

# Java<sup>SE8</sup>

## 技術手冊

林良

- 涵蓋 OCP/JP (原 SCJP) 考試範圍
- Lambda 專案、新時間日期 API、等 Java SE 8 新功能詳細介紹
- JDK 基礎與 IDE 操作交相對照
- 提供實作檔案與操作錄影教學

碁峯資訊

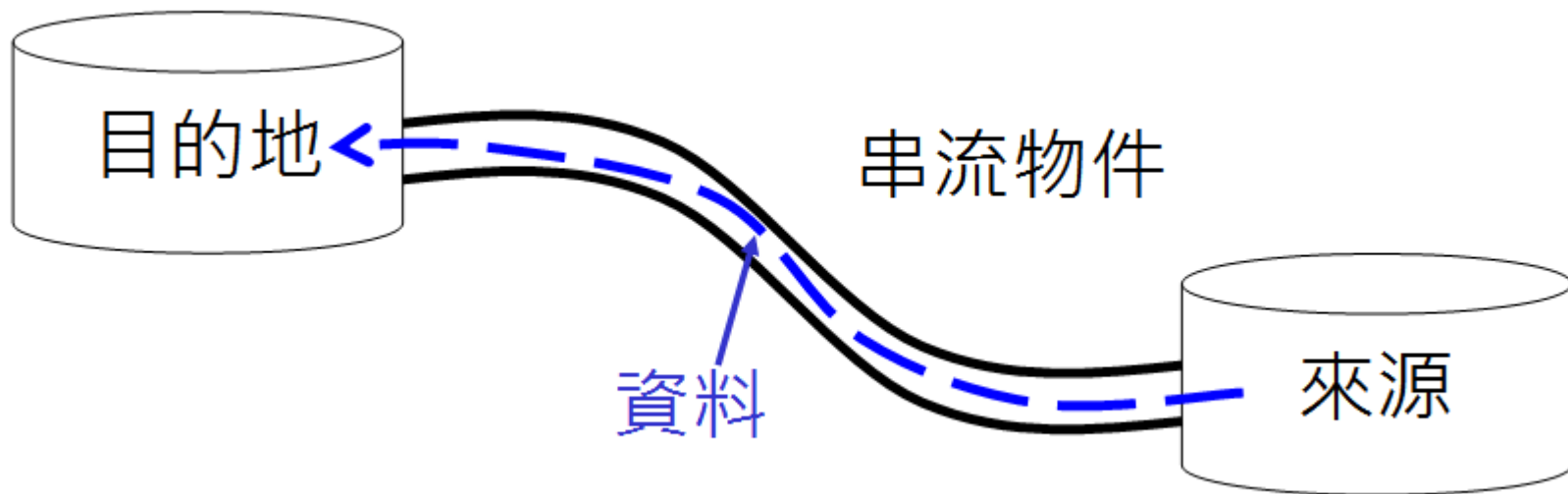
版權聲明：本教學投影片僅供教師授課講解使用，投影片內之圖片、文字及其相關內容，未經著作權人許可，不得以任何形式或方法轉載使用。

## 輸入輸出

### 學習目標

- 瞭解串流與輸入輸出的關係
- 認識 `InputStream`、`OutputStream` 繼承架構
- 認識 `Reader`、`Writer` 繼承架構
- 使用輸入輸出裝飾器類別

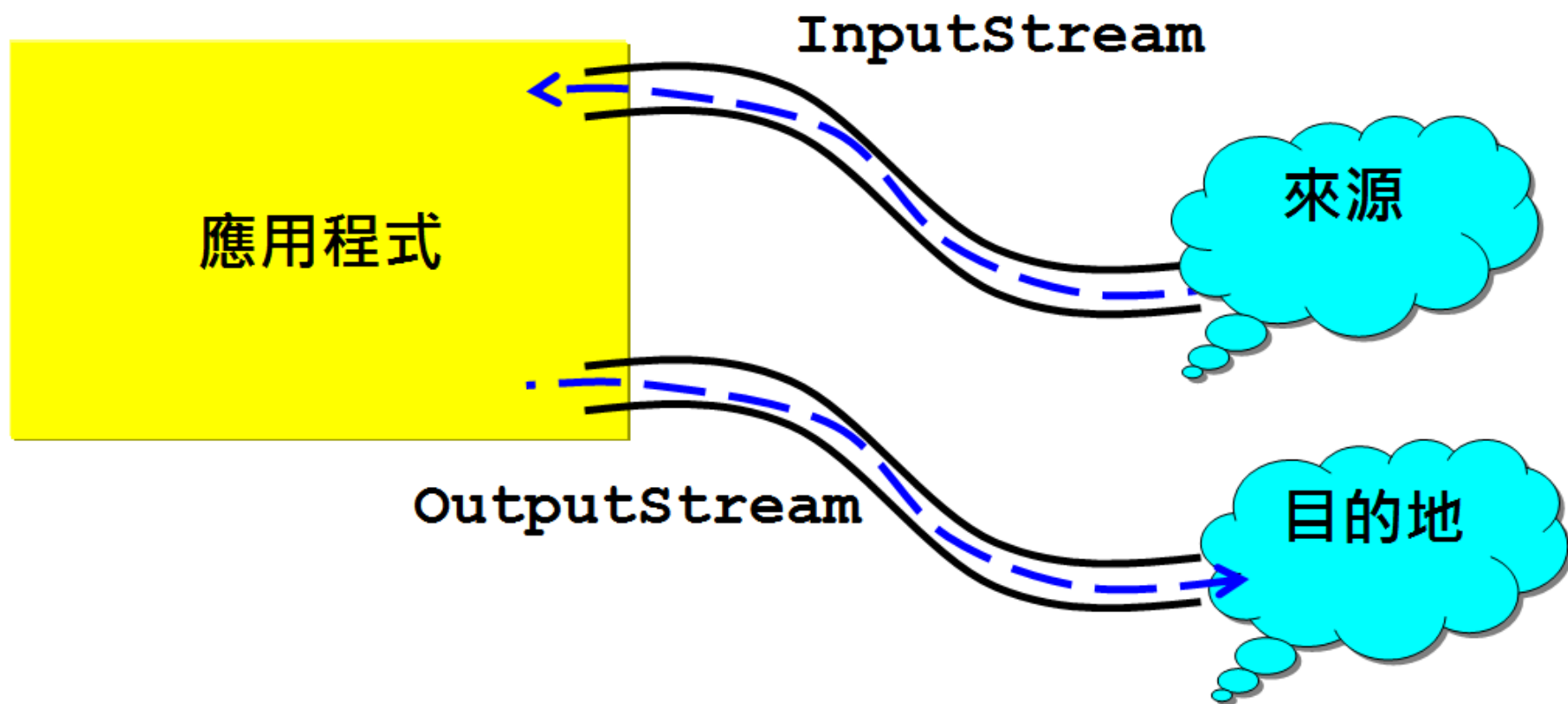
# 串流設計觀念



# 串流設計觀念

- 從應用程式角度來看，如果要將資料從來源取出，可以使用輸入串流，如果要將資料寫入目的地，可以使用輸出串流
- 輸入串流代表物件為 `java.io.InputStream` 實例，輸出串流代表物件為 `java.io.OutputStream` 實例

# 串流設計觀念



# 串流設計觀念

- 來源與目的地都不知道的情況下，如何撰寫程式？

```
import java.io.*;

public class IO {
    public static void dump(InputStream src, OutputStream dest) ← ❶ 資料來源與目的地
        throws IOException { ← ❷ 客戶端要處理例外
        try (InputStream input = src; OutputStream output = dest) { ← ❸ 嘗試自動關
            byte[] data = new byte[1024]; ← ❹ 嘗試每次從來源
            int length;
            while ((length = input.read(data)) != -1) { ← ❺ 讀取資料
                output.write(data, 0, length); ← ❻ 寫出資料
            }
        }
    }
}
```

# 串流設計觀念

- 在不使用 `InputStream` 與 `OutputStream` 時，必須使用 **`close()`** 方法關閉串流
- `InputStream` 與 `OutputStream` 實作了 **`java.io.Closeable`** 介面，其父介面為 **`java.lang.AutoCloseable`** 介面，因此可使用 JDK7 嘗試自動關閉資源語法

# 串流設計觀念

- 如果要將某個檔案讀入並另存為另一個檔案

```
import java.io.*;

public class Copy {
    public static void main(String[] args) throws IOException {
        IO.dump(
            new FileInputStream(args[0]),
            new FileOutputStream(args[1])
        );
    }
}
```

```
> java cc.openhome.Copy c:\workspace\Main.java C:\workspace\Main.txt
```



# 串流設計觀念

- 如果要從 HTTP 伺服器讀取某個網頁，並另存為檔案 …

```
import java.io.*;
import java.net.URL;

public class Download {
    public static void main(String[] args) throws IOException {
        URL url = new URL(args[0]);
        InputStream src = url.openStream();
        OutputStream dest = new FileOutputStream(args[1]);
        IO.dump(src, dest);
    }
}
```

```
> java cc.openhome.Download http://openhome.cc c:\workspace\index.txt
```

# 串流設計觀念

- 使用 `java.net.ServerSocket` 接受客戶端連線的例子：

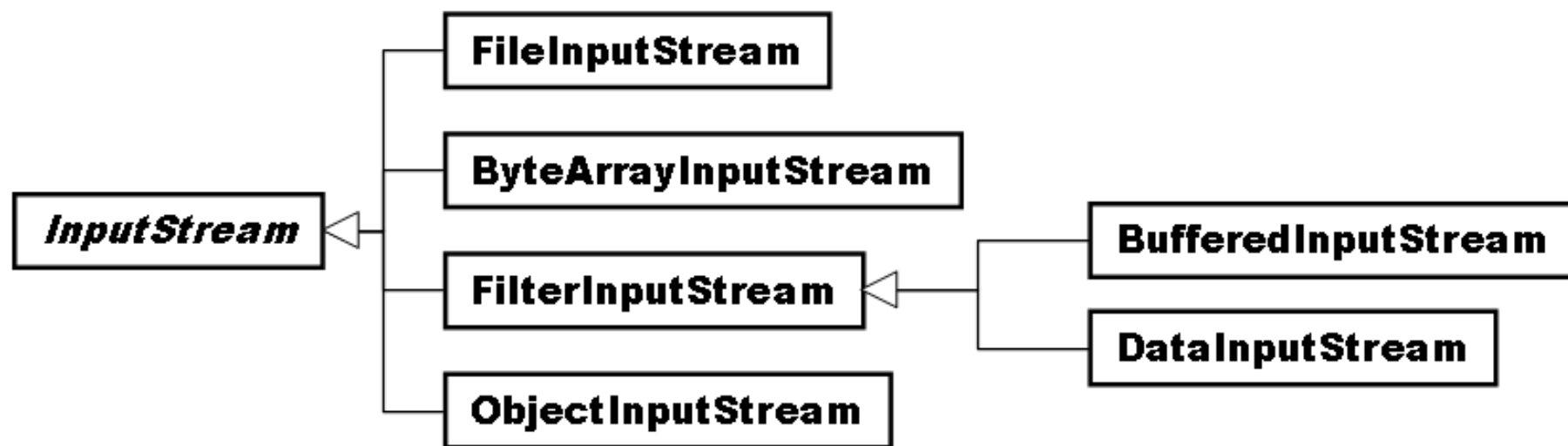
```
ServerSocket server = null;
Socket client = null;
try {
    server = new ServerSocket(port);
    while(true) {
        client = server.accept();
        InputStream input = client.getInputStream();
        OutputStream output = client.getOutputStream();
        // 接下來就是操作 InputStream、OutputStream 實例了...
        ...
    }
}
catch(IOException ex) {
    ....
}
```

# 串流設計觀念

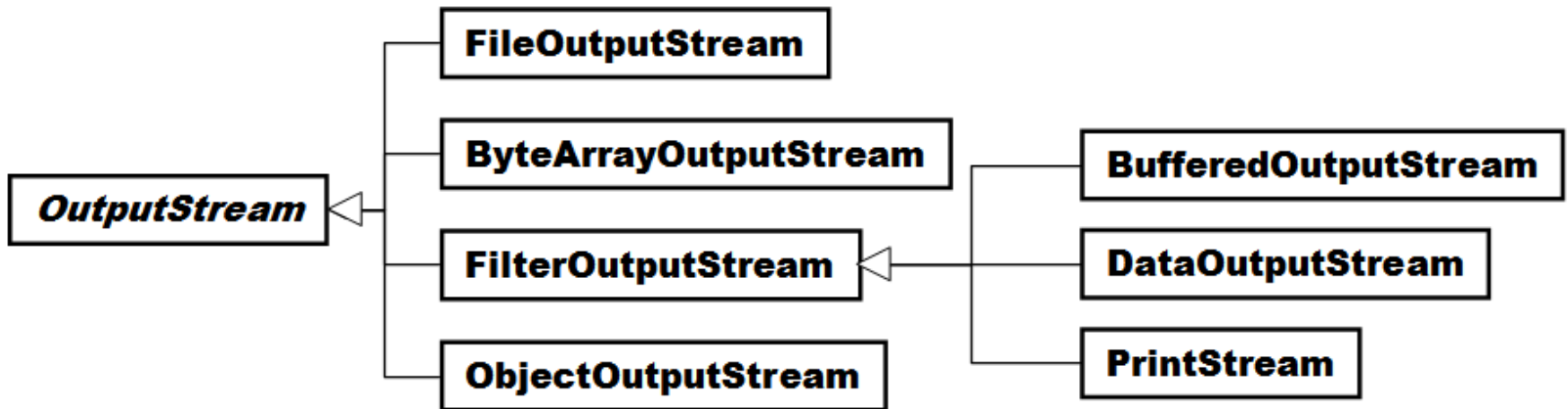
- 將來學到 Servlet，想將檔案輸出至瀏覽器，也會有類似的操作：

```
response.setContentType("application/pdf");
InputStream in = this.getServletContext()
                    .getResourceAsStream("/WEB-INF/jdbc.pdf");
OutputStream out = response.getOutputStream();
byte[] data = new byte[1024];
int length = -1;
while((length = in.read(data)) != -1) {
    out.write(data, 0, length);
}
```

# 串流繼承架構



# 串流繼承架構



# 串流繼承架構

- 記得 `System.in` 與 `System.out` 嗎？分別是 `InputStream` 與 `PrintStream` 的實例 ...
- 較少直接操作 `InputStream` 相關方法，而是如先前章節使用 `java.util.Scanner` 包裹 `System.in`

# 串流繼承架構

- 可以使用 `System` 的 `setIn()` 方法指定 `InputStream` 實例，指定標準輸入來源
- 將標準輸入指定為 `FileInputStream`，可以讀取指定檔案並顯示在文字模式

```
System.setIn(new FileInputStream(args[0]));  
try (Scanner file = new Scanner(System.in)) {  
    while (file.hasNextLine()) {  
        System.out.println(file.nextLine());  
    }  
}
```

# 串流繼承架構

- 若要將 10.1.1 的 Download 範例改為輸出至標準輸出，也可以這麼寫：

```
...  
URL url = new URL(args[0]);  
InputStream src = url.openStream();  
IO.dump(src, System.out);  
...
```



# 串流繼承架構

- 標準輸出可以重新導向至檔案，只要執行程式時使用 > 將輸出結果導向至指定的檔案

```
> java Hello > Hello.txt
```

- 如果使用 >> 則是附加訊息

# 串流繼承架構

- 可以使用 `System` 的 `setOut()` 方法指定 `PrintStream` 實例，將結果輸出至指定的目的地
- 將標準輸出指定至檔案：

```
try (PrintStream printStream = new PrintStream(  
    new FileOutputStream(args[0]))) {  
    System.setOut(printStream);  
    System.out.println("HelloWorld");  
}
```

# 串流繼承架構

- **System.err** 為 `PrintStream` 實例，稱之為標準錯誤輸出串流，用來立即顯示錯誤訊息
- `System.out` 輸出的訊息可以使用 `>` 或 `>>` 重新導向至檔案，但 `System.err` 輸出的訊息一定會顯示在文字模式中，無法重新導向
- 可以使用 **System.setErr()** 指定 `PrintStream`，指定標準錯誤輸出串流

# 串流繼承架構

- `FileInputStream` 是 `InputStream` 的子類，可以指定檔案名稱建構實例，一旦建構檔案就開啟，接著就可用來讀取資料
- `FileOutputStream` 是 `OutputStream` 的子類，可以指定檔案名稱建構實例，一旦建構檔案就開啟，接著就可以用來寫出資料
- 無論是 `FileInputStream` 或 `FileOutputStream`，不使用時都要使用 **`close()`** 關閉檔案

# 串流繼承架構

- `FileInputStream` 主要實作了 `InputStream` 的 `read()` 抽象方法，使之可從檔案中讀取資料
- `FileOutputStream` 主要實作了 `OutputStream` 的 `write()` 抽象方法，使之可寫出資料至檔案

# 串流繼承架構

- `FileInputStream`、`FileOutputStream` 在讀取、寫入檔案時，是以位元組為單位
- 通常會使用一些高階類別加以包裹，進行一些高階操作，像是 `Scanner` 與 `PrintStream` 類別等

# 串流繼承架構

- `ByteArrayInputStream` 是 `InputStream` 的子類，可以指定 `byte` 陣列建構實例，一旦建構就可將 `byte` 陣列當作資料來源進行讀取
- `ByteArrayOutputStream` 是 `OutputStream` 的子類，可以指定 `byte` 陣列建構實例，一旦建構將 `byte` 陣列寫作目的地寫出資料

# 串流繼承架構

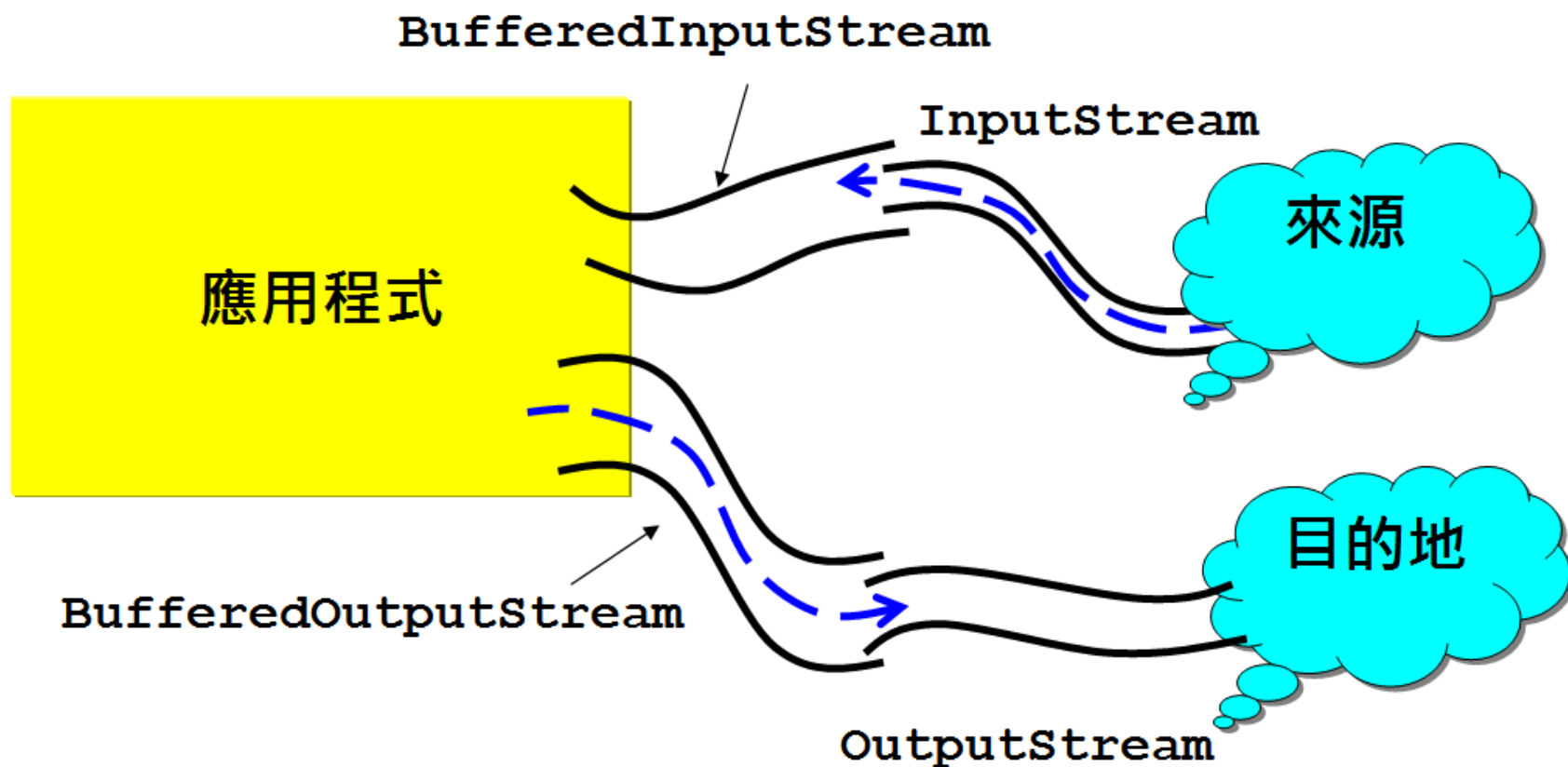
- `ByteArrayInputStream` 主要實作了 `InputStream` 的 `read()` 抽象方法，使之可從 `byte` 陣列中讀取資料
- `ByteArrayOutputStream` 主要實作了 `OutputStream` 的 `write()` 抽象方法，使之可寫出資料至 `byte` 陣列



# 串流處理裝飾器

- `InputStream`、`OutputStream` 提供串流基本操作，如果想要為輸入輸出的資料作加工處理，則可以使用包裹器類別
  - 具備緩衝區作用的 `BufferedInputStream`、`BufferedOutputStream`
  - 具備資料轉換處理作用的 `DataInputStream`、`DataOutputStream`
  - 具備物件序列化能力的 `ObjectInputStream`、`ObjectOutputStream`

# 串流處理裝飾器



# 串流處理裝飾器

- 如果傳入 `IO.dump()` 的是 `FileInputStream`、`FileOutputStream` 實例，每次 `read()` 時都會要求讀取硬碟，每次 `write()` 都會要求寫入硬碟，這會花費許多時間在硬碟定位上

# 串流處理裝飾器

- BufferedInputStream 與 BufferedOutputStream 主要於內部提供緩衝區功能
- 建構  
BufferedInputStream、BufferedOutputStream 必須提供  
InputStream、OutputStream 進行包裹，可以使用預設或自訂緩衝區大小

# 串流處理裝飾器

```
import java.io.*;

public class BufferedIO {
    public static void dump(InputStream src, OutputStream dest)
        throws IOException {
        try(InputStream input = new BufferedInputStream(src);
            OutputStream output = new BufferedOutputStream(dest)) {
            byte[] data = new byte[1024];
            int length;
            while ((length = input.read(data)) != -1) {
                output.write(data, 0, length);
            }
        }
    }
}
```

# 串流處理裝飾器

- `DataInputStream`、`DataOutputStream` 提供讀取、寫入 Java 基本資料型的方法，像是讀寫 `int`、`double`、`boolean` 等的方法

```
public class Member {
    private String number;
    private String name;
    private int age;

    public Member(String number, String name, int age) {
        this.number = number;
        this.name = name;
        this.age = age;
    }
    // 部份程式碼省略...因為只一些 Getter、Setter...
    @Override
    public String toString() {
        return String.format("(%s, %s, %d)", number, name, age);
    }

    public void save() {
        try(DataOutputStream output = ← ① 建立 DataOutputStream
            new DataOutputStream(new FileOutputStream(number))) {
            output.writeUTF(number);
            output.writeUTF(name);
            output.writeInt(age); ← ② 根據不同的型態呼叫 writeXXX() 方法
        } catch(IOException ex) {
            throw new RuntimeException(ex);
        }
    }
}
```

```
public static Member load(String number) {
    Member member = null;
    try(DataInputStream input = ← ❸ 建立 DataInputStream 包裹 FileInputStream
        new DataInputStream(new FileInputStream(number))) {
        member = new Member(
            input.readUTF(), input.readUTF(), input.readInt()); ←
    } catch(IOException ex) {
        throw new RuntimeException(ex);
    }
    return member;
}

Member[] members = {
    new Member("B1234", "Justin", 90),
    new Member("B5678", "Monica", 95),
    new Member("B9876", "Irene", 88)
};
for(Member member : members) {
    member.save();
}
out.println(Member.load("B1234"));
out.println(Member.load("B5678"));
out.println(Member.load("B9876"));
```

❹ 根據不同的型態呼  
叫 readXXX() 方法



# 串流處理裝飾器


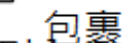

- `ObjectInputStream` 提供 **`readObject()`** 方法將資料讀入為物件，而 `ObjectOutputStream` 提供 **`writeObject()`** 方法將物件寫至目的地
- 可以被這兩個方法處理的物件，必須實作 **`java.io.Serializable`** 介面，這個介面並沒有定義任何方法，只是作為標示之用，表示這個物件是可以序列化的（`Serializable`）

```
public class Member2 implements Serializable { ← ❶ 實作 Serializable
    private String number;
    private String name;
    private int age;

    public Member2(String number, String name, int age) {
        this.number = number;
        this.name = name;
        this.age = age;
    }
    // 部份程式碼省略...因為只一些 Getter、Setter...

    @Override
    public String toString() {
        return String.format("(%s, %s, %d)", number, name, age);
    }

    public void save() {
        try(ObjectOutputStream output = ← ❷ 建立 DataOutputStream
            new ObjectOutputStream(new FileOutputStream(number))) { ← 包裹 FileOutputStream
            output.writeObject(this); ← ❸ 呼叫 writeObject() 方法寫入物件
        } catch(IOException ex) {
            throw new RuntimeException(ex);
        }
    }
}
```

```
public static Member2 load(String number) {  
    Member2 member = null;  
    try(ObjectInputStream input =  ④ 建立 DataInputStream  
        new ObjectInputStream(new  包裹 FileInputStream  
            FileInputStream(number))) {  
        member = (Member2) input.readObject();  ⑤ 呼叫 readObject()  
    } catch(IOException | ClassNotFoundException ex) { 方法讀入物件  
        throw new RuntimeException(ex);  
    }  
    return member;  
}  
}
```

```
Member2[] members = {new Member2("B1234", "Justin", 90),  
                      new Member2("B5678", "Monica", 95),  
                      new Member2("B9876", "Irene", 88)};  
for(Member2 member : members) {  
    member.save();  
}  
out.println(Member2.load("B1234"));  
out.println(Member2.load("B5678"));  
out.println(Member2.load("B9876"));
```

# 串流處理裝飾器

- 如果在作物件序列化時，物件中某些資料成員不希望被寫入，則可以標上 **transient** 關鍵字

# Reader 與 Writer 繼承架構

```
import java.io.*;
```

```
public class CharUtil {  
    public static void dump(Reader src, Writer dest) throws IOException {  
        try(Reader input = src; Writer output = dest) { ← ❸ 嘗試自動關閉資源  
            char[] data = new char[1024]; ← ❹ 嘗試每次從來源讀取 1024 字元  
            int length;  
            while((length = input.read(data)) != -1) { ← ❺ 讀取資料  
                output.write(data, 0, length); ← ❻ 寫出資料  
            }  
        }  
    }  
}
```

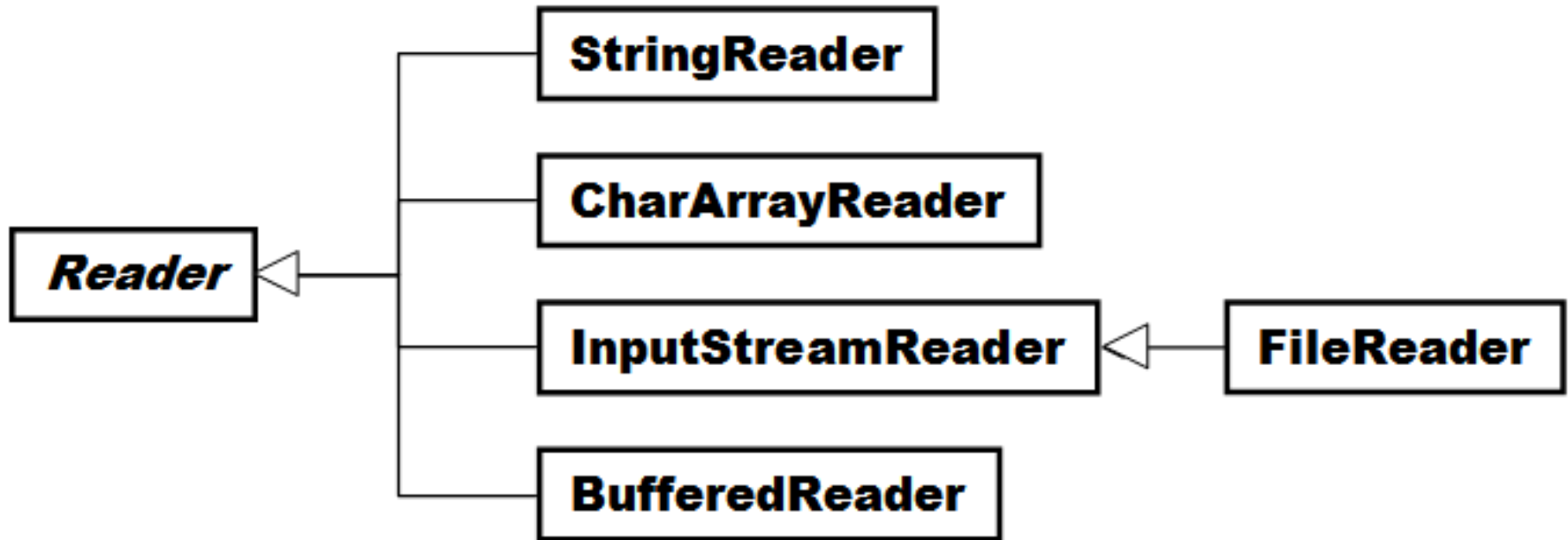
❶ 資料來源與目的地

❷ 客戶端要處理例外

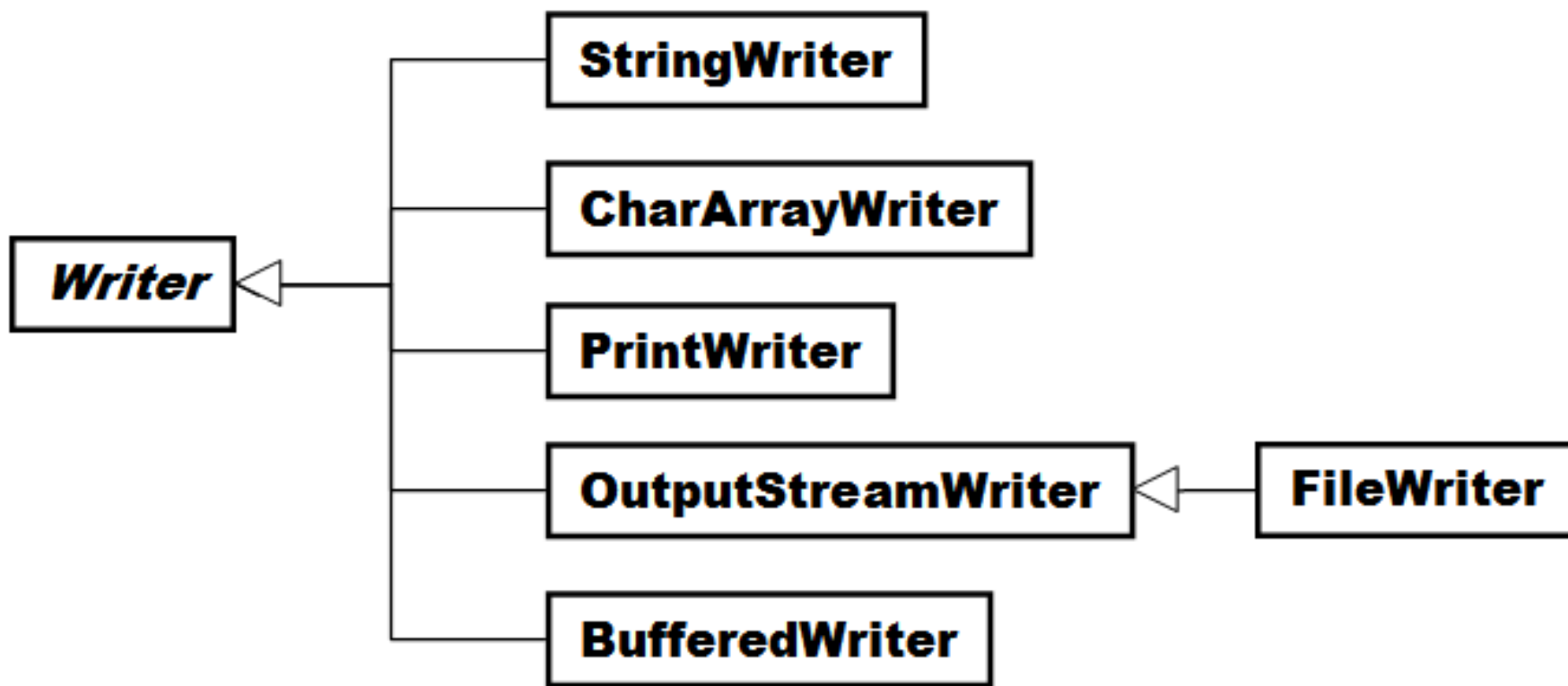
# Reader 與 Writer 繼承架構

- 在不使用 Reader 與 Writer 時，必須使用 **close()** 方法關閉串流，由於 Reader 與 Writer 實作了 **Closeable** 介面，其父介面為 **AutoCloseable** 介面，因此可使用 JDK7 嘗試自動關閉資源語法

# Reader 與 Writer 繼承架構



# Reader 與 Writer 繼承架構





# Reader 與 Writer 繼承架構

- 若要使用 `CharUtil.dump()` 讀入檔案、轉為字串並顯示在文字模式中...

```
FileReader reader = new FileReader(args[0]);  
StringWriter writer = new StringWriter();  
CharUtil.dump(reader, writer);  
System.out.println(writer.toString());
```

# Reader 與 Writer 繼承架構

- `FileReader`、`FileWriter` 可以對檔案作讀取與寫入，讀取或寫入時預設會使用作業系統預設編碼來作字元轉換
- 在啟動 JVM 時，可指定 `-Dfile.encoding` 來指定 `FileReader`、`FileWriter` 所使用的編碼

```
> java -Dfile.encoding=UTF-8 cc.openhome.CharUtil sample.txt
```

# Reader 與 Writer 繼承架構

- `FileReader`、`FileWriter` 沒有可以指定編碼的方法
- 如果在程式執行過程中想要指定編碼，則必須使用

`InputStreamReader`、`OutputStreamWriter`

# 字元處理裝飾器

- 想要將位元組資料轉換為對應的編碼字元，可以使用  
`InputStreamReader`、`OutputStreamWriter` 對串流資料資料包裹
- 在建立 `InputStreamReader` 與 `OutputStreamWriter` 時，可以指定編碼，如果沒有指定編碼，則以 JVM 啟動時所獲取的預設編碼來作字元轉換

# 字元處理裝飾器

```
public static void dump(Reader src, Writer dest) throws IOException {
    try(Reader input = src; Writer output = dest) {
        char[] data = new char[1024];
        int length;
        while((length = input.read(data)) != -1) {
            output.write(data, 0, length);
        }
    }
}

public static void dump(InputStream src, OutputStream dest,
                        String charset) throws IOException {
    dump(
        new InputStreamReader(src, charset),
        new OutputStreamWriter(dest, charset)
    );
}

// 採用預設編碼
public static void dump(InputStream src, OutputStream dest)
                        throws IOException {
    dump(src, dest, System.getProperty("file.encoding"));
}
```

# 字元處理裝飾器

- 想以 UTF-8 處理字元資料，例如讀取 UTF-8 的 Main.java 文字檔案，並另存為 UTF-8 的 Main.txt 文字檔案 …

```
CharUtil2.dump(  
    new FileInputStream("Main.java"),  
    new FileOutputStream("Main.txt"),  
    "UTF-8"  
);
```

# 字元處理裝飾器

- `BufferedReader`、`BufferedWriter` 可對 `Reader`、`Writer` 提供緩衝區作用
- JDK 1.4 之前，標準 API 並沒有 `Scanner` 類別，若要在文字模式下取得使用者輸入的字串，會如下撰寫：

```
BufferedReader reader = new BufferedReader(  
    new InputStreamReader(System.in));  
String name = reader.readLine();  
System.out.printf("Hello, %s!", name);
```

# 字元處理裝飾器

- `PrintWriter` 與 `PrintStream` 使用上極為類似，除了可以對 `OutputStream` 包裹之外，`PrintWriter` 還可以對 `Writer` 進行包裹，提供 `print()`、`println()`、`format()` 等方法