
- F. BasicCar has-a Car

答案 BCE**說明**

選項 A : BasicCar 實作 Car，所以 BasicCar is-a Car。

選項 B：SuperCar 擁有 Car 欄位，並把自己的 launch() 方法轉交給 Car 欄位執行，所以 SuperCar uses composition。

選項 C：SuperCar 裡的 Car c = new BasicCar();，所以使用了多型 (polymorphism)。

考題 8

Given:

```
interface Glommer {  
}  
  
interface Plinkable {  
}  
  
class Flimmer implements Plinkable {  
    List<Target> t = new ArrayList<Target>();  
}  
  
class Flommer extends Flimmer {  
}  
  
class Target {  
    void dostuff() {  
        String s = "yo";  
    }  
}
```

Which three statements concerning the OO concepts "is-a" and "has-a" are true?

- A. Flimmer is-a Plinkable
- B. Flommer has-a Target
- C. Flommer is-a Glommer
- D. Target has-a String
- E. Flommer is-a Plinkable
- F. Flimmer is-a Flommer
- G. Target is-a Plinkable

答案 ABE

說明 選項 C：沒有類別實作 Glommer。選項 D：區域變數不納入 has-a 的範圍。選項 F：角色應該相反。選項 G：沒有實作或繼承關係。

考題 9

Which two compile?

A.

```
interface CompilableA {
    void compile();
}
```

B.

```
interface CompilableB {
    final void compile();
}
```

C.

```
interface CompilableC {
    static void compile();
}
```

D.

```
interface CompilableD {
    abstract void compile();
}
```

E.

```
interface CompilableE {
    protected abstract void compile();
}
```

答案 AD

說明 介面的方法只能是 public 且 abstract。

考題 10

Given a definition of class Customer:

```
class Customer {
    private int id;
    private String name;
    public int getId();
    public String getName();
    public boolean add(Customer c);
    public void delete(int id);
    public Customer find(int id);
    public boolean update(Customer c);
}
```

What two changes should you make to apply the DAO pattern to this class?

- A. Make the Customer class abstract.
- B. Make the Customer class an interface.
- C. Move add, delete, find, and update methods into their own implementation class.
- D. Create an interface that defines the signatures of the add, delete, find, and update methods.
- E. Make add, delete, and find, and update methods private for encapsulation.
- F. Make the getName and getId methods private for encapsulation.

答案 CD

說明 DAO 模式必須負責物件的 C(create/add)、R(read/find)、U(update)、D(delete)。先建立一個 interface，再依需求提供實作。

考題 11

Which is a key aspect of composition?

- A. Using inheritance
- B. Method delegation
- C. Creating abstract classes
- D. Implementing the composite interface

答案 B

說明 參照本章內容：「5.3 使用複合」。

考題 12

Given:

```
class Student {  
    public int listEmails() { }  
    public void sendEmail(String email) { }  
    public Boolean validateEmail() { }  
    public void print(String email) { }  
}
```

Which is correct?

- A. Student takes advantage of composition.
- B. Student "has-an" Email.
- C. Student "is-a" LetterPrinter.
- D. Student has low cohesion.

答案 D

說明 「cohesion」是指類別內邏輯設計的內聚性。內聚性愈高，象徵依賴其他類別的程度降低；而且程式碼因集中而好管理、好維護、容易測試。本類別看得出和 email 的關聯性，但方法太多太分散，故內聚性較弱。

考題 13

Given:

```
class EmployeeApplication {
    public static void main(String[] args) {
        EmployeeDAO empDAO = new EmployeeDAOMemoryImpl(); //line1
        //other codes..
    }
}
```

Which two valid alternatives to //line1 would decouple this application from a specific implementation of EmployeeDAO?

- A. EmployeeDAO empDAO = EmployeeDAO();
- B. EmployeeDAO empDAO = (EmployeeDAO) new Object();
- C. EmployeeDAO empDAO = EmployeeDAO.getInstance();
- D. EmployeeDAO empDAO = (EmployeeDAO) new EmployeeDAOMemoryImpl();
- E. EmployeeDAO empDAO = EmployeeDAOFactory.getInstance();

答案 CE

說明 「decouple this application from a specific implementation of EmployeeDAO」是指「讓程式不要和某一 EmployeeDAO 的實作綁在一起」，本例中指的是 EmployeeDAOMemoryImpl。解決方案就是使用工廠方法，如 getInstance()，詳見本章「5.2.3 工廠設計模式的使用契機」。

考題 14

Which two methods of code reuse that aggregate the features located in multiple classes are?

- A. Inheritance
- B. Copy and Paste
- C. Composition
- D. Refactoring
- E. Virtual Method Invocation

答案 AC

說明 使用繼承 (Inheritance) 和複合 (Composition) 可以將不同類別上的功能聚合起來，達到程式碼重複使用 (code reuse) 的目的。

考題 15

Which represents part of a DAO design pattern?

A.

```
interface MemberDAO {  
    int getID();  
    Student findByID(int id);  
    void update();  
    void delete();  
}
```

B.

```
class MemberDAO {  
    int getID() {  
        return 0;  
    }  
    Student findByID(int id) {  
        return null;  
    }  
    void update() {}  
    void delete() {}  
}
```

C.

```
class MemberDAO {  
    void create(Student e) {}  
    void update(Student e) {}  
}
```

```

void delete(int id) { }
Student findByID(int id) {
    return null;
}
}

```

D.

```

interface MemberDAO {
    void create(Student e);
    void update(Student e);
    void delete(int id);
    Student findByID(int id);
}

```

E.

```

interface MemberDAO {
    void create(Connection c, Student e);
    void update(Connection c, Student e);
    void delete(Connection c, int id);
    Student findByID(Connection c, int id);
}

```

答案 D

說明 DAO design pattern 的設計精神是：

1. 使用 interface。
2. DAO 模式必須負責物件的 C(create/add)、R(read/find)、U(update)、D(delete)。
3. DAO 的 interface 的實作應該要可以抽換為檔案、陣列、資料庫等。選項 E 已經綁定資料庫(方法參數皆必須輸入 Connection 物件)，失去多型的意義。

考題 16

What are two benefits of a Factory design pattern?

- A. Eliminates direct constructor calls in favor of invoking a method
- B. Provides a mechanism to monitor objects for changes
- C. Eliminates the need to overload constructors in a class implementation
- D. Prevents the compiler from complaining about abstract method signatures
- E. Prevents tight coupling between your application and a class implementation

答案 AE

說明 工廠模式的目的是：

1. 使用工廠方法取代使用 new 呼叫建構子，故選項 A。
2. 避免程式綁定某個特殊實作，故選項 E。

考題 17

Suppose you want to create a singleton class by using the Singleton design pattern.

Which two statements ensure the nature of the design is singleton?

- A. Make the class static.
- B. Make the constructor private.
- C. Override equals() and hashCode() methods of the java.lang.Object class.
- D. Use a static reference to point to the single instance.
- E. Implement the Serializable interface.

答案 BD

考題 18

What are benefits of polymorphism (choose two)?

- A. Faster code at runtime
- B. More efficient code at runtime
- C. More dynamic code at runtime
- D. More flexible and reusable code
- E. Code that is protected from extension by other classes

答案 CD

泛型和集合物件

-
- | 6.1 泛型
 - | 6.2 集合物件
 - | 6.3 Map
 - | 6.4 集合物件成員的排序
 - | 6.5 認證考試命題範圍

6.1 泛型(Generics)

Java 有嚴格的型別限制，當成員或方法參數一旦決定一種型別，就只能使用該種型別的資料。Java 5 後加入了泛型的特性，使型別的使用可以具有不同於多型使用的另一種彈性。

Java 的集合物件用來裝填其他物件，搭配泛型後可以限制裝填物件的型別，因此和泛型有密不可分的關係，所以合併在本章介紹。

使用泛型設計的效益

Java 5 後加入了泛型的特性，主要目的是：

1. 提供更彈性的「型別安全 (type safety)」檢查機制。過去執行時期才能知道的型別錯誤，使用泛型後在編譯時期就可以預先發現。
2. 在集合物件 (Collections) 中大量使用，可限制內含物件的型別。
3. 減少轉型 (casting) 的需要，讓程式碼更簡潔。

以泛型技巧設計類別

有兩段雷同的程式碼：

範例一：

Q 範例

```

1  public class UseString {
2      private String message = "";
3      public void add(String message) {
4          this.message = message;
5      }
6      public String get() {
7          return this.message;
8      }
9  }
```

範例二：

Q 範例

```

1  public class UseShirt {
2      private Shirt shirt;
3      public void add(Shirt shirt) {
4          this.shirt = shirt;
5      }
}
```

```

6     public Shirt get() {
7         return this.shirt;
8     }
9 }
```

仔細分析，除了類別名稱和變數名稱不同之外，剩下的差別就是類別內使用的變數的型別。所以，我們可以用以下的程式碼，表現上述兩類別：

範例

```

1 public class UseAny <T> {
2     private T t;
3     public void add(T t) {
4         this.t = t;
5     }
6     public T get() {
7         return this.t;
8     }
9 }
```

如果將程式碼裡使用的符號「T」，置換為「String」，就會是範例一的類別；同樣的情況，若置換為「Shirt」，就會是範例二的類別。依此類推，符號「T」可以「廣泛」被各種參考「型別」取代，這就是泛型設計的概念。

在類別裡用一個「一般化(Generic)的符號」，表示未來該符號可以是任何參考型別；並在宣告類別的時候，加上「< T >」的記號。菱形符號「< >」表示使用泛型宣告，裡面的「T」則表示類別裡的若使用符號「T」，都代表一個可置換的「Type(型別)」。

常見的符號及表示方式，有以下幾種：

- T : Type
- E : Element
- K : Key
- V : Value

本章節中會逐一介紹。

使用泛型設計的類別

完成泛型的類別設計 UseAny <T> 後，可以用以下方式建立物件實例：

範例

```

1 public static void main(String args[]) {
2     // 使用一般方式：
3     UseShirt shirt1= new UseShirt();
4     UseString msg1 = new UseString();
5     msg1.add("test generic");
6     // 使用泛型：
7     UseAny<Shirt> shirt2 = new UseAny<Shirt>();
8     UseAny<String> msg2 = new UseAny<String>();
9     msg2.add("test generic");
10 }
```

由 Java 7 開始，取消了「參考型別」和「建構子」都必須在 $\langle \rangle$ (菱形符號) 內加上置換型別的規定，允許在等號右側直接使用空的菱形符號 $\langle \rangle$ ，因為後者其實可以由前者推斷 (inference) 而知。因此，使用泛型的類別的程式碼可以再簡化為：

 範例

```

7     UseAny<Shirt> shirt2 = new UseAny<>();
8     UseAny<String> msg2 = new UseAny<>();
```

6.2 集合物件(Collections)

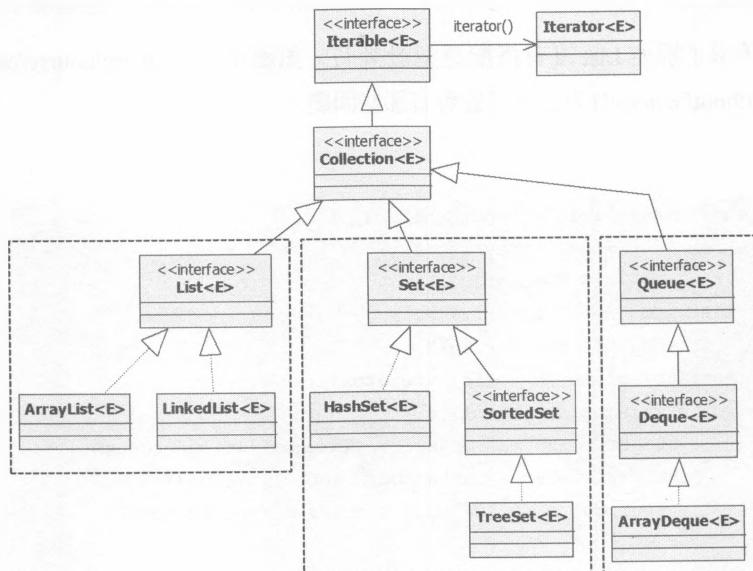
6.2.1 集合物件的定義和種類

Java 使用「集合物件 (Collection)」來裝載及管理群組物件，概念上類似陣列，但相較陣列具備更多管理功能。有幾個特點：

1. 以介面 Collection 為代表。
2. 集合內的物件稱為「elements」，簡寫 E。
3. 集合內的物件不可以是基本型別，若有需要可使用基本型別的包裹類別 (wrapper class)。
4. 有多種常見的資料結構，如 stack、queue、dynamic array 等。
5. 大量使用泛型 (generic)。
6. 都屬於 java.util.* 套件。

7. Collection 介面繼承了 Iterable 介面，因此所有集合物件都具備使用疊代器 (Iterator) 的能力。集合物件雖然種類很多，一旦取得集合物件的疊代器，就可以使用疊代器以相同的方式走訪所有成員，和集合物件種類無關。

由 Collection 家族類別圖可以了解集合物件的種類：



❖ 圖 6-1 Collection 家族類別圖

6.2.2 List

`List` 介面是集合物件裡最常使用的一種。因為具備 `index`，因此成員可以依照放入的先後來區分「順序 (order)」。常用的方法有：

1. 新增成員時使用 `index` 指定插入位置。
2. 新增成員時直接加到尾端。
3. 取得成員的 `index`。
4. 使用 `index` 移除或覆寫成員。
5. 取得 `List` 長度。

`ArrayList` 類別是 `List` 裡最常使用的一種實作類別。依照多型的概念，使用 `ArrayList` 類別時應該用 `List` 介面宣告以方便未來抽換成其他實作類別。它的特色是：

1. 行為和陣列 (array) 相近，但因為長度可以自動成長，又稱為「動態陣列 (dynamically array)」。
2. 使用 index 進行成員新增、存取、修改。
3. 允許重複成員，因為可以使用 index 區分。

之前說過集合物件的特色之一是可以結合泛型設計，接下來會作一些這個主題的探討。

首先，必須了解若 List 沒有搭配泛型設計時，如範例「/OCP/src/course/c06/TestList.java」的 withoutGeneric() 方法，可能會有那些問題：

範例

```

1  private static void withoutGeneric() {
2      List list = new ArrayList(2);
3      list.add( new Integer(1) );
4      list.add( new Integer(2) );
5      list.add( "I am a string!" );
6      Iterator elements = list.iterator();
7      while (elements.hasNext()) {
8          Integer partNumberObject = (Integer) (elements.next());
9          int partNumber = partNumberObject.intValue();
10         System.out.println("Part number: " + partNumber);
11     }
12 }
```

這個範例程式的重點是：

1. 在行 6、7、8 表現了疊代器 (Iterator) 的使用方式。分別是：

行 6：集合物件 list 呼叫 iterator() 方法取得疊代器 elements。

行 7：疊代器可以藉由 hasNext() 方法確認所有成員是否走訪完畢。

行 8：疊代器可以藉由 next() 方法取得目前走訪的成員。

2. List 在未使用泛型的時候，預設將所有成員都視為 Object 類別。因此行 3、4 可以放入 Integer，行 5 可以放入 String 類別。
3. 承上，所以取出的物件都是 Object，都必須進行轉型，才能回復原先型別。
4. 承上，轉型時若發現型別不一致的成員，如行 5 放入 String，行 8 要轉型成 Integer，就會拋出 ClassCastException！

將上面敘述予以歸納。若未使用泛型：

1. 取出的物件必須轉型。
2. 錯放成員時，執行時期才能知道。

接下來，範例「/OCP/src/course/c06/TestList.java」的 withGeneric() 方法，解決了上述難題：

範例

```

1  private static void withGeneric() {
2      List<Integer> list = new ArrayList<>(2);
3      list.add( new Integer(1) );
4      list.add( new Integer(2) );
5      // list.add( "I am a string!" ); //Compile Error!!
6      Iterator<Integer> elements = list.iterator();
7      while (elements.hasNext()) {
8          Integer partNumberObject = elements.next();
9          int partNumber = partNumberObject.intValue();
10         System.out.println("Part number: " + partNumber);
11     }
12 }
```

說明

2	使用泛型宣告：List<Integer>，該 List 只能放 Integer 物件。
5	因為放入 String，無法通過編譯。
6	疊代器也支援泛型宣告：Iterator<Integer>。
8	由疊代器中取出目前走訪成員，因為都是 Integer，不需要再轉型。

因此，集合物件使用泛型的好處是：

1. 放錯成員時，編譯時期就可以檢測到，更落實「型別安全 (type safety)」的設計。
2. 取出成員時，不需要轉型。

6.2.3 自動裝箱 (Boxing) 和開箱 (Unboxing)

Java 是物件導向的程式語言，以參考型別為主；在 java.lang 的套件下，建立一套和八種基本型別對應的八個參考型別，提供了基本型別常見的加減乘除四則運算、比較等操作的替代方案。因為特色在於將各自對應的基本型別視為核心，將之以物件型態「包裹」，也稱為基本型別的「包裹類別 (wrapper class)」。對應關係為：

❖ 表 6-1 基本型別和包裹類別對照表

基本型別	包裹類別	父類別
Byte	Byte	Number
Short	Short	
Int	Integer	
Long	Long	
Float	Float	
Double	Double	
Char	Character	Object
Boolean	Boolean	Object

可以發現命名規則除了類別 Integer 和 Character 較特別外，其餘都是將基本型別的名稱，改第一個單字為大寫，就成為包裹類別的名稱。

為了讓基本型別和包裹類別兩者更相容，Java 允許兩者可以互相自動轉換：

1. 基本型別 → 包裹類別，稱為「裝箱 (boxing)」，像是把基本型別裝進包裹類別的箱子裡。
2. 包裹類別 → 基本型別，稱為「開箱 (unboxing)」，像是把基本型別由包裹類別的箱子裡取出。

必須提醒的是，因為自動發生，所以相當方便。但會有些微效能耗損，迴圈內儘量別使用。

最後，若使用自動開箱的機制，還可以將前述範例的：

❶ 範例

```
8     Integer partNumberObject = elements.next();
9     int partNumber = partNumberObject.intValue();
```

合併只剩一行程式碼：

❷ 範例

```
8     int partNumber = elements.next();
```

6.2.4 Set

Set 介面也是 Collection 物件的一種，特色是：

1. 成員必須是獨一無二 (unique)，不能重複。

2. 沒有 index。
3. 試圖放入重複成員，不會出錯，但無效。
4. 常使用 HashSet 實作類別。TreeSet 類別會依物件特性自動排序。
5. 成員是否唯一，或是排序的先後，則取決於其方法 equals() 和 hashCode() 的覆寫結果。

範例「/OCP/src/course/c06/TestSet.java」顯示 Set 物件的使用方式：

範例

```

1 public class TestSet {
2     public static void main(String[] args) {
3         Set<String> set = new HashSet<>();
4         set.add("one");
5         set.add("two");
6         set.add("three");
7         set.add("three"); // 重複，將不會加入，也不會執行錯誤
8         for (String item : set) {
9             System.out.println("Item: " + item);
10        }
11    }
12 }
```

結果

```

Item: two
Item: one
Item: three
```

若將行 3 改為：

範例

```
3         Set<String> set = new TreeSet<>();
```

則結果為：

結果

```

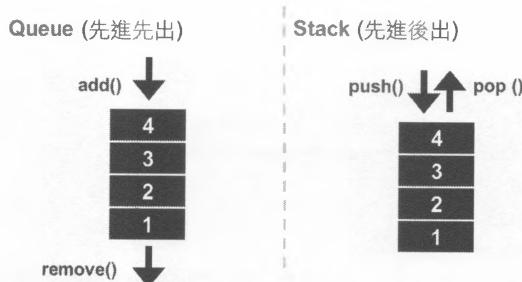
Item: one
Item: three
Item: two
```

會依字串順序排序。

6.2.5 Deque

Deque 介面繼承了 Queue 介面，特色是：

1. 為「Double-Ended Queue」，亦即具備兩端點的 Queue，發音同「deck」。
2. 可同時用於「Stack」和「Queue」兩種資料結構，只要呼叫不同的方法。如下：



❖ 圖 6-2 Queue 和 Stack 的成員存取比較

因此，當使用 Deque 介面的 add() 和 remove() 方法時，Deque 物件表現出 Queue 資料結構的行為模式，亦即成員是「先進先出」。

若改使用 push() 和 pop() 方法，Deque 物件表現出 Stack 資料結構的行為模式，亦即成員是「先進後出」。如範例「/OCP/src/course/c06/TestDeque.java」：

範例

```

1  public class TestDeque {
2      public static void testStack(Deque<String> stack) {
3          stack.push("one");
4          stack.push("two");
5          stack.push("three");
6          System.out.println(stack.pop());
7          System.out.println(stack.pop());
8          System.out.println(stack.pop());
9      }
10     public static void testQueue(Deque<String> queue) {
11         queue.add("one");
12         queue.add("two");
13         queue.add("three");
14         int size = queue.size() - 1;
15         while (size >= 0) {
16             System.out.println(queue.remove());
17             size--;
18         }

```

```

19     }
20     public static void main(String[] args) {
21         Deque<String> deque = new ArrayDeque<>();
22         System.out.println("---- Stack Out ---");
23         testStack(deque);
24         System.out.println("---- Queue Out ---");
25         testQueue(deque);
26     }
27 }
```

結果

```

---- Stack Out ---
Three
Two
One
---- Queue Out ---
One
Two
Three
```

6.3 Map

Map 是「key-value」的集合，但不屬於 Collection 集合物件家族。每個成員都是「key 物件和 value 物件的配對組合」，其中：

- Key 物件：用來尋找 value 物件，因此每個 key 物件都是獨特 (Unique) 而不重複的。
- Value 物件：和 key 物件有著關聯性 (associative)。

例如，某 Map 物件儲存三組成對的「key-value」，如下：

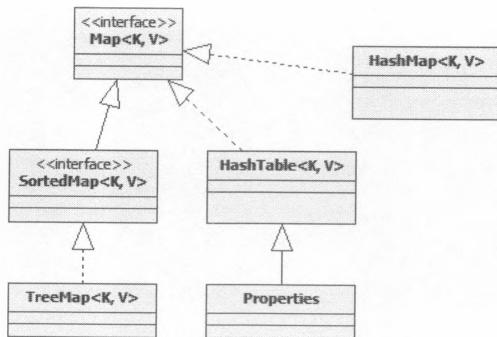
key (身分證字號)	value (人)
A333333333	張三
B444444444	李四
C555555555	王五

本例的 key 是身分證字號，value 則是對應的人。在 Map 中，只要提供「key 物件」，就可以取得對應的「value 物件」。例如當 key 值為 C555555555 時，透過該 Map 就可以取得王五物件。

在其他語言，Map 或稱為「關聯性陣列 (associative arrays)」。以上例而言，所有 key 物件可以構成一個陣列，所有的 value 物件構成第二個陣列，二個陣列有著關聯性，故名。

回顧 List 物件。List 物件是由 index 來找出對應的值，概念上可以 List 看作是把 index 當成 key 值的 Map。而實際上，除了 List 和 Map 都在定義 `java.util.*` 套件中外，兩者並沒有直接的關係。

Map 沒有繼承於 Collection 介面，自立門戶成為 Map 家族：



◆ 圖 6-3 Map 家族類別圖

Map 使用的泛型宣告是「`<K, V>`」。K 表示 key，V 表示 value。

每個分支的代表類別有：

1. TreeMap：特色是 keys 自動排序。
2. HashTable：特色是「執行緒安全¹」且「keys 和 values 不允許為 null」。
3. HashMap：特色是「非執行緒安全」且「keys 和 values 可為 null」。

課堂小秘訣

比較 Set 和 Map 家族，兩者都有可以支援排序的分支：

1. 分支起源都是以 Sorted 命名開頭的介面，如 SortedMap 和 SortedSet。
2. 實作類別都是以 Tree 命名開頭，如 TreeMap 和 TreeSet。
3. 使用方式為：

```
SortedMap map = new TreeMap();
SortedSet set = new TreeSet();
```

¹ 註 1 執行緒安全 (thread-safe) 的話題將在本書後段有專章介紹。大體說來，Java 的程式是由「執行緒」負責執行，而「執行緒安全」是指一個物件被一個執行緒使用和同時被多個執行緒使用時的結果都一致，不會因為多個執行緒而產生奇怪或錯誤的結果。

以下範例顯示 Map 的基本用法及注意事項：

❶ 範例「/OCP/src/course/c06/TestMap.java」：

```

1  public class TestMap {
2      public static void main(String[] args) {
3          Map<String, String> partList = new TreeMap<>();
4          partList.put("A02", "Edwin");
5          partList.put("A01", "Jason");
6          partList.put("A03", "Sonic");
7          partList.put("A03", "Howard"); // Overwrite value
8          // print all values
9          Collection<String> values = partList.values();
10         for (String v : values) {
11             System.out.println(v);
12         }
13         // print all keys & values
14         Set<String> keys = partList.keySet();
15         for (String key : keys) {
16             System.out.println("#" + key + ":" + partList.get(key));
17         }
18     }
19 }
```

❷ 說明

3	使用泛型宣告： <code><String, String></code> ，key 是字串，value 也是字串。 Map 的實作類別選用 <code>TreeMap</code> ，所以可以針對 key 值做排序。
7	放入 Map 的 key-value 對，若 key 和已存在的 key 重複，將覆蓋原先內容。
9	Map 的 <code>values()</code> 方法回傳 <code>Collection</code> 物件，可以取得所有 values 的集合。
14	Map 的 <code>keySet()</code> 方法回傳 <code>Set</code> 物件，可以取得所有 keys 的集合。因為 key 不能重複，符合 <code>Set</code> 集合的要求，因此回傳 <code>Set</code> 而非 <code>Collection</code> 。
16	Map 的 <code>get(key)</code> 方法需要傳入 key，可以取得 value。

❸ 結果

```

Jason
Edwin
Howard
#A01: Jason
#A02: Edwin
#A03: Howard
```

6.4 集合物件成員的排序

6.4.1 排序的做法

Set 和 Map 家族的成員，若使用 TreeSet 和 TreeMap 的實作類別，都具有排序的功能。但有二個進階的問題值得我們思考：

1. 先前範例裡的成員，都是數字或字串，這類型的型別都有預設的順序。但若特殊物件，如類別 Shirt、Employee 等，該如何定義順序？有無可能一個類別，可以定義多種排序標準？
2. 若是 List 家族，該如何重新排序？

針對這些問題，Java 提供兩個介面，供我們選擇：

1. Comparable 介面，必須實作 compareTo() 方法。
2. Comparator 介面，必須實作 compare() 方法。

分別在稍後的兩個小節做介紹。

兩個介面要實作的方法，無論是 compareTo() 或 compare() 方法，都回傳一個「整數」表示比較結果：

1. 若回傳整數：「= 0」，表示兩者相等。
2. 若回傳整數：「< 0」，表示「自己(this)」小於(數值上)或早於(順序上)「傳進來的物件」。
3. 若回傳整數：「> 0」，表示「自己(this)」大於(數值上)或晚於(順序上)「傳進來的物件」。

範例「/OCP/src/course/c06/compare/TestOrder.java」顯示該回傳結果的表示意涵：

範例

```

1  public class TestOrder {
2      public static void main(String[] args) {
3          Calendar today = Calendar.getInstance();
4          Calendar tomorrow = Calendar.getInstance();
5          tomorrow.add(Calendar.DATE, 1);
6          out.println(today.compareTo(tomorrow));
7          out.println("A".compareTo("B"));
8          out.println(new Integer(5).compareTo(new Integer(6)));

```

```

9     }
10    }

```

結果

```

-1
-1
-1

```

說明

6	因為在時間順序上，today 早於 tomorrow，所以回傳 -1。
7	因為在字母順序上，"A" 早於 "B"，所以回傳 -1。
8	因為在數值大小上，5 小於 6，所以回傳 -1。 因為基本型別無方法可使用，故先轉成包裹類別。

6.4.2 使用 Comparable 介面排序

Comparable 介面內容主要為：

```

public interface Comparable<T> {
    public int compareTo(T o);
}

```

特色是：

- 支援泛型設計。
- 必須實作 compareTo() 方法，比較「自己 (this)」和「傳進來的物件」。
- 單一 class 只能實作一次，所以只能提供單一方式的排序。可用於 TreeSet 和 TreeMap 等實作類別，或需要物件之間比較的地方。

假設有一個 Student 類別定義如下：

範例

```

1  public class Student {
2      private String name;
3      private long id;
4      private double score;
5      public Student(String name, long id, double score) {
6          this.name = name;
7          this.id = id;
8          this.score = score;

```

```

9      }
10     public String getName() {
11         return this.name;
12     }
13     public double getScore() {
14         return this.score;
15     }
16     public String toString() {
17         return this.name + "\t" + this.id + "\t" + this.score;
18     }
19 }
```

在開始排序之前，要先問自己，哪一個條件決定 Student 物件的排序先後？是 name ？還是 id ？還是 score ？還是複合條件？決定之後，讓 Student 類別去實作 Comparable 介面：

- 必須提供 compareTo() 方法的內容。
- 只能提供一種排序選擇。

結果為如範例「/OCP/src/course/c06/compare/Student.java」：

範例

```

1  public class Student implements Comparable<Student> {
2      private String name;
3      private long id;
4      private double score;
5      public Student(String name, long id, double score) {
6          this.name = name;
7          this.id = id;
8          this.score = score;
9      }
10     public String getName() {
11         return this.name;
12     }
13     public double getScore() {
14         return this.score;
15     }
16     public String toString() {
17         return this.name + "\t" + this.id + "\t" + this.score;
18     }
19     @Override
20     public int compareTo(Student s) {
21         // use method dedication
22         int sortById = new Long(this.id).compareTo(new Long(s.id));
23         int sortByName = this.name.compareTo(s.getName());
```

```

24     int sortByScore =
25         new Double(this.score).compareTo(new Double(s.score));
26     return sortById;
27 }

```

說明

- | | |
|----|--|
| 22 | sortById 是使用自己的 id 欄位，和傳入物件的 id 欄位的比較結果。 |
| 23 | 承上，sortByName 是兩者的 name 欄位的比較結果。 |
| 24 | 承上，sortByScore 是兩者的 score 欄位的比較結果。 |
| 25 | 選擇使用 id 欄位的比較結果作為物件比較結果，也可以選用其他。 |

實作 Comparable 介面，關鍵是提供的 compareTo() 方法內容。我們可以依照回傳結果是「= 0」、「< 0」或「> 0」來表現比較結果。也可以使用前一章節介紹的「方法委派 (method delegation)」或「方法轉交 (method forwarding)」的概念，直接以某欄位的比較結果作為物件比較結果，也是本範例採用的方式。

接下來，驗證實作 Comparable 介面的 Student 類別集合的排序結果。

範例「/OCP/src/course/c06/compare/TestComparable.java」：

範例

```

1  public class TestComparable {
2      public static void main(String[] args) {
3          System.out.println("..... Before Sort .....");
4          Set<Student> studentList = new HashSet<>();
5          studentList.add(new Student("John", 2, 3.9));
6          studentList.add(new Student("Thomas", 1, 3.8));
7          studentList.add(new Student("George", 3, 3.4));
8          for (Student student : studentList) {
9              System.out.println(student);
10         }
11         System.out.println("..... After Sort .....");
12         Set<Student> sortedStudentList = new TreeSet<>();
13         sortedStudentList.add(new Student("John", 2, 3.9));
14         sortedStudentList.add(new Student("Thomas", 1, 3.8));
15         sortedStudentList.add(new Student("George", 3, 3.4));
16         for (Student student : sortedStudentList) {
17             System.out.println(student);
18         }
19     }
20 }

```

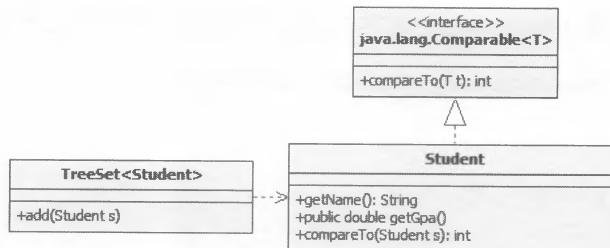
結果

```
----- Before Sort -----
John          2          3.9
Thomas        1          3.8
George        3          3.4
----- After Sort -----
Thomas        1          3.8
John          2          3.9
George        3          3.4
```

說明

- | | |
|----|--|
| 4 | 使用 Set 介面的 HashSet 類別，沒有排序功能。 |
| 12 | 使用 Set 介面的 TreeSet 類別，有排序功能，依 Student 類別提供的 compareTo() 方法內容決定；本例將比較結果轉由欄位 id 的比較結果決定。 |

UML 圖示：



◆ 圖 6-4 Student 實作 Comparable 類別圖

6.4.3 使用 Comparator 介面排序

實作 Comparable 介面的 Student 類別只有提供一次 compareTo() 方法內容的機會，因此物件只有一種排序能力。使用 Comparator 介面則可以提供多種選擇：

```
public interface Comparator<T> {
    int compare(T o1, T o2);
}
```

特色為：

- 支援泛型設計。
- 必須實作 compare() 方法，用以比較「第一個參數物件」和「第二個參數物件」。

3. 可以藉由提供多種 Comparator 類別，達成多種排序方式；主要用於搭配以下方法，可以幫助 List 成員排序：

```
Collections.sort(List<T> list, Comparator<? super T> c)
```

範例「/OCP/src/course/c06/compare/TestComparator.java」：

範例

```

1  class NameSorter implements Comparator<Student> {
2      public int compare(Student s1, Student s2) {
3          return s1.getName().compareTo(s2.getName());
4      }
5  }
6  class ScoreSorter implements Comparator<Student> {
7      public int compare(Student s1, Student s2) {
8          return new Double(s1.getScore()).compareTo(s2.getScore());
9      }
10 }
11 public class TestComparator {
12     private static void showList(List<Student> studentList) {
13         for (int i=0; i<studentList.size(); i++) {
14             System.out.println("index#" + i + ": " + studentList.get(i));
15         }
16     }
17     public static void main(String[] args) {
18         List<Student> studentList = new ArrayList<>(3);
19         studentList.add(new Student("Thomas", 1, 3.8));
20         studentList.add(new Student("John", 2, 3.9));
21         studentList.add(new Student("George", 3, 3.4));
22
23         System.out.println("\n--- Original ----- ");
24         showList(studentList);
25
26         System.out.println("\n--- Sort by name ----- ");
27         Comparator<Student> sortName = new NameSorter();
28         Collections.sort(studentList, sortName);
29         showList(studentList);
30
31         System.out.println("\n--- Sort by score ----- ");
32         Comparator<Student> sortScore = new ScoreSorter();
33         Collections.sort(studentList, sortScore);
34         showList(studentList);
35     }
36 }
```

結果

```
--- Original -----
index#0: Thomas    1    3.8
index#1: John       2    3.9
index#2: George     3    3.4

--- Sort by name -----
index#0: George     3    3.4
index#1: John       2    3.9
index#2: Thomas     1    3.8

--- Sort by score -----
index#0: George     3    3.4
index#1: Thomas     1    3.8
index#2: John       2    3.9
```

說明

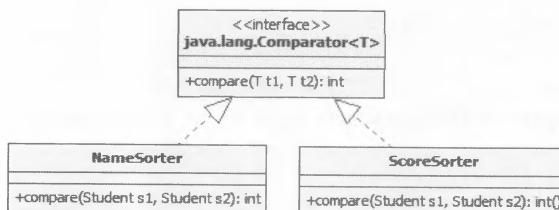
1	建立以 Student 類別的 name 欄位作為排序關鍵的 NameSorter 類別。
6	建立以 Student 類別的 score 欄位作為排序關鍵的 ScoreSorter 類別。
28	Collections.sort() 方法傳入 List 物件和 NameSorter 物件。
33	Collections.sort() 方法傳入 List 物件和 ScoreSorter 物件。

由上例可以發現，List 物件經過方法

```
Collections.sort(List<T> list, Comparator<? super T> c)
```

的排序後，原先的 index 和成員的對應關係也會被更改。

UML 圖示：



❖ 圖 6-5 實作 Comparator 介面的類別圖

6.5 認證考試命題範圍

依據甲骨文公布的考試範圍 (https://education.oracle.com/java-se-8-programmer-ii/pexam_1Z0-809)，和本章相關的主題有：

Generics and Collections

1. Create and use a **generic** class.
2. Create and use **ArrayList**, **TreeSet**, **TreeMap**, and **ArrayDeque** objects.
3. Use **java.util.Comparator** and **java.lang.Comparable** interfaces.

本章擬真試題實戰

考題 1

Given:

```
public class Test {  
    static final Comparator<Integer> MyComparator =  
        new Comparator<Integer>() {  
            public int compare(Integer n1, Integer n2) {  
                return n2.compareTo(n1);  
            }  
    };  
    public static void main(String[] args) {  
        ArrayList<Integer> list = new ArrayList<>();  
        list.add(4);  
        list.add(1);  
        list.add(3);  
        list.add(2);  
        Collections.sort(list, null);  
        System.out.print(Collections.binarySearch(list, 3) + " ");  
        Collections.sort(list, MyComparator);  
        System.out.print(Collections.binarySearch(list, 3));  
    }  
}
```

What is the result?

- A. 4 1
- B. 1 2
- C. 3 2
- D. 2 1
- E. 2 3

答案 D

說明 使用 Collections.binarySearch() 傳入 List 和 List 某成員，可以得到該成員的 index(由 0 起算)。

1. Collections.sort() 傳入要排序的 List 和 Comparator。若 Comparator 是 null，等同於不排序，得到所有成員順序是 [4, 1, 3, 2]，成員 3 的 index 為 2。
2. MyComparator 的 compare() 方法讓 Integer 使用降冪排序(由大到小)，得到所有成員順序是 [4, 3, 2, 1]，成員 3 的 index 為 1。

考題 2

Which statement declares a generic class?

- A. public class Test <T> { }
- B. public class <Test> { }
- C. public class Test <> { }
- D. public class Test (Generic) { }
- E. public class Test (G) { }
- F. public class Test { }

答案 A

考題 3

Given:

```
public static void main(String[] args) {
    Deque<String> d = new ArrayDeque<>();
    d.push("One");
    d.push("Two");
    d.push("Three");
    System.out.println(d.pop());
}
```

What is the result?

- A. Three
- B. One
- C. Compilation fails.
- D. The program runs, but prints no output.

答案 A

說明 使用 push() 和 pop() 方法時，Deque 物件表現出 Stack 資料結構的行為模式，亦即成員是「先進後出」。參考本章「6.2.5 Deque」內容。

考題 4

Given:

```
public static void main(String[] args) {  
    Deque<String> d = new ArrayDeque<>();  
    d.add("One");  
    d.add("Two");  
    d.add("Three");  
    System.out.println(d.remove());  
}
```

What is the result?

- A. Three
- B. One
- C. Compilation fails
- D. The program runs, but prints no output

答案 B

說明 使用 add() 和 remove() 方法時，Deque 物件表現出 Queue 資料結構的行為模式，亦即成員是「先進先出」。參考本章「6.2.5 Deque」內容。

考題 5

Which concept allows generic collections to interoperate with java code that defines collections that use raw types?

- A. bytecode manipulation
- B. casting
- C. autoboxing
- D. auto-unboxing
- E. type erasure

答案 E

說明 raw types 指使用泛型的容器(collections)的物件成員型態若未決定，Java 將以 Object 類別作為成員的型態，這種做法稱為「type erasure(型態抹除)」，參考原廠文件：<https://docs.oracle.com/javase/tutorial/java/generics/erasure.html>。

考題 6

Given:

```
public static void main(String[] args) {
    Map<String, String> m = new TreeMap<>();
    m.put("P002", "Large Stuff");
    m.put("P001", "Stuff");
    m.put("P002", "X-Large Stuff");
    Set<String> keys = m.keySet();
    for (String key : keys) {
        System.out.println(key + " " + m.get(key));
    }
}
```

What is the result?

A. P001 Stuff

P002 X-Large Stuff

B. P002 Large Stuff

P001 Stuff

C. P002 X-large Stuff

P001 Stuff

D. P001 Stuff

P002 Large Stuff

E. Compilation fails

答案 A

說明

1. 放入 Map 的 key-value，若 key 和已存在的 key 重複，將覆蓋原先內容。
2. 使用 TreeMap 實作，會以 key 做排序。

考題 7

Given:

```
public static void main(String[] args) {  
    List<String> list = new ArrayList<>(2);  
    list.add("Jim Tzeng");  
    list.add("George Washington");  
    list.add("Thomas Jefferson");  
    Collections.sort(list);  
    for (String name : list) {  
        System.out.println(name);  
    }  
}
```

What is the result?

A. Jim Tzeng

George Washington

Thomas Jefferson

B. George Washington

Jim Tzeng

Thomas Jefferson

C. Thomas Jefferson

Jim Tzeng

George Washington

D. An exception is thrown at runtime

E. Compilation fails

答案 B

說明 new ArrayList<>(2) 指 initial capacity 為 2，非長度限制。

考題 8

Given:

```
public static void main(String[] args) {  
    Deque<String> d = new ArrayDeque<String>(2);  
    d.addFirst("one");
```

```

d.addFirst("two");
d.addFirst("three");                                //line1
System.out.print(d.pollLast() + " ");
System.out.print(d.pollLast() + " ");
System.out.print(d.pollLast() + " ");    //line2
}

```

What is the result?

- A. An exception is thrown at runtime on //line1.
- B. An exception is thrown at runtime on //line2
- C. one two null
- D. one two three
- E. two one null
- F. three two one

答案 D

說明

1. new ArrayDeque<String>(2) 指 initial capacity 為 2，非長度限制。
2. 呼叫 addFirst() 方法 3 次後，成員排列為：three two one。
3. 呼叫 pollLast() 方法可以取出最後一個，所以依次為：one two three。

考題 9

Given:

```

class FullName implements Comparable<FullName> {
    String first, last;
    public FullName(String first, String last) {
        this.first = first;
        this.last = last;
    }
    @Override
    public int compareTo(FullName other) {
        int cmpLast = this.last.compareTo(other.last);
        return cmpLast != 0 ?
            cmpLast :
            this.first.compareTo(other.first);
    }
}

```

```
    public String toString() {  
        return first + " " + last;  
    }  
}
```

And:

```
public static void main(String[] args) {  
    ArrayList<FullName> names = new ArrayList<FullName>();  
    names.add(new FullName("Jim", "Tzeng"));  
    names.add(new FullName("John", "Chang"));  
    names.add(new FullName("Jane", "Chang"));  
    Collections.sort(names);  
    for (FullName n : names) {  
        System.out.println(n);  
    }  
}
```

What is the result?

A. Jane Chang

 John Chang

 Jim Tzeng

B. John Chang

 Jane Chang

 Jim Tzeng

C. Jim Tzeng

 John Chang

 Jane Chang

D. Jim Tzeng

 Jane Chang

 John Chang

E. Jane Chang

 Jim Tzeng

 John Chang

F. John Chang

 Jim Tzeng

 Jane Chang

答案 A

說明 比較 FullName 物件時，先比 last 欄位，相同時再比 first 欄位。

考題 10

Given:

```
class MyCache<T> {
    private T t;
    public void setValue(T t) {
        this.t = t;
    }
    public T getValue() {
        return this.t;
    }
}
```

And:

```
public static void main(String[] args) {
    MyCache<> c = new MyCache<Integer>();           // line1
    c.setValue(100);                                  // line2
    System.out.print(c.getValue().intValue() + 1);     // line3
}
```

- A. 101
- B. Compilation fails at //line1.
- C. Compilation fails at //line2.
- D. Compilation fails at //line3.

答案 B

說明 Java7 之後的泛型，建構子後方的菱形內容可以為空，因為可以由宣告型別使用的菱形內容推估。

考題 11

Given:

```
public static void main(String[] args) {  
    Set<String> treeSet = new TreeSet<String>(new Comparator<String>() {  
        public boolean compare(String s1, String s2) {  
            return s1.length() > s2.length();  
        }  
    });  
    treeSet.add("peach");  
    treeSet.add("orange");  
    treeSet.add("grape");  
    for (String s : treeSet) {  
        System.out.println(s);  
    }  
}
```

- A. peach
 orange
 grape
- B. peach
 orange
- C. grape
 orange
- D. The program does not compile.
- E. The program generates an exception at runtime.

答案 D

說明 方法 compare(String s1, String s2) 回傳為 int，非 boolean。

考題 12

Which two are valid initialization statements?

- A. Map<String, String> ma = new SortedMap<String, String>();
- B. Collection mb = new TreeMap<Object, Object>();
- C. HashMap<Object, Object> mc = new SortedMap<Object, Object>();
- D. SortedMap<Object, Object> md = new TreeMap<Object, Object>();

E. Hashtable me = new HashMap();
 F. Map<List, ArrayList> mf = new Hashtable<List, ArrayList>();

答案 DF

說明 選項 A : SortedMap 是 interface。選項 B : TreeMap 不屬於 Collection。選項 C : SortedMap 是 interface。選項 E : Hashtable 和 HashMap 同屬於 Map，但兩者無繼承關係。

考題 13

Given:

```
class Car {
    int id;
    String brand;
    public Car(int id, String brand) {
        this.id = id;
        this.brand = brand;
    }
    public String toString() {
        return id + ":" + brand;
    }
}
```

And:

```
public static void main(String[] args) {
    Set<Car> cars = new TreeSet<>();
    cars.add(new Car(23, "Ford"));
    cars.add(new Car(24, "BMW"));
    System.out.println(cars);
}
```

What is the result?

- A. 23 Ford
- 24 BMW
- B. 24 BMW
- 23 Ford
- C. Compilation error.
- D. ClassCastException is thrown at run time.

答案 D

說明 Car 類別要可以排序，必須實作 java.lang.Comparable。

考題 14

Given:

```
public static void main(String[] args) {  
    Map<Integer, String> map = new HashMap<>();  
    map.put(15, "z");  
    map.put(2, "b");  
    map.put(6, "d");  
    map.put(8, "e");  
    map.put(55, "j");  
    Map<Integer, String> treeMap =  
        new TreeMap<Integer, String>(  
            new Comparator<Integer>() {  
                @Override  
                public int compare(Integer o1, Integer o2) {  
                    return o2.compareTo(o1);  
                }  
            }  
        );  
    treeMap.putAll(map);  
    for (Map.Entry<Integer, String> entry : treeMap.entrySet()) {  
        System.out.print(entry.getValue() + " ");  
    }  
}
```

What is the result?

- A. Compilation errors
- B. b d e z j
- C. j z e d b
- D. z b d e j

答案 C

考題 15

Given:

```
interface MyInterface {  
}  
class Super implements MyInterface {  
}  
class Sub extends Super {  
}
```

And:

```
public static void main(String[] args) {  
    List list = new ArrayList();  
    MyInterface m1 = new Super();  
    MyInterface m2 = new Sub();      //line1  
    Super s = new Sub();  
    list.add(m1);  
    list.add(m2);                  //line2  
    list.add(s);  
    for (Object item: list) {  
        System.out.println(item.getClass().getName());  
    }  
}
```

What is the result?

A. Super

Sub

Sub

B. MyInterface

MyInterface

Super

C. Compilation fails at //line1

D. Compilation fails at //line2

答案 A

考題 16

Given:

```
public static void main(String[] args) {  
    Map<Integer, String> m = new TreeMap<>();  
    m.put(1017, "A");  
    m.put(1012, "C");  
    m.put(1011, "B");  
    m.put(1013, "B");  
    System.out.println(m);  
}
```

What is the result?

- A. {1017 = A, 1012 = C, 1011 = B, 1013 = B}
- B. {1011 = B, 1012 = C, 1013 = B, 1017 = A}
- C. {1012 = C, 1013 = B, 1017 = A}
- D. {1017 = A, 1011 = B, 1013 = B, 1012 = C}

答案 B

考題 17

Given:

```
class MyFoo<K, V> {  
    private K key;  
    private V value;  
    public MyFoo(K key, V value) {  
        this.key = key;  
        this.value = value;  
    }  
    public static <T> MyFoo<T, T> twice(T value) {  
        return new MyFoo<T, T>(value, value);  
    }  
}
```

Which option fails?

- A. MyFoo<String, Integer> optionA = new MyFoo<String, Integer>("Duke", 10);
- B. MyFoo<String, String> optionB = MyFoo.<String> twice("Duke");

- C. MyFoo<Object, Object> optionC = new MyFoo<String, Integer>("Duke", 32);
 D. MyFoo<String, String> optionD = new MyFoo<>("Duke", "Java");

答案 C

說明 C 選項須改為：

```
MyFoo<? extends Object, ? extends Object> optionC
    = new MyFoo<String, Integer>("Duke", 32);
```

考題 18

Given:

```
class Exam implements Comparator<Exam> {
    String name;
    double level;
    public Exam() { }
    public Exam(String name, double price) {
        this.name = name;
        this.level = price;
    }
    public int compare(Exam b1, Exam b2) {
        return b1.name.compareTo(b2.name);
    }
    public String toString() {
        return name + ":" + level;
    }
}
```

And:

```
public static void main(String[] args) {
    List<Exam> exams =
        Arrays.asList( new Exam("OCPJP", 2),
                      new Exam("OCAJP", 3));
    Collections.sort(exams, new Exam());
    System.out.print(exams);
}
```

- A. [OCAJP:3.0, OCPJP:2.0]
 B. [OCPJP:2.0, OCAJP:3.0]

- C. Compilation error occurs because the Exam class does not override the abstract method compareTo().
- D. Exception is thrown at run time.

答案 A

考題 19

Given:

```
class GenericTest<T> {  
    private T t;  
    public T get() {  
        return t;  
    }  
    public void set(T t) {  
        this.t = t;  
    }  
}
```

And:

```
public static void main(String args[]) {  
    GenericTest<String> t1 = new GenericTest<>();  
    GenericTest t2 = new GenericTest();      // line1  
    t1.set("Java");  
    t2.set(100);                      // line2  
    System.out.print(t1.get() + " " + t2.get());  
}
```

What is the result?

- A. Java 100
- B. Java.lang.string@<hashcode> java.lang.Integer@<hashcode>
- C. A compilation error occurs. To rectify it, replace //line1 with:

```
GenericTest <Integer> type1 = new GenericTest <>();
```
- D. Compilation error occurs. To rectify it, replace //line2 with:

```
t2.set (Integer(100));
```

答案 A

說明 類別雖然設計為泛型，建立物件時不一定要使用。

Chapter

07

Exceptions和 Assertions

-
- | 7.1 Exceptions
 - | 7.2 Assertions
 - | 7.3 認證考試命題範圍

7.1 Exceptions

程式執行多少會遇到問題。值得信賴 (reliable) 的程式會優雅 (gracefully) 的處理意外狀況：

1. 處理目標是「exception(意外)」，而非預期狀況。
2. 意外必須處理以建立可信賴的程式。
3. 發生原因可能是程式的 bugs 。
4. 發生原因可能是程式無法處理的狀況，如：
 - 資料庫無法連線。
 - 硬碟毀損。

C 語言的錯誤發生，通常是以回傳負值表示，如「int x = printf("hi")」。Java 則在出現錯誤時，由 JVM 拋出例外物件 (Exception)，分類如下。不同種類的例外，有不同處理方式：

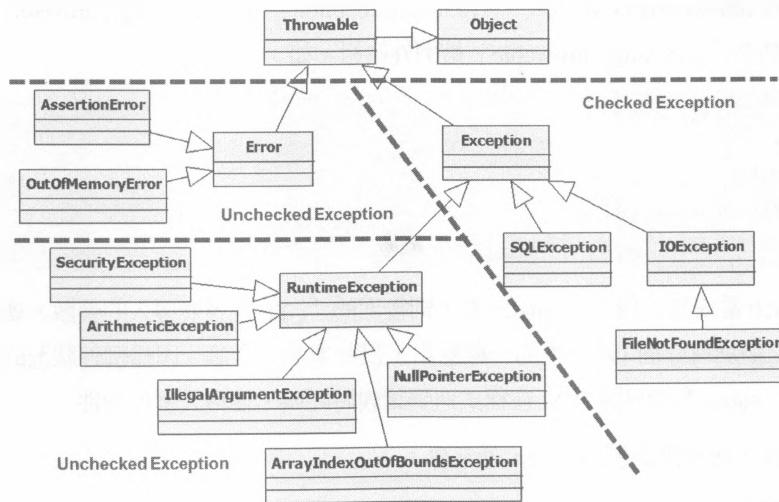
◆表 7-1 Exception 分類表

分類	Checked Exception	Unchecked Exception	
情境	已然預知風險，必須事先預防	無法預知風險，無法事先預防	
處理方式	1. 方法內部自己處理。 2. 方法內部不處理但提醒呼叫者要處理。	不需要事先處理	
代表類別	所有例外類別都是。除： 1.RuntimeException 2.Error 類別和其子類別	RuntimeException 類別和其子類別	Error 類別和其子類別
		歸類程式內部原因： 如資料輸入異常	歸類程式外部原因： 如硬體、網路等

依上表分類，認證考試常見的例外家族成員如圖 7-1。絕大部分的類別都將在本書各章節分別說明。

當呼叫其他類別的方法時，若被呼叫的方法宣告有拋出 checked exception 的風險，編譯器會要求呼叫者方法必須「處理 (handle)」或是也「宣告 (declare)」可能發生的問題：

1. **Handling Exception**：表示必須有程式碼區塊來處理異常狀況，此時使用「try-catch」敘述。
2. **Declaring Exception**：在方法上註記執行可能出現的錯誤，提醒使用的方法必須處理，此時使用「throws」宣告語法。



◆ 圖 7-1 認證考試常見例外

7.1.1 使用 try-catch 程式碼區塊

一般例外狀況

範例「/OCP/src/course/c07/TryCatchDemo.java」使用 try-catch 語法處理例外：

範例

```

1  public class TryCatchDemo {
2      public static void main(String[] args) {
3          try {
4              System.out.println("Opening a file...");
5              InputStream in = new FileInputStream("lostFile.txt");
6              System.out.println("File is opened");
7          } catch (Exception e) {
8              e.printStackTrace();
9          }
10     }
11 }
```

說明

5	程式碼可能在第 5 行出現錯誤。
6	一旦出現錯誤，程式將不會進入第 6 行。
8	一旦出現錯誤，程式將進入第 8 行。

catch 程式碼區塊由 JVM 傳入「java.lang.Exception」或「java.lang.Throwable」的子類別物件。其中「java.lang.Throwable」是例外始祖。如：

```
try {
    //...
} catch (Exception e) {
    e.printStackTrace();
}
```

若把 catch 當成是一種方法 (method)，則後面的「()」代表要傳入的參數。傳入大分類的型別 (父類別或介面) 是多型的一種應用，但此處並不合適，因為例外狀況的處理應該對症下藥。catch 方法只讓 JVM 在程式遇到錯誤時呼叫，並傳入例外物件。

一般來說，在捕捉到例外之後，通常會：

1. 記錄錯誤訊息。
2. 再試一次。
3. 嘗試其他替代方案。
4. 優雅地離開 (return) 或結束程式 (exit)。

複雜例外狀況

一個程式碼區塊或一個方法，有時必須同時處理多種可能的意外狀況：

1. 單一「try」，可以搭配多個「catch」程式碼區塊。此時例外子類別排序應在父類別上面，避免所有例外在一開始就被例外父類別，如 Exception 或 Throwable 所攔截。
2. Catch exceptions 時應該盡可能捕捉最特定 (specific type) 的例外子類別。因為不同的例外，應該有不同的處理方式。
3. Java Persistence API (JPA) 的例外大部分均繼承 RuntimeException，屬於 unchecked exception，慣例上屬於不用處理的例外。但在正式環境裡，還是應該處理。

如範例「/OCP/src/course/c07/TryCatchDemo2.java」：

範例

```
1 public class TryCatchDemo2 {
2     public static void main(String[] args) {
3         try {
4             System.out.println("Opening a file...");
5             InputStream in = new FileInputStream("lostFile.txt");
6             System.out.println("File is opened");
7             int data = in.read();
```

```

8         in.close();
9     } catch (FileNotFoundException e) {
10        e.printStackTrace();
11    } catch (IOException e) {
12        e.printStackTrace();
13    } catch (Exception e) {
14        e.printStackTrace();
15    }
16 }
17 }
```

記錄 (Logging) 錯誤內容

正式環境中，應該要移除「printStackTrace()」或「System.out.println(e.getMessage())」這類程式碼，因為它們只是當下將錯誤訊息呈現在螢幕 (screen) 上。當程式碼執行錯誤時，應該要寫入紀錄 / 日誌檔 (log file)，讓程式設計師可以在事後檢視問題。Java 有多種相關函式庫可以選擇，如：

- Apache's Log4j
- built-in java.util logging framework

都是不錯的選擇。

使用 finally 敘述

當使用外部資源 (resource)，如開啓檔案或連線資料庫時，應該在不使用時關閉 (close) 資源。若在 try 的程式碼區塊中關閉資源，有可能因為執行錯誤導致資源有開啓，但來不及關閉，程式就已經結束。此時，可以使用 finally 敘述：

- Java 保證在 finally 程式碼區塊中，不管是 try 或 catch 執行結束，都一定會進入執行。
- 有時在 finally 程式碼區塊的程式碼也可能出錯，因此需要巢狀的 try-catch 區塊去處理。

如範例「/OCP/src/course/c07/TryCatchFinallyDemo.java」：

範例

```

1 public class TryCatchFinallyDemo {
2     public static void main(String[] args) {
3         InputStream in = null;
4         try {
5             System.out.println("Opening a file...");
6             in = new FileInputStream("lostFile.txt");
7             System.out.println("File is opened");
8             int data = in.read();
```

```

9         in.close();
10    } catch (FileNotFoundException e) {
11        e.printStackTrace();
12    } catch (IOException e) {
13        e.printStackTrace();
14    } catch (Exception e) {
15        e.printStackTrace();
16    } finally {
17        try {
18            if (in != null)
19                in.close(); // try to close file
20        } catch (IOException e) {
21            System.out.println("Failed to close file");
22        }
23    }
24}
25}

```

7.1.2 使用 try-with-resources 程式碼區塊

使用 try-with-resources 敘述

因為 finally 區塊中的程式碼冗長且具備可預測性，Java 7 提供新的「try-with-resources」敘述，可以自動關閉被開啟的「資源 (resource)」：

三 語法

```

try ( 壓告並開啟資源 [; 壓告並開啟其他資源…] ) {
    //...
}
// 在 try 程式碼區塊之後，資源將自動關閉

```

- 這裡定義的資源，必須是實作「java.lang.AutoCloseable」介面的類別。
- 若要開啟多個資源，可使用「;」做區隔。
- 自動關閉的順序將和使用資源的開啟順序相反。

如範例「/OCP/src/course/c07/TryWithResourceDemo.java」：

四 範例

```

1 public class TryWithResourceDemo {
2     public static void main(String[] args) {
3         System.out.println("Opening a file...");
4         try (InputStream in = new FileInputStream("lostFile.txt")) {

```

```

5     System.out.println("File is opened");
6     int data = in.read();
7     in.close();
8 } catch (FileNotFoundException e) {
9     e.printStackTrace();
10 } catch (IOException e) {
11     e.printStackTrace();
12 } catch (Exception e) {
13     e.printStackTrace();
14 }
15 }
16 }
```

比較前後範例，可以發現：

1. 移除 finally 程式碼區塊，如前範例行 16~22。
2. 使用 try-with-resource 告訴要開啟的資源，如本範例行 4。

認識 AutoCloseable 介面

範例中所使用的類別 InputStream，檢查 source code 可以發現實作了介面 java.io.Closeable。

```
public abstract class InputStream implements Closeable {
```

介面 java.io.Closeable 又繼承了介面 java.lang.AutoCloseable：

```
public interface Closeable extends AutoCloseable {
```

「資源(resource)」要藉由 try-with-resources 敘述開啟和自動關閉，必須實作以下兩者之一：

1. 介面 java.lang.AutoCloseable
 - Java 7 新增。
 - 唯一的抽象方法 close() 會拋出 Exception 物件。

```
public interface AutoCloseable {
    void close() throws Exception;
}
```

2. 介面 java.io.Closeable
 - Java 5 就存在，在 Java 7 中修改使其繼承介面 AutoCloseable。
 - 唯一的抽象方法 close() 會拋出 IOException 物件。

```
public interface Closeable extends AutoCloseable {
    public void close() throws IOException;
}
```

程式語言裡有一種方法的實作方式稱為「**Idempotent method**」，表示即使重複執行多次，也不會有「副作用 (side effects)」產生。

Java 要求實作「java.io.Closeable」的 close() 方法時，必須滿足「**idempotent**」的要求；但並未一致要求實作「java.lang.AutoCloseable」的 close() 方法必須比照辦理。即便如此，我們在實作「java.lang.AutoCloseable」時，應該也要做到反覆執行多次，都沒有副作用產生。做法是：

```
// 若資源未關閉 (以 resource == null 判定)
If (resource != null) {
    // 關閉資源，並使 resource = null
}
```

7.1.3 Suppressed Exceptions

使用 try-with-resource 敘述雖然可以由 Java 自動關閉資源，但也延伸出新的例外 (exception) 處理問題必須注意：

1. 若 Java 在開啓資源時拋出例外，程式碼將直接跳到「catch 區塊」。這和一般情況一樣，只拋出一個例外。
2. 若「try 區塊」內拋出例外，Java 準備關閉資源；但在幕後關閉時又不幸拋出例外，此時共產生二個例外。雖然例外有出現順序，但對於接收例外的「catch 區塊」而言，卻是等於必須同時接收二個例外物件。這種情況是因為 try-with-resource 敘述才間接出現，必須重新定義處理流程。
3. 「try 區塊」成功執行完畢，但在關閉資源時出錯；和一般情況一樣，此時只拋出一個例外。

由以上討論，傳統的 try-catch 敘述一次只能處理一個例外；當前述狀況二，也就是有二個例外類別被先後拋出時，Java 7 的解決方案是將「後發生、同時也是和關閉資源有關」的例外，隱匿或擠壓 (suppressed) 到「先發生、同時也是我們程式碼造成」的例外裡，使之可以被保留。

以上，Java 會負責將資源相關的例外隱匿或擠壓進程式碼發生的例外裡，我們只要知道如何將該「suppressed 例外物件」取出來即可：

```

} catch(Exception e) {
    System.out.println(e.getMessage());
    for(Throwable t : e.getSuppressed()) {
        System.out.println(t.getMessage());
    }
}
}

```

由上述範例可知，當對例外物件呼叫其 `getSuppressed()` 方法時，將可以回傳一個 `Throwable` 的陣列。如此，再多在幕後被擠壓 / 隱匿的例外都能被找出並妥善處理。

範例「/OCP/src/course/c07/SuppressedExceptions.java」完整說明 Suppressed Exceptions 的設計流程：

範例

```

1  class TryException extends Exception {
2
3  class FinallyException extends Exception {
4
5  public class SuppressedExceptions {
6      public static void main(String[] args) throws Exception {
7          before7();
8          after7();
9      }
10     private static void before7() {
11         try {
12             try {
13                 throw new TryException(); // This is lost
14             } finally {
15                 throw new FinallyException();
16             }
17         } catch (Exception e) {
18             System.out.println("before7: " + e.getClass());
19         }
20     }
21     private static void after7() {
22         try {
23             Throwable t = null;
24             try {
25                 throw new TryException();
26             } catch (Exception e) {
27                 t = e;
28             } finally {
29                 FinallyException fe = new FinallyException();
30                 if (t != null) {

```

```

31             t.addSuppressed(fe);
32             throw t;
33         } else {
34             throw fe;
35         }
36     }
37 } catch (Throwable e) {
38     System.out.println("after7: " + e.getClass());
39     for (Throwable t : e.getSuppressed()) {
40         System.out.println("after7: " + t.getClass());
41     }
42 }
43 }
44 }
```

結果

```

before7: class course.c07.FinallyException
after7: class course.c07.TryException
after7: class course.c07.FinallyException
```

說明

10-20	在方法before7() 的示範中，可以知道若try-catch敘述同時拋出2個例外物件，本例中為TryException(模擬因程式問題拋出的例外) 和 FinallyException(模擬因關閉資源所拋出的例外)，則catch區塊中只會捕捉到最後一個，亦即FinallyException。
21-43	在方法after7() 的示範中： <ol style="list-style-type: none"> 利用行23和27的設計，保留了第一個拋出的例外物件 TryException。 在行29-35的finally區塊中，若又拋出FinallyException，則將FinallyException隱匿至TryException中，再拋出TryException。 若沒有TryException物件，也可以選擇單獨拋出FinallyException。

7.1.4 使用 multi-catch 語法

在OCA一書的範例中，我們曾經演示過一個程式區塊可能拋出多個例外，如以下做法。此時例外的父類別必須往後擺，避免一開始就被父類別例外所捕捉，導致後面的子類別例外無用武之地：

```

try {
    createTempFile(); // method to create temporary file in file system
} catch (IOException ioe) {
    System.out.println("Known Exception: " + ioe.getClass());
} catch (IllegalArgumentException iae) {
    System.out.println("Known Exception: " + iae.getClass());
```

```

} catch (ArrayIndexOutOfBoundsException aiobe) {
    System.out.println("Known Exception: " + aiobe.getClass());
} catch (SecurityException se) {
    System.out.println("Known Exception: " + se.getClass());
} catch (Exception e) {
    System.out.println(
        "Unexpected Exception: " + e.getClass() + " is caught!");
} catch (IOException ioe) {
}

```

在這裡，不建議直接捕捉例外的父類別如 Exception 或 Throwable 是因為：

1. 每種例外的處理方式應該不同。
2. 要清楚知道究竟有多少例外可能產生。

但若每種例外處裡方式確實相同，可以使用 Java 7 新的「multi-catch」語法，如範例「/OCP/src/course/c07/MultiCatchDemo.java」。它的好處是：

1. 依然可以清楚知道究竟有多少例外可能產生。
2. 多種例外可以用同一種方式處理，程式碼簡潔。

必須注意的是：不同例外以「|」區隔時，前後例外必須「沒有繼承關係」。

範例

```

1 public class MultiCatchDemo {
2     public static void main(String args[]) {
3         before7();
4         after7();
5     }
6     private static void after7() {
7         try {
8             createTempFile();
9         } catch (IOException
10            | IllegalArgumentException
11            | ArrayIndexOutOfBoundsException
12            | SecurityException e) {
13             System.out.println("Known Exception: " + e.getClass());
14         } catch (Exception e) {
15             System.out.println(
16                 "Unexpected Exception: " + e.getClass() + " is caught!");
17         }
18     private static void before7() {
19         try {
20             createTempFile();

```

```

21     } catch (IOException ioe) {
22         System.out.println("Known Exception: " + ioe.getClass());
23     } catch (IllegalArgumentException iae) {
24         System.out.println("Known Exception: " + iae.getClass());
25     } catch (ArrayIndexOutOfBoundsException aiobe) {
26         System.out.println("Known Exception: " + aiobe.getClass());
27     } catch (SecurityException se) {
28         System.out.println("Known Exception: " + se.getClass());
29     } catch (Exception e) {
30         System.out.println(
31             "Unexpected Execption: " + e.getClass() + " is caught!");
32     }
33 }
34 private static void createTempFile() throws IOException {
35     String path = System.getProperty("user.dir") + "/src/course/c07/temp";
36     System.out.println(path);
37     File f = new File(path);
38     File tf = File.createTempFile("ji", null, f);
39     System.out.println("Temp file name: " + tf.getPath());
40     int arr[] = new int[5];
41     arr[5] = 25;
42 }
```

說明

9-12	使用 Java 7 開始的新 multi-catch 作法，無繼承關係的例外類別可以擺在一起，並以「 」區隔時。
13	因為行 22、24、26、28 的例外處理方式都相同，故使用 multi-catch 語法。
14	Exception 類別是例外父類別，不能和其他例外類別放一起。

7.1.5 使用 throws 宣告

除了直接處裡例外，也可以在類別的方法上宣告「throws ExceptionTypes」，讓呼叫該方法的呼叫者處理。如範例「/OCP/src/course/c07/ThrowsExDemo.java」：

範例

```

1 public class ThrowsExDemo {
2     public static void readFromFile1() throws FileNotFoundException,
3                                         IOException,
4                                         Exception {
5         try (InputStream in = new FileInputStream("a.txt")) {
6             // codes go here
7 }
```

```

5      }
6  }
7  public static void readFromFile2() throws Exception,
8          IOException,
9          FileNotFoundException {
10     try (InputStream in = new FileInputStream("a.txt")) {
11         // codes go here
12     }
13 }
14 public static void main(String[] args) {
15     try {
16         readFromFile1();
17     } catch (Exception e) {
18         System.out.println(e.getMessage());
19     }
}

```

說明

2, 7	列出所有例外，順序沒關係！！
3, 8	使用try-with-resource 敘述，不需要finally區塊關閉資源。
15	當呼叫者使用了宣告「throws ExceptionTypes」的方法時，呼叫者有義務要處理。
12	main() 方法也可以宣告 throws Exception，將例外再往外拋，但等同不處理，不建議這樣做。

複寫子類別方法時，若父類別方法有宣告拋出例外，則：

1. 若例外為 checked exception，則子類別複寫方法時，拋出的例外必須：
 - 例外型別必須相同或為子類別，表示覆寫後有精進。
 - 數量若相同或更少，表示問題已被處理，也是一種進步。
2. 若例外為 unchecked exception，如 RuntimeException，則子類別複寫方法時可以不理會。

範例說明如「/OCP/src/course/c07/ExceptionOverrideDemo.java」：

範例

```

1 abstract class Father {
2     abstract void fatherMethod1() throws IOException;
3     abstract void fatherMethod2() throws RuntimeException;
4     abstract void fatherMethod3() throws SQLException;
5 }
6 class Child extends Father {

```

```

7     @Override
8     void fatherMethod1() throws IOException, FileNotFoundException {
9     }
10    @Override
11    void fatherMethod2() {
12    }
13    @Override
14    void fatherMethod3() {
15    }
16 }

```

說明

2,8	父類別宣告拋出 IOException。子類別除拋出 IOException，又拋出 FileNotFoundException，因為 FileNotFoundException 是 IOException 的子類別，並未超出父類別的 IOException 範圍，所以沒問題。
3,11	父類別宣告拋出 RuntimeException，屬於 unchecked exception，子類別可以不處理。
4,14	父類別宣告拋出 SQLException，子類別可以在覆寫的方法內使用 try-catch 敘述將之妥慎處理，故可以不再拋出例外。

7.1.6 Exception 物件的捕捉再拋出

Java 捕捉例外物件後，可以先進行某些處理，再使用「throw」(沒有 s，非複數)的語法拋出。

Java 7 之後，可以更聰明的判讀出例外物件的真正實例，因此編譯時允許再拋出被捕捉的例外物件時，可以比宣告的例外型別更精準。如範例「/OCP/src/course/c07/ReThrowsExDemo.java」：

範例

```

1  class Exception1 extends Exception {
2  }
3  class Exception2 extends Exception {
4  }
5  public class ReThrowsExDemo {
6      public void rethrowExBeforeJ7() throws Exception {
7          try {
8              if (Math.random() > 0.5) {
9                  throw new Exception1();
10             } else {
11                 throw new Exception2();
12             }
13         }
14     }
15 }

```

```

13     } catch (Exception e) {
14         throw e;
15     }
16 }
17 public void rethrowExAfterJ7() throws Exception1, Exception2 {
18     try {
19         if (Math.random() > 0.5) {
20             throw new Exception1();
21         } else {
22             throw new Exception2();
23         }
24     } catch (Exception e) {
25         throw e;
26     }
27 }
28 }
```

說明

6-16	顯示 Java 7 之前對例外捕捉後再拋出的限制。因為行 13 告訴捕捉的例外型別為 Exception，用變數 e 代表，因此使用 throw e 再拋出時，行 6 就必須宣告 throws Exception。
17-27	顯示 Java 7 之後對例外捕捉後再拋出變得較聰明！雖然行 24 和行 13 相同，依然宣告捕捉的例外型別為 Exception，但 Java 7 的編譯器可以判斷變數 e 的真實型別只可能是 Exception1 和 Exception2，因此允許在行 17 告訴 throws Exception1, Exception2。

7.1.7 建立客製的 Exception 類別

我們也可以繼承 Exception 類別，建立客製化的例外子類別 DAOException，如下：

```

public class DAOException extends Exception {
    public DAOException() {
        super();
    }
    public DAOException(String message) {
        super(message);
    }
}
```

標準 Java 不會主動拋出客製化的例外子類別，必須先捕捉標準的例外類別，再拋出。如以下程式片段：

```

try {
    //some codes might error!
```

```

} catch (Exception e) {
    e.printStackTrace();
    throw new DAOException();
}

```

如果希望再拋出的客製化例外子類別，也能保留最初被捕捉的例外類別的訊息，則可以使用包裹例外類別(wrapper exceptions)的程式技巧，將最初的例外類別，包裹進例外子類別中。如範例「/OCP/src/course/c07/DAOException.java」：

範例

```

1 public class DAOException extends Exception {
2     public DAOException(Throwable cause) {
3         super(cause);
4     }
5     public DAOException(String message, Throwable cause) {
6         super(message, cause);
7     }
8 }

```

要取出被包裹的原始例外物件，可使用 `getCause()` 方法，如：

```

try {
    //some codes might error!
} catch (DAOException e) {
    Throwable t = e.getCause();
}

```

這類技巧，對於處理多型設計時，把「不同實作方法所拋出的不同例外類別」一致化有很大幫助。以 DAO 設計模式來說：

1. DAO 設計模式以介面定義資料儲存的方法，然後允許抽換實作為：
 - File-based 的實作類別，將資料儲存在檔案中。
 - JDBC-based 的實作類別，將資料儲存在資料庫中。
2. 因為 File-based 方法可能拋出 `IOException`，JDBC-based 方法可能拋出 `SQLException`，為求兩者一致，DAO 介面的抽象方法過去只能：
 - 不拋出例外物件，所以實作方法必須處理掉例外，稱為「swallow exception」，亦即吞掉 `Exception`。
 - 宣告拋出例外的父類別，如 `Exception` 或 `Throwable`。

現在，使用「包裹例外類別(wrapper exceptions)」就可以解決這個問題。

介面的抽象方法可以宣告拋出 DAOException：

```
Employee findById(int id) throws DAOException;
```

File-based 的方法實作方式為：

```
public Employee findById(int id) throws DAOException {
    try {
        return getEmployeeFromFile(int id);
    } catch (IOException e) {
        throw new DAOException(e);
    }
}
```

JDBC-based 的方法實作方式為：

```
public Employee findById(int id) throws DAOException {
    try {
        return getEmployeeFromDatabase(int id);
    } catch (SQLException e) {
        throw new DAOException(e);
    }
}
```

7.2 Assertions

7.2.1 Assertions 的簡介和語法

Assertions 的簡介

「assertions」中文翻譯為「斷言」，亦即我們常說的「鐵口直斷某件事的結果」。在 Java 裡我們用 assertions 來斷言程式執行的某種結果；斷言可能準確，也可能失準。斷言若失準被認為是嚴重的問題，因為表示程式執行結果和預期有出入，因此將拋出 Assertion Error 並中斷程式執行。

Assertions 的語法

使用語法為：

語法

```
assert <boolean_expression> ;
assert <boolean_expression> : <detail_expression> ;
```

其中，

1. 若 <boolean_expression> 為 false，拋出 AssertionError。
2. <detail_expression> 為 AssertionError 的 getMessage() 方法回傳的字串。
3. AssertionError 屬 Error 的子類別，為 unchecked exception。

Assertions 使用情境

我們使用 assertions 來驗證假設和方法的不變量 (invariant，指不會改變的數值或結果)，通常有幾種情況：

1. 內部的不變量 (internal invariants)。
2. 流程控管的不變量 (control flow invariants)。
3. 事後的狀態和類別不變量 (post-conditions and class invariants)。

將在下一小節逐一介紹。

Assertions 的使用注意事項

因為 assertions 的檢查預設關閉 (disabled)，使用前必須開啓；Assert 若失敗，程式將會中斷，並顯示 debug 訊息。因此要避免不當的使用方式：

1. 不可用於類別方法的參數輸入檢查。
2. 不可以影響程式正常流程。如以下程式碼將物件的生成放在 assertion 的判斷敘述中：

```
SomeType s = null;
assert (s = new SomeType()) != null;
```

7.2.2 Assertions 的使用

內部的不變量 (internal invariants)

使用情境為：

```
if (x > 0) {
    // do if x > 0
} else {
```

```
// do if x = 0 or x < 0
}
```

使用 assertion :

範例

```
1  if (x > 0) {
2      // do if x > 0
3  } else {
4      assert (x == 0);
5 }
```

說明

4	若能進入else 區塊，表示 $x \leq 0$ ；加上 assertion，表示這裡 x 必定 $= 0$ 。
---	---

流程控管的不變量 (control flow invariants)

用於：

範例

```
1  private static void controlFlowInvariants(Gender g) {
2      switch (g) {
3          case MALE:    // ...
4              break;
5          case FEMALE: // ...
6              break;
7          default:
8              assert false : "Unknown gender!!";
9              break;
10     }
11 }
```

說明

8	斷言絕不可能進入 default 條件！直接使用 assert false，表示不用再判斷，馬上失敗。
---	---

事後的狀態和類別不變量 (post-conditions and class invariants)

以我們自己設計的 MyTime 類別為例。無論類別的成員經過任何改變，都應該能通過方法 rule() 的檢驗：

範例

```

1  class MyTime {
2      int hours;
3      int minutes;
4      int seconds;
5      void rule() {
6          assert(0 <= hours && hours < 24);
7          assert(0 <= minutes && minutes < 60);
8          assert(0 <= seconds && seconds < 60);
9      }
10 }

```

Assertions 的開啟與關閉

Assertion 可以關閉，預設是關閉。關閉後程式碼完全不會執行，和註解 (comment) 類似，因此不影響效能。以開啓類別 HelloWorld 的 assertion 關閉為例：

```

java -enableassertions HelloWorld
java -ea HelloWorld

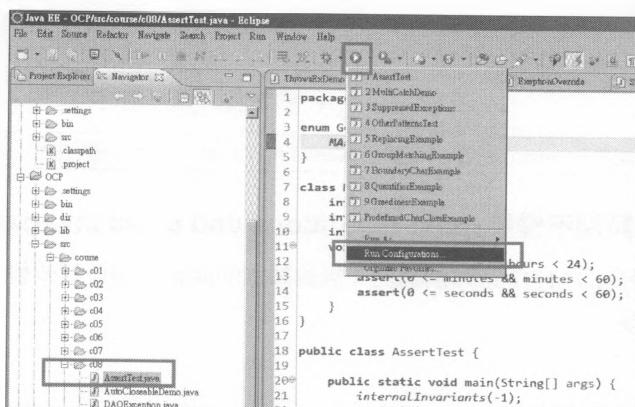
```

「-ea」即是「- enableassertions」的縮寫。若在「-ea」後加上其他 option，可以控制 assertion 的啓用只在某個 package 或 class，讀者可以參閱官方說明：<http://docs.oracle.com/javase/7/docs/technotes/guides/language/assert.html#enable-disable>

在 Eclipse 執行時開啟 assertions

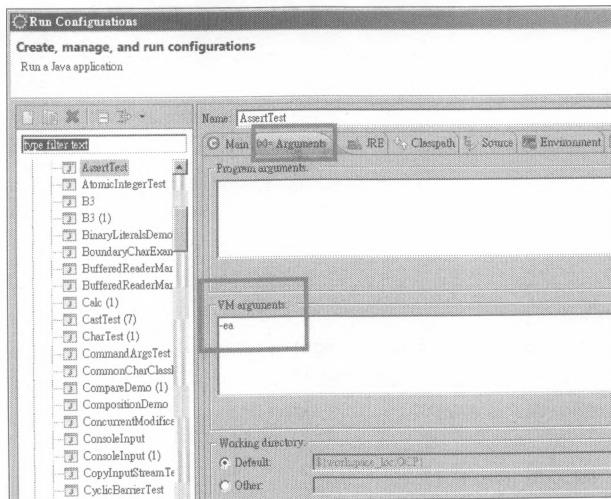
若要在 Eclipse 執行時開啓 assertion，步驟為：

STEP01 點選要執行的類別(需要有 main 方法)，如點選 Run AssertTest，再選擇選項「Run Configurations...」：



◆ 圖 7-2 設定 Run Configurations...

STEP02 在第二個頁籤「Arguments」下，輸入 VM arguments : -ea，再點擊 Run 按鍵，即可在開啟 assertion 的狀態下執行類別程式：



◆ 圖 7-3 輸入 -ea

7.3 認證考試命題範圍

依據甲骨文公布的考試範圍 (https://education.oracle.com/java-se-8-programmer-ii/pexam_1Z0-809)，和本章相關的主題有：

Exceptions and Assertions

1. Use **try-catch** and **throw** statements.
2. Use **catch**, **multi-catch**, and **finally** clauses.
3. Use **Autoclose** resources with a **try-with-resources** statement.
4. Create **custom exceptions** and **Auto-closeable resources**.
5. Test **invariants** by using **assertions**.

本章擬真試題實戰

考題 1

Given:

```
public class Test {  
    public static void main(String[] args) {  
        String[] arr = { "SE", "ee", "ME" };  
        for (String var : arr) {  
            try {  
                switch (var) {  
                    case "SE":  
                        System.out.println("Standard Edition");  
                        break;  
                    case "EE":  
                        System.out.println("Enterprise Edition");  
                        break;  
                    case "ME":  
                        System.out.println("Micro Edition");  
                        break;  
                    default:  
                        assert false;  
                }  
            } catch (Exception e) {  
                System.out.println(e.getClass());  
            }  
        }  
    }  
}
```

And:

```
javac Test.java  
java -ea Test
```

What is the result?

A. Compilation fails

B. Standard Edition

Enterprise Edition

Micro Edition

C. Standard Edition

```
class java.lang.AssertionError
```

Micro Edition

D. Standard Edition is printed and an AssertionError is thrown

答案 D

說明 選項 C：必須把 catch (Exception e) 改成 catch (Error e)。

考題 2

Given:

```
class IllegalAgeException extends IllegalArgumentException {
}
class TestAge {
    void test(int age) throws IllegalArgumentException {
        if (age < 20)
            throw new IllegalAgeException();
        if (age >= 20 && age <= 60)
            System.out.print("General Age");
        else
            System.out.print("Senior Age");
    }
}
public class Test {
    public static void main(String[] args) {
        int age = Integer.parseInt(args[1]);
        try {
            new TestAge().test(age);
        } catch (Exception e) {
            System.out.print(e.getClass());
        }
    }
}
```

And the command-line invocation:

```
java Test 12 11
```

What is the result?

- A. General Age
- B. class IllegalAgeException
- C. class java.lang.IllegalArgumentException
- D. class java.lang.RuntimeException

答案 B

說明 實際拋出的例外物件是 IllegalAgeException。

考題 3

Given:

```
static public void main(String args[]) {  
    System.out.print("start ");  
    try {  
        int i = 1 / 0;  
        System.out.print("try ");  
    } catch (Exception e) {  
        System.out.print("catch ");  
        throw e;  
    } finally {  
        System.out.print("finally ");  
    }  
    System.out.print("after ");  
}
```

What is the result?

- A. start try catch finally after
- B. start catch finally after
- C. start catch after
- D. start catch finally
- E. start catch

答案 D

考題 4

Given:

```

class ValueOutOfBoundsException extends ArrayIndexOutOfBoundsException {
}
class TestException {
    public void test(int[] arr) throws ArrayIndexOutOfBoundsException {
        for (int i = 1; i <= 3; i++) {
            if (arr[i] > 100)
                throw new ValueOutOfBoundsException();
            System.out.println(arr[i]);
        }
    }
    public static void main(String[] args) {
        int[] arr = { 150, 77, 44 };
        try {
            new TestException().test(arr);
        } catch (ArrayIndexOutOfBoundsException
                 | ValueOutOfBoundsException e) {
            System.out.println(e.getClass());
        }
    }
}

```

What is the result?

- A. Compilation fails.
- B. 77 class java.lang.ArrayIndexOutOfBoundsException
- C. class ValueOutOfBoundsException
- D. class java.lang.ArrayIndexOutOfBoundsException

答案 A

說明 使用「multi-catch」語法將不同例外以「|」區隔時，前後例外必須「沒有繼承關係」。

考題 5

Given:

```
static void display(String[] arr) {  
    try {  
        System.out.print(arr[2]);  
    } catch (ArrayIndexOutOfBoundsException | NullPointerException e) {  
        e = new Exception();  
        throw e;  
    }  
}  
  
public static void main(String[] args) {  
    try {  
        String[] arr = { "Java", "C++", "Null" };  
        display(arr);  
    } catch (Exception e) {  
        System.out.print(e.getClass());  
    }  
}
```

What is the result?

- A. Null
- B. class java.lang.ArrayIndexOutOfBoundsException
- C. class java.lang.NullPointerException
- D. class java.lang.Exception
- E. Compilation fails.

答案 E

說明 Multi-catch 區塊的參數不可以再被重新指定。

考題 6

Given:

```
public class Test {  
    public static int getCount(String[] ss) {  
        int count = 0;  
        for (String s : ss) {  
            if (s != null)  
                count++;  
        }  
    }
```

```

        return count;
    }
    public static void main(String[] args) {
        String[] ss = new String[4];
        ss[1] = "C";
        ss[2] = "";
        ss[3] = "Java";
        assert (getCount(ss) < ss.length);
        System.out.print(getCount(ss));
    }
}

```

And the commands:

```

javac Test.java
java -ea Test

```

What is the result?

- A. 2
- B. 3
- C. NullPointerException is thrown at runtime
- D. AssertionError is thrown at runtime
- E. Compilation fails

答案 B

考題 7

Which two statements are true about the try-with-resources statement?

- A. The resources declared in a try-with-resources statement are not closed automatically if an exception occurs inside the try block.
- B. In a try-with-resources statement, any catch or finally block is run after the resources have been closed.
- C. The close methods of resources are called in the reverse order of their creation.
- D. All the resources must implement the java.io.Closeable interface.

答案 BC

說明 選項 A：即便拋出例外還是會自動關閉。選項 D：資源應該實作 `java.lang.AutoCloseable`。選項 B：實際用以下範例進行測試後驗證 resource 關閉後才會進入 catch 或 finally 程式區塊：

範例

```

1  class ResourceOpenException extends Exception {
2  }
3  class ResourceWorkingException extends Exception {
4  }
5  class ResourceCloseException extends Exception {
6  }
7  class Resource implements AutoCloseable {
8      public Resource() throws Exception {
9          System.out.println("The resource is created.");
10         // throw new ResourceOpenException();
11     }
12     public void work() throws Exception {
13         System.out.println("The resource is working.");
14         throw new ResourceWorkingException();
15     }
16     @Override
17     public void close() throws Exception {
18         System.out.println("The resource is closed.");
19         // throw new ResourceCloseException();
20     }
21 }
22 public class ResourceDemo {
23     public static void main(String[] args) throws Exception {
24         try (Resource r = new Resource()) {
25             r.work();
26         } catch (Exception e) {
27             System.out.println("Catch an exception: " + e.getClass());
28         } finally {
29             System.out.println("Final block ... ");
30         }
31     }
32 }
```

結果

```

The resource is created.
The resource is working.
The resource is closed.
```

```
Catch an exception: class course.c07.ResourceWorkingException
Final block ...
```

考題 8

Given:

```
final class Book implements AutoCloseable { // line1
    // line2
    public void open() {
        System.out.print("Open ");
    }
}
```

And:

```
public static void main(String[] args) throws Exception {
    try (Book f = new Book()) {
        f.open();
    }
}
```

Which two modifications enable the code to print Open Close?

A. Replace //line1 with:

```
class Book implements AutoCloseable {
```

B. Replace //line1 with:

```
class Book extends Closeable {
```

C. Replace //line1 with:

```
class Book extends Exception {
```

D. At //line2, insert:

```
final void close () { System.out.print("Close"); }
```

E. At //line2, insert:

```
public void close () throws IOException { System.out.print("Close"); }
```

答案 AE

考題 9

Given:

```
static void doStuff()
    throws ArithmeticException, NumberFormatException, Exception {
    if (Math.random() > -1)
        throw new Exception("Do again");
}
```

And:

```
1. public static void main(String[] args) {
2.     try {
3.         doStuff();
4.     } catch (ArithmeticException | NumberFormatException | Exception e) {
5.         System.out.println(e.getMessage());
6.     } catch (Exception e) {
7.         System.out.println(e.getMessage());
8.     }
9. }
```

Which modification enables the code to print Do again?

A. Comment the lines 6, 7.

B. Replace line 4 with:

```
} catch (Exception | ArithmeticException | NumberFormatException e) {
```

C. Replace line 4 with:

```
} catch (ArithmeticException | NumberFormatException e) {
```

D. Replace line 5 with:

```
throw e;
```

答案 C

說明 前後例外物件不可以有繼承關係。

考題 10

Given:

```
static void test(int[] x) {
    try {
        System.out.println(x[1] / x[1] - x[2]);
    } catch (ArithmaticException e) {
        System.out.println("First exception");
    }
    System.out.println("OK");
}

public static void main(String[] args) {
    try {
        int[] iarr = { 10, 10 };
        test(iarr);
    } catch (IllegalArugmentException e) {
        System.out.println("Second exception");
    } catch (Exception e) {
        System.out.println("Third exception");
    }
}
```

What is the result?

A. 0

OK

B. First Exception

OK

C. Second Exception

D. OK

Third Exception

E. Third Exception

答案 E

說明 拋出例外 : java.lang.ArrayIndexOutOfBoundsException: 2

考題 11

Given:

```
class NoNameException extends Exception {  
}  
class AgeOutOfRangeException extends Exception {  
}  
class Person {  
    String name;  
    int age;  
    public Person(String name, int age) throws Exception {  
        if (name == null) {  
            throw new NoNameException();  
        } else if (age <= 0 || age >= 120) {  
            throw new AgeOutOfRangeException();  
        } else {  
            this.name = name;  
            this.age = age;  
        }  
    }  
    public String toString() {  
        return name + ", " + age;  
    }  
}
```

And:

```
1. public static void main(String[] args) {  
2.     Person p1 = new Person("James", 20);  
3.     Person p2 = new Person("Williams", 32);  
4.     System.out.println(p1);  
5.     System.out.println(p2);  
6. }
```

Which change enables the code to print the following?

James, 20

Williams, 32

A. Replacing line 1 with

```
public static void main (String [] args) throws NoNameException, AgeOutOfRangeException {
```

B. Replacing line 1 with

```
public static void main (String [] args) throws Exception {
```

C. Enclosing line 2 and line 3 within a try block and adding:

```
catch(Exception e1) { //...}
catch (NoNameException e2) { //...}
catch (AgeOutOfRangeException e3) { //...}
```

D. Enclosing line 2 and line 3 within a try block and adding:

```
catch (NoNameException e2) { //...}
catch (AgeOutOfRangeException e3) { //...}
```

答案 B

考題 12

Given:

```
class OutOfEnergyException extends Exception {
}

class VehicleBasic {
    void ride() throws OutOfEnergyException {      // line1
        System.out.println("Have a good day!");
    }
}

class SolarVehicle extends VehicleBasic {
    public void ride() throws Exception {           // line2
        super.ride();
    }
}
```

And:

```
public static void main(String[] args) throws OutOfEnergyException, Exception {
    VehicleBasic v = new SolarVehicle();
    v.ride();
}
```

Which modification enables the code fragment to print Have a good day!?

A. Replace //line1 with

```
public void ride() throws OutOfEnergyException {
```

B. Replace //line1 with

```
protected void ride() throws Exception {
```

C. Replace //line2 with

```
void ride() throws Exception {
```

D. Replace //line2 with

```
private void ride() throws OutOfEnergyException {
```

答案 B

考題 13

Given:

```
public static void main(String[] args) {
    int rate = 0;
    String account = "LOAN";
    switch (account) {
        case "RD":
            rate = 5;
            break;
        case "FD":
            rate = 10;
            break;
        default:
            assert false : "No rate for this account";      // line1
    }
    System.out.println("Rate: " + rate);
}
```

And enable assertion by -ea as launch this program.

What is the result?

- A. Rate: 0
- B. Assertion Error is thrown.
- C. No rate for this account
- D. Compilation error at //line1.

答案 B

考題 14

Given:

```
public static void main(String[] args) {
    String names[] = new String[3];
    names[0] = "Jack Brown";
```

```

names[1] = "Marry Red";
names[2] = "Jobes Orange";
try {
    for (String n : names) {
        try {
            String pwd = n.substring(0, 3) + n.substring(6, 10);
            System.out.println(pwd);
        } catch (StringIndexOutOfBoundsException e) {
            System.out.println("StringIndexOutOfBoundsException");
        }
    }
} catch (ArrayIndexOutOfBoundsException e) {
    System.out.println("ArrayIndexOutOfBoundsException");
}
}

```

What is the result?

A. Jacrown

StringIndexOutOfBoundsException

JobOran

B. Jacrown

StringIndexOutOfBoundsException

ArrayIndexOutOfBoundsException

C. Jacrown

StringIndexOutOfBoundsException

D. Jacrown

MarRed

JobOran

答案 A

說明 substring() 方法可能抛出 StringIndexOutOfBoundsException。

考題 15

Given:

```

public static void main(String[] args) {
    int a = 100;
}

```

```
int b = -10;
assert (b >= 0) : "invalid denominator";
int c = a / b;
System.out.println(c);
}
```

What is the result if run the code with the -ea option?

- A. -10
- B. 0
- C. An AssertionError is thrown.
- D. A compilation error occurs.

答案 C

考題 16

Given:

```
class NamingException extends Exception {
}
class AgeOutOfMaxException extends NamingException {
}
class Test {
    public static void register(String name, int age) throws
        NamingException, AgeOutOfMaxException {
        if (name.length() < 6) {
            throw new NamingException();
        } else if (age >= 60) {
            throw new AgeOutOfMaxException();
        } else {
            System.out.println("User is registered.");
        }
    }
    public static void main(String[] args) throws NamingException {
        register("Albert", 60);
    }
}
```

What is the result?

- A. User is registered.
- B. AgeOutOfMaxException is thrown.

- C. NamingException is thrown.
 D. Compilation error in the main method.

答案 B

考題 17

Given:

```
class Scanner implements AutoCloseable {
    public void close() throws Exception {
        System.out.println("Scanner closed.");
    }
    public void scan() throws Exception {
        System.out.println("Scan.");
        throw new Exception("Unable to scan.");
    }
}
class Printer implements AutoCloseable {
    public void close() throws Exception {
        System.out.println("Printer closed.");
    }
    public void print() {
        System.out.println("Print.");
    }
}
```

And:

```
public static void main(String[] args) {
    try (Scanner ir = new Scanner(); Printer iw = new Printer()) {
        ir.scan();
        iw.print();
    } catch (Exception e) {
        System.out.println(e.getMessage());
    }
}
```

What is the result?

- A. Scan.
 Printer closed.
 Scanner closed.
 Unable to scan.

B. Scan.

Scanner closed.

Unable to scan.

C. Scan.

Unable to scan.

D. Scan.

Unable to scan.

Printer closed.

答案 A

考題 18

Which two are Java Exception classes?

- A. SecurityException
- B. DuplicatePathException
- C. IllegalArgumentException
- D. TooManyArgumentsException
- E. IndexOutOfBoundsException

答案 AC

Java I/O基礎

-
- | 8.1 基礎 I/O
 - | 8.2 由主控台讀寫資料
 - | 8.3 Channel I/O
 - | 8.4 使用序列化技術讀寫物件
 - | 8.5 認證考試命題範圍

8.1 基礎 I/O

Java 將 I/O (input/output)，亦即輸入 / 輸出的概念，以 stream(水流或串流)的抽象概念表達；因為當輸入 / 輸出的行為發生時，就好比串流的流動，要流入(輸入)某個地方，或流出(輸出)某個地方。串流需要有來源及目的，可以是主控台視窗、檔案、資料庫、網路，或是其他程式。藉由建立來源端和目的端的串流物件，就可以串起資料的流動。

8.1.1 何謂 I/O ?

Java 為了讓資料在某些裝置中可以輸入 / 輸出 (input/output，簡稱 I/O)，提供了一系列的類別。在開始使用這些類別之前，需要先有基本認知：

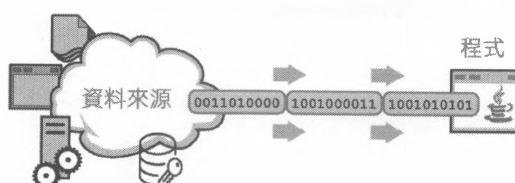
1. 資料的進出像是水流 / 串流：

- 由來源流向目的，具有方向性，在 Java 中稱為「stream(串流)」。來源端點或目的端點可以是作業系統檔案、其他輸出輸入裝置、應用程式或是記憶體陣列等。
- 流動的內容主要分兩類：位元 (byte)、字元 (character)。

2. 水流的流動，若提供管道，稱為「channel」。在概念上若使用 channel 支援 I/O，會更有效率。在章節 9.3 中有進一步說明。

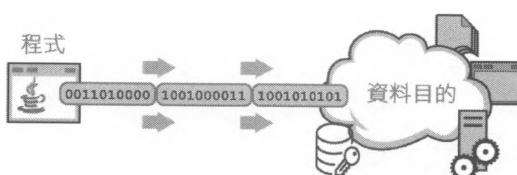
Stream 如水流具有方向性，且以 Java 程式為區分基準，可分二種：

1. 若方向是流入 Java 程式，則稱為「輸入 (input) 串流」或「來源 (source) 串流」：



❖ 圖 8-1 Java Input Stream

2. 若方向是流出 Java 程式，則稱為「輸出 (output) 串流」或「目的 (sink) 串流」：



❖ 圖 8-2 Java Output Stream

依程式開發常用的三種端點來區隔，可以分成：

1. 檔案 (files) 和資料夾 (directories)。
2. 主控台 (console)，分成標準輸入 (standard-in) 和標準輸出 (standard-out)。
3. Socket 程式 (連線遠端系統，需指定 port 通訊)。

8.1.2 處理串流的類別

根據串流內的資料分類，加上流動方向，處理類別可以區分為四大類，這些都是抽象類別：

◆ 表 8-1 stream 類別分類

方向	位元 (byte)	字元 (character)
輸入 Java	InputStream	Reader
輸出 Java	OutputStream	Writer

以下表列出這些抽象串流類別的主要方法與功能。

InputStream 類別：

◆ 表 8-2 InputStream 類別主要方法

基本方法	int read() 功能：每次讀取 1 個 byte。
	int read(byte[] buffer) 功能：每次讀取 1 個 byte[]。
	int read(byte[] buffer, int offset, int length) 功能：每次讀取 1 個 byte[]，且可以指定偏移量 (offset) 和讀取長度 (length)。
其他方法	void close() 功能：關閉 stream。
	int available() 功能：有多少個 bytes 可供讀取。
	long skip(long n) 功能：讀取時略過 n 個 bytes。
	boolean markSupported() void mark(int readlimit) void reset() 功能：合併用於改變檔案中的讀取位置，特別是回到過去某個指定的讀取位置。又稱 push-back 操作。

OutputStream 類別：

❖ 表 8-3 OutputStream 類別主要方法

基本方法	void write(int c) 功能：將 int 寫入 OutputStream。
	void write(byte[] buffer) 功能：將 byte[] 寫入 OutputStream。
	void write(byte[] buffer, int offset, int length) 功能：寫入 byte[] 到 OutputStream，且指定長度 (length) 和偏移量 (offset)。
其他方法	void close() 功能：關閉 stream。
	void flush() 功能：強制將 OutputStream 中的資料寫入目的地。

Reader 類別：

❖ 表 8-4 Reader 類別主要方法

基本方法	int read() 功能：每次讀取 1 個 char。
	int read(char[] buffer) 功能：每次讀取 1 個 char[]。
	int read(char[] buffer, int offset, int length) 功能：每次讀取 1 個 char[]，且指定偏移量 (offset) 和讀取長度 (length)。
其他方法	void close() 功能：關閉 stream。
	boolean ready() 功能：確認 stream 是否已經準備好進行資料讀取。
	long skip(long n) 功能：讀取時略過 n 個 chars。
	boolean markSupported() void mark(int readAheadLimit) void reset() 功能：合併用於改變檔案中的讀取位置，特別是回到過去某個指定的讀取位置。又稱 push-back 操作。

Writer 類別：

❖ 表 8-5 Writer 類別主要方法

基本方法	void write(int c) 功能：將 int 寫入 Writer。
	void write(char[] buffer) 功能：將整個 char[] 寫入 Writer。
	void write(char[] buffer, int offset, int length) 功能：寫入 char[] 到 Writer，且指定長度 (length) 和偏移量 (offset)。
其他方法	void close() 功能：關閉 stream。
	void flush() 功能：強制將 Writer 中的資料寫入目的地。

以下範例顯示如何使用 InputStream/OutputStream 類別處理位元串流：

❶ 範例「/OCP/src/course/c08/CopyByteStream.java」

```

1  public class CopyByteStream {
2      public static void main(String[] args) {
3          String source = "";
4          String target = "";
5          byte[] b = new byte[128];
6          int bLen = b.length;
7          try (FileInputStream fis = new FileInputStream(source);
8               FileOutputStream fos = new FileOutputStream(target)) {
9              System.out.println("Will copy bytes: " + fis.available());
10             int read = 0;
11             while ((read = fis.read(b)) != -1) {
12                 if (read < bLen) {
13                     fos.write(b, 0, read);
14                 } else {
15                     fos.write(b);
16                 }
17             }
18         } catch (FileNotFoundException fne) {
19             fne.printStackTrace();
20         } catch (IOException ioe) {
21             ioe.printStackTrace();
22         }
23     }
24 }
```

❷ 說明

11	每次都讀入一個byte[128]，回傳 -1 表示讀完。
12	剩下不滿byte[128]時。
13	寫入位置為0 ~ 變數read。

以下範例顯示如何使用 Reader/Writer 相關類別處理字元串流：

❶ 範例「/OCP/src/course/c08/CopyCharStream.java」

```

1  public class CopyCharStream {
2      public static void main(String[] args) {
3          String source = "";
4          String target = "";
5          char[] c = new char[128];
6          int cLen = c.length;
```

```

7      try (FileReader fr = new FileReader(source);
8          FileWriter fw = new FileWriter(target)) {
9              int read = 0;
10             while ((read = fr.read(c)) != -1) {
11                 if (read < cLen) {
12                     fw.write(c, 0, read);
13                 } else {
14                     fw.write(c);
15                 }
16             }
17         } catch (FileNotFoundException fne) {
18             fne.printStackTrace();
19         } catch (IOException ioe) {
20             ioe.printStackTrace();
21         }
22     }
23 }
```

說明

10	每次都讀入一個 char[128]，回傳 -1 表示讀完。
11	剩下不滿 char [128] 時。
12	寫入位置為 0~ 變數 read 的長度。

8.1.3 串流類別的串接

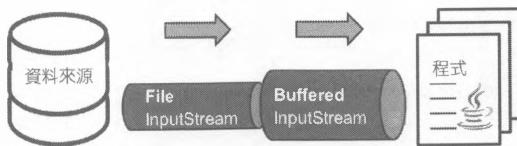
處理 I/O 的程式，很少使用單一串流物件完成；經常將多個串流物件組成「串接 (chain)」，藉由不同的串流物件提供不同的功能，一起完成。此為「裝飾者設計模式 (Decorator Pattern)」的應用。這些類別常見的有：

✿表 8-6 常見串接類別

功能	字元串流 (Character Streams)	位元串流 (Byte Streams)
Buffering (緩衝)	BufferedReader BufferedWriter	BufferedInputStream BufferedOutputStream
Filtering (過濾)	FilterReader FilterWriter	FilterInputStream FilterOutputStream
Conversion (位元轉換為字元)	InputStreamReader OutputStreamWriter	
Object serialization (物件序列化)		ObjectInputStream ObjectOutputStream
Data conversion (資料型態轉換)		DataInputStream DataOutputStream

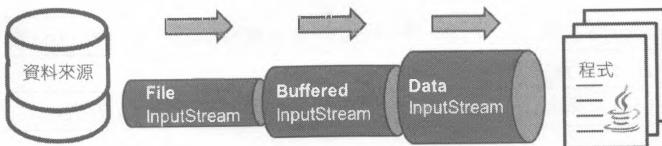
功能	字元串流 (Character Streams)	位元串流 (Byte Streams)
Counting(計算行數)	LineNumberReader	LineNumberInputStream
Printing(列印)	PrintWriter	PrintStream

如下圖。一開始使用 FileInputStream 類別將資料由檔案中讀出，然後可以加上一段 BufferedInputStream 類別，則資料讀取就增加緩衝的功能 (資料讀出後先放到記憶體裡，蓄積一定份量後再寫入目的地，避免 I/O 忙碌) :



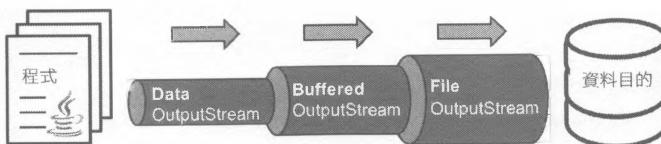
❖ 圖 8-3 使用二個串流類別組成輸入串流

若再加上 DataInputStream 類別，則資料可以被轉換成不同型態 (可以是 int, double 等) ，再進入 Java 程式。如下圖：



❖ 圖 8-4 使用三個串流類別組成輸入串流

由程式輸出到其他資料目的端的時候，自然也可以使用串接輸出串流，如下圖：



❖ 圖 8-5 使用三個串流類別組成輸出串流

以下範例將 FileReader/FileWriter 加上緩衝 (buffered) 的功能，讓讀寫更有效率：

❸ 範例「 /OCP/src/course/c08/CopyBufferedStream.java 」

```

1  public class CopyBufferedStream {
2      public static void main(String[] args) {
3          String source = "";
4          String target = "";
  
```

```

5      try (BufferedReader in =
6          new BufferedReader(new FileReader(source)));
7      BufferedWriter out =
8          new BufferedWriter(new FileWriter(target))) {
9      String line = "";
10     while ((line = in.readLine()) != null) {
11         out.write(line);
12         out.newLine();
13     }
14 } catch (FileNotFoundException fne) {
15     fne.printStackTrace();
16 } catch (IOException ioe) {
17     ioe.printStackTrace();
18 }

```

說明

5	使用 BufferedReader 類別裝飾 FileReader 類別。
6	使用 BufferedWriter 類別裝飾 FileWriter 類別。
8	BufferedReader 每次讀一行（內容 + 换行符號），但只有內容被保留在變數 line 裡。
9	BufferWriter 遂行寫出資料。
10	BufferWriter 必須自己每行加入新的換行符號，所以呼叫 newLine() 方法。

以下範例顯示使用 BufferedReader 類別裝飾串流物件後，得到「返回檔案已經讀取的位置」的能力，又稱為「push-back 操作」：

範例「/OCP/src/course/c08/BufferedReaderMarkResetDemo.java」

```

1 public class BufferedReaderMarkResetDemo {
2     public static void main(String[] args) throws Exception {
3         InputStream is = null;
4         InputStreamReader isr = null;
5         BufferedReader br = null;
6         try {
7             System.out.println("Is markSupported on... ");
8             String f = System.getProperty("user.dir") +
9                         "\\\src\\\\course\\\\c08\\\\test.txt"; // 檔案內容為：ABCDEF
10            is = new FileInputStream(f);
11            System.out.println("FileInputStream? " + is.markSupported());
12            // create new input stream reader
13            isr = new InputStreamReader(is);

```

```

13     System.out.println("InputStreamReader? " + isr.markSupported());
14     // create new buffered reader
15     br = new BufferedReader(isr);
16     System.out.println("BufferedReader? " + br.markSupported());
17     System.out.println("-----");
18     // reads and prints BufferedReader
19     System.out.println((char) br.read()); // 讀取字元A
20     System.out.println((char) br.read()); // 讀取字元B
21     br.mark(99); // 做記號
22     System.out.println("mark() invoked");
23     System.out.println((char) br.read()); // 讀取字元C
24     System.out.println((char) br.read()); // 讀取字元D
25     br.reset(); // 回到做記號的地方
26     System.out.println("reset() invoked");
27     System.out.println((char) br.read()); // 讀取字元為C, 非E
28 } catch (Exception e) {
29     e.printStackTrace();
30 } finally {
31     if (is != null)
32         is.close();
33     if (isr != null)
34         isr.close();
35     if (br != null)
36         br.close();
37 }
38 }
39 }
```

結果

```

Is markSupported on...
FileInputStream? false
InputStreamReader? false
BufferedReader? true
-----
A
B
mark() invoked
C
D
reset() invoked
C
```

 **說明**

15	逐類別測試後發現，只有 BufferedReader 支援 mark 機制。
19~27	BufferedReader 使用 read() 方法逐字讀取檔案內容。讀取完畢後，使用 mark() 方法做記號。在繼續往下讀取內容的過程中，若呼叫 reset() 方法，就會回到做記號的地方。

8.2 由主控台讀寫資料

8.2.1 主控台的 I/O

java.lang.System 類別裡有三個 static 欄位，分別為：

◆表 8-7 使用主控台輸入輸出的 System 類別欄位

欄位	欄位型別	功能
System.out	PrintStream	將訊息輸出至主控台(console)上，又稱「標準輸出(standard output)」。可以接受由主控台再經「>」或「>>」的「重新導向指令」，將輸出內容導向至另一個檔案。
System.in	InputStream	由主控台(console)接收來自鍵盤或其它來源的訊息輸入，又稱「標準輸入(standard input)」。
System.err	PrintStream	和 System.out 都是將訊息輸出至主控台(console)。但因為主要用於輸出錯誤訊息，比較急迫，必須立即顯示，因此「重新導向指令」無效，依然顯示在主控台上。 若使用 Eclipse，則輸出顏色為紅色，以為警訊。

8.2.2 使用標準輸出方法

標準輸出的 PrintStream 類別有 2 個膾炙人口的方法：

1. println() 方法最後會自動輸出換行符號(\n)；print() 則無。
2. println()、print() 對於大部分基本型別：boolean、char、int、long、float 和 double，及參考型別：char[]、Object、String 等都有多載 (overloading) 的方法支援。
3. 傳入參考型別時，將呼叫該物件的 toString() 方法。

8.2.3 使用標準輸入由主控台取得輸入資料

以下範例顯示如何使用「標準輸入」由主控台取得使用者輸入的資料：

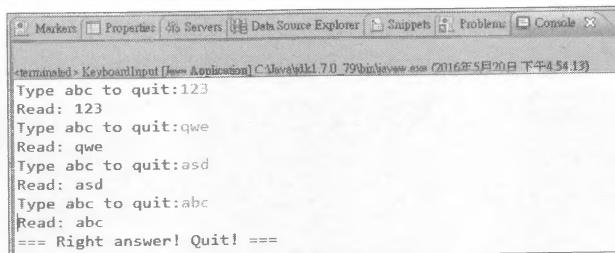
◎ 範例「/OCP/src/course/c08/KeyboardInput.java」

```

1  public class KeyboardInput {
2      public static void main(String[] args) {
3          try (BufferedReader in =
4              new BufferedReader(
5                  new InputStreamReader(System.in))) {
6              String s = "";
7              while (s != null) {
8                  System.out.print("Type abc to quit:");
9                  s = in.readLine();
10                 if (s != null) {
11                     s = s.trim();
12                 }
13                 System.out.println("Read: " + s);
14                 if (s.equals("abc")) {
15                     System.out.println("==== Right answer! Quit! ===");
16                     System.exit(0);
17                 }
18             } catch (IOException e) {
19                 e.printStackTrace();
20             }
21         }
22     }

```

本例可以使用 Eclipse 執行。執行過程中不斷輸入字串，點擊 **Enter** 鍵後，程式會比對輸入資料是否和字串 abc 相同；如此反覆進行，直到相同才結束程式：



◆ 圖 8-6 使用 Eclipse 支援標準輸入顯示

◎ 說明

3	將 <code>System.in</code> 物件以 <code>InputStreamReader</code> 和 <code>BufferedReader</code> 先後包覆。 <code>BufferedReader</code> 具有一次讀取一行輸入的能力。
7	一旦使用者完成資料輸入後，點擊 Enter 鍵，會觸發 <code>readLine()</code> 方法，Java 可以取得主控台輸入的資料。
14	滿足條件後離開程式。

8.2.4 java.io.Console 類別介紹

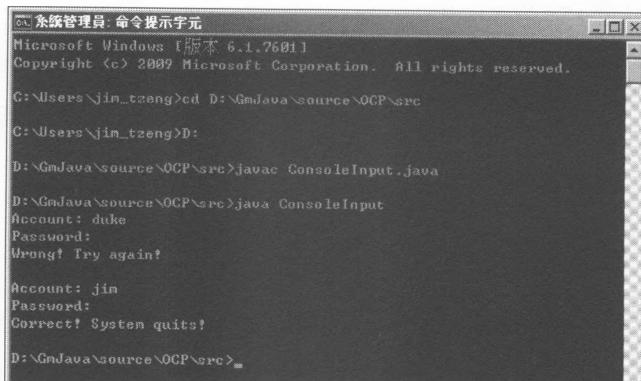
除了使用前述 System.in 物件取得主控台的標準輸入外，也可以使用「java.io.Console 物件」，可由 java.lang.System 取得，如以下範例。但本範例只能由主控台的指令列輸入，無法使用 Eclipse。

範例「/OCP/src/ConsoleInput.java」

```

1  public class ConsoleInput {
2      public static void main(String[] args) {
3          Console cons = System.console();
4          boolean userValid = false;
5          if (cons != null) {
6              String account;
7              String password;
8              do {
9                  account = cons.readLine("%s", "Account: ");
10                 password = new String(cons.readPassword("%s", "Password: "));
11                 if (account.equals("jim") & password.equals("password")) {
12                     System.out.println("Correct! System quits!");
13                     userValid = true;
14                 } else {
15                     System.out.println("Wrong! Try again!\n");
16                 }
17             } while (!userValid);
18         }
19     }
20 }
```

執行過程與結果：



❖ 圖 8-7 使用 Console 物件取得主控台指令列的輸入 (input)

 說明

3	使用 System.console() 方法取得 Console 物件。
9	Console 物件的 readLine() 方法可以取得指令列輸入的資料。
10	Console 物件的 readPassword() 方法除了可以取得指令列輸入的資料，還有隱藏使用者輸入內容的貼心設計。

8.3 Channel I/O

Channel 有通道、頻道的意思，可以指兩個設備之間傳送資訊所經過的通路或連接。於 JDK 1.4 中導入，屬於 java.nio 套件(非 java.io)。功用是可以一次大量讀入位元和字元，不需要以迴圈每次讀取少量內容，因此程式更簡潔，程式效能也更好。如下：

 課堂小秘訣

我們將 Java 的 I/O 比喻成水流或串流(stream)，概念上，可以將 Channel I/O 想成讓 stream 流動在專屬的管道，或是加上水管，當然水流會更順暢！

 範例「/OCP/src/course/c08/CopyByteChannel.java」

```

1 import java.io.*;
2 import java.nio.ByteBuffer;
3 import java.nio.channels.FileChannel;
4 public class CopyByteChannel {
5     public static void main(String[] args) {
6         String source = "", target = "";
7         try (FileChannel in = new FileInputStream(source).getChannel();
8              FileChannel out = new FileOutputStream(target).getChannel()) {
9             ByteBuffer buff = ByteBuffer.allocate((int) in.size());
10            in.read(buff);
11            buff.position(0);
12            out.write(buff);
13            catch (FileNotFoundException f) {
14                f.printStackTrace();
15            } catch (IOException i) {
16                i.printStackTrace();
17            }
18        }
19    }

```

 **說明**

9	建立和檔案 size 大小相同的 ByteBuffer 物件。
10	將實體檔案一次全數讀入 ByteBuffer 物件中。
11	將 ByteBuffer 裡的標示位置移到最前面。
12	將 ByteBuffer 裡的資料全數倒到 FileChannel 輸出物件裡；再由 FileChannel 形成輸出檔案。因為 FileChannel 的 read() 和 write() 方法都是透過 ByteBuffer 物件一次搞定，所以不用迴圈 (loop) 分次處理。

8.4 使用序列化技術讀寫物件

8.4.1 了解序列化技術

Java 裡的資料保存

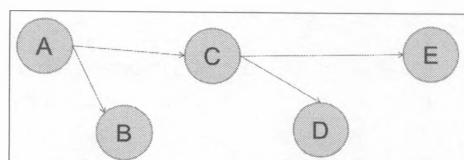
將資料保存於永久性的儲存硬體中，稱為「persistence」：

- 支援 persistence 的 Java 物件可以儲存於本機硬體，或是經由網路到另一個硬體裝置；反之只能存活於執行中的 JVM。
- 支援 persistence 的物件必須 implements 「java.io.Serializable」，讓 Java 知道。java.io.Serializable 是一個「maker interface」，因此沒有任何方法需要實作。
- Java 將物件儲存於硬體成為實體檔案的標準機制，稱為「序列化 (serialization)」，未來可用來建構物件的副本或是還原物件狀態。

序列化和物件圖譜

當物件被序列化時，只有「欄位的值」會被保留，因為欄位的值包含資料，也代表物件的狀態。若欄位是參考型別，且被欄位參照的物件也支援序列化，則該欄位物件將一併被序列化。

物件的欄位可以參照其他物件，被參照的物件又可以再參照更多物件…，如此形成一個樹狀結構，稱為「物件圖譜 (Object Graphs)」：



❖ 圖 8-8 物件圖譜

不需要參加序列化的欄位

物件進行序列化時，預設所有欄位都會一併進行序列化。若欄位所屬類別未實作 `java.io.Serializable`，將會丟出 `NotSerializableException` 例外，並中斷執行。

若物件的欄位只是記錄當下系統狀態的某些資訊，如目前時間，屬「`transient`(短暫的)」資訊，因此不需要在序列化過程中被保留，重建副本時亦不需要回復。這類欄位不需要參與序列化流程，可以加上「`transient`」宣告。

此外，「`static`」欄位和物件狀態無關，其值在物件序列化過程中也不會被保留。

如下：

範例「/OCP/src/course/c08/SerializeExample/Order.java」

```

1  public class Order implements Serializable {
2      private Set<Shirt> shirts = new HashSet<>();
3      static int staticField = 100;
4      transient int transientField = 100;
5  }
```

宣告 `transient` 和 `static` 的欄位在「反序列化(還原)」時：

1. `static` 欄位會得到類別內原本定義的宣告值。
2. `transient` 欄位會得到該型態的預設值。

8.4.2 定義物件保存的版本號碼

「序列化」僅將物件當下的狀態(欄位值)保留，並未保留類別架構；因此「反序列化」是將「過去物件某個狀態」，再搭配「目前類別架構」進行還原。若還原時發現「過去物件狀態」和「目前類別架構」並未一致，如過去的某個欄位現在已經不存在；或類別後來新增必要欄位，但序列化時未有該欄位等的狀況，都可能產生不預期的問題或錯誤。

為了物件序列化後的檔案可以順利以反序列化還原為物件，必須：

1. 在可序列化的類別上先定義一個欄位「`serialVersionUID`」，作為版本控管號碼。每次增減類別欄位時，都應該同步修改版本號碼，並且記錄。該欄位必須宣告為「`static`」且「`long`」。如：

```
private static final long serialVersionUID = 1L;
```

2. 假設目前版本號為 1，則序列化後得到的檔案內存版本號也會是 1；後來類別增減了欄位，修改版本號為 2。若程式開發人員以「版本號為 2 的最新類別定義」去還原「序列化時版本號為 1 的檔案」，Java 就會丟出 `InvalidClassException`，阻止繼續還原。這是一種預先保護機制，避免檔案和類別不一致時產生不預期的問題或錯誤。
3. 若類別已經 implements `Serializable`，但未主動宣告 `serialVersionUID`，則 Java 預設將主動宣告並提供欄位值。該值將考慮開發時的環境因素如 IDE 工具或 Java 版本而計算出一個複雜的長整數，如：

```
private static final long serialVersionUID = 3696676879791539369L;
```

該值在每次類別程式碼異動後的編譯階段，即便只是某方法內的字串微調整，也可能因環境再改變，而自動改變版本號，導致先前序列化的檔案均因版本號改變，而無法還原物件。因此建議應該自己宣告 `serialVersionUID` 欄位。

8.4.3 序列化和反序列化範例

在接下來的序列化和反序列化範例，`Order` 類別含數個 `Shirt(s)` 類別，我們將以 `Order` 物件進行序列化產出檔案，再將該檔案還原回物件。過程中設計了幾個特殊情境：

1. `Shirt(s)` 的價錢會因時因地改變，因此在序列化過程中，不需要特別保存，以 `transient` 宣告。
2. Java 賦予物件可以控制自身序列化和反序列化的流程。因此我們在序列化時將額外寫入一個 `java.util.Date` 物件，並在反序列化時將它讀出，如此可以知道進行序列化時間點。
3. 在反序列化時，我們希望 `Shirt(s)` 的價錢可以參考成本價，再加上 50 元的管銷費用，在還原完時一併設定完成。

如此，完整範例在路徑「/OCP/src/course/c08/SerializeExample」下。

範例「/OCP/src/course/c08/ser/Shirt.java」

```
1  public class Shirt implements Serializable {
2
3      private static final long serialVersionUID = 1L;
4      private String brand;
5      private int quantity;
6      private double cost;
7      private transient double price;
```

```

9   public Shirt(String brand, int quantity, double cost) {
10      this.brand = brand;
11      this.quantity = quantity;
12      this.cost = cost;
13      this.price = 2 * cost;
14  }
15
16  // This method is called post-serialization
17  private void readObject(ObjectInputStream ois)
18      throws IOException, ClassNotFoundException {
19      ois.defaultReadObject();
20      // perform other initialization
21      this.price = this.cost + 50;
22  }
23
24  @Override
25  public String toString() {
26      return "Shirt: " + this.brand + "\n"
27          + "Quantity: " + this.quantity + "\n"
28          + "Cost: " + this.cost + "\n"
29          + "Price: " + this.price + "\n"
30          + "-----\n";
31  }
32 }
```

說明

17	要修改 Java 反序列化（將物件自檔案中讀出）的流程，必須定義本方法。
18	ois.defaultReadObject() 為物件原本的反序列化流程，因此仍需呼叫。
20	自檔案中讀出 / 還原物件後，將 price 欄位加上 50 元。

範例「/OCP/src/course/c08.ser/Order.java」

```

1  public class Order implements Serializable {
2
3      private static final long serialVersionUID = 1L;
4      private List<Shirt> shirts = new ArrayList<>();
5      static int staticField = 100;
6      transient int transientField = 100;
7
8      public Order(Shirt... shirts) {
9          for (Shirt s : shirts) {
10              this.shirts.add(s);
11          }
12          staticField = 99;
```

```

13         transientField = 99;
14         System.out.println("--- Constructor is launched ---");
15     }
16
17     private void writeObject(ObjectOutputStream oos) throws IOException {
18         oos.defaultWriteObject();
19         // keep the serialization date
20         Date now = new Date();
21         oos.writeObject(now);
22         System.out.println("\nSerialized at: " + now + "\n");
23     }
24
25     private void readObject(ObjectInputStream ois)
26             throws IOException, ClassNotFoundException {
27         ois.defaultReadObject();
28         System.out.println(
29                 "\nRestored from date: " + (Date) ois.readObject());
30         System.out.println("Restored at: " + new Date() + "\n");
31     }
32
33     public String toString() {
34         StringBuilder sb = new StringBuilder("Order Summary ===\n");
35         for (Shirt s : shirts) {
36             sb.append(s);
37         }
38         sb.append("staticField = " + staticField);
39         sb.append("\ntransientField = " + transientField);
40         sb.append("\n-----");
41         return sb.toString();
42     }
43 }
```

 說明

17	要修改 Java 序列化（將物件寫入檔案）的流程，必須使用本方法。
18	第一行就呼叫 <code>oos.defaultWriteObject()</code> ，進行物件原本的序列化流程。
20	建立日期物件。
21	將日期物件寫入檔案。
25	要修改 Java 反序列化（將檔案還原物件）的流程，必須定義本方法。
26	第一行就呼叫 <code>ois.defaultReadObject()</code> ，進行物件原本的反序列化流程。
27	將序列化時的日期物件讀出。讀出時為 <code>Object</code> 型態，必須再轉型回 <code>Date</code> 型態。
28	印出還原時的日期。

範例「/OCP/src/course/c08/ser/SerializeOrder.java」

```

1  public class SerializeOrder {
2
3      public static void main(String[] args) {
4
5          String output = System.getProperty("user.dir") +
6                          "\\\src\\\course\\\c08\\\ser\\\file\\\Order.ser";
7
8          serialization(output);
9
10         System.out.println("\n-----\n");
11
12         deSerialization(output);
13     }
14
15     private static void serialization(String output) {
16         // Create a shirts Order
17         Shirt s1 = new Shirt("Brand1", 100, 100);
18         Shirt s2 = new Shirt("Brand2", 100, 200);
19         Shirt s3 = new Shirt("Brand3", 100, 300);
20         Order o = new Order(s1, s2, s3);
21
22         Order.staticField = 22;
23         // Write out the Order
24         try (FileOutputStream fos = new FileOutputStream(output);
25              ObjectOutputStream out = new ObjectOutputStream(fos)) {
26             out.writeObject(o);
27         } catch (IOException i) {
28             i.printStackTrace();
29         }
30         System.out.println("== Before Serialized, " + o);
31     }
32
33     private static void deSerialization(String output) {
34         // Read the Order back in
35         try (FileInputStream fis = new FileInputStream(output);
36              ObjectInputStream in = new ObjectInputStream(fis)) {
37             Order restoredOrder = (Order) in.readObject();
38             System.out.println("== After Serialized, " + restoredOrder);
39         } catch (ClassNotFoundException | IOException i) {
40             i.printStackTrace();
41         }
42     }
}

```

結果

```
--- Constructor is launched ---  
  
Serialized at: Fri May 27 08:13:08 CST 2016  
  
==== Before Serialization, Order Summary ====  
Shirt: Brand1  
Quantity: 100  
Cost: 100.0  
Price: 200.0  
-----  
Shirt: Brand2  
Quantity: 100  
Cost: 200.0  
Price: 400.0  
-----  
Shirt: Brand3  
Quantity: 100  
Cost: 300.0  
Price: 600.0  
-----  
staticField = 22  
transientField = 99  
-----
```

```
-----  
  
Restored from date: Fri May 27 08:13:08 CST 2016  
Restored at: Fri May 27 09:08:50 CST 2016
```

```
==== After Serialization, Order Summary ====  
Shirt: Brand1  
Quantity: 100  
Cost: 100.0  
Price: 150.0  
-----  
Shirt: Brand2  
Quantity: 100  
Cost: 200.0  
Price: 250.0  
-----  
Shirt: Brand3  
Quantity: 100  
Cost: 300.0  
Price: 350.0
```

```
-----  
staticField = 100  
transientField = 0  
-----
```

說明

1. Shirt 物件生成時的 price 都是 cost 的 2 倍，還原時在 readObject() 方法被改成 price=cost+50。
2. Order 物件序列化時在 writeObject() 方法被額外寫入的日期，還原時在 readObject() 方法可以一併輸出。
3. static 欄位會得到類別內原本定義的宣告值。
4. transient 欄位會得到該型態的預設值。

本範例執行時，建議執行序列化方法後，先終止 JVM，再啓動程式執行反序列化方法。若將兩者接連執行，會產生 static 欄位值前後不變的情況，造成混淆。

8.5 認證考試命題範圍

依據甲骨文公布的考試範圍 (https://education.oracle.com/java-se-8-programmer-ii/pexam_1Z0-809)，和本章相關的主題有：

Java I/O Fundamentals

1. Read and write data from the **console**.
2. Use **BufferedReader**, **BufferedWriter**, **File**, **FileReader**, **FileWriter**, **InputStream**, **OutputStream**, **ObjectOutputStream**, **ObjectInputStream**, and **PrintWriter** in the **java.io** package.

本章擬真試題實戰

考題 1

Given the existing destination file, and source file only 100 bytes long, and the code fragment:

```
public static void process(String from, String to) {  
    try (InputStream fis = new FileInputStream(from);  
         OutputStream fos = new FileOutputStream(to)) {  
        byte[] buff = new byte[256];  
        int i;  
        while ((i = fis.read(buff)) != -1) {  
            fos.write(buff, 0, i); // line1  
        }  
    } catch (IOException e) {  
        e.printStackTrace();  
    }  
}
```

What is the result?

- A. Overrides the content of the destination file with the source file content
- B. Appends the content of the source file to the destination file after a new line
- C. Appends the content of the source file to the destination file without a break in the flow
- D. Throws a runtime exception at //line1

答案 A

考題 2

Given:

```
public static void main(String[] args) {  
    String file = "somefile.txt";  
    try (BufferedReader br = // line 4  
         new BufferedReader(new FileReader(file))) {
```

```

String line = "";
int count = 1;
line = br.readLine(); // line 7
do {
    line = br.readLine();
    System.out.print(count + ":" + line);
} while (line != null);
} catch (IOException | FileNotFoundException e) {
    e.printStackTrace();
}
}
}

```

What is the result, if the file somefile.txt does not exist?

- A. A runtime exception is thrown at line 4
- B. A runtime exception is thrown at line 7
- C. Creates a new file and prints no output
- D. Compilation fails

答案 D

說明 使用 multi-catch 語法，符號「|」的前後類別不能有繼承關係。

考題 3

Given:

```

public static void main(String args[]) {
    try (BufferedReader br =
        new BufferedReader(new FileReader("report1.dat"))) { //line1
        String line;
        while ((line = br.readLine()) != null) {
            System.out.println(line);
        }
        br.close();
        br = new BufferedReader(new FileReader("report2.dat")); // line2
        while ((line = br.readLine()) != null) {
            System.out.println(line);
        }
    } catch (IOException e) {
        System.err.println(e.getClass());
    }
}

```

What is the result, if the file report1.dat does not exist?

- A. Compilation fails only at //line1
- B. Compilation fails only at //line2
- C. Compilation fails both at //line1 and //line2
- D. Class java.io.IOException
- E. Class java.io.FileNotFoundException

答案 B

說明 編譯失敗的訊息是：The resource br of a try-with-resources statement cannot be assigned。語法 try-with-resources 內的資源不可以重複指定。

考題 4

Given:

```
class Base {  
    public void printFile() throws IOException {  
        FileReader fr = new FileReader("file.txt");  
        BufferedReader br = new BufferedReader(fr);  
        String line;  
        while ((line = br.readLine()) != null) {  
            System.out.println(line);  
        }  
    }  
}  
  
class Derived extends Base {  
    public void printFile() throws Exception {  
        super.printFile();  
        System.out.println("Success");  
    }  
}
```

And:

```
public static void main(String[] args) {  
    try {  
        new Derived().printFile();  
    } catch (Exception e) {  
        System.err.println(e.getClass());  
    }  
}
```

If the file file.txt does not exist, what is the result?

- A. An empty file is created and success is printed
- B. class java.io.FileNotFoundException
- C. class java.io.IOException
- D. class java.lang.Exception
- E. Compilation fails

答案 E

說明 子類別覆寫方法時，不能拋出父類別方法的例外父類別。

考題 5

Given:

```
public static void main(String[] args) {
    try (BufferedReader br = new BufferedReader(new FileReader("file.txt"))) {
        String line = "";
        int count = 1;
        br.mark(1);
        line = br.readLine();
        System.out.println(count + ":" + line);
        line = br.readLine();
        System.out.println(++count + ":" + line);
        br.reset();
        line = br.readLine();
        System.out.print(++count + ":" + line);
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

And: file.txt that contains

```
First
Second
Third
```

What is the result?

- A. 1:First
2:Second
3:Third
- B. 1:First
2:Second
3:First
- C. 1:First
2:First
3:First
- D. Throws an IOException
- E. Compilation fails

答案 B

說明 參照本書「8.1.3 串流類別的串接」及範例「/OCP/src/course/c08/BufferedReaderMarkResetDemo.java」

考題 6

Given:

```
class Super {  
    BufferedReader br;  
    String record;  
    public void process() throws FileNotFoundException {  
        br = new BufferedReader(new FileReader("file.txt"));  
    }  
}  
class Sub extends Super {  
    // insert code here.  
}  
public class Test {  
    public static void main(String[] args) {  
        try {  
            new Sub().process();  
        } catch (Exception e) { }  
    }  
}
```

Which code fragment inserted at //insert code here, enables the code to compile?

```

A. public void process() throws FileNotFoundException, IOException {
    super.process();
    while ((record = br.readLine()) != null) {
        System.out.println(record);
    }
}

B. public void process() throws IOException {
    super.process();
    while ((record = br.readLine()) != null) {
        System.out.println(record);
    }
}

C. public void process() throws Exception {
    super.process();
    while ((record = br.readLine()) != null) {
        System.out.println(record);
    }
}

D. public void process() {
    try {
        super.process();
        while ((record = br.readLine()) != null) {
            System.out.println(record);
        }
    } catch (IOException | FileNotFoundException e) {
    }
}

E. public void process() {
    try {
        super.process();
        while ((record = br.readLine()) != null) {
            System.out.println(record);
        }
    } catch (IOException e) {
    }
}

```

答案 E

- 說明** 選項 A：子類別覆寫方法時，不能拋出父類別方法的例外父類別(IOException)。
 選項 B：子類別覆寫方法時，不能拋出父類別方法的例外父類別(IOException)。
 選項 C：子類別覆寫方法時，不能拋出父類別方法的例外父類別(Exception)。
 選項 D：使用 multi-catch 語法，符號「|」的前後類別不能有繼承關係。

考題 7

Given:

```
public static void main(String[] args) {  
    String file1 = "file.txt";  
    String file2 = "newfile.txt";  
    try (BufferedReader in = new BufferedReader(new FileReader(file1));  
         BufferedWriter out = new BufferedWriter(new FileWriter(file2))) {  
        String line = "";  
        int count = 1;  
        while (line != null) {  
            out.write(count + ":" + line);  
            out.newLine();  
            count++;  
            line = in.readLine();  
        }  
    } catch (IOException e) {  
        e.printStackTrace();  
    }  
}
```

And: file.txt that contains

```
First  
Second  
Third
```

What is the result?

A. newfile.txt contains:

```
1:  
2:First  
3:Second  
4:Third
```

B. newfile.txt contains:

```
1: First  
2: Second  
3: Third
```

C. newfile.txt is empty

D. an exception is thrown at runtime

E. compilation fails

答案 A

考題 8

Which code fragment correctly appends "JavaSE 7" to the end of the file msg.txt?

- A.

```
FileWriter wa = new FileWriter("msg.txt");
    wa.append("JavaSE 7");
    wa.close();
```
- B.

```
FileWriter wb = new FileWriter("msg.txt", true);
    wb.append("JavaSE 7");
    wb.close();
```
- C.

```
FileWriter wc = new FileWriter("msg.txt", FileWriter.MODE_APPEND);
    wc.append("JavaSE 7");
    wc.close();
```
- D.

```
FileWriter wd = new FileWriter("msg.txt", Writer.MODE_APPEND);
    wd.append("JavaSE 7");
    wd.close();
```

答案 B

說明 選項 A：執行時覆蓋原檔案。選項 C、D：應傳入 true/false。true 表示寫入內容採 append 方式。

考題 9

Given:

```
public static void print(int[] nums) {
    for (int number : nums) {
        System.out.println(number);
    }
}
```

Assume the method printNums is passed a valid array containing data.

Why is this method not producing output on the console?

- A. There is a compilation error.

- B. There is a runtime exception.
- C. The variable number is not initialized.
- D. Standard error is mapped to another destination.

答案 D

說明 本題使用 System.err，不是 System.out。

考題 10

Assuming the port statements are correct, which three code fragments create a one byte file?

- A.

```
OutputStream fos = new FileOutputStream(new File("fileA"));
OutputStream bos = new BufferedOutputStream(fos);
DataOutputStream dos = new DataOutputStream(bos);
dos.writeByte(0);
dos.close();
```
- B.

```
OutputStream fos = new FileOutputStream("fileB");
DataOutputStream dos = new DataOutputStream(fos);
dos.writeByte(0);
dos.close();
```
- C.

```
OutputStream fos = new FileOutputStream(new File("fileC"));
DataOutputStream dos = new DataOutputStream(fos);
dos.writeByte(0);
dos.close();
```
- D.

```
OutputStream fos = new FileOutputStream("fileD");
fos.writeByte(0);
fos.close();
```

答案 ABC

說明 選項 D：writeByte(0) 方法可以寫入 0 (bytes) 到檔案，但只存在 DataOutputStream 類別，故無法編譯。

考題 11

Given:

```
public static void main(String[] args) throws IOException {
    BufferedReader br =
```

```

        new BufferedReader(new InputStreamReader(System.in));
System.out.print("Enter num: ");
// line1
System.out.println(num);
}

```

Which code fragment, when inserted at //line1, enables the code to read the Num from the user?

- A. int num = Integer.parseInt(br.readLine());
- B. int num = br.read();
- C. int num = br.nextInt();
- D. int num = Integer.parseInt(br.next());

答案 A

考題 12

There is content:

welcome1=OCPJP

in file /resources/Message.properties.

And given:

```

public static void main(String[] args) throws IOException {
    Properties p = new Properties();
    FileInputStream fis =
        new FileInputStream("resources/Message.properties");
    p.load(fis);
    System.out.println(p.getProperty("welcome1"));
    System.out.println(p.getProperty("welcome2", "Java")); // line1
    System.out.println(p.getProperty("welcome3"));
}

```

What is the result?

A. OCPJP

Java

Then followed by an Exception stack trace

B. OCPJP

Then followed by an Exception stack trace

C. OCPJP

Java

null

D. Compilation error at line1.

答案 C

說明 Properties 類別的 getProperty(String key) 方法，若該 key 找不到對應的 value，將回傳 null。另一 overloading 版本：

```
String getProperty(String key, String defaultValue)
```

若該 key 找不到對應的 value，將回傳 defaultValue。

考題 13

Suppose that ocpjp.txt is accessible and there is some text in it:

Course::Java

Given:

```
public static void main(String[] args) {
    int i;
    char c;
    try (FileInputStream fis = new FileInputStream("ocjp.txt");
        InputStreamReader in = new InputStreamReader(fis);) {
        while (in.ready()) { // line1
            in.skip(2);
            i = in.read();
            c = (char) i;
            System.out.print(c);
        }
    } catch (Exception e) {
        System.out.println(e);
    }
}
```

What is the result?

A. ur :: va

B. ueJa

- C. Prints nothing.
- D. Compilation error at line1.

答案 B

說明 本題在每次讀取時，先跳過2個字元，再讀取1個字元，所以只有第3、6、9、12的字元被讀取。

NIO.2

-
- | 9.1 NIO.2 基礎
 - | 9.2 使用Path 介面定義檔案/ 目錄
 - | 9.3 使用Files 類別操作檔案/ 目錄
 - | 9.4 使用Files 類別操作channel 和stream I/O
 - | 9.5 讀寫檔案/ 目錄的屬性
 - | 9.6 遞迴存取目錄結構
 - | 9.7 使用PathMatcher 類別找尋檔案/ 目錄
 - | 9.8 其他
 - | 9.9 認證考試命題範圍

9.1 NIO.2基礎

9.1.1 java.io.File 的限制

前一章節已經對 Java 的基礎 I/O 有了認識，讓我們更進一步了解基礎 I/O 存在那些不方便的地方：

1. 很多方法遇到錯誤，是回傳 false，而非丟出例外。
2. 缺少很多存取檔案常用功能，如複製 (copy)、移動 (move)。
3. 不是每個作業系統都支援重新命名。
4. 對「Symbolic Links」類型的檔案沒有支援。
5. 對於檔案「metadata (描述檔案的資料)」的取得很有限，如檔案權限、檔案擁有者、安全性設定等都沒有。
6. 存取檔案的「metadata (描述檔案的資料)」沒有效率，一次只能一個，每次呼叫都會轉呼叫系統指令 (system call)。
7. 很多方法遇到檔案較大時，會呈現卡住 (hang) 的狀態，久久沒有回應，甚至當掉。
8. 遞迴目錄結構時，遇到「Symbolic Links」類型的檔案無法適當處理。
9. 遇到新型態的作業系統或新檔案型態時，API 不易擴充。

9.1.2 Java I/O 套件發展歷史

為了處理這些基礎 I/O 為人詬病的地方，Java 陸續進行改善方案。歷史如下：

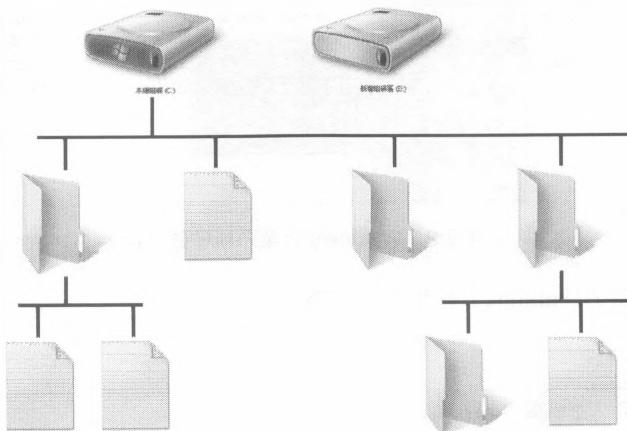
◆表 9-1 Java I/O 演進

功能 (Feature)	JSR	版本	套件 (Package)
I/O		Java 2	java.io.*
New I/O (NIO)	51	Java 4	java.nio.*
New I/O 2 (NIO.2)	203	Java 7	java.nio.file.*

我們已經在上一章節討論過基礎 I/O，「Channel I/O」就屬於 NIO 的一環。本章重頭戲就是 NIO.2 版本介紹。

9.1.3 檔案系統、路徑和檔案

檔案系統 (File System) 是樹狀結構。根目錄 (root directories) 可以有多個，如 Windows 的 C:、D:。



❖ 圖 9-1 Windows 檔案系統

在 NIO.2 中，檔案 / 目錄都以「路徑 (path)」來表達，區分「絕對路徑 (absolute path)」和「相對路徑 (relative path)」：

1. 絕對路徑

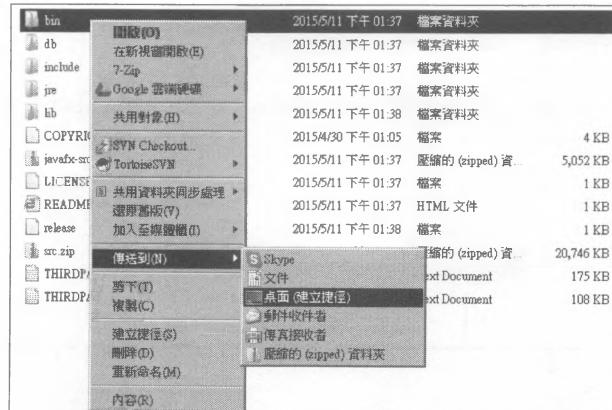
- 包含根目錄。
- 定位檔案位置所必需。
- 如 `/opt/jboss-eap-6.2/domain/servers/server-8080/log/server.log` 或 `C:\Program Files\Java\jre1.8.0_77\bin`。

2. 相對路徑

- 必須再結合絕對路徑才有可能找到檔案真正位置。
- 如 `jboss-eap-6.2/domain/servers/server-8080/log/server.log` 或 `jre1.8.0_77\bin`。

9.1.4 Symbolic Links

又稱為「Symlink」或「Soft Link」。以微軟公司的 Windows 作業系統而言，並「非」我們熟悉的「捷徑 (short cut)」：



❖ 圖 9-2 Windows 作業系統中建立捷徑

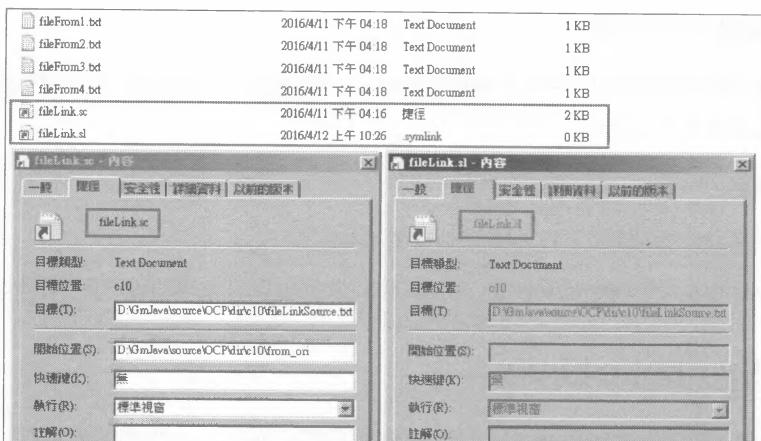
而是以 DOS 指令「mklink」產生。指令為：

三 語法

`mklink 連結檔案 來源檔案`

建立之後，乍看和捷徑很像。但詳細比較，以 Windows 為例，有諸多不同的地方：

1. 檔案類型不同，如下圖。
2. 檔案大小不同，捷徑佔有大小 2KB，symlink 則是 0，如下圖。
3. 分別複製兩者時，捷徑本身會被複製，symlink 則是複製「來源檔案」，並非 symlink 本身。如下圖：



❖ 圖 9-3 捷徑和 Symbolic Link 的比較

9.1.5 NIO.2 的基本架構

在 JDK 7 之前，`java.io.File` 是所有檔案 / 目錄的操作基礎。JDK 7 推出 NIO.2 後則改為三個基礎：

1. `java.nio.file.Path`：用來定義檔案 / 目錄。
2. `java.nio.file.Files`：用來操作檔案 / 目錄。
3. `java.nio.file.FileSystem`：用來建立 Path 或其他存取檔案系統的物件。

NIO.2 所有方法都丟出 `IOException`，或其子類別。

9.2 使用Path介面定義檔案/目錄

9.2.1 Path 介面和其主要功能

`java.nio.file.Path` 是介面，也是 NIO.2 架構的進入點。取得 Path 物件的作法有二種，物件建立後為 `immutable`，亦即狀態不能修改：

作法一：藉由 `FileSystem` 類別的 `getPath()` 方法：

```
FileSystem fs = FileSystems.getDefault();
Path p1 = fs.getPath ("D:\\labs\\resources\\myfile.txt");
```

作法二：藉由 `java.nio.file.Paths` 類別的靜態 `get()` 方法：

```
Path p0 = Paths.get ("D:\\labs\\resources\\myfile.txt");
Path p1 = Paths.get ("D:/labs/resources/myfile.txt");
Path p2 = Paths.get ("D:", "labs", "resources", "myfile.txt");
Path p3 = Paths.get ("/temp/foo");
Path p4 = Paths.get (URI.create ("file:///~/somefile"));
```

值得注意的是，Windows 的檔案路徑接受二種方向的斜線，如行 1 和 2。使用「\」必須再加上跳脫符號「\」，使用「/」則不必。

`Path` 介面用來找出檔案 / 目錄。常用方法分三類：

1. 分解路徑

取得構成路徑的所有檔案 / 目錄。主要分成「root(根目錄)」和「name」2 種路徑成員。
`root` 路徑成員只有一個，`name` 路徑成員則可以有多個：

- `getFileName()`
- `getParent()`
- `getRoot()`
- `getNameCount()`，不含根目錄

2. 操作路徑

- `normalize()`
- `toUri()`
- `toAbsolutePath()`
- `subpath()`
- `resolve()`
- `relativize()`

3. 比較路徑

- `startsWith()`
- `endsWith()`
- `equals()`

範例「/OCP/src/course/c09/PathTest.java」對 Path 介面的不同方法進行展示：

範例

```
1 private static void testSplit() {  
2     Path p1 = Paths.get("D:/Temp/Foo/file1.txt");  
3     System.out.format("getFileName: %s%n", p1.getFileName());  
4     System.out.format("getParent: %s%n", p1.getParent());  
5     System.out.format("getNameCount: %d%n", p1.getNameCount());  
6     System.out.format("getRoot: %s%n", p1.getRoot());  
7     System.out.format("isAbsolute: %b%n", p1.isAbsolute());  
8     System.out.format("toAbsolutePath: %s%n", p1.toAbsolutePath());  
9     System.out.format("toURI: %s%n", p1.toUri());  
10 }
```

結果

```
getFileName: file1.txt  
getParent: D:\Temp\Foo  
getNameCount: 3  
getRoot: D:\
```

```
isAbsolute: true
toAbsolutePath: D:\Temp\Foo\file1.txt
toURI: file:///D:/Temp/Foo/file1.txt
```

說明

注意 root 路徑成員和 name 路徑成員不同，`getNameCount()` 方法只計算 name 路徑成員個數，編碼由 0 到 2，所以長度是 3。如下表：

◆ 表 9-2 路徑組成分析

路徑組成	D:	Temp	Foo	file1.txt
成員分類	root	name		
		0	1	2

9.2.2 移除 Path 的多餘組成

許多檔案系統使用：

- 「..」代表目前目錄。
- 「...」代表上一層目錄。

以下路徑內有多餘的組成：

```
/home/./clarence/foo
/home/peter/../clarence/foo
```

使用 `normalize()` 方法可以移除多餘的部分：「..」和「directory/..」。該方法只是語法上處理，不檢查實際檔案狀況，如下：

範例

```
1 private static void testNormalize() {
2     Path p1 = Paths.get("/home/./clarence/foo");
3     p1 = p1.normalize();
4     System.out.println(p1);
5     Path p2 = Paths.get("/home/peter/../clarence/foo");
6     p2 = p2.normalize();
7     System.out.println(p2);
8 }
```

結果

```
\home\clarence\foo
\home\clarence\foo
```

9.2.3 建立子路徑

使用 `subpath()` 方法可以取得路徑裡的部分路徑，如下：

範例

```

1 private static void testSubPath() {
2     Path p1 = Paths.get("D:/Temp/foo/bar");
3     p1.subpath(1, 3);           // immutable test
4     System.out.println(p1);
5     p1 = p1.subpath(1, 3);
6     System.out.println(p1);
7 }
```

結果

```
D:\Temp\foo\bar
foo\bar
```

說明

`subpath(1, 3)` 表示 name 路徑成員中，由 `index=1` 開始取，不含 `index=3` 的成員，亦即取出成員 1 和 2。下表說明 `index` 的位置表示：

◆ 表 9-3 路徑組成分析

D	Temp	foo	bar
root	0	1	2

所以結果為 `foo\bar`。

9.2.4 結合 2 個路徑

使用 `resolve()` 方法結合 2 個路徑：

- 若傳入「相對路徑」，則將該「相對路徑」連接在「原路徑」之後。
- 若傳入「絕對路徑」，則方法回傳該「絕對路徑」。

範例如下：

範例

```

1 private static void testResolve() {
2     String p = "/home/clarence/foo";
3     Path p1 = Paths.get(p).resolve("bar");
4     System.out.println(p1);
```

```

5     Path p2 = Paths.get(p).resolve("/home/clarence");
6     System.out.println(p2);
7 }
```

結果

```
\home\clarence\foo\bar
\home\clarence
```

9.2.5 建立連接 2 個路徑的路徑

使用 `relativize()` 方法建構 2 個路徑間的路徑，由原路徑到 `relativize()` 方法所傳入的路徑，範例如下：

範例

```

1 private static void testRelativize() {
2     Path p1 = Paths.get("peter");
3     Path p2 = Paths.get("jim");
4     Path p1Top2 = p1.relativize(p2);      // 由 p1 到 p2 的走法
5     System.out.println(p1Top2);
6     Path p2Top1 = p2.relativize(p1);      // 由 p2 到 p1 的走法
7     System.out.println(p2Top1);
8 }
```

結果

```
..\jim
..\peter
```

說明

4, 6	先到上一層，再走下來。以行 4 為例，目前在 <code>peter</code> 目錄內，必須先回到上一層目錄，才能下來到 <code>jim</code> 目錄。
------	---

9.2.6 連結檔案

Hard Link

部分檔案系統支援「Hard Links」類型的檔案，相對於「Soft/Symbolic Link」，有更多限制：

1. 目標檔案一定要存在。
2. 目標不可以是目錄，只能是檔案。
3. 目標不可以跨磁碟，如不可以在 D 磁碟建立 C 磁碟的檔案的 hard links。
4. 行為、外觀、屬性等和一般檔案相似，不易判斷。

處理 Symbolic Link

NIO.2 的類別可以感知 Link 類型檔案的存在，稱為「Link Aware」。相關方法具備以下能力：

1. 偵測是否遇到 Symbolic Link 檔案。
2. 設定遇到 Symbolic Link 檔案時的處理方式。

方法如：

```
Files.createSymbolicLink(Path p1, Path p2, FileAttribute<?>);  
Files.createLink(Path p1, Path p2);      // 建立hard link  
Files.isSymbolicLink(Path p1);  
Files.readSymbolicLink(Path p1);        // 找出Symbolic Link的target
```

9.3 使用Files類別操作檔案/目錄

9.3.1 檔案 / 目錄的基本處理

先使用 Path 物件定位檔案 / 目錄。再使用 Files 類別操作 Path 物件，以達成：

1. 檔案與目錄的：
 - 檢查 (check)
 - 刪除 (delete)
 - 複製 (copy)
 - 移動 (move)
2. 管理屬性資料 (metadata)。
3. 讀 / 寫和建立檔案。
4. 隨機存取檔案。
5. 讀取目錄 (directory) 內的檔案。

檢查檔案 / 目錄是否存在

Path 代表檔案 / 目錄的位置。在存取之前，應該要先使用 Files 類別檢查是否存在 (Symbolic Link 也算檔案)。使用方法為：

```
Files.exists(Path p, LinkOption... option);
Files.notExists(Path p, LinkOption... option);
```

若兩個方法的測試結果若都是 false，表示狀態「無法確認 (unknown)」。可能原因很多，常見如：

1. 沒有權限。
2. 離線磁碟機 (off-line drive)，像 CD-ROM。

如範例程式 「/OCP/src/course/c09/FilesTest.java」 的 testExists() 方法：

範例

```
1 private static void testExists() {
2     String thisJava = System.getProperty("user.dir") +
3         "\\src\\course\\c09\\FilesTest.java";
4     Path p = Paths.get(thisJava);
5     boolean b = Files.exists(p, LinkOption.NOFOLLOW_LINKS);
6     System.out.format("%s exists: %b%n", p, b);
7     b = Files.notExists(p, LinkOption.NOFOLLOW_LINKS);
8     System.out.format("%s does not exists: %b%n", p, b);
}
```

結果

```
D:\GmJava\source\OCP\src\course\c09\FilesTest.java exists: true
D:\GmJava\source\OCP\src\course\c09\FilesTest.java does not exists: false
```

說明

- | | |
|---|--|
| 4 | 檔案若是 link，預設來源檔案也必須同時存在，才算存在。
使用 LinkOption.NOFOLLOW_LINKS，表示不檢查來源檔案是否存在。 |
|---|--|

檢查檔案 / 目錄屬性

檢查權限的使用方法為：

```
Files.isReadable (Path p);
Files.isWritable (Path p);
Files.isExecutable (Path p);
```

檢查是否為同一檔案的方法 (常用於 Symbolic Links) :

```
Files.isSameFile(Path p1, Path p2);
```

以上檢查一旦結束，就不再保證結果，因為檔案可能馬上被其他系統指令更改。

建立檔案 / 目錄

建立檔案的方法為：

```
Files.createFile (Path file);
```

建立單一目錄的方法為：

```
Files.createDirectory (Path dir);
```

建立多重目錄的方法如下。通常用於將路徑裡缺少的 name 成員一次全部建立：

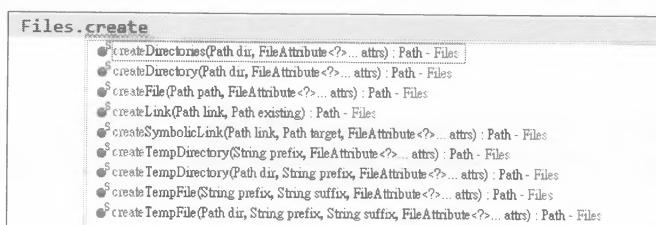
```
Files.createDirectories (Path dir);
```

假設只有 D:/Temp 目錄存在。使用

```
Files.createDirectories (Paths.get("D:/Temp/foo/bar/example"));
```

可以將缺少的目錄一次建立完成。

Files 類別還有更多建立各式檔案 / 目錄的 static 方法：



❖ 圖 9-4 Files 類別下的 create 方法

刪除檔案 / 目錄

刪除檔案 / 目錄使用的方法為：

```
Files.delete(Path p);
```

失敗時可能丟出以下例外：

1. `NoSuchFileException`：要刪除的檔案不存在。

2. `DirectoryNotEmptyException`：要刪除的目錄不為空。

3. `IOException`：其他錯誤。

也可以刪除檔案 / 目錄前先確認是否存在，方法為：

```
Files.deleteIfExists (Path p);
```

則檔案不存在就不會刪除，因此不會有 `NoSuchFileException`，但其他錯誤還是可能發生。

9.3.2 複製和移動檔案 / 目錄

複製和移動檔案 / 目錄的方法為：

```
Files.copy (Path source, Path target, CopyOption...);
Files.move (Path source, Path target, CopyOption...);
// source : 來源路徑，可以是目錄 / 檔案
// target : 目標路徑，可以是目錄 / 檔案
```

其中，傳入的 `CopyOption` 是介面，允許同時多個。本文中將陸續介紹二個實作它的 `enum`。

將複製和移動等二個方法一併討論，是因為 OCP 考試時經常會有比較觀念的需要。比如說無論複製或移動，都需要指定「來源路徑」和「目標路徑」。兩者相同點是：

1. 若目標路徑已經存在，但操作前未指定可以覆蓋(使用 `StandardCopyOption.REPLACE_EXISTING`)，將失敗。
2. 若目標路徑不存在，則操作後將自動建立。
3. 在操作前來源和目標「非一致」是檔案 / 目錄將不影響結果。因為在操作成功後，「目標路徑」會和「來源路徑」相同。因此：
 - 可以指定來源是檔案，但目標是目錄。成功複製 / 移動後目標目錄將被取代為檔案，而非檔案放到目錄下。
 - 可以指定來源是目錄，但目標是檔案。成功複製 / 移動後目標路徑將被取代為目錄。

「目標路徑」有自己必須注意的地方

1. 若目標路徑是存在的「檔案」或「空目錄」，使用 `REPLACE_EXISTING` 或 `ATOMIC_MOVE` 可避免拋出 `java.nio.file.FileAlreadyExistsException`。
2. 若目標路徑是存在的「非空目錄」，用 `REPLACE_EXISTING` 還不夠，還是會拋出 `java.nio.file.DirectoryNotEmptyException`。

「來源路徑」也有自己必須注意的地方

1. 必須存在，否則拋出 `java.nio.file.NoSuchFileException`。
2. 來源路徑是「目錄」時，即便「複製」成功也無法複製內含檔案，只會產生新目錄，且過程不會出錯。
3. 來源路徑是「目錄」時，若「移動」成功，內含的檔案 / 目錄將一併搬家。

複製或移動路徑時，可以在第三個參數傳入有實作介面 `CopyOption` 的列舉型態：

◆表 9-4 `CopyOption` 型態列舉

介面	列舉型別 (enum)	列舉項目 (types)
CopyOption	LinkOption	NOFOLLOW_LINKS
	StandardCopyOption	REPLACE_EXISTING
		COPY_ATTRIBUTES
		ATOMIC_MOVE

「複製」檔案 / 目錄時注意事項

1. 來源路徑是 Symbolic Link 時，預設將複製「Link 指向的檔案」。若要複製「Link 本身」，必須加上列舉型態 `LinkOption.NOFOLLOW_LINKS`。
2. 列舉型態 `StandardCopyOption.COPY_ATTRIBUTES` 用於將檔案屬性一併複製。大部分屬性將依檔案系統不同而可能不被複製；但檔案最後修改時間 (last-modified-Time) 將被支援。

「移動」檔案 / 目錄時注意事項

1. 使用列舉型態 `StandardCopyOption.ATOMIC_MOVE`，若檔案系統不支援將丟出例外；若支援，則可避免移動過程中有其他系統程序存取該檔案。如用於耗時較久的大檔案移動，可以保證接下來要存取該檔案的系統程序都可存取到完整的檔案。
2. 若移動 Symbolic Link 檔案，不需要使用列舉型態 `LinkOption.NOFOLLOW_LINKS`。

以上說明的原則可以使用範例「/OCP/src/course/c09/FilesCopyTest.java」和「/OCP/src/course/c09/FilesMoveTest.java」進行演練測試。這兩個範例演練所需要的檔案 / 目錄都建置在「/OCP/dir/c09」目錄中，因為程式碼內容較長，不在本文內提供。

9.3.3 Stream 和 Path 互相複製

檔案複製的來源或目標除了 Path 之外，也可以是基礎 I/O 裡提到的串流 (Stream) 物件。方法為：

```
Files.copy(InputStream source, Path target, CopyOption... options);
Files.copy(Path source, OutputStream target);
// source：檔案複製來源，使用 InputStream
// target：檔案複製後的輸出，使用 OutputStream
```

範例「/OCP/src/course/c09/CopyInputStreamTest.java」顯示如何將遠端的網頁轉換成 InputStream 物件後，再複製為本機的檔案：

範例

```
1 public class CopyInputStreamTest {
2     public static void main(String[] args) throws IOException {
3         Path to = Paths.get("dir/c09/Java_logo.png").toAbsolutePath();
4         URL url = URI.create("https://upload.wikimedia.org/wikipedia/zh/8/
88/Java_logo.png").toURL();
5         try (InputStream from = url.openStream()) {
6             Files.copy(from, to, StandardCopyOption.REPLACE_EXISTING);
7             System.out.println("Copy finished...");
8         }
9     }
10 }
```

9.3.4 列出自目錄內容

DirectoryStream 介面可以找出目錄下的所有檔案 / 目錄，但只能限制在目錄下的第一層，可支援大目錄。如範例「/OCP/src/course/c09/DirectoryStreamTest.java」將印出 D 磁碟下的所有檔案 / 目錄名稱，但只限於第一層：

範例

```
1 public class DirectoryStreamTest {
2     public static void main(String[] args) {
3         Path dir = Paths.get("D:/");
4         try (DirectoryStream<Path> stream =
5             Files.newDirectoryStream(dir, "*")) {
6             for (Path file : stream) {
7                 System.out.println(file.getFileName());
8             }
9         }
10 }
```

```

7      }
8      } catch (PatternSyntaxException | DirectoryIteratorException
9          | IOException x) {
10     System.err.println(x);
11 }
12 }

```

9.3.5 讀取和寫入檔案

Files 類別的 `readAllBytes()` 和 `readAllLines()` 方法可以一次讀取檔案全部內容，但檔案不能太大。`write()` 則提供寫入檔案的功能。

如範例「/OCP/src/course/c09/FileReadWriteAllLineTest.java」：

範例

```

1 public class FileReadWriteAllLineTest {
2     public static void main(String[] args) throws IOException {
3         Path source = Paths.get("dir/c09/file.txt").toAbsolutePath();
4         Charset cs = Charset.defaultCharset();
5         List<String> lines = Files.readAllLines(source, cs);
6         Path target = Paths.get("dir/c09/file2.txt").toAbsolutePath();
7         Files.write(target, lines, cs,
8             StandardOpenOption.CREATE,
9             StandardOpenOption.TRUNCATE_EXISTING,
10            StandardOpenOption.WRITE);
11     System.out.println("done...");
12 }

```

9.4 使用Files類別操作channel和stream I/O

Channel 介面和 ByteBuffer 類別

Channel 介面和 ByteBuffer 類別搭配使用，可以提高 I/O 效率：

1. Stream I/O 每次讀取一個位元或字元；Channel I/O 則每次讀取一塊記憶體 (buffer)。
2. `java.nio.channels.ByteChannel` 介面繼承 Channel 介面，提供基本讀寫功能。
3. `java.nio.channels.SeekableByteChannel` 介面繼承 ByteChannel，提供在 channel 中讀寫時記錄目前位置，並且改變讀寫位置的能力，讓「隨機存取 (random access) 檔案」變的可能。

4. 使用 `Files.newByteChannel(Path, OpenOption...)` 方法回傳 `SeekableByteChannel` 實例後，也可以再轉型為 `java.nio.channels.FileChannel` 類別，該類別曾經在 Java I/O 基礎中介紹。以下將逐步介紹使用方式。

隨機存取檔案

使用 `SeekableByteChannel` 介面，可以進行檔案內容的「隨機存取」。步驟為

1. 打開檔案。
2. 找到存取位置。
3. 開始讀寫。

常用方法有：

1. `position()`：回傳在 `channel` 中的位置。
2. `position(long)`：設定在 `channel` 中的位置。
3. `read(ByteBuffer)`：由 `channel` 中將資料讀入 `buffer`。
4. `write(ByteBuffer)`：將資料由 `buffer` 中寫入 `channel`。

以上 `channel` 由檔案產生。

範例「/OCP/src/course/c09/SeekableByteChannelTest.java」顯示如何使用 `SeekableByteChannel` 介面，將新增字串放在檔案的指定位置，本例為最尾端：

範例

```

1  public class SeekableByteChannelTest {
2      public static void main(String[] args) throws IOException {
3          Path path = Paths.get("dir/c09/file.txt").toAbsolutePath();
4          try (SeekableByteChannel sbc =
5              Files.newByteChannel(path, StandardOpenOption.WRITE)) {
6              long channelSize = sbc.size();
7              sbc.position(channelSize);
8              System.out.println("position: " + sbc.position());
9              ByteBuffer buffer = ByteBuffer.wrap("\n" + "0").getBytes();
10             sbc.write(buffer);
11             System.out.println("position: " + sbc.position());
12         }
13     }

```

 **說明**

4	由 <code>Files.newByteChannel()</code> 方法取得 <code>SeekableByteChannel</code> 介面的實作物件。該物件指向 <code>path</code> 所在的檔案，且指定屬性為 <code>StandardOpenOption.WRITE</code> ，所以可以寫入。
5	取得行 4 檔案物件的大小。
6	將檔案的讀取位置移到最尾端。
7	印出目前檔案最尾端的位置。
8	建立 <code>ByteBuffer</code> 物件，內容為字串： <code>"\n" + "0"</code> 。
9	將行 8 的 <code>ByteBuffer</code> 物件的內容，寫入檔案裡。
10	印出目前檔案最尾端的位置。

對文字檔提供 `Buffered I/O` 方法

NIO. 2 一樣可以使用 `BufferedReader`/ `BufferedWriter` 物件來提高檔案讀寫效率。

1. 使用 `Files.newBufferedReader()` 方法取得 `java.io.BufferedReader` 物件：

```
BufferedReader reader = Files.newBufferedReader(path, charset);
line = reader.readLine();
```

2. 使用 `Files.newBufferedWriter()` 方法取得 `java.io.BufferedWriter` 物件：

```
BufferedWriter writer = Files.newBufferedWriter(path, charset);
writer.write(s, 0, s.length());
```

取得位元 Streams 的方法

NIO. 2 也可以取得 `InputStream`/`OutputStream` 物件。

1. 使用 `Files.newInputStream()` 方法取得 `java.io.InputStream` 物件：

```
InputStream in = Files.newInputStream(path);
BufferedReader r = new BufferedReader(new InputStreamReader(in));
String line = r.readLine();
```

2. 使用 `Files.newOutputStream()` 方法取得 `java.io.OutputStream` 物件：

```
Path path = Paths.get("dir/c09/logFile.txt").toAbsolutePath();
String s = "Hi, Jim...";
byte data[] = s.getBytes();
try (OutputStream out = Files.newOutputStream(path, CREATE, APPEND)) {
    BufferedOutputStream bot = new BufferedOutputStream(out);
    out.write(data, 0, data.length());
}
```

9.5 讀寫檔案/目錄的屬性

9.5.1 使用 Files 管理屬性資料

Files 類別提供了若干管理檔案 / 目錄屬性的方法，如下：

◆表 9-5 Files 類別常用的屬性資料管理方法

方法	說明
size	回傳檔案大小 (bytes)
isDirectory	判斷是否為目錄
isRegularFile	判斷是否為檔案
isSymbolicLink	判斷是否為 Symbolic Link
isHidden	判斷是否為隱藏檔
getLastModifiedTime	取得最後修改時間
setLastModifiedTime	設定最後修改時間
getAttribute	取得屬性
setAttribute	設定屬性

範例「/OCP/src/course/c09/MetadataTest.java」顯示如何使用 Files 類別取得檔案 / 目錄屬性：

④ 範例

```

1  public class MetadataTest {
2      public static void main(String[] args) throws IOException {
3          Path basic = Paths.get("dir/c09/metadata").toAbsolutePath();
4          out.println("basic path: " + basic);
5
6          Path common = basic.resolve("file.txt");
7          Path shortcut = basic.resolve("dir.shortcut");
8          Path hidden = basic.resolve("hiddenFile");
9          Path symlink = basic.resolve("dir.sl");
10
11         out.println("size: " + Files.size(common));
12         out.println("isDirectory: " + Files.isDirectory(common));
13         out.println("isRegularFile: " + Files.isRegularFile(common));
14         out.println("isSymbolicLink(dir.sl): " +
15                     Files.isSymbolicLink(symlink));
16         out.println("isSymbolicLink(dir.shortcut): " +
17                     Files.isSymbolicLink(shortcut));
18         out.println("isHidden: " + Files.isHidden(hidden));

```

```

17     // 取得並修改 last modified time 屬性
18     out.println("getLastModifiedTime: " +
19         Files.getLastModifiedTime(common));
20     FileTime time = FileTime.fromMillis(new Date().getTime());
21     Files.setLastModifiedTime(common, time);
22     out.println("getLastModifiedTime: " +
23         Files.getLastModifiedTime(common));
24     // 修改 hidden 屬性
25     Files.setAttribute(hidden, "dos:hidden", new Boolean(false));
26     out.println("isHidden: " +
27         Files.getAttribute(hidden, "dos:hidden"));
}

```

說明

3	在相對路徑「dir/c09/metadata」下事先建立所有測試檔案。
19	要設定檔案的時間屬性，必須使用類別 FileTime。
23	使用 setAttribute() 方法設定 hidden 屬性。
24	除了使用 isHidden() 方法測試屬性是否為 hidden 外，也可以使用 getAttribute() 方法搭配字串 "dos:hidden"。

9.5.2 讀取檔案屬性

過去 Java I/O 存取檔案屬性比較沒有效率，一次只能一個，每次呼叫都必須轉呼叫系統指令 (system call)。NIO.2 改進了這個問題，可以使用 DosFileAttributes 介面「一次」取回檔案 / 目錄的所有屬性。

以 Windows 的 DOS 為例，使用 Files 類別取得其物件實例：

```
DosFileAttributes attrs = Files.readAttributes(path, DosFileAttributes.class);
```

參見範例「/OCP/src/course/c09/DosFileAttributesReadDemo.java」。需要注意的是，在 Java 7 中，DosFileAttributes 介面只能讀取屬性，不能修改：

範例

```

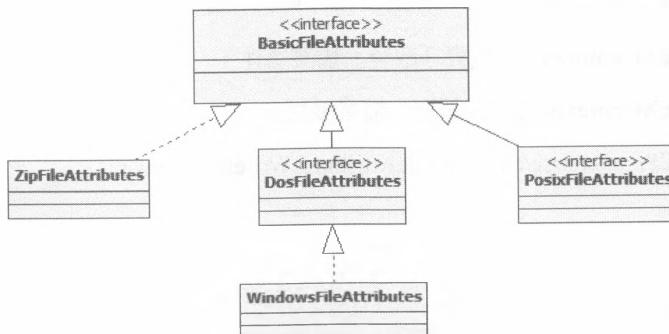
1  public class DosFileAttributesReadDemo {
2      public static void main(String[] args) throws IOException {
3          Path p = Paths.get("dir/c09/attributeTest.txt").toAbsolutePath();
4          DosFileAttributes attrs =
5              Files.readAttributes(p, DosFileAttributes.class);
}

```

```

6      // basic
7      FileTime creation = attrs.creationTime();
8      FileTime modified = attrs.lastModifiedTime();
9      FileTime lastAccess = attrs.lastAccessTime();
10     if (!attrs.isDirectory()) {
11         long size = attrs.size();
12     }
13     boolean isDirectory = attrs.isDirectory();
14     boolean isRegularFile = attrs.isRegularFile();
15     boolean isSymbolicLink = attrs.isSymbolicLink();
16     boolean isOther = attrs.isOther();
17
18     // only for DOS
19     boolean archive = attrs.isArchive();
20     boolean hidden = attrs.isHidden();
21     boolean readOnly = attrs.isReadOnly();
22     boolean systemFile = attrs.isSystem();
23 }
24 }
```

相關類別的 UML 架構為：



❖ 圖 9-5 BasicFileAttributes 家族類別架構

9.5.3 修改檔案屬性

DOS 的檔案屬性

檔案建立後，可以使用類別 Files 更改屬性：

```

Files.createFile (path);
Files.setAttribute (path, "dos:hidden", true);
```

`setAttribute()` 方法可以設定四種 DOS 屬性，必須指定屬性字串：

1. dos:hidden
2. dos:readonly
3. dos:system
4. dos:archive

介面 `DosFileAttributeView` 也提供了設定屬性的相關方法：

1. `setHidden()`
2. `setReadOnly()`
3. `setSystem()`
4. `setArchive()`

可以使用 `Files` 類別取得該介面的物件實例：

```
DosFileAttributeView view =
    Files.getFileAttributeView(p, DosFileAttributeView.class);
```

`DosFileAttributeView` 有直接的方法可以設定檔案屬性；若要讀取屬性，需先使用 `readAttributes()` 方法取得 `DosFileAttributes` 物件實例。基本的分工是：

1. 介面 **DosFileAttributeView** 負責「改變」檔案屬性。
2. 介面 **DosFileAttributes** 負責「讀取」檔案屬性。

如範例「/OCP/src/course/c09/DosFileAttributesWriteDemo.java」：

範例

```
1  public class DosFileAttributesWriteDemo {
2      public static void main(String[] args) throws IOException {
3          Path p = Paths.get("dir/c09/attributeTest.txt").toAbsolutePath();
4
5          DosFileAttributeView view =
6              Files.getFileAttributeView(p, DosFileAttributeView.class);
7          view.setArchive(true);
8          view.setReadOnly(true);
9          view.setHidden(true);
10         view.setSystem(true);
11
12         FileTime lastModifiedTime = FileTime.fromMillis(new Date().getTime());
13         FileTime lastAccessTime = FileTime.fromMillis(new Date().getTime());
14         FileTime createTime = FileTime.fromMillis(new Date().getTime());
```

```

14     view.setTimes( lastModifiedTime,
15                     lastAccessTime,
16                     createTime);
17     }
18 }
```

說明

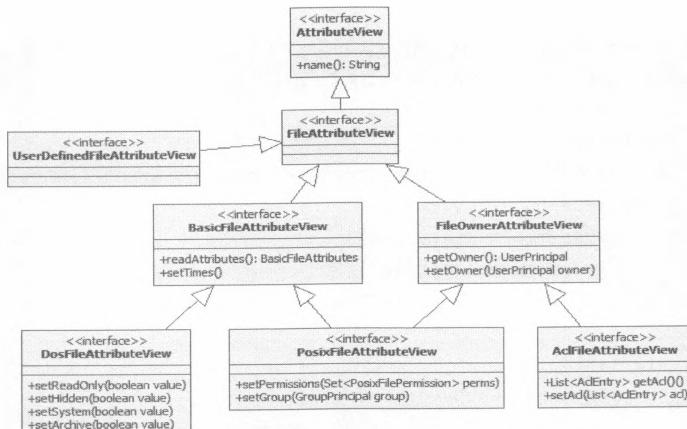
5	取得 DosFileAttributeView 的方法。
14	設定檔案的相關時間。以 Windows 作業系統為例，每個檔案屬性都有 3 個時間：建立日期 (createTime)、修改日期 (lastModifiedTime)、存取日期 (lastAccessTime)。
16	取得 DosFileAttributes 的方法。

DOS 之外的檔案屬性

除了 DOS 之外，NIO.2 裡其他可支援的 attribute views 還包含：

1. BasicFileAttributeView：提供所有檔案系統都支援的基本屬性。
2. PosixFileAttributeView：支援 POSIX 家族，如 UNIX。
3. FileOwnerAttributeView：支援所有具備「檔案擁有者 (file owner)」概念的檔案系統。
4. AclFileAttributeView：支援讀寫檔案的「存取控制清單 (ACL : Access Control List)」。
5. UserDefinedFileAttributeView：讓使用者自行定義。

類別家族的 UML 架構為：



❖ 圖 9-6 AttributeView 家族類別架構

POSIX 檔案系統的權限

在 NIO.2 中，可以建立檔案 / 目錄在 POSIX (Portable Operating System Interface) 檔案系統中，如 MacOS、Linux 和 Solaris 等；Windows 則非 POSIX 相容。以下程式碼顯示如何取得目前作業系統支援的所有 AttributeView：

```
FileSystems.getDefault().supportedFileAttributeViews();
```

POSIX 檔案系統中，使用 ACL 來進行檔案 / 目錄的權限控管。ACL 是 Access Control List (存取控制清單) 的縮寫，主要的目的在提供對檔案 / 目錄擁有權相關的三種使用者：

1. owner (檔案擁有者)
2. group (檔案擁有者所在群組)
3. others (非 owner 和 group 的其他人)

對檔案 / 目錄的 read(讀)、write(寫)、execute(執行) 權限的設定。

ACL 權限設定格式如下：

表 9-6 ACL 權限設定格式表

Owner			Group			Other		
r	w	x	r	w	x	r	w	x

其中 r 表示「read」，w 表示「write」，x 表示「execute」。如範例「/OCP/src/course/c09/POSIXpermissionTest.java」：

範例

```

1  public class POSIXpermissionTest {
2      public static void main(String[] args) {
3          boolean isPOSIX = false;
4          Set<String> views =
5              FileSystems.getDefault().supportedFileAttributeViews();
6          for (String s : views) {
7              System.out.println(s);
8              if (s.equals("posix"))
9                  isPOSIX = true;
10         }
11         if (isPOSIX) {
12             Path p = Paths.get(args[0]);
13             Set<PosixFilePermission> perms =
14                 PosixFilePermissions.fromString("rwxr-x---");
15             FileAttribute<Set<PosixFilePermission>> attrs =
16                 PosixFilePermissions.asFileAttribute(perms);
17         }
18     }
19 }
```

```

14         try {
15             Files.createFile(p, attrs);
16         } catch (FileAlreadyExistsException f) {
17             f.printStackTrace();
18         } catch (IOException i) {
19             i.printStackTrace();
20         }
21     }
22 }
23 }
```

說明

4	取得目前作業系統支援的所有檔案系統，若是 Windows，則不支援 POSIX。
5~9	判斷是否有 "posix" 關鍵字。若有，表示屬於 POSIX 家族。
12	設定預備建立的檔案權限： owner : r、w、x group : r、-、x other : -、-、- 不具備的權限以「-」表示。
13	將檔案權限轉換為檔案屬性。
15	以檔案屬性物件 attrs 建立檔案。

9.6 遞迴存取目錄結構

9.6.1 對檔案目錄進行遞迴操作

DirectoryStream 物件可以拜訪目錄下的所有檔案 / 目錄，但被限制在以下一層。使用 Files.walkFileTree (Path start, FileVisitor<T> visitor) 則可以遞迴拜訪所有層級的所有檔案 / 目錄，並對拜訪過的所有檔案 / 目錄採取「特定動作」。

「特定動作」就由覆寫 FileVisitor 介面的方法來提供：

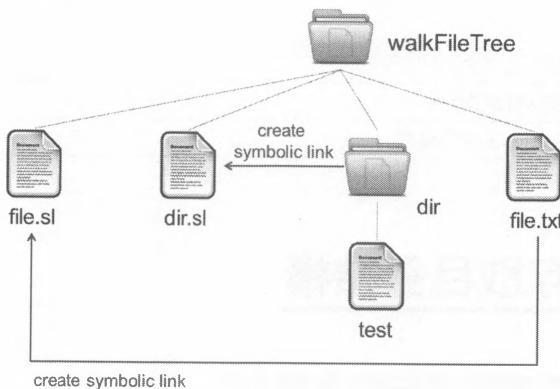
1. preVisitDirectory()：拜訪目錄前要做的事。
2. visitFile()：拜訪檔案時要做的事。
3. postVisitDirectory()：拜訪目錄後要做的事。
4. visitFileFailed()：拜訪檔案若失敗要做的事。

藉由每次拜訪檔案 / 目錄後的回傳值 (列舉型別 FileVisitResult 的列舉項目) 來決定是否繼續拜訪其他檔案 / 目錄：

1. CONTINUE : 繼續。
2. SKIP_SIBLINGS : 略過同一層的檔案 / 目錄。
3. SKIP_SUBTREE : 略過下一層檔案樹。
4. TERMINATE : 結束。

FileVisitor 是介面，因此實作的類別必須覆寫所有抽象方法，比較麻煩，可以考慮改繼承 SimpleFileVisitor 類別。該類別已經實作 FileVisitor 介面的所有抽象方法，且都回傳 CONTINUE，因此只要覆寫真正需要的方法即可，較簡單。

範例「/OCP/src/course/c09/WalkFileTreeExample.java」顯示如何使用 Files. walkFileTree() 並搭配 FileVisitor 介面遞迴所有目錄。準備拜訪的範例目錄結構是：



◆ 圖 9-7 walkFileTree 範例路徑結構

範例

```

1  class SimplePrintTree extends SimpleFileVisitor<Path> {
2  }
3
4  class PrintTree implements FileVisitor<Path> {
5      private int i;
6      public FileVisitResult preVisitDirectory(
7          Path dir, BasicFileAttributes attrs) throws IOException {
8          System.out.println(++i + ". preVisitDirectory: " + dir);
9          return FileVisitResult.CONTINUE;
10     }
11     public FileVisitResult visitFile(
  
```

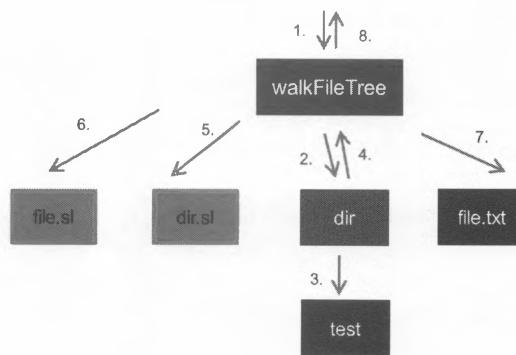
```

11         Path file, BasicFileAttributes attrs) throws IOException {
12             System.out.println(++i + ". visitFile: " + file);
13             if (attrs.isSymbolicLink()) {
14                 System.out.println(
15                     "\t--> " + file.getFileName() + " is SymbolicLink");
16             }
17             return FileVisitResult.CONTINUE;
18         }
19     public FileVisitResult visitFileFailed(
20         Path file, IOException exc) throws IOException {
21         System.out.println(++i + ". visitFileFailed: " + file);
22         return FileVisitResult.CONTINUE;
23     }
24     public FileVisitResult postVisitDirectory(
25         Path dir, IOException exc) throws IOException {
26         System.out.println(++i + ". postVisitDirectory: " + dir);
27         return FileVisitResult.CONTINUE;
28     }
29 }
30
31 public class WalkFileTreeExample {
32     public static void main(String[] args) {
33         Path path = Paths.get("dir/c09/walkFileTree").toAbsolutePath();
34         try {
35             Files.walkFileTree(path, new PrintTree());
36             //Files.walkFileTree(path, new SimplePrintTree ());
37         } catch (IOException e) {
38             System.out.println("Exception: " + e);
39         }
40     }
41 }
42 }
```

結果

1. preVisitDirectory: D:\GmJava\source\OCP\dir\c09\walkFileTree
2. preVisitDirectory: D:\GmJava\source\OCP\dir\c09\walkFileTree\dir
3. visitFile: D:\GmJava\source\OCP\dir\c09\walkFileTree\dir\test
4. postVisitDirectory: D:\GmJava\source\OCP\dir\c09\walkFileTree\dir
5. visitFile: D:\GmJava\source\OCP\dir\c09\walkFileTree\dir.sl
--> dir.sl is SymbolicLink
6. visitFile: D:\GmJava\source\OCP\dir\c09\walkFileTree\file.sl
--> file.sl is SymbolicLink
7. visitFile: D:\GmJava\source\OCP\dir\c09\walkFileTree\file.txt
8. postVisitDirectory: D:\GmJava\source\OCP\dir\c09\walkFileTree

執行順序示意圖：



❖ 圖 9-8 walkFileTree 範例路徑走訪順序

9.7 使用PathMatcher類別找尋檔案/目錄

9.7.1 搜尋檔案

在某個路徑下，若想找出所有 java 的程式碼檔案，含搜尋子目錄，在 Windows 下可以使用指令「dir /s *.java」。

在 Java 裡，則使用 `java.nio.file.PathMatcher` 介面，用來搜尋符合特定字串的路徑：

```
PathMatcher matcher =
FileSystems.getDefault().getPathMatcher(String syntaxAndPattern);
```

其中，參數 `syntaxAndPattern`，語法為：「syntax:pattern」。而 `syntax` 有二種：

1. glob 樣式，即 global command 的簡寫。
2. regex 樣式，即 regular expression 的簡寫。

「glob」相較「regex」簡單許多，且廣泛應用於檔案系統中的檔案搜尋，是本節重點。而「regex」即為正規表示式。

9.7.2 glob 樣式語法介紹

下表為 glob 樣式內常見的字元或符號的代表意義：

◆表 9-7 glob 樣式使用說明

字元	使用方式
*	任何個數的萬用字元，不跨目錄。
**	任何個數的萬用字元，跨目錄。
?	代表 1 個字元。
\	跳脫(Escape)符號。
[]	找出符合的單一字元，如： [abc] 表示 a 或 b 或 c。 [a-z] 表示可以是 a~z 的任一字元。 [abce-g] 表示 a 或 b 或 c 或 e 或 f 或 g。 [!a-c] 表示 非(a 或 b 或 c)。 [] 裡面的「*」和「?」和「\」失去特殊意義； 符號「-」若排第一個，或是僅次於「！」，也只代表符號本身。
{ }	{ } 內可以有多個 sub-pattern，用「,」區隔，滿足一個就成立。
.	檔名前面以「.」開頭，如「.login」，比較方式和一般檔案相同。另這類檔案通常都是 hidden，可以使用 Files.isHidden 測試。

下表為 glob 樣式的使用範例：

◆表 9-8 glob 樣式使用範例

樣式內容	比對符合
*.java	檔名以「.java」結尾
.*	檔名中間有「.」
*.{java,class}	檔名以「.java」或「.class」結尾
foo.?	檔名以「foo.」開頭，後面接 1 個字元
C:**	C:\foo 或 C:\bar 均符合，在 java 中，樣式為 C:*
/home/*	滿足 /home/bus (未跨路徑)
/home/**	滿足 /home/bus/data (未跨路徑)
/home/*/*	滿足 /home/bus 和 /home/bus/data (跨路徑)

以下範例「/OCP/src/course/c09/PathMatcherTest1.java」展示如何使用相關 API：

④ 範例

```

1  public class PathMatcherTest1 {
2
3      public static void main(String[] args) {
4
5          FileSystem fileSystem = FileSystems.getDefault();
6

```

```

7     Path path = Paths.get("D:/Test.java");
8     System.out.println(path);
9
10    PathMatcher pathMatcher1 =
11        fileSystem.getPathMatcher("glob:D:/*.java");
12    System.out.println(pathMatcher1.matches(path));
13
14    PathMatcher pathMatcher2 =
15        fileSystem.getPathMatcher("glob:D:/*/*.java");
16    System.out.println(pathMatcher2.matches(path));
17
18    PathMatcher pathMatcher3 =
19        fileSystem.getPathMatcher("glob:D:/**/*.java");
20    System.out.println(pathMatcher3.matches(path));
}
}

```

結果

```

D:\Test.java
true
false
false

```

說明

10	滿足 D 磁碟下的 java 程式檔 (未跨目錄) , 結果 true。
13	滿足 D 磁碟下且第 1 層目錄裡的 java 程式檔 (未跨目錄) , 結果 false。
16	滿足 D 磁碟下且跨目錄 (需有目錄) 的 java 程式檔 , 結果 false。

延伸案例：

◆ 表 9-9 延伸前範例的測試結果列表

樣式內容 (glob)		D:/*.java	D:/*/*.java	D:/**/*.java
Path				
Path	D:/Test.java	TRUE	false	false
	D:/1/Test.java	false	TRUE	TRUE
	D:/1/2/Test.java	false	false	TRUE
	D:/1/2/3/Test.java	false	false	TRUE

也可以用 `Files.walkFileTree()` 架構走訪所有檔案，過程中搭配使用 `PathMatcher` 物件的 `FileVisitor` 介面實作類別，以判斷檔名是否符合 glob 樣式。如範例「/OCP/src/course/c09/PathMatcherTest2.java」：

範例

```

1  class Finder extends SimpleFileVisitor<Path> {
2      PathMatcher matcher =
3          FileSystems.getDefault().getPathMatcher("glob:" + "*.java");
4      int numMatches;
5
6      private void find(Path file) {
7          Path name = file.getFileName();
8          if (name != null && matcher.matches(name)) {
9              numMatches++;
10             System.out.println(file);
11         }
12     }
13     public FileVisitResult visitFile(Path file, BasicFileAttributes attrs) {
14         find(file);
15         return CONTINUE;
16     }
17 }
18
19 public class PathMatcherTest2 {
20     public static void main(String[] args) {
21         Path root = Paths.get("").toAbsolutePath();
22         System.out.println("root: " + root);
23         Finder finder = new Finder(); //implements FileVisitor
24         try {
25             Files.walkFileTree(root, finder);
26         } catch (IOException e) {
27             System.out.println("Exception: " + e);
28         }
29         System.out.println("----\n" + finder.numMatches + " found!!!");
30     }
31 }
```

9.8 其他

9.8.1 使用 FileStore 類別

類別 FileStore 用來提供檔案系統的使用狀況，如同 Windows 作業系統裡的磁碟、磁區，或是 Linux 作業系統的掛載點 (mount point)，可以取得總容量、已用空間、未用空間等數據，如範例「/OCP/src/course/c09/DiskUsage.java」：

範例

```

1  public class DiskUsage {
2      static void printFileStore(FileStore store) throws IOException {
3          long toGB = 1024 * 1024 * 1024;
4          long total = store.getTotalSpace() / toGB;
5          long used = store.getTotalSpace() / toGB -
6              store.getUnallocatedSpace() / toGB;
7          long avail = store.getUsableSpace() / toGB;
8          System.out.format(
9              "%-20s %12d(GB) %12d(GB) %12d(GB)\n",
10             store.toString(), total, used, avail);
11     }
12
13     public static void main(String[] args) throws IOException {
14         System.out.format(
15             "%-20s %12s %12s %12s\n",
16             "Filesystem", "total", "used", "avail");
17         if (args.length == 0) {
18             for (FileStore store :
19                 FileSystems.getDefault().getFileStores()) {
20                 printFileStore(store);
21             }
22         } else {
23             for (String file : args) {
24                 FileStore store = Files.getFileStore(Paths.get(file));
25                 printFileStore(store);
26             }
27         }
28     }
29 }
```

結果

Filesystem	total	used	avail
(C:)	100 (GB)	88 (GB)	12 (GB)
新增磁碟區 (D:)	138 (GB)	51 (GB)	87 (GB)

說明

- | | |
|----|--------------------------|
| 13 | 取得檔案系統裡所有 FileStore 物件。 |
| 18 | 根據 Path 取得 FileStore 物件。 |

9.8.2 使用 WatchService

介面 WatchService 可以用來監控目錄 Path 內的檔案何時被新增、刪除、修改，可參照範例「/OCP/src/course/c09/WatchServiceTest.java」：

範例

```

1  class MyWatchService implements Runnable {
2      private WatchService myWatcher;
3      public MyWatchService(WatchService myWatcher) {
4          this.myWatcher = myWatcher;
5      }
6      @Override
7      public void run() {
8          try {
9              WatchKey key = myWatcher.take();
10             while (key != null) {
11                 for (WatchEvent event : key.pollEvents()) {
12                     System.out.printf(
13                         "Received event: %s for file: %s\n",
14                         event.kind(), event.context());
15                 }
16                 key.reset();
17                 key = myWatcher.take();
18             }
19         } catch (InterruptedException e) {
20             e.printStackTrace();
21         }
22     }
23
24     public class WatchServiceTest {
25         final static String DIRECTORY_TO_WATCH = "D://WatchServiceTest";
26         public static void main(String[] args) throws Exception {
27
28             Path watchPath = Paths.get(DIRECTORY_TO_WATCH);
29             if (Files.exists(watchPath) == false) {
30                 Files.createDirectories(watchPath);
31             }
32
33             WatchService myWatcher = watchPath.getFileSystem().newWatchService();
34
35             MyWatchService fileWatcher = new MyWatchService(myWatcher);

```

```

36     Thread thread = new Thread(fileWatcher);
37     thread.start();
38
39     // register a file
40     watchPath.register( myWatcher,
41                         ENTRY_CREATE,
42                         ENTRY_MODIFY,
43                         ENTRY_DELETE);
44     thread.join();
45 }
```

9.8.3 由基礎 I/O 轉換至 NIO.2

在 JDK.7，傳統的 `java.io.File` 類別新增方法使其可以轉換至 NIO.2：

```
Path path = file.toPath();
Files.delete (path);
```

`Path` 也可以轉換至傳統的 `java.io.File` 物件：

```
File file = path.toFile();
```

方便我們由基礎 I/O 升級 NIO.2。

9.9 認證考試命題範圍

依據甲骨文公布的考試範圍 (https://education.oracle.com/java-se-8-programmer-ii/pexam_1Z0-809)，和本章相關的主題有：

Java File I/O(NIO.2)

1. Use **Path** interface to operate on file and directory paths.
2. Use **Files** class to check, read, delete, copy, move, manage metadata of a file or directory.

本章擬真試題實戰

考題 1

Given:

```
public static void main(String[] args) {  
    Path path = Paths.get("\\sales\\quarter\\...\\quarterReport.txt");  
    path.relativize(Paths.get("\\sales\\annualReport.txt"));  
    if (path.endsWith("annualReport.txt")) {  
        System.out.println(true);  
    } else {  
        System.out.println(false);  
    }  
    System.out.println(path);  
}
```

What is the result?

A. false

\sales\quarter\...\\quarterReport.txt

B. false

\quarter\\..\\quarterReport.txt

C. true

..\\..\\..\\annualReport.txt

D. true

\\..\\..\\annualReport.txt

答案 A

考題 2

Given:

```
public static void main(String[] args) {
    String source = "D:\\company\\info.txt";
    String dest = "D:\\company\\emp\\info.txt";

    // insert codes here

} catch (Exception e) {
    e.printStackTrace();
}

}
```

Which two try statements, when inserted at // insert codes here, will enable the code to:

1. Successfully move the file info.txt to the destination directory,
2. Even if a file by the same name already exists in the destination directory.

- A.

```
try (FileChannel in = new FileInputStream(source).getChannel();
      FileChannel out = new FileOutputStream(dest).getChannel()) {
    in.transferTo(0, in.size(), out);
```
- B.

```
try {
    Files.copy(Paths.get(source), Paths.get(dest));
    Files.delete(Paths.get(source));
```
- C.

```
try {
    Files.copy(Paths.get(source), Paths.get(dest),
               StandardCopyOption.REPLACE_EXISTING);
    Files.delete(Paths.get(source));
```
- D.

```
try {
    Files.move(Paths.get(source), Paths.get(dest));
```
- E.

```
try (BufferedReader br = Files.newBufferedReader(Paths.get(source),
                                                    Charset.forName("UTF-8"));
      BufferedWriter bw = Files.newBufferedWriter(Paths.get(dest),
                                                Charset.forName("UTF-8"))); {
    String record = "";
    while ((record = br.readLine()) != null) {
        bw.write(record);
        bw.newLine();
    }
    Files.delete(Paths.get(source));
```

答案 CE

說明 選項 A：將檔案內容由 source 複製到 dest，檔案並未移動。選項 B：拋出 java.nio.file.FileAlreadyExistsException。選項 D：拋出 java.nio.file.FileAlreadyExistsException。

考題 3

There are below files in a directory:

Log-Jan 2009
 log_01_2010
 log_Feb2010
 log_Feb2011
 log_10.2012
 log-sum-2012

How many files does the matcher in this fragment match?

```
PathMatcher matcher =
    FileSystems.getDefault ().getPathMatcher ("glob:*???_*1?");
```

- A. One
- B. Two
- C. Three
- D. Four
- E. Five
- F. Six

答案 B**說明**

1. 該 glob pattern 的解讀重點為「_ 前必須有三個字元(???)」。所以二個檔案排除：

Log-Jan 2009
 log-sum-2012

2. 結尾是「1?」，倒數第二個字元必須是數字的「1」。所以二個檔案再排除：

log_01_2010
 log_Feb2010

其倒數第二個字元是英文字母「L」的小寫「l」。

考題 4

Given the following files in test directory:

- index.htm
- service.html
- logo.gif
- userguide.txt

And the code fragment:

```
class Finder extends SimpleFileVisitor<Path> {  
    PathMatcher matcher =  
        FileSystems.getDefault().getPathMatcher("glob:*.htm,html,xml");  
    private void find(Path file) {  
        Path name = file.getFileName();  
        if (name != null && matcher.matches(name)) {  
            System.out.println(file);  
        }  
    }  
    public FileVisitResult visitFile(Path file, BasicFileAttributes attrs) {  
        find(file);  
        return CONTINUE;  
    }  
}
```

And:

```
public static void main(String[] args) {  
    Path root = Paths.get("test");  
    try {  
        Files.walkFileTree(root, new Finder());  
    } catch (IOException e) {  
        System.out.println("Exception: " + e);  
    }  
}
```

What is the result, if test is present in the current directory?

A. No output is produced.

B. index.htm

C. index.htm
userguide.txt
logo.gif

```
D. index.htm
    service.html
    userguide.txt
    logo.gif
```

答案 A

說明 glob 樣式的語法不對，若改為「glob:*.{htm,html,xml}」，則結果為：

```
index.htm
service.html
```

考題 5

Given:

```
public static void main(String[] args) throws IOException {
    Path p1 = Paths.get("D://biz.txt");
    Path p2 = Paths.get("D://company");
    // insert code here
}
```

Which code fragment, when inserted independently at // insert code here, will:

1. Move the biz.txt file to the company directory.
 2. At the same level, replacing the file if it already exists.
- A. Files.move(p1, p2, StandardCopyOption.REPLACE_EXISTING,
StandardCopyOption.ATOMIC_MOVE);
 - B. Files.move(p1, p2, StandardCopyOption.REPLACE_EXISTING,
LinkOption.NOFOLLOW_LINKS);
 - C. Files.move(p1, p2, StandardCopyOption.REPLACE_EXISTING,
StandardCopyOption.COPY_ATTRIBUTES,
StandardCopyOption.ATOMIC_MOVE);
 - D. Files.move(p1, p2, StandardCopyOption.REPLACE_EXISTING,
StandardCopyOption.COPY_ATTRIBUTES,
LinkOption.NOFOLLOW_LINKS);

答案 B

說明

選項 A：含 StandardCopyOption.ATOMIC_MOVE 將拋出

java.nio.file.AccessDeniedException: D:\biz.txt -> D:\company

選項 C、D：含 StandardCopyOption.COPY_ATTRIBUTES 將拋出

java.lang.UnsupportedOperationException: Unsupported copy option

考題 6

Given:

```
public static boolean checkContent() throws IOException {
    Path p1 = Paths.get("exam/p1/report.txt");
    Path p2 = Paths.get("exam/p2/report.txt");
    Files.copy(p1, p2, StandardCopyOption.REPLACE_EXISTING,
               StandardCopyOption.COPY_ATTRIBUTES,
               LinkOption.NOFOLLOW_LINKS);
    if (Files.isSameFile(p1, p2)) {
        return true;
    } else {
        return false;
    }
}
public static void main(String[] args) {
    try {
        boolean flag = checkContent();
        if (flag) {
            System.out.println("Equal");
        } else {
            System.out.println("Not Equal");
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

What is the result if both files existed?

- A. The program replaces the file contents and the file's attributes and prints Equal.
- B. The program replaces the file contents as well as the file attributes and prints Not Equal.
- C. An UnsupportedOperationException is thrown at runtime.
- D. The program replaces only the file attributes and prints Not Equal.

答案 B**說明**

1. 檔案確實已經被複製，因此兩者內容相同，但不能說是相同檔案。
2. `Files.isSameFile(p1, p2)` 通常用於 `symlink`，確認連結來源和連結本身是否指向相同檔案。

考題 7

An application is waiting for notification of changes to a test directory using the following code statements:

```
Path dir = Paths.get("test")
WatchKey key = dir.register (watcher, ENTRY_CREATE,
                            ENTRY_DELETE,
                            ENTRY_MODIFY) ;
```

In the test directory, the user renames the file from fileA to fileB,

Which statement is true?

- A. The events received and the order of events are consistent across all platforms.
- B. The events received and the order of events are consistent across all Microsoft Windows versions.
- C. The events received and the order of events are consistent across all UNIX platforms.
- D. The events received and the order of events are platform dependent.

答案 D

說明 參考 API 文件的說明：<https://docs.oracle.com/javase/8/docs/api/java/nio/file/WatchService.html>：

Platform dependencies

The implementation that observes events from the file system is intended to map directly on to the native file event notification facility where available, or to use a primitive mechanism, such as polling, when a native facility is not available. Consequently, many of the details on how events are detected, their timeliness, and whether their ordering is preserved are highly implementation specific. For example, when a file in a watched directory is modified then it may result in a single `ENTRY_MODIFY` event in some implementations but several events in other implementations. Short-lived files (meaning files that are deleted very quickly after they

are created) may not be detected by primitive implementations that periodically poll the file system to detect changes.

考題 8

Given:

```
static String displayNamePart(String path, int location) {  
    Path p = new File(path).toPath();  
    String name = p.getName(location).toString();  
    return name;  
}  
public static void main(String[] args) {  
    String path = "program//doc//index.htm";  
    String result = displayNamePart(path, 2);  
    System.out.print(result);  
}
```

What is the result?

- A. doc
- B. index.htm
- C. IllegalArgumentException is thrown at runtime.
- D. InvalidPathException is thrown at runtime.
- E. Compilation fails.

答案 B

說明

p.getName(0).toString() : program
p.getName(1).toString() : doc
p.getName(2).toString() : index.htm

考題 9

Given:

```
public static void main(String[] args) {  
    Path path = Paths.get("D:\\education\\school\\student\\report.txt");  
    System.out.printf("getName(0):%s\n", path.getName(0));  
    System.out.printf("subpath(0,2):%s", path.subpath(0, 2));  
}
```

What is the result?

- A. getName(0): D:\
subpath(0, 2): D:\education\report.txt
- B. getName(0): D:\
subpath(0, 2): D:\education
- C. getName(0): education
subpath(0, 2): education\school
- D. getName(0): education
subpath(0, 2): education\school\student
- E. getName(0): report.txt
subpath(0, 2): insritute\student

答案 C

說明 參見「9.2.2 Path 介面主要功能」。

考題 10

Given:

```
public static void main(String[] args) {
    Path file = Paths.get("D:\\\\");
    try {
        // insert codes here
    } catch (Exception e) {
    }
}
```

And suppose this is a DOS-based file system.

Which option, containing statement(s), inserted at // insert codes here, will create the file and set its attributes to hidden and read-only?

- A. DosFileAttributes attrs = Files.setAttribute(file, "dos:hidden", "dos: readonly");
Files.createFile(file, attrs);
- B. Files.createFile(file);
Files.setAttribute(file, "dos:hidden", "dos:readonly");
- C. Files.createFile(file, "dos:hidden", "dos:readonly");
- D. Files.createFile(file);
Files.setAttribute(file, "dos:hidden", true);
Files.setAttribute(file, "dos:readonly", true);

答案 D

說明 選項 A、C：無法通過編譯。選項 B：可通過編譯，但參數使用錯誤。應該如同 D。

考題 11

When using the default file system provider with a JVM running on a DOS-based file system, which statement is true?

- A. DOS file attributes can be read as a set in a single method call.
- B. DOS file attributes can be changed as a set in a single method call.
- C. DOS file attributes can be modified for symbolic links and regular files.
- D. DOS file attributes can be modified in the same method that creates the file.

答案 A

考題 12

Given:

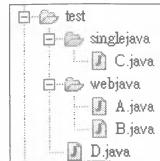
```
class Finder extends SimpleFileVisitor<Path> {  
    private final PathMatcher matcher;  
    private static int numMatches = 0;  
    Finder() {  
        matcher = FileSystems.getDefault().getPathMatcher("glob:*java");  
    }  
    void find(Path file) {  
        Path name = file.getFileName();  
        if (name != null && matcher.matches(name))  
            numMatches++;  
    }  
    void report() {  
        System.out.println("Matched:" + numMatches);  
    }  
    @Override  
    public FileVisitResult visitFile(Path file, BasicFileAttributes attrs) {  
        find(file);  
        return FileVisitResult.CONTINUE;  
    }  
  
    public static void main(String[] args) throws IOException {
```

```

        Finder finder = new Finder();
        Files.walkFileTree(Paths.get("test"), finder);
        finder.report();
    }
}

```

and below directory and file structure:



❖ 圖 9-9 Exam 12

What is the result?

- A. Compilation fails
- B. 6
- C. 4
- D. 1
- E. 3
- F. Not possible to answer due to missing exhibit.

答案 C

說明 因為 Finder 類別只覆寫 visitFile() 方法，因此只會檢查檔案名稱是否滿足 glob 樣式，故有四個 *.java 的檔案滿足。

考題 13

Given the directory structure that contains three directories: Company, Salesdat, and Finance:
Company

- Salesdat
 - * Target.dat
- Finance
 - * Salary.dat
 - * Annual.dat

And:

```
class FileFinder extends SimpleFileVisitor<Path> {
    private final PathMatcher matcher;
    FileFinder() {
        matcher = FileSystems.getDefault().getPathMatcher("glob:*dat");
    }
    void find(Path file) {
        Path name = file.getFileName();
        if (name != null && matcher.matches(name)) {
            System.out.println(name);
        }
    }
    public FileVisitResult visitFile(Path file, BasicFileAttributes atr) {
        find(file);
        return FileVisitResult.CONTINUE;
    }
    public static void main(String[] args) throws IOException {
        FileFinder obj = new FileFinder();
        Files.walkFileTree(Paths.get("Company"), obj);
    }
}
```

If Company is the current directory, what is the result?

- A. Annual.dat
- B. Salesdat,
Annual.dat
- C. Annual.dat,
Salary.dat,
Target.dat
- D. Prints at least :
Salesdat,
Annual.dat,
Salary.dat,
Target.dat

答案 C

說明 因為 FileFinder 類別只覆寫 visitFile() 方法，因此只會檢查檔案名稱是否滿足 glob 樣式，故有三個 *.dat 的檔案滿足。

考題 14

Which is false regarding the java.nio.file.Path Interface?

- A. The interface extends WatchService interface
- B. Implementations of this interface are immutable.
- C. Implementations of this interface are safe for use by multiple concurrent threads.
- D. Paths associated with the default provider are generally interoperable with the java.io.File class.

答案 A

說明 根據 Java API 文件的說明：<https://docs.oracle.com/javase/7/docs/api/java/nio/file/Path.html>，選項 B、C、D 均正確，選項 A 須改為 Watchable。

考題 15

Which two methods are defined in the FileStore class print disk space information?

- A. getTotalSpace()
- B. getFreeSpace ()
- C. getUsableSpace ()
- D. getTotalCapacity ()
- E. getUsed ()

答案 AC

說明 參考範例「/OCP/src/course/c09/DiskUsage.java」。

考題 16

There are /green.txt and /colors/blue.txt files and given:

```
public static void main(String[] args) throws IOException {
    Path from = Paths.get("green.txt");
    Path to = Paths.get("colors/blue.txt");
    Files.move(from, to, StandardCopyOption.ATOMIC_MOVE);
    Files.delete(from);
}
```

Which statement is true?

- A. The green.txt file content is replaced by the blue.txt file content and the blue.txt file is deleted.
- B. The blue.txt file content is replaced by the green.txt file content and an exception is thrown.
- C. The file green.txt is moved to the colors directory.
- D. A FileAlreadyExistsException is thrown at runtime.

答案 B

說明 使用 Files.move() 會使原本檔案 (source) 不見，再呼叫 delete() 會拋出：

```
java.nio.file.NoSuchFileException: green.txt
```

考題 17

Given:

```
public static void deleteClass(String dir) throws IOException {  
    File[] files = new File(dir).listFiles();  
    if (files != null && files.length > 0) {  
        for (File f : files) {  
            if (f.isDirectory()) {  
                deleteClass(f.getAbsolutePath());  
            } else {  
                if (f.getName().endsWith(".class"))  
                    f.delete();  
            }  
        }  
    }  
}
```

And:

```
public static void main(String[] args) throws IOException {  
    deleteClass("dir0");  
}
```

Assume that "dir0" contains subdirectories that contain .class files and is passed as an argument to the deleteClass () method when it is invoked.

What is the result?

- A. The method deletes all the .class files in the "dir0" directory and its subdirectories.

- B. The method deletes the .class files of the "dir0" directory only.
- C. The method executes and does not make any changes to the "dir0" directory.
- D. The method throws an IOException.

答案 A

考題 18

Given:

```
public static void main(String[] args) {
    Path p1 = Paths.get("/app./sys/");
    Path r1 = p1.resolve("logsome");
    System.out.println(r1);
    Path p2 = Paths.get("/server/exe/");
    Path r2 = p2.resolve("/readsome/");
    System.out.println(r2);
}
```

- A. /app/sys/logsome
/readme/server/exe
- B. /app/log/sys
/server/exe/readsome
- C. /app./sys/logsome
/readsome
- D. /app./sys/logsome
/server/exe/readme

答案 C

考題 19

Given:

```
public static void main(String[] args) throws IOException {
    Path from = Paths.get("data/xx/log.txt");
    Path destination = Paths.get("data/");
    Files.copy(from, destination);
}
```

Known that the file /data/xx/log.txt is accessible and contains some texts.

What is the result?

- A. A file with the name log.txt is created in the data directory and the content of the data/xx/log.txt file is copied to it.
- B. The program executes successfully and does NOT change the file system.
- C. A FileNotFoundException is thrown at run time.
- D. A FileAlreadyExistsException is thrown at run time.

答案 D

說明 選項 A 必須修改題目 Path destination = Paths.get("data/log.txt");

考題 20

Given:

```
public static void main(String[] args) {  
    Path p = Paths.get("/Pics/MyPicture.jpg");  
    System.out.println(  
        p.getNameCount() + ":" + p.getName(1) + ":" + p.getFileName());  
}
```

Assume that the Pics directory does NOT exist.

What is the result?

- A. An exception is thrown at run time.
- B. 2:MyPicture.jpg: MyPicture.jpg
- C. 1:Pics:/Pics/ MyPicture.jpg
- D. 2:Pics: MyPicture.jpg

答案 B

執行緒

-
- | 10.1 執行緒介紹
 - | 10.2 執行緒常見的問題
 - | 10.3 執行緒的synchronized 與等待
 - | 10.4 其他執行緒方法介紹
 - | 10.5 認證考試命題範圍

10.1 執行緒介紹

10.1.1 名詞說明

先佔式多工 (Preemptive multitasking)

現代電腦要執行的程式個數，經常遠多於 CPU 核數。為了讓程式都可以有機會執行，每個要執行的任務會被分配到一小段 CPU 時間 (time slice)，讓每個任務都可以分享到 CPU 資源來完成工作。

CPU 時間通常都是以毫秒 (milliseconds) 來計算。一旦使用完畢，該任務就暫停執行，等待下一次分配。



課堂小祕訣

「先佔式多工」的做法就和我們上餐館用餐的情境相似。當餐廳生意很好，有多桌客人在等待上菜的時候，通常會輪桌上菜，避免客人等待過久。

任務排程 (Task Scheduling)

大部分的作業系統都支援多工 (multitasking)，把 CPU 時間分配給所有「執行程式」。程式的二個重要組成是：

1. Process(程序)

- 擁有 memory(記憶體) 來儲存 data(資料) 和 code(程式碼) 。
- 使用 thread(執行緒) 接受分配 CPU 時間以執行程式。

2. Thread(執行緒)

- Process 可同時擁有多個 threads 各司其職；這些 threads 共享 memory 裡的 data 。

10.1.2 常見的效能瓶頸

要讓程式快速執行，必須避免「效能瓶頸 (performance bottlenecks)」。常見的瓶頸有：

1. 資源競爭 (Resource Contention)

多個任務搶奪同一獨佔資源，未搶到必須等待。

2. 輸出 / 輸入操作阻礙 (I/O Operations Blocking)

通常是等待硬碟或網路傳輸資料。

3. CPU 資源未充分使用 (Underutilization of CPUs)

單一執行緒程式只用到單核 CPU。

10.1.3 執行緒類別

Java 裡使用類別 Thread 來啓動 threads(多執行緒)。有二種建立方式：

1. 直接繼承 Thread 類別，好處是較簡單。
2. 實作 Runnable 介面。好處是較有彈性，可以再繼承其他類別。

以下程式碼顯示如何以「直接繼承 Thread 類別」來建立執行緒類別：

1. 繼承 java.lang.Thread。
2. 覆寫 run() 方法。

```
class ExampleThread extends Thread {
    @Override
    public void run() {
        for (int i = 0; i < 100; i++) {
            System.out.println("i:" + i);
        }
    }
}
```

若要啓用執行緒，要呼叫 start() 方法，Java 會啓動獨立執行緒執行 run() 方法內容。若直接呼叫 run() 方法，將和一般方法無異：

```
public static void main(String[] args) {
    ExampleThread t1 = new ExampleThread();
    t1.start();
}
```

若要以「實作 Runnable 介面」的方式建立執行緒類別，則可以用以下方式：

1. 實作 java.lang.Runnable 介面。
2. 覆寫 run() 方法。

```
class ExampleRunnable implements Runnable {
    @Override
    public void run() {
```

```

        for (int i = 0; i < 100; i++) {
            System.out.println("i:" + i);
        }
    }
}

```

若要使用實作 Runnable 介面的類別啓動執行緒，必須將其實作放入 Thread 類別的建構子，並使用 start() 啓動：

```

public static void main(String[] args) {
    ExampleRunnable r1 = new ExampleRunnable();
    Thread t1 = new Thread(r1);
    t1.start();
}

```

10.2 執行緒常見的問題

執行緒常遇到的問題歸咎於三類原因，分別是：

1. 使用分享的資料 (shared data)。
2. 使用可分段的方法 (non-atomic function)。
3. 使用快取的資料 (cached data)。

將分別介紹如後。

10.2.1 使用 Shared Data 可能造成的問題

執行緒會潛在共用 static 和 instance 欄位，必須注意可能造成的問題：

1. 執行緒物件目的在執行其 run() 方法。若多個執行緒都要執行 run()，就要注意該方法共用的部分，如 instance 欄位，會被同時 (concurrently) 存取。
2. static 欄位原本就是分享的資料，也無法避免同時被存取的情況。

如範例「/OCP/src/course/c10/ExampleRunnable.java」：

範例

```

1  public class ExampleRunnable implements Runnable {
2      private int i;    // 將被共用 !!
3
4      @Override

```

```

5     public void run() {
6         for (i = 0; i < 10; i++) {
7             System.out.print("i:" + i + ", ");
8         }
9     }
10
11    public static void main(String[] args) {
12        ExampleRunnable r1 = new ExampleRunnable();
13        Thread t1 = new Thread(r1);
14        t1.start();
15        Thread t2 = new Thread(r1);
16        t2.start();
17    }
18 }
```

執行前述範例，「可能」得到以下結果：

意外結果

```
i:0, i:0, i:1, i:2, i:3, i:4, i:5, i:6, i:7, i:8, i:9,
```

和我們預期的有落差，將在下一小節探究原因：

預期結果

```
i:0, i:1, i:2, i:3, i:4, i:5, i:6, i:7, i:8, i:9,
```

static 和 instance 欄位資料被多個執行緒共用而出現執行異常時，IDE 如 Eclipse 等是無法警報的！因此安全地 (safely) 處理被分享的資料就成為程式設計師的義務。

此外，資料若因為多個執行緒同時存取而產生錯誤，一般而言不好處理：

1. 執行緒的分配由作業系統決定，程式設計師無法干預。
2. 每台機器的 CPU 效能、個數不盡相同。
3. 其他程式也會占用 CPU 時間。

因此，有可能在測試環境時沒有異常，但部署到正式環境後卻經常發生奇怪狀況。這類問題不易處理，因此應儘可能使用「執行緒安全 (thread-safe)」的設計，減少「shared data」的使用。

類別內有些資料不會被多執行緒分享，因此永遠都是「thread-safe」，如：

1. 區域變數 (local variables)。
2. 方法參數 (method parameters)。
3. 例外處理參數 (exception handler parameters)。

10.2.2 使用 Non-Atomic Operations 可能造成的問題

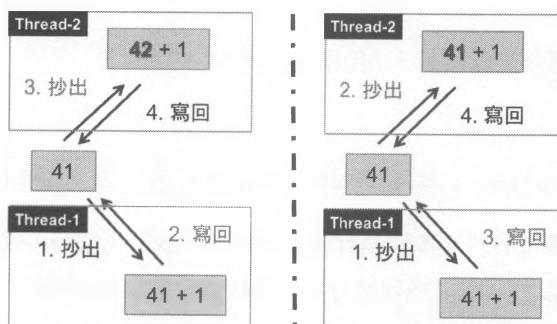
「atomic function」用原子的概念來描述一個 function。因為「原子」是極小的粒子，無法再分割，所以代表「single function」，亦即該 function 只有一個步驟。

在 Java 裡，即便程式碼只有一句敘述，也不表示就是 atomic function。以整數 i 使用遞增運算子的「 $i++$ 」為例，其實 Java 以三個步驟去執行：

1. 對整數「 i 」建立暫時副本。
2. 暫時副本增加 1。
3. 將暫時副本的結果回寫「 i 」。

此外有些 64 bit 變數的存取也可以使用二個 32 bit 的操作完成。

已知使用遞增運算子並非「atomic function」，現在使用多執行緒來進行「 $i++$ 」，並用下圖表示可能的執行結果：



◆ 圖 10-1 使用多執行緒執行遞增運算子

假設 i 為 41。經過 2 個執行緒呼叫 $i++$ 後，理想情況為 i 變為 43，如上圖左半側：

1. Thread-1 先抄出 41 的值建立副本，加 1 成為 42，再將副本寫回，因此 i 值變為 42。
2. Thread-2 再抄出 42 的值建立副本，加 1 成為 43，再將副本寫回，因此 i 值變為 43。

但是必須了解 Thread-1 和 Thread-2 是並行的執行緒，因此 Thread-2 不必等待 Thread-1 執行結束才去抄出值建立副本，也有可能四個執行緒同時去抄值，都取得 41 建立副本，然後都加 1 後寫回 42，結果就是 42，如上圖右半側示意。

10.2.3 使用 Cached Data 可能造成的問題

執行緒(thread)因程序(process)需要同時執行工作而產生。為求執行效能，執行緒在啓動時，會將程序裡的「main memory」內的分享資料複製一份放在自己的「working memory」內作為 cached copies，工作結束後寫回，如此可以避免程式進行過程中執行緒必須不斷向程序要求資料所造成的效果問題。

這樣的設計，讓執行過程中的每個執行緒各自努力，無法「即時」和其他執行緒分享自己的工作成果，必須等到工作結束。

只有以下情況發生，才能讓執行緒將各自「working memory」的異動結果寫回「main memory」：

1. 使用到「volatile」宣告的變數。
2. 使用到「synchronized」宣告的方法，亦即準備鎖定和解鎖物件 monitor。
3. 執行緒執行時的第一個動作或最後一個動作。
4. 執行緒啓動或執行緒結束時。

因此，若程式的設計是需要多執行緒在工作的過程中，仍能互相溝通訊息，就必須善用上述四個條件，特別是使用「volatile」關鍵字宣告。我們來了解它的由來和使用方式：

1. 在程式設計的領域裡，若資料經常維持不變，可以將之固定在記憶體裡，或複製出來使用，稱為「快取複製(cached copies)」。
2. 單字「volatile」解釋為「易變的，反覆無常的」。加上這個宣告，等於告訴java該欄位經常有變化，不適合產生快取複製，因此所有執行緒將接存取同一份資料。如：

```
volatile int i;
```

3. 必須了解宣告 volatile 只是不產生快取複製，和執行緒安全是兩回事，還是必須利用前兩節的做法來保證執行緒安全。volatile 宣告可以應用在「精準終止執行緒的執行」，如範例「/OCP/src/course/c10/StopThreadExample.java」。

範例

```
1  class MyRunnable implements Runnable {
2      public volatile boolean running = true;
3      @Override
4      public void run() {
5          System.out.println("Thread started");
6          while (running) {
```

```

7          // ...
8      }
9      System.out.println("Thread finishing");
10 }
11 }
12
13 public class StopThreadExample {
14     public static void main(String[] args) {
15         MyRunnable r1 = new MyRunnable();
16         Thread t1 = new Thread(r1);
17         t1.start();
18         // ...
19         r1.running = false;      // 馬上停止執行緒執行!
20     }
21 }
```

執行緒「t1」在啓動前預設會自己複製一份變數 running(值為 true) 作為 cached copy；所以若沒宣告變數 running 是 volatile，即便在主執行緒「main」將 running 改為 false，執行緒「t1」不一定會馬上知道，必須等到有事件觸發讓 working memory 和 main memory 同步，執行緒「t1」才會收到通知，才會停止。

10.3 執行緒的synchronized與等待

10.3.1 使用 synchronized 關鍵字

要建立執行緒安全的程式，除了儘量使用執行緒安全的變數之外，就是使用「synchronized」關鍵字宣告方法或是更小的程式碼區塊：

- 和 volatile 宣告有類似的功用。執行緒在執行該區塊的最初和最後時，會將變數值寫回 main memory。
- 該區塊為獨佔執行 (exclusive execution)，亦即同一時間只允許一個執行緒使用，可解決 non-atomic 的問題，所以區塊內為執行緒安全。

執行緒取得獨佔執行權的機制是：

- 每個 Java 物件都有一個「object monitor」，執行緒可以對它進行鎖定 (lock) 和解鎖 (unlock)。若鎖定成功，表示取得該物件的獨佔執行權，此時其他執行緒無法使用該物件的 synchronized 程式區塊，等同單一執行緒的環境。

2. 要使用宣告 `synchronized` 的方法，就必須取得「this」的 object monitor。
3. 要使用宣告 `static` 的 `synchronized` 方法，同理也必須取得類別的「class monitor」。
4. 要使用宣告 `synchronized` 區塊，必須指定要使用哪個物件的 monitor。如：

```
synchronized ( this ) {
    ...
}
```

5. 使用 `synchronized` 區塊可以有巢狀 (nested) 結構，且可以使用不同的 object monitor。

範例「/OCP/src/course/c10/SynchronizedTest.java」展示 `synchronized` 的方法特性：

④ 範例

```
1  class SynchronizedAll {
2      private void sleep() {
3          try {
4              Thread.sleep(5000);
5          } catch (InterruptedException e) {}
6      }
7      public synchronized void m1() {
8          sleep();
9          System.out.println("-- Run m1() at: " + new Date());
10     }
11     public synchronized void m2() {
12         sleep();
13         System.out.println("-- Run m2() at: " + new Date());
14     }
15 }
16
17 class M1Runner extends Thread {
18     SynchronizedAll o;
19     M1Runner(SynchronizedAll o) {
20         this.o = o;
21     }
22     public void run() {
23         o.m1();
24     }
25 }
26
27 class M2Runner extends Thread {
28     SynchronizedAll o;
29     M2Runner(SynchronizedAll o) {
30         this.o = o;
31     }
```

```

32     public void run() {
33         o.m2();
34     }
35 }
36
37 public class SynchronizedTest {
38     public static void main(String[] args) {
39         SynchronizedAll o = new SynchronizedAll();
40         System.out.println("Start main at: " + new Date());
41         Thread m1 = new M1Runner(o);
42         m1.start();
43         Thread m2 = new M2Runner(o);
44         m2.start();
45         System.out.println("End main at: " + new Date());
46     }
47 }

```

說明

1~15	建立 SynchronizedAll 類別，將二個 public 的方法都宣告為 synchronized。如此建立出來的物件，m1() 和 m2() 方法將同時只能有一個被呼叫執行。
17	建立執行緒類別 M1Runner，在建構子傳入一個 SynchronizedAll 物件後，呼叫 m1() 方法。
27	建立執行緒類別 M2Runner，在建構子傳入一個 SynchronizedAll 物件後，呼叫 m2() 方法。
39	建立一個 SynchronizedAll 物件。
41~42	建立 M1Runner 執行緒物件，並傳入之前建立的 SynchronizedAll 物件。
43~44	建立 M2Runner 執行緒物件，並傳入和 M1Runner 所傳入的相同的 SynchronizedAll 物件。

結果

```

Start main at: Mon May 30 15:35:50 CST 2016
End main at: Mon May 30 15:35:50 CST 2016
-- Run m1() at: Mon May 30 15:35:55 CST 2016
-- Run m2() at: Mon May 30 15:36:00 CST 2016

```

因為 SynchronizedAll 內 m1() 及 m2() 方法都是 synchronized，要執行都要取得 this 的 object monitor，故同時間只能有一個方法被呼叫。

10.3.2 使用 synchronized 的時機

思考購物車範例「/OCP/src/course/c10/ShopingCart.java」的程式碼：

 範例

```

1  class Item {
2      String desc() {
3          return "...";
4      }
5  }
6
7  public class ShoppingCart {
8      private List<Item> cart = new ArrayList<>();
9      public void addItem(Item item) {
10         cart.add(item);
11     }
12     public void removeItem(int index) {
13         cart.remove(index);
14     }
15     public String getSummary() {
16         StringBuilder note = new StringBuilder();
17         Iterator<Item> iter = cart.iterator();
18         while (iter.hasNext()) {
19             Item i = iter.next();
20             note.append("Item:" + i.desc() + "\n");
21         }
22         return note.toString();
23     }
24 }
```

假如某個執行緒呼叫購物車的 `getSummary()` 方法時，正好另一個執行緒在使用 `addItem()` 或 `removeItem()` 方法，會有甚麼情況？

或許有些人主張 `getSummary()` 的結果應該納入 `addItem()` 和 `removeItem()` 的影響，但有些人不這樣認爲。Java 對於這類模稜兩可的情況，處理的方式是直接拋出 `Exception`！亦即一旦啓動集合物件的 `iteration`，若 Java 偵測到集合物件的內容將被同時修改（不限定是否多執行緒），就會馬上拋出 `java.util.ConcurrentModificationException`，此為「fail-fast」行為模式（對於錯誤，或是可能造成錯誤的情況馬上作出反應）。

爲了避免這類「fail-fast」的狀況發生，就應該避免在執行 `getSummary()` 方法時，其他二個方法被同時呼叫！所以可以將這三個方法都宣告爲 `synchronized`，如此要呼叫方法前必須取得 `ShoppingCart` 物件的唯一 `object monitor`，就不會有被同時執行的可能性。

另外，`ConcurrentModificationException` 並非只有在多執行緒的情況下才會發生。範例「/OCP/src/course/c10/ConcurrentModificationExceptionTest.java」顯示了另外的可能性：

 範例

```

1  public class ConcurrentModificationExceptionTest {
2      public static void main(String[] args) {
3          Map<Integer, String> map = new HashMap<>();
4          map.put(1, "a");
5          map.put(2, "b");
6          map.put(3, "c");
7
8          // fail-fast 1
9          try {
10              Iterator<Integer> iter = map.keySet().iterator();
11              while (iter.hasNext()) {
12                  Integer key = iter.next();
13                  if (key >= 2) {
14                      map.remove(key);
15                  }
16              }
17          } catch (java.util.ConcurrentModificationException e) {
18              e.printStackTrace();
19          }
20
21          // fail-fast 2
22          try {
23              for (Integer key : map.keySet()) {
24                  if (key >= 0) {
25                      map.remove(key);
26                  }
27              }
28          } catch (java.util.ConcurrentModificationException e) {
29              e.printStackTrace();
30          }
31      }
32  }

```

範例中的 fail-fast 1 和 fail-fast 2 會拋出 `ConcurrentModificationException`，是因為在走訪 `map` 物件的成員時 (使用 `iterator` 或進階 `for` 迴圈)，同時去刪除 `map` 成員所導致。要避免這類問題，可以複製一個 `map`，讓新複製的 `map` 物件用於走訪成員，再用取得的 `key` 去刪除另一個 `map` 物件的成員。

10.3.3 縮小 synchronized 的程式區塊

物件裡所有 synchronized 方法在執行前都必須取得 object monitor，因此愈多的方法使用 synchronized，或是被 synchronized 的方法內容愈長，都會造成「執行等待」。

所以，儘可能使用 synchronized 程式區塊，而非直接 synchronized 整個方法，將減少被 synchronized 的程式碼，有助減少「執行等待」的情況。

修改範例「/OCP/src/course/c10/ShopingCart.java」的 getSummary() 方法如下：

範例

```

15     public String getSummary() {
16         StringBuilder note = new StringBuilder();
17         synchronized (this) {
18             Iterator<Item> iter = cart.iterator();
19             while (iter.hasNext()) {
20                 Item i = iter.next();
21                 note.append("Item:" + i.desc() + "\n");
22             }
23         }
24         return note.toString();
25     }

```

本例可以在行 15 的方法上直接宣告 synchronized 方法，但較好的做法是檢討方法內真正影響執行緒安全的程式碼，再加上行 17 和行 23 的 synchronized 程式區塊，可以將影響區域縮小。根據行 17 的宣告，要執行本區塊必須取得 this 的 object monitor，和執行方法時需要取得的 object monitor 相同。

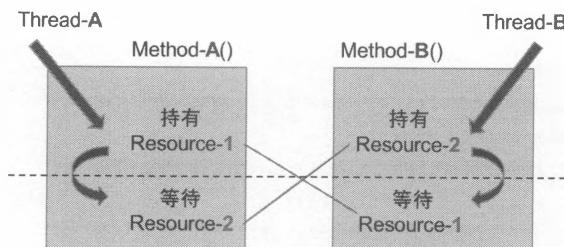
10.3.4 其他執行等待的情況

除了 synchronized 程式區塊造成的執行等待外，還有幾種情形必須知道，並設法預防：

1. Starvation：因搶不到資源而排隊。指當兩個執行緒共搶一個資源時，某一個執行緒經常可以取得資源（稱 greedy thread，貪心執行緒），導致另一個經常無法取得資源（稱 starved thread，飢餓執行緒）。如同二個學生分開掃地，但因為掃把只有一支，導致必須搶奪；又某身強力壯的同學經常可以搶到掃把，另一個同學只能常常排隊等待。
2. Live Lock：因太忙碌而排隊。當多個執行緒後因等待資源而排隊，如執行緒 B 等待執行緒 A，執行緒 C 等待執行緒 B，執行緒 D 等待執行緒 C …，等 A 完成可以輪到 B，

B 完成可以輪到 C，C 完成可以輪到 D…，等待陸續忙完，排隊等待的情況自動解開。好比十個學生排隊等待一枝掃把掃地，掃完的給下一個。

3. Dead Lock：二個執行緒互卡對方，誰都不讓，導致永遠等待。如執行緒 A 要完成工作必須先後取得「資源 1」和「資源 2」；而執行緒 B 要完成工作必須先後取得「資源 2」和「資源 1」。當執行緒 A 取得「資源 1」，準備要取「資源 2」時，發現「資源 2」在執行緒 B 手上，而執行緒 B 正缺「資源 1」，於是兩個執行緒就陷入進退兩難的僵局，永遠無法結束，除非有一方讓出資源或被終止。Deadlock 示意圖如下：



❖ 圖 10-2 Deadlock 示意圖

範例「/OCP/src/course/c10/DeadLockDemo.java」模擬 dead lock 產生的過程。兩個執行緒都刻意在方法中休眠一秒鐘，以確定可以先各自取得 resource1 和 resource2：

範例

```

1  public class DeadLockDemo {
2      private static void _sleep() {
3          try {
4              Thread.sleep(1000);
5          } catch (Exception e) {
6          }
7      }
8      public static void main(String[] args) {
9          final String resource1 = "jim1";
10         final String resource2 = "jim2";
11
12         Thread t1 = new Thread() {
13             public void run() {
14                 synchronized (resource1) {
15                     out.println("Thread 1: locked resource 1");
16                     _sleep();
17                     out.println("Thread 1: try to lock resource 2....");
18                     synchronized (resource2) {
19                         out.println("Thread 1: locked resource 2");
20                     }
21                 }
22             }
23         };
24         Thread t2 = new Thread() {
25             public void run() {
26                 synchronized (resource2) {
27                     out.println("Thread 2: locked resource 2");
28                     _sleep();
29                     out.println("Thread 2: try to lock resource 1....");
30                     synchronized (resource1) {
31                         out.println("Thread 2: locked resource 1");
32                     }
33                 }
34             }
35         };
36         t1.start();
37         t2.start();
38     }
39 }
```

```

20         }
21     }
22 }
23 };

24

25     Thread t2 = new Thread() {
26         public void run() {
27             synchronized (resource2) {
28                 out.println("Thread 2: locked resource 2");
29                 _sleep();
30                 out.println("Thread 2: try to lock resource 1....");
31                 synchronized (resource1) {
32                     out.println("Thread 2: locked resource 1");
33                 }
34             }
35         };
36     };
37
38     t1.start();
39     t2.start();
40 }
41 }
```

結果

```

Thread 1: locked resource 1
Thread 2: locked resource 2
Thread 1: try to lock resource 2....
Thread 2: try to lock resource 1....
```

課堂小秘訣

要產生 dead lock，如先前描述，至少需要二個執行緒，和二個被搶奪的資源物件。資源可以是任何物件，使用 new Object() 都可以。以本例來說，在行 9 和行 10 使用二個字串物件作為被搶奪的資源：

```
final String resource1 = "jim1";
final String resource2 = "jim2";
```

必須注意的是，字串物件若內容相同，因為字串池有重複使用的機制，所以以下二個變數實際指向同一字串物件：

```
final String resource1 = "jim";
final String resource2 = "jim";
```

因為被搶奪的資源只有一個，就無法構成 dead lock，只是 live lock。

10.4 其他執行緒方法介紹

10.4.1 使用 interrupt() 方法

除了利用「volatile」宣告的變數來達成停止執行中的執行緒外，也可以使用 `interrupt()` 方法，如範例「/OCP/src/course/c10/InterruptThreadExample.java」。

範例

```

1  class InterruptedRunnable implements Runnable {
2      @Override
3      public void run() {
4          System.out.println("Thread started");
5          while (!Thread.interrupted()) {
6              System.out.println("I am running...");
7          }
8          System.out.println("Thread finishing");
9      }
10 }
11 public class InterruptThreadExample {
12     public static void main(String[] args) {
13         InterruptedRunnable r1 = new InterruptedRunnable();
14         Thread t1 = new Thread(r1);
15         t1.start();
16         try {
17             Thread.sleep(1);
18         } catch (InterruptedException e) {}
19         t1.interrupt();
20     }
21 }
```

說明

19	對執行中的執行緒下達 <code>t1.interrupt()</code> 的指令。
5~7	執行中的執行緒可以藉由 <code>Thread.interrupted()</code> 方法不停確認是否收到中斷指令。若是，就中斷目前執行工作。

10.4.2 使用 sleep() 方法

若要使執行緒暫停一段時間，可以呼叫 `Thread` 類別的靜態 `sleep()` 方法：

```
public static native void sleep(long millis) throws InterruptedException;
```

如呼叫 `Thread.sleep(4000)` 指暫停執行四秒鐘。四秒之後再等待 CPU 分配時間，拿到才能繼續執行任務，因此停止時間為「至少」四秒鐘。

又，休眠中的執行緒隨時有被叫醒而中斷休眠的可能，所以被要求必須處理 `InterruptedException`，並在 catch block 中決定被終止休眠後要做的事。

```
long start = System.currentTimeMillis();
try {
    Thread.sleep(4000);
} catch (InterruptedException ex) {
    // What to do?
}
long time = System.currentTimeMillis() - start;
System.out.println("Slept for " + time + " ms");
```

10.4.3 使用其他方法

類別 `Thread` 上還有其他常用方法：

1. `setName(String)`，`getName()` 和 `getId()`：和執行緒的識別有關。
2. `isAlive()`：判斷執行緒是否已經結束。
3. `isDaemon()` 和 `setDaemon(boolean)`：可以將執行緒設為 daemon 或判斷是否為 daemon。執行緒預設是「non-daemon」，JVM 會等執行中的 non-daemon 執行緒都結束，才會結束；此時若還有其他 daemon 的執行緒正在執行，一樣會結束 JVM。
4. `join()`：插隊到目前執行緒的前面，執行完後才輪到目前執行緒。
5. `Thread.currentThread()`：取得執行中的執行緒。



課堂小秘訣

「daemon」的中文翻譯是「惡魔、魔鬼」的意思，或是民間常說的「魔神仔」。這些對我們而言是一種抽象、虛無的表徵，很難證明其實質影響力，因此在多執行緒的情形下，無法影響 JVM 的結束。

以下三個方法繼承自 `Object` 類別：

1. `wait()`：不限時間的等待，等候 `notify()` 方法被呼叫後醒過來。
2. `notify()` 和 `notifyAll()`：通知 `wait()` 中的執行緒。

範例「/OCP/src/course/c10/ThreadMethodTest.java」展示方法 `join()` 和 `setDaemon()` 的使用方式及影響：

 範例

```

1  class ThreadTest extends Thread {
2      public void run() {
3          try {
4              System.out.println("Thread T is starting...");
5              for (int i = 0; i < 3; i++) {
6                  Thread.sleep(1000);
7                  System.out.println("Thread T is running...");
8              }
9              System.out.println("Thread T is ending...");
10         } catch (InterruptedException e) {
11             e.printStackTrace();
12         }
13     }
14 }
15
16 public class ThreadMethodTest {
17     public static void main(String[] args) {
18         System.out.println("Thread main is starting...");
19         testNormal();
20         //testJoin();
21         //testDaemon();
22         System.out.println("Thread main is ending...");
23     }
24
25     private static void testNormal() {
26         Thread t = new ThreadTest();
27         t.start();
28     }
29
30     private static void testJoin() {
31         Thread t = new ThreadTest();
32         t.start();
33         try {
34             t.join();
35         } catch (InterruptedException e) {
36             e.printStackTrace();
37         }
38     }
39
40     private static void testDaemon() {
41         Thread t = new ThreadTest();
42         t.setDaemon(true);
43         t.start();

```

```

44         //t.setDaemon(true); //java.lang.IllegalThreadStateException
45     }
46 }
```

將行 19、20、21 的三個方法依此順序進行測試，每次只測試一個方法，關閉其他二個方法。結果分別為：

1. 只測試 testNormal() 方法

結果

```

Thread main is starting...
Thread main is ending...
Thread T is starting...
Thread T is running...
Thread T is running...
Thread T is running...
Thread T is ending...
```

在一般的情況下，執行緒 main 和 T 在啓動後各走各的，互不影響。兩者都結束後，程式結束，JVM 關閉。

2. 只測試 testJoin() 方法

結果

```

Thread main is starting...
Thread T is starting...
Thread T is running...
Thread T is running...
Thread T is running...
Thread T is ending...
Thread main is ending...
```

因為執行緒 main 啓動 Thread T，執行 t.join() 時，Thread T 會插隊到 main 前面。只有 Thread T 結束後，才會執行 main。

3. 只測試 testDaemon() 方法

結果

```

Thread main is starting...
Thread main is ending...
Thread T is starting...
```

將 Thread T 設為 daemon 後，一旦 non-daemon 的執行緒 main 結束，JVM 就關閉。所以執行緒 main 結束後，即便 Thread T 尚未結束，JVM 也會關閉，和先前的二種測試過程明顯不同。

10.4.4 不建議使用的方法

類別 Thread 有一些方法不建議使用，如：

1. 可能造成問題，避免使用：

- setPriority(int)
- getPriority()

2. 已經 deprecated，不該使用：

- destroy()
- resume()
- suspend()
- stop()

deprecated 的描述表示方法可能有問題，或命名不合傳統，未來可能移除。註記方式是在方法的宣告上加上 @Deprecated，如：

```
@Deprecated
public final void stop() {
    SecurityManager security = System.getSecurityManager();
    if (security != null) {
        checkAccess();
        if (this != Thread.currentThread()) {
            security.checkPermission(SecurityConstants.STOP_THREAD_PERMISSION);
        }
    }
    // A zero status value corresponds to "NEW", it can't change to
    // not-NEW because we hold the lock.
    if (threadStatus != 0) {
        resume(); // Wake up thread if it was suspended; no-op otherwise
    }
    // The VM can handle all thread states
    stop0(new ThreadDeath());
}
```

❖ 圖 10-3 使用 @Deprecated 註記

10.5 認證考試命題範圍

依據甲骨文公布的考試範圍 (https://education.oracle.com/java-se-8-programmer-ii/pexam_1Z0-809)，和本章相關的主題有：

Java Concurrency

1. Create worker threads using **Runnable**, **Callable** and use an **ExecutorService** to concurrently execute tasks.
2. Identify potential threading problems among **deadlock**, **starvation**, **livelock**, and **race conditions**.
3. Use **synchronized** keyword and `java.util.concurrent.atomic` package to control the order of thread execution.

本章擬真試題實戰

考題 1

Given:

```
public static void main(String[] args) {  
    Thread t1 = new Thread() {  
        public void run() {  
            System.out.println("Hi");  
        }  
    };  
    Thread t2 = new Thread(t1); // line1  
    t2.run();  
}
```

Which two are true?

- A. A runtime exception is thrown on line1.
- B. No output is produced.
- C. Hi is printed once.
- D. Hi is printed twice.
- E. No new threads of execution are started within the main method.
- F. One new thread of execution is started within the main method.
- G. Two new threads of execution are started within the main method.

答案 CE

說明 呼叫 run() 方法，將無法啓動新執行緒。必須呼叫 start()。

考題 2

Which two demonstrate the valid usage of the keyword synchronized?

- A. interface ThreadSafeA {
 synchronized void doIt();
}

```

B. abstract class ThreadSafeB {
    synchronized abstract void doIt();
}

C. class ThreadSafeC {
    synchronized static void soIt() {
    }
}

D. enum ThreadSafeD {
    ONE, TWO, Three;
    synchronized final void doIt() {
    }
}

```

答案 CD

說明 synchronized 關鍵字不可用於抽象方法。

考題 3

Given:

```

class MyThread implements Runnable {
    public void run() {
        System.out.print(Thread.currentThread().getName() + " ");
    }
}

class Launch1 {
    static MyThread t = new MyThread();

    Launch1() { new Thread(t, "const").start(); } // line1

    public static void main(String[] args) {
        new Launch1();
        new Launch2().go();
    }
}

static class Launch2 {
    void go() { new Thread(t, "inner").start(); } // line2
}
}

```

What is the result?

- A. Both const and inner will be in the output.

- B. Only const will be in the output.
- C. Compilation fails due to an error on line1.
- D. Compilation fails due to an error on line2.
- E. An Exception is thrown at runtime.

答案 A

考題 4

Given:

```
class MyThreadClass extends Thread {  
    Thread otherThread;  
    MyThreadClass(String title) {  
        super(title);  
    }  
    public void run() {  
        // use variable "otherThread" to do time-consuming stuff  
    }  
    public void setThread(Thread x) {  
        otherThread = (Thread) x;  
    }  
}
```

And:

```
public static void main(String[] args) {  
    MyThreadClass a = new MyThreadClass("Thread A");  
    MyThreadClass b = new MyThreadClass("Thread B");  
    a.setThread(b);  
    b.setThread(a);  
    a.start();  
    b.start();  
}
```

If Thread A and Thread B are running, but not completing, which two could be occurring?

- A. live lock
- B. dead lock
- C. starvation
- D. loose coupling

E. cohesion

答案 AB

說明 二個 thread 互相持有自己的參考，可能產生互相 blocking 的情況，故選項 A 和 B 有可能。選項 C：由程式碼看不出二個 thread 有資源分配不均的問題。選項 D、E：和多執行緒無關。

考題 5

Which three statements are true about thread's sleep method?

- A. The sleep (long) method parameter defines a delay in milliseconds.
- B. The sleep (long) method parameter defines a delay in microseconds.
- C. A thread is guaranteed to continue execution after the exact amount of time defined in the sleep (long) parameter.
- D. A thread can continue execution before the amount of time defined in the sleep (long) parameter.
- E. A thread can continue execution after the amount of time defined in the sleep (long) parameter
- F. Only runtime exceptions are thrown by the sleep method.
- G. A thread loses all object monitors (lock flags) when calling the sleep method.

答案 ADE

說明

選項 A、B：是以毫秒 (milliseconds) 定義。

選項 C：錯誤。無法精確到 "exact amount"。因為 thread 在 sleep 的時間到了之後，還需要等待分配 CPU 時間才能執行。

選項 D：正確。如果發生 InterruptedException，就可能提早。

選項 E：正確。

選項 F：錯誤。InterruptedException 是 checked exception。

選項 G：錯誤。根據 API 文件 [https://docs.oracle.com/javase/7/docs/api/java/lang/Thread.html#sleep\(long\)](https://docs.oracle.com/javase/7/docs/api/java/lang/Thread.html#sleep(long)) 的說明：The thread does not lose ownership of any monitors.

考題 6

Given:

```
class ThreadRun extends Thread {  
    public void run(String name) {  
        System.out.print("String");  
    }  
    public void run(Runnable r) {  
        r = new Runnable() {  
            public void run() {  
                System.out.print("Runnable");  
            }  
        };  
    }  
}
```

And:

```
public static void main(String[] args) {  
    Thread t = new ThreadRun();  
    t.start();  
}
```

Which two are true?

- A. String is printed
- B. Runnable is printed
- C. No output is produced
- D. No new threads of execution are started within the main method
- E. One new thread of execution is started within the main method
- F. Two new threads of exclusion are started within the main method

答案 CE

說明 執行緒可以正常啟動，但未覆寫方法 `public void run() { //… }`，因此沒有內容可執行。以下都不是正確方法：

```
public void run(String name) { }  
public void run(Runnable r) { }
```

都不會被執行。

考題 7

Given:

```
class LockThread extends Thread {
    int i = 10;
    public synchronized void display(LockThread obj) {
        try {
            Thread.sleep(5);
            obj.inrement(this);
        } catch (InterruptedException ex) {
            ex.printStackTrace();
        }
    }
    public synchronized void inrement(LockThread obj) {
        i++;
    }
}
```

And:

```
public static void main(String[] args) {
    final LockThread obj1 = new LockThread();
    final LockThread obj2 = new LockThread();
    new Thread(new Runnable() {
        public void run() {
            obj1.display(obj2);
        }
    }).start();
    new Thread(new Runnable() {
        public void run() {
            obj2.display(obj1);
        }
    }).start();
}
```

From what threading problem does the program suffer?

- A. deadlock
- B. livelock
- C. starvation
- D. race condition

答案 A

考題 8

Given:

```
class Container {  
    static ArrayList<Integer> list = new ArrayList<>(); // line2  
    public static void addItem() { // line3  
        list.add(0); // line4  
    }  
}  
  
class AddThread implements Runnable {  
    static Object bar = new Object();  
    public void run() { // line5  
        for (int i = 0; i < 5000; i++) { Container.addItem(); } // line6  
    }  
}
```

And:

```
public static void main(String[] args) {  
    Thread t1 = new Thread(new AddThread());  
    Thread t2 = new Thread(new AddThread());  
    t1.start(); t2.start(); // line1  
}
```

Which four are valid modifications to synchronize access to the valid list between threads t1 and t2?

A. Replace //line1 with:

```
synchronized (t2) { t1.start(); } synchronized (t1) { t2.start(); }
```

B. Replace //line2 with:

```
static CopyOnWriteArrayList<Integer> list = new CopyOnWriteArrayList<>();
```

C. Replace //line3 with:

```
synchronized public static void addItem () {
```

D. Replace //line4 with:

```
synchronized (list) { list.add(1); }
```

E. Replace //line5 with:

```
synchronized public void run () {
```

F. Replace //line6 with:

```
synchronized (this) { for (int i = 0; i<5000; i++) Container.addItem(); }
```

G. Replace //line6 with:

```
synchronized (bar) { for (int i = 0; i<5000; i++) Container.addItem(); }
```

答案 BCDG

說明 選項 A：錯誤。main 方法只有 main 執行緒會執行，synchronized 在 main() 方法的程式碼沒有意義。選項 E、F：同義，都錯誤。因為二個 thread 是各自獨立物件，只有 lock 同一個 monitor 才有效果，所以選項 G 正確。因為 lock 一個共用的 static 物件。

考題 9

Jim has designed an application. It segregates tasks that are critical and executed frequently from tasks that are non-critical and executed less frequently. He has prioritized these tasks based on their criticality and frequency of execution. After close scrutiny, he finds that the tasks designed to be non-critical are rarely getting executed.

From what kind of problem is the application suffering?

- A. race condition
- B. starvation
- C. deadlock
- D. livelock

答案 B

說明 Jim 設計的程式，依據重要性(criticality)和執行頻率(frequency)來區分程式執行先後順序(priority)。這類做法，經常導致 priority 低的 task 很難拿到執行權，稱為「starvation」。參考本書「10.3.4 其他執行等待的情況」的說明。

考題 10

Given:

```
public static void main(String[] args) {
    Thread t = new Thread(); // line1
    t.start(); // line2
    t.join(); // line3
    // ..... // line4
}
```

Which three are true?

- A. On //line3, the current thread stops and waits until the t thread finishes.

- B. On //line3, the t thread stops and waits until the current thread finishes.
- C. On //line4, the t thread is dead.
- D. On //line4, the t thread is waiting to run.
- E. This code cannot throw a checked exception.
- F. This code may throw a checked exception.

答案 ACF

說明 在//line3 時執行 t.join()，表示執行緒t將「插隊」到執行緒main前面；只有t執行結束，才會輪到main。所以在//line4的時候，執行緒t執行完畢，也結束生命週期，才輪到main執行。

考題 11

Given:

```
public static void main(String[] args) throws InterruptedException {  
    Runnable r = new Runnable() {  
        public void run() {  
            System.out.print(Thread.currentThread().getName());  
        }  
    };  
    Thread t1 = new Thread(r, "one ");  
    t1.start();  
    t1.sleep(4000);  
    Thread t2 = new Thread(r, "two ");  
    t2.start();  
    t2.sleep(2000);  
    System.out.print("more ");  
}
```

What is the most likely result?

- A. more one two
- B. more two one
- C. one two more
- D. one more two
- E. two more one

答案 C

說明 Thread.sleep(long) 是 static 方法，使用物件參考呼叫容易讓人混淆。應該改成：Thread.sleep(4000); 和 Thread.sleep(2000);。所以只對 main 執行緒有用。

執行緒與並行API

-
- | 11.1 使用並行API
 - | 11.2 使用ExecutorService 介面同時執行多樣工作
 - | 11.3 使用Fork-Join 框架
 - | 11.4 認證考試命題範圍

11.1 使用並行API

11.1.1 並行 API 介紹

並行 API (Concurrency API)，是指在套件「`java.util.concurrent`」下的相關類別和介面。這些 API 在 Java 5 時導入，在 Java 7 做了擴充，用於支援多執行緒執行，亦即 concurrent programming 的領域，包含：

1. 執行緒安全 (thread-safe) 的集合物件。
2. 取代傳統 synchronization 和 locking 的替代方案。
3. 執行緒池 (thread pools)，又分成：
 - 執行緒數量「固定」或「浮動」的執行緒池。
 - 分進合擊的 Fork-Join Framework。

11.1.2 AtomicInteger 類別

類別 `AtomicInteger` 位於套件「`java.util.concurrent.atomic`」下，提供執行緒安全 (thread-safe)，且不用使用 synchronization 和 locking 機制的物件。類別下的方法均為 atomic function，如範例「/OCP/src/course/c11/AtomicIntegerTest.java」：

範例

```

1  public class AtomicIntegerTest {
2      public static void main(String[] args) {
3          AtomicInteger ai = new AtomicInteger(5);
4          if (ai.compareAndSet(5, 42)) {
5              System.out.println("after compareAndSet(): " + ai);
6          }
7          ai.getAndIncrement();
8          System.out.println("after getAndIncrement(): " + ai);
9      }
10 }
```

結果

```

after compareAndSet(): 42
after getAndIncrement(): 43
```

 說明

4	compareAndSet (a,b) : 判斷數值是否為 a，若是設定為 b。亦即將 a 取代為 b。
7	getAndIncrement () : 取得數值後加1。作用和遞增運算子相同，但為 atomic function，不需使用 synchronized 區塊。

11.1.3 ReentrantReadWriteLock 類別

有別於過去的 synchronization 和 monitor 機制，提供另一種鎖定 (locking)，並可以根據不同情況 (conditions) 調整執行緒等待 (wait) 的架構：

- 過去的 monitor 並未分類，一個執行緒取得 monitor 後，其他執行緒必須等待鎖定該 monitor。
- 使用 ReentrantReadWriteLock 類別，將原本由每個物件唯一的 object monitor，改成由 ReentrantReadWriteLock 物件提供 read lock 和 write lock 兩種鎖定機制：
 - 有執行緒先取得 read lock 時，其他執行緒可以再取得 read lock，亦即允許多個執行緒同時 read；但無執行緒可以取得 write lock。
 - 一旦有執行緒取得 write lock 後，將排擠其他執行緒取得 read lock 和 write lock。
- 「Reentrant」指若某個執行緒已經進入某個 synchronized 方法，亦即取得某個物件的 object monitor，則該執行緒可以繼續進入其他使用相同 object monitor 的任一個 synchronized 方法。白話說，就是申請一把鑰匙後，可以繼續打開每個使用相同的鑰匙的門，不用每次開門都繳回再重新排隊申請。

ReentrantReadWriteLock 類別的使用範例詳見「/OCP/src/course/c11/ReadWriteLockTest.java」：

 範例

```

1  class TableData {
2      private final ReentrantReadWriteLock rwl = new ReentrantReadWriteLock();
3      static void sleep(int secs) {
4          try {
5              Thread.sleep(1000 * secs);
6          } catch (Exception e) {}
7      }
8      public void update() {
9          rwl.writeLock().lock();
10         System.out.println("holding write lock");

```

```
11         sleep(3);
12         rwl.writeLock().unlock();
13         System.out.println("released write lock");
14     }
15     public void delete() {
16         rwl.writeLock().lock();
17         System.out.println("holding write lock");
18         sleep(3);
19         rwl.writeLock().unlock();
20         System.out.println("released write lock");
21     }
22     public void query() {
23         rwl.readLock().lock();
24         System.out.println("holding read lock");
25         sleep(3);
26         rwl.readLock().unlock();
27         System.out.println("released read lock");
28     }
29 }
30
31 public class ReadWriteLockTest {
32     static void query(final TableData d) {
33         new Thread() {
34             public void run() {
35                 d.query();
36             }
37         }.start();
38     }
39     static void delete(final TableData d) {
40         new Thread() {
41             public void run() {
42                 d.delete();
43             }
44         }.start();
45     }
46     static void update(final TableData d) {
47         new Thread() {
48             public void run() {
49                 d.update();
50             }
51         }.start();
52     }
53     static void counting() {
54         new Thread() {
```

```
55     public void run() {
56         int i = 0;
57         while (true) {
58             TableData.sleep(1);
59             System.out.println(i++);
60         }
61     }
62     }.start();
63 }
64
65 public static void main(String[] args) {
66     counting();
67     final TableData table = new TableData();
68     query(table);
69     query(table);
70     update(table);
71     delete(table);
72     query(table);
73     query(table);
74     update(table);
75 }
76 }
```

結果

```
holding read lock
holding read lock
0
1
2
released read lock
released read lock
holding write lock
3
4
5
released write lock
holding write lock
6
7
8
released write lock
```

```

holding read lock
holding read lock
9
10
11
released read lock
released read lock
holding write lock
12
13
14
released write lock

```

因為是多執行緒程式，每次結果不一定相同。但可以看出固定的規則是：

1. 允許多執行緒同時「holding read lock」。
2. 同一時間一旦有一個執行緒「holding write lock」，就不會再有其他執行緒「holding read lock」或「holding write lock」。
3. 所以上方結果的左側，若是代表「holding read lock」的框線將會有重疊的情況，而代表「holding write lock」的框線，則不會有重疊的情況出現，象徵其獨佔性質。

11.1.4 執行緒安全的集合物件

套件「java.util」下的集合物件預設「非」執行緒安全；若想要執行緒安全就必須特別處理：

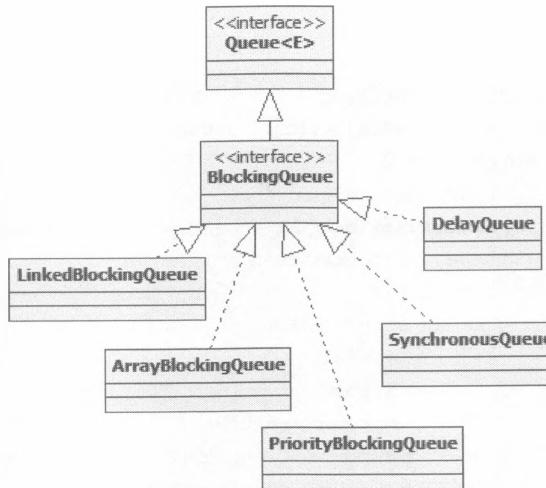
1. 對所有修改集合物件的程式碼，都必須放在 synchronize 區塊。
2. 使用特定類別及方法來建立 synchronized wrapper 類別，如：java.util.Collections.synchronizedList(List<T>)。
3. 改用套件「java.util.concurrent」下的集合物件。但需注意即便是執行緒安全的集合物件，不表示成員也是。

常用的執行緒安全的集合物件如下：

◆表11-1 執行緒安全集合物件與一般集合物件對照表

java.util.concurrent.*	java.util.*
CopyOnWriteArrayList	ArrayList
ConcurrentHashMap	HashMap
ConcurrentSkipListMap	TreeMap

Queue 家族的執行緒安全的集合物件如下：



❖ 圖 11-1 執行緒安全的 Queue 家族

课堂小祕訣

在支援執行緒安全的集合物件裡，常常可以看到「CopyOnWrite」的命名方式。這暗示該集合物件是如何支援執行緒安全：

- 當集合物件內要增加成員時(On Write)，不直接添加，而是先將當前集合物件複製出一個新的集合物件(Copy)，然後在新的集合物件裡添加成員。
- 添加完成員之後，再將原集合物件的物件參考指向新的集合物件。
- 這個做法的好處是集合物件可以讀寫並行，而不需要在修改的時候排除其他行為，因為當前集合物件不會添加任何元素。所以 CopyOnWrite 集合物件是一種讀寫分離的思想實踐，對不同的集合物件讀和寫。

11.1.5 常用的同步器工具類別

套件「java.util.concurrent」下也提供數種支援特殊情境的同步器(synchronizers)類別：

❖ 表 11-2 常見的同步器類別

Class	Description
Semaphore	傳統的 concurrency (平行執行) 工具。
CountDownLatch	暫停 thread 直到某種情境達成。如信號數量、事件、預設條件。
CyclicBarrier (循環路障)	於平行執行時提供同步點，可以循環使用。
Phaser	更有彈性的 barrier。

範例「/OCP/src/course/c11/CyclicBarrierTest.java」以「搭乘貓纜時，人數滿了才發車」的情境示範如何使用 CyclicBarrier 類別：

範例

```

1 public class CyclicBarrierTest {
2     public static void main(String[] args) {
3         int stopUntil = 2;
4         int totalThreadCnt = 3;
5         final CyclicBarrier barrier = new CyclicBarrier(stopUntil);
6         for (int i=1; i<=totalThreadCnt; i++) {
7             new Thread() {
8                 public void run() {
9                     try {
10                         System.out.println("before await");
11                         barrier.await();
12                         System.out.println("after await");
13                     } catch (BrokenBarrierException | 
14                         InterruptedException ex) {
15                         }
16                     }.start();
17                 }
18             }
19         }

```

結果

```

before await
before await
before await
after await
after await

```

說明

3	設定多少個 thread 抵達才放行的條件。
4	可任意調整啟動的 thread 個數，測試 CyclicBarrier 類別的效果。
5	宣告並初始化 CyclicBarrier 類別。
11	await() 方法像是一個柵欄，平時放下；只有滿足 CyclicBarrier 的建構子設定的 stopUntil 個數抵達後，才會放行。以本例而言，一共啟動 3 個執行緒，但只有 2 個 thread 可以通過 await() 方法，因此印出 before await 計 3 次，after await 計 2 次，亦即有 1 個 thread 在等待放行而無法通過。而 JVM 也因此無法終止。

11.2 使用ExecutorService介面同時執行多樣工作

11.2.1 使用更高階的多執行緒執行方案

使用多執行緒的程式架構可以同時執行工作，提升效率；但也容易延伸一些問題，因此必須小心操控。鑑於傳統 API 不容易被適當使用，可以考慮使用以下兩個更高階的替代方案：

1. 執行者服務 (java.util.concurrent.ExecutorService)

- 建立並重複使用多執行緒。
- ExecutorService 介面除了可以使用過去的 Runnable 介面定義工作內容外，也可以使用新的 Callable 介面定義工作內容，此時允許在未來工作結束後，檢視結果。

2. 分進合擊程式框架 (Fork-Join Framework)

- Java 7 推出的特殊「工作竊取 (work-stealing)」平行運算架構，用於多執行緒執行，是一種特化的 ExecutorService。程式運行時，除了不斷將整體工作進行切割之外，也讓有能力、較有餘裕的執行緒在做完份內工作後，可以竊取 (stealing) 別人的工作來執行。是一個支援「能者多勞」理念的多執行緒執行架構。

11.2.2 ExecutorService 概觀

ExecutorService 介面是執行緒池 (thread pool) 的概念，讓使用過的執行緒都可以回收池內繼續下一次使用；執行緒池也負責管理所有執行緒的生命週期。當使用 ExecutorService 來執行多執行緒工作時：

1. 不需自己建立和管理多執行緒，且可以平行執行。
2. 任務可分成二種：
 - java.lang.Runnable
 - java.util.concurrent.Callable
3. 使用 Executors 類別可以取得 ExecutorService 介面的實作，分成二種：
 - 快取式執行緒池 (cached thread pool)。
 - 固定式執行緒池 (fixed thread pool)。

其中，「快取式執行緒池(cached thread pool)」的特點為：

◆表 11-3 快取式執行緒池特點

特徵分類	描述
數量控制	執行緒數量由執行緒池自動調控
是否重複使用	執行緒工作完成後回收重複使用
工作量大時	遇到需要大量CPU運算的工作，執行緒可能會一直增生
生命週期	預設閒置超過60秒就終止生命週期

建立方式為：

```
ExecutorService es = Executors.newCachedThreadPool();
```

「固定式執行緒池(fixed thread pool)」的特點為：

◆表 11-4 固定式執行緒池特點

特徵分類	描述
數量控制	執行緒數量固定
是否重複使用	執行緒工作完成後回收重複使用
工作量大時	工作太多時必須等待忙碌的執行緒釋出
生命週期	因為數量固定，所以不會主動終止生命週期

建立方式為：

```
int cpuCount = Runtime.getRuntime().availableProcessors();
// 參考主機CPU數量建立執行緒數量
ExecutorService es = Executors.newFixedThreadPool(cpuCount);
```

11.2.3 使用 java.util.concurrent.Callable

使用高階 API：ExecutorService 可以協助我們管理執行緒，但我們還是需要自己定義執行緒的工作內容。除了使用傳統的「java.lang.Runnable 介面」之外，也可以使用較新的「java.util.concurrent.Callable 介面」的實作類別來定義要執行的工作。

比較兩者的定義：

- java.lang.Runnable :

```
public interface Runnable {
    void run();
}
```

- `java.util.concurrent.Callable` :

```
public interface Callable<V> {
    V call() throws Exception;
}
```

可以發現 `Callable` 和 `Runnable` 相似，主要不同的地方是：

1. 可以回傳結果(使用泛型)。
2. 可以拋出 `Exception`。

若使用傳統的 `Runnable` 介面的實作物件作為 `ExecutorService` 的執行工作內容，可以用下例執行工作：

```
static void useRunnable(ExecutorService es, Runnable runnable) {
    es.submit(runnable);
}
```

若使用 `Callable` 介面的實作物件作為 `ExecutorService` 的執行工作內容，則改用下例執行工作：

```
static <V> void useCallable(ExecutorService es, Callable<V> callable) {
    Future<V> future = es.submit(callable);
    try {
        V result = future.get();
        System.out.println(result.toString());
    } catch (Exception ex) {
        ex.printStackTrace();
    }
}
```

再比較兩者被執行的方式，可以發現最大的不同就是 `ExecutorService` 的 `submit()` 方法傳入 `Callable` 實作物件的時候可以回傳 `Future` 類別的物件。這個 `Future` 物件就是下一小節的重點。

11.2.4 使用 `java.util.concurrent.Future`

`ExecutorService` 執行 `Callable` 工作後的回傳結果屬於 `Future<V>` 類別：

```
Future<V> future = es.submit(callable);
```

但真正的工作結果是隱藏在 Future 物件裡，所以必須再呼叫 get() 方法去取得結果。這裡的回傳型態 V，必須和 Callable<V> 裡定義的泛型物件型態一致：

```
V result = future.get();
```

會有這樣的要求，主要是因為 main 執行緒在前景 (foreground) 執行工作，而非 main 的執行緒都在背景 (background) 執行工作。要讓在背景的執行緒把工作成果交給在前景的 main 執行緒，並不容易。以下的課堂小祕訣讓您了解差別：

課堂小祕訣

比如說，您到外面的餐館去吃麵。通常可以有「內用」和「外帶」兩種選項。當選擇內用時，您就必須到餐館裡找位置坐下來，靜候老闆把麵煮好。但若外帶，老闆可以給您一個「號碼牌」，接下來您可以去其他地方做自己的事，最後再回來用號碼牌和老闆取麵即可。

這裡的「號碼牌」其實隱含著一種「未來交貨」的意思。假設煮一碗麵要 15 分鐘，取號之後您可以在餐館外等，就是需要 15 分鐘；您也可以選擇到其他地方逛逛，15 分鐘後再回來拿；如果回來的時候離 15 分鐘還早，就是再等到時間到為止。或者是半小時後再來拿，此時拿號碼牌給老闆，馬上就可以拿到自己點的麵。在這個日常生活的經驗裡，

1. 「號碼牌」就是對比到 Java 裡的 Future 物件。
2. 要拿「麵」需要「號碼牌」，所以「麵」可以對比隱藏在 Future 裡的泛型物件，呼叫 Future 的 get() 方法可以取得該泛型物件。
3. 「煮麵」就是 Callable 介面裡 call() 方法定義的內容。
4. 「煮麵的師傅」就是 ExecutorService 裡的執行緒。
5. 「餐館」就是 ExecutorService。
6. 「您」就是 main 執行緒。

「號碼牌」這樣的做法相當常見，買任何東西只要需要時間等待，大部分商家 (ExecutorService) 都會給您一張號碼牌 (Future)，讓您先到其他地方消磨時間，差不多時再回來領取 (get) 購買的真正商品。

所以，呼叫 Future 物件的 get() 方法時若結果尚未出爐，就必須等待結果回傳，此時 main 執行緒就會進入等待的狀態。所以較好的做法是：

1. 在呼叫前 get() 方法前應先丟出 (submit) 所有工作，因為呼叫 get() 方法後只能耐心等候結果。
2. 呼叫 get() 方法前也可以先呼叫 isDone() 方法確認是否工作完成，就跟取貨前先打個電話問一下是否完成，以避免到了商家必須等候一樣。或是呼叫 overloading 的另一版本 get(long timeout, TimeUnit unit) 方法，只等待一段時間。

11.2.5 關閉 ExecutorService

因為 ExecutorService 建立的執行緒都是 non-daemon，所以 JVM 會因為這些執行緒存在而導致永遠不會結束。所以若要結束程式，必須呼叫 ExecutorService 介面的 shutdown() 方法，如下：

範例

```

1 es.shutdown();
2 try {
3     es.awaitTermination(5, TimeUnit.SECONDS);
4 } catch (InterruptedException ex) {
5     System.out.println("Stopped waiting early");
6 }
```

說明

- | | |
|---|--|
| 1 | shutdown() 方法會在所有執行緒的工作都結束後，關閉 ExecutorService 並終止所有執行緒的生命週期，讓程式結束。 |
| 3 | 若已執行 shutdown() 指令，但執行緒工作結束後還是無法關閉，則可以使用 awaitTermination(5, TimeUnit.SECONDS) 方法傳入等待時間，時間到後會強制關閉。 |

11.2.6 ExecutorService 完整範例

本節重點是介紹 ExecutorService 的使用方式，也陸續帶入了 Callable 介面和 Future 類別。以下是完整的應用範例「/OCP/src/course/c11/ExecutorServiceTest.java」：

範例

```

1 class CallableTask implements Callable<String> {
2     @Override
3     public String call() throws Exception {
4         Thread.sleep(20000);
5         System.out.println(new Date() + ": finish job");
6         return (new Date() + ": done");
7     }
8 }
9
10 public class ExecutorServiceTest {
11     public static void main(String[] args) {
12         ExecutorService es = Executors.newCachedThreadPool();
13         //ExecutorService es = Executors.newFixedThreadPool(5);
14 }
```

```

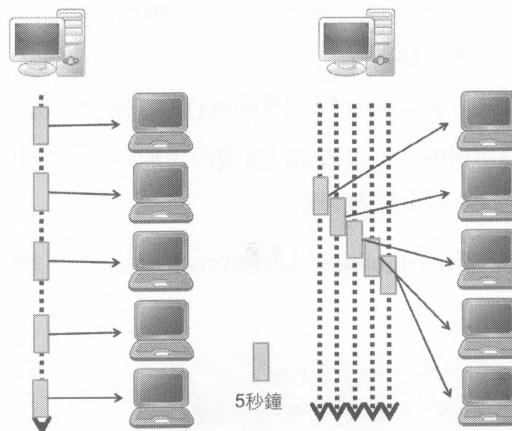
15     Callable<String> task = new CallableTask();
16     Future<String> future = es.submit(task);
17     try {
18         String result = future.get();
19         System.out.println(result);
20     } catch (Exception ex) {
21         ex.printStackTrace();
22     }
23
24     es.shutdown();
25     System.out.println(new Date() + ": service shutdown");
26
27     try {
28         es.awaitTermination(5, TimeUnit.SECONDS);
29     } catch (InterruptedException ex) {
30         System.out.println("Stopped waiting early");
31     }
32 }
33 }
```

說明

1~8	實作 Callable 介面。
12	以 cached thread pool 作為 ExecutorService 的實例。 若執行緒未接受派工，將於 60 秒後結束生命週期。
13	以 fixed thread pool 作為 ExecutorService 的實例。 因為數量固定，不會因為久未接受派工而結束生命週期。 與行 12 需二擇一。
15	建立 Callable 實作物件。
16	讓 ExecutorService 分派執行緒進行 Callable 實作物件定義的工作內容，並取得「號碼牌」Future 物件。
17~22	拿「號碼牌」Future 物件兌換執行緒工作成果。若呼叫 get() 方法時工作尚未結束，就會進入等待狀態。
24	ExecutorService 的 shutdown() 方法會在所有執行緒「工作結束」後，才關閉自己。
28	若已執行 shutdown() 指令，但所有執行緒工作結束後還是無法關閉，則可以使用 awaitTermination(5, TimeUnit.SECONDS) 方法傳入等待時間，時間到後會強制關閉。在本例中，本行程式碼無用武之地。

11.2.7 ExecutorService 進階範例

如以下圖示。若一台主機需要和五台電腦進行程式連線，且每台連線需要 5 秒鐘。左側示意圖採用單一執行緒逐一相連，右側示意圖則採用不同執行緒各與一台電腦相連線：



◆ 圖 11-2 單執行緒與多執行緒執行比較

使用單一執行緒和多執行緒進行工作，和請一個人做一件事，或請五個人同時做一件事所花的時間孰長孰短一樣容易判斷。

範例路徑「/OCP/src/course/c11/testMulti/*」下的類別將以上圖示內容實作；不過需要先對 Java 的 Socket 網路程式架構有基本認識。

Java 的 Socket 程式架構，提供了讓網路世界的二台獨立主機上的 Java 程式互相連線的機制。一般我們區分為：

1. 主機端 (server)：提供網路連線的程式。
2. 用戶端 (client)：要求網路連線的程式。

要成功建立連線需要了解：

1. 主機端必須提供一個固定位置 (包含 IP 和 port)，用戶端要連接就要先知道這位置：

IP 同如網路世界的住址，可以幫助用戶端在網際網路裡找到單一主機，但主機裡可能有諸多程式正在執行，必須有機制來辨認要溝通的程式，port 號像是住宅的門牌編號，用來識別電腦執行的程式。Port 號可以是 0~65535 之間的任一整數，0~1024 的 port 號已經被作業系統保留，因此其他程式通常只能選擇 1024 之後的編號當成自己的程式 port 號。

2. Java 的 Socket 程式主要使用 ServerSocket 和 Socket 兩個類別：

- 主機端用 ServerSocket 物件的 listen() 方法來監聽來自用戶端的連線請求。
- 用戶端用 Socket 物件和主機端做連接。
- 主機端用 ServerSocket 物件的 accept() 方法來串接用戶端的 Socket 物件。

3. 當主機端和用戶端連接之後，就可以使用 Socket 物件的 `getInputStream()` 以及 `getOutputStream()` 方法來傳輸資料。

4. 資料傳輸完成後，記得呼叫 Socket 物件的 `close()` 方法來結束相關資源。

本範例為了執行的簡單與便利，主機端程式或用戶端程式都在本機電腦「localhost」。Port 號則使用 10000~10004 共計五個號碼。

範例「/OCP/src/course/c11/testMulti/SocketServersStartup.java」扮演主機端程式的角色：

範例

```

1  public class SocketServersStartup {
2      public static void main(String[] args) {
3          for (int port = 10000; port < 10005; port++) {
4              new Thread(new SocketServer(port)).start();
5          }
6      }
7  }
8
9  class SocketServer implements Runnable {
10     int port;
11     SocketServer(int port) {
12         this.port = port;
13     }
14     @Override
15     public void run() {
16         System.out.println("Server " + port + ": Listening...");
17         while (true) {
18             try {
19                 // Listen on port
20                 ServerSocket serverSock = new ServerSocket(port);
21                 // Get connection
22                 Socket clientSock = serverSock.accept();
23                 //sleep for 5 seconds
24                 try {
25                     Thread.sleep(5000);
26                 } catch (InterruptedException e) {
27                     e.printStackTrace();
28                 }
29                 // feedback & output
30                 PrintWriter pw =
31                         new PrintWriter(clientSock.getOutputStream(), true);
32                 pw.println("feedback_from_" + port);
33                 // close
34             }
35         }
36     }
37 }
```

```

33         serverSock.close();
34         clientSock.close();
35     } catch (Exception e) {
36         e.printStackTrace();
37     }
38 }
39 }
40 }
```

結果

```

Server 10001: Listening...
Server 10000: Listening...
Server 10002: Listening...
Server 10004: Listening...
Server 10003: Listening...
```

執行本程式時，會啓動五個執行緒，分別使用 port 號 10000~10004，扮演主機端的角色，並等待 / 傾聽 (listening) 用戶端連線的要求。

若用戶端連線成功，主機端在停頓五秒鐘後將輸出「feedback_from_1000X」的訊息給用戶端。

範例「/OCP/src/course/c11/testMulti/SocketBean.java」：

範例

```

1  public class SocketBean {
2      public String host;          // 主機端 IP
3      public int port;            // 主機端 port 號
4      public String feedback;    // 主機端回覆的訊息
5
6      public SocketBean(String host, int port) {
7          this.host = host;
8          this.port = port;
9      }
10 }
```

範例「/OCP/src/course/c11/testMulti/SingleThreadTest.java」：

範例

```

1  public class SingleThreadTest {
2      public static void main(String[] args) {
3          out.println("SingleThread starts at: " + new Date());
```

```

4     String host = "localhost";
5     for (int port = 10000; port < 10005; port++) {
6         SocketBean bean = new SocketBean(host, port);
7         try (Socket sock = new Socket(bean.host, bean.port);
8             Scanner scanner = new Scanner(sock.getInputStream());) {
9             bean.feedback = scanner.next();
10            out.println("Call " + bean.host + ":" + bean.port +
11                ", and get: " + bean.feedback + " at " + new Date());
12        } catch (NoSuchElementException | IOException ex) {
13            out.println("Error talking to " + host + ":" + port);
14        }
15    }
16 }

```

結果

```

SingleThread starts at: Thu Jun 02 09:20:11 CST 2016
Call localhost:10000,
    and get: feedback_from_10000 at Thu Jun 02 09:20:16 CST 2016
Call localhost:10001,
    and get: feedback_from_10001 at Thu Jun 02 09:20:21 CST 2016
Call localhost:10002,
    and get: feedback_from_10002 at Thu Jun 02 09:20:26 CST 2016
Call localhost:10003,
    and get: feedback_from_10003 at Thu Jun 02 09:20:31 CST 2016
Call localhost:10004,
    and get: feedback_from_10004 at Thu Jun 02 09:20:36 CST 2016

```

- 使用「單一執行緒」逐 port 連線主機的結果，必須使用「5(秒) \times 5 = 25(秒)」的時間。

接下來的範例將使用多執行緒架構連線，因此需要使用 Callable 介面定義執行緒的工作內容。詳見範例「/OCP/src/course/c11/testMulti/SocketClientCallable.java」：

範例

```

1  public class SocketClientCallable implements Callable<SocketBean> {
2      private SocketBean bean;
3      public SocketClientCallable(SocketBean bean) {
4          this.bean = bean;
5      }
6      @Override
7      public SocketBean call() throws IOException {
8          try (Socket sock = new Socket(bean.host, bean.port);
9              Scanner scanner = new Scanner(sock.getInputStream());) {

```

```

10         bean.feedback = scanner.next();
11     return bean;
12   }
13 }
14 }
```

範例「/OCP/src/course/c11/testMulti/MultiThreadTest.java」：

範例

```

1  public class MultiThreadTest {
2      public static void main(String[] args) {
3          System.out.println("MultiThreadTest starts at: " + new Date());
4          ExecutorService es = Executors.newCachedThreadPool();
5          String host = "localhost";
6
7          Map<SocketBean, Future<SocketBean>> callables = new HashMap<>();
8          //依不同port號送出工作
9          for (int port = 10000; port < 10005; port++) {
10              SocketBean bean = new SocketBean(host, port);
11              SocketClientCallable callable = new SocketClientCallable(bean);
12              Future<SocketBean> future = es.submit(callable);
13              callables.put(bean, future);
14          }
15
16          //關閉ExecutorService
17          es.shutdown();
18          try {
19              es.awaitTermination(5, TimeUnit.SECONDS);
20          } catch (InterruptedException ex) {
21              System.out.println("Stopped waiting early");
22          }
23
24          //取回結果
25          for (SocketBean bean : callables.keySet()) {
26              Future<SocketBean> future = callables.get(bean);
27              try {
28                  bean = future.get();
29                  System.out.println("Call " + bean.host + ":" + bean.port +
30                                     ", and get: " + bean.feedback + " at " + new Date());
31              } catch (ExecutionException | InterruptedException ex) {
32                  System.out.println("Error talking to " +
33                                     bean.host + ":" + bean.port);
34              }
35          }
36      }
37  }
```

```

33     }
34     }
35 }
```

說明

7	定義 Map<SocketBean, Future<SocketBean>> callables 存放 Callable 的執行結果，以 SocketBean 作為 key 物件，Future<SocketBean> 為 value 物件。
9~14	使用迴圈讓 ExecutorService 可以依據 port 號送出 Callable 工作給多執行緒去執行。
12	定義 future 物件承接執行結果，但先不呼叫 get() 方法。
13	將 future 物件先存在 Map 裡，並以 SocketBean 物件作為 key。
25~33	在 map 物件中找出所有 future 物件，並呼叫 get() 取回結果。

結果

```

MultiThreadTest starts at: Thu Jun 02 09:21:55 CST 2016
Call localhost:10004,
    and get: feedback_from_10004 at Thu Jun 02 09:22:00 CST 2016
Call localhost:10000,
    and get: feedback_from_10000 at Thu Jun 02 09:22:00 CST 2016
Call localhost:10002,
    and get: feedback_from_10002 at Thu Jun 02 09:22:00 CST 2016
Call localhost:10003,
    and get: feedback_from_10003 at Thu Jun 02 09:22:00 CST 2016
Call localhost:10001,
    and get: feedback_from_10001 at Thu Jun 02 09:22:00 CST 2016
```

雖然每個執行緒還是要花五秒鐘，但因為是同時執行，程式完成也只花五秒鐘時間，比單一執行緒完成的總時間快許多。

11.3 使用Fork-Join框架

11.3.1 平行處理的策略

現在的電腦多含數個CPU。為了有效運用運算效能，可以讓程式同時執行工作，「平行處理 (Parallelism)」的策略很重要：

1. 將工作分切成小段，各自完成後工作就可以解決，稱為「Divide and conquer」處理策略。但使用前需確認這些小段工作可以平行處理。

2. 分割時注意硬體效能問題，切割太細可能有反效果。若工作內容需要大量 CPU 計算而非 I/O 存取，需考慮 CPU 數量：

```
int count = Runtime.getRuntime().availableProcessors();
```

切割資料讓多執行緒可以平行執行，是平行處理策略的第一步。但資料該如何切割？最理想的情況是讓所有 CPU 可以充分被所有執行緒利用，直到工作結束。

若是採取「平均分配」的做法，讓每個執行緒各自占用不同的 CPU 處理相同份量的工作，容易因為：

1. 每個 CPU 可能效率不同。
2. 某些 CPU 可能也在執行其他程式。

而導致無法發揮硬體最高性能。就好比一塊空地平均切割為十等份請十位同學打掃。乍看之下最有利，但每個同學掃地效率不同，不會同時完成。

另一種做法稱為「工作竊取 (work-stealing)」平行運算架構。一樣將工作平均切割，但數量遠多於執行緒個數：

1. 工作不會馬上完成，所以每個執行緒會有很多待辦工作在自己的佇列上。
2. 如果某個執行緒已經做完自己的工作，可以竊取 (steal) 別人的工作來處理。
3. 合適的切割分量不易達成。切割太多時，切割本身就是一個負擔；切割太少時，將無法充分利用 CPU 資源。

如此做法可以讓每個執行緒都很忙，因此可以發揮硬體最高性能。這好比將一塊空地分成一百塊請十位同學打掃；掃完自己區域的同學，可以再去打掃其他區域，達到「能者多勞」的理念。

Java 7 推出 Fork-Join 的框架，是實踐「工作竊取 (work-stealing)」的平行運算架構。

11.3.2 套用 Fork-Join 框架

使用單一執行緒處理

要在 1G 的 int 陣列裡找出最大的數字，可以使用範例程式「/OCP/src/course/c11/forkJoin/SingleThreadTest.java」的做法：

範例

```

1 public class SingleThreadTest {
2     public static void main(String[] args) {
3         int[] bigData = new int[1024 * 1024 * 256]; // 1G
4         for (int i = 0; i < bigData.length; i++) {
5             bigData[i] = ThreadLocalRandom.current().nextInt();
6         }
7         int max = Integer.MIN_VALUE;
8         for (int value : bigData) {
9             if (value > max) {
10                 max = value;
11             }
12         }
13         System.out.println("Found max value:" + max);
14     }
15 }
```

使用 Fork-Join 框架平行處理

先前我們的多執行緒架構使用 Executors 取得合適的 ExecutorService 實作物件，通常是 cached thread pool 或是 fixed thread pool，來執行「Callable 介面」定義的工作內容。

若要使用 Fork-Join 框架，可以直接使用 ExecutorService 特化的子類別「java.util.concurrent.ForkJoinPool」來執行「java.util.concurrent.ForkJoinTask<V> 抽象類別」定義的工作內容：

1. ForkJoinTask 物件代表需要執行的工作，中文可以解釋為「分進合擊」，亦即一開始大家分頭進行，逐漸會師合併成果。
2. ForkJoinTask 物件包含要處理的資料和處理方式，和 Runnable 及 Callable 相似。
3. 大量的工作可以由 Fork-Join pool 內少數執行緒處理，因此每個執行緒會工作滿檔，所以可以發揮硬體效能極致。
4. 開發者通常繼承 ForkJoinTask 的子類別，再實作 compute() 方法：
 - RecursiveAction：compute() 方法沒有回傳結果。

```

public abstract class RecursiveAction extends ForkJoinTask<Void> {
    protected abstract void compute();
}
```

- RecursiveTask : compute() 方法有回傳結果。

```
public abstract class RecursiveTask<V> extends ForkJoinTask<V> {
    protected abstract V compute();
}
```

其中 compute() 方法的實作方式顯示了分進合擊的主要精神，假設需要回傳 RESULT，架構與概念如下：

範例

```
1  protected RESULT compute () {
2      if (DATA_SMALL_ENOUGH) {
3          PROCESS_DATA
4          return RESULT;
5      } else {
6          SPLIT_DATA_INTO_LEFT_AND_RIGHT_PARTS
7          TASK t1 = new TASK (LEFT_DATA);
8          t1.fork();
9          TASK t2 = new TASK (RIGHT_DATA);
10         return COMBINE ( t2.compute(), t1.join() );
11     }
12 }
```

說明

2	DATA_SMALL_ENOUGH：工作量夠少時。
3	PROCESS_DATA：不再切割工作並開始處理資料，可以得到結果 RESULT。
4	回傳處理結果 RESULT。
5	若資料處理量還是太大，仍需再切割。
6	SPLIT_DATA_INTO_LEFT_AND_RIGHT_PARTS：將工作量切割為左、右兩部分。
7	建立一個稱為 t1 的 ForkJoinTask 處理左半部工作。
8	呼叫 t1.fork() 做非同步處理（分進、分頭行事）。
9	建立另一個稱為 t2 的 ForkJoinTask 處理右半部工作。
10	<ol style="list-style-type: none"> 1. 呼叫 t1.join()：可以等到 t1.fork() 結束後，合併得到結果（合擊）。 2. 呼叫 t2.compute()：t2 使用遞迴結構持續進行分進合擊策略。若工作量夠少時，已經不須再切割，就會進行如第 3 行的計算，並得到結果。 3. COMBINE：若 t1, t2 均有結果，則合併結果。

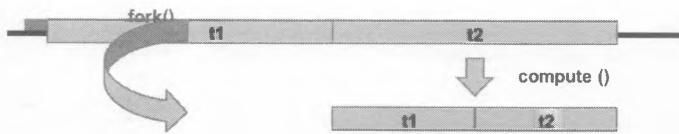
使用圖解方式更容易了解分進合擊的過程：

STEP01 原始資料處理量太大，將其分成兩部分，分別派給 t1、t2 兩個 ForkJoinTask：



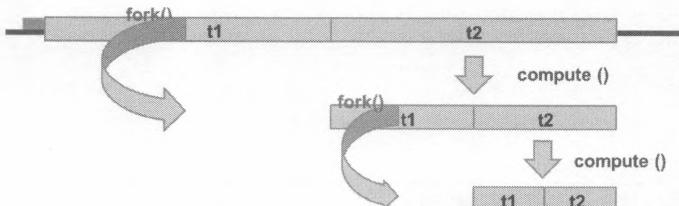
◆ 圖 11-3 分進合擊圖解步驟一

STEP02 呼叫 t1.fork() 方法時將調用 1 個執行緒去處理資料，此為「分進」。呼叫 join() 時將取回結果，即為「合擊」。目前累計 1 個計算結果未取回。呼叫 t2.compute() 時若資料處理量依然太大，持續進行切割工作：



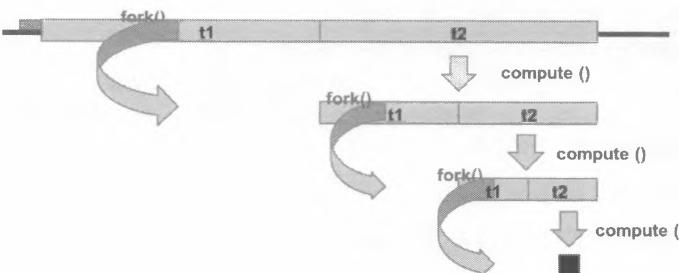
◆ 圖 11-4 分進合擊圖解步驟二

STEP03 同前一步驟。目前累計二個計算結果未取回：



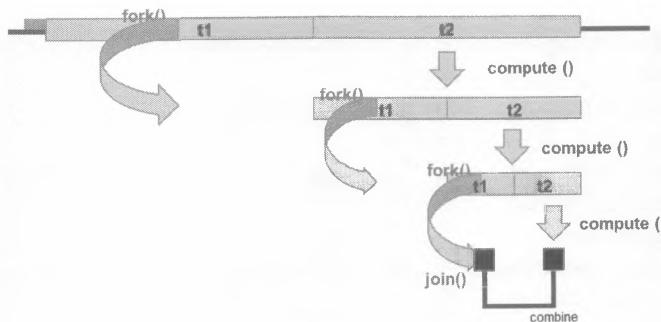
◆ 圖 11-5 分進合擊圖解步驟三

STEP04 同前一步驟。每次切割出去的左半部資料的處理結果已經累計三個未取回；此時右側資料已經夠少，呼叫 t2.compute() 不再切割，將直接計算並得到一個結果。



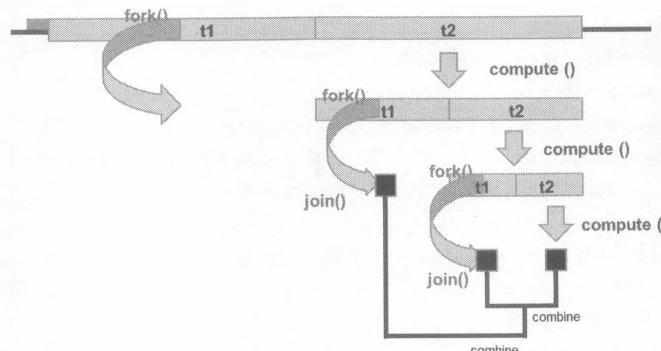
◆ 圖 11-6 分進合擊圖解步驟四

STEP05 開始將處理結果進行合併。要取得左半部的資料處理結果需要呼叫 `t1.join()` 方法，再呼叫 `combine()` 方法合併結果：



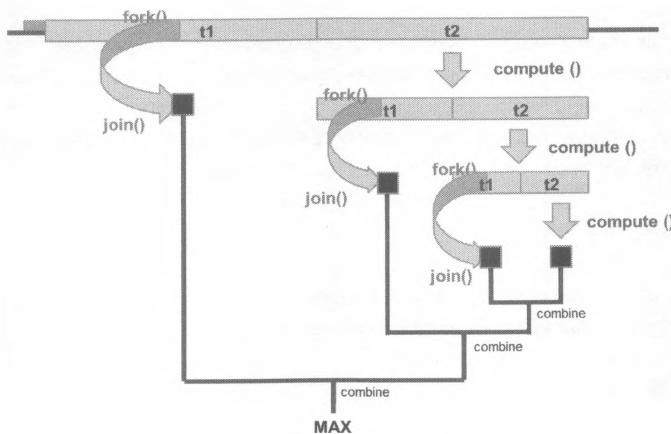
❖ 圖 11-7 分進合擊圖解步驟五

STEP06 承上步驟，持續取得 (`join()`) 和合併 (`combine()`)：



❖ 圖 11-8 分進合擊圖解步驟六

STEP07 完全合併後，即得結果：



❖ 圖 11-9 分進合擊圖解步驟七

完整範例為「/OCP/src/course/c11/forkJoin/ForkJoinMultiThreadTest.java」：

範例

```

1  class FindMaxTask extends RecursiveTask<Integer> {
2      static int counter = 0;
3      private static final long serialVersionUID = 1L;
4      private final int threshold;
5      private final int[] data;
6      private int begin;
7      private int end;
8      public FindMaxTask(int[] data, int begin, int end, int threshold) {
9          this.data = data;
10         this.begin = begin;
11         this.end = end;
12         this.threshold = threshold;
13     }
14     @Override
15     protected Integer compute() {
16         if (end - begin < threshold) {
17             System.out.printf("%02d", ++counter);
18             System.out.print(": " + Thread.currentThread().getName());
19             System.out.println(" | " + begin + " ~ " + end );
20             int max = Integer.MIN_VALUE;
21             for (int i = begin; i <= end; i++) {
22                 int n = data[i];
23                 if (n > max) {
24                     max = n;
25                 }
26             }
27             return max;
28         } else {
29             int mid = (end - begin) / 2 + begin;
30             FindMaxTask a1 = new FindMaxTask(data, begin, mid, threshold);
31             a1.fork();
32             FindMaxTask a2 = new FindMaxTask(data, mid + 1, end, threshold);
33             return Math.max(a2.compute(), a1.join());
34         }
35     }
36 }
37 public class ForkJoinMultiThreadTest {
38     public static void main(String[] args) {
39         Date begin = new Date();
40         // 製作資料

```

```

41     int[] bigData = new int[1024 * 1024 * 256]; // 1G
42     for (int i = 0; i < bigData.length; i++) {
43         bigData[i] = ThreadLocalRandom.current().nextInt();
44     }
45     // 使用 fork-join 框架
46     FindMaxTask task =
47         new FindMaxTask(bigData,
48                         0,
49                         bigData.length - 1,
50                         bigData.length / 16);
51     ForkJoinPool pool = new ForkJoinPool();
52     Integer max = pool.invoke(task);
53     System.out.println("\nMax value found:" + max);
54     // 計時
55     long t = new Date().getTime() - begin.getTime();
56     System.out.println("Complete task within " + t + " milliseconds");
57 }

```

說明

17~19	顯示目前是第 N 個進行的運算，由哪個執行緒執行。因為本範例切割為 16 等分，所以 N<=16。
20~26	計算資料陣列裡的最大值。
29~33	不斷切割資料到門檻值。
41~44	製作資料陣列。
46	建立 Fork-Join 框架預備進行的工作物件：FindMaxTask。且資料陣列將會切割為 16 等分才進行運算。
47	建立 ForkJoinPool。
48	使用 ForkJoinPool 類別的 invoke() 方法傳入要執行的工作，該方法會呼叫 FindMaxTask 的 compute() 方法。

結果

```

01: ForkJoinPool-1-worker-1 |Range: 251658240 ~ 268435455
02: ForkJoinPool-1-worker-3 |Range: 184549376 ~ 201326591
03: ForkJoinPool-1-worker-2 |Range: 117440512 ~ 134217727
05: ForkJoinPool-1-worker-1 |Range: 234881024 ~ 251658239
06: ForkJoinPool-1-worker-3 |Range: 167772160 ~ 184549375
07: ForkJoinPool-1-worker-1 |Range: 218103808 ~ 234881023
08: ForkJoinPool-1-worker-1 |Range: 201326592 ~ 218103807
09: ForkJoinPool-1-worker-3 |Range: 134217728 ~ 150994943
04: ForkJoinPool-1-worker-4 |Range: 150994944 ~ 167772159
10: ForkJoinPool-1-worker-4 |Range: 50331648 ~ 67108863

```

```
11: ForkJoinPool-1-worker-2 |Range: 100663296 ~ 117440511
13: ForkJoinPool-1-worker-4 |Range: 33554432 ~ 50331647
12: ForkJoinPool-1-worker-3 |Range: 16777216 ~ 33554431
14: ForkJoinPool-1-worker-2 |Range: 83886080 ~ 100663295
15: ForkJoinPool-1-worker-4 |Range: 0 ~ 16777215
16: ForkJoinPool-1-worker-1 |Range: 67108864 ~ 83886079

Max value found:2147483647
Complete task within 1527 milliseconds
```

由執行結果分析，可以理解：

1. 資料陣列平均切割為十六等分後才進行運算。
2. Java 一共使用四個執行緒處理十六份工作。

因為數字比較大的關係，可能較不易看出；可以將行 1G 的長度改為 32，結果就會很清楚。

11.3.3 Fork-Join 框架的使用建議

使用 Fork-Join 框架有幾個需要知道並注意的地方：

1. 預設每核 CPU 會建立一個對應的執行緒執行工作。
2. 使用時應先排除 I/O 或是其他可能卡住執行緒工作的情況發生。
3. 了解自己的硬體：
 - 單個 CPU 時，使用 Fork-Join 框架反而會比較慢。
 - 有些 CPU 只使用單核時，會比使用多核快。因此讓使用 Fork-Join 框架的成效感覺更少些。
4. 相較於單一執行緒的循序執行，平行執行會有先切割工作的額外負擔，延長執行時間。

11.4 認證考試命題範圍

依據甲骨文公布的考試範圍（https://education.oracle.com/java-se-8-programmer-ii/pexam_1Z0-809），和本章相關的主題有：

Java Concurrency

1. Create worker threads using **Runnable**, **Callable** and use an **ExecutorService** to concurrently execute tasks.
2. Use synchronized keyword and **java.util.concurrent.atomic** package to control the order of thread execution.
3. Use **java.util.concurrent** collections and classes including **CyclicBarrier** and **CopyOnWriteArrayList**.

本章擬真試題實戰

考題 1

Which two are differences between Callable and Runnable?

- A. A Callable can return a value when executing, but a Runnable cannot.
- B. A Callable can be executed by a ExecutorService, but a Runnable cannot.
- C. A Callable can be passed to a Thread, but a Runnable cannot.
- D. A Callable can throw an Exception when executing, but a Runnable cannot.

答案 AD

說明 兩者差別在於 A. 是否回傳結果。D. 是否拋出例外。

考題 2

Given:

```
interface IdProducer {  
    int getNextId();  
}
```

Which class implements IdProducer in a thread-safe manner, so that no threads can get a duplicate id value even with concurrent access?

A.

```
class ProducerA implements IdProducer {  
    private AtomicInteger id = new AtomicInteger(0);  
    public int getNextId() {  
        return id.getAndIncrement();  
    }  
}
```

B.

```
class ProducerB implements IdProducer {
```

```

private int id = 0;
public int getNextId() {
    return ++id;
}
}

```

C.

```

class ProducerC implements IdProducer {
    private volatile int Id = 0;
    public int getNextId() {
        return ++Id;
    }
}

```

D.

```

class ProducerD implements IdProducer {
    private int id = 0;
    public int getNextId() {
        synchronized (new ProducerD()) {
            return ++id;
        }
    }
}

```

E.

```

class ProducerE implements IdProducer {
    private int id = 0;
    public int getNextId() {
        synchronized (id) {
            return ++id;
        }
    }
}

```

答案 A**說明**

選項 B : `++id` 不是 atomic function。

選項 C : 同選項 B，加上 `volatile` 壓告也無濟於事。

選項 D : 應改為 `synchronized (this)`，使用 `new ProducerD()` 會導致每次都使用新的 object monitor，失去 lock 的意義。

選項 E : 同選項 D，應改為 `synchronized (this)`。`synchronized(…)` 只能傳入 object，因為需要該 object 的 monitor。

考題 3

Given:

```
class ConcurrentModificationTest {  
    static CopyOnWriteArrayList<String> arr = new CopyOnWriteArrayList<>();  
    static void test() {  
        String var = "";  
        Iterator<String> e = arr.iterator();  
        while (e.hasNext()) {  
            var = e.next();  
            if (var.equals("A")) {  
                arr.remove(var);  
            }  
        }  
    }  
    public static void main(String[] args) {  
        ArrayList<String> list1 = new ArrayList<>();  
        list1.add("A");  
        list1.add("B");  
        ArrayList<String> list2 = new ArrayList<>();  
        list1.add("A");  
        list1.add("D");  
        arr.addAll(list1);  
        arr.addAll(list2);  
        test();  
        for (String var : arr) {  
            System.out.print(var + " ");  
        }  
    }  
}
```

What is the result?

- A. Null B D
- B. Null B null D
- C. B D
- D. D
- E. An exception is thrown at runtime

答案 C

說明 因為使用 CopyOnWrite 的容器，可以讀寫分離，因此不會拋出「ConcurrentModificationException」。請參照本書「11.1.4 執行緒安全的集合物件」。

考題 4

Given:

```
ConcurrentMap<String, String> cMap = new ConcurrentHashMap<>();
String key = "someKey";
```

Which fragment puts a key/value pair in cMap without the responsibility of overwriting an existing key?

- A. cMap.put(key, "Blue Shirt");
- B. cMap.putIfAbsent(key, "Blue Shirt");
- C. cMap.putIfNotLocked(key, "Blue Shirt");
- D. cMap.putAtomic(key, "Blue Shirt");

答案 B

說明 ConcurrentHashMap.putIfAbsent(key, value) 只有在沒有 key 存在的情況下會放入 value。

考題 5

Given:

```
class FibonacciTask extends RecursiveTask<Integer> {
    final int i;
    FibonacciTask(int n) {
        this.i = n;
    }
    protected Integer compute() {
        if (i < 1) {
            return i;
        } else {
            FibonacciTask f1 = new FibonacciTask(i - 1);
            f1.fork();
            FibonacciTask f2 = new FibonacciTask(i - 2);
            return f2.compute() + f1.join(); // Line X
        }
    }
}
```

And:

```
public static void main(String[] args) {  
    FibonacciTask task = new FibonacciTask(11);  
    ForkJoinPool pool = new ForkJoinPool();  
    int r = pool.invoke(task);  
    System.out.println("result = " + r);  
}
```

What is the likely result?

- A. The program produces the correct result, with similar performance to the original.
- B. The program produces the correct result, with performance degraded to the equivalent of being single threaded.
- C. The program produces an incorrect result.
- D. The program goes into an infinite loop.
- E. An exception is thrown at runtime.
- F. The program produces the correct result, with better performance than the original.

答案 F

說明 本題使用 Fork/Join 框架執行費氏 (Fibonacci) 數列。數列的特徵是每個數字為前二個數字的加總，如：「1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, …」，可參考原廠文件：
<https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/RecursiveTask.html>

考題 6

Given:

```
class Test {  
    private AtomicInteger i = new AtomicInteger(0);  
    public void increment() {  
        // insert code here  
    }  
}
```

Which line of code, inserted inside the increment () method, will increment the value of i?

- A. i.addAndGet();
- B. i++;
- C. i = i + 1;
- D. i.getAndIncrement();

答案 D

說明 詳見本書「11.1.2 AtomicInteger 類別」。

考題 7

How many threads are created when passing task to an Executor instance?

- A. A new thread is used for each task.
- B. A number of threads equal to the number of CPUs Is used to execute tasks.
- C. A single thread is used to execute all tasks.
- D. A developer-defined number of threads is used to execute tasks.
- E. A number of threads determined by system load is used to execute tasks.
- F. The method used to obtain the Executor determines how many threads are used to execute tasks.

答案 F

說明 介面 Executor 的定義為：

```
public interface Executor {
    void execute(Runnable command);
}
```

介面 ExecutorService 繼承了介面 Executor：

```
public interface ExecutorService extends Executor {
    //....abstract methods
}
```

藉由不同的 Executors 類別的 static 方法，可以取得不同的 ExecutorService，也決定了啓動 threads 的數量。如：

```
ExecutorService s1 = Executors.newCachedThreadPool()
ExecutorService s2 = Executors.newFixedThreadPool(int qty)
ExecutorService s3 = Executors.newSingleThreadExecutor()
...
```

考題 8

Given:

```
class CowThread extends Thread {  
    static List<Integer> cowList = new CopyOnWriteArrayList<>();  
    public void run() {  
        try {  
            Thread.sleep(500);  
        } catch (Exception e) {  
            System.out.print("e2");  
        }  
        cowList.add(88);  
        System.out.print("size:" + cowList.size() + ",elements:");  
    }  
}
```

And:

```
public static void main(String[] args) {  
    cowList.add(11);  
    cowList.add(22);  
    cowList.add(33);  
    cowList.add(44);  
    new CowThread().start();  
    for (Integer i : cowList) {  
        try {  
            Thread.sleep(1000);  
        } catch (Exception e) {  
            e.printStackTrace();  
        }  
        System.out.print(" " + i);  
    }  
}
```

What is the most likely result?

- A. size: 4, elements: 11 22 33 44
- B. size: 5, elements: 11 22 33 44
- C. size: 4, elements: 11 22 33 44 88
- D. size: 5, elements: 11 22 33 44 88
- E. a ConcurrentModificationException is thrown

答案 B

說明

1. 因為使用 CopyOnWrite 的容器，可以讀寫分離，所以二個執行緒看到的內容不同。執行緒 main 已經開始讀取容器內容，此時執行緒 CowThread 想寫入容器，只能複製一份出來再寫入；等執行緒 main 完成讀取之後，Java 會將新的容器物件指回原物件參考。
2. 不支援 Concurrency 的容器，才會拋出 ConcurrentModificationException。

考題 9

Given the incomplete pseudo-code for a fork/join framework application:

```
submit(Data data) {
    if (data.size < SMALL_ENOUGH) {
        _____(data); //line1
    } else {
        List<Data> x = _____(data); //line2
        for (Data d: x) {
            _____(d); //line3
        }
    }
}
```

And given the missing methods: "process", "submit", and "splitInHalf".

Which three insertions properly complete the pseudo-code?

- A. Insert submit at line1.
- B. Insert splitInHalf at line1.
- C. Insert process at line1.
- D. Insert process at line2.
- E. Insert splitInHalf at line2.
- F. Insert process at line3.
- G. Insert submit at line3.

答案 CEG

說明 fork/join framework 的基本精神是：

1. 資料夠少才進行處理：process(data)。
2. 資料還是太大就進行分割：splitInHalf(data)，再進行遞迴流程 submit(data)。

考題 10

Which method would you supply to a class implementing the Callable interface?

- A. callable ()
- B. executable ()
- C. call ()
- D. run ()
- E. start ()

答案 C

考題 11

Which statement creates a low overhead, low-contention random number generator that is isolated to thread to generate a random number between 1 and 100?

- A. int a = ThreadLocalRandom.current().nextInt(1, 101);
- B. int b = ThreadSafeRandom.current().nextInt(1, 101);
- C. int c = (int) Math.random()*100+1;
- D. int d = (int) Math.random(1, 101);
- E. int e = new Random().nextInt(100)+1;

答案 A

說明

選項 B：不存在類別 ThreadSafeRandom。

選項 C、D：要滿足 low overhead 且 low-contention，必須選擇 ThreadLocalRandom。

選項 E：根據 API 文件 <https://docs.oracle.com/javase/7/docs/api/java/util/Random.html> 的說明：
Instances of java.util.Random are threadsafe. However, the concurrent use of the same
java.util.Random instance across threads may encounter contention and consequent poor
performance. Consider instead using ThreadLocalRandom in multithreaded designs.

可知類別 Random 是執行緒安全，但並非 low overhead 且 low-contention。

考題 12

Given:

```
public static void main(String[] args) {
    AtomicInteger[] atomicInts = new AtomicInteger[6];
    for (int i = 0; i < atomicInts.length; i++) {
        atomicInts[i] = new AtomicInteger();
    }
    for (int i = 0; i < atomicInts.length; i++) {
        atomicInts[i].incrementAndGet();
        if (i == 2) {
            atomicInts[i].compareAndSet(2, 4);
        }
        System.out.print(atomicInts[i] + " ");
    }
}
```

What is the result?

- A. 1 1 1 1 1 1
- B. 1 2 3 4 5 6
- C. 0 1 2 3 4 5
- D. 0 1 4 3 4 5

答案 A

說明

1. AtomicInteger 的初始值是 0。
2. 呼叫 incrementAndGet() 後，atomicInts 的成員會全部變成 1。
3. 呼叫 compareAndSet(2, 4)，只有值是 2 的才會被取代為 4。

考題 13

Which type of ExecutorService supports the execution of tasks after a fixed delay?

- A. DelayedExecutorService
- B. ScheduledExecutorService
- C. TimedExecutorService
- D. FixedExecutorService

E. FutureExecutorService

答案 B

說明 選項B：根據 API 文件 <https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/ScheduledExecutorService.html> 的說明：An ExecutorService that can schedule commands to run after a given delay, or to execute periodically。其他選項的類別均不存在。

考題 14

Given:

```
class Caller implements Callable<String> {
    String s;
    public Caller(String s) {
        this.s = s;
    }
    public String call() throws Exception {
        return s + " Caller";
    }
}
class Runner implements Runnable {
    String s;
    public Runner(String s) {
        this.s = s;
    }
    public void run() {
        System.out.println(s + " Runner");
    }
}
```

And:

```
public static void main(String[] args) throws Exception {
    ExecutorService es = Executors.newFixedThreadPool(2);
    Future<String> callFu = es.submit(new Caller("Call"));
    Future runFu = es.submit(new Runner("Run"));
    String strCall = (String) callFu.get();
    String strRun = (String) runFu.get(); // line n1
    System.out.println(strCall + "-" + strRun);
}
```

What is the result?

A. The program prints:

Run Runner

Call Caller-null

And the program does not terminate.

B. The program prints:

Run Runner

Call Caller-null

And the program terminates.

C. Compilation occurs at line 1.

D. Execution is thrown at runtime.

答案 A

說明

1. 使用 Executors.newFixedThreadPool()，不會因為 idle，而關閉 thread。
2. run() 方法沒有回傳，故取得 null。

考題 15

Given:

```
class CallableThread implements Callable<String> {
    String s;
    public CallableThread(String s) {
        this.s = s;
    }
    public String call() throws Exception {
        return s + " Call";
    }
}
```

And:

```
public static void main(String[] args) throws Exception {
    ExecutorService es = Executors.newFixedThreadPool(2); // line n1
    Future<String> f1 = es.submit(new CallableThread("Call"));
    System.out.println(f1.get());
}
```

Which statement is true?

- A. The program prints Call Call and terminates.
- B. The program prints Call Call and does not terminate.
- C. A compilation error occurs at line n1.
- D. An ExecutionException is thrown at run time.

答案 B

說明 使用 Executors.newFixedThreadPool()，不會因為 idle，而關閉 thread。

考題 16

Given:

```
class GetThread implements Runnable {  
    private static AtomicInteger counter = new AtomicInteger(0);  
    public void run() {  
        int c = counter.incrementAndGet();  
        System.out.print(c + " ");  
    }  
}
```

And:

```
public static void main(String[] args) {  
    Thread t1 = new Thread(new GetThread());  
    Thread t2 = new Thread(new GetThread());  
    Thread t3 = new Thread(new GetThread());  
    Thread[] threads = { t1, t2, t3 };  
    for (Thread t : threads) {  
        t.start();  
    }  
}
```

Which statement is true?

- A. The program prints 1 2 3 and the order would be changed.
- B. The program prints 1 2 3.
- C. The program prints 1 1 1.
- D. Compilation error.

答案 A

考題 17

Given:

```
class Worker extends Thread {
    CyclicBarrier cb;
    public Worker(CyclicBarrier cb) {
        this.cb = cb;
    }
    public void run() {
        try {
            cb.await();
            System.out.println("Worker...");
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }
}
class BarrierAction implements Runnable {           // line1
    public void run() {
        System.out.println("Master...");
    }
}
```

And:

```
public static void main(String[] args) {
    BarrierAction action = new BarrierAction();
    // line2
    Worker w = new Worker(cb);
    w.start();
}
```

Which modification could ensure that the run methods of both the Worker and BarrierAction are executed?

- A. At line 2, insert `CyclicBarrier cb = new CyclicBarrier(2, action);`
- B. Replace line 1 with `class BarrierAction extends Thread {`
- C. At line 2, insert `CyclicBarrier cb = new CyclicBarrier(1, action);`
- D. At line 2, insert `CyclicBarrier cb = new CyclicBarrier(action);`

答案 C

說明

1. 將同一個 CyclicBarrier barrier 物件分別傳入需要路障的執行緒中。這些執行緒的 run() 方法必須呼叫 barrier.await() 方法，才能發揮路障效果。
2. CyclicBarrier 有二種建構子：
 - CyclicBarrier(int parties)：決定多少個 thread 可放行。
 - CyclicBarrier(int parties, Runnable barrierAction)：決定多少個 thread 可放行，和遇到路障時該做什麼事。

考題 18

Given:

```

class SumAction extends RecursiveAction {      // line1
    static final int DATA_MAX_LENGTH = 3;
    int startIdx, endIdx;
    int[] data;
    public SumAction(int[] data, int startIdx, int endIdx) {
        this.data = data;
        this.startIdx = startIdx;
        this.endIdx = endIdx;
    }
    protected void compute() {
        int sum = 0;
        if (endIdx - startIdx <= DATA_MAX_LENGTH) {
            for (int i = startIdx; i < endIdx; i++) {
                sum += data[i];
            }
            System.out.println(sum);
        } else {
            new SumAction(data,
                          startIdx + DATA_MAX_LENGTH,
                          endIdx).fork();
            new SumAction(data,
                          startIdx,
                          Math.min(endIdx, startIdx + DATA_MAX_LENGTH) )
                            .compute();
        }
    }
}

```

And:

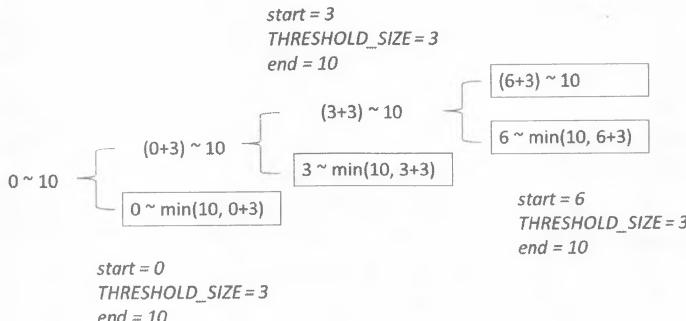
```
public static void main(String[] args) {
    ForkJoinPool p = new ForkJoinPool();
    int data[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
    p.invoke(new SumAction(data, 0, data.length));
}
```

Known the sum of 1 to 10 is 55, which statement is true?

- A. The application prints many values and the sum of them is 55.
- B. The application prints 55.
- C. Compilation error at line1.
- D. The application prints many values and the sum of them exceeds 55.

答案 A

說明



本題沒有呼叫 join()方法，也沒有將多次結果進行 COMBINE，改將過程分次印出，共4次。

❖ 圖 11-10 考題講解

若將上述擬真試題以 RecursiveTask 的形式改寫，加總結果不變：

```
1 class SumTask extends RecursiveTask<Integer> {
2     static final int DATA_MAX_LENGTH = 3;
3     int startIdx, endIdx;
4     int[] data;
5     public SumTask(int[] data, int startIdx, int endIdx) {
6         this.data = data;
7         this.startIdx = startIdx;
8         this.endIdx = endIdx;
9     }
```

```
10     protected Integer compute() {
11         int sum = 0;
12         if (endIdx - startIdx <= DATA_MAX_LENGTH) {
13             for (int i = startIdx; i < endIdx; i++) {
14                 sum += data[i];
15             }
16             return sum;
17         } else {
18             SumTask s1 =
19                 new SumTask (
20                     data,
21                     startIdx + DATA_MAX_LENGTH,
22                     endIdx);
23             s1.fork();
24             SumTask s2 =
25                 new SumTask (
26                     data,
27                     startIdx,
28                     Math.min(endIdx, startIdx + DATA_MAX_LENGTH));
29             return s2.compute() + s1.join();
30         }
31     }
32 }
33
34 public class Test {
35     public static void main(String[] args) {
36         ForkJoinPool p = new ForkJoinPool();
37         int data[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
38         int x = p.invoke(new SumTask(data, 0, data.length));
39         System.out.println(x);
40     }
41 }
```

使用 JDBC 建立 資料庫連線

-
- | 12.1 了解 Database、DBMS 和SQL
 - | 12.2 使用Eclipse 連線並存取資料庫
 - | 12.3 使用JDBC
 - | 12.4 使用JDBC 進行交易
 - | 12.5 使用JDBC 4.1 的RowSetProvider 和RowSetFactory
 - | 12.6 回顧DAO 設計模式
 - | 12.7 認證考試命題範圍

12.1 了解Database、DBMS和SQL

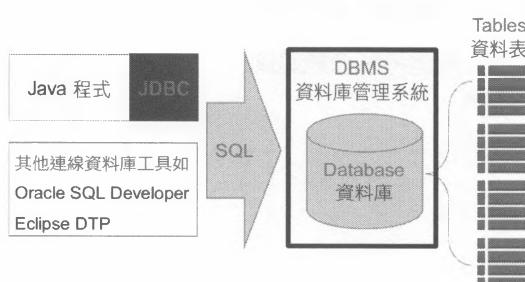
在 Java 裡要保存資料，除了使用物件序列化技術，或使用 I/O 將資料儲存於檔案中，也可以將之儲存於資料庫中。尤其在企業裡，因為資料龐大，程式保存資料的首選就是資料庫。本章簡單介紹資料庫的組成，及如何以特有的 SQL 查詢語法存取資料，最後告訴讀者如何由 Java 透過 JDBC 存取資料庫，及再次檢視 DAO 設計模式。

12.1.1 基本名詞介紹

資料庫領域裡有幾個基本名詞必須先知道：

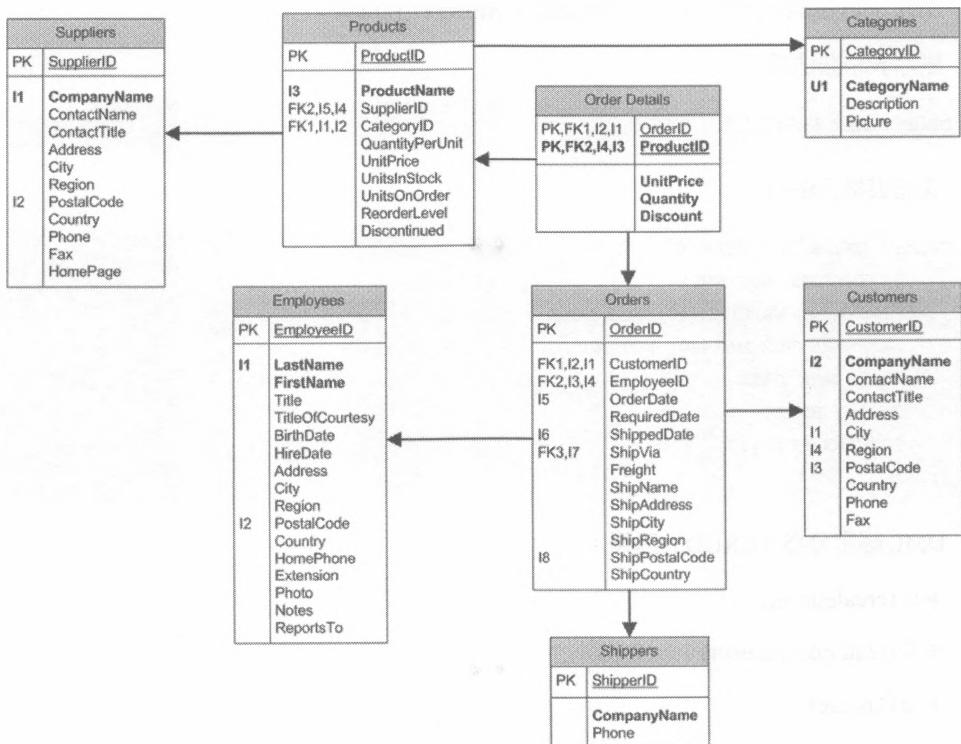
1. Database：資料庫，放置電子資料的地方。
2. DBMS (Database Management System)：為管理資料庫而設計的電腦軟體系統，一般具有儲存、擷取、安全保障、備份等基礎功能。資料庫管理系統可以依據它所支援的資料庫模型來作分類，例如：關聯式、XML 等。透過 DBMS 可以存取 Database。
3. SQL (Structured Query Language)：結構化查詢語言。一種特殊目的之程式語言，用於存取資料庫中的資料。
4. JDBC (Java Database Connectivity)：Java 資料庫連線。是 Java 規範用戶端程式如何來存取資料庫的應用程式介面，提供了諸如查詢和更新資料庫中資料的方法。
5. Table：資料表。是資料庫中呈現資料的邏輯性作法，類似 MS Excel 檔案裡的 sheet。

結合以上，可以繪出資料庫存取架構示意圖：



◆ 圖 12-1 資料庫存取架構示意圖

此外，資料庫裡眾多資料表之間通常都具備「關聯性」，一般可用以下方式表達。和 UML 的類別圖有些相似：



❖ 圖 12-2 資料表關聯性示意圖

12.1.2 使用 SQL 存取資料庫

要存取資料庫必須使用「SQL (Structured Query Language)」，中文為「結構化查詢語言」，常用的有兩大類：

1. DDL (Data Definition Language)，常用於：

- 建立、修改、刪除資料表。
- 描述資料庫中的資料，包括欄位、型態和資料結構等。

2. DML (Data Manipulation Language)，用於：

- 操作資料表。
- 允許使用者存取或是處理資料庫中的資料，而處理的內容包括：擷取資料庫中的資訊，新增、刪除記錄至資料庫中，及更新資料庫中的資料等。

DDL 有許多語法，以本章使用的表格 EMPLOYEE 為例。

刪除表格語法為：

```
DROP TABLE EMPLOYEE;
```

建立表格語法為：

```
CREATE TABLE EMPLOYEE (
    ID INTEGER NOT NULL,
    FIRSTNAME VARCHAR(40) NOT NULL,
    LASTNAME VARCHAR(40) NOT NULL,
    BIRTHDATE DATE,
    SALARY REAL,
    PRIMARY KEY (ID)
);
```

DML 經常分為「CRUD」四類：

- C (create\insert)
- R (read\query\select)
- U (update)
- D (delete)

有時也稱為「新增\刪除\修改\查詢」：

- 新 (新增)
- 刪 (刪除)
- 改 (修改)
- 查 (查詢)

新增資料至 EMPLOYEE 表格的 SQL 範例為：

```
INSERT INTO EMPLOYEE VALUES (1, 'Troy', 'Hammer', '1966-03-31', 100000.00);
INSERT INTO EMPLOYEE VALUES (2, 'Michael', 'Walton', '1966-08-25', 90000.20);
```

查詢 EMPLOYEE 表格的資料的 SQL 範例為：

```
SELECT * FROM EMPLOYEE;
SELECT ID, SALARY FROM EMPLOYEE WHERE ID = 1;
```

修改 EMPLOYEE 表格的資料的 SQL 範例為：

```
UPDATE EMPLOYEE SET SALARY = 0 WHERE ID = 1;
```

刪除 EMPLOYEE 表格的資料的 SQL 範例為：

```
DELETE FROM EMPLOYEE WHERE ID = 1;
```

12.1.3 Derby 資料庫介紹

Apache 軟體基金組織的 Derby 資料庫是一個純用 Java 開發的關聯式資料庫。最初稱作 Cloudscape，為 IBM 所有。IBM 在 2004 年將它捐獻給了 Apache 軟體基金組織，目前是完全免費的。

Java 將 Derby 納為內建資料庫，安裝 JDK 7 或 8 時都會一併安裝，位置在 JDK 目錄的 db 資料夾內：



◆ 圖 12-3 Derby 資料庫安裝目錄

Derby 資料庫的特色有：

- 100% Java 開發。
- 輕量級，大小約 2.6(MB)。
- 支援 JDBC 4.0。
- 支援大部分 ANSI SQL 92 標準。
- 有 Table 和 View。
- 支援 BLOB 和 CLOB 資料類型。
- 支援「預存程序 (Stored Procedure)」。

Derby 的運行模式分二種：

1. 內嵌模式 (embedded mode)

- Derby 資料庫與應用程式共用 JVM，應用程式會在啓動和關閉時分別自動啓動或停止資料庫。
- 使用 derby.jar 支援 Derby 資料庫引擎和嵌入式 JDBC 驅動程式。

2. 網路伺服器模式 (network server mode)

- Derby 資料庫獨佔一個 JVM，作為伺服器上的一個獨立程序 (process) 運行。在這種模式下，允許有多個應用程式來連線同一個 Derby 資料庫。
- 使用 derbyclient.jar 支援 Derby Network Server。

管理 Derby 資料庫的指令檔都放在路徑「\$JAVA_HOME/db/bin」下：



◆ 圖 12-4 Derby 資料庫指令目錄

主要指令檔案用途如下：

◆ 表 12-1 主要指令檔案用途表

指令	用途
startNetworkServer.bat	可啟動網路伺服器的批次檔
stopNetworkServer.bat	可停止網路伺服器的批次檔
ij.bat	互動式 JDBC 批次檔工具
dblook.bat	可檢視資料庫全部或部分 DDL 的批次檔
sysinfo.bat	可顯示有關環境版本資訊的批次檔
NetworkServerControl.bat	可在 NetworkServerControl API 上執行指令的批次檔

12.1.4 操作 Derby 資料庫

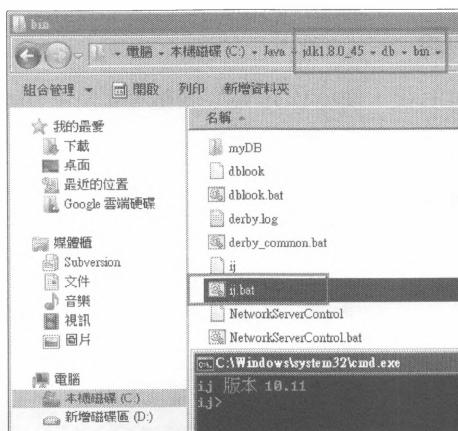
啓動 / 關閉 Derby 資料庫：

- 啓動：點擊 startNetworkServer.bat，出現如下視窗。若 JDK 安裝目錄中有空白(space)存在，如「Program Files」，可能造成問題。
- 關閉：**Ctrl + C**，或是直接關閉啓動時出現的視窗。



❖ 圖 12-5 啟動 Derby 資料庫

若要與 Network Server 互動，點擊 ij.bat，會出現對話主控台 (console)，開始輸入指令和 Network Server 互動：



❖ 圖 12-6 開啟與 Derby 資料庫互動的指令視窗

點擊 ij.bat 後出現的對話主控台 (console)，可以輸入指令：

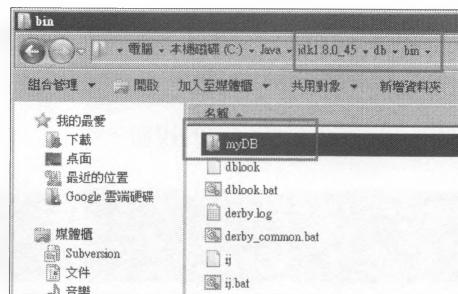
1. 連線 Derby 資料庫

```
connect
'jdbc:derby://localhost:1527/myDB;
create=true;
user=root;
password=sa';
```

行 1 表示要求連線 Derby 資料庫。

行 2 表述資料庫連線位址，使用 JDBC URL，包含位址、port、資料庫名稱。

行 3 表示若資料庫不存在將自動建立。將建立在 bin 目錄下。



❖ 圖 12-7 自動建立新的 Derby 資料庫 -myDB

行 4 提供建立連線的帳號。

行 5 提供建立連線的帳號的密碼。

2. 連線目標資料庫 myDB

連線到目標資料庫 myDB 後，使用 DDL 建立表格，並使用 DML 進行資料的新刪改查。如下圖共包含指令：

- 建立連線(建立資料庫)。
- 建立資料表。
- 建立資料。
- 查詢資料表。

```
cmd /Windows\system32\cmd.exe
i> 连接 30.8
i> connect 'jdbc:derby://localhost:1527/myDB;create=true;user=root;password=aa';
i> CREATE TABLE EMPLOYEE (
ID INTEGER NOT NULL,
FIRSTNAME VARCHAR(40) NOT NULL,
LASTNAME VARCHAR(40) NOT NULL,
BIRTHDATE DATE,
SALARY REAL,
PRIMARY KEY (ID)
);
i>
INSERT INTO EMPLOYEE VALUES (110, 'Troy', 'Hammer', '1965-03-31', 182349.15);
INSERT INTO EMPLOYEE VALUES (123, 'Michael', 'Walton', '1986-08-25', 92400.20);
> > > > > 已插入/更新/刪除 0 列
i> 已插入/更新/刪除 1 列
i> 已插入/更新/刪除 1 列
i> SELECT * FROM EMPLOYEE;
ID          FIRSTNAME          LASTNAME
110         Troy               Hammer
123         Michael            Walton
已選取 2 列
i>
```

❖ 圖 12-8 使用指令與 Derby 資料庫：myDB 互動

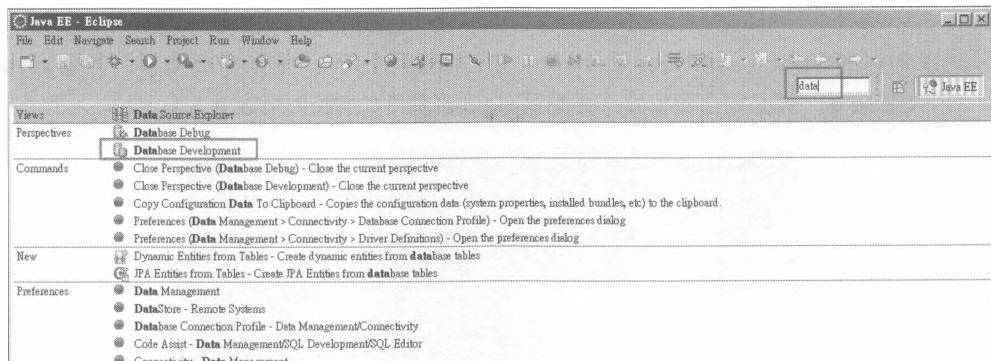
12.2 使用Eclipse連線並存取資料庫

使用 ij.bat 雖然可以連線並操作資料庫，但畢竟不是視窗畫面，一般程式開發人員較不習慣。Eclipse 提供模組「DTP」連線各式資料庫，雖然和專業的資料庫操作軟體還有距離，但已經相當實用，以下簡述操作方式。

12.2.1 連線資料庫

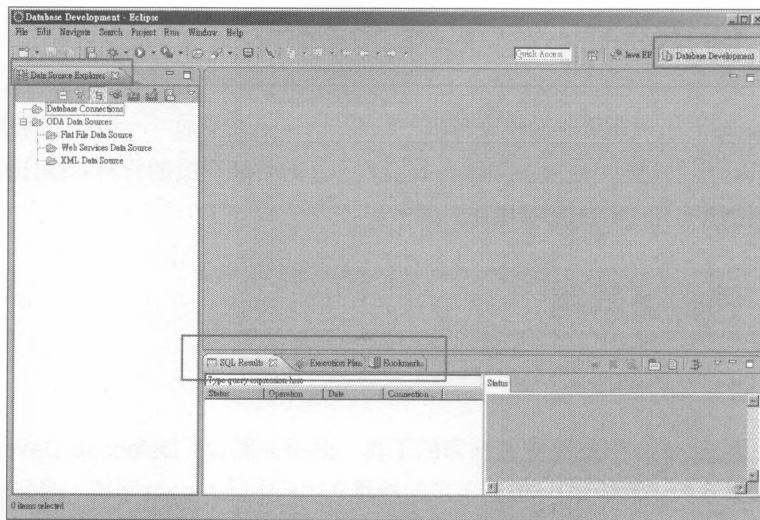
使用 Eclipse 連線 Derby 資料庫步驟為：

STEP01 要使用 Eclipse 作為連線資料庫的工具，必須先開啟「Database Development」的視景 (perspectives)。於右上角的搜尋文字框中輸入 data 字樣，將列出符合的視景，此時點選「Database Development」：



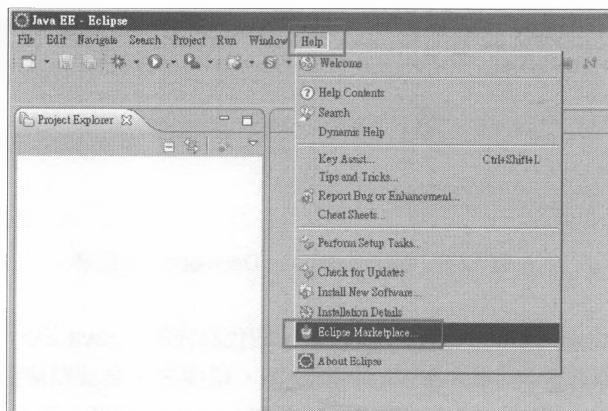
◆ 圖 12-9 選擇 Database Development 視景

STEP02 出現「Database Development」視景，取代原先的「Java EE」視景。Eclipse 的功能將由原先開發 Java 為主的相關視窗，切換主題為資料庫開發。如左側改為「Data Source Explorer」，下方改為「SQL Results」和「Execution Plans」：



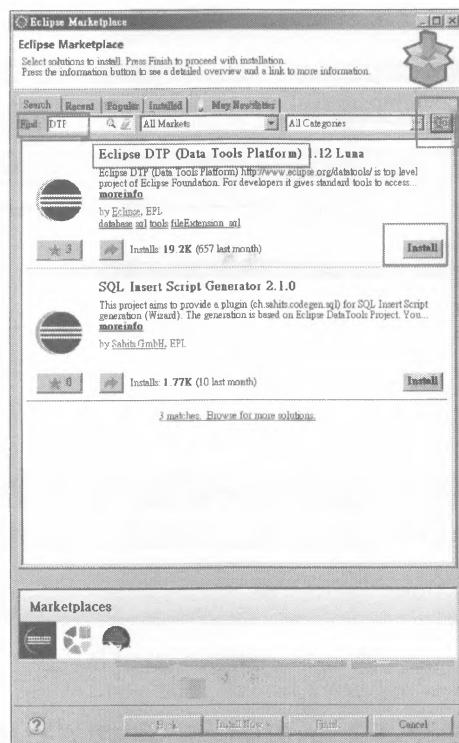
❖ 圖 12-10 開啟 Database Development 視景

STEP03 前述的「Database Development」在 Eclipse 的 Java EE 版本有支援。若是 Java SE 版本，就必須自己安裝。先點選 Help 頁籤下的 Marketplace：



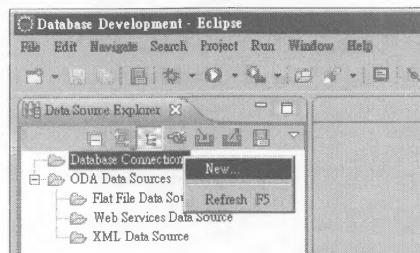
❖ 圖 12-11 點選 Marketplace

STEP04 選擇 Search 頁籤，在 find 欄位裡輸入關鍵字 "DTP"，點選 Go，開始搜尋相關產品。搜尋結束後，選擇「Eclipse DTP (Data Tools Platform)」，點選 Install 後進行安裝。安裝結束需重啟 Eclipse，再依 **STEP01** 選擇並開啟「Database Development」視景：



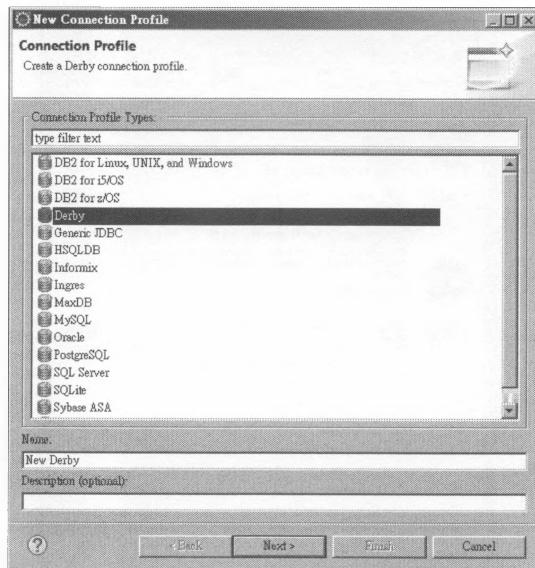
◆圖 12-12 安裝新功能

STEP05 由左側的「Data Source Explorer」，點選節點「Database Connections」，再點選滑鼠右鍵，選擇「New」：



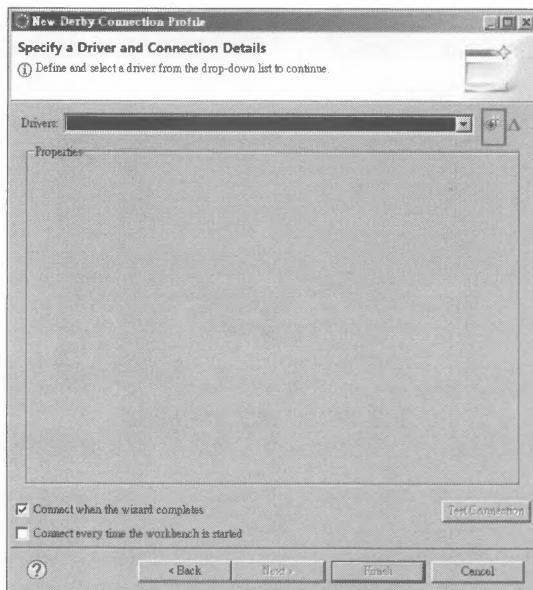
◆圖 12-13 建立資料庫連線

STEP06 選擇要連線的資料庫種類「Derby」，再點選 Next：



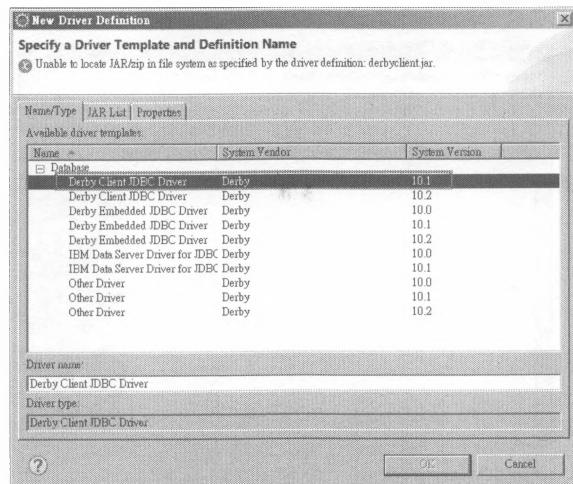
❖ 圖 12-14 選擇連線資料庫種類

STEP07 出現視窗選擇連線資料庫 Derby 的 Drivers (驅動程式)。因為是第一次使用，目前沒有任何可用 driver，必須自己建立。先點選右側的「New Driver Definition」：



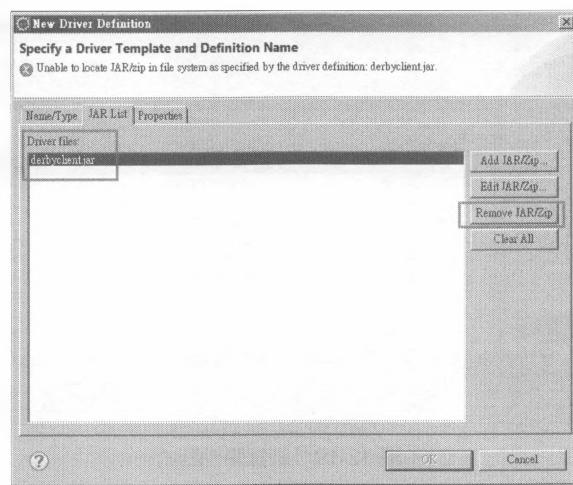
❖ 圖 12-15 選擇資料庫驅動程式

STEP08 在第一個頁籤「Name/Type」裡選擇第一筆「Derby Client JDBC Driver」：



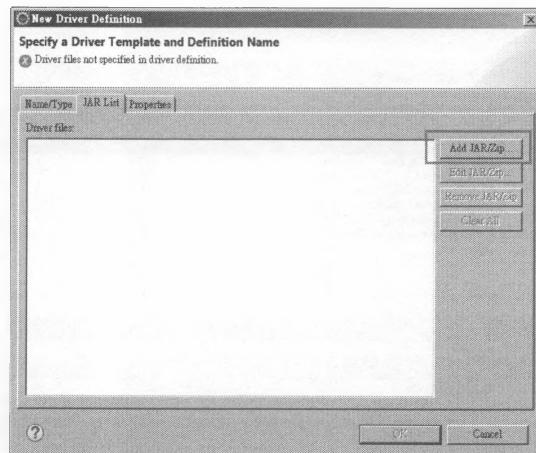
❖ 圖 12-16 選擇版本

STEP09 切換第二個「JAR List」頁籤，點選預設的 derbyclient.jar 檔案，再點擊「Remove JAR/Zip」進行移除作業：



❖ 圖 12-17 移除預設驅動程式

STEP10 切換第二個「JAR List」頁籤，並點選「Add JAR/Zip」按鍵，選擇連線 Derby 資料庫需要的 JAR 檔案：



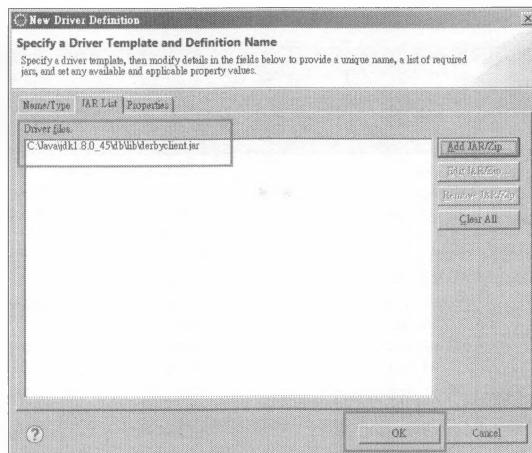
◆ 圖 12-18 新增驅動程式

STEP11 因為在安裝 JDK8 時，預設就會安裝 Derby，因此可以到 JDK8 安裝目錄的 db/lib 下找到 derbyclient.jar 檔案，再點擊「開啟舊檔」：



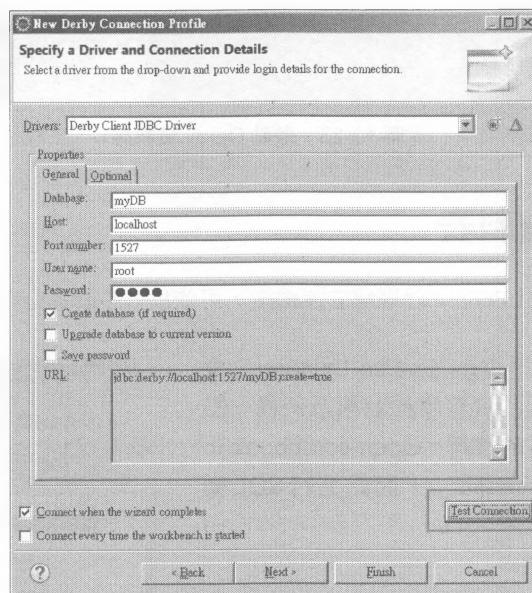
◆ 圖 12-19 選擇驅動程式

STEP12 完成後如下，點選 OK：



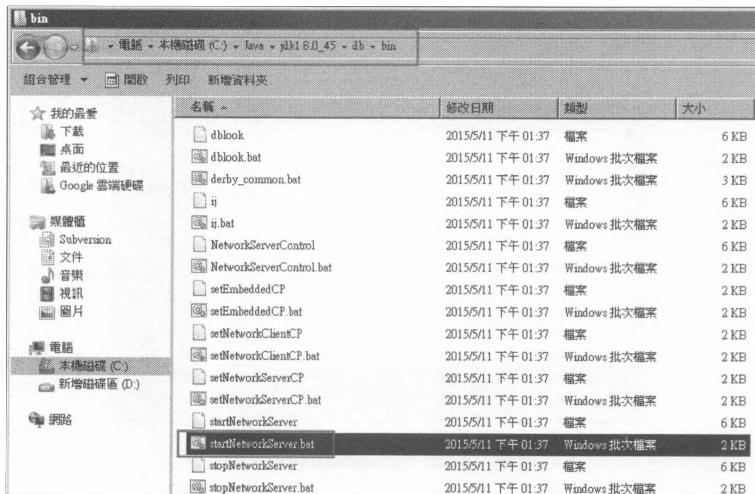
❖ 圖 12-20 新增驅動程式完畢

STEP13 鍵入要連線的 Derby 資料庫的相關資訊後，點擊「Test Connection」可測試連線；要連線前記得先啟動 Derby 資料庫。全部完畢後點擊「Finish」。若 JDK 安裝目錄中有空白 (space) 存在，如「Program Files」，可能造成問題。



❖ 圖 12-21 輸入資料庫連線資訊

STEP14 若發現 Derby 資料庫未啟動，則點擊 JDK 安裝目錄下的「db\bin」內的批次檔「startNetworkServer.bat」：



❖ 圖 12-22 Derby 資料庫啟動批次指令檔

STEP15 彈出以下視窗，顯示 Derby 資料庫已經開啟：

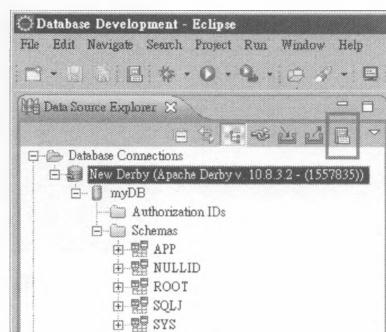


❖ 圖 12-23 啟動 Derby 資料庫

12.2.2 存取資料表

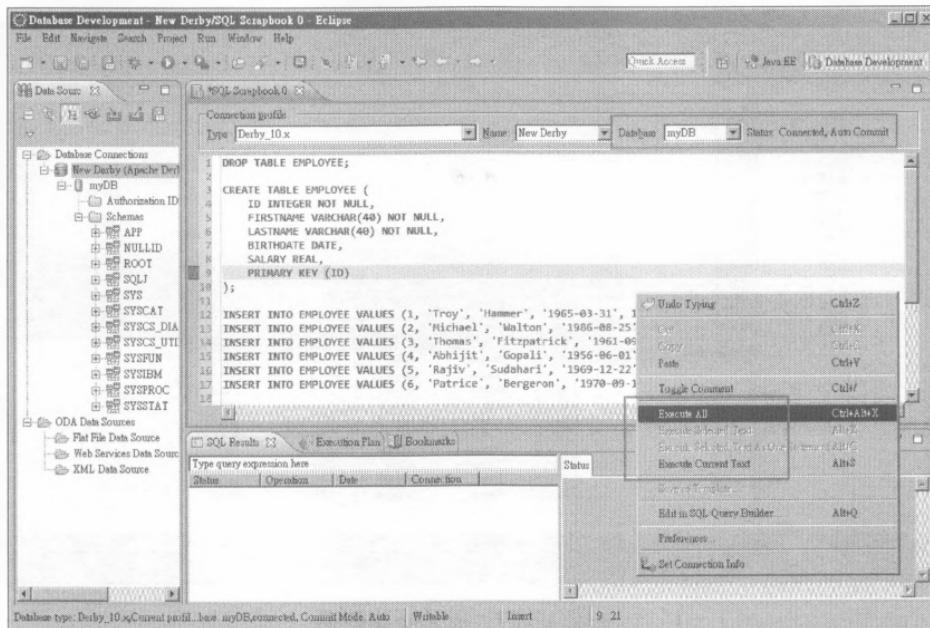
Derby 資料庫啓動且 Eclipse 已經建立連線後，可依以下步驟進行存取：

STEP01 建立連線後，「Data Source Explorer」會出現目前連線的資料庫的狀態及架構。點選頁籤右上方的按鍵「Open scrapbook to edit SQL statements」，開始進行 SQL 編輯及執行：



❖ 圖 12-24 編輯 SQL 指令

STEP02 在右側的 SQL 編輯視窗，選擇要連線的資料庫名稱，並輸入 SQL 指令。最後滑鼠右鍵後選擇執行方式，即可執行 SQL 指令：



◆ 圖 12-25 執行 SQL 指令

STEP03 右下方的「SQL Results」可以看到執行結果：

The screenshot shows the Eclipse Database Development interface. On the left, the Database Connections panel lists a connection named 'myDB' under 'New Derby (Apache Derby)'. The SQL Scratches 0 tab is active, displaying the following SQL code:

```
9 PRIMARY KEY ( ID )
10 );
11
12 INSERT INTO EMPLOYEE VALUES (1, 'Troy', 'Hammer', '1965-03-31', 102349.15);
13 INSERT INTO EMPLOYEE VALUES (2, 'Michael', 'Walton', '1986-08-25', 92400.20);
14 INSERT INTO EMPLOYEE VALUES (3, 'Thomas', 'Fitzpatrick', '1961-09-22', 75123.45);
15 INSERT INTO EMPLOYEE VALUES (4, 'Abhijit', 'Gopal', '1956-06-01', 98345.00);
16 INSERT INTO EMPLOYEE VALUES (5, 'Rajiv', 'Sudharsi', '1969-12-22', 68400.22);
17 INSERT INTO EMPLOYEE VALUES (6, 'Patrice', 'Bergeron', '1970-09-18', 156345.00);
18
19 INSERT INTO EMPLOYEE VALUES (8, 'Matthieu', 'Williams', '1966-05-31', 150345.15);
20 INSERT INTO EMPLOYEE VALUES (9, 'Michael', 'McGinn', '1979-01-25', 97800.20);
21 INSERT INTO EMPLOYEE VALUES (10, 'Thomas', 'Heiner', '1967-07-22', 79343.45);
22 INSERT INTO EMPLOYEE VALUES (11, 'Peter', 'Forrester', '1965-11-01', 80345.00);
23 INSERT INTO EMPLOYEE VALUES (12, 'Kenny', 'Arlington', '1959-10-22', 45405.22);
24
25 SELECT * FROM EMPLOYEE WHERE FIRSTNAME LIKE 'T%'
```

The SQL Results tab shows the execution details and the results of the query:

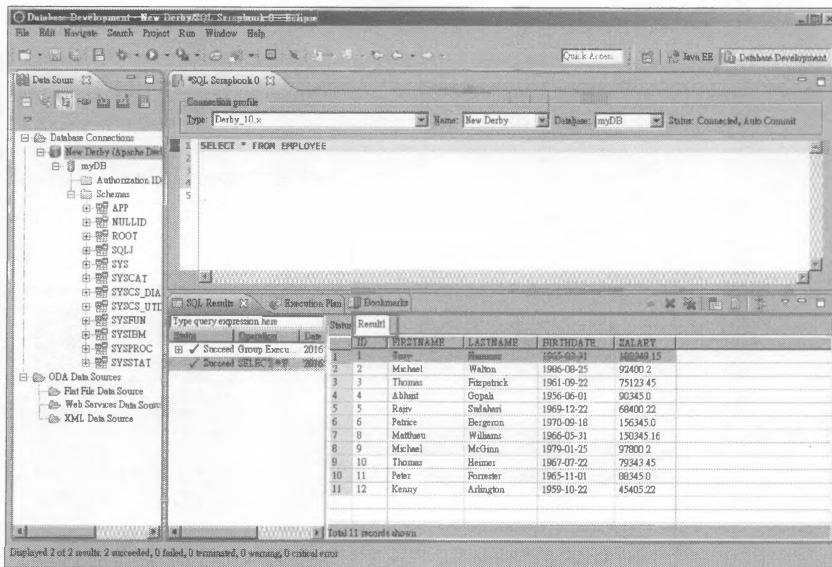
Status	Operation	Date	Connection
Succeed	Group Exec	2016年6月6日	New Derby

Status: INSEET INTO EMPLOYEE VALUES (6, Michael, McGinn, 1979-01-25, 97800.20)
INSEET INTO EMPLOYEE VALUES (10, Thomas, Heiner, 1967-07-22, 79343.45)
INSEET INTO EMPLOYEE VALUES (11, Peter, Forrester, 1965-11-01, 80345.00)
INSEET INTO EMPLOYEE VALUES (12, Kenny, Arlington, 1959-10-22, 45405.22)
SELECT * FROM EMPLOYEE WHERE FIRSTNAME LIKE 'T%'
Elapsed Time: 0 hr, 0 min, 0 sec, 7 ms.

Displayed 1 of 1 results. 1 succeed, 0 failed, 0 terminated, 0 warning, 0 copied rows.

❖ 圖 12-26 檢視執行結果

STEP04 若執行查詢，也可以得到結果：

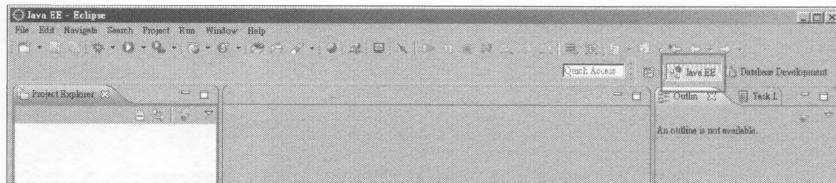


◆ 圖 12-27 顯示查詢結果

12.2.3 建立預存程序

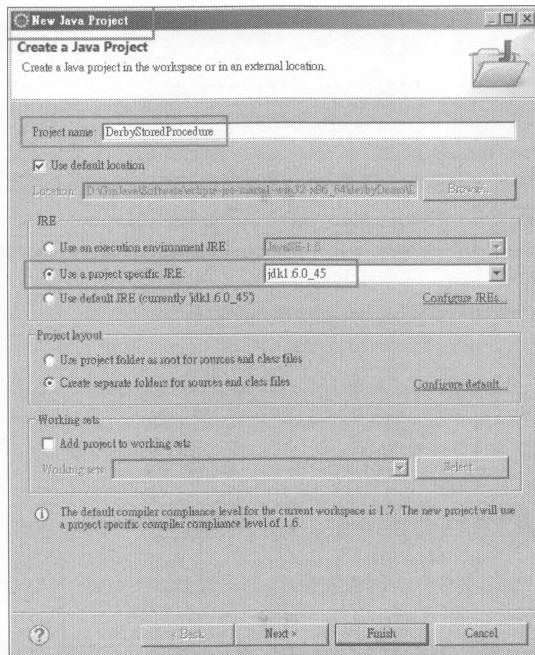
一般資料庫建立預存程序是以 ANSI SQL 為主，再輔以自家客製化 SQL 語言，如 Oracle 的 PLSQL 語言，或 MSSQL 的 TSQL 語言。Derby 資料庫也支援建立「預存程序（Stored Procedure）」，但它的預存程序是使用純 Java 語言建立。使用 Eclipse 建立 Derby 資料庫的預存程序的步驟為：

STEP01 建立 Derby 裡的 Java Stored Procedure (Java 預存程序)。先切回 Java EE 視景：



◆ 圖 12-28 切回 Java EE 視景

STEP02 建立 Java Project 「DerbyStoredProcedure」，並指定 JDK 版本 6：



❖ 圖 12-29 建立 Java 專案

**STEP03 建立類別「DerbyStoredProcedure」，如範例「/DerbyStoredProcedure/src/
EmployeeStoredProcedure.java」：**

② 範例

```

1  public class EmployeeStoredProcedure {
2      public static void countSalary(float salary, int[] count)
3          throws SQLException {
4      String url = "jdbc:default:connection";
5      Connection con = DriverManager.getConnection(url);
6      String query = "SELECT COUNT(*) AS count
7          FROM Employee WHERE Salary > ?";
8      PreparedStatement ps = con.prepareStatement(query);
9      ps.setFloat(1, salary);
10     ResultSet rs = ps.executeQuery();
11     if (rs.next()) {
12         count[0] = rs.getInt(1);
13     } else {
14         count[0] = 0;
15     }

```

```

14         con.close();
15     }
16     public static void countAge(Date birthDay, int[] count)
17             throws SQLException {
18         String url = "jdbc:default:connection";
19         Connection con = DriverManager.getConnection(url);
20         String query =
21             "SELECT COUNT(*) " + "AS count "
22             + "FROM Employee " + "WHERE Birthdate <= ?";
23         PreparedStatement ps = con.prepareStatement(query);
24         ps.setDate(1, birthDay);
25         ResultSet rs = ps.executeQuery();
26         if (rs.next()) {
27             count[0] = rs.getInt(1);
28         } else {
29             count[0] = 0;
30         }
31         con.close();
32     }

```

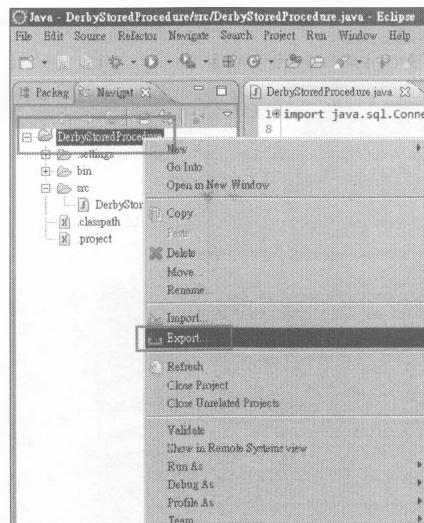
```

1 import java.sql.Connection;
2 import java.sql.Date;
3 import java.sql.DriverManager;
4 import java.sql.PreparedStatement;
5 import java.sql.ResultSet;
6 import java.sql.SQLException;
7
8 public class EmployeeStoredProcedure {
9
10    public static void countSalary(float salary, int[] count) throws SQLException {
11        String url = "jdbc:default:connection";
12        Connection con = DriverManager.getConnection(url);
13        String query = "SELECT COUNT(*) AS count FROM Employee WHERE Salary > ?";
14        PreparedStatement ps = con.prepareStatement(query);
15        ps.setFloat(1, salary);
16        ResultSet rs = ps.executeQuery();
17        if (rs.next()) {
18            count[0] = rs.getInt(1);
19        } else {
20            count[0] = 0;
21        }
22        con.close();
23    }
24
25    public static void countAge(Date birthDay, int[] count) throws SQLException {
26
27    }
28
29 }

```

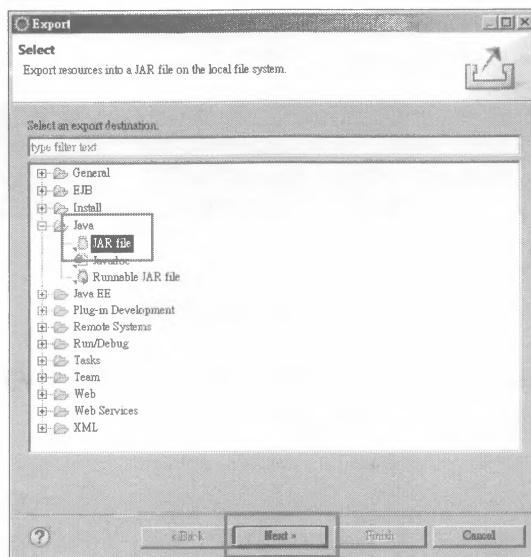
✿ 圖 12-30 建立類別 EmployeeStoredProcedure

STEP04 使用 Eclipse 將專案匯出。點選專案後使用滑鼠右鍵，選擇「Export」：



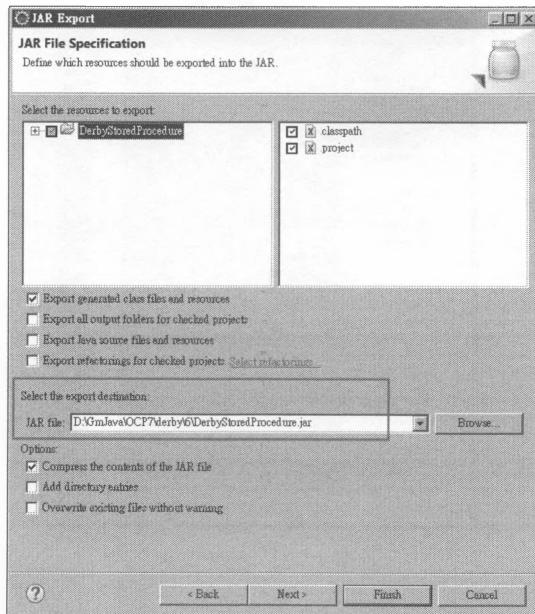
❖ 圖 12-31 匯出專案

STEP05 將檔案匯出為「JAR file」：



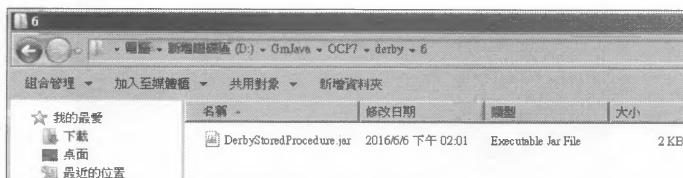
❖ 圖 12-32 選擇 JAR 檔案為匯出格式

STEP06 選擇匯出位置後完成。



◆ 圖 12-33 選擇匯出位置

STEP07 檢視生成目錄，的確看到 JAR file 已經建立：



◆ 圖 12-34 JAR 檔案成功匯出

STEP08 切回「Database Development」視景，執行以下指令安裝先前步驟產生的 JAR 檔案，並建立 (Java) Stored Procedure：

◎ 範例

```

1  CALL SQLJ.remove_jar (
      'ROOT.DerbyStoredProcedure',
      0
);
2  CALL SQLJ.install_jar (
      'D:\GmJava\OCP7\derby\6\DerbyStoredProcedure.jar',

```

```

        'ROOT.DerbyStoredProcedure',
        0
    );
3 CALL syscs_util.syscs_set_database_property(
        'derby.database.classpath',
        'ROOT.DerbyStoredProcedure'
);
4 DROP PROCEDURE EMP_SALARY_COUNT;
5 CREATE PROCEDURE EMP_SALARY_COUNT (
    IN salary REAL,
    OUT num INTEGER)
    LANGUAGE JAVA
    PARAMETER STYLE JAVA
    DYNAMIC RESULT SETS 0
    READS SQL DATA
    EXTERNAL NAME 'EmployeeStoredProcedure.countSalary';
6 DROP PROCEDURE EMP_AGE_COUNT;
7 CREATE PROCEDURE EMP_AGE_COUNT (
    IN birthDay DATE,
    OUT num INTEGER)
    LANGUAGE JAVA
    PARAMETER STYLE JAVA
    DYNAMIC RESULT SETS 0
    READS SQL DATA
    EXTERNAL NAME 'EmployeeStoredProcedure.countAge';

```

說明

1	使用「SQLJ.remove_jar」指令移除過去安裝的 JAR 檔案，名稱為 DerbyStoredProcedure，假如已經存在。
2	使用「SQLJ.install_jar」指令安裝 JAR 檔案。需指定名稱 DerbyStoredProcedure，和檔案所在位置。
3	將安裝的 JAR 檔案，設定為 Derby 的屬性「derby.database.classpath」值。若有多個 JAR 檔案以「：」符號區隔。執行步驟 2 安裝 JAR 檔案後，一定要設定本屬性以告知 Derby 由那個 JAR 檔案去載入 class。
4	使用 SQL 語法刪除預存程序「EMP_SALARY_COUNT」，假如已經存在資料庫裡。
5	使用 SQL 語法在資料庫裡建立預存程序，給予名稱「EMP_SALARY_COUNT」，並連結外部的類別名稱與方法「EmployeeStoredProcedure.countSalary」。因架構需要，若類別方法要讓 Derby 的預存程序呼叫，必須宣告為 static。



```

createStoredProcedures.sql
Connection profile: New Derby 7
Type: Derby_10.x
Name: myDB
Database: myDB
Status: Connected, Auto Commit

1 CALL SQLJ.remove_jar ('ROOT.DerbyStoredProcedure', e);
2
3 CALL SQLJ.install_jar ('D:\GmJava\OCP7\derby\6\DerbyStoredProcedure.jar','ROOT.DerbyStoredProcedure',e);
4
5 CALL syscs_util.syscs_set_database_property('derby.database.classpath','ROOT.DerbyStoredProcedure');
6
7 DROP PROCEDURE EMP_SALARY_COUNT;
8 CREATE PROCEDURE EMP_SALARY_COUNT (
9 IN age REAL,
10 OUT num INTEGER)
11 LANGUAGE JAVA
12 PARAMETER STYLE JAVA
13 DYNAMIC RESULT SETS 0
14 READS SQL DATA
15 EXTERNAL NAME 'EmployeeStoredProcedure.countSalary';
16
17 DROP PROCEDURE EMP_AGE_COUNT;
18 CREATE PROCEDURE EMP_AGE_COUNT (
19 IN birthDay DATE,
20 OUT num INTEGER)
21 LANGUAGE JAVA
22 PARAMETER STYLE JAVA
23 DYNAMIC RESULT SETS 0
24 READS SQL DATA
25 EXTERNAL NAME 'EmployeeStoredProcedure.countAge';
26

```

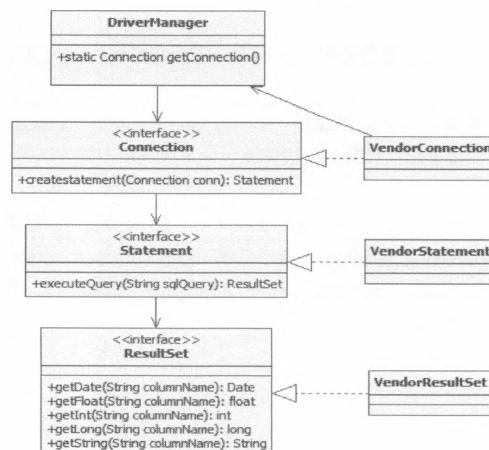
❖ 圖 12-35 建立 Java Stored Procedure

如此，Derby 裡的預存程序就建立完成！

12.3 使用 JDBC

12.3.1 JDBC API 概觀

和 JDBC 有關的 UML 類別圖如下：



❖ 圖 12-36 JDBC 相關 API 類別圖

由 UML 看出，JDBC API 主要由一個 class 和三個 interface 組成，除 DriverManager 外都是介面。關係是：

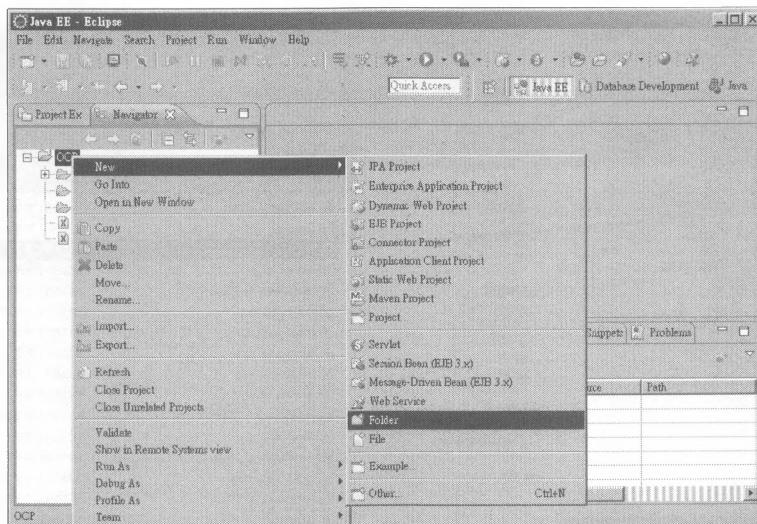
- 使用 DriverManager 取得 Connection。
- 使用 Connection 取得 Statement。
- 使用 Statement 取得 ResultSet。

因為市場上有多種資料庫且由不同廠商負責，連線資料庫的機制該由廠商提供。因此 JDBC 只定義抽象介面，由各家資料庫廠商提供實作類別；亦即在上一章介紹 Eclipse 連線資料庫時，需要選擇的 JAR 檔案，就是驅動程式。

12.3.2 取得 JDBC 驅動程式

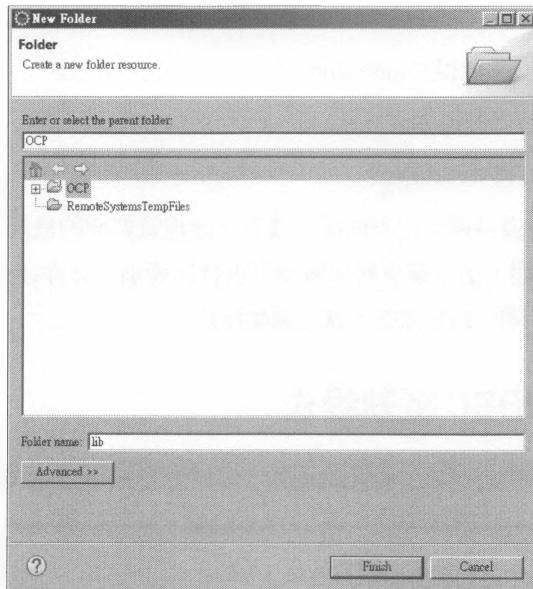
在章節「12.2.1 連線資料庫」時，**STEP11** 示範如何在 Eclipse 的模組「DTP」中加入連線 Derby 資料庫所需要的驅動程式，亦即 JAR 檔案。若要使用 Java 程式直接連線 Derby 資料庫，也需要在 Eclipse 的專案裡放入該 JAR 檔，如此程式才知道去哪裡找到 JAR 檔案。步驟為：

STEP01 在專案中新建 Folder：



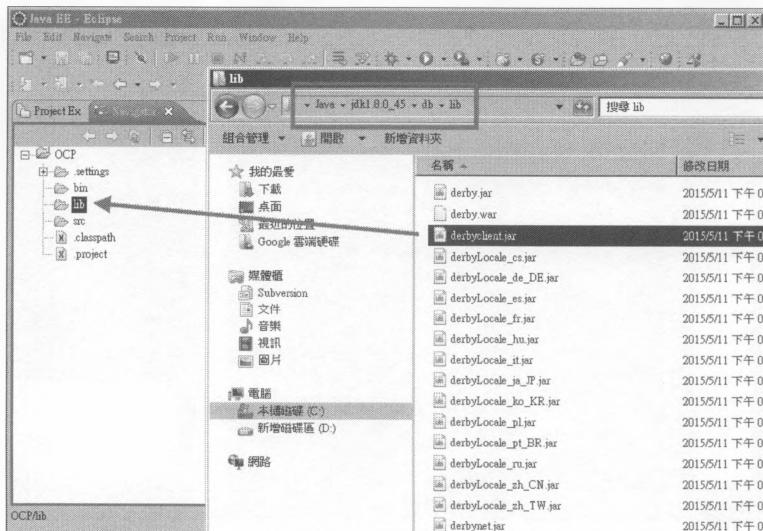
❖ 圖 12-37 專案中新增 Folder

STEP02 將 Folder 取名「lib」：



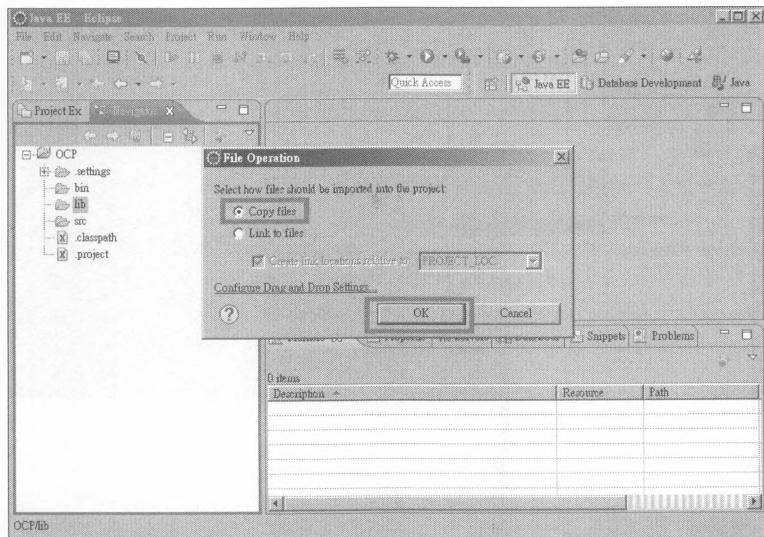
❖ 圖 12-38 命名 Folder

STEP03 將 JAR 檔案「derbyclient.jar」以拖拉的方式複製到該新建的 lib 資料夾中：



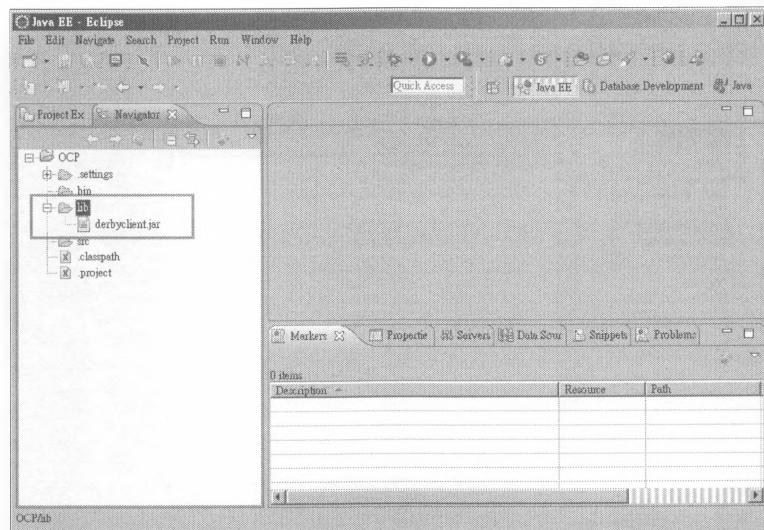
❖ 圖 12-39 複製 derbyclient.jar 檔案至 lib 資料夾

STEP04 點選「Copy files」，再點擊「OK」：



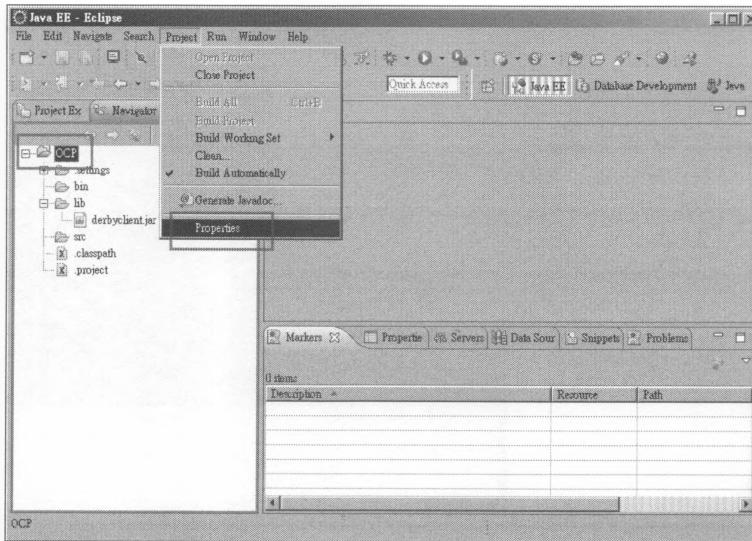
❖ 圖 12-40 同意複製檔案

STEP05 複製 JAR 檔案至 Eclipse 專案的步驟完成：



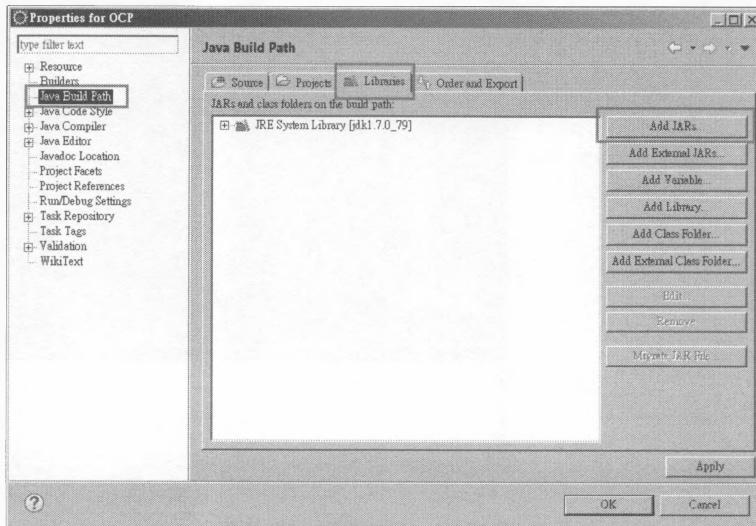
❖ 圖 12-41 完成複製 JAR 檔案

STEP06 點選專案節點，並開啟 Project 頁籤的 Properties 選項：

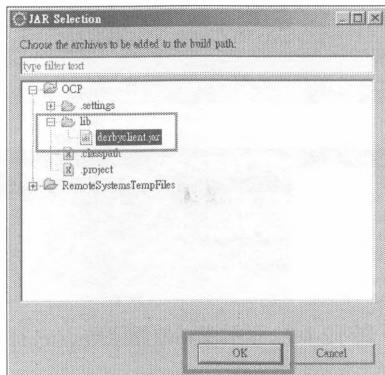


❖ 圖 12-42 設定專案屬性

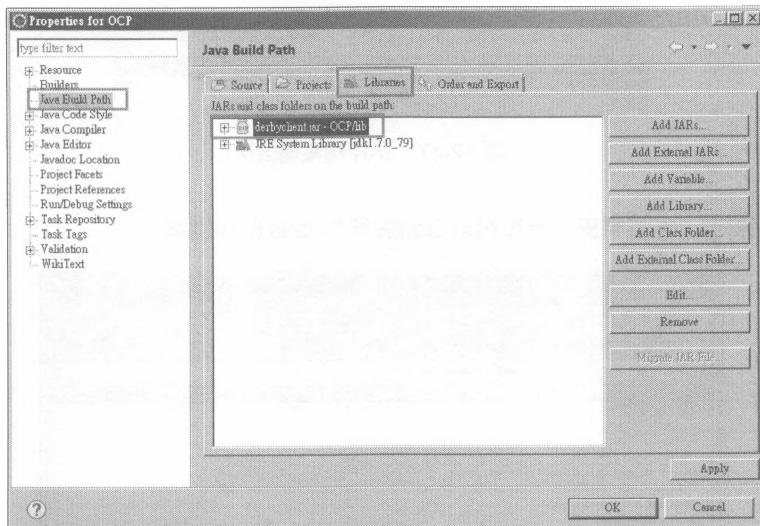
STEP07 在專案屬性視窗的左側，選擇「Java Build Path」，右側選擇「Libraries」頁籤，再點擊「Add JARs」按鍵：



❖ 圖 12-43 將複製後的 JAR 檔案加入專案屬性中

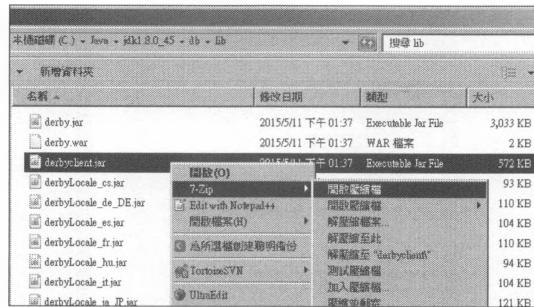
STEP08 選擇檔案位置：

❖ 圖 12-44 選擇檔案位置

STEP09 設定完成後，可以看到 derbyclient.jar 檔案將會出現在專案的 Libraries 中：

❖ 圖 12-45 完成專案屬性設定

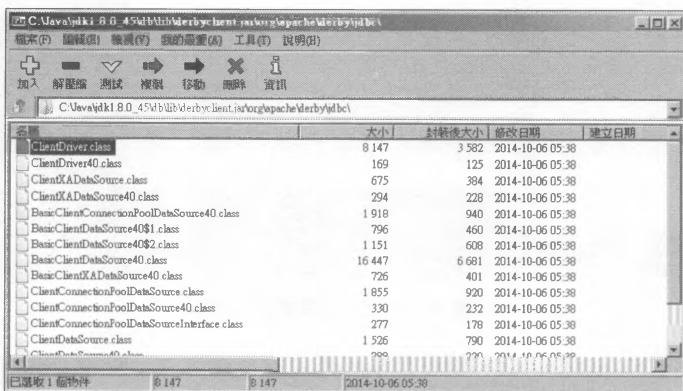
事實上，Java 的 JAR 檔案是一個壓縮檔，裡面通常會包含一些設定檔和類別檔，所以可以嘗試用 7-Zip 等免費的壓縮軟體去打開。步驟如下：

STEP01 點選 JAR 檔案，以 7-Zip 軟體開啟壓縮檔：

❖ 圖 12-46 以解壓縮軟體開啟 JAR 檔案

STEP02 開啟後，可以看到內容基本結構：

❖ 圖 12-47 JAR 檔案開啟後

STEP03 逐層點開 org 資料夾，最後可以看到許多 *.class 的類別檔：

❖ 圖 12-48 JAR 檔案由許多類別檔組成

因此，告訴 Java 如何找到 derbyclient.jar 檔案其實還不夠。還必須告訴 Java 如何在這麼一堆類別檔案裡，找到主要的入門 / 入口檔。

這部分，在「JDBC 4.0 前 / 後」有很大不同。

1. JDBC 4.0 之前

在呼叫 `DriverManager.getConnection()` 前，必須明確指出驅動程式的主類別為何。以 Derby 的驅動程式為例，必須如下。因為有可能類別字串寫錯，或是忘記將 JAR 檔案加入 Eclipse 的專案中而找不到，所以必須處理例外類別「`ClassNotFoundException`」：

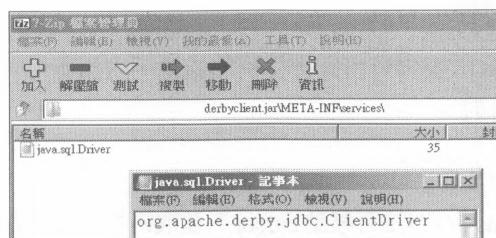
```
try {
    //java.lang.Class.forName("<fully qualified path of the driver>");
    java.lang.Class.forName("org.apache.derby.jdbc.ClientDriver");
} catch (ClassNotFoundException c) {
}
```

或是在指令列裡指定：

```
java -djdbc.drivers=<fully qualified path to the driver> <class to run>
```

2. JDBC 4.0 之後

因為驅動程式的主類別已經註冊在 JAR 檔案裡的「`META-INF/services/java.sql.Driver`」，如下圖。因此，在 `DriverManager.getConnection()` 時，就不需要在程式碼裡特別註記驅動程式主類別。



◆ 圖 12-49 JDBC 4.0 之後的驅動程式變革

12.3.3 開發 JDBC 程式

開發 JDBC 程式的幾個主要步驟是：

1. 指定 URL (Uniform Resource Locator)

URL 用來指出連線資料庫時使用的 driver 名稱 / 種類，資料庫位置 (IP + Port)，或是其他建立連線時需要的一併提供的屬性名稱與值。語法為：

■ 語法

`jdbc:<driver>:[sub_protocol:] [databaseName] [;attribute=value]`

每一個資料庫廠商都應該負責提供自己的 JDBC 驅動程式。以 Derby 資料庫為例：

```
String url = "jdbc:derby://localhost:1527/EmployeeDB";
```

以 Oracle 資料庫為例：

```
String url = "jdbc:oracle:thin:@//myhost:1521/orcl";
```

2. 使用 DriverManager 取得 java.sql.Connection 的物件，該物件將建立與資料庫的連線 (session)：

```
Connection con = DriverManager.getConnection(url, username, password);
```

3. 使用 java.sql.Connection 取得 java.sql.Statement 物件：

```
Statement stmt = con.createStatement();
```

4. 使用 java.sql.ResultSet 取得 java.sql.Statement 執行 SQL 後的查詢結果：

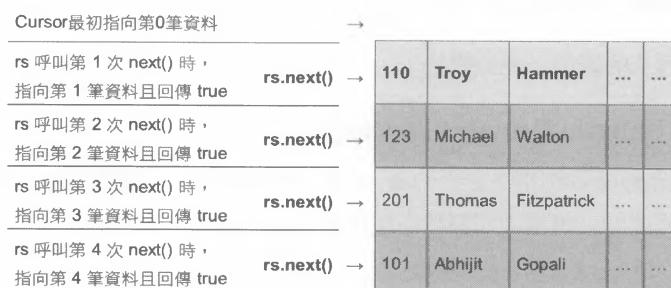
```
String query = "SELECT * FROM Employee";
ResultSet rs = stmt.executeQuery(query);
```

5. ResultSet 的特性與使用方式

ResultSet 物件代表 SQL 查詢資料庫後得到的結果。內部使用「游標 (cursor)」的移動代表目前所讀取的資料列：

- Cursor 最初指向第 0 筆資料。
- 呼叫 ResultSet 的 next() 方法可移動 cursor，並取得指向某筆資料的 cursor。
- Cursor 若回傳 false，表示已無資料可以讀取。

示意圖如下：



❖ 圖 12-50 ResultSet 的 next() 方法與 cursor 移動示意圖

此外，ResultSet 物件具有多種屬性可以設定：

◆表 12-2 ResultSet 常用屬性列表

分類依據	屬性	用途
Concurrency	CONCUR_READ_ONLY	指向資料是唯讀
	CONCUR_UPDATABLE	指向資料可修改
Type	TYPE_FORWARD_ONLY	游標只能往前
	TYPE_SCROLL_INSENSITIVE	游標可往前往後，無法感知資料被修改
	TYPE_SCROLL_SENSITIVE	游標可往前往後，可以感知資料被修改

必須在取得之前就進行設定：

```
Statement stmt =
    con.createStatement( ResultSet.TYPE_SCROLL_INSENSITIVE,
                        ResultSet.CONCUR_UPDATABLE );
ResultSet rs = stmt.executeQuery("SELECT a, b FROM TABLE2");
```

設定後的實際情況必須視該種資料庫的廠商否實作該屬性而定。

ResultSet 可以使用回傳各種型態的 getter 方法取得每筆資料的每個欄位內容。結合使用 DriverManager、Connection、Statement 與 ResultSet 的完整範例如「/OCP/src/course/c12/SimpleJDBCTest.java」。本範例可以輸出資料庫 myDB 內的資料表 Employee 的所有資料：

範例

```
1  public class SimpleJDBCTest {
2      public static void main(String[] args) {
3          String url = "jdbc:derby://localhost:1527/myDB";
4          String username = "root";
5          String password = "sa";
6          String query = "SELECT * FROM Employee";
7          try (Connection con =
8              DriverManager.getConnection(url, username, password)) {
9              Statement stmt = con.createStatement();
10             ResultSet rs = stmt.executeQuery(query) {
11                 while (rs.next()) {
12                     int empID = rs.getInt("ID");
13                     String first = rs.getString("FirstName");
14                     String last = rs.getString("LastName");
15                     Date birthDate = rs.getDate("BirthDate");
16                     float salary = rs.getFloat("Salary");
17                     System.out.println(empID + "\t" + first + "\t"
18                         + last + "\t" + birthDate + "\t" + salary);
19             }
20         }
21     }
22 }
```

```

17     }
18 } catch (SQLException e) {
19     System.out.println("SQL Exception: " + e);
20 }
21 }
22 }

```

12.3.4 結束 JDBC 相關物件的使用

JDBC 用來存取資料庫的主要物件：Connection、Statement 和 ResultSet 都實作了「java.lang.AutoCloseable」介面，都屬於外部資源，因此使用後都必須呼叫 close() 方法予以關閉。其原則是：

1. 關閉 Connection 物件，會自動關閉相關 Statement 物件；關閉 Statement 物件，也會自動關閉相關 ResultSet 物件；但此時 ResultSet 所對應的相關資源 (resource)，並未被自動關閉或釋出；必須等待 Java 啓動 GC 機制。只有明確呼叫 ResultSet 的 close() 方法，才能馬上釋放相關資源。
2. 若使用相同 Statement 物件重新執行查詢，則原先已開啟的 ResultSet 將自動關閉，再使用該 ResultSet 就會出錯。
3. 應該明確呼叫 Connection、Statement 和 ResultSet 的 close() 方法，或是使用「try-with-resource」敘述。關閉資源的順序會和開啓時順序相反：

```

try (Connection con =
      DriverManager.getConnection(url, username, password);
      Statement stmt = con.createStatement();
      ResultSet rs = stmt.executeQuery(query)) {
    //...
}

```

4. 只有在「try-with-resource」區塊裡明確宣告的物件才會被自動關閉。因此以下做法只有 ResultSet 物件會被自動關閉，不是好習慣：

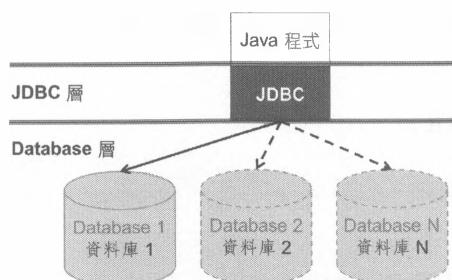
```

try (ResultSet rs =
      DriverManager.getConnection(url, username, password)
      .createStatement()
      .executeQuery(query)) {
    //...
}

```

12.3.5 開發可攜式的 JDBC 程式碼

JDBC 相關 API 的設計，是希望能使用物件導向裡多型的概念，讓 Java 程式碼可以依賴於 JDBC 建立的抽象層，而不是和底層的資料庫綁定太深，太依賴資料庫。如此未來若需要抽換資料庫，可以影響最小。因此主要的 Connection、Statement 和 ResultSet 都為介面，再讓資料庫廠商去實作。這種將系統架構「分層 (insulating layer)」的概念，示意如下：



❖ 圖 12-51 以 JDBC 的 API 將 Java 程式和資料庫分層

除此之外，由「美國國家標準學會 (American National Standards Institute, ANSI)」所定義的「SQL-92 Entry-level specification」，也希望所有資料庫廠商在開發自己的資料庫時，都能夠支援 SQL-92 的標準查詢語法，儘量讓相同的語法可以在不同的資料庫間使用，減少使用者在不同資料庫間語法轉換的困擾。

確認使用中的資料庫是否有支援 SQL-92 的標準查詢語法，可以藉由 DatabaseMetaData 介面的 supportsANSI92EntryLevelSQL() 方法回傳 true 或 false 來確認，如下：

```

DatabaseMetaData dbm = con.getMetaData();
if (dbm.supportsANSI92EntryLevelSQL()) {
}
  • supportsAlterTableWithAddColumn() : boolean - DatabaseMetaData
  • supportsAlterTableWithDropColumn() : boolean - DatabaseMetaData
  • supportsANSI92EntryLevelSQL() : boolean - DatabaseMetaData
  • supportsANSI92FullSQL() : boolean - DatabaseMetaData
  • supportsANSI92IntermediateSQL() : boolean - DatabaseMetaData
  • supportsBatchUpdates() : boolean - DatabaseMetaData
  • supportsCatalogInDataManipulation() : boolean - DatabaseMetaData
  • supportsCatalogIndexDefinitions() : boolean - DatabaseMetaData
  • supportsCatalogInPrivilegeDefinitions() : boolean - DatabaseMetaData
  • supportsCatalogInProcedureCalls() : boolean - DatabaseMetaData
  • supportsCatalogInTableDefinitions() : boolean - DatabaseMetaData
  • supportsColumnAliasing() : boolean - DatabaseMetaData
  • supportsConvert() : boolean - DatabaseMetaData
  • supportsConvert(int fromType, int toType) : boolean - DatabaseMetaData
  
```

❖ 圖 12-52 以 DatabaseMetaData 呼叫各種 supportsXXX() 方法

如範例「/OCP/src/course/c12/SimpleJDBCTest2.java」的方法「showDatabaseMetaData()」：

範例

```

1  private static void showDatabaseMetaData(Connection con)
2                                throws SQLException {
3      DatabaseMetaData dbm = con.getMetaData();
4      System.out.println("Support for Entry-level SQL-92 standard: "
5                          + dbm.supportsANSI92EntryLevelSQL());
6  }

```

即便如此，仍須注意撰寫 SQL 時的語法。如以下展示在不同資料庫裡要撈取十筆資料時的 SQL 語法：

- MS SQL Server (TSQL) :

```
Select top 10 * from some_table
```

- Oracle (PLSQL) :

```
Select * from some_table where rownum <= 10
```

這種使用各資料庫原生 (native) 的 SQL 的程式碼將造成轉換資料庫的困難。

12.3.6 使用 java.sql.SQLException 類別

`SQLException` 類別用於回報存取資料庫時產生的各種錯誤。和一般 `Exception` 不同，可以得到更多訊息。以下程式片段使用 `getSQLState()`、`getErrorCode()`、`getMessage()` 和 `getNextException()` 等方法取得和資料庫相關的錯誤訊息：

```

catch(SQLException ex) {
    while(ex != null) {
        System.out.println("SQLState: " + ex.getSQLState());
        System.out.println("Error Code in DB:" + ex.getErrorCode());
        System.out.println("Message: " + ex.getMessage());
        Throwable t = ex.getCause();
        while(t != null) {
            System.out.println("Cause:" + t);
            t = t.getCause();
        }
        ex = ex.getNextException();
    }
}

```

12.3.7 Statement 介面與 SQL 敘述的執行

要執行 SQL 敘述(statement)時，必須要有 Statement 的物件；也可以說 Statement 介面是 SQL statement 的包裹類別(wrapper class)：

```
Statement stmt = con.createStatement();
String sqlStatement = "select * ...";
ResultSet rs = stmt.executeQuery(sqlStatement);
```

根據 SQL 敘述的不同種類，有不同執行方式：

SQL 敘述的種類	方法	回傳
SELECT	executeQuery(sql)	ResultSet
INSERT、UPDATE、DELETE、DDL…	executeUpdate(sql)	int (影響的資料筆數)
任何 SQL 指令	execute(sql)	boolean (若有 ResultSet)

12.3.8 使用 ResultSetMetaData 介面

使用 ResultSetMetaData 介面取得 ResultSet 的

- 欄位數量，使用 getColumnCount() 方法。
- 欄位名稱，使用 getColumnName() 方法。
- 欄位型態，使用 getColumnTypeName() 方法。

必須注意的是，指定欄位的 index 由「1」起算，非「0」。如範例「/OCP/src/course/c12/SimpleJDBCTest2.java」的方法「showResultSetMetaData()」：

◎ 範例

```
1 private static void showResultSetMetaData(ResultSet rs)
2     throws SQLException {
3     int numCols = rs.getMetaData().getColumnCount();
4     String[] colNames = new String[numCols];
5     String[] colTypes = new String[numCols];
6     for (int i = 0; i < numCols; i++) {
7         colNames[i] = rs.getMetaData().getColumnName(i + 1);
8         colTypes[i] = rs.getMetaData().getColumnTypeName(i + 1);
9     }
10    System.out.println("Number of columns returned: " + numCols);
11    System.out.println("Column names/types returned: ");
```

```

11     for (int i = 0; i < numCols; i++) {
12         System.out.println(colNames[i] + " : " + colTypes[i]);
13     }
14 }
```

說明

- | | |
|---|---|
| 6 | 資料表的欄位位置由 1 起算，但迴圈的 i 由 0 開始，所以必須是 i+1。 |
| 7 | 資料表的欄位位置由 1 起算，但迴圈的 i 由 0 開始，所以必須是 i+1。 |

結果

```

Number of columns returned: 5
Column names/types returned:
ID : INTEGER
FIRSTNAME : VARCHAR
LASTNAME : VARCHAR
BIRTHDATE : DATE
SALARY : REAL
```

12.3.9 取得查詢結果的資料筆數

要取得查詢結果的資料筆數，可以利用游標(cursor)的前後移動，所以必須先設定屬性：

```

Statement stmt = con.createStatement (
    ResultSet.TYPE_SCROLL_INSENSITIVE,
    ResultSet.CONCUR_READ_ONLY);
```

執行查詢取得 ResultSet 物件後，可以由範例「/OCP/src/course/c12/SimpleJDBCTest2.java」的方法「rowCountByCursor ()」，取得結果的資料筆數：

範例

```

1 private static int rowCountByCursor (ResultSet rs) throws SQLException {
2     int rowCount = 0;
3     int currRow = rs.getRow();
4     if (!rs.last())
5         return -1;
6     rowCount = rs.getRow();
7     if (currRow == 0)
8         rs.beforeFirst();
9     else
```

```

10     rs.absolute(currRow);
11     return rowCount;
12 }

```

說明

3	先紀錄目前 cursor 位置。
4~5	使用 last() 方法將游標移到最後一筆資料。同時，若發現沒資料，該方法會回傳 false。
6	因為游標已經移到最後一筆資料，游標所在位置即為資料筆數。
7~10	將 cursor 移回原先位置。

也可以使用 SQL 語法裡的 count() 函數直接取得滿足條件的資料筆數。因為毋須調整 cursor，使用預設的 Statement 物件即可。如範例「/OCP/src/course/c12/SimpleJDBCTest2.java」的方法「rowCountBySQL()」：

範例

```

1 private static int rowCountBySQL(Connection con, String money)
                                    throws SQLException {
2
3     String query =
4         "SELECT COUNT(*) FROM Employee WHERE Salary > "
5         + money;
6     try (Statement stmt = con.createStatement();
7          ResultSet rs = stmt.executeQuery(query)) {
8         rs.next();
9         int count = rs.getInt(1);
10        return count;
11    }
12 }

```

說明

3	每個 Statement 物件只能用來執行 1 次 SQL，不能重複使用，所以傳入 Connection 物件，以之重新產生 Statement 物件，再執行 SQL。
---	--

12.3.10 控制 ResultSet 每次由資料庫取回的筆數

使用 Statement 物件執行 Query 後取得的 ResultSet 物件，並不是由資料庫將資料一次全部拿回來：

```

ResultSet rs = stmt.executeQuery(query));
while ( rs.next() ) {
    //...
}

```

當呼叫 `while (rs.next()) { }` 時，Java 會自資料庫中「每次抓回一定的筆數」才進行迴圈，以減少對資料庫的頻繁 I/O。該筆數預設由 JDBC 驅動程式控制。若要自己控制，則做法為：

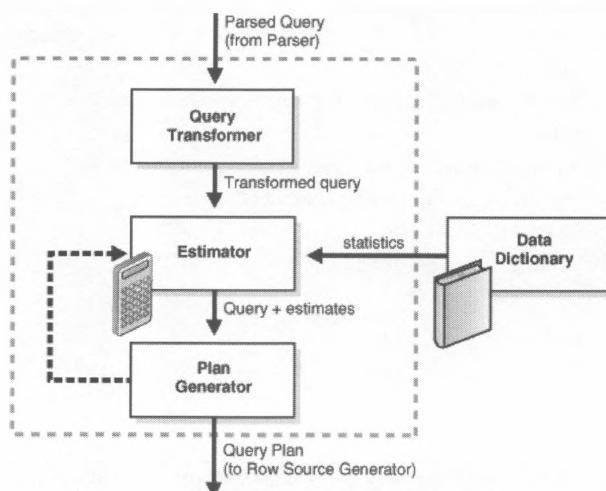
```
rs.setFetchSize(25);
```

所以查詢後每次只拿 25 筆資料，第 26 筆時再連線資料庫撈取下一批資料(25 筆)。

主要是避免若一次由資料庫中取得太多資料，影響 JVM。

12.3.11 使用 PreparedStatement 介面

當我們對資料庫發出 SQL 語句請求查詢或執行指令時，資料庫需對該 SQL 語句進行編譯程序，以產出最佳執行計畫 (execution plan)，來確保查詢效能。以 Oracle 資料庫而言，過程大致如下：



◆ 圖 12-53 產出 SQL 執行計畫的流程 (資料來源：Oracle 網站)

這樣的過程，目的在讓 SQL 的執行效率可以最佳化。但因為流程本身必須使用 CPU 資源進行分析，若頻率太高，將造成資料庫主機的負載。

解決方式在於讓「類似的 SQL」可以重複使用相同的執行計畫，避免編譯的工作一再發生。例如，以下 SQL：

```
SELECT * FROM Employee WHERE Salary > 100;
SELECT * FROM Employee WHERE Salary > 1000;
```

```
SELECT * FROM Employee WHERE Salary > 9;
SELECT * FROM Employee WHERE Salary > 11023;
```

因為真正改變的只有 Salary 欄位的金額，因此以「？」取代 Salary 欄位的金額後，先將 SQL 送出至資料庫，Salary 欄位的金額後送，如此二段式的改變就可以避免該 SQL 敘述送至資料庫後因每次金額不同被反覆編譯：

```
SELECT * FROM Employee WHERE Salary > ?;
```

這樣的做法，在 Oracle 資料庫裡稱為使用「繫結變數(Bind Variables)」。在 JDBC 裡則稱為使用「PreparedStatement 介面」，該介面繼承 Statement 介面，讓預先編譯好 (precompiled) 的 SQL 可以再和傳入的參數配合。如範例「/OCP/src/course/c12/SimpleJDBCTest2.java」的方法「runPreparedStatement()」：

範例

```
1 private static void runPreparedStatement(
2         Connection con, double value) throws SQLException {
3     String query = "SELECT * FROM Employee WHERE Salary > ? ";
4     PreparedStatement pStmt = con.prepareStatement(query);
5     pStmt.setDouble(1, value);
6     ResultSet rs = pStmt.executeQuery();
7     printResultSet(rs);
}
```

前述範例中，將回傳 Salary > value 的員工查詢結果。SQL 中每個「？」號都必須有相應數值藉由 setXXX(index, value) 的 setter 方法帶入 value。Index 由 1 起算，配合「？」的出現順序。如此可避免每次執行 SQL 時，資料庫重新編譯 SQL 所耗費的資源。

此外，避免「SQL injection」的網路駭客攻擊也是使用 PreparedStatement 的一個原因。範例「/OCP/src/course/c12/SimpleJDBCTest2.java」的方法「runSqlInjection()」做了一個容易招致 SQL injection 的示範：

範例

```
1 private static void runSqlInjection(Connection con, String value)
2         throws SQLException {
3     String query = "SELECT * FROM Employee WHERE Salary > " + value;
4     Statement stmt = con.createStatement();
5     ResultSet rs = stmt.executeQuery(query);
6     printResultSet(rs);
}
```

比較兩個方法的傳入參數：

`runPreparedStatement()` 的第二個參數使用 `double`，`runSqlInjection()` 的第二個參數則使用 `String`。若有心人士了解資料表結構，或是故意在傳入的字串後面加上「`or 1=1`」，讓原先正常的 SQL：

```
SELECT * FROM Employee WHERE Salary > 100;
```

被惡意更改為：

```
SELECT * FROM Employee WHERE Salary > 100 or 1=1;
```

這樣的查詢語法，將會帶出表格 `Employee` 內所有資料，就可能讓公司營業機密資料或個資外洩。

12.3.12 使用 CallableStatement 介面

我們在 12.2.3 節的 **STEP08** 中已經建立了預存程序，其定義為：

```
CREATE PROCEDURE EMP_SALARY_COUNT (
    IN salary REAL,
    OUT num INTEGER)
LANGUAGE JAVA
PARAMETER STYLE JAVA
DYNAMIC RESULT SETS 0
READS SQL DATA
EXTERNAL NAME 'EmployeeStoredProcedure.countSalary';
```

這個預存程序的名稱為「`EMP_SALARY_COUNT`」：

- 第一個參數作為「輸入」用途，在變數 `salary` 前使用「`IN`」宣告。
- 第二個參數則作為「輸出」用途，在變數 `num` 前使用「`OUT`」宣告。

要呼叫資料庫裡的預存程序，可以使用 JDBC 的 `CallableStatement` 介面。該介面繼承 `Statement` 介面，使用方式如範例「`/OCP/src/course/c12/SimpleJDBCTest2.java`」的「`runCallableStatement()`」方法：

範例

```
1  private static void runCallableStatement(Connection con, float salary)
                                         throws SQLException {
2      CallableStatement cStmt =
          con.prepareCall("{CALL EMP_SALARY_COUNT(?, ?)}");
```

```

3   cStmt.setFloat(1, salary);
4   cStmt.registerOutParameter(2, Types.INTEGER);
5   boolean result = cStmt.execute();
6   int count = cStmt.getInt(2);
7   System.out.println("Result : " + result);
8   System.out.println("There are " + count +
9                     " Employees over the salary of " + salary);
}

```

說明

2	使用 CallableStatement 介面並搭配字串「{CALL EMP_SALARY_COUNT (?, ?)}」呼叫預存程序。
3	設定第 1 個參數的值。該值為 float 型態，且為「輸入」用途。
4	因為第 2 個參數為「輸出」用途，使用 registerOutParameter() 方法將其註冊為「輸出」用途，並設定輸出型態是整數 (integer)。
5	執行 CallableStatement 後若回傳 true，表示結果是 ResultSet。若是 false，表示結果可能是資料更新筆數或是其他。必須注意的是，取得這個結果無助於取得我們需要的數值。
8	因為行 4 已經註冊了第 2 個參數是負責輸出，因此資料庫會將預存程序裡真正要輸出的值寫入該參數。本例中藉由 getInt(2) 取得第 2 個參數的整數值，即為結果。

12.4 使用 JDBC 進行交易

12.4.1 何謂資料庫交易？

讓「多個」對資料庫的存取的行為，含 query、delete、insert 和 update 等指令，視同一個，且同進退；亦即一起發生 / 提交 (commit)，或一起未發生 / 返回 (rollback)。一個交易裡的行為可以跨資料庫，但會更複雜。

交易具有四項特徵，簡稱 ACID：

1. Atomicity (原子性)：交易裡的所有行為，將一起完成，或一起未完成 (回復至未進行交易的狀態)。
2. Consistency (一致性)：交易使系統由原來一致性的狀態，轉換至另一個一致性的狀態。
3. Isolation (獨立性)：兩個同時發生的交易，彼此互不影響。
4. Durability (持久性)：已經完成的交易將繼續保持，即便系統毀損也可以使用資料庫的交易紀錄日誌 (transaction log) 還原。

以銀行轉帳為例。假設帳號 A 有存款 500 元，帳號 B 有存款 1000 元。若打算由帳號 A 轉帳 100 元到帳號 B，則轉帳結束後，帳號 A 該有 400 元，帳號 B 有 1100 元。

若銀行無交易管控機制，一旦轉帳失敗，帳號 A 少 100 元，但帳號 B 金額未增加。如圖 12-54 所示：

轉帳成功：



轉帳失敗：



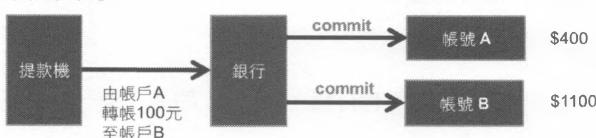
❖ 圖 12-54 銀行轉帳若無交易機制

若銀行有交易管控機制，一旦轉帳完成，帳號 A 及帳號 B 同時下達「commit」指令，完成交易。如圖 12-55 所示：

啟動交易：



確認交易：



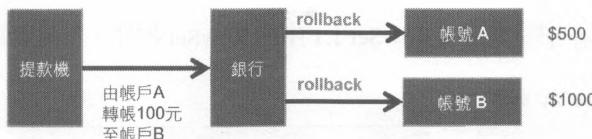
❖ 圖 12-55 銀行轉帳時使用交易且過程順利

若銀行有交易管控機制，且過程中遇到問題失敗，則帳號 A 及帳號 B 同時下達「rollback」指令，取消交易。如圖 12-56 所示：

啟動交易：



取消交易：



❖ 圖 12-56 銀行轉帳時使用交易但過程遇到問題

12.4.2 使用 JDBC 的交易

當 connection 物件建立時，預設為「自動提交 (auto commit)」模式。此時單一 SQL 將被視為獨立交易，完成後自動 commit。若要將二個以上的 SQL 作成交易群組，必須先關閉「auto commit」模式：

```
con.setAutoCommit (false);
```

以後，完成交易時必須呼叫方法：

```
con.commit();
```

也可以取消交易：

```
con.rollback();
```

JDBC 沒有明確的方法「啓動」一個交易。依據 JDBC JSR (221) 所提供的綱要：

1. 以關閉「auto commit」模式的時候開始，接下來的所有 SQL 都算成同一個交易，直到 commit 或 rollback 方法被執行。
2. 若交易進行中「auto commit」模式被改變，則交易將自動 commit。

12.5 使用JDBC 4.1的RowSetProvider和RowSetFactory

Java 7 導入新版 RowSet 1.1，使用「javax.sql.rowset.RowSetProvider」取得 RowSetFactory 物件，其預設實作是「com.sun.rowset.RowSetFactoryImpl」：

```
RowSetFactory myRowSetFactory = RowSetProvider.newFactory();
```

而 RowSetFactory 則用來建立 RowSet 1.1 中的 RowSet 物件，常見有以下數種：

◆表 12-3 常見 RowSet 列表

功能	功能
CachedRowSet	可以將資料庫取得的資料儲存在記憶體中，避免經常連線。
FilteredRowSet	繼承 CachedRowSet，可以有過濾資料功能。
JdbcRowSet	是 ResultSet 的 wrapper 物件，讓 ResultSet 的行為像 JavaBean。也可以和資料庫保持連線狀態。
JoinRowSet	可以將兩個不同的 RowSet 合併成一個 JoinRowSet，功能像 SQL 的表格 join。
WebRowSet	支援將 RowSet 以標準的 XML 格式表現。

範例「/OCP/src/course/c12/JdbcRowSetTest.java」顯示如何使用「JdbcRowSet」的物件：

◎ 範例

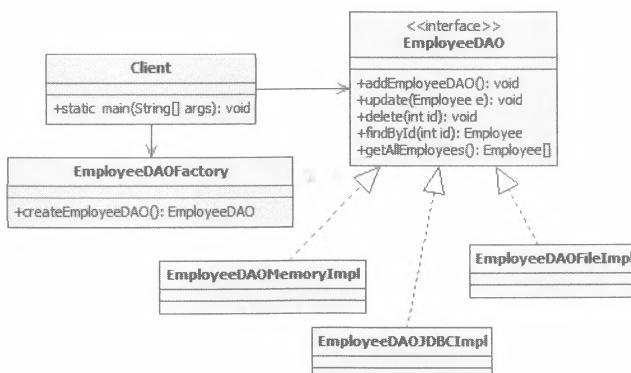
```
1 public class JdbcRowSetTest {
2     public static void main(String[] args) throws SQLException {
3         String url = "jdbc:derby://localhost:1527/myDB";
4         String username = "root";
5         String password = "sa";
6         RowSetFactory myRowSetFactory = RowSetProvider.newFactory();
7         try (JdbcRowSet jdbcRs = myRowSetFactory.createJdbcRowSet()) {
8             jdbcRs.setUrl(url);
9             jdbcRs.setUsername(username);
10            jdbcRs.setPassword(password);
11            jdbcRs.setCommand("SELECT * FROM Employee");
12            jdbcRs.execute();
13            while (jdbcRs.next()) {
14                int empID = jdbcRs.getInt("ID");
15                String first = jdbcRs.getString("FirstName");
16                String last = jdbcRs.getString("LastName");
17                Date birthDate = jdbcRs.getDate("BirthDate");
18                float salary = jdbcRs.getFloat("Salary");
19                System.out.println(
20                    "ID: " + empID + "\t" +
21                    "Employee Name: " + first + " " + last + "\t" +
```

```

22         "Birth Date: " + birthDate + "\t" +
23         "Salary: " + salary);
24     }
25   }
26 }
27 }
```

12.6 回顧DAO設計模式

了解 JDBC 的程式開發方式後，介面 EmployeeDAO 就可以多一種實作類別：EmployeeDAOJDBCImpl：



◆ 圖 12-57 DAO 設計模式回顧

12.7 認證考試命題範圍

依據甲骨文公布的考試範圍 (https://education.oracle.com/java-se-8-programmer-ii/pexam_1Z0-809)，和本章相關的主題有：

Building Database Applications with JDBC

1. Describe the interfaces that make up the core of the JDBC API including the Driver, **Connection**, **Statement**, and **ResultSet** interfaces and their relationship to provider implementations.

2. Identify the components required to connect to a database using the **DriverManager** class including the JDBC URL.
3. Submit queries and read results from the database including **creating** statements, **returning** result sets, **iterating** through the results, and properly **closing** result sets, statements, and connections.

本章擬真試題實戰

考題 1

Which code fragment show the proper way to handle JDBC resources? And suppose variables of

String query;

Connection con;

are valid :

A.

```
try {  
    ResultSet rs = stmt.executeQuery(query);  
    Statement stmt = con.createStatement();  
    while (rs.next()) /* . . . */  
} catch (SQLException e) {  
}
```

B.

```
try {  
    Statement stmt = con.createStatement();  
    ResultSet rs = stmt.executeQuery(query);  
    while (rs.next()) /* . . . */  
} catch (SQLException e) {  
}
```

C.

```
Statement stmt;  
ResultSet rs;  
try {  
    stmt = con.createStatement();  
    rs = stmt.executeQuery(query);  
    while (rs.next()) /* . . . */  
} finally {  
    rs.close();  
    stmt.close();  
}
```

D.

```

Statement stmt;
ResultSet rs;
try {
    rs = stmt.executeQuery(query);
    stmt = con.createStatement();
    while (rs.next()) {/* . . . */}
} finally {
    rs.close();
    stmt.close();
}

```

答案 C**說明** 使用 JDBC 資源後應該使用 finally 區塊關閉。

考題 2

Suppose url user, password are valid, and given:

```

public static void main(String[] args) throws SQLException {
    String url = "...";
    String user = "...";
    String password = "...";
    Connection conn = DriverManager.getConnection(url, user, password);
    Statement stmt = conn.createStatement();
    ResultSet rs;
    try {
        stmt.executeUpdate("delete from student");
        conn.setAutoCommit(false);
        stmt.executeUpdate("insert into student values(1, 'Sam')");
        Savepoint save1 = conn.setSavepoint("point1");
        stmt.executeUpdate("insert into student values(2, 'Jane')");
        conn.rollback();
        stmt.executeUpdate("insert into student values(3, 'John')");
        conn.setAutoCommit(true);
        stmt.executeUpdate("insert into student values(4, 'Jack')");
        rs = stmt.executeQuery("select * from student");
        while (rs.next()) {
            System.out.println(rs.getString(1) + " " + rs.getString(2));
        }
    } catch (Exception e) {
        System.out.print(e.getMessage());
    }
}

```

```

    }
}

```

What is the result after execution?

A. 1 Sam

B. 4 Jack

C. 3 John

4 Jack

D. 1 Sam

3 John

4 Jack

答案 C

說明 建立 Savepoint save1 = conn.setSavepoint("point1") 後，可以搭配：

```
conn.rollback(save1);
```

將交易退回到建立 save1 的時間點，或稱檢查點 (check point)。此時，答案為 D。

但因為本題實際執行：

```
conn.rollback();
```

因此，將把所有先前的交易全部取消，故為 C。

考題 3

Which two actions can be used in registering a JDBC 3.0 driver?

- A. Add the driver class to the META-INF/services folder of the JAR file.
- B. Set the driver class name by using the jdbc.drivers system property.
- C. Include the JDBC driver class in a jdbcproperties file.
- D. Use the java.lang.class.forName method to load the driver class.
- E. Use the DriverManager.getDriver method to load the driver class.

答案 BD

說明 參照本書「12.3.2 取得 JDBC 驅動程式」內容。

考題 4

Suppose url and query are valid, and given:

```
public static void main(String[] args) {  
    String url = "...";  
    String query = "...";  
    try (Connection con = DriverManager.getConnection(url)) {  
        Statement stmt = con.createStatement();  
        ResultSet rs = stmt.executeQuery(query);  
        while (rs.next()) {  
            // ...  
        }  
    } catch (SQLException e) {  
        e.printStackTrace();  
    }  
}
```

What change should you make to apply good coding practices to this fragment?

- A. Add nested try-with-resources statements for the statement and ResultSet declarations.
- B. Add the Statement and ResultSet declarations to the try-with-resources statement.
- C. Add a finally clause after the catch clause.
- D. Rethrow SQLException.

答案 B

說明 本題只將 Connection 的宣告放在 try-with-resource 的程式區塊裡，應該再將 Statement 及 ResultSet 的宣告一併放入，才能在結束時自動關閉資源。

考題 5

Which two statements are true about RowSet implementation?

- A. A JdbcRowSet object provides a JavaBean view of a result set.
- B. A CachedRowSet provides a connected view of the database.
- C. A FilteredRowSet object filter can be modified at any time.
- D. A WebRowSet returns JSON-formatted data.

答案 AC

說明 選項 B : CachedRowSet 可以將資料庫取得的資料儲存在記憶體中，避免經常連線。
選項 D : WebRowSet 是以 XML 格式呈現。

考題 6

Which code fragment is required to load a JDBC 3.0 driver?

- A. DriverManager.loadDriver ("org.apache.derby.jdbc.ClientDriver");
- B. Class.forName("org.apache.derby.jdbc.ClientDriver");
- C. Connection con = Connection.getDriver("jdbc:derby://localhost:1527/myDB");
- D. Connection con = DriverManager.getConnection("jdbc:derby://localhost:1527/myDB");

答案 B

考題 7

Which three are true?

- A. setAutoCommit (false) method will start a transaction context.
- B. An instance of Savepoint represents a point in the current transaction context.
- C. A rollback () method invocation rolls a transaction back to the last savepoint.
- D. A rollback () method invocation releases any database locks currently held by this connection object.
- E. After calling rollback(mysavepoint), you must close the savepoint object by calling mySavepoint.close() .

答案 ABD

說明 選項 C：呼叫 rollback(mysavepoint) 才會。

選項 E：Savepoint 來自 Connection，一旦 Connection 結束，就跟著結束；並且沒有 close() 方法。

考題 8

Suppose the connection object is valid and there are records in Employee table. Given:

```
public static void main(String[] args) {
    try {
        // other JDBC codes
        Statement statement = connection.createStatement();
        statement.execute("SELECT * FROM Employee where id = 1"); //line1
        ResultSet rs = statement.getResultSet(); //line2
```

```

        System.out.println(rs.getInt("id"));
    } catch (SQLException ex) {
        System.out.println("SQLException...");
    }
}

```

Assume that the SQL query matches one record. What is the result of compiling and executing this code?

- A. The code prints SQLException....
- B. The code prints the employee ID.
- C. Compilation fails due to an error at //line1.
- D. Compilation fails due to an error at //line2.

答案 A

說明 本題執行時拋出 `java.sql.SQLException: Invalid operation at current cursor position。` 呼叫 `rs.getInt("id")` 前，應先呼叫 `rs.next()` 使 cursor 移到第一筆資料的地方。

考題 9

Suppose the connection object is valid and there are records in Employee table. Given:

```

public static void main(String[] args) {
    try {
        // other JDBC codes
        String query = "SELECT * FROM Employee";
        Statement stmt = connection.createStatement();
        ResultSet rs = stmt.executeQuery(query);
        ResultSetMetaData rsmd = rs.getMetaData();      // line1
        int colCount = rsmd.getColumnCount();
        while (rs.next()) {
            for (int i = 1; i <= colCount; i++) {
                System.out.print(rs.getObject(i) + " ");    // line2
            }
            System.out.println();
        }
    } catch (SQLException se) {
        System.out.println("SQLException...");
    }
}

```

What is the result?

- A. Compilation fails due to error at //line2
- B. The program prints SQLException...
- C. The program prints each record
- D. Compilation fails at //line1

答案 C

考題 10

Given:

```
CREATE TABLE STORE (
    ID INTEGER,
    NAME VARCHAR(20),
    PRICE REAL,
    QUANTITY INTEGER
);

INSERT INTO STORE VALUES (1, 'Mug', 5.5, 25);
INSERT INTO STORE VALUES (2, 'Notebook', 7.25, 30);
INSERT INTO STORE VALUES (3, 'Towel', 10.75, 35);
INSERT INTO STORE VALUES (4, 'Wallet', 9.50, 15);
INSERT INTO STORE VALUES (5, 'Chocolate Bar', 1.50, 45);
```

And:

```
class ColumnFilter implements Predicate {
    float min;
    float max;
    public ColumnFilter(float n, float x) {
        this.min = n;
        this.max = x;
    }
    public boolean evaluate(RowSet rs) {
        CachedRowSet crs = (CachedRowSet) rs;
        try {
            float columnValue = crs.getFloat(3);
            if ((columnValue >= this.min) && (columnValue <= this.max)) {
                return true;
            }
        }
        return false;
    }
}
```

```
        } catch (Exception e) {
            e.printStackTrace();
            return false;
        }
    }
// other methods
}
```

And:

```
public static void main(String[] args) throws SQLException {
    ColumnFilter filter = new ColumnFilter(5.5f, 10.5f);
    RowSetFactory factory = RowSetProvider.newFactory();
    FilteredRowSet frs = factory.createFilteredRowSet();
    frs.setUrl("jdbc:derby://localhost:1527/myDB");
    frs.setUsername("root");
    frs.setPassword("sa");
    frs.setCommand("SELECT * FROM STORE");
    frs.setFilter(filter);
    frs.execute();
    while (frs.next()) {
        int empID = frs.getInt("ID");
        System.out.print(empID + ", ");
    }
}
```

What records with below IDs will exist after filter?

- A. 1, 2, 3, 4, 5,
- B. 1, 3, 4, 5,
- C. 1, 2, 4,
- D. nothing
- E. Compile failed

答案 C

說明 過濾所有資料的第三個欄位值，必須介於 5.5f ~ 10.5f 之間。故只有第 ID = 1, 2, 4 的資料符合。

考題 11

The advantage of a CallableStatement over a PreparedStatement is:

- A. Is easier to construct
- B. Supports transactions
- C. Runs on the database
- D. Uses Java instead of native SQL

答案 C

說明 CallableStatement 執行資料庫裡的預存程序 (Stored Procedure) , PreparedStatement 則執行 SQL。選項 A：必須撰寫預存程序 (Stored Procedure) ，並不輕鬆。選項 B：兩者都有交易機制。選項 C：預存程序 (Stored Procedure) 在資料庫裡執行，並回傳結果。選項 D：使用 native SQL。

考題 12

What design pattern does the DriverManager.getConnection () method characterize?

- A. DAO
- B. Factory
- C. Singleton
- D. Composition

答案 B

考題 13

Given:

```
class Test {
    static Connection newConnection = null;
    public static Connection getDBConnection() throws SQLException {
        String URL = "...";
        String username = "...";
        String password = "...";
        try (Connection con =
            DriverManager.getConnection(URL, username, password)) {
            newConnection = con;
        }
    }
}
```

```
        }
        return newConnection;
    }

    public static void main(String[] args) throws SQLException {
        getDBConnection();
        Statement st = newConnection.createStatement();
        st.executeUpdate("INSERT INTO student VALUES (102, 'Kelvin')");
    }
}
```

Known that:

- The required database driver is configured in the classpath.
- The database could be accessed with the URL, username, and password.
- The SQL query is valid.
- The table structure is Student (id INTEGER, name VARCHAR).

What is the result?

- A. The program executes successfully and the STUDENT table is updated with one record.
- B. The program executes successfully and the STUDENT table is NOT updated with any record.
- C. A SQLException is thrown as runtime.
- D. A NullPointerException is thrown as runtime.

答案 C

說明 使用 try-with-resource 語法建立 Connection 物件並連線資料庫時，一旦離開 try 程式區塊，雖然 Connection 物件依然存在，不會拋出 NullPointerException；但資料庫連線已經被關閉，因此拋出 SQLException。

考題 14

Given:

```
public static void main(String[] args) {
    try {
        String url = "...";
        String username = "...";
        String password = "...";
```

```

Connection connection =
    DriverManager.getConnection(url, username, password);
Statement statement = connection.createStatement(
    ResultSet.TYPE_SCROLL_INSENSITIVE,
    ResultSet.CONCUR_UPDATABLE);
statement.execute("SELECT * FROM Student");
ResultSet rs = statement.getResultSet();
while (rs.next()) {
    if (rs.getInt(1) == 2) {
        rs.updateString(2, "Jim");
    }
}
rs.absolute(2);
System.out.println(rs.getInt(1) + " " + rs.getString(2));
} catch (SQLException ex) {
    ex.printStackTrace();
}
}
}

```

Known that:

- The required database driver is configured in the classpath.
- The database could be accessed with the url, username, and password.
- The table structure is Student (id INTEGER, name VARCHAR), and there are 2 records in this table :

id=1, name=Jack

id=2, name=Duke

What is the result?

A. The Employee table is updated with the row:

2 Jim

and the program prints:

2 Duke

B. The Employee table is updated with the row:

2 Jim

and the program prints:

2 Jim

C. The Employee table is not updated and the program prints:

2 Duke

D. Exception is thrown.

答案 C

說明 必須在 : rs.updateString(2, "Jim") 之後加上 : rs.updateRow() 才能真正更新資料庫資料。

考題 15

Which action can be used to load a database driver by using JDBC3.0?

- A. Add the driver class to the META-INF/services folder of the JAR file.
- B. Include the JDBC driver class in a jdbc.properties file.
- C. Use the java.lang.Class.forName method to load the driver class.
- D. Use the DriverManager.getDriver method to load the driver class.

答案 C

說明 A 是 JDBC 4.0 的功能。

考題 16

Which statement is true about the DriverManager class?

- A. It returns an instance of Connection.
- B. it executes SQL statements to the database.
- C. It only queries metadata of the database.
- D. it is written by different vendors for their specific database.

答案 A

說明

選項 A：使用 DriverManager 的 getConnection()。

選項 B：使用 Statement 的 executeQuery() 方法。

選項 C：使用 Connection 物件的 getMetaData() 方法取得 DatabaseMetaData 物件。

選項 D：和 Connection、Statement、ResultSet 不同，非 interface。

考題 17

Given:

```
public static void main(String[] args) throws SQLException {
    String url = "...";String user = "...";String pwd = "...";
    Connection connection = DriverManager.getConnection(url, user, pwd);
    String query = "SELECT id FROM Employee";
    try (Statement statement = connection.createStatement()) {
        ResultSet rs = statement.executeQuery(query);
        statement.executeQuery("SELECT id FROM Student "); //line1
        while (rs.next()) {
            System.out.println("ID: " + rs.getInt("id"));
        }
    } catch (Exception e) {
        e.printStackTrace();
        System.out.println ("Error");
    }
}
```

Known that:

- The required database driver is configured in the classpath.
- The database could be accessed with the url, username, and password.
- Employee & Student tables have the same structure: (id INTEGER, name VARCHAR),

What is the result of compiling and executing this code fragment?

- A. The program prints employee IDs.
- B. The program prints student IDs.
- C. The program throws Exception.
- D. Compilation error on //line1.

答案 C

說明 Statement 物件在執行 executeQuery(sql) 後可取得 ResultSet 物件，但若再使用相同 Statement 執行另一查詢，將導致原來已開啟的 ResultSet 被關閉，因此使用 ResultSet 將拋出例外及錯誤訊息：java.sql.SQLException: ResultSet not open。

考題 18

Given:

```
public static void main(String[] args) {  
    String url = "....";  
    String username = "....";  
    String pwd = "....";  
    try {  
        Connection conn =  
            DriverManager.getConnection(url, username, pwd);  
        String query = "Select * FROM Item WHERE ID = 111";  
        Statement stmt = conn.createStatement();  
        ResultSet rs = stmt.executeQuery(query);  
        while (rs.next()) {  
            System.out.println("ID:" + rs.getInt("Id"));  
            System.out.println("Description:" + rs.getString("Description"));  
            System.out.println("Price:" + rs.getDouble("Price"));  
            System.out.println("Quantity:" + rs.getInt("Quantity"));  
        }  
    } catch (SQLException e) {  
        System.out.println("Exception");  
    }  
}
```

Known that:

- The required database driver is configured in the classpath.
- The database could be accessed with the url, username, and password.
- Item table

ID, INTEGER: PK

DESCRIPTION, VARCHAR(100)

PRICE, REAL

QUANTITY, INTEGER

What is the result?

- A. An exception is thrown at runtime.
- B. Compilation fails.
- C. The code prints Exception.
- D. The code prints information of Item 111.

答案 D

考題 19

Choose three objects which a vendor must implement in its JDBC driver?

- A. Time
- B. Date
- C. Statement
- D. ResultSet
- E. Connection
- F. SQLException
- G. DriverManager

答案 CDE

Java的區域化 (Localization)

-
- | 13.1 了解Java 的軟體區域化做法
 - | 13.2 使用DateFormat 類別
 - | 13.3 使用NumberFormat 類別
 - | 13.4 認證考試命題範圍

13.1 了解Java的軟體區域化做法

怎麼全世界都在談論「國際化(internationalization)」趨勢的時候，Java 却在討論「區域化(Localization)」？！事實上，這是一個一體兩面的議題。當程式設計師要將自己的軟體推廣到全球的時候，勢必會遇到不同地域的使用者，如何提供當地的語言就成了必須解決的問題。Java 在最初就掌握了www的趨勢，自然是胸有成竹，本章將介紹Java 對區域化軟體的支援做法。

Java 軟體的「區域化(localization)」是藉由增加和「特定地區\地域」相關的元件和翻譯文字，使軟體可以呈現「特定地區\地域」的語言文字，及日期、數字、幣別等與文化相關的特殊格式。

Java 要滿足「軟體區域化」和「支援多國語系」的需要，不是藉由複製程式碼後修改和文字呈現相關程式碼，如此會讓程式碼愈來愈多份，違背了DRY法則；而是事先準備多份各國語系的文字檔，依需求載入JVM，再嵌入(plug-in)文字呈現的畫面或功能中。要達到這樣的目標，需要有三個核心元件：

1. Locale 類別，用來代表特定地區\地域。
2. 多國語系的文字檔(或稱資源綁定檔案)，用來存放各國文字，檔案各自獨立。
3. ResourceBundle 類別，用來對應多國語系的文字檔。建立物件時，檔案內容自動載入到物件裡。

13.1.1 使用 Locale 類別

Java 使用 Locale 這個單字(中文解釋為場所、場域)，卻不用國別決定不同的語言，主要考量是有些國家幅員廣大，不一定只有單一語言。因此用 Locale 類別來代表特定「語言(language)」和「國家(country)」的組合。規則是：

1. 語言(language)

- 使用 alpha-2 或 alpha-3 ISO 639 code。
- 小寫，如：**de** 代表 German，**en** 代表 English，**fr** 代表 French，**zh** 代表 Chinese。

2. 國家(country)

- 使用 ISO 3166 alpha-2 country code 或 UN M.49 numeric area code。
- 大寫，如**DE** 代表 Germany，**US** 代表 United States，**FR** 代表 France，**CN** 代表 China。

建構 Locale 物件的常見方式有二種：

1. 使用 Locale 類別已經定義的常數。
2. 提供 language 和 country 的代碼字串作為建構子的輸入參數。

如以下兩者同義：

```
Locale twLocale1 = Locale.TAIWAN;
Locale twLocale2 = new Locale("zh", "TW");
```

13.1.2 建立多國語系文字檔

多國語系的文字檔，或稱資源綁定檔案 (resource bundle files)，製作方式是：

1. 以「.properties」做副檔名。
2. 針對程式需要支援的每種語系建立獨立檔案。每個檔案的「主要檔名」相同，再加上「語言」和「國家」代碼做區隔，亦即需要對系統會使用到的 locale 建立對應的檔案。若檔案上全無「語言」和「國家」代碼，則為預設檔，作為程式找不到對應多國語系文字檔時的最後防線。
3. 檔案內包含許多成對的 key 和 value。每個檔案的 key 的數量及內容都一致，因為會被使用於程式碼中；value 則為各 locale 的當地文字。

以範例「/OCP/src/course/c13/LocaleTest.java」中使用的多國語系文字檔為例。其主要檔名為「MessageBundle」，因此預設檔案是「MessageBundle.properties」，再依需求建立了多個文字檔，檔案命名方式為：

```
MessageBundle_xx_YY.properties
xx : language 代碼，小寫
YY : country 代碼，大寫
```

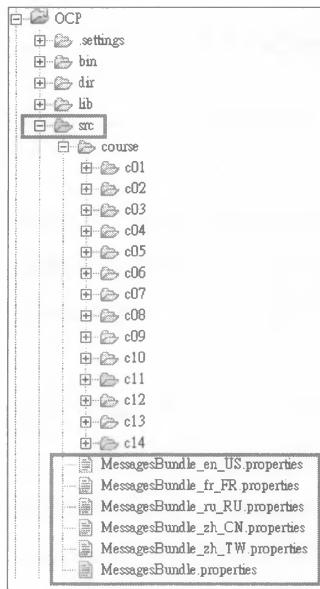
檔名和內容可以是：

表 13-1 多國語系文字檔

MessageBundle.properties	MessagesBundle_zh_CN.properties
menu1 = Set to English	menu1 = 设置为英语
menu2 = Set to French	menu2 = 设置为法语
menu3 = Set to Chinese	menu3 = 设置为中文
menu4 = Set to Russian	menu4 = 设置为俄罗斯
menu5 = Show the Date	menu5 = 显示日期
menu6 = Show the money	menu6 = 显示钱！
menuq = Enter q to quit	menuq = 输入 q 退出

MessagesBundle_zh_TW.properties	MessagesBundle_fr_FR.properties
menu1 = 設定成英文	menu1 = Régler à l'anglais
menu2 = 設定成法文	menu2 = Régler au français
menu3 = 設定成中文	menu3 = Réglez chinoise
menu4 = 設定成俄文	menu4 = Définir pour la Russie
menu5 = 顯示日期	menu5 = Afficher la date
menu6 = 顯示金額	menu6 = Montrez-moi l'argent!
menuq = 輸入 q 退出	menuq = Saisissez q pour quitter

檔案位置可以在 Eclipse 專案的「src」路徑下：



❖ 圖 13-1 多國語系文字檔位置

13.1.3 使用 ResourceBundle 類別

ResourceBundle 類別的名稱由兩個複合字組成，分別是 resource(資源) 和 bundle(綁定)。使用該類別，顧名思義可以綁定某個資源，而這個資源就是前述的「多國語系文字檔」，所以我們也稱它「資源綁定檔案 (resource bundle files)」。事實上，資源還可以是 class 檔，只是一般較少使用。

以該類別建立物件時，必須提供：

1. 多國語系文字檔的主要檔名，如前述的「MessageBundle」。
2. Locale 物件，代表某一「語言」和「國家」的組合，如 zh 和 TW 。

這兩個資料，已經清楚暗示要「綁定」的檔案為「MessagesBundle_zh_TW.properties」。

```
Locale twLocale = new Locale("zh", "TW");
ResourceBundle bundle =
    ResourceBundle.getBundle("MessagesBundle", twLocale);
```

建立 ResourceBundle 物件時，Java 將自動「載入」對應的多國語系文字檔的內容到物件裡。

綜合範例為「/OCP/src/course/c13/LocaleTest.java」：

範例

```
1  public class LocaleTest {
2      public static void main(String[] args) {
3          Locale usLocale = Locale.US;
4          Locale frLocale = Locale.FRANCE;
5          Locale zhLocale = new Locale("zh", "CN");
6          Locale twLocale = new Locale("zh", "TW");
7          Locale ruLocale = new Locale("ru", "RU");
8          Locale defaultLocale = Locale.getDefault();
9          Locale itLocale = Locale.ITALY;      // 不存在檔案
10         Locale koLocale = Locale.KOREAN;    // 不存在檔案
```

```
11
12     List<Locale> locales = new ArrayList<Locale>();
13     locales.add(usLocale);
14     locales.add(frLocale);
15     locales.add(zhLocale);
16     locales.add(twLocale);
17     locales.add(ruLocale);
18     locales.add(defaultLocale);
19     locales.add(itLocale);
20     locales.add(koLocale);
21
22     for (Locale l: locales) {
23         showLocaleValue(l, "menu1");
24     }
25 }
26 private static void showLocaleValue(Locale locale, String key) {
27     System.out.print(locale + ": ");
28     ResourceBundle bundle =
29             ResourceBundle.getBundle("MessagesBundle", locale);
30     System.out.println(bundle.getLocale() + ": " + bundle.getString(key));
31 }
```

說明

3	對應 <code>MessagesBundle_en_US.properties</code> 。
4	對應 <code>MessagesBundle_fr_FR.properties</code> 。
5	對應 <code>MessagesBundle_zh_CN.properties</code> 。
6	對應 <code>MessagesBundle_zh_TW.properties</code> 。
7	對應 <code>MessagesBundle_ru_RU.properties</code> 。
8	依目前所在地的 locale，台灣為 <code>MessagesBundle_zh_TW.properties</code> 。
9	無對應多國語系檔案。
10	無對應多國語系檔案。
28	<p>對應的 ResourceBundle 檔案若不存在：</p> <ol style="list-style-type: none"> 先用 <code>MessagesBundle_zh_TW.properties</code>，亦即 default locale 對應的多國語系檔案。 再找 <code>MessagesBundle.properties</code>，亦即預設的多國語系檔案。 還是找不到檔案就會出錯： <code>java.util.MissingResourceException:</code> <code>Can't find bundle for base name MessagesBundle, ...</code>

結果

```

en_US: en_US: Set to English
fr_FR: fr_FR: Positionner sur Anglais
zh_CN: zh_CN: 設置成英文
zh_TW: zh_TW: 設定成英文
ru_RU: ru_RU: Установить английский
zh_TW: zh_TW: 設定成英文
it_IT: zh_TW: 設定成英文
ko: zh_TW: 設定成英文

```

13.2 使用DateFormat類別

Java 可以使用 `DateFormat` 類別搭配 `Locale` 物件以提供日期的區域化顯示。步驟為：

1. 取得 `java.util.Date` 日期物件。
2. 搭配 `Locale` 取得 `DateFormat` 物件，並挑選格式。
3. 呼叫 `DateFormat` 物件的 `format()` 方法，並傳入 `java.util.Date` 日期物件。

日期的格式選項

日期的格式選項可以是：

1. 由 DateFormat 類別提供的常數來指定：

- **SHORT**：如「12.13.52」或「3:30 pm」。
- **MEDIUM**：如「Jan 12, 1952」。
- **LONG**：如「January 12, 1952」或「3:30:32 pm」。
- **FULL**：如「Tuesday, April 12, 1952 AD」或「3:30:42 pm PST」。

2. 由 SimpleDateFormat 類別 (為 DateFormat 子類別) 指定特定格式：

- yyyy/MM/dd HH:mm:ss
- yyyy/MMM/dd HH:mm:ss
- yyyy/MMMM/dd HH:mm:ss

範例「/OCP/src/course/c13/FormatDate.java」示範了上述兩種方式的使用：

範例

```

1  public class FormatDate {
2      public static void useDateFormat() {
3          System.out.println("===== UseDateFormat ()");
4          Date today = new Date();
5          Locale locale = Locale.US;
6          DateFormat df;
7          df = DateFormat.getDateInstance(DateFormat.DEFAULT, locale);
8          System.out.println("DateFormat.DEFAULT: " + df.format(today));
9          df = DateFormat.getDateInstance(DateFormat.SHORT, locale);
10         System.out.println("DateFormat.SHORT: " + df.format(today));
11         df = DateFormat.getDateInstance(DateFormat.MEDIUM, locale);
12         System.out.println("DateFormat.MEDIUM: " + df.format(today));
13         df = DateFormat.getDateInstance(DateFormat.LONG, locale);
14         System.out.println("DateFormat.LONG: " + df.format(today));
15         df = DateFormat.getDateInstance(DateFormat.FULL, locale);
16         System.out.println("DateFormat.FULL: " + df.format(today));
17     }
18     public static void useSimpleDateFormat() {
19         System.out.println("===== UseSimpleDateFormat ()");
20         Date today = new Date();
21         Locale locale = Locale.US;
22         DateFormat df;
23         df = new SimpleDateFormat("yyyy/MM/dd HH:mm:ss", locale);
24         System.out.println(df.format(today));
25         df = new SimpleDateFormat("yyyy/MMM/dd HH:mm:ss", locale);
26         System.out.println(df.format(today));

```

```

27     df = new SimpleDateFormat("yyyy/MMM/d HH:mm:ss", locale);
28     System.out.println(df.format(today));
29 }
30 public static void main(String[] args) {
31     useDateFormat();
32     useSimpleDateFormat();
33 }
34 }
```

結果

```

===== UseDateFormat()
DateFormat.DEFAULT: Jun 22, 2016
DateFormat.SHORT: 6/22/16
DateFormat.MEDIUM: Jun 22, 2016
DateFormat.LONG: June 22, 2016
DateFormat.FULL: Wednesday, June 22, 2016
=====
===== UseSimpleDateFormat()
2016/06/22 09:48:52
2016/Jun/22 09:48:52
2016/June/22 09:48:52
```

13.3 使用NumberFormat類別

Java 可以使用 NumberFormat 類別搭配 Locale 物件以提供幣別的區域化顯示。步驟為：

1. 以 Locale 物件傳入 NumberFormat 類別的 static 工廠方法中，以取得幣別的物件實例。
2. 呼叫幣別物件實例的 format() 方法，並傳入數字金額，可為基本型別或其包裹類別。

範例「/OCP/src/course/c13/FormatCurrency.java」顯示使用方式：

範例

```

1  public class FormatCurrency {
2      public static void main(String[] args) {
3          NumberFormat numberFormat;
4          numberFormat = NumberFormat.getCurrencyInstance(Locale.US);
5          System.out.println("Locale.US: " + numberFormat.format(1000000));
6          numberFormat = NumberFormat.getCurrencyInstance(Locale.TAIWAN);
7          System.out.println("Locale.TAIWAN: "
8              + numberFormat.format(new Double(1000000.00)));
9          numberFormat = NumberFormat.getCurrencyInstance(Locale.JAPAN);
```

```
9     System.out.println("Locale.US: " + numberFormat.format(1000000.00));  
10    }  
11 }
```

結果

```
Locale.US: $1,000,000.00  
Locale.TAIWAN: NT$1,000,000.00  
Locale.JAPAN: 1,000,000
```

13.4 認證考試命題範圍

依據甲骨文公布的考試範圍 (https://education.oracle.com/java-se-8-programmer-ii/pexam_1Z0-809)，和本章相關的主題有：

Localization

1. Read and set the locale by using the **Locale** object.
2. Create and read a **Properties** file.
3. Build a **resource bundle** for each locale and load a resource bundle in an application.

本章擬真試題實戰

考題 1

Suppose you got an exception as running application:

Exception in thread "main" java.util.MissingResourceException:

Can't find bundle for base name MessageBundle, locale XXX

And given that the MessageBundle.properties file has been created, exists on your disk, and is properly formatted.

What is the cause of the error message?

- A. The file is not in the environment path.
- B. The file is not in the classpath.
- C. The file is not in the javapath.
- D. You cannot use a file to store a ResourceBundle.

答案 B

考題 2

Which two statements correctly create a Locale variable?

- A. Locale loc1 = "UK";
- B. Locale loc2 = Locale.getInstance("ru");
- C. Locale loc3 = Locale.getLocaleFactory("RU");
- D. Locale loc4 = Locale.UK;
- E. Locale loc5 = new Locale("ru", "RU");

答案 DE

考題 3

Which option displays the three-character month abbreviation?

- A.

```
SimpleDateFormat sdfA = new SimpleDateFormat("mm", Locale.UK);
System.out.println("Result:" + sdfA.format(new Date()));
```
- B.

```
SimpleDateFormat sdfB = new SimpleDateFormat("MM", Locale.UK);
System.out.println("Result:" + sdfB.format(new Date()));
```
- C.

```
SimpleDateFormat sdfC = new SimpleDateFormat("MMM", Locale.UK);
System.out.println("Result:" + sdfC.format(new Date()));
```
- D.

```
SimpleDateFormat sdfD = new SimpleDateFormat("MMMM", Locale.UK);
System.out.println("Result:" + sdfD.format(new Date()));
```

答案 C

考題 4

Given three resources bundles with these values set for menu1: (the default resource bundle in US English.)

1. United States Resource Bundle

```
menu1 = Set to English
```

2. French Resource Bundle

```
menu1 = Positionner sur Anglais
```

3. Russia Resource Bundle

```
menu1 =
```

And Given:

```
public static void main(String[] args) {
    Locale.setDefault(new Locale("es", "ES"));
    // Set default to Spanish and Spain
    Locale l = Locale.getDefault();
    ResourceBundle msg = ResourceBundle.getBundle("MessagesBundle", l);
    System.out.println(msg.getString("menu1"));
}
```

What is the result?

- A. No message is printed
- B. Positionner sur Anglais
- C. Set to English
- D. A runtime error is produced

答案 D

說明 預設 Locale 已經被改為 Spanish and Spain，找不到檔案時將拋出：

```
Exception in thread "main" java.util.MissingResourceException:  
  Can't find bundle for base name MessagesBundle, locale es_ES
```

考題 5

Which is a factory method from the `java.text.NumberFormat` class?

- A. `format (long number)`
- B. `getInstance()`
- C. `getMaxiraumFractionDigits ()`
- D. `getAvailableLocales ()`
- E. `isGroupingUsed()`

答案 B

考題 6

Select 4 options to create a `NumberFormat` instance using a factory method:

- A. `NumberFormat nf1 = new DecimalFormat();`
- B. `NumberFormat nf2 = new DecimalFormat("0.00");`
- C. `NumberFormat nf3 = NumberFormat.getInstance();`
- D. `NumberFormat nf4 = NumberFormat.getIntegerInstance();`
- E. `NumberFormat nf5 = DecimalFormat.getNumberInstance();`
- F. `NumberFormat nf6 = NumberFormat.getCurrencyInstance();`

答案 CDEF

考題 7

Given the ocpjp.properties file containing:

```
HELLO_MSG=Hello, everyone!
GOODBYE_MSG=Goodbye everyone!
```

And:

```
public static void main(String[] args) {
    ResourceBundle rb = ResourceBundle.getBundle("ocpjp", Locale.US);
    System.out.print(rb.getObject(1));
}
```

What is the result?

- A. Compilation fails
- B. HELLO_MSG
- C. GOODBYE_MSG
- D. Hello, everyone!
- E. Goodbye everyone!

答案 A

說明 應該改為：System.out.print(rb.getString("HELLO_MSG"));

考題 8

Which statement defines a new DateFormat object that displays the default date format for the UK Locale?

- A. DateFormat dfA =


```
DateFormat.getDateInstance(DateFormat.DEFAULT, Locale.UK);
```
- B. DateFormat dfB =


```
DateFormat.getDateInstance(DateFormat.DEFAULT, UK);
```
- C. DateFormat dfC =


```
DateFormat.getDateInstance(DateFormat.DEFAULT, Locale.UK);
```

```
D. DateFormat dfD =  
    new DateFormat.getDateInstance(DateFormat.DEFAULT, Locale.UK);  
  
E. DateFormat dfE =  
    new DateFormat.getDateInstance(DateFormat.DEFAULT, Locale(UK));
```

答案 C

說明 A 和 B 和 E : Locale 表示方式錯誤。D 和 E : 使用工廠方法，不該使用 new 關鍵字。

考題 9

Which two codes correctly represent a standard language locale code?

- A. ES
- B. FR
- C. U8
- D. es
- E. fr
- F. u8

答案 DE

說明 必須是小寫，常用的有：de:German、en:English、fr:French、ru:Russian、ja:Japanese、ko:Korean、zh:Chinese。

考題 10

Given a language code of "fr" and a country code of "FR", which file name represents a resource bundle file?

- A. MessageBundle_fr_FR.properties
- B. MessageBundle_fr_FR.profile
- C. MessageBundle_fr_FR.java
- D. MessageBundle_fr_FR.locale

答案 A

考題 11

Given:

```
public static void main(String[] args) {
    SimpleDateFormat sdf = new SimpleDateFormat("zzzz", Locale.US);
    System.out.println(sdf.format(new Date()));
}
```

What type of result is printed?

- A. Time zone abbreviation
- B. Full-text time zone name
- C. Era
- D. Julian date
- E. Time of the Epoch (in milliseconds)

答案 A

說明 會印出時區名稱。若本例在台北執行，會印出「China Standard Time」。

考題 12

Which two options are true about localization?

- A. Support for new regional languages does not require recompilation of the code.
- B. Textual elements (messages and labels) are hard-coded in the source code.
- C. Language and region-specific programs are created using localized data.
- D. Resource bundle files include data and currency information.
- E. Language codes use lowercase letters and region codes use uppercase letters.

答案 AE

說明 B、C：程式碼裡使用「Key」，Value 在對應的語系檔案裡。E：如 en_US 或 zh_TW。

考題 13

If creating a ResourceBundle with a properties file to localize an application, which code example specifies valid keys of "menu1" and "menu2" with values of "File Menu" and "View Menu"?

- A. <key name="menu1">File Menu</key>
<key name="menu2">View Menu</key>
- B. <key>menu1</key><value>File Menu</value>
<key>menu2</key><value>View Menu</value>
- C. menu1, File Menu, menu2, View Menu
- D. menu1=File Menu
menu2=View Menu

答案 D

Interfaces與lambda 表示式的應用

-
- | 14.1 使用interface 的static 和default 方法
 - | 14.2 使用lambda 表示式
 - | 14.3 使用基礎的內建functional interfaces
 - | 14.4 在泛型內使用萬元字元
 - | 14.5 使用進階的內建functional interfaces
 - | 14.6 使用方法參照
 - | 14.7 認證考試命題範圍

14.1 使用interface的static和default方法

interfaces 再回顧

Java 使用 interface 作為抽象型別的一種表達方式：

1. 可以作為參考型別，是許多設計模式 (design patterns) 的核心。
2. 類似 abstract class，所有方法必須被 class 實作，被視為類別的合約 (contract)。因為一旦宣告實作 interface，等同合約締成，就有提供所有抽象方法內容的履約義務。
3. 方法只可以是 public 且 abstract。若未加，Java 預設補上。
4. 欄位只可以是 public、static 且 final。若未加，Java 預設補上。

假設有一家五金公司販售差異性極高的幾種商品，但又希望在財務報表上有一致的呈現方式。它的商品包含：

- 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)。

原始的類別設計都放在套件「course.c14.asis」下，分別為：

範例

```

1  public class Rock {
2      private String name = this.getClass().getSimpleName();
3      private double unitPrice;
4      private double unitCost;
5      private double weight;
6      public Rock(double unitPrice, double unitCost, double weight) {
7          this.unitPrice = unitPrice;
8          this.unitCost = unitCost;
9          this.weight = weight;
10     }
11 }
```

範例

```

1  public class Paint {
2      private String name = this.getClass().getSimpleName();
3      private double unitPrice;
4      private double unitCost;
```

```

5     private double volume;
6     public Paint(double unitPrice, double unitCost, double volume) {
7         this.unitPrice = unitPrice;
8         this.unitCost = unitCost;
9         this.volume = volume;
10    }
11 }
```

範例

```

1  public class Widget {
2      private String name = this.getClass().getSimpleName();
3      private double unitPrice;
4      private double unitCost;
5      private double quantity;
6      public Widget(double unitPrice, double unitCost, double quantity) {
7          this.unitPrice = unitPrice;
8          this.unitCost = unitCost;
9          this.quantity = quantity;
10     }
11 }
```

五金公司希望它的財務報表可以綜合所有商品後呈現：

- 售價 (Sales Price)。
- 成本 (Cost)。
- 利潤 (Profit)。

因為這個需求，我們設計了介面 ReportAble，載明所有商品若需要在財務表報上出現，所需要提供的數字：

範例

```

1  public interface ReportAble {
2      public String getName();
3      public double getTotalPrice();
4      public double getTotalCost();
5      public double getProfit();
6  }
```

接下來，我們陸續讓三種商品實作介面 ReportAble。以「/OCP/src/course/c14/tobe/Rock.java」為例：

範例

```

1  public class Rock implements ReportAble {
2      private String name = this.getClass().getSimpleName();
3      private double unitPrice;
4      private double unitCost;
5      private double weight;
6      public Rock(double unitPrice, double unitCost, double weight) {
7          this.unitPrice = unitPrice;
8          this.unitCost = unitCost;
9          this.weight = weight;
10     }
11     @Override
12     public String getName() {
13         return this.name;
14     }
15     @Override
16     public double getTotalPrice() {
17         return this.weight * this.unitPrice;
18     }
19     @Override
20     public double getTotalCost() {
21         return this.weight * this.unitCost;
22     }
23     @Override
24     public double getProfit() {
25         return getTotalPrice() - getTotalCost();
26     }
27 }
```

「/OCP/src/course/c14/tobe/Paint.java」和「/OCP/src/course/c14/tobe/Widget.java」也是比照辦理。最後建立一個工具型 (utility) 的類別 ReportProduct，專司產出商品報表。經由這樣的過程可以發現，公司所有商品雖然不同，但若全部實作 ReportAble 介面，就可以提供一致的報表內容；而且可以只用一種 ReportAble 作為全部商品的參考型別：

範例

```

1  public class ReportProduct {
2      public ReportProduct() {
3          System.out.println("** Sales Report **");
4          System.out.println("Price\tCost\tProfit\t");
5          System.out.println("=====");
6      }
7      public void show(ReportAble item) {
```

```

8      System.out.println(item.getTotalPrice() +
9          "\t" + item.getTotalCost() +
10         "\t" + item.getTotalPrice());
11    }
12  public static void main(String args[]) {
13      ReportAble[] itemList = new ReportAble[5];
14      ReportProduct report = new ReportProduct();
15      itemList[0] = new Rock(15.0, 10.0, 50.0);
16      itemList[1] = new Rock(11.0, 6.0, 10.0);
17      itemList[2] = new Paint(13.0, 8.0, 25.0);
18      itemList[3] = new Widget(7.0, 5.0, 10);
19      itemList[4] = new Widget(12.0, 12.0, 20);
20      for (ReportAble item : itemList) {
21          report.show(item);
22      }
23  }
24 }
```

結果

```

** Sales Report **
Name      Price     Cost     Profit
=====
Rock      750.0    500.0    750.0
Rock      110.0    60.0     110.0
Paint     325.0    200.0    325.0
Widget    70.0     50.0     70.0
Widget    240.0    240.0    240.0
```

使用介面的 default 和 static 方法

回顧先前例子。鑑於單一責任制(SRP)法則，新增 ReportProduct 工具類別，只用來處理有實作 ReportAble 的商品，但感到有些「勞師動眾」。是否有機會將唯一的方法移至介面 ReportAble 裡？這樣依然不違背 SRP 法則，因為該介面只是集合了所有和列印報表相關的方法。

在 Java 8 之前，這個答案是否定的，因為介面裡只能放「沒有內容的抽象方法」。但由 Java 8 開始，在介面裡推出「default 方法」來達成類似需求！它的規則是：

1. 使用 default 關鍵字宣告。
2. 可以擁有實作內容，但非 default 的方法還是一樣不能有內容。
3. 可被有實作該介面的子類別使用。

4. 因為 `default` 的方法已經有內容，因此不會強制子類別都必須實作該方法；就算後來修改介面才加入 `default` 方法也不會影響已實作該介面的子類別。

除了 `default` 的方法之外，Java 8 也允許在介面裡建立「`static` 方法」，同樣允許有實作內容！使用這種 `static` 方法就和使用類別的 `static` 方式相同。

綜合以上 `default` 和 `static` 方法，我們修改介面 `ReportAble`，如範例「/OCP/src/course/c14/tobe/ReportAble.java」：

範例

```

1  public interface ReportAble {
2
3      public String getName();
4      public double getTotalPrice();
5      public double getTotalCost();
6      public double getProfit();
7
8      public default void show() {
9          System.out.println(this.getTotalPrice() +
10                 "\t" + this.getTotalCost() +
11                 "\t" + this.getTotalPrice());
12     }
13
14     public static void report (ReportAble item) {
15         System.out.println(item.getTotalPrice() +
16                 "\t" + item.getTotalCost() +
17                 "\t" + item.getTotalPrice());
18     }
19 }
```

並使用範例「/OCP/src/course/c14/tobe/TestReportAble.java」進行測試：

範例

```

1  public class TestReportAble {
2      private static void testDefault() {
3          ReportAble[] itemList = new ReportAble[5];
4          itemList[0] = new Rock(15.0, 10.0, 50.0);
5          itemList[1] = new Rock(11.0, 6.0, 10.0);
6          itemList[2] = new Paint(13.0, 8.0, 25.0);
7          itemList[3] = new Widget(7.0, 5.0, 10);
8          itemList[4] = new Widget(12.0, 12.0, 20);
9          System.out.println("** Sales Report from default **");
10         System.out.println("Name\tPrice\tCost\tProfit\t");

```

```

11         System.out.println("=====");
12         for (ReportAble item : itemList) {
13             item.show();
14         }
15     }
16     private static void testStatic() {
17         ReportAble[] itemList = new ReportAble[5];
18         itemList[0] = new Rock(15.0, 10.0, 50.0);
19         itemList[1] = new Rock(11.0, 6.0, 10.0);
20         itemList[2] = new Paint(13.0, 8.0, 25.0);
21         itemList[3] = new Widget(7.0, 5.0, 10);
22         itemList[4] = new Widget(12.0, 12.0, 20);
23         System.out.println("** Sales Report from static **");
24         System.out.println("Name\tPrice\tCost\tProfit\t");
25         System.out.println("=====");
26         for (ReportAble item : itemList) {
27             ReportAble.report(item);
28         }
29     }
30     public static void main(String args[]) {
31         testDefault();
32         System.out.println();
33         testStatic();
34     }
35 }
```

結果

** Sales Report from default **

Name	Price	Cost	Profit
Rock	750.0	500.0	750.0
Rock	110.0	60.0	110.0
Paint	325.0	200.0	325.0
Widget	70.0	50.0	70.0
Widget	240.0	240.0	240.0

** Sales Report from static **

Name	Price	Cost	Profit
Rock	750.0	500.0	750.0
Rock	110.0	60.0	110.0
Paint	325.0	200.0	325.0
Widget	70.0	50.0	70.0
Widget	240.0	240.0	240.0

另外，interface 提供有內容的方法，讓人聯想到 Java 不支援多繼承的理由之一就是擔心不同的父類別可能存在相同簽名的方法，讓子類別混淆而無所適從。現在故意建立一個具備相同簽名方法的父類別「/OCP/src/course/c14/tobe/ClassWithShow.java」：

範例

```

1 public abstract class ClassWithShow {
2     public abstract String getName();
3     public void show() {
4         System.out.println(this.getName() + "\t" + "-----");
5     }
6 }
```

建立類別「/OCP/src/course/c14/tobe/Rock2.java」，同時繼承類別 ClassWithShow 並實作 Reportable，則 Rock2 可以自父類別和介面得到相同簽名的方法「void show()」。所以我們關心的問題是：

1. 可否通過編譯？
2. 執行結果為何？

範例

```

1 public class Rock2 extends ClassWithShow implements Reportable {
2     private String name = this.getClass().getSimpleName();
3     private double unitPrice;
4     private double unitCost;
5     private double weight;
6     public Rock2(double unitPrice, double unitCost, double weight) {
7         this.unitPrice = unitPrice;
8         this.unitCost = unitCost;
9         this.weight = weight;
10    }
11    @Override
12    public String getName() {
13        return this.name;
14    }
15    @Override
16    public double getTotalPrice() {
17        return this.weight * this.unitPrice;
18    }
19    @Override
20    public double getTotalCost() {
21        return this.weight * this.unitCost;
```

```

22     }
23     @Override
24     public double getProfit() {
25         return getTotalPrice() - getTotalCost();
26     }
27 }
```

使用以下方法進行測試：

範例

```

1 private static void testDefault2() {
2     ReportAble[] itemList = new ReportAble[5];
3     itemList[0] = new Rock2(15.0, 10.0, 50.0);
4     itemList[1] = new Rock2(11.0, 6.0, 10.0);
5     itemList[2] = new Paint(13.0, 8.0, 25.0);
6     itemList[3] = new Widget(7.0, 5.0, 10);
7     itemList[4] = new Widget(12.0, 12.0, 20);
8     System.out.println("** Sales Report from default2 **");
9     System.out.println("Name\tPrice\tCost\tProfit\t");
10    System.out.println("=====");
11    for (ReportAble item : itemList) {
12        item.show(item);
13    }
14 }
```

結果

```

** Sales Report from default2 **
Name      Price      Cost      Profit
=====
Rock2 -----
Rock2 -----
Paint    325.0    200.0    325.0
Widget   70.0     50.0     70.0
Widget   240.0    240.0    240.0
```

由上述執行結果可以驗證：

1. Java 8 之後，若父類別和介面有相同簽名的方法，且介面使用 `default` 壓告，並提供方法實作內容，則子類別還是可以同時繼承和實作。
2. 此時「父類別」的方法將有更高優先權。亦即繼承了來自類別和介面的相同簽名方法的子類別，將優先選用來自「父類別」的方法。

14.2 使用lambda表示式

我們在第四章中曾經介紹匿名內部類別 (Anonymous Inner Classes)，而其使用時機通常是：

1. 只使用一次，因此不需要特別定義類別，藉此減少程式碼的撰寫。
2. 希望把相關程式碼擺在同一地方。
3. 增加封裝程度。
4. 提高程式碼可讀性。

現在，我們使用 Java 8 推出的「functional interface」來複習一下匿名類別的用法，其特色為：

1. 只有一個方法的 interface。
2. 可以加上 annotation 「@FunctionalInterface」。

建立一個 functional interface，如範例「/OCP/src/course/c14/StringAnalyzer.java」：

範例

```

1  @FunctionalInterface
2  public interface StringAnalyzer {
3      public boolean analyze(String target, String keyStr);
4  }
```

並建立類別 ContainsAnalyzer 去實作該介面，如「/OCP/src/course/c14/ContainsAnalyzer.java」：

範例

```

1  public class ContainsAnalyzer implements StringAnalyzer {
2      public boolean analyze(String target, String keyStr) {
3          return target.contains(keyStr);
4      }
5  }
```

在這個實作中，我們要分析的是目標字串 (target) 裡，是否含有 (contains) 關鍵字串 (keyStr)。

進一步在類別 StringAnalyzerTest 中建立如下方法，輸入字串陣列、關鍵字串和 StringAnalyzer 的實作類別。StringAnalyzer 的實作類別會幫我們找出字串陣列裡符合分析結果的字串。如範例「/OCP/src/course/c14/StringAnalyzerTest.java」：

範例

```

1 static void searchArr(String[] strArr, String keyStr, StringAnalyzer analyzer) {
2     for (String str : strArr) {
3         if (analyzer.analyze(str, keyStr)) {
4             System.out.println(str);
5         }
6     }
7 }
```

在這樣的過程中，當然我們也可以不實作介面、拋棄多型，直接建立單一類別處理：

範例

```

1 public class StringAnalyzerTool {
2     public boolean contains(String target, String searchStr) {
3         return target.contains(searchStr);
4     }
5 }
```

差別在於，若字串分析的需求很多樣，目前雖然只是字串的包含(contains)，未來可能會有「是否由某字串開頭(startsWith)」、「是否以某字串結尾(endsWith)」等。則使用單一類別，就必須不斷修改程式以增加其他類似方法，因此違反物件導向的 OCP 法則(Open Close Principle)：程式設計歡迎擴充但拒絕修改(open for extension, close for modification)。

回到我們的主軸。在類別 StringAnalyzerTest 中新增 test1() 和 test() 兩個方法，比較是否使用「匿名內部類別(Anonymous Inner Classes)」的差異：

範例

```

1 static void test1() {
2     String[] strArr = { "abc", "bcd", "efg" };
3     searchArr(strArr, "b",
4             new ContainsAnalyzer()
5         );
6 }
7 static void test2() {
8     String[] strArr = { "abc", "bcd", "efg" };
9     searchArr(strArr, "b",
10            new StringAnalyzer() {
11                public boolean analyze(String target, String keyStr) {
12                    return target.contains(keyStr);
13                }
14            }
15        );
16 }
```

```

14     }
15   );
16 }
```

說明

4	使用一般類別建立實例。
10~14	使用匿名內部類別建立實例。

比較兩者後，可以發現使用「匿名內部類別」的兩個缺點：

1. 程式碼字元數目和一般類別相當，因此減少程式碼撰寫的效果有限；但寫法讓人感覺複雜。
2. 因為沒有名稱，編譯後的類別檔案 (*.class) 的命名方式如下。因為同一個外部類別內的匿名類別只用出現順序編號分辨，若使用太多，辨識來源將變得困難：

表 14-1 匿名內部類別之編譯檔案命名規則

名稱組成	外部類別名稱	\$	出現順序編號	.	class
	OuterClassName	\$	1	.	class

Java 8 開始，可以使用 lambda 表示式真正簡化程式碼的撰寫。lambda 表示式必須含三部分：

1. 方法參數 (argument list)。
2. 箭頭符號 (arrow token)，為「->」。
3. 方法內容 (body)。

以下為簡單示範：

```
(int x, int y) -> x + y
(x, y) -> x + y
(x, y) -> { system.out.println(x + y);}
(String s) -> s.contains("word")
s -> s.contains("word")
```

熟悉 lambda 表示式的語法是 OCAJP 的認證範圍，本書假設讀者已經了解，因此不再贅述。接下來僅由匿名內部類別的語法角度，說明 lambda 表示式的形成過程。

Lambda 表示式能夠減少程式碼的撰寫，關鍵在於「編譯器如何善用已知的資訊」，目的在於「用方法定義取代類別定義」。所以，簡化的步驟為：

STEP01 在呼叫 StringAnalyzerTest 的 searchArr() 方法時，編譯器已經知道第三個參數要求傳入 StringAnalyzer 的參照變數，所以程式碼撰寫的時候不需要強調「new StringAnalyzer() {}」，可以刪除：

範例

```

9  searchArr(strArr, "b",
10   _____ new StringAnalyzer() {
11       public boolean analyze (String target, String keyStr) {
12           return target.contains(keyStr);
13       }
14   _____
15 );

```

STEP02 因為是 functional interface，唯一的方法的定義相當清楚，所以方法的名稱、回傳、存取層級等都已經知道，因此「public boolean analyze {}」都可以刪除，只需要內容，和內容使用的變數與方法參數的對照關係：

範例

```

9  searchArr(strArr, "b",
11   _____ public boolean analyze (String target, String keyStr) {
12       return target.contains(keyStr) ;
13   _____
15 );

```

STEP03 修改一下語法，把「return」關鍵字改「->」，又可以少打幾個字。同時移除「;」，因為已經不是一個敘述(statement)的結束：

範例

```

9  searchArr(strArr, "b",
11   _____ (String target, String keyStr)
12       -> target.contains(keyStr)
15 );

```

STEP04 因為知道 functional interface 上唯一的方法的定義，所以二個參數的型別：analyze (String ?, String ?)，也可以一併拿掉：

範例

```

9  searchArr(strArr, "b",
11   _____ (String target, String keyStr)
12       -> target.contains(keyStr)
15 );

```

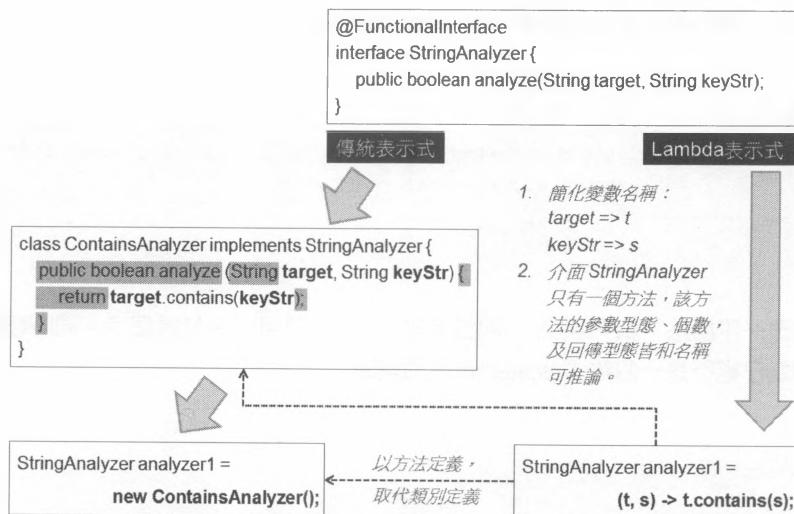
STEP05 最後，變數的名稱愈簡單，打字可以愈少，所以把 target 改為 t，keyStr 改為 s，就大功告成：

範例

```
9  searchArr(strArr, "b",
11          (t, s) -> t.contains(s)
15      );
```

以上流程，可以用下圖表示：

- 若介面是 functional interface，因為建構物件實例時多數程式碼均可推估，可以使用 lambda 表示式簡化程式碼。
- 使用傳統方式建構物件實例時，因為必須使用 new 關鍵字呼叫該類別建構子，所以必須先定義類別並實作介面。



◆ 圖 14-1 Lambda 表示式推斷示意圖

依照前述步驟，可以整理出 `StringAnalyzerTest` 類別的 `test3()` 方法。

由物件導向的多型開發方式，我們可以依需求抽換實作內容，如 `test4()` 方法：

範例

```
1 static void test3() {
2     String[] strArr = { "abc", "bcd", "efg" };
3     searchArr(strArr, "b", (String t, String s) -> t.contains(s));
4 }
```

```

5 static void test4() {
6     String[] strArr = { "abc", "bcd", "efg" };
7     searchArr(strArr, "b", (t, s) -> t.contains(s));
8     searchArr(strArr, "b", (t, s) -> t.startsWith(s));
9     searchArr(strArr, "b", (t, s) -> t.endsWith(s));
10 }

```

最後，lambda 表示式也可以用在變數。如此就可以定義一次，使用多次，如 test5() 方法：

範例

```

1 static void test5() {
2     String[] strArr = { "abc", "bcd", "efg" };
3
4     StringAnalyzer sa1 = (t, s) -> t.contains(s);
5     StringAnalyzer sa2 = (t, s) -> t.startsWith(s);
6     StringAnalyzer sa3 = (t, s) -> t.endsWith(s);
7
8     searchArr(strArr, "b", sa1);
9     searchArr(strArr, "b", sa2);
10    searchArr(strArr, "b", sa3);
11 }

```

14.3 使用基礎的內建functional interfaces

Java 8 導入了 functional interface 的使用：

- 只能有一個方法需要實作。
- Lambda 表示式必須搭配這類型的介面。

因為 functional interface 只有一個方法，可以預期使用方式，因此 Java 8 在套件「java.util.function」下內建了許多的 functional interfaces，可以直接使用。基礎的有四種：

- 評斷型 (Predicate)：使用泛型傳入參數，且回傳 true/false。
- 消費型 (Consumer)：使用泛型傳入參數，且沒有回傳 (void)。
- 功能型 (Function)：將傳入的參數由 T 型別轉換成 U 型別後回傳。
- 供應型 (Supplier)：如同工廠方法，回傳 T 型別的實例 / 物件。

以下將分別介紹。

評斷型 (Predicate)

代表的介面是 **Predicate**，其定義為：

```
package java.util.function;
public interface Predicate<T> {
    public boolean test(T t);
}
```

使用 **Predicate<T>** 介面時，可以提供一個類別 **T** 滿足泛型需要；唯一的方法使用 **T** 類別作為參數，方法內容通常和測試 **T** 類別的某些欄位或方法有關，結果必須回傳 **true** 或 **false**。如範例「**/OCP/src/course/c14/basicFI/PredicateDemo.java**」：

範例

```
1  public class PredicateDemo {
2      public static void main(String[] args) {
3          Predicate<Person> olderThan23 =
4              p -> p.getAge() >= 23;
5          for (Person p : Person.createList()) {
6              if (olderThan23.test(p)) {
7                  System.out.println(p);
8              }
9          }
10     }
```

說明

3 和以下匿名類別同義：

```
Predicate<Person> olderThan23 = new Predicate<Person>() {
    public boolean test(Person p) {
        return p.getAge() >= 23;
    }
};
```

結果：印出 age >= 23 的 persons

```
Name=Jane, Age=25, email=jane@x.com
Name=John, Age=25, email=johnx@x.com
Name=Phil, Age=55, email=phil@x.com
Name=Betty, Age=85, email=betty@x.com
```

消費型 (Consumer)

代表的介面是 Consumer，其定義為：

```
package java.util.function;
public interface Consumer<T> {
    public void accept(T t);
}
```

使用 Consumer<T> 介面時，可以提供一個類別 T 滿足泛型需要；唯一的方法使用 T 類別作為參數，方法內容通常和 T 類別的某些欄位或方法有關，無結果回傳。如範例「/OCP/src/course/c14/basicFI/ConsumerDemo.java」：

範例

```
1  public class ConsumerDemo {
2      public static void main(String[] args) {
3          Consumer<Person> printPerson =
4              p -> p.printPerson();
5          for (Person p: Person.createList()) {
6              printPerson.accept(p);
7          }
8      }
}
```

說明

3	和以下匿名類別同義： <pre>Consumer<Person> printPerson = new Consumer<Person>() { public void accept(Person p) { p.printPerson(); } };</pre>
---	---

結果：印出集合物件裡的所有 persons

```
Name=Bob, Age=21, email=bob@x.com
Name=Jane, Age=25, email=jane@x.com
Name=John, Age=25, email=johnx@x.com
Name=Phil, Age=55, email=phil@x.com
Name=Betty, Age=85, email=betty@x.com
```

功能型 (Function)

代表的介面是 Function，其定義為：

```
package java.util.function;
public interface Function<T,R> {
    public R apply(T t);
}
```

使用 Function<T, R> 介面時，可以提供二個類別 T 和 R 滿足泛型需要；唯一的方法使用 T 類別作為參數，回傳型態為 R 類別，R 字元可以聯想為 result 或 reply。範例「/OCP/src/course/c14/basicFI/FunctionDemo.java」演練了使用方式：

範例

```
1 public class FunctionDemo {
2     public static void main(String[] args) {
3         Function<Person, String> getNameFromPerson =
4             p -> p.getName();
5         for (Person p: Person.createList()) {
6             System.out.println(getNameFromPerson.apply(p));
7         }
8     }
}
```

說明

3 和以下匿名類別等價：

```
Function<Person, String> getNameFromPerson =
    new Function<Person, String>() {
        public String apply(Person p) {
            return p.getName();
        }
    };
};
```

供應型 (Supplier)

代表的介面是 Supplier，其定義為：

```
package java.util.function;
public interface Supplier<T> {
    public T get();
}
```

使用 Supplier<T> 介面時，可以提供類別 T 滿足泛型需要；唯一的方法沒有參數，回傳型態為 T 類別。範例「/OCP/src/course/c14/basicFI/SupplierDemo.java」演練了使用方式：

範例

```

1 public class SupplierDemo {
2     public static void main(String[] args) {
3         Supplier<Person> personSupplier =
4             () -> new Person("New", "new@x.com", 21);
5         System.out.println(personSupplier.get());
6     }

```

說明

3	和以下匿名類別等價： <code>Supplier<Person> personSupplier = new Supplier<Person>() { public Person get() { return new Person("New", "new@x.com", 21); } };</code>
---	---

14.4 在泛型內使用萬用字元

在泛型裡若要使用萬用字元 (Wildcards)，需要使用「?」符號。因為在接下來的主題裡將會被頻繁使用，因此先予說明：

語法

<?>

表示型別可以是任何類別，沒有上 / 下限。

語法

<? extends T>

表示型別必須是 T 或 T 的子類別。此時以類別 T 為上限，但沒有下限。

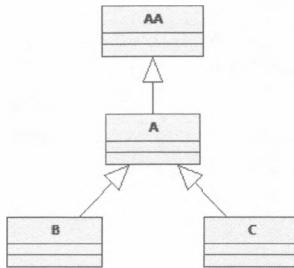
語法

<? super T>

表示型別必須是 T 或 T 的父類別。此時以類別 T 為下限，但沒有上限。

在泛型裡使用多型

我們將在範例「/OCP/src/course/c14/WildcardTest.java」裡逐步介紹在泛型裡使用萬用字元(Wildcards)的概念。首先，假設類別 AA、A、B、C 具備繼承關係如下：



◆ 圖 14-2 類別 AA、A、B、C 的繼承關係

這樣的程式碼很單純，沒有問題：

```
List<A> listA = new ArrayList<A>();
List<B> listB = new ArrayList<B>();
```

這樣的程式碼無法通過編譯：

```
listA = listB; // compile error!
List<A> listA = new ArrayList<B>(); // compile error!
```

這兩行程式碼表示相同的意思。亦即將宣告為 List<A> 的變數，指向 ArrayList 的物件實例。試想，宣告為 List<A>，表示 List 裡可以放入 A、B、C 的物件，但若物件實例是 ArrayList，則實際只能放入 B 的物件，兩者不能相提並論。

這樣的程式碼也無法通過編譯：

```
listB = listA; // compile error!
List<B> listB = new ArrayList<A>(); // compile error!
```

這兩行程式碼表示相同的意思。亦即將宣告為 List 的變數，指向 ArrayList<A> 的物件實例。宣告為 List，表示 List 裡只可能拿出 B 的物件，但若物件實例是 ArrayList<A>，卻可能拿出 A、B、C 的物件，兩者也無法畫上等號。

這樣的結果告訴我們，「=」運算子左邊是 List 型別，右邊可以是 ArrayList 型別，這樣表示多型概念的實作。但 List<A> 的泛型使用型別 A，ArrayList 的泛型卻不能是型別 B 或 C，還是只能是 A。如果想在泛型裡使用繼承或多型的概念，必須使用「<? super T>」或「<? extends T>」。如下：

```
List<?> listUnknown0 = new ArrayList<A>();
List<? extends A> listUnknown1 = new ArrayList<A>();
List<? extends A> listUnknown11 = new ArrayList<B>();
List<? super A> listUnknown2 = new ArrayList<A>();
```

上述程式碼的意思為：

1. 第一行的<?>表示 List 裡可以是任何物件，所以物件實例可以是 ArrayList<A>。
2. 第二行的<? extends A>表示 List 裡必須是 A 或 A 的子類別，所以物件實例可以是 ArrayList<A>。
3. 同上，所以物件實例也可以是 ArrayList。
4. 第四行的<? super A>表示 List 裡必須是 A 或 A 的父類別，所以物件實例可以是 ArrayList<A>。但這樣的程式碼無法通過編譯，因為 B 是 A 的子類別：

```
List<? super A> listUnknown22 = new ArrayList<B>(); // compile error!
```

存取使用 ? 的泛型的集合物件

以下方法說明使用「<?>」時的注意事項：

範例

```
1  private static void processElements(List<?> elements) {
2      // elements.add(new A());           // compile error!
3      // elements.add(new B());           // compile error!
4      // elements.add(new C());           // compile error!
5      // elements.add(new Object());     // compile error!
6      for (Object o : elements) {
7          System.out.println(o);
8      }
9  }
10 private static void testProcessElements() {
11     List<A> listA = new ArrayList<A>();
12     processElements(listA);
13     List<B> listB = new ArrayList<B>();
14     processElements(listB);
15     List<C> listC = new ArrayList<C>();
16     processElements(listC);
17 }
```

說明

| | |
|-----|--|
| 2~5 | 因為傳進來的 List 有可能是 List<A>、List 或 List<String> 等，完全發散，無法預期究竟 List 裡可以放入甚麼型別的物件，所以不允許使用 add() 方法加入物件。避免是 List，卻要加入 new C() 的情況。 |
| 6~8 | 但無論如何，List 裡的物件一定是 Object 子類別，所以可以使用 Object 類別作為參照型別。 |

以下方法說明使用「<? extends A>」時的注意事項：

範例

```

1 private static void processExtendsElements(List<? extends A> list) {
2     // elements.add(new A());           // compile error!
3     // elements.add(new B());           // compile error!
4     // elements.add(new C());           // compile error!
5     // elements.add(new Object());      // compile error!
6     for (A a : list) {
7         System.out.println(a.getClass().getName());
8     }
9 }
10 private static void testProcessExtendsElements() {
11     List<A> listA = new ArrayList<A>();
12     processExtendsElements(listA);
13     List<B> listB = new ArrayList<B>();
14     processExtendsElements(listB);
15     List<C> listC = new ArrayList<C>();
16     processExtendsElements(listC);
17 }
```

說明

| | |
|-----|---|
| 2~5 | 使用「<? extends A>」表示泛型必須是 A 或 A 的子類別，若傳入方法的物件實例是：
1.ArrayList<A>，則可以放物件 A、B、C。
2.ArrayList，則可以放物件 B。
3.ArrayList<C>，則可以放物件 C。
因為大家無交集，所以不允許使用 add() 方法放入物件。 |
| 6~8 | 但無論如何，List 裡的物件一定是 A 或 A 的子類別，所以可以使用 A 類別作為參照型別。 |

以下方法說明使用「<? super A>」時的注意事項：

範例

```

1 public static void insertElements(List<? super A> list) {
2     list.add(new A());
3     list.add(new B());
4     list.add(new C());
```

```

5  /* compile error!
6  for (A a : list) {
7      System.out.println(a.getClass().getName());
8  }*/
9  Object object = list.get(0);
10 }
11 private static void testInsertElements() {
12     List<A> listA = new ArrayList<A>();
13     insertElements(listA);
14     List<AA> listAA = new ArrayList<AA>();
15     insertElements(listAA);
16     List<Object> listObject = new ArrayList<Object>();
17     insertElements(listObject);
18 }

```

 說明

| | |
|-----|--|
| 2~4 | 使用「<? super A>」表示泛型必須是 A 或 A 的父類別，若傳入方法的物件實例是：
1. <code>ArrayList<AA></code> ，則可以放物件 A、B、C。
2. <code>ArrayList<A></code> ，則可以放物件 A、B、C。
所以編譯器允許使用 <code>add()</code> 方法將 A 或 A 的子類別放進來。 |
| 5~8 | 因為可能是 A、AA、A 的其他父類別或 <code>Object</code> 類別。所以只能使用 <code>Object</code> 類別作為物件參考，其他都不行。 |
| 9 | 任何類別都是 <code>Object</code> 的子類別。 |

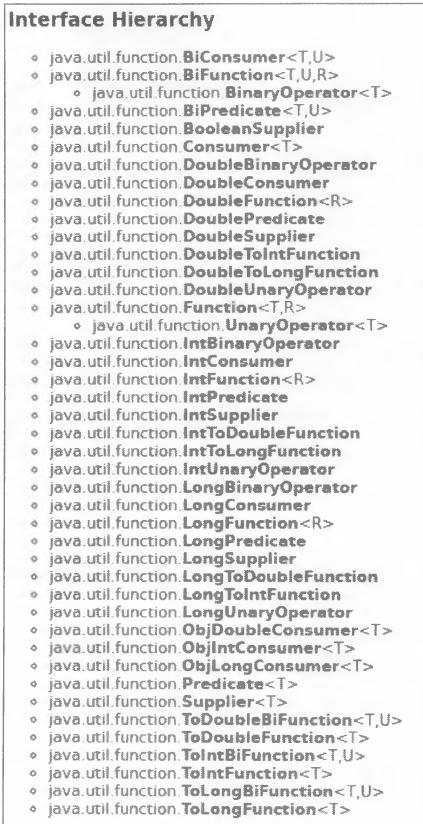
彙整後的結論是：

表 14-2 泛型使用萬用字元 (wildcards) 注意事項

| 使用萬用字元 | <?> | <? extends T> | <? super T> |
|--------|---------------------------------|----------------|---------------------------------|
| 檢視成員 | 只能使用 <code>Object</code> 型別檢視成員 | 使用 T 或其父類別檢視成員 | 只能使用 <code>Object</code> 型別檢視成員 |
| 增加成員 | 無法增加成員 | 無法增加成員 | 可以增加 T 或其子類別的成員 |

14.5 使用進階的內建functional interfaces

除了之前介紹的四個基礎 functional interfaces，分別為 `Predicate`、`Consumer`、`Function`、`Supplier` 外，若參照 API 文件：<https://docs.oracle.com/javase/8/docs/api/java/util/function/package-tree.html> 可以發現 functional interfaces 還有很多：

❖ 圖 14-3 在 `java.util.function` 的 functional interfaces

常用的有以下三類：

1. 四個基礎 functional interfaces 的基本型別變形版

將方法傳入或回傳的物件的其中一個或全部改為基本型別，如 `DoubleFunction`、`ToDoubleFunction`。

2. Binary (二運算元相關) 及其基本型別變形版

將方法傳入的參數由一個變二個，如 `BiPredicate`。

3. Unary (單一運算元相關) 及其基本型別變形版

繼承介面 `Function<T, T>`，但將泛型數量由二個降為一個，如 `UnaryOperator`，其方法傳入和回傳的型別一致。

本節稍後的範例均共用類別 Person：「/OCP/src/course/c14/Person.java」：

範例

```

1  public class Person {
2      private String name, email;
3      private int age;
4      public Person() {}
5      public Person(String name, String email, int age) {
6          this.name = name;
7          this.age = age;
8          this.email = email;
9      }
10     public String getName() {
11         return name;
12     }
13     public int getAge() {
14         return age;
15     }
16     public String getEmail() {
17         return email;
18     }
19     @Override
20     public String toString() {
21         return "Name=" + name + ", Age=" + age + ", email=" + email;
22     }
23     public void printPerson() {
24         System.out.println(this);
25     }
26     public static List<Person> createList() {
27         List<Person> people = new ArrayList<>();
28         people.add(new Person("Bob", "bob@x.com", 21));
29         people.add(new Person("Jane", "jane@x.com", 34));
30         people.add(new Person("John", "johnx@x.com", 25));
31         people.add(new Person("Phil", "phil@x.com", 65));
32         people.add(new Person("Betty", "betty@x.com", 55));
33         return people;
34     }
35 }
```

四個基礎 functional interfaces 的基本型別變形版

常見的介面如下表：

❖ 表 14-3 基礎 functional interfaces 的基本型別變形版

| 來源 | 原功能 | 基本型別變形版 | |
|----------------|--------------|--|--|
| Predicate<T> | T -> boolean | Int, long, double -> boolean | IntPredicate
LongPredicate
DoublePredicate |
| Consumer<T> | T -> void | Int, long, double -> void | IntConsumer
LongConsumer
DoubleConsumer |
| Function<T, R> | T -> R | Int, long, double -> R | IntFunction<R>
LongFunction<R>
DoubleFunction<R> |
| | | T -> Int, long, double | ToIntFunction<T>
ToDoubleFunction<T>
ToLongFunction<T> |
| | | Int, long, double -> Int, long, double | LongToDoubleFunction
LongToIntFunction
DoubleToIntFunction
DoubleToLongFunction
IntToDoubleFunction
IntToLongFunction |
| Supplier<T> | () -> T | | BooleanSupplier
IntSupplier
LongSupplier
DoubleSupplier |

其特徵為方法傳入或回傳物件的其中一個或全部是基本型別，以介面 ToDoubleFunction 為例：

```
package java.util.function;
public interface ToDoubleFunction<T> {
    public double applyAsDouble(T t);
}
```

使用 ToDoubleFunction<T> 介面時，需要提供一個 T 類別作為泛型；唯一的方法傳入 T 類別物件，並回傳 double 基本型別，方法內容可以和測試該類別的某些欄位或方法有關。如範例「/OCP/src/course/c14/advanceFI/ToDoubleFunctionDemo.java」：

🔍 範例

```
1  public class ToDoubleFunctionDemo {
2      public static void main(String[] args) {
3          List<Person> pl = Person.createList();
4          Person first = pl.get(0);
5          ToDoubleFunction<Person> convertAgeToDouble = p -> p.getAge();
6          System.out.println(convertAgeToDouble.applyAsDouble(first));
```

```

7     }
8 }
```

說明

和以下匿名類別同義：

```
ToDoubleFunction<Person> convertAgeToDouble =
    new ToDoubleFunction<Person>() {
        public double applyAsDouble(Person p) {
            return p.getAge();
        }
   };
```

再以介面 DoubleFunction 為例：

```
package java.util.function;
public interface DoubleFunction<R> {
    public R apply(double value);
}
```

使用 DoubleFunction<R> 介面時，需要提供一個 R 類別作為泛型；唯一的方法傳入 double 基本型別，並回傳 R 類別的物件。如範例「/OCP/src/course/c14/advanceFI/DoubleFunctionDemo.java」：

範例

```

1 public class DoubleFunctionDemo {
2     public static void main(String[] args) {
3         DoubleFunction<String> calc = t -> String.valueOf(t * 10);
4         String result = calc.apply(3.1415926);
5         System.out.println("New value is: " + result);
6     }
7 }
```

說明

和以下匿名類別同義：

```
DoubleFunction<String> calc = new DoubleFunction<String>() {
    public String apply(double v) {
        return String.valueOf(v * 3);
    }
};
```

Binary (二運算元相關) 及其基本型別變形版

常見的介面如下表：

◆ 表 14-4 Binary (二運算元相關) 及其基本型別變形版

| Functional Interface | 功能 | 基本型別變形版 | |
|----------------------|-------------------|--------------------------------|---|
| BinaryOperator <T> | (T, T) -> T | 將 T 置換為
int,long,double | IntBinaryOperator
LongBinaryOperator
DoubleBinaryOperator |
| BiPredicate <L, R> | (L, R) -> boolean | None | |
| BiConsumer <T, U> | (T, U) -> void | 將 U 置換為
int,long,double | ObjIntConsumer<T>
ObjLongConsumer<T>
ObjDoubleConsumer<T> |
| BiFunction <T, U, R> | (T, U) -> R | 將 R(回傳) 置換為
int,long,double | ToIntBiFunction<T, U>
ToLongBiFunction<T, U>
ToDoubleBiFunction<T, U> |

這類型的介面的方法都有二個傳入參數，故名 Binary (二運算元相關)。比較介面 Predicate 和 BiPredicate 可以發現方法由原本的一個參數，提高為二個參數：

```
package java.util.function;
public interface BiPredicate<T, U> {
    public boolean test(T t, U u);
}
```

使用 BiPredicate <T, U> 介面時，需要提供一個 T 和 U 類別作為泛型；唯一的方法使用 T 和 U 類別作為參數，結果必須回傳 true/false。如範例「/OCP/src/course/c14/advanceFI/BiPredicateDemo.java」：

範例

```
1  public class BiPredicateDemo {
2      public static void main(String[] args) {
3          List<Person> pl = Person.createList();
4          Person first = pl.get(0);
5          String testName = "john";
6          BiPredicate<Person, String> nameBiPred =
7              (p, s) -> p.getName().equalsIgnoreCase(s);
8          System.out.println("Is the first john? "
9                  + nameBiPred.test(first, testName));
10     }
11 }
```

 **說明**

6 和以下匿名類別同義：

```
BiPredicate<Person, String> nameBiPred =
    new BiPredicate<Person, String>() {
        public boolean test(Person p, String s) {
            return p.getName().equalsIgnoreCase(s);
        }
   };
```

Unary (單一運算元相關) 及其基本型別變形版

常見的介面如下表：

◆表 14-5 Unary (單一運算元相關) 及其基本型別變形版

| Functional Interface | 功能 | 基本型別變形版 |
|----------------------|--------|---|
| UnaryOperator <T> | T -> T | 將 T 置換為 int, long, double
IntUnaryOperator
LongUnaryOperator
DoubleUnaryOperator |

介面的定義為：

```
package java.util.function;
public interface UnaryOperator<T> extends Function<T,T> {
    public T apply(T t);
}
```

UnaryOperator<T> 繼承介面 Function<T, T>，但將泛型數量由二個降為一個，因此方法也只傳入一個物件，且傳入和回傳的型別一致。這個過程中通常會改變 T 物件的某些狀態。如範例「/OCP/src/course/c14/advanceFI/UnaryOperatorDemo.java」：

 **範例**

```
1  public class UnaryOperatorDemo {
2      public static void main(String[] args) {
3          List<Person> pl = Person.createList();
4          Person first = pl.get(0);
5          UnaryOperator<String> unaryStr = s -> s.toUpperCase();
6          System.out.println("Before: " + first.getName());
7          System.out.println("After: " + unaryStr.apply(first.getName()));
8      }
9  }
```

 **說明**

5 和以下匿名類別同義：

```
UnaryOperator<String> unaryStr = new UnaryOperator<String>() {
    public String apply(String s) {
        return s.toUpperCase();
    }
};
```

14.6 使用方法參照

Lambda 表示式所呈現的「匿名方法」必須包含三部分：

1. 方法參數 (argument list)。
2. 箭頭符號 (arrow token)，為「->」。
3. 方法內容 (body)。

若方法內容 (body) 只是呼叫另一個方法 (method)，可將 lambda 表示式再改寫為「方法參照 (method reference)」，讓語法更簡潔。依據被呼叫的方法的種類和來源，可分成以下四類：

1. 方法是靜態 (static) 方法，來自某類別。
2. 方法是實例 (instance) 方法，來自 lambda 表示式之「外」的參考變數。
3. 方法是實例 (instance) 方法，來自 lambda 表示式之「內」的參考變數。
4. 使用 new 呼叫建構子。

將分別詳述於後。

在開始之前，因為接下來的範例都將使用 Arrays 類別的 static 搜尋方法「sort()」：

```
public static <T> void sort (T[ ] a, Comparator<? super T> c) {
    ...
}
```

有必要先知道 Comparator 介面的宣告，在 Java 8 時已經被加上 @FunctionalInterface 的 annotation：

```
@FunctionalInterface
public interface Comparator<T> { ... }
```

所以唯一的抽象方法可以使用 lambda 表示式：

```
int compare(T o1, T o2);
```

1. 方法參照使用靜態方法，來自某類別

三 語法

ContainingClass::staticMethodName

範例「/OCP/src/course/c14/methodRefer/MethodReferenceDemo1.java」演練了使用方式：

四 範例

```
1 class Utility1 {
2     static int compareIgnoreCase(String s1, String s2) {
3         return s1.compareToIgnoreCase(s2);
4     }
5 }
6 public class MethodReferenceDemo1 {
7     public static void main(String[] args) {
8         String[] arr = { "a", "b", "c", "d", "e" };
9         // Arrays.sort(arr, (a, b) -> Utility1.compareIgnoreCase(a, b));
10        Arrays.sort(arr, Utility1::compareIgnoreCase);
11    }
12 }
```

2. 方法參照使用實例方法，來自 lambda 表示式之「外」的參考變數

三 語法

objectReference::instanceMethodName

範例「/OCP/src/course/c14/methodRefer/MethodReferenceDemo2.java」演練了使用方式：

四 範例

```
1 class Utility2 {
2     int compare(String s1, String s2) {
3         return s1.compareTo(s2);
4     }
5 }
6 public class MethodReferenceDemo2 {
7     public static void main(String[] args) {
8         String[] arr = { "a", "b", "c", "d", "e" };
9         Utility2 u = new Utility2();
```

```

10     //    Arrays.sort(arr, (a, b) -> u.compare(a, b) );
11     Arrays.sort(arr, u::compare);
12 }
13 }
```

3. 方法參照使用實例方法，來自 lambda 表達式之「內」的參考變數

語法

ObjectReferenceType::instanceMethodName

範例「/OCP/src/course/c14/methodRefer/MethodReferenceDemo3.java」演練了使用方式：

範例

```

1 public class MethodReferenceDemo3 {
2     public static void main(String[] args) {
3         String[] arr = { "a", "b", "c", "d", "e" };
4         //    Arrays.sort(arr, (a, b) -> a.compareToIgnoreCase(b) );
5         Arrays.sort(arr, String::compareToIgnoreCase);
6     }
7 }
```



課堂小祕訣

使用物件參考的實例方法的情況有兩種，一種物件參考來自 lambda 表達式外面，要傳進來當然只能使用原來的變數名稱；剩餘的因為不能再使用變數名稱，就改用類別名稱。但這樣和使用靜態方法時很像，都是把類別名稱放前面。以本例而言，String 類別的 compareToIgnoreCase() 方法畢竟不是 static，還是可以區分。

4. 使用 new 呼叫建構子

語法

ClassName::new

範例「/OCP/src/course/c14/methodRefer/MethodReferenceDemo4.java」演練了使用方式：

範例

```

1 public class MethodReferenceDemo4 {
2     String x;
3     MethodReferenceDemo4() {
4     }
5     MethodReferenceDemo4(String x) {
```

```

6      this.x = x;
7  }
8  void printX() {
9      System.out.println(this.x);
10 }
11 private static void createObjectWithInput() {
12     // Function<String, MethodReferenceDemo4> factory =
13             (s) -> new MethodReferenceDemo4(s);
14     Function<String, MethodReferenceDemo4> factory =
15             MethodReferenceDemo4::new;
16     MethodReferenceDemo4 demo4 =
17             factory.apply("use constructor to input");
18     demo4.printX();
19 }
20 private static void createObjectWithoutInput() {
21     // Supplier<MethodReferenceDemo4> supplier =
22             () -> new MethodReferenceDemo4();
23     Supplier<MethodReferenceDemo4> supplier =
24             MethodReferenceDemo4::new;
25     MethodReferenceDemo4 demo4 = supplier.get();
26 }
27 public static void main(String[] args) {
28     createObjectWithInput();
29     createObjectWithoutInput();
30 }
31 }
```

說明

使用方法參照呼叫建構子時，語法上只能使用類別名稱，無法傳入建構子的參數，因此必須轉個彎由其他地方傳入呼叫建構子時需要的參數，使用 Function 介面搭配行 13 和 14 的程式碼就可以達到這樣的目的。反之，若使用 Supplier 介面搭配行 19 和行 20 的程式碼，就只能呼叫不帶參數的建構子。

範例「/OCP/src/course/c14/methodRefer/MethodReferenceDemo_.java」是另一個使用方法參照的例子，請各位讀者自行參閱。

14.7 認證考試命題範圍

依據甲骨文公布的考試範圍 (https://education.oracle.com/java-se-8-programmer-ii/pexam_1Z0-809)，和本章相關的主題有：

Advanced Java Class Design

1. Create and use **Lambda** expressions.

Lambda Built-in Functional Interfaces

1. Use the built-in interfaces included in the `java.util.function` package such as **Predicate**, **Consumer**, **Function**, and **Supplier**.
2. Develop code that uses **primitive** versions of **functional interfaces**.
3. Develop code that uses **binary** versions of **functional interfaces**.
4. Develop code that uses the **UnaryOperator** interface.

本章擬真試題實戰

考題 1

Given:

```
public static void main(String[] args) {
    BiFunction<Integer, Double, Integer> result = (v1, v2) -> v1 + v2; // line n1
    System.out.println(result.apply(10, 10.5)); //line n2
}
```

What is the result?

- A. 20
- B. 20.5
- C. A compilation error occurs at line n1.
- D. A compilation error occurs at line n2.

答案 C

說明

```
@FunctionalInterface
public interface BiFunction<T, U, R> {
    R apply(T t, U u);
}
```

比對題目後：

T : Integer

U : Double

R : Integer

因此必須回傳 Integer，然原式回傳 Double，故編譯失敗。可以將 lambda 表示式改為 (v1, v2) -> v1 + v2.intValue();

考題 2

Given:

```
public static void main(String[] args) {  
    UnaryOperator<Double> uod = d -> d + 100.0;  
    List<Integer> ints = Arrays.asList (15, 25);  
    ints.replaceAll(uod);  
    ints.forEach(i -> System.out.println(i));  
}
```

What is the result?

- A. 20.0 30.0
- B. 10
- C. A compilation error occurs.
- D. A NumberFormatException is thrown at run time.

答案 C

說明 必須先知道：

```
@FunctionalInterface  
public interface UnaryOperator<T> extends Function<T, T> {  
    static <T> UnaryOperator<T> identity() {  
        return t -> t;  
    }  
}
```

及 Java 8 在 List 介面裡新增方法：

```
default void replaceAll (UnaryOperator<E> operator)
```

UnaryOperator<T> 的方法可以傳入 T 型別，傳出 T 型別，所以可以用來將 List 裡的物件成員全部改變，但只限於同型別！本題的 List<Integer> 的成員型別是 Integer，透過 UnaryOperator<Double> 的方法卻變成 Double，故編譯失敗。

考題 3

Given:

```
interface CarFactory {
    Car getCar(String brand);
}

class Car {
    private String brand;
    public Car(String brand) {
        this.brand = brand;
    }
}
```

Which code fragment creates an instance of Car?

- A. Car auto = Car("MyCar")::new;
- B. Car auto = Car::new;
Car vehicle = auto::getCar("MyCar");
- C. CarFactory rider = Car::new;
Car vehicle = rider.getCar("MyCar");
- D. Car vehicle = CarFactory::new::getCar("MyCar");

答案 C

說明 請參閱本書範例「/OCP/src/course/c14/methodRefer/MethodReferenceDemo4.java」及相關內容。

考題 4

Given:

```
class Bird {
    public void fly() {
        System.out.println("Can fly");
    }
}
class Rooster extends Bird {
    public void fly() {
        System.out.println("Not fly");
    }
}
```

And:

```
public class Test {  
    public static void main(String[] args) {  
        fly(() -> new Bird());  
        fly(Rooster::new);  
    }  
    /* line */  
}
```

Which code fragment, when inserted at line, enables the Test class to compile?

A.

```
static void fly (Consumer<Bird> bird) {  
    bird::fly  
}
```

B.

```
static void fly (Consumer<? extends Bird> bird) {  
    bird.accept().fly();  
}
```

C.

```
static void fly(Supplier<Bird> bird) {  
    bird.get().fly();  
}
```

答案 C

說明

```
@FunctionalInterface  
public interface Consumer<T> {  
    void accept(T t);  
}
```

介面 Consumer 的方法無回傳

```
@FunctionalInterface  
public interface Supplier<T> {  
    T get();  
}
```

介面 Supplier 的方法可以回傳物件，故選擇 C。

考題 5

Given:

```
public static void main(String[] args) {
    String str = "Java is the most popular language";
   ToIntFunction<String> getIndex = str::indexOf;           //line n1
    int i = getIndex.applyAsInt("Java");                      //line n2
    System.out.println(i);
}
```

What is the result?

- A. 0
- B. 1
- C. A compilation error occurs at line n1.
- D. A compilation error occurs at line n2.

答案 A

說明 方法參照 str::indexOf 同義於：

```
s -> str.indexOf(s);
```

考題 6

Given:

```
class Check {
    public static int checkItem(String s1, String s2) {
        return s1.length() - s2.length();
    }
}
```

And:

```
public static void main(String[] args) {
    String[] arr = new String[] { "Tiger", "Dog", "Cat", "Lion" };
    // insert code here
    for (String s : arr) {
        System.out.print(s + " ");
    }
}
```

Which code fragment should be inserted at //insert code here to enable the code to print Dog Cat Lion Tiger?

- A. `Arrays.sort(arr, Check::checkItem);`
- B. `Arrays.sort(arr, (Check::new)::checkItem);`
- C. `Arrays.sort(arr, (Check::new).checkItem);`
- D. `Arrays.sort(arr, Check::new::checkItem);`

答案 A

說明 `checkItem()` 是 static 方法，必須使用 `Check::checkItem`。

考題 7

Given:

```
interface CanMove<T> {  
    public default void walk(T length) {  
        System.out.println("walk");  
    }  
    public void run(T length);  
}
```

Which statement is compiled?

A.

```
CanMove<Integer> animal = n -> System.out.println("run" + n);  
animal.run(200);  
animal.walk(10);
```

B.

```
CanMove<Integer> animal = n -> n + 10;  
animal.run(200);  
animal.walk(10);
```

C.

```
CanMove animal = (Integer n) -> System.out.println(n);  
animal.run(200);  
CanMove.walk(10);
```

D.

`CanMove` cannot be used in a lambda expression.

答案 A**說明**

選項 B : CanMove<Integer> animal = n -> n + 10; 表示有回傳值，與 void run(T length); 定義不合。

選項 C : CanMove animal = (Integer n) -> System.out.println(n); 應改為 CanMove<Integer> animal = (Integer n) -> System.out.println(n); 才能編譯。CanMove.walk(10); 應改為 animal.walk(10);

選項 D : 可以使用，如選項 A。

使用Stream API

-
- | 15.1 建構者模式和方法鏈結
 - | 15.2 使用Optional 類別
 - | 15.3 Stream API 介紹
 - | 15.4 Stream API 操作
 - | 15.5 Stream API 和NIO.2
 - | 15.6 Stream API 操作平行化
 - | 15.7 認證考試命題範圍

15.1 建構者模式和方法鏈結

類別經常具備不定數量的屬性\欄位。某些欄位若和物件生成有密切關係，我們通常會使用建構子(constructor)要求建構物件時一併傳入。這可能導致幾個問題：

1. 某些類別的設計會依傳入建構子的欄位不同而建構出不同功能性的物件，如咖啡加入不同佐料將產生不同口味的飲品，此時會發現有許多 overloading 的建構子。
2. 建構子的參數可能很多。
3. 建構子裡的參數若有多個屬於相同型別，將造成參數組合複雜且設計困難。
4. 必須判斷 null 的情況。

由以上描述，問題的癥結在建構物件的方式有多種時，只依賴建構子的 overloading 顯然不夠，此時可以將建構物件的邏輯抽出介面，改用不同的實作類別來協助產生建構子。這樣的設計模式(design pattern)稱為「建構者模式(builder pattern)」，範例「/OCP/src/course/c15/Person.java」使用建構者模式來建構 Person 類別：

④ 範例

```

1  public class Person {
2      private String name, email;
3      private int age;
4      private Person (Person.Builder builder) {
5          this.name = builder.name;
6          this.age = builder.age;
7          this.email = builder.email;
8      }
9      public static class Builder {
10         private String name = "", email = "";
11         private int age = 0;
12         public Person.Builder name(String name) {
13             this.name = name;
14             return this;
15         }
16         public Person.Builder age(int val) {
17             this.age = val;
18             return this;
19         }
20         public Person.Builder email(String val) {
21             this.email = val;
22             return this;
}

```

```

23     }
24     public Person build() {
25         return new Person(this);
26     }
27 }
28 public String getName() {
29     return name;
30 }
31 public int getAge() {
32     return age;
33 }
34 public String getEmail() {
35     return email;
36 }
37 @Override
38 public String toString() {
39     return "Name=" + name + ", Age=" + age + ", email=" + email + "\n";
40 }
41 public void printPerson() {
42     System.out.println(this);
43 }
44 }

```

說明

| | |
|------|---|
| 2,3 | Person 類別具有屬性欄位 name、email 和 age。三個欄位都 private，有 public 的 getter() 方法，但取消 setter 方法，所以必須依賴其他方式讓物件建構後設定欄位值。 |
| 4~8 | <ol style="list-style-type: none"> 1. 建構子為 private，只能在類別內建立物件。 2. 該建構子以靜態巢狀類別 (static nested class)：Person.Builder 物件作為參數。 3. 要建立 Person 物件，必須先有 Person.Builder 物件。Person.Builder 類別具有和 Person 類別對應且相同的屬性欄位。 |
| 9~27 | <ol style="list-style-type: none"> 1. 本程式區段為靜態巢狀類別 (static nested class)：Person.Builder 的定義方式。 2. 行 12、16、20 為 Person.Builder 的 setter() 方法。因為是 Builder 類別，慣例上命名不以「set」開頭，而且都回傳 Person.Builder 類別。 3. 最後的 build() 方法，回傳被建立的 Person 物件。 |

完成 Person 類別以及其 Builder 類別的設計後，使用範例「/OCP/src/course/c15/TestPerson.java」進行測試。範例裡有多個方法，各有不同測試目的。方法 createPersonList() 用來顯示如何以建構者模式 (Builder Pattern)，使用 Builder 類別來建構 Person 物件：

範例

```

1 public static List<Person> createPersonList() {
2     List<Person> people = new ArrayList<>();
3     people.add(
4         new Person.Builder()
5             .name("Bob")
6             .age(21)
7             .email("bob@x.com")
8             .build()
9     );
10    people.add(
11        new Person.Builder()
12            .name("Jane")
13            .age(25)
14            .email("jane@x.com")
15            .build()
16    );
17    people.add(new Person.Builder()
18        .name("John").age(25).email("johnx.com").build());
19    people.add(new Person.Builder()
20        .name("Phil").age(55).email("phil@x.com").build());
21    people.add(new Person.Builder()
22        .name("Betty").age(85).email("betty@x.com").build());
23    return people;
24 }

```

說明

| | |
|-------|---|
| 4~8 | 本區段為使用建構者模式建構 Person 物件，並逐一設定相關屬性。為了讓 Builder 類別的使用更明顯，每行只呼叫一個屬性值的設定方法。 |
| 11~15 | 同上。 |

一般而言，當需要使用 builder 建立物件時，通常表示需要多種 builder，所以需要共同的 interface 協助 builder 抽換，因此不會以靜態巢狀類別的方式定義。然本例的重點在於強調「建構者模式 (builder pattern)」讓建立物件可以「方法鏈結 (method chaining)」的方式進行：

1. 多個方法可以用一行程式碼表達，讓程式碼理解更容易。
2. 更有彈性的物件建立方式。
3. 每個設定屬性欄位的方法都回傳物件自己。
4. 程式碼更加流暢 (fluent)。

這是 Java SE8 推廣的程式撰寫風格，也是範例主要表達的意涵。

15.2 使用Optional類別

在撰寫 Java 程式碼的過程中，我們一定都會遇過要處理 null 的狀況。比如設計了一個「允許輸入不同數量的整數，以計算平均值」的方法；但若遇到呼叫該方法但沒有輸入任何整數的情況，回傳值該是多少比較合理？是 0 嗎？若沒有任何輸入時回傳 0，那該如何區分真正輸入多個 0 時，會得到 0 的結果？這時候，大部分的做法都會直接回傳一個 null，同時將方法的回傳改用基本型別的包裹類別(wrapper class)。如範例「/OCP/src/course/c15/OptionalDemo.java」的「average0()」方法：

範例

```

1 public static Double average0(int... scores) {
2     if (scores.length == 0)
3         return null;
4     int sum = 0;
5     for (int score : scores)
6         sum += score;
7     return (double) sum / scores.length;
8 }
```

在方法的實作內容裡，若傳進的整數個數為 0，馬上回傳 null。所以呼叫 average0() 方法的程式後，若有需要將結果再延伸使用，就必須處理可能是 null 的情形。

但 null 是甚麼？null 好處理嗎？檢視同一個範例類別裡的「testNull()」方法執行結果：

範例

```

1 public static void testNull() {
2     char str[] = { 'D', 'u', 'k', 'e' };
3     String s = null;
4     for (char c : str) {
5         s = s + c;
6     }
7     System.out.println(s);
8     Object o = null;
9     System.out.println(o);
10    // System.out.println(null); //can't compiled!!
11 }
```

結果

```
nullDuke
null
```

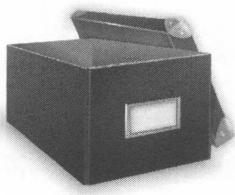
 **說明**

| | |
|------|---|
| 5, 7 | 都說 String s 是 null 了，竟然可以和字元相連！？ |
| 9 | System.out.println() 處理物件，遇到 null 時就印出 null ！ |
| 11 | null 不屬於任何型別，無法通過編譯。 |

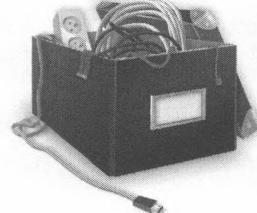
由此，「null」存在的最大優點和缺點都在他的「模糊不清」。遇到不知道該如何處理的時候，就回傳一個 null；也因為這樣的開發習慣，呼叫任何方法只要回傳的是參考型別，就得時常檢查「if xxx != null」，提防不定時炸彈「NullPointerException」的出現，令人困擾。

Java 8 推出了 `Optional<T>` 類別，來改善這種長久以來存在的問題：

1. 屬於 `java.util` 套件。
2. 使用上像是一個「容器 / 箱子」，使用泛型表現箱子裡可以存放 T 物件，也可以是空 (empty) 的，這時就等同於 null 的概念。如下示意：



❖ 圖 15-1 `Optional.empty()`



❖ 圖 15-2 `Optional.of(xx)`

3. 使用「empty()」方法建立一個空的 Optional 物件；使用「of(xx)」方法建立一個內包 xx 物件的 Optional 物件，該物件不可為 null。
4. 可以使用 `isPresent()` 方法確認內容物 T 是否存在，若回傳 true 可以再使用 `get()` 方法取得物件 T。
5. 和 functional interface 一樣，尚有其他支援基本型別的擴充版：
 - `OptionalDouble`
 - `OptionalInt`
 - `OptionalLong`
6. `Optional<T>` 其他常見的方法有：

| 方法 | Optional 為空 | Optional 含物件 |
|-------------------------|--------------------------|-------------------|
| get() | 拋出 Exception | 取得物件 |
| ifPresent(Consumer c) | 不做任何事情 | 呼叫 Consumer 定義的方法 |
| isPresent() | 回傳 false | 回傳 true |
| orElse(T other) | 回傳指定的 other 物件 (同為 T 型別) | 取得物件 |
| orElseGet(Supplier s) | 回傳 Supplier 定義的方法的執行結果 | 取得物件 |
| orElseThrow(Supplier s) | 回傳 Supplier 定義的方法所拋出的例外 | 取得物件 |

了解 Optional 類別的定義和作用後，改寫先前的求平均方法為「average1()」：

② 範例

```

1 public static Optional<Double> average1(int... scores) {
2     if (scores.length == 0)
3         return Optional.empty();
4     int sum = 0;
5     for (int score : scores)
6         sum += score;
7     return Optional.of((double) sum / scores.length);
8 }
```

③ 說明

| | |
|---|--|
| 1 | 使用泛型宣告回傳的 Optional 物件的內含物件必須是 Double 型別。 |
| 3 | 回傳空的 Optional 物件。 |
| 7 | 回傳內含 Double 物件的 Optional 物件。 |

④ 範例

```

1 public static void testOptional() {
2     System.out.println("show01: " + average1(90, 100));
3     System.out.println("show02: " + average1());
4
5     Optional<Double> optOK = average1(90, 100);
6     if (optOK.isPresent()) {
7         System.out.println("show03: " + optOK.get()); // 95.0
8     }
9     Optional<Double> optNG = average1();
10    // System.out.println("show04: " + optNG.get());
11
12    Optional<Double> opt1 = average1(90, 100);
13    opt1.ifPresent(d -> System.out.println("show05: " + d));
14
15    // no value in it
```

```

16     Optional<Double> opt2 = average1();
17     System.out.println("show06: " + opt2.orElse(Double.NaN));
18     System.out.println("show07: " +
19         opt2.orElseGet(() -> Math.random()));
20     // System.out.println("show08: " +
21         opt2.orElseThrow(() -> new IllegalStateException()));
22
23     // there is value in it
24     Optional<Double> opt3 = average1(90, 100);
25     System.out.println("show09: " + opt3.orElse(Double.NaN));
26     System.out.println("show10: " +
27         opt3.orElseGet(() -> Math.random()));
28     System.out.println("show11: " +
29         opt3.orElseThrow(() -> new IllegalStateException()));
30 }
```

結果

```

show01: Optional[95.0]
show02: Optional.empty
show03: 95.0
show05: 95.0
show06: NaN
show07: 0.14309330358868522
show09: 95.0
show10: 95.0
show11: 95.0
```

說明

| | |
|-------|---|
| 10 | 若執行，將發生：
Exception in thread "main" java.util.NoSuchElementException: No value present
at java.util.Optional.get(Optional.java:135)
at course.c15.OptionalDemo.testOptional(OptionalDemo.java:57)
at course.c15.OptionalDemo.main(OptionalDemo.java:45) |
| 16~19 | 因opt2不含值，orElse()、orElseGet()、orElseThrow()都會執行方法定義的做法。
其中行19若執行將拋出：
Exception in thread "main" java.lang.IllegalStateException
at course.c15.OptionalDemo.lambda\$2(OptionalDemo.java:66)
at course.c15.OptionalDemo\$\$Lambda\$3/1831932724.get(Unknown Source)
at java.util.Optional.orElseThrow(Optional.java:290)
at course.c15.OptionalDemo.testOptional(OptionalDemo.java:66)
at course.c15.OptionalDemo.main(OptionalDemo.java:45) |
| 22~25 | 因opt3含值，orElse()、orElseGet()、orElseThrow()都直接回傳Optional所包含的值。 |

有些時候，當已經有一個型別為 T 的物件，因為可能是 null，想把它包裝成 Optional 物件再回傳，則可以使用以下方法搭配三元運算子處理：

範例

```

1 public static <T> Optional<T> createOptional(T value) {
2     Optional<T> o = (value == null) ? Optional.empty() : Optional.of(value);
3     return o;
4 }
```

或者將語法更精簡，將行 2 程式碼改成以下：

範例

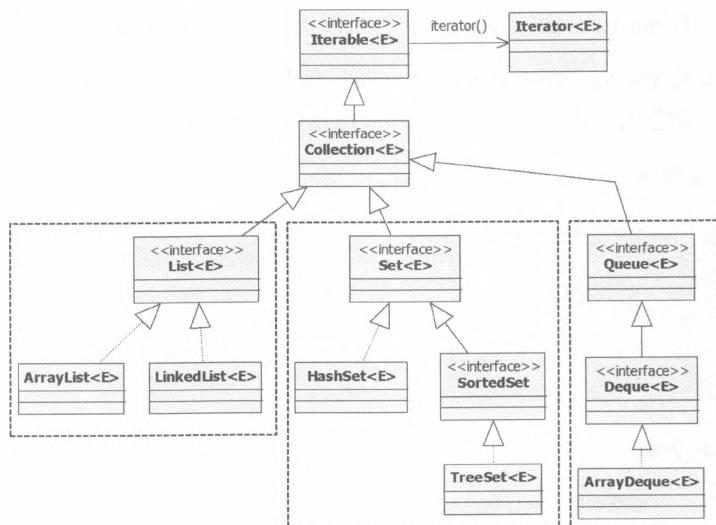
```

1 public static <T> Optional<T> createOptional(T value) {
2     Optional o = Optional.ofNullable(value);
3     return o;
4 }
```

15.3 Stream API 介紹

15.3.1 介面 Iterable 和 Collection 的擴充

Stream API 的使用，和集合物件 (Collection) 有密不可分的關係。在本書的第六章對 Collection 家族有過詳細介紹：



◆ 圖 15-3 Collection 家族類別圖

Java 8 在 Iterable 介面上新增了宣告為 default 的「`forEach()`」方法，允許傳入實作 Consumer 介面的參考物件，該方法會對集合物件的所有成員執行 Consumer 定義的方法，有「針對每一個」的意思。搭配 Lambda 表示式後，可以如範例「/OCP/src/course/c15/TestPerson.java」的「`test1()`」方法：

🔍 範例

```
1 public static void test1() {
2     List<Person> pl = createPersonList();
3     pl.forEach( p -> System.out.println(p) );
4 }
```

🔊 說明

| | |
|---|--|
| 3 | 對每個 Person p 使用 <code>System.out.println(p)</code> 。 |
|---|--|

除此之外，Java 8 在 Collection 介面中也新增了宣告為 default 的「`stream()`」方法，`stream` 的中文解釋是水流，因此如同幫 Collection 容器物件裝了水龍頭，讓成員可以逐一流出。流出的成員為 Stream 物件，利用其 API 可取代過去必須用迴圈來存取 Collection 成員物件的方式。

Stream 物件如同建構者模式，具備許多可以使用「方法鏈結 (method chaining)」的方法，語法相當流暢 (fluent)，如：

1. `filter()` 方法：接受實作 Predicate 介面的參考物件，會對流過的集合物件成員使用 Predicate 介面的 `test()` 方法進行篩選，符合的 (return true) 才能流到下一個方法。
2. `forEach()` 方法：同 Iterable 介面的 `forEach()` 方法。接受實作 Consumer 介面的參考物件，會對每一個流入的集合物件成員操作 `accept()` 方法。

🔍 範例

```
1 public static void test2() {
2     List<Person> pl = createPersonList();
3     pl.stream()
4         .filter( p -> p.getAge() >= 23 && p.getAge() <= 65 )
5         .forEach( p -> System.out.println(p) );
6 }
```

🔊 說明

| | |
|---|--|
| 4 | 過濾成員，只有滿足 <code>getAge ()>=23</code> 且 <code>getAge ()<=65</code> 的 Person 物件才能通過篩選，進到下一個方法。 |
| 5 | 對每個通過 <code>filter()</code> 方法的 Person p 使用 <code>System.out.println(p)</code> 。 |

也可以將 Lambda 表示式改用參考變數的方式呈現，增加程式的重複使用性：

範例

```

1 public static void test3() {
2     List<Person> pl = createPersonList();
3     Predicate<Person> criteria = p -> p.getAge() >= 23 && p.getAge() <= 65;
4     Consumer<Person> action = p -> System.out.println(p);
5     pl.stream()
6         .filter(criteria)
7         .forEach(action);
8 }
```

15.3.2 Stream API

Java 8 新增了 Stream API：

1. 使用套件 `java.util.stream`。
2. 中文意思是「水流」，代表一連串的物件成員，可以將多種鏈結方法(chaining methods)套用在所有成員。
3. Collection 和 Stream 兩個物件都有成員，看似接近，其實有大區別：
 - Collection 介面依成員物件的不同特性(如 List、Set、Queue)不同而提供不同的管理和存取方式。
 - Stream 介面「沒有」提供直接存取「特定成員」的方式，只是以「宣告式」的描述做法，告知即將對 Stream 的來源(通常是 Collection)進行各種操作。

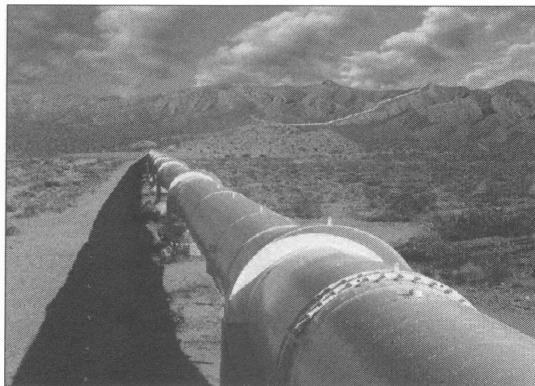


❖ 圖 15-4 水流(stream)示意圖

Stream 特性

特性是：

1. Stream 成員一旦被使用過，就不能再被使用（水流不能回頭），因此是 immutable(不可改變)。
2. Stream 介面定義的鏈結方法 (chaining methods) 的作用方式，可以是：
 - 連續的 (serial / sequential)，此為預設。
 - 平行的 (parallel)，將使用多執行緒支援。
3. Stream 介面定義的鏈結方法，也稱為「管線操作 (pipeline operations)」。概念上如同將水流 (stream) 約束在管線 (pipeline) 裡流動，管線可以視為需要對水流做操作。



❖ 圖 15-5 管線 (pipeline) 示意圖

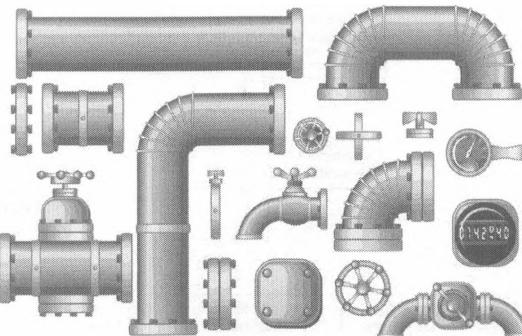
管線操作 (pipeline operations) 的特性

對於管線 (pipeline) 的操作：

1. Stream 在管線裡傳輸。
2. 管線區分多段，定義來源 (source) 後，每一段代表一個作業 (operation)。可以分成：
 - 來源 (Source)，通常是 Collection 物件、檔案、Stream 物件等。
 - 中間作業 (Intermediate Operation)，可以零或多個。
 - 終端作業 (Terminal Operation)，只有一個。
 - 短路型終端作業 (Short-Circuit Terminal Operation)，只有一個。
3. Java 會順著每段管線一路向下執行，但會先確認終端作業方式才會回頭要求 Stream 開始輸送資料，我們形容這種情況為「lazy」：只在開始執行時才要求輸送資料。

4. 若搭配「短路型終端作業」，一旦滿足終端作業定義條件馬上停止輸送資料，相較傳統迴圈處理，就有效能上的優勢。

這些來自 Stream 界面的操作方法，可以想像成管線的各種元件，提供不同功能：



❖ 圖 15-6 各式管線操作元件 (pipeline operations) 示意圖

Stream 的管線操作 (pipeline operations) 都可以使用鏈結方法，如範例「/OCP/src/course/c15/TestPerson.java」的「test4()」方法：

④ 範例

```

1  public static void test4() {
2      List<Person> pl = createPersonList();
3      pl.stream()
4          .filter (p -> p.getAge() >= 23 && p.getAge() <= 65)
5          .filter (p -> p.getEmail().startsWith("j"))
6          .forEach (Person::printPerson);
7  }
```

使用 filter() 方法的概念，和查詢資料庫時使用 SQL 的 where 的語法類似。也可以視情況與需要將邏輯概念合併，如方法 test5()：

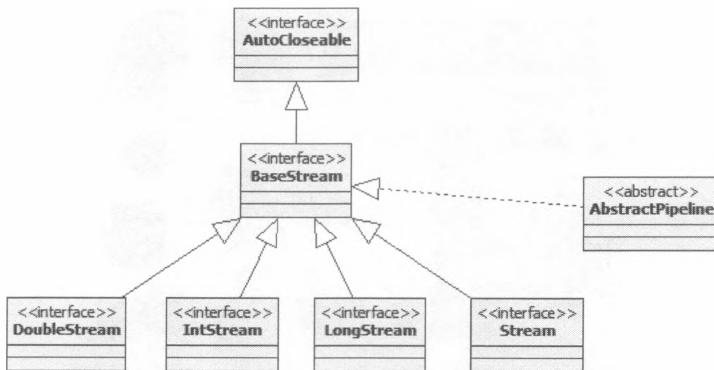
④ 範例

```

1  public static void test5() {
2      List<Person> pl = createPersonList();
3      pl.stream()
4          .filter(p -> p.getAge() >= 23 &&
5                  p.getAge() <= 65 &&
6                  p.getEmail().startsWith("j") )
7          .forEach(Person::printPerson);
8  }
```

15.4 Stream API操作

Stream 家族的 UML 類別圖：



◆ 圖 15-7 Stream 家族

產生 Stream 物件的常見方式，可以參考範例「/OCP/src/course/c15/streamBasic/IntermediateOperationDemo.java」的「createStream()」方法，如下。若想要知道 stream 物件的成員，就使用管線操作方法，如 forEach() 要求列印所有成員：

範例

```

1  public static void createStream() {
2      Stream<String> s1 = Arrays.asList("1", "2", "3", "4").stream();
3      Stream<Integer> s2 = Arrays.asList(1, 2, 3, 4).stream();
4
5      Stream<String> s3 = Stream.of("1", "2", "3", "4");
6      Stream<Integer> s4 = Stream.of(1, 2, 3, 4);
7
8      Stream<String> s5 = Arrays.stream(new String[] {"1", "2", "3", "4"});
9      IntStream s6 = Arrays.stream(new int[] {1, 2, 3, 4});
10
11     s1.forEach(System.out::println);
12 }
  
```

我們透過管線操作 (pipeline operations) 來控制水流 (stream)，這些方法都定義在 `java.util.stream.Stream` 介面下。分類與其常用方法為：

◆ 表 15-1 管線操作 (pipeline operations) 分類

| 分類 | 常用方法 |
|---|---|
| Intermediate Operation
(中間作業) | filter(), map(), peek(), sorted(), flatMap() |
| Terminal Operation
(終端作業) | forEach(), count(), sum(), average(), min(), max(), collect() |
| Short-Circuit Terminal Operation
(短路型終端作業) | findFirst(), findAny(), anyMatch(), allMatch(), noneMatch() |

15.4.1 中間作業

map()

方法宣告：

```
<R> Stream<R> map(Function<? super T, ? extends R> mapper);
```

使用 map() 方法來轉換資料成員。使用 Function 介面的實作物件作為參數，表示要對資料成員套用的方法是「傳入某型別，經過某些流程後，回傳另一種型別」；所以 map() 方法可建立轉換後的「對應 (mapping)」關係，如範例「/OCP/src/course/c15/streamBasic/IntermediateOperationDemo.java」的「testMap()」方法：

範例

```

1  public static void testMap() {
2      Function<Integer, Integer> mapper = n -> 2 * n;
3      Stream<Integer> mapResult =
4          Stream.of(1, 2, 3, 4)
5              .map(mapper);
6
7      Object[] arr = mapResult.toArray();
8      List<Object> list = Arrays.asList(arr);
9      System.out.println(list);
10 }
```

說明

- 2 使用 Lambda 表達式定義需要轉換的工作，本例為將原本的值改成 2 倍。同：
`Function<Integer, Integer> mapper = new Function<Integer, Integer>() {
 @Override
 public Integer apply(Integer t) {
 return 2 * t;
 }
};`

| | |
|---|--|
| 5 | 對 Stream 物件呼叫 map() 方法，並傳入 Function 物件參考，該方法回傳仍是 Stream 物件，同建構者模式。 |
| 7 | Stream 介面提供 toArray() 方法，將 Stream 物件轉成陣列 (Array)。 |
| 8 | 使用 Arrays.asList() 可將陣列轉成 List 物件。 |

結果

```
[2, 4, 6, 8]
```

map() 方法也有相關「基本型別」的擴充版：

- mapToInt()
- mapToLong()
- mapToDouble()

如範例「/OCP/src/course/c15/streamBasic/IntermediateOperationDemo.java」的「testMapToInt()」方法：

範例

```
1 public static void testMapToInt() {
2    ToIntFunction<String> mapper = Integer::parseInt;
3     IntStream mapResult =
4         Stream.of("a1", "a2", "a3")
5             .map(s -> s.substring(1))
6             .mapToInt(mapper);
7     mapResult.forEach(i -> System.out.print(i + ", "));
8 }
```

說明

| | |
|---|--|
| 2 | 將原本的字串成員轉換成 Integer。同：
<code>ToIntFunction<String> mapper = s -> Integer.parseInt(s);</code>
或：
<code>ToIntFunction<String> mapper = new ToIntFunction<String>() {
 @Override
 public int applyAsInt(String value) {
 return Integer.parseInt(value);
 }
};</code> |
| 5 | 使用 map() 方法搭配 Function 介面將成員進行轉換，轉換內容是對字串成員呼叫 substring(1) 方
法，將依序回傳 "1", "2", "3"。 |
| 6 | 使用 mapToInt() 方法來搭配 ToIntFunction 介面，將字串轉成整數 (Integer)，並回傳
IntStream 的物件。 |

結果

```
1, 2, 3,
```

peek()

方法宣告：

```
Stream<T> peek(Consumer<? super T> action);
```

使用 peek() 方法來「窺視(peek)」 Stream 內容：

1. 使用 Consumer 介面，表示需要對資料成員套用的方法是「可以傳入參數，且沒有回傳 (void)」；方法結束後，成員回歸 stream 。
2. peek() 方法主要用於 debug，用於需要了解當 stream 成員經過其他「中間作業」後的變化情況。
3. 若管線作業沒有定義「終端作業」，將不會啓動 peek()，這也反映 stream 物件的「lazy」特質：

```
Stream.of("a", "aa", "aaa", "aaaa")
    .filter(e -> e.length() > 3)
    .peek(e -> System.out.println("Filtered value: " + e)) // debug filter() result
    .map(String::toUpperCase)
    .peek(e -> System.out.println("Mapped value: " + e)) // debug map() result
    .forEach(System.out::println);
```

4. 使用 peek() 也可以更改資料成員，但在平行執行時可能會有「執行緒安全(thread-safe)」的問題，強烈建議不可用來修改資料成員。

如範例「/OCP/src/course/c15/streamBasic/IntermediateOperationDemo.java」的方法「testPeek()」：

範例

```
1 public static void testPeek() {
2     Consumer<Integer> action = System.out::println;
3     Stream<Integer> stream =
4         Stream.of(1, 2, 3, 4)
5             .peek(action);
6     System.out.println("Length: " + stream.toArray().length);
7     /* Streams may be lazy. Computation on the source data is
       performed only when the terminal operation is initiated, and
       source elements are consumed only as needed. */
8 }
```

 **說明**

2 等價於：

```
Consumer<Integer> action = new Consumer<Integer>() {
    public void accept(Integer t) {
        System.out.print(t);
    }
};

或：

Consumer<Integer> action = t -> System.out.println(t);
```

6 行3建立的stream物件並未呼叫終端作業。因為toArray()算是終端作業的一種，如果再拿掉此行，peek()將不會被發動。

使用 sorted () 做基本排序

方法宣告有二種：

1. 將 Stream 成員依「自然順序」重新排序：

```
Stream<T> sorted();
```

2. 將 Stream 成員依「Comparator 定義的順序」重新排序：

```
Stream<T> sorted(Comparator<? super T> comparator);
```

如範例「/OCP/src/course/c15/streamBasic/IntermediateOperationDemo.java」的方法
「testSorted()」：

 **範例**

```
1 public static void testSorted() {
2     List<String> lt = Arrays.asList("a2", "a1", "b1", "c2", "c1");
3     lt.stream()
4         .sorted()
5         .forEach(s -> System.out.print(s + " "));
6     System.out.println();
7     lt.stream()
8         .sorted(String::compareTo)
9         .forEach(s -> System.out.print(s + " "));
10    System.out.println();
11    lt.stream()
12        .sorted((s1, s2) -> s1.compareTo(s2) * -1)
13        .forEach(s -> System.out.print(s + " "));
14 }
```

結果

```
a1 a2 b1 c1 c2
a1 a2 b1 c1 c2
c2 c1 b1 a2 a1
```

說明

| | |
|----|---|
| 4 | 依「自然順序」重新排序。 |
| 8 | 使用字串的 <code>compareTo()</code> 方法排序。 |
| 12 | 使用字串的 <code>compareTo()</code> 方法排序，並將順序倒置。 |

搭配 Comparator 進行多段式排序

Stream 的 `sorted()` 方法可以藉由 Comparator 介面進行多段式的排序，而 Comparator 介面本身也支援使用「方法鏈結」的方式呈現排序條件。常見有三個狀況：

1. 「先比較」成員的特定欄位或特定條件。

```
comparing (Function<? super T, ? extends U> keyExtractor)
```

2. 「再比較」成員的額外欄位或額外條件(若步驟1 比較後不分上下)，視需要而定。

```
thenComparing (Function<? super T, ? extends U> keyExtractor)
```

3. 將比較結果倒置，視需要而定。

```
reversed ()
```

如範例「/OCP/src/course/c15/streamBasic/IntermediateOperationDemo.java」的方法
「`testComparing()`」：

範例

```
1  public static void testComparing() {
2      List<Person> people = Arrays.asList(
3          new Person("Max", 18),
4          new Person("Peter", 23),
5          new Person("Pamela", 23),
6          new Person("David", 12));
7
8      Function<Person, String> getPersonNames = Person::getName;
9      Function<Person, Integer> getPersonAges = Person::getAge;
10     Comparator<Person> comp =
11         Comparator.comparing(getPersonAges)
12             .thenComparing(getPersonNames);
```

```

12     people.stream()
13         .sorted(comp)
14         .forEach(s -> System.out.print(s + ", "));
15
16     people.stream()
17         .sorted(comp.reversed())
18         .forEach(s -> System.out.print(s + ", "));
19 }

```

結果

```

David, Max, Pamela, Peter,
Peter, Pamela, Max, David,

```

flatMap()

方法宣告：

```
flatMap (Function< ? super T, ? extends Stream<? extends R>> mapper )
```

flatMap() 可以使用 Function 介面將 stream 成員的「集合物件」欄位，以 stream 的形式展開 / 呈現，因此有層層展開，將之攤平(flat)的效果。如範例「/OCP/src/course/c15/streamBasic/IntermediateOperationFlatMapDemo.java」的方法「FlatMapDemo1()」：

範例

```

1 public static void FlatMapDemo1() {
2     class Inner {
3         String name;
4         Inner(String name) {
5             this.name = name;
6         }
7         public String toString() {
8             return this.name;
9         }
10    }
11    class Outer {
12        String name;
13        Outer(String name) {
14            this.name = name;
15        }
16        List<Inner> inners = new ArrayList<>();
17        public String toString() {
18            return this.name;

```

```

19     }
20 }
21
22 List<Outer> outers = new ArrayList<>();
23
24 IntStream.range(1, 4)
25     .forEach(i -> outers.add(new Outer("Outer_" + i)));
26
27 outers.forEach(
28     outer ->
29         IntStream.range(1, 4)
30             .forEach(
31                 i ->
32                     outer.inners.add(
33                         new Inner("Inner_" + i + " , from <" + outer.name + ">"))
34                 )
35             )
36 );
37 long memberQuantity =
38     outers.stream()
39         //.peek(System.out::println)
40         .flatMap(outer -> outer.inners.stream())
41         //.peek(System.out::println)
42         .count();
43 }
```

說明

| | |
|-------|--|
| 2~10 | 在方法內宣告內部巢狀類別 Inner。 |
| 11~20 | 在方法內宣告內部巢狀類別 Outer，Outer 的屬性欄位 inners 為 List<Inner>。 |
| 22 | 方法內建立 outers 的 List<Outer> 物件。 |
| 24~25 | 使用 IntStream 的 static 方法 range(1, 4) ，可以決定對 forEach() 操作幾次：forEach 裡的 i 變數，會由 1 開始，到 3 結束，不含 4。
這裡會在 outers 裡建立 3 個 Outer 物件。 |
| 27~36 | 對 outers 裡的每個 Outer 物件的 inners 欄位，各新增 3 個 Inner 物件。 |
| 38 | 將集合物件 outers (為 List<Outer>)，轉成 Stream<Outer> 物件。 |
| 39 | Enable 本行，可以 peek(窺視) 行 38 的結果：
Outer_1
Outer_2
Outer_3 |
| 40 | 套用 flatMap() 方法後，可以取出每個 Outer 物件裡的 inners 欄位的 Inner 物件，再組成 stream，所以共得到 $3 * 3 = 9$ 個成員，都是 Inner 物件。 |

| | |
|--------|--|
| 41 | Enable 本行，可以 peek(窺視) 行 40 的結果：
<pre>Inner_1 , from <Outer_1> Inner_2 , from <Outer_1> Inner_3 , from <Outer_1> Inner_1 , from <Outer_2> Inner_2 , from <Outer_2> Inner_3 , from <Outer_2> Inner_1 , from <Outer_3> Inner_2 , from <Outer_3> Inner_3 , from <Outer_3></pre> |
| 39, 41 | 行 39 和行 41 每次只 enable 一行，較能看到結果。 |

方法「FlatMapDemo2()」則是將檔案「/OCP/src/course/c15/streamBasic/flatMap.txt」裡的所有資料行讀入 Stream<String> 物件，再使用 flatMap 方法，以每行字串裡的空白作為切割符號，攤平成更長的 Stream<String> 物件。我們一樣使用 peek() 方法來窺視每段轉換的成果：

範例

```

1  public static void FlatMapDemo2() throws IOException {
2      Path p = Paths.get("src/course/c15/streamBasic/flatMap.txt")
3          .toAbsolutePath();
4      long matches =
5          Files.lines(p)
6              // .peek(System.out::println)
7              .flatMap(line -> Stream.of(line.split(" ")))
8              // .peek(System.out::println)
9              .filter(word -> word.contains("book"))
10             // .peek(System.out::println)
11             .count();
12     System.out.println("# of Matches: " + matches);
13 }
```

結果

```
# of Matches: 2
```

說明

| | |
|----------|--|
| 5 | 將檔案裡的所有資料行轉成 Stream<String> 物件的字串成員。 |
| 6 | Enable 本行，可以 peek(窺視) 行 5 的結果，也是檔案的原始內容：
my book is cheap
your computer is expensive
his/her book are interesting |
| 7 | 將原本 Stream<String> 物件裡的 3 個字串 (來自檔案的 3 個資料行)，透過 flatMap() 方法，再由空白符號切割出更多小字串，最後組成一個更長的 Stream<String> 物件。語法同：
<pre>Function<String, Stream<String>> mapper =
 new Function<String, Stream<String>>() {
 public Stream<String> apply(String line) {
 return Stream.of(line.split(" "));
 }
 };</pre> |
| 8 | Enable 本行，可以 peek(窺視) 行 7 的結果：
my
book
is
cheap
your
computer
is
expensive
his/her
book
are
interesting |
| 9 | 使用 filter() 方法，留下含有 "book" 的 Stream 成員。 |
| 10 | Enable 本行，可以 peek(窺視) 行 9 的結果：
book
book |
| 6, 8, 10 | 行 6、行 8、行 10 每次只 enable 一行，較能看到結果。 |

15.4.2 終端作業

count()

方法宣告：

```
count();
```

使用 count() 方法來回傳成員個數，如範例「/OCP/src/course/c15/streamBasic/TerminalOperationDemo.java」的「testCount()」方法：

範例

```
1 public static void testCount() {
2     long cnt = Stream.of("Hello", "World").count();
3     System.out.println(cnt);
4 }
```

結果

```
2
```

max() 和 min()

方法宣告：

```
max (Comparator<? super T> comparator);
min (Comparator<? super T> comparator);
```

使用 max() 和 min() 方法搭配 Comparator 的比較邏輯，得到最大值和最小值，如範例「/OCP/src/course/c15/streamBasic/TerminalOperationDemo.java」的「testMaxMin()」方法。必須注意的是，因為 Stream 可能沒有成員，所以回傳的值為 Optional 物件：

範例

```
1 public static void testMaxMin() {
2     Comparator<String> comparator = String::compareTo;
3
4     Optional<String> os = Stream.of("x", "y").max(comparator);
5     System.out.println(os);
6
7     List<String> list = new ArrayList<String>();
8     Optional<String> empty = list.stream().max(comparator);
9     System.out.println(empty);
10
11    OptionalInt oi = Stream.of(1, 2, 3)
12        .mapToInt(i -> i)
13        .min();
14
15    System.out.println(oi);
16 }
```

結果

```
Optional[y]
Optional.empty
OptionalInt[1]
```

說明

| | |
|----|---|
| 2 | 等價於：
<pre>Comparator<String> comparator = new Comparator<String>() { @Override public int compare(String o1, String o2) { return o1.compareTo(o2); } };</pre> 或
<pre>Comparator<String> comparator = (o1, o2) -> o1.compareTo(o2);</pre> |
| 4 | Stream 內為字串成員，使用 Comparator 介面決定排序方式，進而找出最大值。 |
| 7 | 建立空的 List 物件。 |
| 8 | 使用空的 List 物件建立空的 Stream 物件，將回傳內容 empty 的 Optional 物件。 |
| 11 | 1. Stream 呼叫 mapToInt() 將回傳 IntStream 物件，再呼叫 min() 方法，得到 OptionalInt 物件。
2. 整數的比較方式不需要以 Comparator 定義。 |

average() 和 sum()

方法宣告：

```
average();
sum();
```

Stream 成員若要進行如 average() 和 sum() 的數學計算，必須是屬於基礎型別的擴充型 Stream，才能使用相關方法。如：

- DoubleStream
- IntStream
- LongStream

使用 average() 方法取得數字平均值，使用 sum() 方法取得數字總和，如範例「/OCP/src/course/c15/streamBasic/TerminalOperationDemo.java」的「testAverage()」和「testSum()」方法：

範例

```

1 public static void testAverage() {
2     OptionalDouble avg = Stream.of(1, 2, 3, 4)
3         .mapToInt(i -> i)
4         .average();
5
6     System.out.println(avg);
7     System.out.println(avg.getAsDouble());
8
9     IntStream is = Arrays.stream(new int[] {});
10    OptionalDouble emptyAvg = is.average();
11    System.out.println(emptyAvg);
12 }

```

結果

```

OptionalDouble[2.5]
2.5
OptionalDouble.empty

```

說明

- 2 方法average()只有IntStream、LongStream和DoubleStream才具備。Stream<Integer>必須透過mapToInt()轉換為IntStream才能使用。

範例

```

1 public static void testSum() {
2     // Stream.of(1, 2, 3, 4).sum(); //compile error!
3     //IntStream
4     int iSum = Stream.of(1, 2, 3, 4).mapToInt(i -> i).sum();
5     System.out.println(iSum);
6     //LongStream
7     long lSum = Stream.of(1, 2, 3, 4).mapToLong(i -> i).sum();
8     System.out.println(lSum);
9     //DoubleStream
10    double dSum = Stream.of(1, 2, 3, 4).mapToDouble(i -> i).sum();
11    System.out.println(dSum);
12    //Empty Stream
13    int zero = IntStream.of().sum();
14    System.out.println(zero);
15 }

```

結果

```
10
10
10.0
0
```

說明

- 2 方法 `sum()` 只有類別 `IntStream`、`LongStream` 和 `DoubleStream` 才具備。即便 `Stream<Integer>` 也沒有 `sum()` 方法，故本行編譯失敗。

collector() 方法的基本使用方式

方法宣告：

```
collect( Collector<? super T,A,R> collector);
```

使用 `collect()` 方法可以：

1. 將 stream 成員經過若干管線作業後的結果另存為 List、Set 或 Map 等。
2. 可以使用 `Collectors` 類別的 static 方法協助：
 - `stream().collect(Collectors.toList())`
 - `stream().collect(Collectors.toSet())`
 - `stream().collect(Collectors.toMap());`

如 範 例「/OCP/src/course/c15/streamBasic/TerminalOperationCollectDemo.java」的
 「`testCollect()`」方法：

範例

```
1  public static void testCollect() {
2      String[] sArr = new String[] {"jim1", "jim2", "jim1", "jim2"};
3
4      Stream<String> s1 = Stream.of(sArr);
5      Set<String> set = s1.collect(Collectors.toSet());
6      set.stream().forEach(i -> System.out.print(i + ", "));
7      System.out.print("\n-----\n");
8      Stream<String> s2 = Stream.of(sArr);
9      List<String> list = s2.collect(Collectors.toList());
10     list.stream().forEach(i -> System.out.print(i + ", "));
11 }
```

結果

```
jim1, jim2,
-----
jim1, jim2, jim1, jim2,
```

說明

- | | |
|---|----------------------|
| 5 | 轉存為 Set 後，重複的字串自動移除。 |
| 9 | 轉存為 List 後，所有字串保留。 |

collect() 方法搭配 Collectors 類別做進階轉存

使用 collect() 方法了可以將 stream 物件轉存成 Set、List、Map 等集合物件外，也可以透過 Collectors 類別的協助，將要轉存的結果更加精煉，甚至只得到一個數字(double、long、int) 或 String 等。接下來多個範例都會使用到「getPersonList()」方法作為資料來源：

範例

```
1 public static List<Person> getPersonList() {
2     List<Person> persons = Arrays.asList(
3         new Person("Max", 18),
4         new Person("Peter", 23),
5         new Person("Pamela", 23),
6         new Person("David", 12));
7     return persons;
8 }
```

1. 使用 collect() 方法搭配 Collectors.averagingDouble()

```
Collectors.averagingDouble (ToDoubleFunction<? super T> mapper);
```

使用 ToDoubleFunction 定義的方法可以將輸入的物件轉換成 Double。使用 collect() 方法並傳入 Collectors.averagingDouble(ToDoubleFunction)，可以將眾多 stream 成員的特定屬性轉換成 double，並求得平均值，如範例「/OCP/src/course/c15/streamBasic/TerminalOperationCollectDemo.java」的「testAveragingDouble()」方法：

範例

```
1 public static void testAveragingDouble() {
2     Double averageAge =
3         getPersonList().stream()
4             .collect( Collectors.averagingDouble( p -> p.age ) );
5     System.out.println(averageAge); // 19.0
6 }
```

結果

19.0

說明

- | | |
|---|---|
| 4 | getPersonList() 取得的 Person 成員，age 分別是 18、23、23、12，average 是 19。 |
|---|---|

2. 使用 collect() 方法搭配 Collectors.joining()

joining();

使用 collect() 方法並傳入 Collectors.joining()，可以將輸入的字串成員逐一附加(append)一起。如範例「/OCP/src/course/c15/streamBasic/TerminalOperationCollectDemo.java」的「testJoining()」方法：

範例

```

1 public static void testJoining() {
2     List<String> sl = Arrays.asList("a", "b", "c", "d");
3
4     String s1Join = sl.stream().collect(Collectors.joining());
5     System.out.println(s1Join);
6
7     String s2Join = sl.stream().collect(Collectors.joining("-"));
8     System.out.println(s2Join);
9
10    String s3Join = sl.stream().collect(Collectors.joining("-", "/*", "*/"));
11    System.out.println(s3Join);
12 }
```

結果

```
abcd
a-b-c-d
/*a-b-c-d*/
```

說明

- | | |
|----|----------------------------------|
| 4 | 將所有字串成員直接相連(join)。 |
| 7 | 指定字串成員相連時的「連結字串」。 |
| 10 | 指定字串成員相連時的「連結字串」，及連結後的「起頭」和「結尾」。 |

3. 使用 collect() 方法搭配 Collectors.groupingBy()

```
groupingBy(Function<? super T, ? extends K> classifier);
```

使用 collect() 方法並傳入 Collectors.groupingBy()，可以將 stream 的成員做分類，步驟為：

- (1) 使用 Function 以輸入的類別 T 取得另一種類別 K。K 可能是 T 的屬性，或是 T 物件經處理後的某個結果，此為分類的依據：key。
- (2) 以不同的 key 作為分類基準，得到各自成員清單 List<T>，此為相對應的 value。
- (3) 將 key/value 組合後，回傳 Map 物件。

如範例「/OCP/src/course/c15/streamBasic/TerminalOperationCollectDemo.java」的「testGroupingBy()」方法：

範例

```

1  public static void testGroupingBy() {
2      /*
3          Function<Person, Integer> classifier =
4              new Function<Person, Integer>() {
5                  public Integer apply(Person t) {
6                      return t.age;
7                  }
8          }; */
9          Function<Person, Integer> classifier = p -> p.age;
10         Map<Integer, List<Person>> personsByAge =
11             getPersonList().stream()
12                 .collect(Collectors.groupingBy(classifier));
13         // Key:age, Value:personList
14         personsByAge.forEach(
15             (age, personList) ->
16                 System.out.format("age %s: %s\n", age, personList)
17         );
18     }

```

結果

```

age 18: [Max]
age 23: [Peter, Pamela]
age 12: [David]

```

說明

9 | 使用 Function 介面定義的方法由每個 Person 物件中抽出屬性 age 作為分類 (grouping) 基準。

4. 使用 collect() 方法搭配 Collectors.partitioningBy()

```
partitioningBy(Predicate<? super T> predicate);
```

使用 collect() 方法並傳入 Collectors.partitioningBy()，可以將 stream 的成員依 Predicate 定義的方法區分二類，亦即滿足 (true) 和不滿足 (false)，步驟為：

- (1) 使用 Predicate 以輸入的物件 <T> 進行測試，依測試結果 true/false 區分二類，此為分組的 key。
- (2) 以 true/false 作為分類基準，得到各自成員清單 List<T>，此為相對應的 value。
- (3) 將 key/value 組合後，回傳 Map 物件。

如範例「/OCP/src/course/c15/streamBasic/TerminalOperationCollectDemo.java」的「testPartitioningBy()」方法：

範例

```

1  public static void testPartitioningBy() {
2      Map<Boolean, List<Person>> personsByAge =
3          getPersonList().stream()
4              .collect(Collectors.partitioningBy(s -> s.age > 20));
5      System.out.println("Is age > 20 ?");
6      personsByAge.forEach(
7          (k, v) -> System.out.println(
8              k + ": \t"
9              + v.stream()
10             .map(s -> s.name)
11             .collect( Collectors.joining(","))
12         )
13     );
14 }
```

結果

```
Is age > 20 ?
false: Max,David
true: Peter,Pamela
```

 **說明**

| | |
|-----|--|
| 2~4 | 以 age 是否大於 20 作為分類基準，得到 Map<Boolean, List<Person>>。 |
| 6 | 呼叫 map 的 forEach() 方法，可以逐 key-value 檢視 Map<Boolean, List<Person>>。 |
| 10 | 由 map 的 value : List<Person>，取得每個 Person 的 name，轉換為 List<String>。 |
| 11 | 將每個 name，以「,」區隔後，相連一起。 |

15.4.3 短路型終端作業

短路型終端作業 (Short-Circuit Terminal Operation) 是指在 stream 的所有成員都被接觸 / 處理之前，能因為某些情況而提前終止。

這類作業的方法以「搜尋」為主，搜尋時會以最小成本執行並結束工作。Stream<T> 依搜尋後的結果回傳可以分成兩類：

| 回傳 | 方法 |
|-------------|--|
| boolean | boolean allMatch(Predicate<? super T> predicate); |
| | boolean noneMatch(Predicate<? super T> predicate); |
| | boolean anyMatch(Predicate<? super T> predicate); |
| Optional<T> | Optional<T> findFirst(); |
| | Optional<T> findAny(); |

若是回傳 boolean，則方法都需要傳入 Predicate 介面的參考變數，以 test() 方法的實作內容，作為判斷是否滿足搜尋條件的基準。

allMatch()

方法宣告：

```
boolean allMatch (Predicate<? super T> predicate);
```

使用 allMatch() 方法時：

- 若所有成員都滿足定義在 Predicate 介面的 test() 方法裡的搜尋條件，則回傳 true。
- 一旦發現不滿足條件的成員則直接回傳 false，提前結束搜尋。
- 如果 stream 為空，不會使用 Predicate 介面的方法，直接回傳 true。

如範例「/OCP/src/course/c15/streamBasic/TerminalShortCircuitOperationDemo.java」的「testAllMatch()」方法：

④ 範例

```

1 public static void testAllMatch() {
2     List<String> list = Arrays.asList("jim1", "jim2", "jim3", "jim4");
3     boolean containsJim =
4         list.stream()
5             .allMatch(p -> p.contains("jim"));
6     boolean contains1 =
7         list.stream()
8             .allMatch(p -> p.contains("1"));
9     System.out.println(containsJim + " - " + contains1);
10 }

```

⑤ 結果

true - false

⑥ 說明

| | |
|---|----------------------|
| 5 | 是否所有成員都包含 "jim" 關鍵字？ |
| 8 | 是否所有成員都包含 "1" 關鍵字？ |

noneMatch ()

方法宣告：

```
boolean noneMatch (Predicate<? super T> predicate);
```

使用 noneMatch () 方法時：

- 若所有成員都不滿足定義在 Predicate 介面的 test() 方法裡的搜尋條件，將回傳 true。
- 一旦發現滿足條件者則直接回傳 false，提前結束搜尋。
- 如果 stream 為空，不會使用 Predicate 介面的方法，直接回傳 true。

如範例「/OCP/src/course/c15/streamBasic/TerminalShortCircuitOperationDemo.java」的「testNoneMatch()」方法：

④ 範例

```

1 public static void testNoneMatch() {
2     List<String> list = Arrays.asList("jim1", "jim2", "jim3", "jim4");
3     boolean contains5 =
4         list.stream()
5             .noneMatch(p -> p.contains("5"));
6     System.out.println(contains5);
7 }

```

結果

true

說明

5 是否所有成員都不包含 "5" 關鍵字？

anyMatch()

方法宣告：

```
boolean anyMatch (Predicate<? super T> predicate);
```

使用 anyMatch() 方法時：

1. 找到任何一個成員滿足定義在 Predicate 介面的 test() 方法裡的搜尋條件時，直接回傳 true，結束搜尋。
2. 如果 stream 為空，不會使用 Predicate 介面的方法，直接回傳 false。

如範例「/OCP/src/course/c15/streamBasic/TerminalShortCircuitOperationDemo.java」的「testAnyMatch()」方法：

範例

```
1 public static void testAnyMatch() {
2     boolean lengthOver5 = Stream.of("two", "three", "eighteen")
3         .anyMatch(s -> s.length() > 5);
4     System.out.println(lengthOver5);
5 }
```

結果

true

說明

3 是否有成員字串長度 > 5 ？

findFirst()

方法宣告：

```
Optional<T> findFirst();
```

使用 findFirst() 方法時：

1. 找到 Stream<T> 裡的「第一個」成員即回傳 Optional<T>，然後程式結束。
2. 每次找的結果都會固定，稱為「決定性 (deterministic)」。
3. 沒有成員時，回傳 empty 的 Optional 物件。

如範例「/OCP/src/course/c15/streamBasic/TerminalShortCircuitOperationDemo.java」的「testFindFirst()」方法：

① 範例

```

1  public static void testFindFirst() {
2      Optional<String> val =
3          Stream.of("one", "two")
4              .findFirst();
5      System.out.println(val);
6  }

```

② 結果

```
Optional [one]
```

③ 說明

| | |
|---|------------------------------|
| 4 | 尋找 stream 的第一個成員，永遠都是 "one"。 |
|---|------------------------------|

findAny()

方法宣告：

```
Optional<T> findAny();
```

使用 findAny() 方法時：

1. 找到 Stream<T> 裡的「任何一個」成員即回傳 Optional<T>，然後程式結束。
2. 每次找的結果不一定相同，稱為「非決定性 (nondeterministic)」，特別是「平行執行」的時候。若要得到固定結果，需改用 findFirst()。

如範例「/OCP/src/course/c15/streamBasic/TerminalShortCircuitOperationDemo.java」的「testFindAny()」方法：

① 範例

```

1  public static void testFindAny() {
2      List<String> list = Arrays.asList("jim1", "jim2", "jim3", "jim4");
3      Optional<String> val =

```

```

4     list.stream()
5         .findAny();
6     System.out.println(val);
7 }
```

結果

Optional[jim1]

說明

5 找一個 stream 成員，每一次搜尋的結果不一定相同。

15.5 Stream API和NIO.2

在 Java 8 的類別「`java.nio.file.Files`」裡，新增了一些方法支援 Stream API，讓 NIO.2 也可以使用流暢的語法撰寫。我們將 OCP 考試裡常見的方法蒐錄在範例「`/OCP/src/course/c15/streamBasic/NIO2Demo.java`」，並個別解釋如後。

list()

方法宣告：

```
public static Stream<Path> list(Path dir)
```

該方法會列出 `Path dir` 下的所有檔案，但只在第一層，亦即並不是以「遞迴(recursive)」的方式列出各層的所有檔案和目錄。如範例方法 `testList()`：

範例

```

1 public static void testList() throws IOException {
2     try (Stream<Path> stream =
3             Files.list(Paths.get("src/course/c15"))) {
4         Stream
5             .filter(path -> path.toString().endsWith(".txt"))
6             .forEach(System.out::println);
7     }
8 }
```

結果

`src\course\c15\notJava.txt`

find()

方法宣告：

```
public static Stream<Path> find (
    Path start,
    int maxDepth,
    BiPredicate<Path, BasicFileAttributes> matcher,
    FileVisitOption... options
)
```

該方法可以找出符合條件的目錄或檔案：

1. 以 Path start 為搜尋起始目錄。
2. 以 int maxDepth 所指定的層數進行搜尋，非遞迴搜尋全部。
3. 以 BiPredicate<Path, BasicFileAttributes> matcher 定義的方法作為搜尋條件。
4. 以 FileVisitOption... options 定義其他搜尋條件，假如有需要。

如範例方法 testFind()：

範例

```
1 public static void testFind() throws IOException {
2     try (Stream<Path> stream =
3          Files.find(Paths.get("src"),
4                     4,
5                     (path, attr) -> path.toString().endsWith(".txt"))) {
6         stream.forEach(System.out::println);
7     }
8 }
```

結果

```
src\course\c15\test.txt
src\course\c15\notJava.txt
src\course\c15\streamBasic\flatMap.txt
```

walk()

方法宣告有兩種：

1. 指定要遞迴的層數

```
public static Stream<Path> walk (
    Path start,
    int maxDepth,
    FileVisitOption... options
)
```

2. 不指定層數，要遞迴全部

```
public static Stream<Path> walk (
    Path start,
    FileVisitOption... options
)
```

這兩個 overloading 的方法的概念和 NIO.2 裡的方法相似：

```
Files.walkFileTree (Path start, FileVisitor<T> visitor)
```

可以遞迴「拜訪」相關層級的所有檔案 / 目錄。當要對拜訪的檔案 / 目錄採取「特定動作」時：

- Files.walkFileTree() 方法是由實作 FileVisitor 的物件決定。
- Files.walk() 方法則是開啟 stream 後，再由 stream 的管線 (pipeline) 方法決定，如範例方法 testWalk()：

範例

```
1 public static void testWalk() throws IOException {
2     try (Stream<Path> stream =
3             Files.walk(Paths.get("dir/NIO2Demo"), 4)) {
4         Stream
5             .filter(path -> path.toString().endsWith(".txt"))
6             .forEach(System.out::println);
7     }
8     try (Stream<Path> stream =
9             Files.walk(Paths.get("dir/NIO2Demo"))) {
10        Stream
11            .filter(path -> path.toString().endsWith(".txt"))
12            .forEach(System.out::println);
13    }
14 }
```

說明

3, 9 行 3 定義的 Files.walk() 方法只遞迴 4 層，行 9 則遞迴所有層數。拜訪檔案時，若附檔名是 txt，就將其輸出。

結果：只遞迴初始路徑 dir\NIO2Demo\ 下的 4 層，不含初始路徑

```
dir\NIO2Demo\1\2\3\4.txt
```

結果：遞迴全部層數

```
dir\NIO2Demo\1\2\3\4\5.txt
dir\NIO2Demo\1\2\3\4.txt
```

lines()

方法宣告：

```
public static Stream<String> lines(Path path, Charset cs)
public static Stream<String> lines(Path path) // 預設使用UTF-8 charset.
```

本方法將檔案的內容讀出為 Stream<String>，如範例方法 testLines()：

範例

```
1 public static void testLines() throws IOException {
2     try (Stream<String> stream =
3             Files.lines(Paths.get("data.txt"))) {
4         Stream
5             .map(String::toLowerCase)
6             .forEach(System.out::println);
7     }
8 }
```

結果

```
txt1
txt2
txt3
txt4
```

Files.lines() 和 Files.readAllLines() 兩者處理過程類似，但 readAllLines() 方法把所有檔案內容一次載入 JVM，回傳 List<String>；而 lines() 方法回傳 Stream<String>，天性是 lazy 的，搭配管線作業只會處理需要的內容，會有較好的效能。

此外，Java 8 開始，類別 BufferedReader 也新增了方法 lines() 回傳 Stream<String>：

```
public Stream<String> lines()
```

如範例方法 testNewBufferedReader()：

範例

```
1 public static void testNewBufferedReader() throws IOException {
2     try (BufferedReader reader =
3             Files.newBufferedReader(Paths.get("data.txt"))) {
4         Reader
5             .lines()
6             .map(String::toLowerCase)
7             .forEach(System.out::println);
8     }
9 }
```

結果

```
txt1
txt2
txt3
txt4
```

必須注意的是，Stream 也有實作 AutoCloseable 介面，所以可以放在 try-with-resource 的程式區塊裡。當 Stream 用在 Collection 時不需要特別在使用結束後關閉；但搭配 NIO.2 開啓檔案資源就必須在結束時主動關閉，或交由 try-with-resource 架構處理。

15.6 Stream API操作平行化

15.6.1 平行化的前提

Stream 物件特色是：

1. 無法更改內容 (immutable)。一旦更改都將回傳新物件，或是拋出 Exception。
2. 無法重複使用。要使用就必須再產生新物件。
3. 可以使用：

- 循序處理 (sequential)
- 平行處理 (parallel)

Stream API 的使用，支援如建構者設計模式般的「流暢 (fluent)」撰寫風格。比較範例「/OCP/src/course/c15/streamParallel/Difference.java」裡幾段程式碼的差異：

範例

```

1 public static void imperativeProgramming() {
2     double sum = 0;
3     for (Employee e : getEmployees()) {
4         if (e.name.startsWith("Jim") && e.salary >= 1500) {
5             e.show();
6             sum += e.salary;
7         }
8     }
9     System.out.print(sum);
10 }
```

這樣寫法，稱之為「指令式編程 (Imperative Programming)」，它的特色是：

- 迴圈必須經歷所有集合成員。
- 知道程式做了哪些事 (how)，但目的 (what) 不是很清楚。
- 迴圈中必須有其他變數，如 sum。
- 不容易以平行處理提高效能。

範例

```

1 public static void streamingProgramming() {
2     double sum =
3         getEmployees().stream()
4             .filter(e -> e.name.startsWith("Jim"))
5             .filter(e -> e.salary >= 1500)
6             .peek(e -> e.show())
7             .mapToDouble(e -> e.salary)
8             .sum();
9     System.out.print(sum);
10 }
```

這樣寫法，稱之為「流暢式編程 (Streaming Programming)」，它的特色是：

- 程式本身清楚陳述目的 (what)。
- 不需額外變數。

3. 可藉由「懶人 (lazy) 優化機制」提升效能。
4. 可平行處理提高效能。

至於這樣的做法：

範例

```

1 public static void streamingProgramming2() {
2     Stream<Employee> s1 = getEmployees().stream();
3     Stream<Employee> s2 = s1.filter(e -> e.name.startsWith("Jim"));
4     Stream<Employee> s3 = s2.filter(e -> e.salary >= 1500);
5     Stream<Employee> s4 = s3.peek(e -> e.show());
6     DoubleStream s5 = s4.mapToDouble(e -> e.salary);
7     double sum = s5.sum();
8     System.out.print(sum);
9 }
```

失去了 stream 管線操作的好處，如懶人優化法和平行化處理，因此不建議這樣做。

15.6.2 平行化的做法

基本原則

Stream 的管線操作可以使用平行 (parallel) 處理，亦即啟動多執行緒來減少執行時間，此時：

1. 建議硬體應具備多核心 CPU 或 GPU。
2. 底層使用 Fork/Join 架構。但不建議開發者直接使用，應使用高階 API。
3. 有許多因素可以影響平行執行的加速效果，如資料大小、拆解方法、結果聚合方式、CPU 核心數等。因此平行處理不是每次都比循序處理快。
4. 可以藉由以下方式啟動：

- 由 Collection (集合物件) 發動，使用 **parallelStream()** 方法：

```

public static void parallelStreamingFromCollection() {
    double sum =
        getEmployees()
        .parallelStream()
        .filter(e -> e.name.startsWith("Jim"))
        .filter(e -> e.salary >= 1500)
        .peek(e -> e.show())
        .mapToDouble(e -> e.salary)
```

```

        .sum();
    System.out.print(sum);
}

```

- 由 Stream (串流物件) 發動，使用 **parallel()** 方法：

```

public static void parallelStreamingFromStream() {
    double sum =
        getEmployees().stream()
            .filter(e -> e.name.startsWith("Jim"))
            .filter(e -> e.salary >= 1500)
            .peek(e -> e.show())
            .mapToDouble(e -> e.salary)
            .parallel()
            .sum();
    System.out.print(sum);
}

```

- 未呼叫 **.parallel()** 時，預設 **.sequential()**。
- 由 Stream (串流物件) 發動平行化處理時，若要各段管線操作都可以平行化，必須在「尾端」呼叫。
- 過程中不可以修改來源物件 (如 Collection)。

管線操作的變數必須是「沒有狀態 (stateless)」

以下範例無法平行化加速。因為變數 **blockList** 有狀態 (成員持續增加)，導致多執行緒無法分頭進行：

範例

```

1 public static void statefullStreaming() {
2     List<Employee> eList = getEmployees();
3     List<Employee> blockList = new ArrayList<>();
4     eList.parallelStream()
5         .filter(e -> e.name.startsWith("Jim"))
6         .forEach(e -> blockList.add(e));
7 }

```

若有這類需求，可以改用 **collect()** 方法搭配 **Collectors.toList()**，讓 Java 視需要自動調度，例如：何時建立 List 物件、何時新增成員、何時 merge 成員等，所以程式設計師無需介入物件狀態的維護：

範例

```

1 public static void statelessStreaming() {
2     List<Employee> eList = getEmployees();
3     List<Employee> nonblockList =
4         eList.parallelStream()
5             .filter(e -> e.name.startsWith("Jim"))
6             .collect(Collectors.toList());
7 }
```

平行處理可能讓結果不同

對 streams 發動平行處理，大部分結果都會固定，這也符合了「決定性演算法 (Deterministic Algorithm)」；亦即只要輸入相同參數，無論執行幾次結果都會相同，使用 sum() 就是很好的一個例子。因為方法本身無關乎平行處理時的順序先後，因此結果一定相同：

範例

```

1 public static void checkParallelResultOfSum() {
2     List<Employee> eList = getEmployees();
3     double r1 = eList.stream()
4         .filter(e -> e.name.startsWith("Jim"))
5         .mapToDouble(Employee::getSalary)
6         .sequential()
7         .sum();
8     double r2 = eList.stream()
9         .filter(e -> e.name.startsWith("Jim"))
10        .mapToDouble(Employee::getSalary)
11        .parallel()
12        .sum();
13     System.out.println(r1 == r2);
14 }
```

至於 findAny() 方法，因為只要找出一個就結束，因此平行化處理時，很有可能每次得到不同結果：

範例

```

1 public static void checkParallelResultOfFindAny() {
2     List<Employee> eList = getEmployees();
3     Optional<Employee> e1 = eList.stream()
4         .filter(e -> e.name.startsWith("Jim"))
5         .sequential()
```

```

6     .findAny();
7     Optional<Employee> e2 = eList.stream()
8         .filter(e -> e.name.startsWith("Jim"))
9         .parallel()
10        .findAny();
11     System.out.println(e1.get().id == e2.get().id);
12 }
```

15.6.3 Reduction 操作

Reduction 的基礎操作

Reduction Operation，字面翻譯為「歸納或簡化操作」，是指：

- 接受一連串項目 (items) 的輸入。
- 將這些輸入的項目 (items)，經由逐一、反覆使用某結合功能 (combining function) 後得到單一結果。因為過程會讓原先的輸入項目 (items) 逐漸減少，故名「reduce」。

Stream 物件使用「reduce()」方法達成這個目的。以整數的 sum() 方法為例。若將 sum() 方法改以 reduction 的概念實作，就是

1. 以數字「0」作為基礎值 (base value)。
2. 使用運算子「+」作為結合功能 (combining function)：

則原先的求總和：

```
sum = a1 + a2 + ... + an
```

概念上也可以這樣表達：

```
sum = (((0 + a1) + a2) + ...) + an
```

因為我們目前關注在「整數」的加總，因此可以使用介面 IntStream 的 reduce() 方法：

```
int reduce(int identity, IntBinaryOperator op);
```

介面 IntBinaryOperator 的定義是：

```

@FunctionalInterface
public interface IntBinaryOperator {
    int applyAsInt(int left, int right);
}
```

所以，若將整數的 sum() 以 IntStream 介面的 reduce() 方法來處理，搭配 Lambda 表示式後可以如下：

```
.reduce(0, (a, b) -> a + b);
```

或是

```
.reduce(0, ((sum, element) -> sum + element));
```

完整範例是「/OCP/src/course/c15/streamParallel/ParallelDemo.java」的「testReduceInSequential()」方法：

範例

```
1 public static void testReduceInSequential() {
2     int result =
3         IntStream
4             .rangeClosed(1, 4)
5             .reduce(0, ((sum, element) -> sum + element));
6     System.out.println("Result = " + result);
7 }
```

結果

```
Result = 10
```

關於行 4 的「rangeClosed(start, end)」方法，過去在「/OCP/src/course/c15/streamBasic/IntermediateOperationFlatMapDemo.java」的方法「FlatMapDemo1()」中，也曾使用過類似的「range(start, end)」

兩者都是介面 IntStream 的 static 方法，差別在於參數 end 是否被包含(inclusive/exclusive)。

- 使用 rangeClosed() :

```
public static IntStream rangeClosed(int startInclusive, int endInclusive) {...}
因為包含，所以等同於：
for (int i = startInclusive; i <= endInclusive ; i++) {
    //.....
}
```

- 使用 range() :

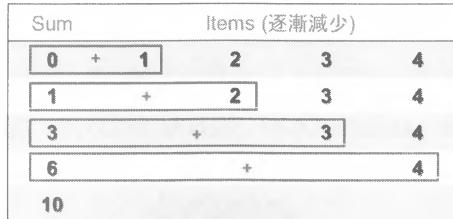
```
public static IntStream range(int startInclusive, int endExclusive) {...}
因為不包含，所以等同於：
```

```

for (int i = startInclusive; i < endExclusive ; i++) {
    //.....
}

```

圖解這樣的概念與步驟：



❖ 圖 15-8 以 reduction 概念實作 sum() 方法的分解示意圖

Reduction 的概念，除了 sum() 之外，經常使用的 max() 和 min() 也可以套用。此外，也可以使用「方法參照(method reference)」讓程式碼更簡潔(compactly)，如方法「testReduceWithCompactly()」：

範例

```

1 public static void testReduceWithCompactly() {
2     int sum =
3         IntStream
4             .rangeClosed(1, 4)
5             .reduce(0, Integer::sum);
6     System.out.println("sum = " + sum);
7     int max =
8         IntStream
9             .rangeClosed(1, 4)
10            .reduce(0, Integer::max);
11     System.out.println("max = " + max);
12     int min =
13         IntStream
14             .rangeClosed(1, 4)
15             .reduce(0, Integer::min);
16     System.out.println("min = " + min);
17 }

```

結果

```

sum = 10
max = 4
min = 0

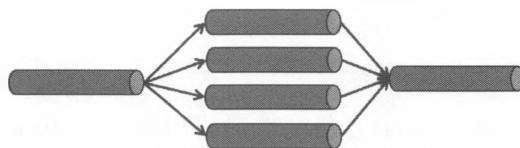
```

Reduction 的平行操作

若「結合功能 (combining function)」是「可組合的 (associative)」，亦即個別項目沒有特定關係，其順序不影響結果，則可以使用平行化處理，如 sum()、min()、max()、average()、count() 等。若否，使用 reduce() 將得到錯誤結果。其中的 count()，其實就是 sum() 的小變形：

```
.map(item -> 1).sum()
```

以管線操作的概念來看，就是將原本的一根管線予以分流，加快處理速度：



◆ 圖 15-9 管線操作的平行化處理

以下列方法「testReduceInParallel()」將 reduce() 方法加上平行化處理：

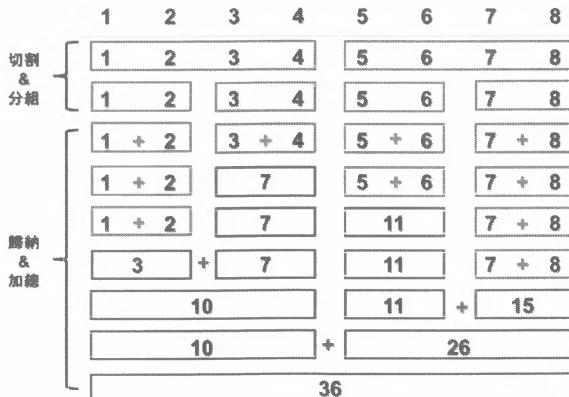
範例

```
1 public static void testReduceInParallel() {
2     int result =
3         IntStream
4             .rangeClosed(1, 8)
5             .parallel()
6             .reduce(0, (sum, element) -> sum + element);
7     System.out.println("Result = " + result);
8 }
```

結果

```
Result = 36
```

管線操作的平行化處理，底層是使用 Fork/Join 架構，因此會先將所有參與加總的整數進行「切割 & 分組 (decomposition)」；加上 reduce() 方法的處理架構，所有整數會逐漸「歸納 & 加總 (merging)」，所以可以加速得到結果：



◆ 圖 15-10 reduce() 平行化處理分解示意圖

平行化處理的注意事項

1. 平行處理效能不一定比較快，有時甚至會比循序處理慢。必須要有硬體支援，如多核 CPU 和 GPU。
2. 平行處理必須考量「最初拆解」、「最終合併」的做法是否合適。中間作業如 filter() 也會影響拆解和合併的效能。
3. 因為自動「開箱/裝箱(boxing/unboxing)」會降低執行效率，直接使用基本型別 (primitive) 的 streams，如 IntStream、LongStream、DoubleStream 會有比較好的效能表現。

15.7 認證考試命題範圍

依據甲骨文公布的考試範圍 (https://education.oracle.com/java-se-8-programmer-ii/pexam_1Z0-809)，和本章相關的主題有：

Generics and Collections

1. Collections Streams and Filters.
2. Iterate using **forEach** methods of Streams and List.
3. Describe Stream interface and Stream pipeline.
4. Filter a collection by using **lambda** expressions.
5. Use **method references** with Streams.

Java File I/O(NIO.2)

1. Use Stream API with NIO.2.

Java Concurrency

1. Use **parallel** Streams including **reduction**, **decomposition**, **merging** processes, **pipelines** and performance.

Java Stream API

1. Develop code to extract data from an object using **peek()** and **map()** methods including primitive versions of the **map()** method.
2. Search for data by using search methods of the Stream classes including **findFirst**, **findAny**, **anyMatch**, **allMatch**, **noneMatch**.
3. Develop code that uses the **Optional** class.
4. Develop code that uses **Stream** data methods and calculation methods.
5. Sort a collection using **Stream** API.
6. Save results to a collection using the **collect** method and group/partition data using the **Collectors** class.
7. Use **flatMap()** methods in the Stream API.

Java Concurrency

1. Use parallel Fork/Join Framework.

本章擬真試題實戰

考題 1

Given:

```
public static void main(String[] args) {
    List<Integer> list = Arrays.asList(1, 2, 3);
    list.stream().map(i -> i * 2)           // line 1
        .peek(System.out::print)      // line 2
        .count();
}
```

What is the result?

- A. 246
- B. The code produces nothing.
- C. A compilation error occurs at line 1.
- D. A compilation error occurs at line 2.

答案 A

說明 使用 peek() 可以 debug 管線操作的過程。

考題 2

Given:

```
class Skill {
    String name;
    Skill(String name) {
        this.name = name;
    }
}
```

And:

```
public static void main(String[] args) {  
    List<Skill> list =  
        Arrays.asList(  
            new Skill("Java-"),  
            new Skill("Oracle DB-"),  
            new Skill("J2SE-"));  
    Stream<Skill> stream = list.stream();  
    // line  
}
```

Which should be inserted at // line to print Java-Oracle DB-J2SE-?

- A. stream.forEach(System.out::print);
- B. stream.map(a-> a.name).forEach(System.out::print);
- C. stream.map(a-> a).forEachOrdered(System.out::print);
- D. stream.forEachOrdered(System.out::print);

答案 B

考題 3

Given:

```
class FileThreadPrint implements Runnable {  
    String fName;  
    public FileThreadPrint(String fName) {  
        this.fName = fName;  
    }  
    public void run() {  
        System.out.println(fName);  
    }  
}
```

And:

```
public static void main(String[] args) throws Exception {  
    ExecutorService es = Executors.newCachedThreadPool();  
    Stream<Path> files = Files.walk(Paths.get("Projects"));  
    files.forEach(  
        f ->  
        {es.execute(new FileThreadPrint(f.getFileName().toString()));} // line1  
    );
```

```

        es.shutdown();
        es.awaitTermination(5, TimeUnit.DAYS);      // line2
    }
}

```

The "Projects" directory exists and contains a list of files.

What is the result?

- A. The program throws a runtime exception at //line2.
- B. The program prints files names concurrently.
- C. The program prints files names sequentially.
- D. A compilation error occurs at //line1.

答案 B

說明 本題可以正常執行。若將 run() 方法改為：

```

public void run() {
    //System.out.println(fName);
    System.out.println(Thread.currentThread().getName());
}

```

可以知道的確是由不同 Thread 執行：

```

pool-1-thread-1
pool-1-thread-2
pool-1-thread-3

```

考題 4

Given:

```

class Emp {
    String firstName;
    String lastName;
    public Emp(String fn, String ln) {
        firstName = fn;
        lastName = ln;
    }
    public String getFirstName() {
        return firstName;
    }
    public String getLastname() {
        return lastName;
    }
}

```

```
    }
    public String toString() {
        return getFirstName() + "-" + getLastName();
    }
}
```

And:

```
public static void main(String[] args) {
    List<Emp> emp =
        Arrays.asList(
            new Emp("J", "S"),
            new Emp("P", "S"),
            new Emp("P", "W"));
    emp.stream()
        // line1
        .collect(Collectors.toList());
}
```

Which code fragment, when inserted at //line1, sorts the employees in the descending order of firstName and then ascending order of lastName?

- A. .sorted(Comparator
 .comparing(Emp::getFirstName)
 .reversed()
 .thenComparing(Emp::getLastName))
- B. .sorted (Comparator
 .comparing(Emp::getFirstName)
 .thenComparing(Emp::getLastName))
- C. .map(Emp::getFirstName)
 .sorted(Comparator.reverseOrder())
- D. .map(Emp::getFirstName)
 .sorted(Comparator.reverseOrder())
 .map(Emp::getLastName)
 .reserved

答案 A

說明 在 collect() 方法前加上 .peek(System.out::println)，可協助輸出排序狀況。

考題 5

Given:

```
public static void main(String[] args) {
    UnaryOperator<Integer> uo = s -> s * 2;      // line1
    List<Double> list = Arrays.asList(100.0, 200.0);
    list.stream()
        .filter(lv -> lv >= 150)
        .map(lv -> uo.apply(lv))      // line2
        .forEach(s -> System.out.print(s + " "));
}
```

What is the result?

- A. 400.0
- B. 400
- C. A compilation error occurs at line1.
- D. A compilation error occurs at line2.

答案 D

說明

1. UnaryOperator<Integer> 的方法將回傳 Integer。
2. 使用 .map(lv -> uo.apply(lv))，嘗試把 list 內的成員由 Double 轉換成 Integer，造成和 List<Double> list 宣告不合。

考題 6

Given:

```
public static void main(String[] args) {
    List<String> list = Arrays.asList("my", "book", "is", "your", "book");
    Predicate<String> test = s -> {
        int i = 0;
        boolean b = s.contains("book");
        System.out.print(i++ + ":");
        return b;
    };
    list.stream().filter(test).findFirst().ifPresent(System.out::print);
}
```

What is the result?

- A. 0:0:book
- B. 0:1:book
- C. 0:0:0:0:book
- D. 0:1:2:3:4:
- E. A compilation error occurs.

答案 A

說明 找第二次就找到 book，故印出 2 個 0，最後再印出 book。

考題 7

Given:

```
List<Integer> list1 = Arrays.asList(10, 20);  
List<Integer> list2 = Arrays.asList(30, 40);  
//line1
```

Which code fragment, when inserted at line1, prints 10 20 30 40?

- A. Stream
 - .of(list1, list2)
 - .flatMap(list -> list.stream())
 - .forEach(s -> System.out.print(s + " "));
- B. Stream
 - .of(list1, list2)
 - .flatMap(list -> list.intStream())
 - .forEach(s -> System.out.print(s + " "));
- C. list1.stream()
 - .flatMap(list2.stream())
 - .flatMap(e1 -> e1.stream())
 - forEach(s -> System.out.println(s + " "))
- D. Stream.of(list1, list2)
 - flatMapToInt(list -> list.stream())
 - .forEach(s -> System.out.print(s + " "));

答案 A

說明 選項 A：將 Stream<List<Integer>> 攤平為 Stream<Integer>，所以可以依序印出 10 20 30 40。選項 B：List 無此方法：intStream()。選項 C：無法通過編譯。選項 D：無法將 Stream<List<Integer>> 攤平為 IntStream，與 Stream<Integer> 不合。

考題 8

Given:

```
public static void main(String[] args) {
    List<String> list = Arrays.asList("blue", "green", "yellow");
    Predicate<String> test = n -> {
        System.out.println("finding...");
        return n.contains("blue");
    };
    list.stream().filter(c -> c.length() > 4).allMatch(test);
}
```

What is the result?

- A. finding...
- B. finding... finding...
- C. finding... finding... finding...
- D. compilation error

答案 A

說明 allMatch() 是短路型終端作業，必須全部滿足才 return true。一旦發現一個不合，馬上中止搜尋，並 return false。本題只找一次就足以判定結果為 false，直接結束。

考題 9

Given:

```
class Product {
    int seq;
    int qty;
    public Product(int id, int price) {
        this.seq = id;
        this.qty = price;
    }
}
```

```
    public String toString() {
        return seq + ":" + qty;
    }
}
```

And:

```
public static void main(String[] args) {
    List<Product> ps =
        Arrays.asList(
            new Product(1, 10),
            new Product(2, 20),
            new Product(3, 30));
    Product p =
        ps.stream()
        .reduce(new Product(4, 0),
            (p1, p2) -> {
                p1.qty += p2.qty;
                return new Product(p1.seq, p1.qty);
            });
    ps = new ArrayList<Product>(ps);
    ps.add(p);
    ps.stream()
        .parallel()
        .reduce((p1, p2) -> p1.qty > p2.qty ? p1 : p2)
        .ifPresent(System.out::println);
}
```

What is the result?

- A. 2:30
- B. 4:0
- C. 4:60
- D. 4:60
3:20
2:30
1:10
- E. The program prints nothing.

答案 C

說明

1. 第一次使用 reduce，產出的 Product 其 seq=4，qty 為所有產品的加總。故得到 new Product(4, 60)。
2. Arrays.asList(…) 取得的 List，無法新增或刪除成員，必須經由 ps = new ArrayList<Product>(ps); 來重新建構物件才可以。
3. 第二次使用 reduce，產出的 Product 必須是 qty 最大的，故選擇 new Product(4, 60)。

考題 10

Given:

```
public static void main(String[] args) throws IOException {
    BufferedReader brCopy = null;
    try (BufferedReader br =
        new BufferedReader(new FileReader("test.txt"))) { // line1
        br.lines().forEach(System.out::println);
        brCopy = br; // line2
    }
    brCopy.ready(); // line3;
}
```

Assume that the ready method of the BufferedReader, when called on a closed BufferedReader, throws an exception, and test.txt is accessible and contains some text.

What is the result?

- A. A compilation error occurs at line3.
- B. A compilation error occurs at line1.
- C. A compilation error occurs at line2.
- D. The code prints the content of the test.txt file and throws an exception at line 3.

答案 D

說明 Resource 在 line3 之前已關閉，呼叫時會拋出 java.io.IOException: Stream closed。

考題 11

Given:

```
class Student {  
    String major, name, home;  
    public Student(String name, String course, String city) {  
        this.major = course;  
        this.name = name;  
        this.home = city;  
    }  
    public String getMajor() {  
        return major;  
    }  
    public String toString() {  
        return major + ":" + name + ":" + home;  
    }  
}
```

And:

```
public static void main(String[] args) {  
    List<Student> stds = Arrays.asList(  
        new Student ("Jason", "Java ME", "Chicago"),  
        new Student ("Helen", "Java SE", "Houston"),  
        new Student ("Mark", "Java ME", "Chicago"));  
    stds.stream()  
        .collect(Collectors.groupingBy(Student::getMajor))  
        .forEach((key, val) -> System.out.println(key));  
}
```

What is the result?

A. [Java SE: Helen:Houston]

[Java ME: Jason:Chicago, Java ME: Mark:Chicago]

B. Java ME

Java SE

C. [Java ME: Jason:Chicago, Java ME: Mark:Chicago]

[Java SE: Helen:Houston]

D. A compilation error occurs.

答案 B

考題 12

Given:

```
class Home {
    String city = "New York";
    public String getCity() {
        return city;
    }
    public String toString() {
        return city;
    }
}
```

And:

```
public static void main(String[] args) {
    Home home = null;
    Optional<Home> addr = Optional.ofNullable(home);
    String s = (addr.isPresent()) ? addr.get().getCity() : "City Not available";
    System.out.println(s);
}
```

What is the result?

- A. New York
- B. City Not available
- C. null
- D. A NoSuchElementException is thrown at run time.

答案 B

考題 13

Given:

```
public static void main(String[] args) {
    List<String> list = Arrays.asList("Jim", "Duke", "Alice", "Tom");
    System.out.println(
        // line1
    );
}
```

Which code fragment, when inserted at line1, enables the code to print the quantity of string elements whose length is greater than 3?

- A. `list.stream().filter(x -> x.length()>3).count()`
- B. `list.stream().map(x -> x.length()>3).count()`
- C. `list.stream().peek(x -> x.length()>3).count().get()`
- D. `list.stream().filter(x -> x.length()>3).mapToInt(x -> x).count()`

答案 A

說明 D 選項須改為：

```
listVal.stream().filter(x -> x.length()>3).mapToInt(x -> 1).count()
```

考題 14

Which statement is true about `java.util.stream.Stream`?

- A. A stream cannot be consumed more than once.
- B. The execution mode of streams can be changed during processing.
- C. Streams are intended to modify the source data.
- D. A parallel stream is always faster than an equivalent sequential stream.

答案 A

說明

A. 執行兩次會出錯，如下：

```
public static void main(String[] args) {  
    IntStream stream = IntStream.of(1, 2);  
    stream.forEach(System.out::println); // 第一次  
    stream.forEach(System.out::println); // 第二次  
}
```

將拋出會拋出例外：

```
java.lang.IllegalStateException: stream has already been operated upon or closed
```

B. `parallel` 和 `sequential` 在執行中無法變更。

C. 不該如此。

D. 不一定。

考題 15

Given:

```
public static void main(String[] args) throws IOException {
    Path file = Paths.get("test.txt");
    // line1
}
```

Assume the test.txt is accessible.

Which code fragment can be inserted at //line1 to enable the code to print the content of the test.txt file?

- A. List<String> fc = Files.list(file);
fc.stream().forEach(s -> System.out.println(s));
- B. Stream<String> fc = Files.readAllLines(file);
fc.forEach(s -> System.out.println(s));
- C. List<String> fc = readAllLines(file);
fc.stream().forEach(s -> System.out.println(s));
- D. Stream<String> fc = Files.lines(file);
fc.forEach(s -> System.out.println(s));

答案 D

說明 選項 A : Files.list() 回傳 Stream<String>。選項 B : Files.readAllLines() 回傳 List<String>。選項 C : 缺少 Files。選項 D : Files.lines() 回傳 Stream<String>。

考題 16

Given:

```
class Staff {
    private String eid;
    private Integer age;
    Staff(String eid, Integer age) {
        this.eid = eid;
        this.age = age;
    }
    public Integer getAge() {
        return age;
    }
}
```

```
    public String getEid() {  
        return eid;  
    }  
}
```

And:

```
public static void main(String[] args) {  
    List<Staff> list = Arrays.asList(new Staff("Duke", 25),  
                                    new Staff("John", 60),  
                                    new Staff("Jim", 51));  
    Predicate<Staff> testAge = s -> s.getAge() > 50;           // line n1  
    list = list.stream().filter(testAge).collect(Collectors.toList());  
    Stream<String> eids = list.stream().map(Staff::getEid);      // line n2  
    eids.forEach(n -> System.out.print(n + " "));  
}
```

What is the result?

- A. Duke John Jim
- B. John Jim
- C. A compilation error occurs at line1.
- D. A compilation error occurs at line2.

答案 B

考題 17

Given:

```
interface TechFilter extends Predicate<String> {  
    public default boolean test(String str) {  
        return str.equals("Java");  
    }  
}
```

And:

```
public static void main(String[] args) {  
    List<String> strs = Arrays.asList("Java", "Java FX", "Java SE");  
    Predicate<String> p1 = s -> s.length() > 3;  
    Predicate<String> p2 = new TechFilter() {           // line1
```

```

public boolean test(String s) {
    return s.contains("Java");
}
};

long qty =
    strs
        .stream()
        .filter(p1)
        .filter(p2)      // line2
        .count();
System.out.println(qty);
}

```

What is the result?

- A. 2
- B. 3
- C. A compilation error occurs at line1.
- D. A compilation error occurs at line2.

答案 B

考題 18

Given:

```

public static void main(String[] args) {
    Stream<Path> paths =
        Stream.of(Paths.get("ocp.doc"),
                  Paths.get("ocp.txt"),
                  Paths.get("ocp.xml"));

    paths
        .filter(s -> s.toString().endsWith("txt"))
        .forEach(
            s -> {
                try {
                    Files.readAllLines(s).stream().forEach(System.out::println); // line1
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        );
}

```

Supposed "ocp.doc", "ocp.txt" and "ocp.xml" files are accessible and contain text. What is the result?

- A. The program prints the content of "ocp.txt" file.

- B. The program prints:

Exception

<<The content of the ocp.txt file>>

Exception

- C. Compilation error at line1.

- D. The program prints the content of these three files.

答案 A

考題 19

Given:

```
public static void main(String[] args) {  
    Stream<List<String>> s1 =  
        Stream.of((Arrays.asList("1", "Duke"), Arrays.asList("2", null));  
    Stream<String> s2 = s1.flatMap((x) -> x.stream());  
    s2.forEach(System.out::print);  
}
```

What is the result?

- A. 1Duke2null

- B. 12

- C. A NullPointerException is thrown at run time.

- D. A compilation error occurs.

答案 D

說明 須改為：Stream<String> s2 = s1.flatMap((x) -> x.stream()); 才能通過編譯。執行後，答案為 A。

考題 20

Given:

```
public static void main(String[] args) throws IOException {
    String home = System.getProperty("user.home");
    Stream<Path> files = Files.walk(Paths.get(home));
    files.forEach(f -> {           // line1
        try {
            Path p = f.toAbsolutePath();      // line2
            System.out.println(f + ":" +
                Files
                    .readAttributes(p, BasicFileAttributes.class)
                    .creationTime());
        } catch (IOException ex) {
            ex.printStackTrace();
        }
    });
}
```

What is the result?

- A. All files and directories under the home directory are listed along with their attributes.
- B. A compilation error occurs at line1.
- C. The files in the home directory are listed along with their attributes.
- D. A compilation error occurs at line2.

答案 A

考題 21

Given:

```
public static void main(String[] args) {
    List<String> list = Arrays.asList("Jim", "Duke", "Mary");
    Function<String, String> apply = s -> " Hi:".concat(s);
    list.stream().map(apply).peek(System.out::print);
}
```

What is the result?

- A. Hi:Jim Hi:Duke Hi:Mary

- B. Jim Duke Mary
- C. The program prints nothing.
- D. Compilation error.

答案 C

說明 Stream 是 lazy 的，必須遇上終端作業 (Terminal Operation) 才會做事。

考題 22

Given:

```
public static void main(String[] args) {  
    List<String> list =  
        Arrays.asList( "100, Duke, HR",  
                      "220, Mary, Sales",  
                      "110, Peter, R&D");  
    list  
        .stream()  
        .filter(s -> s.contains("1"))  
        .sorted()  
        .forEach(System.out::println); // line n1  
}
```

- A. 100, Duke, HR
110, Peter, R&D
- B. A compilation error occurs at line n1.
- C. 100, Duke, HR
110, Peter, R&D
220, Mary, Sales
- D. 100, Duke, HR
220, Mary, Sales
110, Peter, R&D

答案 A

考題 23

Given:

```
class Nation {
    public enum Land {
        ASIA, EUROPE
    }
    String name;
    Land land;
    public Nation(String na, Land land) {
        this.name = na;
        this.land = land;
    }
    public String getName() {
        return name;
    }
    public Land getLand() {
        return land;
    }
}
```

And:

```
public static void main(String[] args) {
    List<Nation> list =
        Arrays.asList(
            new Nation("Japan", Nation.Land.ASIA),
            new Nation("Italy", Nation.Land.EUROPE),
            new Nation("French", Nation.Land.EUROPE));
    Map<Nation.Land, List<String>> lands =
        list.stream()
            .collect(
                Collectors
                    .groupingBy(Nation::getLand,
                        Collectors.mapping(Nation::getName, Collectors.toList())));
    System.out.println(lands);
}
```

What is the output?

- A. EUROPE = [Italy, French], ASIA = [Japan]
- B. {ASIA = [Japan], EUROPE = [Italy, French]}
- C. {EUROPE = [French, Italy], ASIA = [Japan]}
- D. {EUROPE = [French], EUROPE = [Italy], ASIA = [Japan]}

答案 B

說明 一般是：

```
Map<Nation.Land, List<Nation>> lands1 =  
    list.stream()  
        .collect(  
            Collectors.groupingBy(Nation::getLand));
```

本題使用 groupingBy() 的 overloading 版本：

```
.groupingBy(Nation::getLand,  
            Collectors.mapping(Nation::getName, Collectors.toList()))
```

將產出的 Map 物件的 value 型態，由 List<Nation>，轉換為 List<String>。

考題 24

Given:

```
public static void main(String[] args) {  
    List<String> exts = Arrays.asList("XLS", "MPEG4", "JPEG");  
    exts.forEach(e -> System.out.print(e + " "));  
    String exts2 =  
        exts.stream()  
            .filter(s -> s.contains("PEG"))  
            .reduce((e1, e2) -> e1 + e2)  
            .get();  
    System.out.println("\n" + exts2);  
}
```

What is the result?

- A. XLS MPEG4 JPEG
 MPEG4JPEG
- B. XLS MPEG4 MPEG4JPEG
 MPEG4MPEGJ4PEG
- C. MPEG4JPEG
 MPEG4JPEG
- D. The order of the output is unpredictable.

答案 A

考題 25

Given:

```
public static void main(String[] args) {
    IntStream stream = IntStream.of(10, 20, 30);
    IntFunction<Integer> IF = x -> y -> x*y;           // line1
    IntStream newStream = stream.map(IF.apply(15));        // line2
    newStream.forEach(System.out::println);
}
```

Which modification enables the code fragment to compile?

A. Replace //line1 with:

```
IntFunction<UnaryOperator> IF = x -> y -> x*y;
```

B. Replace //line1 with:

```
IntFunction<IntUnaryOperator> IF = x -> y -> x*y;
```

C. Replace //line1 with:

```
BiFunction<IntUnaryOperator> IF = x -> y -> x*y;
```

D. Replace //line2 with:

```
IntStream newStream = stream.map(IF.applyAsInt (15));
```

答案 B

說明

IntFunction 回傳一個 IntUnaryOperator 定義的方法，該方法負責將所有成員 *15 後回傳。
本題可以由方法輸入輸出的觀點來檢視答案。

選項 A：IntFunction：方法輸入 int；UnaryOperator：方法輸入輸出型態未定，故 IntFunction <UnaryOperator> 不能搭配。

選項 B：IntFunction：方法輸入 int；IntUnaryOperator：方法輸入輸出均為 int，故 IntFunction<IntUnaryOperator> 可以搭配。

選項 C：BiFunction：方法輸入有二個參數，且型態未定；IntUnaryOperator：方法輸入輸出均為 int，故 BiFunction < IntUnaryOperator > 不能搭配。

選項 D：若選 B，//line2 可以正常編譯。

考題 26

Given:

```
public static void main(String[] args) {  
    List<Integer> ints = Arrays.asList(11, 22, 8);  
    System.out.println(  
        // line1  
    );  
}
```

Which code fragment must be inserted at //line1 to enable the code to print the maximum number in the ints list?

- A. ints.stream().max(Comparator.comparing(a -> a)).get()
- B. ints.stream().max(Integer::max).get()
- C. ints.stream().max()
- D. ints.stream().map(a -> a).max()

答案 A

說明

錯誤原因：

選項 C、D：支援基本型態的 stream 物件如 IntStream、LongStream、DoubleStream 才可以直接呼叫 max() 方法。D 選項必須改為 ints.stream().mapToInt(a -> a).max()。

選項 B：Stream.max() 傳入的參數在決定要比較的欄位或成員，並非比較的邏輯。

Date/Time API

-
- | 16.1 Java 8 在Date 和Time 相關類別的進步
 - | 16.2 當地日期與時間
 - | 16.3 時區和日光節約時間
 - | 16.4 描述日期與時間的數量
 - | 16.5 認證考試命題範圍

16.1 Java 8在Date和Time相關類別的進步

為何日期 (Date) 和時間 (Time) 重要？

在程式裡，經常有需求必須表現日期 / 時間，或是以之用於計算。如：

1. 取得當地 (locally) 的現在、過去、或未來的日期和時間。
2. 比較兩個時間點的差異，可能用 years, months, days, hours, minutes, seconds 來表示。
3. 不同國家顯示的時差 (time zone)。
4. 日光節約時間 (daylight savings time) 的調整。
5. 描述日期 / 時間區間：
 - 使用 duration 來描述時間 (hours, minutes, seconds) 的區間。
 - 使用 period 來描述日期 (years, months, days) 的區間。
6. 閏年 (leap year) 時 2 月份的天數。
7. 日期 / 時間顯示格式 (format)。

Java 8 之前的日期和時間 API

Java 8 之前常使用 `java.util.Date` 和 `java.util.Calendar` 等類別將日期 & 時間合併表達，不足的地方是：

1. 不支援流暢 (fluent) 的語法，亦即無法以類似 builder pattern 的方式撰寫。
2. 物件實例都是 mutable，且和 lambda 表示式不相容。
3. 非執行緒安全 (thread-safe)。
4. API 種類不多。

Java 8 之後的日期和時間 API

Java 8 之後可以使用不同類別，將日期 & 時間分開表達：

1. 相關類別和方法的使用相當直覺化。
2. 支援流暢 (fluent) 的語法。
3. 物件實例都是 immutable，且相容於 lambda 表示式。
4. 以 ISO 標準定義日期和時間。
5. 執行緒安全 (thread-safe)。

6. API 種類多，且方便開發者自行擴充。
7. `toString()` 方法回傳有意義、可讀性高的說明。

16.2 當地日期與時間

Java 8 使用套件「`java.time`」下的部分 API 來定義「當地(local)」的日期和時間。這裡強調當地，是因為不含「時區(time zone)」的概念：

類別 `LocalDate`

用於儲存 `years`, `months`, `days` 資訊，只有日期，未包含時間。其 `toString()` 方法回傳「`YYYY-MM-DD` (ISO 8601 格式)」。使用 `LocalDate` 類別的屬性和方法，可以取得以下問題的答案：

1. 某個日期屬於過去或未來？
2. 是否是閏年 (leap year) ？
3. 是一週裡面的哪一天？
4. 是一個月裡的哪一天？
5. 下週二是哪一天？

過去常用的日期類別 `java.util.Date` 包含時間，而程式設計人員有時會使用「午夜 12 點 (midnight)」來表現某一天。而但某些時區在「日光節約時間」的那一天是沒有午夜 12 點的，因此造成一些問題，這部分我們在下一節會介紹。

範例「`/OCP/src/course/c16/LocalDateExample.java`」呈現類別 `LocalDate` 的建立方式與常用方法：

範例

```

1  public class LocalDateExample {
2      public static void main(String[] args) {
3          LocalDate now = LocalDate.now();
4          out.println("Now: " + now);
5
6          LocalDate d = LocalDate.of(1995, 5, 23);      // Java's Birthday
7          out.println("Java's Bday: " + d);
8          out.println("Is Java's Bday in the past? " +
9                      d.isBefore(now));

```

```

10      out.println("Is Java's Bday in a leap year? " +
11                  d.isLeapYear());
12      out.println("Java's Bday day of the week: " +
13                  d.getDayOfWeek());
14      out.println("Java's Bday day of the Month: " +
15                  d.getDayOfMonth());
16      out.println("Java's Bday day of the Year: " +
17                  d.getDayOfYear());
18
19      LocalDate nowAfter1Month = now.plusMonths(1);
20      out.println("The date after 1 month: " + nowAfter1Month);
21
22      LocalDate nextMonday =
23          now.with( TemporalAdjusters.next( DayOfWeek.MONDAY ) );
24      out.println("First Monday after now: " + nextMonday);
25  }
}

```

結果

```

Now: 2016-07-08
Java's birthday: 1995-05-23
Is Java's birthday in the past? true
Is Java's birthday in a leap year? false
Java's birthday day of the week: TUESDAY
Java's birthday day of the Month: 23
Java's birthday day of the Year: 143
The date after 1 month: 2016-08-08
First Monday after now: 2016-07-11

```

類別 LocalTime

用於儲存 hours, minutes, seconds, nanoseconds 等一天內的時間資訊，未包含日期，由午夜 12 點 (midnight) 起算。其 `toString()` 方法回傳「HH:mm:ss.SSSS (ISO 8601 格式)」。
 類別 `LocalTime` 用於取得以下問題的答案：

- 何時可以用餐？
- 用餐時間過了嗎？
- 1 小時又 30 分鐘後是幾點？
- 用餐時間還要幾分鐘、幾小時？
- 如何個別使用 `hours` 和 `minutes` 來追蹤時間？

範例「/OCP/src/course/c16/LocalTimeExample.java」呈現類別 LocalTime 的建立方式與常用方法：

範例

```

1  public class LocalTimeExample {
2      public static void main(String[] args) {
3          LocalTime now = LocalTime.now();
4          out.println("Now is: " + now);
5
6          LocalTime nowPlus =
7              now.plusHours(1).plusMinutes(15);
8          out.println("The Time after 1 hour 15 minutes: " + nowPlus);
9
10         LocalTime nowHrsMins =
11             now.truncatedTo(ChronoUnit.MINUTES);
12         out.println("Truncate now to minutes: " + nowHrsMins);
13         out.println("Now is " + now.toSecondOfDay()
14                         + " seconds after midnight");
15         LocalTime lunch = LocalTime.of(12, 5);
16         out.println("Do I miss lunch? " + lunch.isBefore(now));
17
18         long minsUntilLunch =
19             now.until(lunch, ChronoUnit.MINUTES);
20         out.println("Minutes until lunch: " + minsUntilLunch);
21
22         LocalTime bedtime = LocalTime.of(23, 20);
23         long hrsToBedtime = now.until(bedtime, ChronoUnit.HOURS);
24         out.println("How many hours until bedtime? " + hrsToBedtime);
25     }
26 }
```

結果

```

Now is: 23:50:20.468
The Time after 1 hour 15 minutes: 01:05:20.468
Truncate now to minutes: 23:50
Now is 85820 seconds after midnight
Do I miss lunch? true
Minutes until lunch: -705
How many hours until bedtime? 0
```

類別 `LocalDateTime`

類別 `LocalDateTime` 是 `LocalDate` 和 `LocalTime` 的結合。對於事件發生的時間點可以更精準描述：

1. 會議何時召開？
2. 假期何時開始？
3. 若會議展延至週五，將會是何日何時？
4. 如果假期由週一早上 8 點開始，週五下午 5 點結束，一共經歷多少小時？

範例「/OCP/src/course/c16/LocalDateTimeExample.java」呈現類別 `LocalDateTime` 的建立方式與常用方法：

範例

```

1  public class LocalDateTimeExample {
2      public static void main(String[] args) {
3          LocalDate flightDate = LocalDate.of(2016, Month.JULY, 2);
4          LocalTime flightTime = LocalTime.of(21, 45);
5          LocalDateTime flight = LocalDateTime.of(flightDate, flightTime);
6          out.println("Airplane leaves: " + flight);
7
8          LocalDateTime seminarStart =
9              LocalDateTime.of(2016, Month.JULY, 2, 9, 30);
10         out.println("Seminar starts: " + seminarStart);
11         LocalDateTime seminarEnd =
12             seminarStart.plusDays(2).plusHours(8);
13         out.println("Seminar ends: " + seminarEnd);
14
15         long seminarHrs =
16             seminarStart.until(seminarEnd, ChronoUnit.HOURS);
17         out.println("Seminar is: " + seminarHrs + " hours long.");
18     }
19 }
```

結果

```

Airplane leaves: 2016-07-02T21:45
Seminar starts: 2016-07-02T09:30
Seminar ends: 2016-07-04T17:30
Seminar is: 56 hours long.
```

16.3 時區和日光節約時間

16.3.1 時區和日光節約時間的簡介

在開始介紹 Java 8 支援時區及日光節約時間的 API 前，幾個名詞必須先了解：

GMT (Greenwich Mean Time：格林威治標準時間)

十七世紀，英國格林威治皇家天文台為了海上霸權的擴張計畫而進行天體觀測。1675 年舊皇家觀測所正式成立，到了 1884 年決定以通過格林威治的子午線作為劃分地球東西兩半球的經度零度。觀測所門口牆上有一個標誌 24 小時的時鐘，顯示當下的時間；在 1924 年開始，格林威治天文台每小時就會向全世界播報時間。對全球而言，這裡所設定的時間是世界時間參考點，全球都以格林威治的時間作為標準來設定時間，這就是我們熟悉的「格林威治標準時間 (Greenwich Mean Time，簡稱 G.M.T.)」的由來。

UTC (Universal Time Coordinated：國際協調時間)

隨著科技的進步，在西元 1967 年國際度量衡大會把「秒」的定義改成鉻原子進行固定震盪次數的時間。爾後搭配平均太陽時（以格林威治時間 GMT 為準）、地軸運動修正後的新時標、及以「秒」為單位的國際原子時等所綜合精算而成的時間，就產生了「國際協調時間 (Universal Time Coordinated，簡稱 U.T.C.)」。

UTC 計算過程相當嚴謹精密，因此若以「世界標準時間」的角度來說，UTC 比 GMT 來得更加精準。但為了不讓兩者的時間差讓世人混淆，其誤差值必須保持在 0.9 秒以內，若大於 0.9 秒則由位於巴黎的國際地球自轉事務中央局發布「閏秒」，使兩者一致。所以就現行日常生活的使用來說，GMT 與 UTC 的精確度是沒有差別的。

全球 24 個時區的劃分

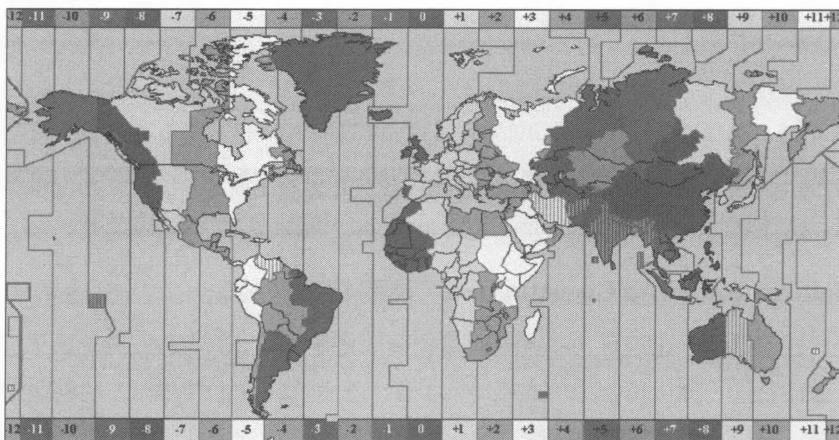
過去世界各地原本各自訂定當地時間，但隨著交通和電訊的發達，各地交流日益頻繁，不同的地方時間，造成許多困擾。

在西元 1884 年的國際會議上制定了全球性的標準時，明定以英國倫敦格林威治這個地方為零度經線的起點，亦稱為「本初子午線」，並以地球由西向東每 24 小時自轉一周 360° ，訂定每隔經度 15° ，時差 1 小時。

每 15° 的經線則稱為該時區的中央經線，將全球劃分為 24 個時區，其中包含 23 個整時區及 180° 經線左右兩側的 2 個半時區。就全球的時間來看，東經的時間比西經要

早，也就是如果格林威治時間是中午 12 時，則中央經線 15° E 的時區為下午 1 時，中央經線 30° E 時區的時間為下午 2 時；反之，中央經線 15° W 的時區時間為上午 11 時，中央經線 30° W 時區的時間為上午 10 時。以台灣為例，台灣位於東經 121° ，換算後與格林威治就有 8 小時的時差。

如果兩人同時從格林威治的 0° 各往東、西方前進，當他們在經線 180° 時，就會相差 24 小時，所以經線 180° 被定為「國際換日線」，由西向東通過此線時日期要減去一日，反之，若由東向西則要增加一日。全球時區的劃分，可以參考下圖：



◆ 圖 16-1 全球時區劃分

日光節約時間

1. 日光節約時間的定義

「日光節約時間 (Daylight Saving Time，簡稱 D.S.T.)」，是希望人們能在天亮較早的夏季，多善用自然光源，以人為的方式將時間提前一小時，使人早睡早起，多多從事戶外活動，充分使用光照資源，並減少照明量(用燈量)來達到節約用電的效果，又稱為「夏令時間 (Summer Time)」。

這個構想於 1784 年由美國班傑明·富蘭克林提出來，1915 年德國成為第一個正式實施夏令日光節約時間的國家，以削減燈光照明和耗電開支。自此以後，全球以歐洲和北美為主的約 70 個國家都引用這個做法。

美國的 DST，起始於每年 3 月的第二個星期日，結束日期為每年 11 月的第一個星期日。這幾年分別是 2013/3/10、2014/3/9、2015/3/8，今年則為 2016/3/13。但也不是每個地方都採用，如美國亞利桑那州 (Arizona) 就不採用 DST。

2. 日光節約時間的調整

DST 時間的計算是基於和 UTC 的「差量 (offset)」，如在標準時間的情況下，美國紐約是「UTC - 5 小時」；在 DST 的情況下，紐約則是「UTC - 4 小時」。

於 2014/03/09，紐約啓用 DST 的前後幾秒，時間變化為：

❖ 表 16-1 啟用 DST 的時間變化

| 當地時間 | UTC 差量 |
|--------------------------|-----------------------|
| 1:59:58 AM | UTC-5h EST (美東標準時間) |
| 1:59:59 AM | UTC-5h EST |
| 2:00:00 AM >> 3:00:00 AM | UTC-4h EDT (美東夏令時間) |
| 3:00:01 AM | UTC-4h EDT |

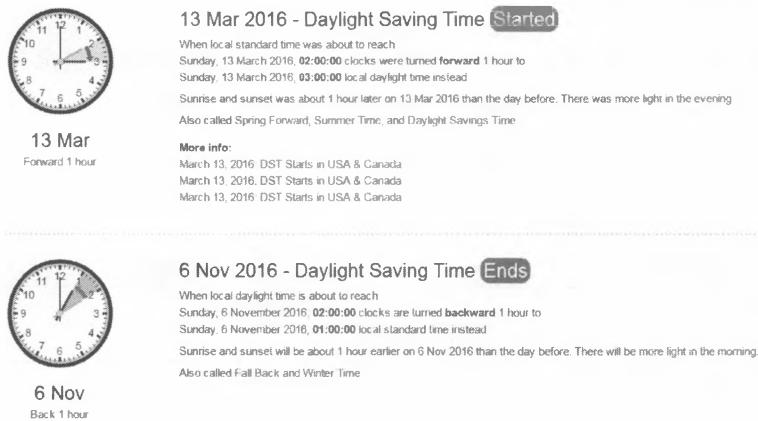
於 2014/11/02，紐約結束 DST 的前後幾秒，時間變化為：

❖ 表 16-2 結束 DST 的時間變化

| 當地時間 | UTC 差量 |
|--------------------------|-----------------------|
| 1:59:58 AM | UTC-4h EDT (美東夏令時間) |
| 1:59:59 AM | UTC-4h EDT |
| 2:00:00 AM >> 1:00:00 AM | UTC-5h EST (美東標準時間) |
| 1:00:01 AM | UTC-5h EST |

3. 日光節約時間的影響

- 啓用 DST，會讓時間「跳空」一小時；結束 DST，則會讓時間「重複」一小時。全年度來看，正好抵銷影響。
- 因為 DST 調整將導致調整前的時間不存在，且每個地方調整時間的始點不同，若調整在午夜凌晨 (midnight)，將導致無法使用 midnight 代表該天。
- 網址：<http://www.timeanddate.com/time/change/usa/new-york?year=2016>，有美國 DST 的相關訊息，也可以模擬時間的調整，具趣味性。



❖ 圖 16-2 www.timeanddate.com 顯示 DST 的調整

16.3.2 Java 8 在時區和日光節約時間的應用

Java 8 新增數個類別支援「時區 (Time Zones)」的應用，常見的有：

類別 ZoneId

ZoneId 類別代表某個時區，類別內也定義了所有時區。和 Locale 類別的用法相似。可以使用類別的 static 方法取得特定時區物件 ZoneId；再以 ZoneId 取得 ZoneRules 物件：

```
ZoneId taipei = ZoneId.systemDefault();
ZoneId newYork = ZoneId.of("America/New_York");
ZoneRules taipeiRules = taipei.getRules();
```

類別 ZoneRules

顧名思義，ZoneRules 類別擁有方法可以取得該時區的相關規則 (rules)，如：

- isDaylightSavings(Instant)：傳入的時間 (使用 Instant 物件) 是否是日光節約時間？
- getStandardOffset(Instant)：依據傳入的時間 (Instant) 判斷和 UTC 的標準時間差，不考慮日光節約時間的影響，並回傳 ZoneOffset 物件。
- getOffset(Instant)：依據傳入的時間 (Instant) 判斷和 UTC 的時間差，目前是否是日光節約時間將影響結果，並回傳 ZoneOffset 物件。

在 JDK 8 裡，會幫每個時區 (ZoneId) 內建相關 ZoneRules。再由 ZoneRulesProvider 類別提供時區的 ZoneRules。若 System.getProperty("java.time.zone.DefaultZoneRulesProvider")

可以取得設定值，就由該設定決定；若無，則由 TzdbZoneRulesProvider 類別提供。由「java.time.zone.ZoneRulesProvider」的來源碼可以看出如上邏輯：

```
public Object run() {
    String prop =
        System.getProperty("java.time.zone.DefaultZoneRulesProvider");
    if (prop != null) {
        try {
            Class<?> c =
                Class.forName(prop, true, ClassLoader.getSystemClassLoader());
            ZoneRulesProvider provider =
                ZoneRulesProvider.class.cast(c.newInstance());
            registerProvider(provider);
            loaded.add(provider);
        } catch (Exception x) {
            throw new Error(x);
        }
    } else {
        registerProvider(new TzdbZoneRulesProvider());
    }
    return null;
}
```

此時時區資訊將由「IANA Time Zone Database (TZDB)」定義，該檔案位置為「jdk1.8.0/jre/lib/tzdb.dat」。可以由 TzdbZoneRulesProvider 的來源碼看到：

```
public TzdbZoneRulesProvider() {
    try {
        String libDir = System.getProperty("java.home") + File.separator + "lib";
        try (DataInputStream dis = new DataInputStream(
            new BufferedInputStream(new FileInputStream(
                new File(libDir, "tzdb.dat"))))) {
            load(dis);
        }
    } catch (Exception ex) {
        throw
            new ZoneRulesException("Unable to load TZDB time-zone rules", ex);
    }
}
```

類別 ZoneOffset

代表該時區和 UTC 時間的「差量 (offset)」。因為繼承 ZonedDateTime 類別，也具備 ZonedDateTime 的欄位和方法。

綜合示範

範例「/OCP/src/course/c16/TimeZoneTest.java」顯示類別 ZoneId、ZoneRules、ZoneOffset 的關聯性和使用方式：

範例

```

1  public class TimeZoneTest {
2      public static void main(String[] args) {
3          // ZoneId
4          ZoneId taipei = ZoneId.systemDefault();
5          // System.out.println("Zone ID: " + taipei.getId());
6          ZoneId newYork = ZoneId.of("America/New_York");
7          // System.out.println("Zone ID: " + newYork.getId());
8
9          // ZoneId >> ZoneRules
10         ZoneRules taipeiRules = taipei.getRules();
11         ZoneRules newyorkRules = newYork.getRules();
12
13         //US started DST in 2016-03-13
14         Instant beforeUsDST = Instant.parse("2016-03-12T00:00:00Z");
15         Instant inUsDST = Instant.parse("2016-03-14T00:00:00Z");
16         Instant now = Instant.now();
17
18         System.out.println("Method Call\t\t"
19             + "Taipei\t"
20             + "NewYork(inDST)\t"
21             + "NewYork(beforeDST)");
22         System.out.println("-----");
23
24         System.out.println("isDaylightSavings():\t"
25             + taipeiRules.isDaylightSavings(now) + "\t\t"
26             + newyorkRules.isDaylightSavings(inUsDST) + "\t\t"
27             + newyorkRules.isDaylightSavings(beforeUsDST));
28
29         System.out.println("getDaylightSavings():\t"
30             + taipeiRules.getDaylightSavings(now).toHours() + "\t\t"
31             + newyorkRules.getDaylightSavings(inUsDST).toHours() + "\t\t"
32             + newyorkRules.getDaylightSavings(beforeUsDST).toHours());
33
34         ZoneOffset os = taipeiRules.getOffset(LocalDateTime.now());
35         System.out.println("getOffset():\t\t"
36             + os + "\t\t"
37             + newyorkRules.getOffset(inUsDST) + "\t\t"
38             + newyorkRules.getOffset(beforeUsDST));

```

```

39
40     ZoneOffset sos = taipeiRules.getStandardOffset(now);
41     System.out.println("getStandardOffset():\t"
42         + sos + "\t\t"
43         + newyorkRules.getStandardOffset(inUsDST) + "\t\t"
44         + newyorkRules.getStandardOffset(beforeUsDST));
45     }
46 }

```

結果

| Method Call | Taipei | NewYork(inDST) | NewYork(beforeDST) |
|-----------------------|--------|----------------|--------------------|
| isDaylightSavings(): | false | true | false |
| getDaylightSavings(): | 0 | 1 | 0 |
| getOffset(): | +08:00 | -04:00 | -05:00 |
| getStandardOffset(): | +08:00 | -05:00 | -05:00 |

說明

| | |
|-------|---|
| 4 | 使用 ZoneId.systemDefault() 取得程式執行所在地的時區。 |
| 6 | 使用 ZoneId.of("America/New_York")，取得指定地區的時區。 |
| 10~11 | 由 ZoneId 取得 ZoneRules。 |
| 14~16 | 使用類別 Instant 建立目前時間和指定時間，做法和 LocalDateTime 類別相似，在後續小節會有更詳細介紹。
台灣地區並未實施日光節約時間 (DST)，今年美國的 DST 開始時間是 2016-03-13。 |
| 14 | Instant 變數 beforeUsDST 代表美國啟動 DST 之前的时间 (2016-03-12)。 |
| 15 | Instant 變數 inUsDST 代表美國啟動 DST 之後的時間 (2016-03-14)。 |
| 24~27 | ZoneRules.isDaylightSavings(Instant) 方法回傳「該時區」的「指定時間」是否是日光節約時間。 |
| 29~32 | ZoneRules.getDaylightSavings(Instant).toHours() 方法將日光節約時間的差量轉換為小時。 |
| 34~38 | ZoneRules.getOffset(Instant) 方法依據傳入的時間 (Instant) 判斷和 UTC 的時間差，目前是否是日光節約時間將影響結果。
該方法有傳入 LocalDateTime 物件和 Instant 物件的 overloading 版本。 |
| 40~44 | ZoneRules.getStandardOffset(Instant) 方法依據傳入的時間 (Instant) 判斷和 UTC 的標準時間差，不考慮日光節約時間的影響。 |

類別 ZonedDateTime

相對於 LocalDateTime 類別只能處理當地、不含時區概念的日期和時間，而 ZonedDateTime 物件可以結合 LocalDateTime、ZoneId 和 ZoneOffset 的資訊，如範例「/OCP/src/course/c16/ZonedDateTimeDemo1.java」：

範例

```

1  public class ZonedDateTimeDemo1 {
2      public static void main(String[] args) {
3
4          LocalDateTime tpNow =
5              LocalDateTime
6                  .now()
7                  .truncatedTo(ChronoUnit.MINUTES);
8          System.out.println("Now in Taipei : " + tpNow);
9
10         ZoneId newYork = ZoneId.of("America/New_York");
11         ZonedDateTime nyNow =
12             ZonedDateTime
13                 .now(newYork)
14                 .truncatedTo(ChronoUnit.MINUTES);
15         System.out.println("Now in NewYork: " + nyNow);
16         System.out.println("Offset of NewYork: " + nyNow.getOffset());
17         System.out.println("Time Zone: " + nyNow.getZone());
18
19         ZonedDateTime time1 = ZonedDateTime.of(tpNow, newYork);
20         System.out.println("Time-1: " + time1);
21
22         ZonedDateTime time2 = time1.plusDays(1).minusMinutes(15);
23         System.out.println("Time-2: " + time2);
24     }
25 }
```

結果

```

Now in Taipei : 2016-07-10T09:33
Now in NewYork: 2016-07-09T21:33-04:00[America/New_York]
Offset of NewYork: -04:00
Time Zone: America/New_York
Time-1: 2016-07-10T09:33-04:00[America/New_York]
Time-2: 2016-07-11T09:18-04:00[America/New_York]
```

說明

| | |
|-------|---|
| 7 | 使用 <code>truncatedTo(ChronoUnit.MINUTES)</code> 讓時間顯示到「分」，其餘刪除。 |
| 12~13 | 搭配 <code>ZoneId</code> 的時區物件，建立 <code>ZonedDateTime</code> 物件。 |
| 16 | <code>ZonedDateTime.getOffset()</code> 可以取得和 UTC 的時差 (offset)。 |
| 17 | <code>ZonedDateTime.getZone()</code> 可以取得所在時區 (zone)。 |
| 19 | 藉由指定當地時間 (<code>LocalDateTime</code>) 和時區 (<code>ZoneId</code>)，直接建立 <code>ZonedDateTime</code> 物件。 |

ZonedDateTime 也可以在時間跨過 DTS 時，正確處理：

- 當地時間 (LocalDateTime) 沒有改變。
- 和 UTC 的時差 (offset) 可以正確被管理。

如範例 「/OCP/src/course/c16/ZonedDateTimeDemo2.java」：

範例

```

1  public class ZonedDateTimeDemo2 {
2      public static void main(String[] args) {
3          ZoneId usEast = ZoneId.of("America/New_York");
4          // DST Begins: 2016/03/13
5          LocalDateTime beforeStartDTS =
6              LocalDateTime.of(2016, 03, 12, 16, 00);
7          ZonedDateTime timeS1 =
8              ZonedDateTime.of(beforeStartDTS, usEast);
9          System.out.println("TimeS-1: " + timeS1);
10         ZonedDateTime timeS2 = timeS1.plusDays(1);
11         System.out.println("TimeS-2: " + timeS2);
12         ZonedDateTime timeS3 = timeS1.plusHours(24);
13         System.out.println("TimeS-3: " + timeS3);
14         // DST Ends: 2016/11/06
15         LocalDateTime beforeEndDTS =
16             LocalDateTime.of(2016, 11, 05, 16, 00);
17         ZonedDateTime timeE1 =
18             ZonedDateTime.of(beforeEndDTS, usEast);
19         System.out.println("TimeE-1: " + timeE1);
20         ZonedDateTime timeE2 = timeE1.plusDays(1);
21         System.out.println("TimeE-2: " + timeE2);
22         ZonedDateTime timeE3 = timeE1.plusHours(24);
23         System.out.println("TimeE-3: " + timeE3);
24     }
25 }
```

結果

```

TimeS-1: 2016-03-12T16:00-05:00[America/New_York]
TimeS-2: 2016-03-13T16:00-04:00[America/New_York]
TimeS-3: 2016-03-13T17:00-04:00[America/New_York]
TimeE-1: 2016-11-05T16:00-04:00[America/New_York]
TimeE-2: 2016-11-06T16:00-05:00[America/New_York]
TimeE-3: 2016-11-06T15:00-05:00[America/New_York]
```

 **說明**

| | |
|----|---|
| 9 | 在美東的 DTS (2016/03/13) 開始之前的時間：
2016-03-12T16:00，和 UTC 時差為 5 小時。 |
| 11 | 若 plusDays(1)，將進入 DTS：
2016-03-13T16:00，和 UTC 時差減少為 4 小時。 |
| 13 | 若 plusHours(24)，理論上和 plusDays(1) 應該相同，但顯示：
2016-03-13T17:00
因為開始 DTS 凌晨時間被直接往前調進 1 小時，那天實質只有 23 小時。 |
| 19 | 當要結束 DTS (2016/11/06) 之前的時間：
2016-11-05T16:00，和 UTC 時差為 4 小時。 |
| 21 | 若 plusDays(1)，將結束 DTS：
2016-11-06T16:00，和 UTC 時差回復為 5 小時。 |
| 23 | 若 plusHours(24)，理論上和 plusDays(1) 應該相同，但顯示：
2016-11-06T15:00
因為結束 DTS 凌晨時間被往後調回 1 小時，那天實質有 25 小時。 |

類別 ZoneOffsetTransition

單字 transition 翻譯為「過渡期」，由 ZoneRules.getTransition(LocalDateTime) 可以取得 ZoneOffsetTransition 物件，可以判斷啓動或結束 DTS 時發生的「時間斷層(gap)」或「時間重疊(overlap)」：

1. 啓動 DTS 時會將時間「快轉 1 小時」，造成時間的「斷層(gap)」。
2. 結束 DTS 時會將時間「倒回 1 小時」，造成時間的「重疊(overlap)」。

如範例「/OCP/src/course/c16/ZoneOffsetTransitionDemo.java」：

 **範例**

```

1  public class ZoneOffsetTransitionDemo {
2      // Ask the rules if there was a gap or overlap
3      private static void gapOrOverlap(ZoneId usEast, LocalDateTime dt) {
4          ZoneOffsetTransition zot = usEast.getRules().getTransition(dt);
5          System.out.print(dt + " is ");
6          if (zot != null) {
7              if (zot.isGap())
8                  System.out.println("gap");
9              if (zot.isOverlap())
10                  System.out.println("overlap");
11          } else {
12              System.out.println("-- ");
13          }
14      }

```

```

15     public static void main(String[] args) {
16         ZoneId usEast = ZoneId.of("America/New_York");
17         // DST Begins: 2016/03/13, 02->03
18         gapOrOverlap(usEast, LocalDateTime.of(2016, 03, 13, 1, 59));
19         gapOrOverlap(usEast, LocalDateTime.of(2016, 03, 13, 2, 01));
20         gapOrOverlap(usEast, LocalDateTime.of(2016, 03, 13, 2, 59));
21         gapOrOverlap(usEast, LocalDateTime.of(2016, 03, 13, 3, 01));
22         // DST Ends: 2016/11/06 , 02->01
23         gapOrOverlap(usEast, LocalDateTime.of(2016, 11, 6, 0, 59));
24         gapOrOverlap(usEast, LocalDateTime.of(2016, 11, 6, 1, 01));
25         gapOrOverlap(usEast, LocalDateTime.of(2016, 11, 6, 1, 59));
26         gapOrOverlap(usEast, LocalDateTime.of(2016, 11, 6, 2, 01));
27     }
28 }
```

結果

```

2016-03-13T01:59 is --
2016-03-13T02:01 is gap
2016-03-13T02:59 is gap
2016-03-13T03:01 is --
2016-11-06T00:59 is --
2016-11-06T01:01 is overlap
2016-11-06T01:59 is overlap
2016-11-06T02:01 is -
```

說明

19~20 美東 DST 的啟動時間是在 2016/03/13，在凌晨 02 時，將時間快轉到 03 時，造成這段時間的「斷層 (gap)」：



13 Mar

Forward 1 hour

❖ 圖 16-3 啟動 DST 時間快轉 1 小時

24~25

美東 DTS 的結束時間是在 2016/11/06，在凌晨 02 時，將時間回調到 01 時，造成這段時間的「重疊 (overlap)」。



6 Nov

Back 1 hour

❖ 圖 16-4 結束 DTS 時間回調 1 小時

類別 OffsetDateTime

而使用類別 `OffsetDateTime` 可以處理跨時區的問題。範例「/OCP/src/course/c16/TimeZoneAcrossDemo.java」中，準備在台北時間 2016/07/10 的 11: 30 分，和英國倫敦及美國紐約進行電話會議，因此必須事先預約時間：

範例

```

1 public class TimeZoneAcrossDemo {
2     public static void main(String[] args) {
3         LocalDateTime meeting = LocalDateTime.of(2016, 07, 10, 11, 30);
4
5         ZoneId taipei = ZoneId.systemDefault();
6         ZonedDateTime host = ZonedDateTime.of(meeting, taipei);
7         OffsetDateTime offset = host.toOffsetDateTime();
8
9         ZoneId london = ZoneId.of("Europe/London");
10        ZonedDateTime callLondon = offset.atZoneSameInstant(london);
11
12        ZoneId newYork = ZoneId.of("America/New_York");
13        ZonedDateTime callNewYork = offset.atZoneSameInstant(newYork);
14
15        System.out.println("conf call (Taipei) at: " + host);
16        System.out.println("conf call (London) at: " + callLondon);
17        System.out.println("conf call (NewYork) at: " + callNewYork);
18    }
19 }
```

結果

```

conf call (Taipei) at: 2016-07-10T11:30+08:00[Asia/Taipei]
conf call (London) at: 2016-07-10T04:30+01:00[Europe/London]
conf call (NewYork) at: 2016-07-09T23:30-04:00[America/New_York]
```

16.4 描述日期與時間的數量

類別 Instant

Instant 類別用來儲存時間軸上一剎那的時間，分成二部分來儲存：

1. epoch-seconds(long)

EPOCH 時間是指由 UTC/GMT 的 1970-01-01T00:00:00Z 開始起算後經歷的時間。因為認為該時間是 Unix 作業系統的時間起算點，所以也稱為 Unix epoch 或 Unix time 或 POSIX time 或 Unix timestamp。之後為正，之前為負。

2. nanosecond-of-second(int)

儲存值在 0 和 999,999,999 間。

Instant 類別依賴於 EPOCH 時間的狀況和建構 java.util.Date 物件，以及 System.currentTimeMillis() 的作法相當接近：

```
/*
 * Allocates a <code>Date</code> object and initializes it so that
 * it represents the time at which it was allocated, measured to the
 * nearest millisecond.
 *
 * @see     java.lang.System#currentTimeMillis()
 */
public Date() {
    this(System.currentTimeMillis());
}

/*
 * Allocates a <code>Date</code> object and initializes it to
 * represent the specified number of milliseconds since the
 * standard base time known as "the epoch", namely January 1,
 * 1970, 00:00:00 GMT.
 *
 * @param   date   the milliseconds since January 1, 1970, 00:00:00 GMT.
 * @see     java.lang.System#currentTimeMillis()
 */
public Date(long date) {
    fastTime = date;
}
```

❖ 圖 16-5 類別 java.util.Date 的建構方式

```
/*
 * Returns the current time in milliseconds. Note that
 * while the unit of time of the return value is a millisecond,
 * the granularity of the value depends on the underlying
 * operating system and may be larger. For example, many
 * operating systems measure time in units of tens of
 * milliseconds.
 *
 * <p> See the description of the class <code>Date</code> for
 * a discussion of slight discrepancies that may arise between
 * "computer time" and coordinated universal time (UTC).
 *
 * @return  the difference, measured in milliseconds, between
 *          the current time and midnight, January 1, 1970 UTC.
 * @see     java.util.Date
 */
public static native long currentTimeMillis();
```

❖ 圖 16-6 System.currentTimeMillis() 的定義

使用 Instant 和 java.util.Date 類別可以取得一致的 EPOCH 時間：

```
System.out.println(Instant.now().getEpochSecond());
-- 取得 EPOCH 時間，以秒計
--1468131459
System.out.println(new java.util.Date().getTime());
-- 取得 EPOCH 時間，以毫秒計
--1468131459758
```

範例「/OCP/src/course/c16/InstantDemo.java」顯示類別 Instant 的使用方式：

範例

```
1 public class InstantDemo {
2     public static void main(String[] args) throws InterruptedException {
3         Instant now = Instant.now();
4         Thread.sleep(0, 1); // long milliseconds, int nanoseconds
5         Instant later = Instant.now();
6         System.out.println("now is before later? " + now.isBefore(later));
7         System.out.println(" Now: " + now);
8         System.out.println("Later: " + later);
9         Instant epoch = Instant.parse("1970-01-01T00:00:00Z");
10        System.out.println("EPOCH: " + epoch);
11    }
12 }
```

結果

```
now is before later? true
Now: 2016-07-10T06:24:08.574Z
Later: 2016-07-10T06:24:08.575Z
EPOCH: 1970-01-01T00:00:00Z
```

類別 Period 和 Duration

Period 類別

- 使用 years, months, days 來建構日期的差量，都依據 ISO-8601 規範。API：文件的描述為：「This class models a quantity or amount of time in terms of years, months and days.」
- 使用 plus() 和 minus() 方法時，都是以天為概念，因此可以保留日光節約時間的變化。

Duration 類別

- 使用 seconds 和 nanoseconds 來建構時間的差量。也可以換算為 hours 和 minutes。
- API 文件的描述為：「This class models a quantity or amount of time in terms of seconds and nanoseconds. It can be accessed using other duration-based units, such as minutes and hours.」
- 「每一天」被「24 小時」的概念取代，因此沒有日光節約時間的概念。

如範例 「/OCP/src/course/c16/PeriodAndDurationDemo.java」：

範例

```

1  public class PeriodAndDurationDemo {
2      public static void main(String[] args) {
3          LocalDateTime beforeDST =
4              LocalDateTime.of(2016, 03, 12, 12, 00);
5          ZonedDateTime t = ZonedDateTime
6              .of(beforeDST, ZoneId.of("America/New_York"));
7          // show Period
8          Period day1Period = Period.ofDays(1);
9          System.out.println("Period of 1 day: " + day1Period);
10         System.out.println("Before: " + t);
11         System.out.println("After: " + t.plus(day1Period));
12         // show Duration
13         Duration hours24Duration = Duration.ofHours(24);
14         System.out.println("Duration of 24 hours: " + hours24Duration);
15         System.out.println("Before: " + t);
16         System.out.println("After: " + t.plus(hours24Duration));
17     }
18 }
```

結果

```

Period of 1 day: P1D
Before: 2016-03-12T12:00-05:00[America/New_York]
After: 2016-03-13T12:00-04:00[America/New_York]
Duration of 24 hours: PT24H
Before: 2016-03-12T12:00-05:00[America/New_York]
After: 2016-03-13T13:00-04:00[America/New_York]
```

說明

| | |
|----|--|
| 11 | 以 Period 物件讓 ZonedDateTime 物件增加 1 天。 |
| 16 | 以 Duration 物件讓 ZonedDateTime 物件增加 24 小時。因 DST 的起始日 2016/03/13 只有 23 小時，故增加 24 小時，會變成隔天再多 1 小時，顯示沒有日光節約時間的概念。 |

若要計算兩個日期的差距，可以使用：

1. ChronoUnit.DAYS.between(LocalDate, LocalDate)

回傳差距的總天數。

2. Period.between(LocalDate, LocalDate)

再使用 getMonths() 回傳差幾個月，使用 getDays() 回傳差幾天。

如範例「/OCP/src/course/c16/DayDiffDemo.java」：

範例

```

1  public class DayDiffDemo {
2      public static void main(String[] args) {
3          LocalDate christmas = LocalDate.of(2016, 12, 25);
4          LocalDate today = LocalDate.now();
5          System.out.println("Today is " + today);
6          long days = ChronoUnit.DAYS.between(today, christmas);
7          System.out.println("There are " + days + " days until Christmas");
8          Period untilXMas = Period.between(today, christmas);
9          System.out.println("There are "
10             + untilXMas.getMonths() + " months, "
11             + untilXMas.getDays() + " days until Christmas");
12     }
13 }
```

結果

```

Today is 2016-07-10
There are 168 days until Christmas
There are 5 months, 15 days until Christmas
```

使用流暢 (fluent) 的程式風格

範例

```

1  public class FluentDemo {
2      public static void main(String[] args) {
3          LocalDate myDay0 = LocalDate.of(1977, 6, 11);
4          LocalDate myDay1 = Year.of(1977).atMonth(06).atDay(11);
5
6          LocalDateTime meeting =
7              LocalDate.of(2016, 07, 10).atTime(11, 30);
```

```

9   ZonedDateTime host =
10  meeting.atZone(ZoneId.systemDefault());
11  System.out.println(host);
12  ZonedDateTime meetingUK =
13  host.withZoneSameInstant(ZoneId.of("Europe/London"));
14  System.out.println(meetingUK);
15  ZonedDateTime meetingSF =
16  host.withZoneSameInstant(ZoneId.of("America/New_York"));
17  System.out.println(meetingSF);
18 }
19 }
```

結果

```

2016-07-10T11:30+08:00[Asia/Taipei]
2016-07-10T04:30+01:00[Europe/London]
2016-07-09T23:30-04:00[America/New_York]
```

說明

| | |
|------|--|
| 3 | 使用一般語法。 |
| 4 | 使用流暢語法。 |
| 6~17 | 和範例程式 TimeZoneAcrossDemo 比較結果相同，但使用較流暢的語法。 |

16.5 認證考試命題範圍

依據甲骨文公布的考試範圍 (https://education.oracle.com/java-se-8-programmer-ii/pexam_1Z0-809)，和本章相關的主題有：

Use Java SE 8 Date/Time API

1. Create and manage date-based and time-based events including a combination of date and time into a single object using **LocalDate**, **LocalTime**, **LocalDateTime**, **Instant**, **Period**, and **Duration**.
2. Work with dates and times across **timezones** and manage changes resulting from **daylight savings** including Format date and times values.
3. Define and create and manage date-based and time-based events using **Instant**, **Period**, **Duration**, and **TemporalUnit**.

本章擬真試題實戰

考題 1

Which statement is true about `java.time.Duration`?

- A. It tracks time zones.
- B. It preserves daylight saving time.
- C. It defines time-based values.
- D. It defines date-based values.

答案 C

考題 2

Given:

```
public static void main(String[] args) {  
    LocalDate someDate = LocalDate.of(2016, Month.FEBRUARY, 13);  
    LocalDate afterYear = someDate.plusYears(1);  
    afterYear.plusDays(15); // line n1  
    System.out.println(afterYear);  
}
```

What is the result?

- A. 2017-02-13
- B. A `DateTimeException` is thrown.
- C. 2017-02-28
- D. A compilation error occurs at line n1.

答案 A

說明 `LocalDate` 物件是 immutable。

考題 3

Given:

```
public static void main(String[] args) {  
    ZonedDateTime start =  
        ZonedDateTime.of(2016, 7, 15, 3, 0, 0, 0, ZoneId.of("UTC-7"));  
    ZonedDateTime end =  
        ZonedDateTime.of(2016, 7, 15, 9, 0, 0, 0, ZoneId.of("UTC-5"));  
    long diff = ChronoUnit.HOURS.between(start, end); // line n1  
    System.out.println("Difference time is " + diff + " hours");  
}
```

What is the result?

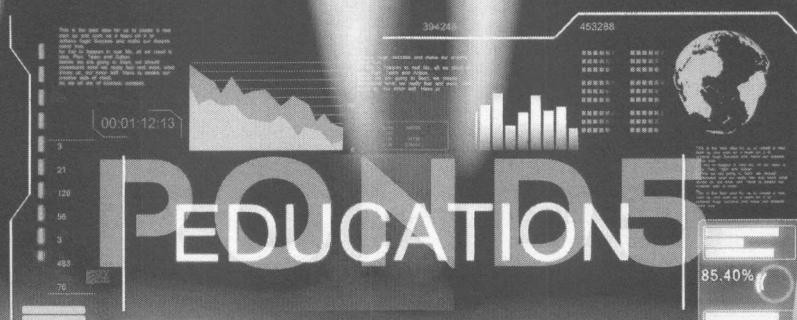
- A. Difference time is 4 hours
- B. Difference time is 6 hours
- C. Difference time is 8 hours
- D. An exception is thrown at line n1.

答案 A

說明 將相對時間調回絕對時間後再相減： $(9+5) - (3+7) = 4$



博碩文化



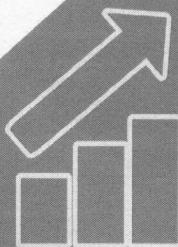
博碩軟體教育雲

上線囉!!



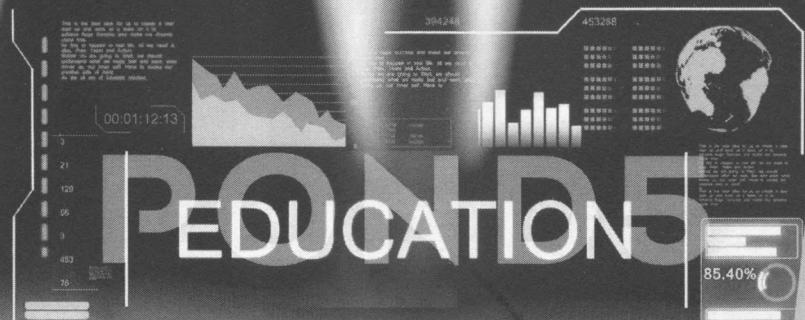
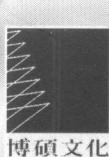
隨著雲端世代的來臨

博碩文化為特別成立了一個線上平台
除了提供 SaaS/ IaaS/ PaaS 服務之外
相關的教學工具書也都可在平台上找到
為大家提供更方便的服務



首波主打 IBM SPSS 特惠專案已經開跑囉
軟體與工具書可以一次擁有 詳請請上
<https://www.drmaster.net/software/>





博碩軟體教育雲

上線囉!!



隨著雲端世代的來臨

博碩文化為特別成立了一個線上平台
除了提供 SaaS/ IaaS/ PaaS 服務之外
相關的教學工具書也都可在平台上找到
為大家提供更方便的服務



首波主打 IBM SPSS 特惠專案已經開跑囉
軟體與工具書可以一次擁有 詳請請上
<https://www.drmaster.net/software/>

