

Java^{SE8} 技術手冊

- 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。

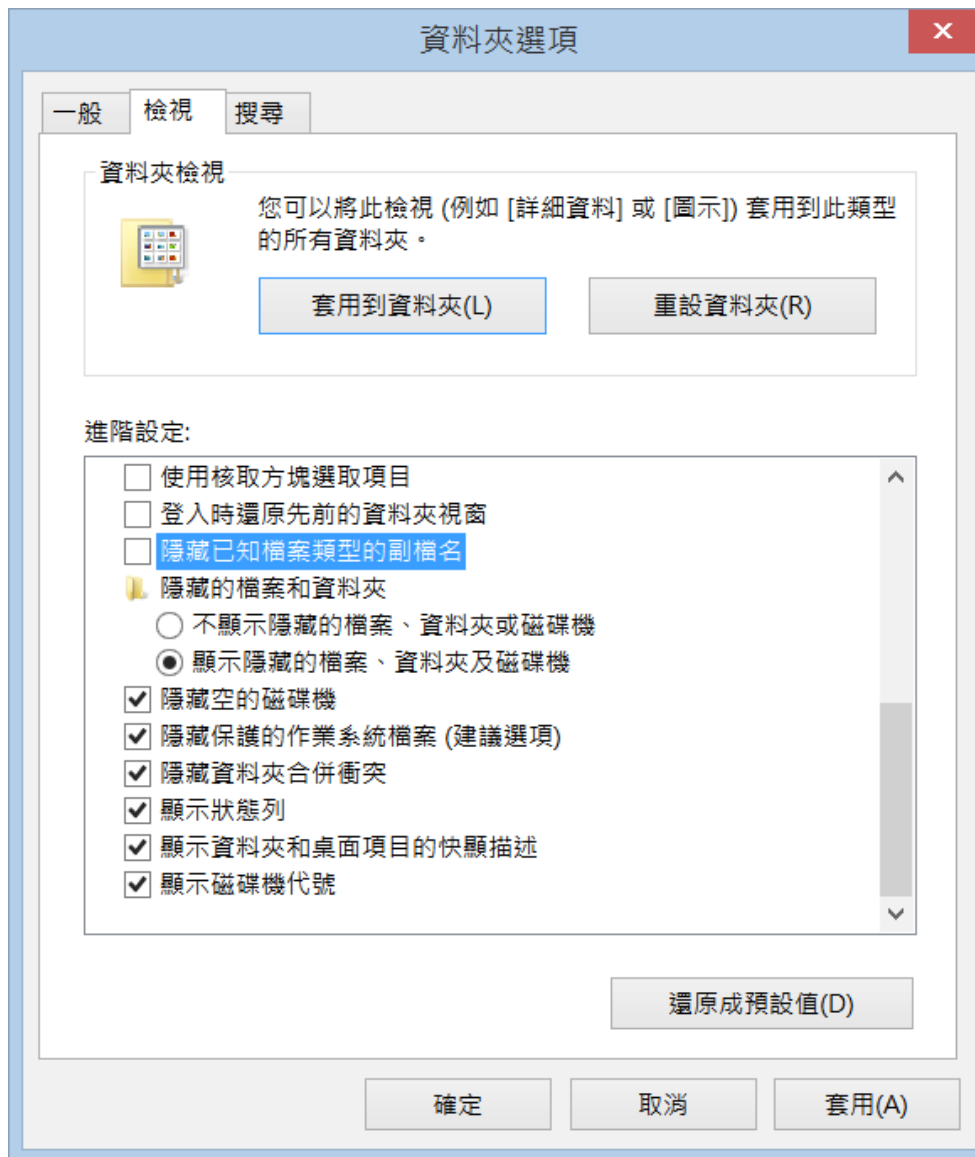
CHAPTER

從 JDK 到 IDE

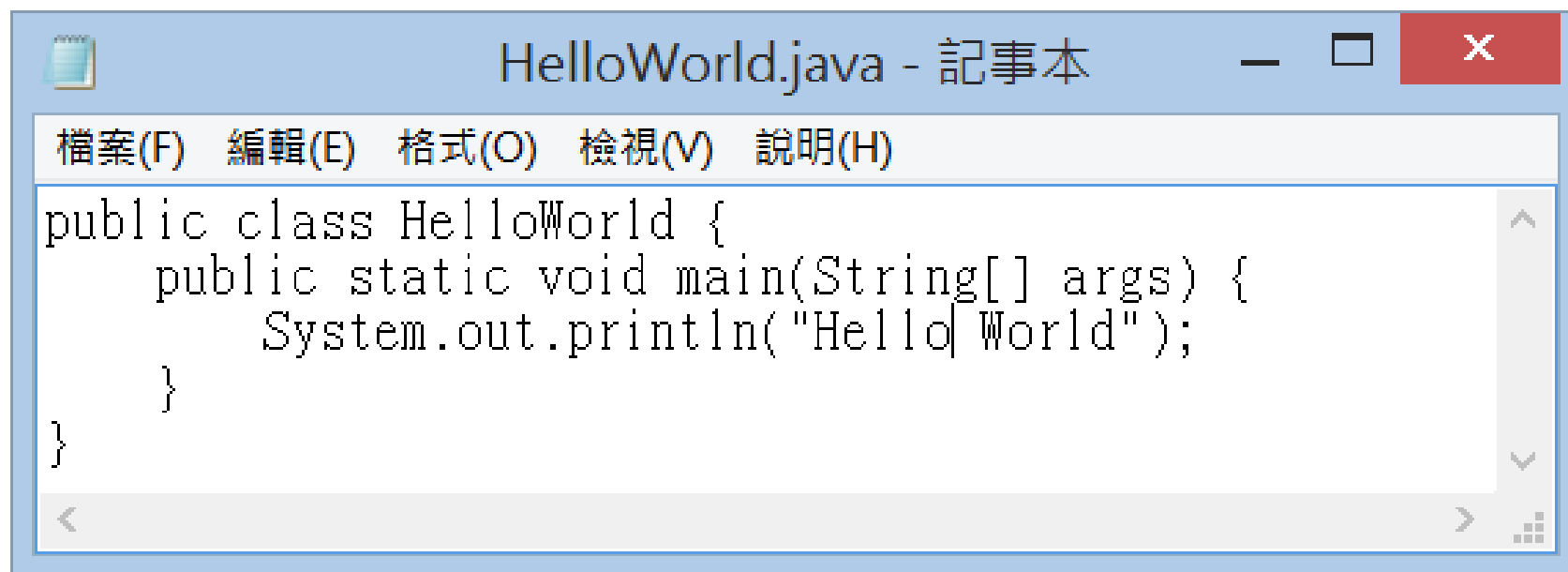
學習目標

- 瞭解與設定 PATH
- 瞭解與指定 CLASSPATH
- 瞭解與指定 SOURCEPATH
- 使用 `package` 與 `import` 管理類別
- 初步認識 JDK 與 IDE 的對應

撰寫 Java 原始碼



撰寫 Java 原始碼

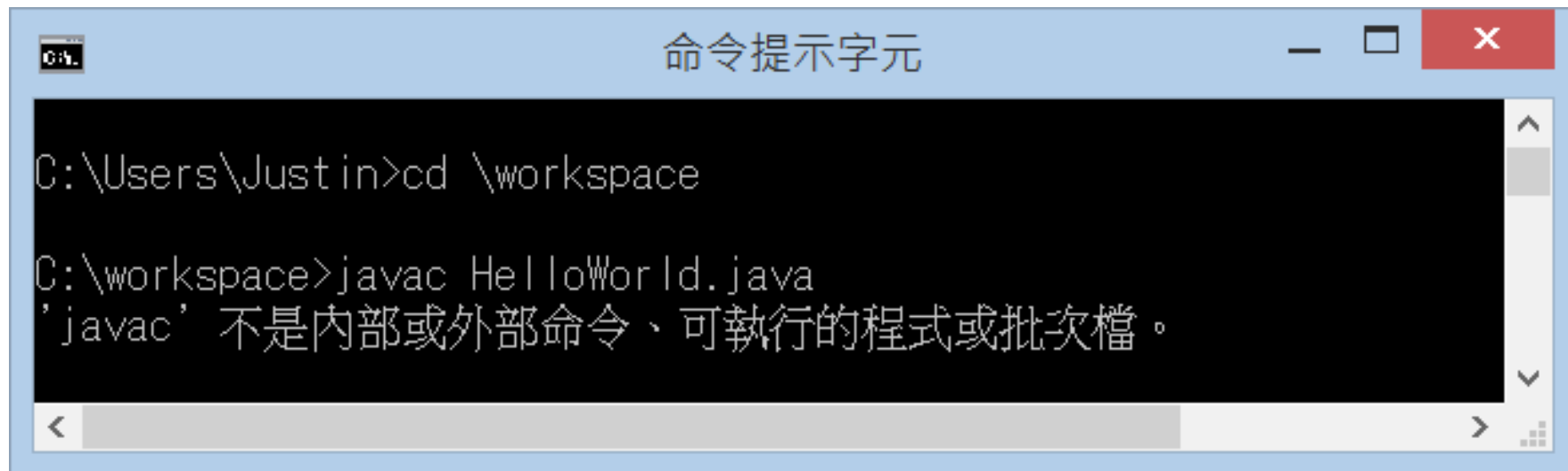


```
public class HelloWorld {  
    public static void main(String[] args) {  
        System.out.println("Hello World");  
    }  
}
```

撰寫 Java 原始碼

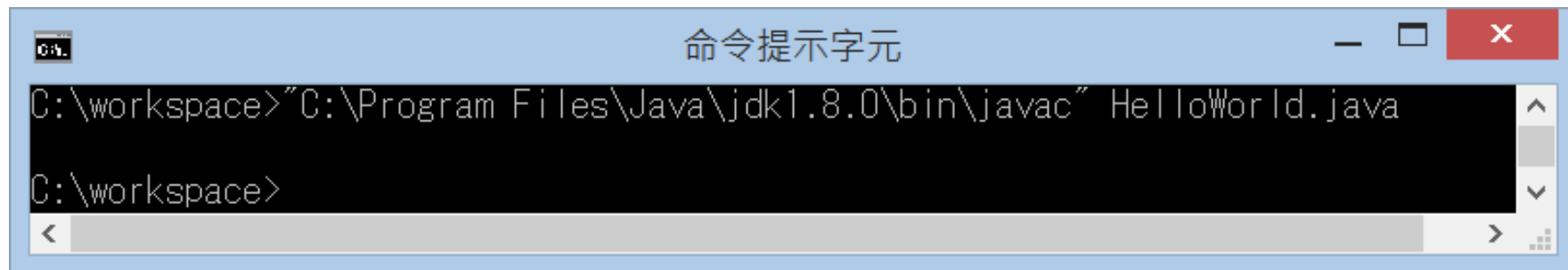
- 副檔名是 .java
- 主檔名與類別名稱必須相同
- 注意每個字母大小寫
- 空白只能是半型空白字元或是 Tab 字元

PATH 是什麼？



```
C:\Users\Justin>cd \workspace

C:\workspace>javac HelloWorld.java
'javac' 不是内部或外部命令、可执行的程式或批次檔。
```



```
C:\workspace>"C:\Program Files\Java\jdk1.8.0\bin\javac" HelloWorld.java

C:\workspace>
```

PATH 是什麼？



The screenshot shows a Windows Command Prompt window titled "選取 命令提示字元". The command prompt is at the directory "C:\workspace". The command "echo %PATH%" has been entered, and the output is displayed as a single line of text, which is truncated in the image. The output shows a list of directories separated by semicolons, including paths for Microsoft Shared, Intel, Windows Live, and various system directories.

```
C:\workspace>echo %PATH%
C:\Program Files\Common Files\Microsoft Shared\Windows Live;C:\Program Files (x86)\Common Files\Microsoft Shared\Windows Live;C:\Program Files (x86)\Intel\iCLS Client\;C:\Program Files\Intel\iCLS Client\;C:\WINDOWS\system32;C:\WINDOWS;C:\WINDOWS\System32\Wbem;C:\WINDOWS\System32\WindowsPowerShell\v1.0\;C:\Program Files\Intel\Intel(R) Management Engine Components\DAL;C:\Program Files\Intel\Intel(R) Management Engine Components\IPT;C:\Program Files (x86)\Intel\Intel(R) Management Engine Components\DAL;C:\Program Files (x86)\Intel\Intel(R) Management Engine Components\IPT;C:\Program Files (x86)\Intel\OpenCL SDK\2.0\bin\x86;C:\Program Files (x86)\Intel\OpenCL SDK\2.0\bin\x64;C:\Program Files\Intel\WiFi\bin\;C:\Program Files\Common Files\Intel\WirelessCommon\;C:\Program Files (x86)\Windows Live\Shared
```

PATH 是什麼？



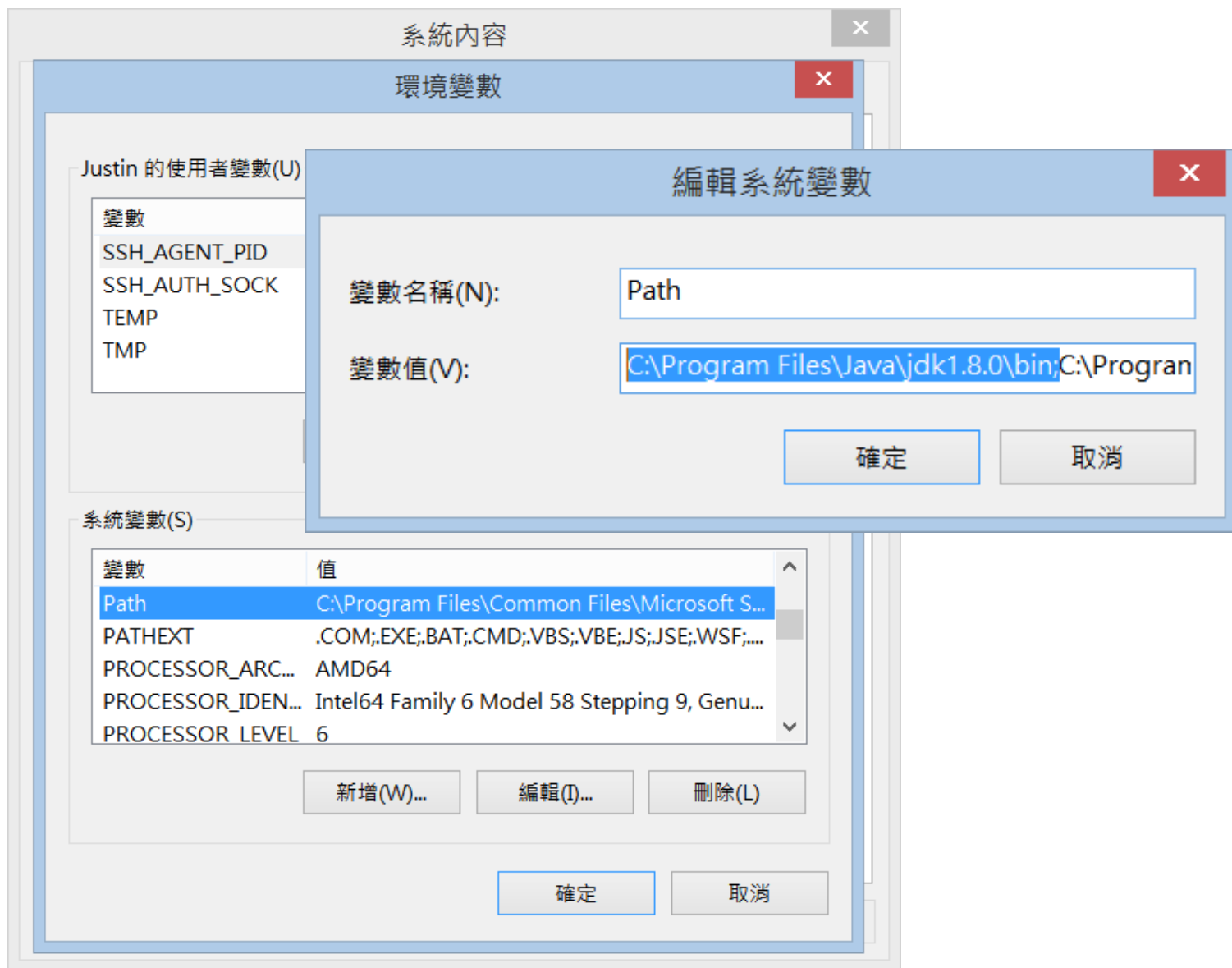
```
C:\workspace>SET PATH="C:\Program Files\Java\jdk1.8.0\bin";%PATH%

C:\workspace>echo %PATH%
"C:\Program Files\Java\jdk1.8.0\bin";C:\Program Files\Common Files\Microsoft Shared\Windows Live;C:\Program Files (x86)\Common Files\Microsoft Shared\Windows Live;C:\Program Files (x86)\Intel\iCLS Client\;C:\Program Files\Intel\iCLS Client\;C:\WINDOWS\system32;C:\WINDOWS;C:\WINDOWS\System32\Wbem;C:\WINDOWS\System32\WindowsPowerShell\v1.0\;C:\Program Files\Intel\Intel(R) Management Engine Components\DAL;C:\Program Files\Intel\Intel(R) Management Engine Components\IPT;C:\Program Files (x86)\Intel\Intel(R) Management Engine Components\DAL;C:\Program Files (x86)\Intel\Intel(R) Management Engine Components\IPT;C:\Program Files (x86)\Intel\OpenCL SDK\2.0\bin\x86;C:\Program Files (x86)\Intel\OpenCL SDK\2.0\bin\x64;C:\Program Files\Intel\WiFi\bin\;C:\Program Files\Common Files\Intel\WirelessCommon\;C:\Program Files (x86)\Windows Live\Shared

C:\workspace>javac HelloWorld.java

C:\workspace>
```

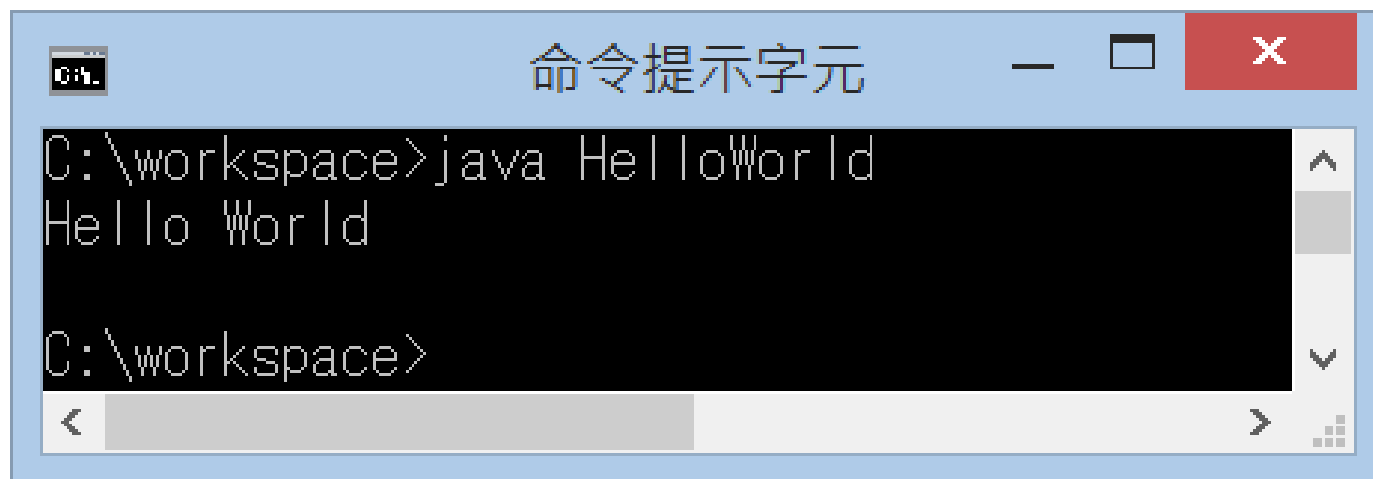

PATH 是什麼？



PATH 是什麼？

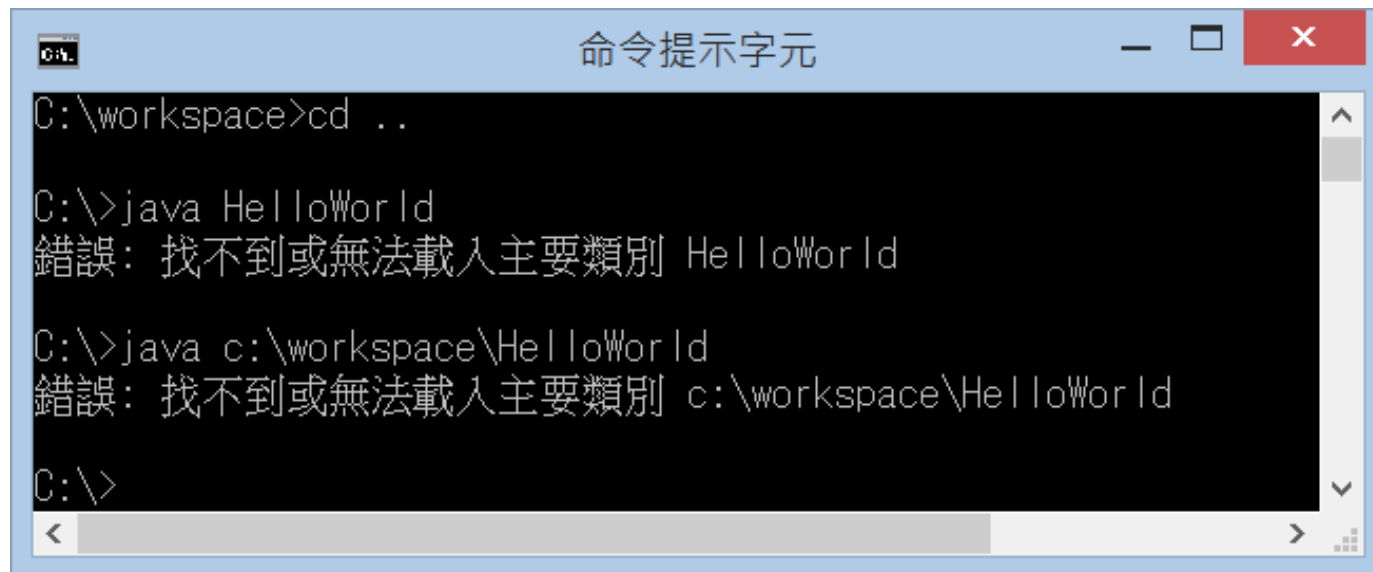
- 建議將 JDK 的 bin 路徑放在 Path 變數的最前方
 - 因為系統搜尋 Path 路徑時，會從最前方開始，如果路徑下找到指定的工具程式就會直接執行
- 若系統中安裝兩個以上 JDK 時，Path 路徑中設定的順序，將決定執行哪個 JDK 下的工具程式
- 在安裝了多個 JDK 或 JRE 的電腦中，確定執行了哪個版本的 JDK 或 JRE 非常重要，確定 PATH 資訊是一定要作的動作

JVM (java) 與 CLASSPATH



```
C:\workspace>java HelloWorld
Hello World

C:\workspace>
```



```
C:\workspace>cd ..
C:\>java HelloWorld
錯誤: 找不到或無法載入主要類別 HelloWorld

C:\>java c:\workspace\HelloWorld
錯誤: 找不到或無法載入主要類別 c:\workspace\HelloWorld

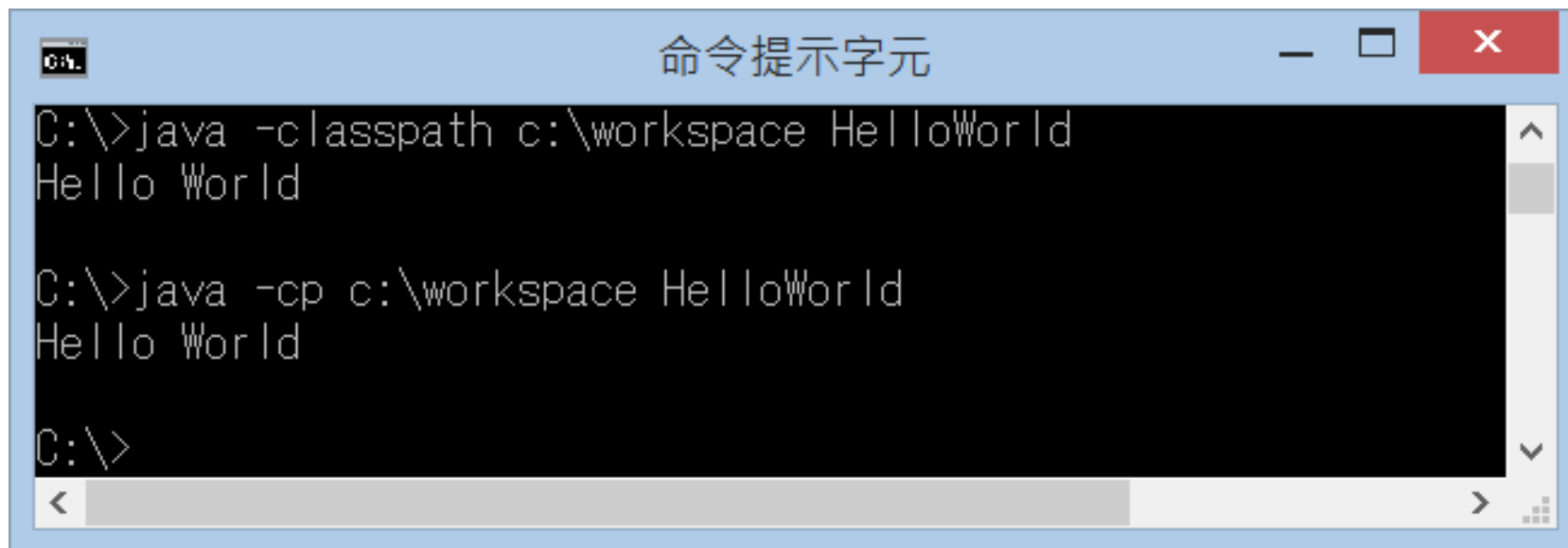
C:\>
```

JVM (java) 與 CLASSPATH

- 實體作業系統下執行某個指令時，會依 PATH 中的路徑資訊，試圖找到可執行檔案
- JVM 是 Java 程式唯一認得的作業系統，對 JVM 來說，可執行檔就是副檔名為 .class 的檔案
- 想在 JVM 中執行某個可執行檔（.class），就要告訴 JVM 這個虛擬作業系統到哪些路徑下尋找檔案，方式是透過 CLASSPATH 指定其可執行檔（.class）的路徑資訊

JVM (java) 與 CLASSPATH

作業系統	搜尋路徑	可執行檔
Windows	PATH	.exe 、 .bat
JVM	CLASSPATH	.class



```
C:\>java -classpath c:\workspace HelloWorld
Hello World

C:\>java -cp c:\workspace HelloWorld
Hello World

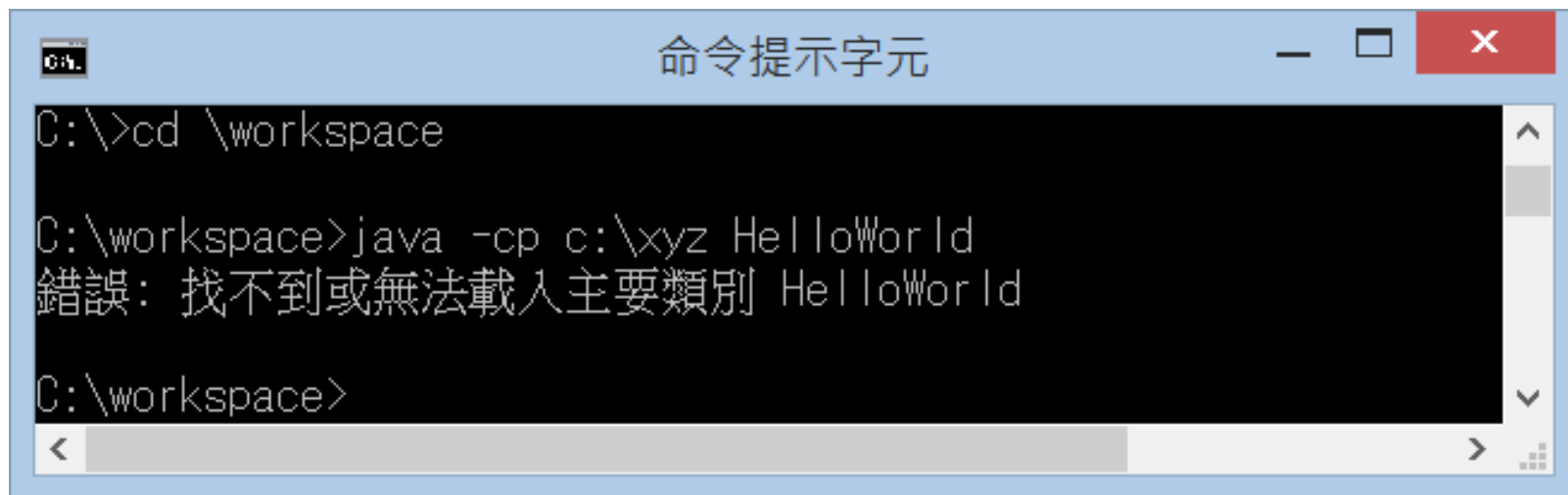
C:\>
```

JVM (java) 與 CLASSPATH

- 如果在 JVM 的 CLASSPATH 路徑資訊中都找不到指定的類別檔案
 - JDK7 前會出現 `java.lang.NoClassDefFoundError` 訊息，而 JDK7 之後的版本會有比較友善的中文錯誤訊息

JVM (java) 與 CLASSPATH

- JVM 預設的 CLASSPATH 就是讀取目前資料夾中的 .class
- 如果自行指定 CLASSPATH，則以你指定的為主



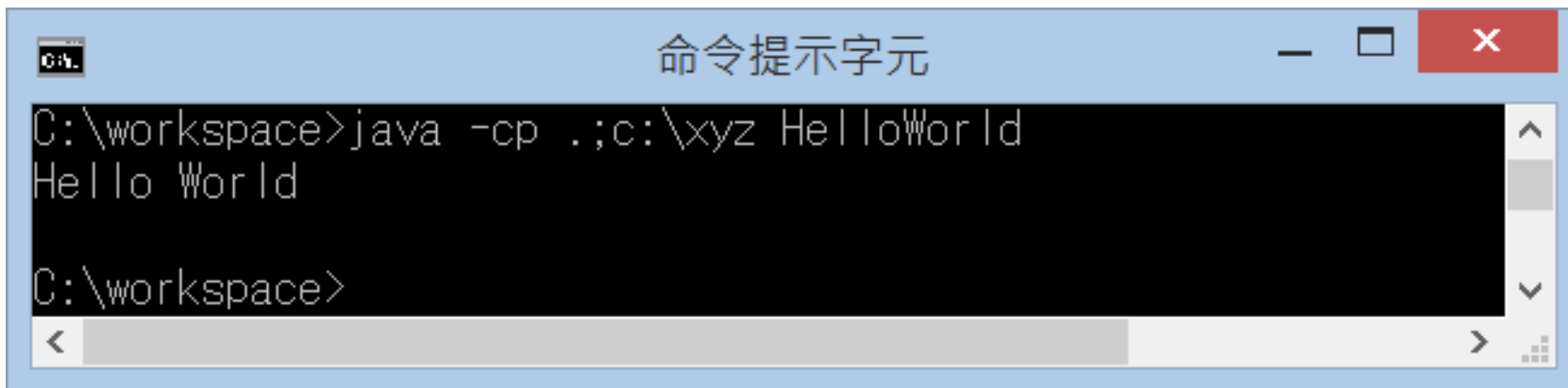
```
C:\>cd \workspace

C:\workspace>java -cp c:\xyz HelloWorld
錯誤: 找不到或無法載入主要類別 HelloWorld

C:\workspace>
```

JVM (java) 與 CLASSPATH

- 希望也從目前資料夾開始尋找類別檔案，則可以使用 . 指定



```
C:\workspace>java -cp .;c:\xyz HelloWorld
Hello World
C:\workspace>
```


JVM (java) 與 CLASSPATH

- 程式庫中的類別檔案，會封裝為 JAR (Java Archive) 檔案，也就是副檔名為 .jar 的檔案
 - 使用 ZIP 格式壓縮，當中包含一堆 .class 檔案
- 例如，有 abc.jar 與 xyz.jar 放在 C:\lib 底下，執行時使用如下檔案中的類別檔案。
`java -cp C:\workspace;C:\lib\abc.jar;C:\lib\xyz.jar SomeApp`

JVM (java) 與 CLASSPATH

- 如果有些類別路徑很常使用，其實也可以透過環境變數設定。例如：

```
SET CLASSPATH=C:\classes;C:\lib\abc.jar;C:\lib\xyz.jar
```

- 在啟動 JVM 時，也就是執行 java 時，若沒使用 -cp 或 -classpath 指定 CLASSPATH，就會讀取 CLASSPATH 環境變數

JVM (java) 與 CLASSPATH

- 從 Java SE 6 開始，可以使用 * 表示使用資料夾中所有 .jar 檔案
- 例如指定使用 C:\jars 下所有 JAR 檔案：

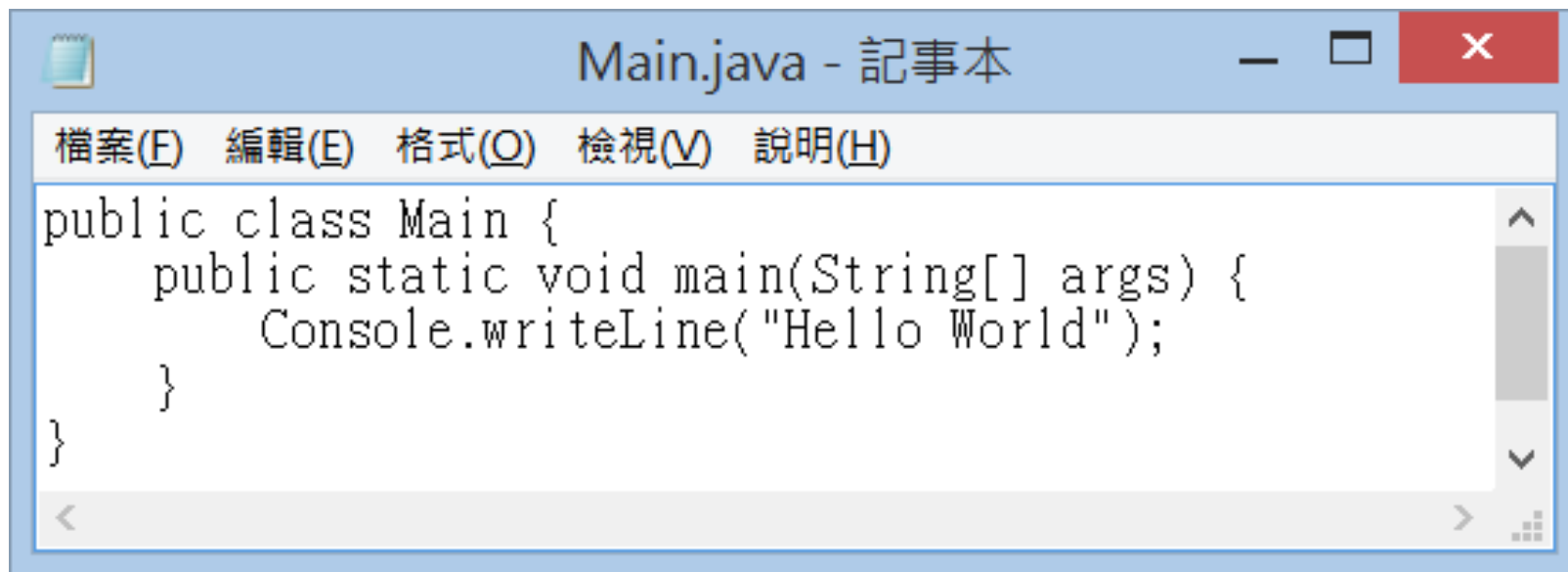
```
java -cp .;C:\jars\* cc.openhome.JNotePad
```

編譯器 (javac) 與 CLASSPATH

- 在光碟中 labs/CH2 資料夾中有個 classes 資料夾，請將之複製至 C:\workspace 中，確認 C:\workspace\classes 中有個已編譯的 Console.class

編譯器 (javac) 與 CLASSPATH

- 可以在 C:\workspace 中開個 Main.java , 如下使用 Console 類別



```
public class Main {  
    public static void main(String[] args) {  
        Console.WriteLine("Hello World");  
    }  
}
```

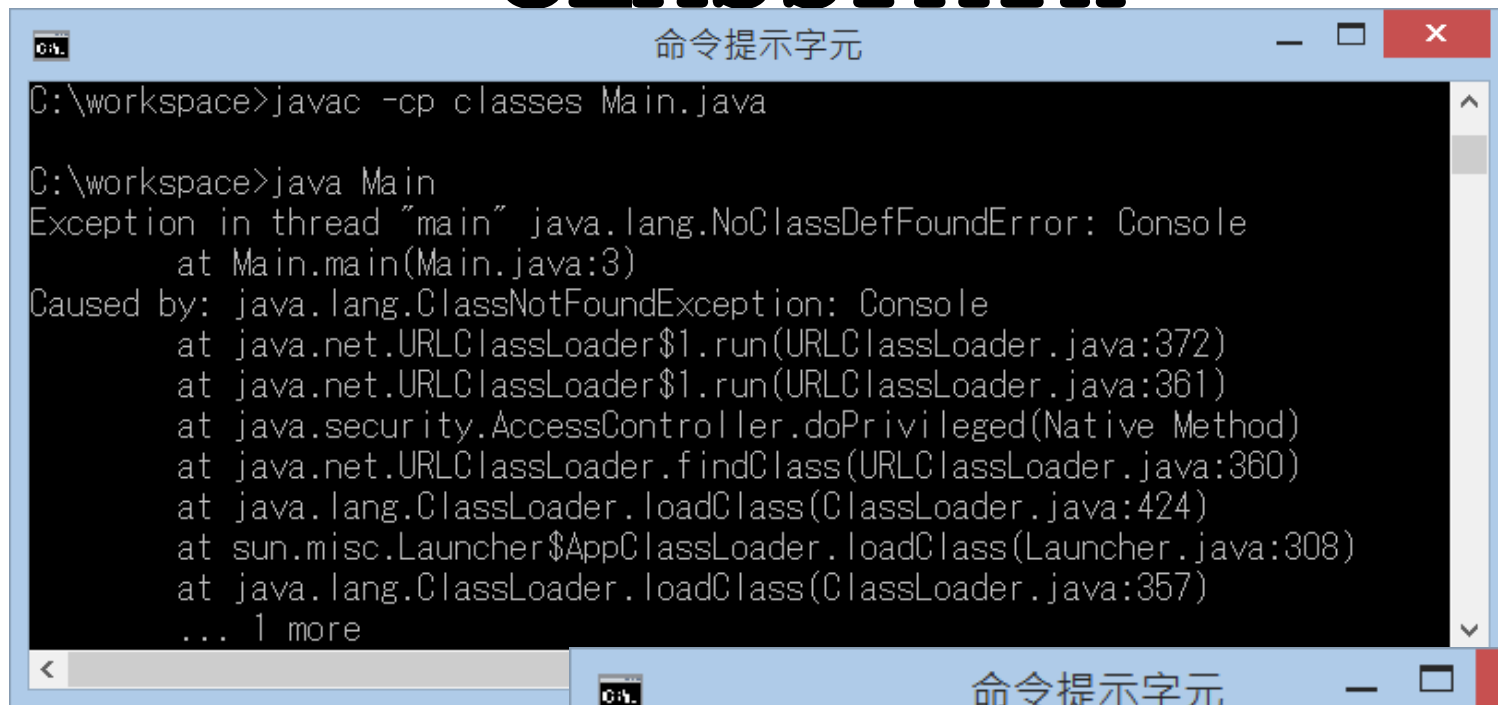
編譯器（javac）與 CLASSPATH



A screenshot of a Windows Command Prompt window titled "命令提示字元". The window shows the command `C:\workspace>javac Main.java` and the resulting error message: `Main.java:3: error: cannot find symbol`. The error points to the `Console` variable in the line `Console.WriteLine("Hello World");`. The error details are: `symbol: variable Console`, `location: class Main`, and `1 error`. The window has a blue title bar, a standard Windows icon, and a red close button. The text is displayed on a black background with white font.

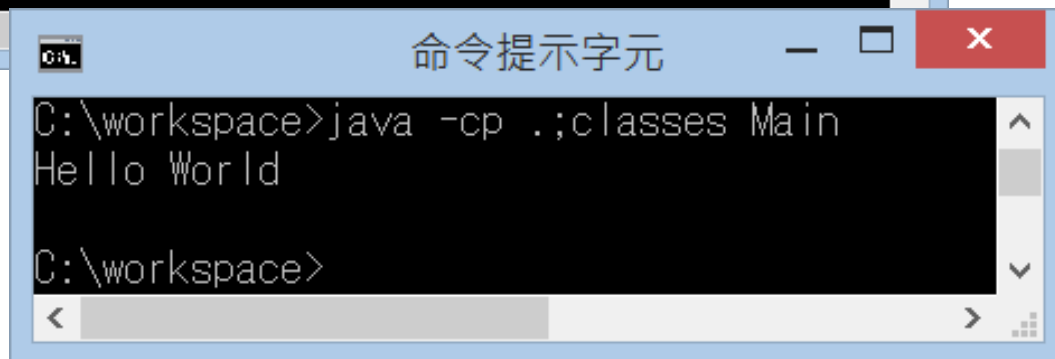
```
C:\workspace>javac Main.java
Main.java:3: error: cannot find symbol
    Console.WriteLine("Hello World");
    ^
symbol:   variable Console
location: class Main
1 error
```

編譯器（javac）與 CLASSPATH



```
命令提示字元
C:\workspace>javac -cp classes Main.java

C:\workspace>java Main
Exception in thread "main" java.lang.NoClassDefFoundError: Console
    at Main.main(Main.java:3)
Caused by: java.lang.ClassNotFoundException: Console
    at java.net.URLClassLoader$1.run(URLClassLoader.java:372)
    at java.net.URLClassLoader$1.run(URLClassLoader.java:361)
    at java.security.AccessController.doPrivileged(Native Method)
    at java.net.URLClassLoader.findClass(URLClassLoader.java:360)
    at java.lang.ClassLoader.loadClass(ClassLoader.java:424)
    at sun.misc.Launcher$AppClassLoader.loadClass(Launcher.java:308)
    at java.lang.ClassLoader.loadClass(ClassLoader.java:357)
    ... 1 more
```



```
命令提示字元
C:\workspace>java -cp .;classes Main
Hello World

C:\workspace>
```

管理原始碼與位元碼檔案

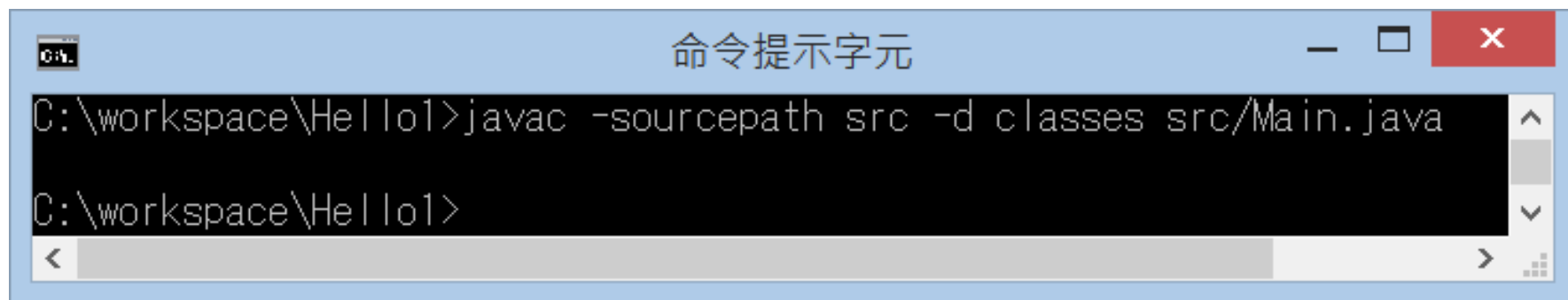
- 來觀察一下目前你的 C:\workspace，原始碼（.java）檔案與位元碼檔案（.class）都放在一起
- 想像一下，如果程式規模稍大，一堆 .java 與 .class 檔案還放在一起，會有多麼混亂
- 你需要有效率地管理原始碼與位元碼檔案

編譯器（`javac`）與 `SOURCEPATH`

- 請將光碟中 labs 資料夾的 Hello1 資料夾複製至 C:\workspace 中
- Hello1 資料夾中有 src 與 classes 資料夾，src 資料夾中有 Console.java 與 Main.java 兩個檔案

編譯器（`javac`）與 `SOURCEPATH`

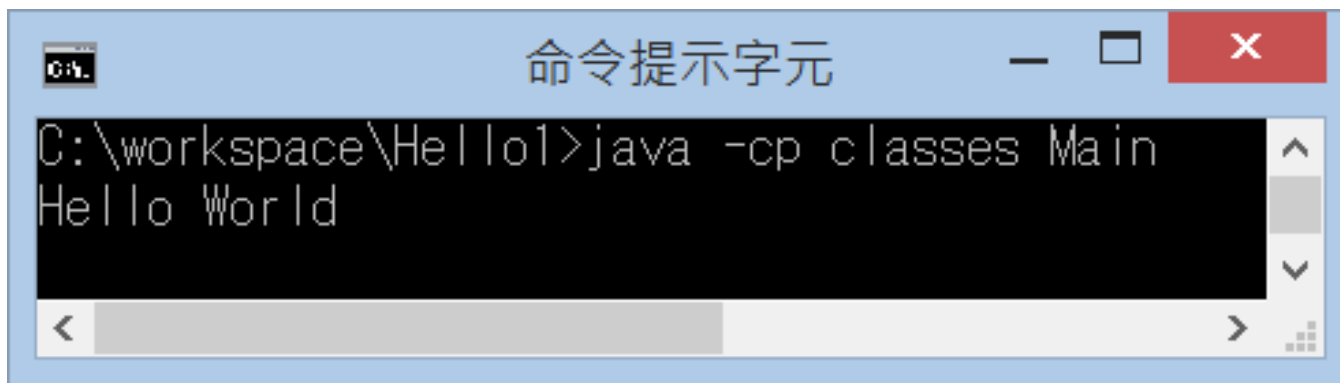
- `src` 資料夾將用來放置原始碼檔案，而編譯好的位元碼檔案，希望能指定存放至 `classes` 資料夾



```
C:\workspace\Hello1>javac -sourcepath src -d classes src/Main.java
C:\workspace\Hello1>
```

編譯器（`javac`）與 `SOURCEPATH`

- 使用 `-sourcepath` 指定從 `src` 資料夾中尋找原始碼檔案，而 `-d` 指定了編譯完成的位元碼存放資料夾
- 編譯器會將使用到的相關類別原始碼也一併進行編譯



```
C:\workspace\Hello1>java -cp classes Main
Hello World
```

編譯器（`javac`）與 `SOURCEPATH`

- 可以在編譯時指定 `-verbose` 引數，看到編譯器進行編譯時的過程

命令提示字元

```
C:\workspace\Hello1>javac -verbose -sourcepath src -d classes src/Main.java
[parsing started RegularFileObject[src/Main.java]]
[parsing completed 28ms]
[search path for source files: src] ❶
[search path for class files: C:\Program Files\Java\jdk1.8.0\jre\lib\resources.jar,C:\Program Files\Java\jdk1.8.0\jre\lib\rt.jar,C:\Program Files\Java\jdk1.8.0\jre\lib\sunrsasign.jar,C:\Program Files\Java\jdk1.8.0\jre\lib\jsse.jar,C:\Program Files\Java\jdk1.8.0\jre\lib\jce.jar,C:\Program Files\Java\jdk1.8.0\jre\lib\charsets.jar,C:\Program Files\Java\jdk1.8.0\jre\lib\jfr.jar,C:\Program Files\Java\jdk1.8.0\jre\classes,C:\Program Files\Java\jdk1.8.0\jre\lib\ext\access-bridge-64.jar,C:\Program Files\Java\jdk1.8.0\jre\lib\ext\cldrdata.jar,C:\Program Files\Java\jdk1.8.0\jre\lib\ext\dnsns.jar,C:\Program Files\Java\jdk1.8.0\jre\lib\ext\jacc.jar,C:\Program Files\Java\jdk1.8.0\jre\lib\ext\jfxrt.jar,C:\Program Files\Java\jdk1.8.0\jre\lib\ext\localedata.jar,C:\Program Files\Java\jdk1.8.0\jre\lib\ext\nashorn.jar,C:\Program Files\Java\jdk1.8.0\jre\lib\ext\sunec.jar,C:\Program Files\Java\jdk1.8.0\jre\lib\ext\sunjce_provider.jar,C:\Program Files\Java\jdk1.8.0\jre\lib\ext\sunmscapi.jar,C:\Program Files\Java\jdk1.8.0\jre\lib\ext\sunpkcs11.jar,C:\Program Files\Java\jdk1.8.0\jre\lib\ext\zipfs.jar,..]
[loading ZipFileIndexFileObject[C:\Program Files\Java\jdk1.8.0\lib\ct.sym(META-I
```

略..

```
[parsing completed 0ms]
[wrote RegularFileObject[classes/Main.class]] ❸
[checking Console]
```

略..

```
NE/sym/rt.jar/java/io/Flushable.class))]
[wrote RegularFileObject[classes/Console.class]] ❹
[total 347ms]
```

編譯器（`javac`）與 `SOURCEPATH`

- 實際專案中會有數以萬計的類別，如果每次都要重新將 `.java` 編譯為 `.class`，那會是非常費時的工作
- 編譯時若類別路徑中已存在位元碼，且上次編譯後，原始碼並沒有修改，無需重新編譯會比較節省時間



命令提示字元



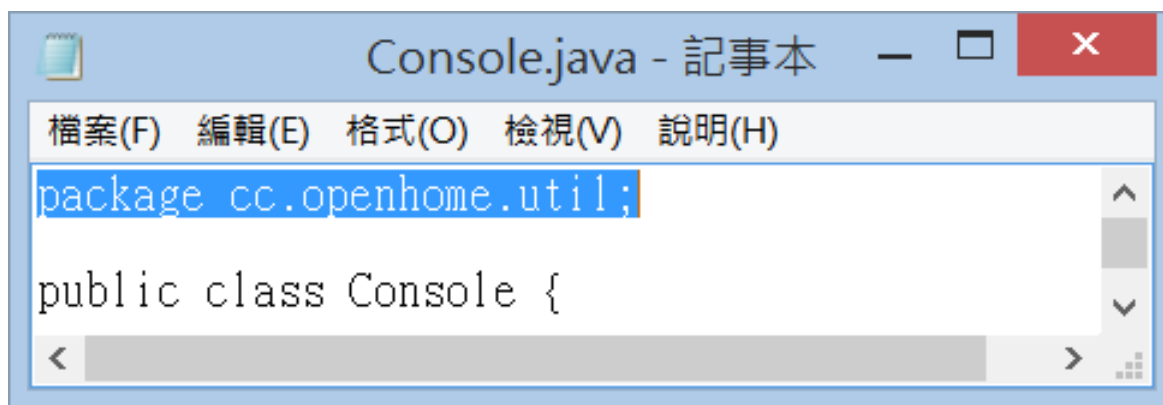
```
C:\workspace\Hello1>javac -verbose -sourcepath src -cp classes -d classes src/Main.java
[parsing started RegularFileObject[src\Main.java]]
[parsing completed 33ms]
[search path for source files: src] ①
[search path for class files: C:\Program Files\Java\jdk1.8.0\jre\lib\resources.jar,C:\Program Files\Java\jdk1.8.0\jre\lib\rt.jar,C:\Program Files\Java\jdk1.8.0\jre\lib\sunrsasign.jar,C:\Program Files\Java\jdk1.8.0\jre\lib\jsse.jar,C:\Program Files\Java\jdk1.8.0\jre\lib\jce.jar,C:\Program Files\Java\jdk1.8.0\jre\lib\charsets.jar,C:\Program Files\Java\jdk1.8.0\jre\lib\jfr.jar,C:\Program Files\Java\jdk1.8.0\jre\classes,C:\Program Files\Java\jdk1.8.0\jre\lib\ext\access-bridge-64.jar,C:\Program Files\Java\jdk1.8.0\jre\lib\ext\cldrdata.jar,C:\Program Files\Java\jdk1.8.0\jre\lib\ext\dnsns.jar,C:\Program Files\Java\jdk1.8.0\jre\lib\ext\jacc.jar,C:\Program Files\Java\jdk1.8.0\jre\lib\ext\jfxrt.jar,C:\Program Files\Java\jdk1.8.0\jre\lib\ext\localedata.jar,C:\Program Files\Java\jdk1.8.0\jre\lib\ext\nashorn.jar,C:\Program Files\Java\jdk1.8.0\jre\lib\ext\sunec.jar,C:\Program Files\Java\jdk1.8.0\jre\lib\ext\sunjce_provider.jar,C:\Program Files\Java\jdk1.8.0\jre\lib\ext\sunmscapi.jar,C:\Program Files\Java\jdk1.8.0\jre\lib\ext\sunpkcs11.jar,C:\Program Files\Java\jdk1.8.0\jre\lib\ext\zipfs.jar_classes] ②
略..
[loading ZipFileIndexFileObject[C:\Program Files\Java\jdk1.8.0\lib\ct.sym(META-INF/sym/rt.jar/java/lang/Void.class)]]
[loading RegularFileObject[classes\Console.class]]
[wrote RegularFileObject[classes\Main.class]] ③
[total 323ms]
```

使用 `package` 管理類別

- `.java` 放在 `src` 資料夾中，編譯出來的 `.class` 放置在 `classes` 資料夾下
- 就如同你會分不同資料夾來放置不同作用的檔案，類別也應該分門別類加以放置，
- 無論是實體檔案上的分類管理，或是類別名稱上的分類管理，有個 `package` 關鍵字，可以協助你達到這個目的

使用 package 管理類別

- 用編輯器開啟 2.2.2 中 Hello1/src 資料夾中的 Console.java，在開頭鍵入下圖反白的文字：

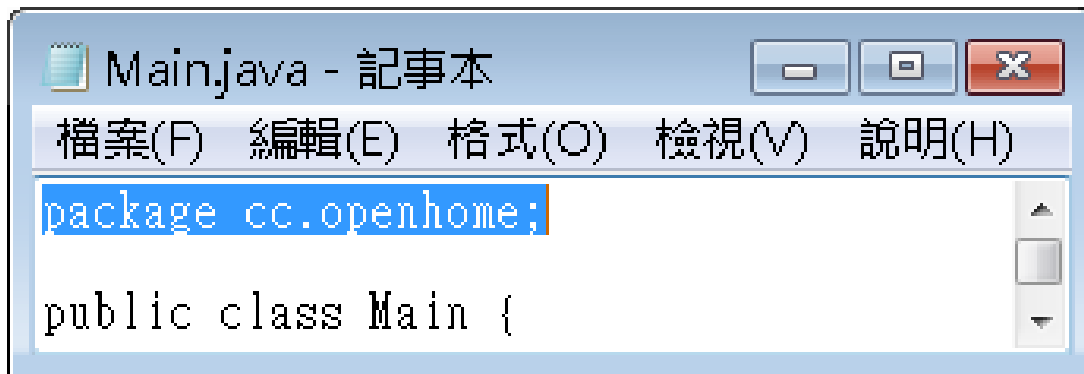


```
package cc.openhome.util;  
  
public class Console {
```

- Console 類別將放在 cc.openhome.util 的分類下，以 Java 的術語來說，Console 這個類別將放在 cc.openhome.util 套件（package）

使用 package 管理類別

- 再用文字編輯器開啟 2.2.2 中 Hello1/src 資料夾中的 Main.java，在開頭鍵入下圖反白的文字



```
package cc.openhome;

public class Main {
```

- 這表示 Main 類別將放在 cc.openhome 的分類下

使用 `package` 管理類別

- 原始碼檔案要放置在與 `package` 所定義名稱階層相同的資料夾階層
- `package` 所定義名稱與 `class` 所定義名稱，會結合而成類別的完全吻合名稱（**Fully qualified name**）
- 位元碼檔案要放置在與 `package` 所定義名稱階層相同的資料夾階層
- 要在套件間可以直接使用的類別或方法（`Method`）必須宣告為 `public`

原始碼檔案與套件管理

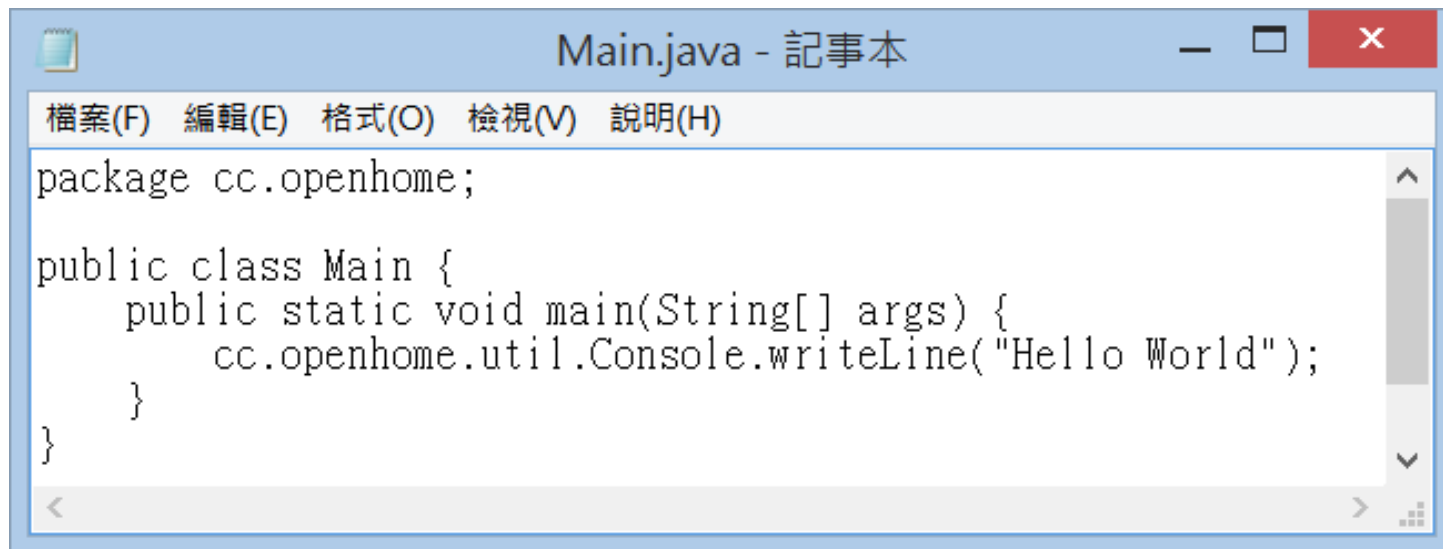
- 由於 Console 類別使用 package 定義在 `cc.openhome.util` 套件下，所以 `Console.java` 必須放在 src 資料夾中的 `cc/openhome/util` 資料夾
- Main 類別使用 package 定義在 `cc.openhome` 套件下，所以 `Main.java` 必須放在 src 資料夾中的 `cc/openhome` 資料夾

完全吻合名稱（Fully qualified name）

- Main 類別是位於 `cc.openhome` 套件分類中，其完全吻合名稱是
`cc.openhome.Main`
- Console 類別是位於 `cc.openhome.util` 分類中，其完全吻合名稱為
`cc.openhome.util.Console`

完全吻合名稱 (Fully qualified name)

- 如果是相同套件中的類別，只要使用 `class` 所定義的名稱即可
- 不同套件的類別，必須使用完全吻合名稱
- 由於 `Main` 與 `Console` 類別是位於不同的套件中



```
package cc.openhome;

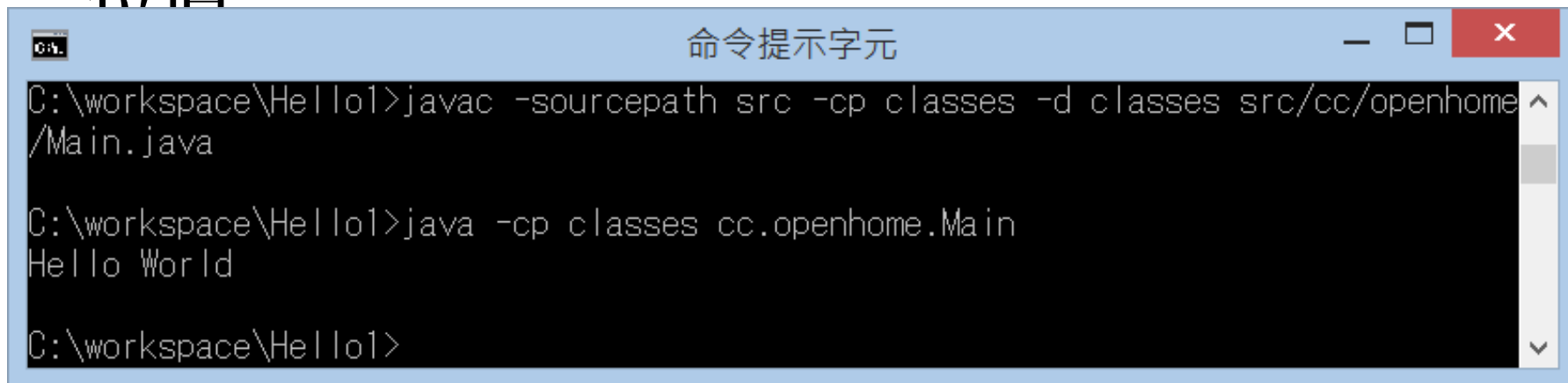
public class Main {
    public static void main(String[] args) {
        cc.openhome.util.Console.WriteLine("Hello World");
    }
}
```

位元碼檔案與套件管理

- 由於 Console 類別使用 package 定義在 `cc.openhome.util` 套件下，所以編譯出來的 `Console.class` 必須放在 `classes` 資料夾中的 `cc/openhome/util` 資料夾
- `Main` 類別使用 package 定義在 `cc.openhome` 套件下，所以 `Main.class` 必須放在 `classes` 資料夾中的 `cc/openhome` 資料夾

位元碼檔案與套件管理

- 在編譯時若有使用 `-d` 指定位元碼的存放位置，就會自動建立出對應套件階層的資料夾，並將編譯出來的位元碼檔案放置至應有的位置



```
命令提示字元

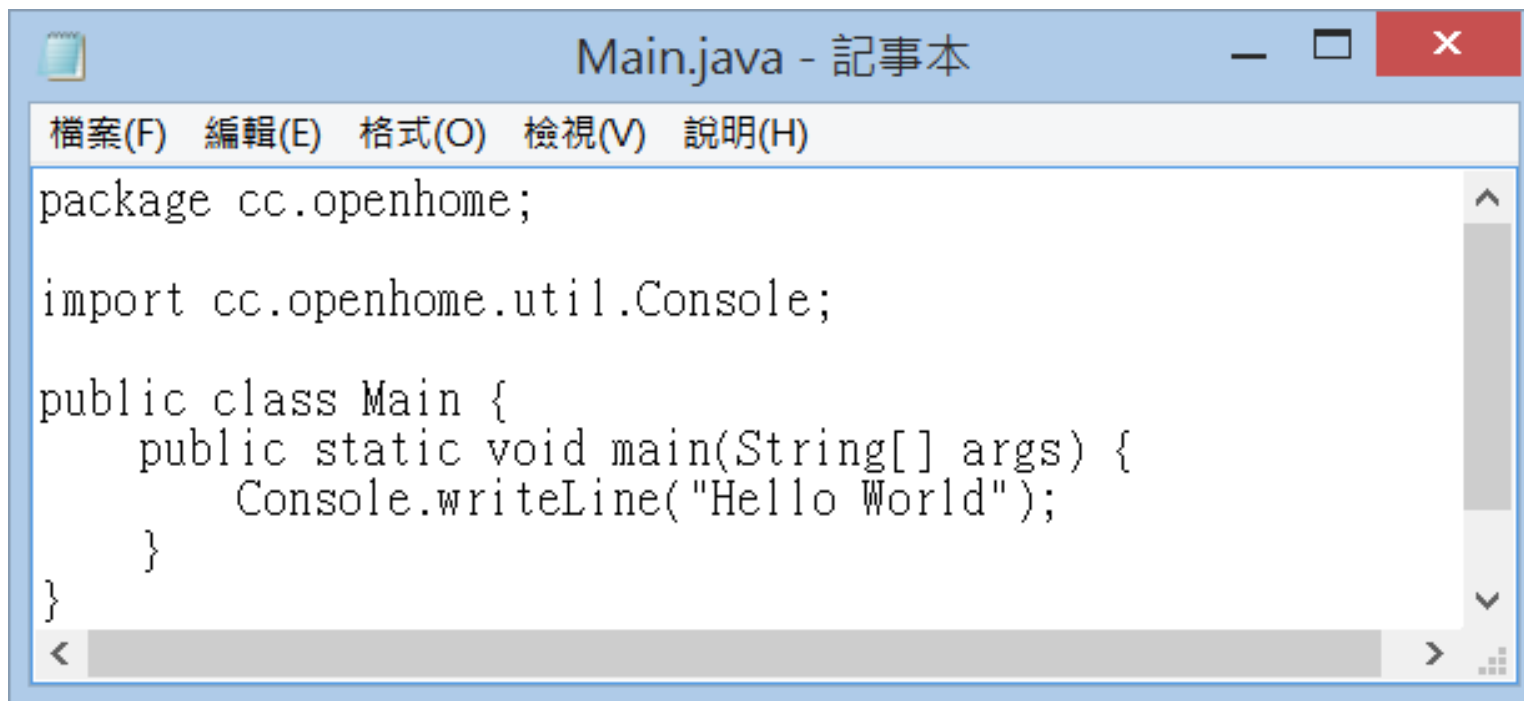
C:\workspace\Hello1>javac -sourcepath src -cp classes -d classes src/cc/openhome/Main.java

C:\workspace\Hello1>java -cp classes cc.openhome.Main
Hello World

C:\workspace\Hello1>
```


使用 `import` 偷懶

- 每次撰寫程式時，都得鍵入完全吻合名稱，也是件麻煩的事 …



```
package cc.openhome;

import cc.openhome.util.Console;

public class Main {
    public static void main(String[] args) {
        Console.WriteLine("Hello World");
    }
}
```

使用 `import` 偷懶

- `import` 只是告訴編譯器，遇到不認識的類別名稱，可以嘗試使用 `import` 過的名稱
- `import` 讓你少打一些字，讓編譯器多為你作一些事

使用 `import` 偷懶

- 如果同一套件下會使用到多個類別，你也許會多次使用 `import`：

```
import cc.openhome.Message;  
import cc.openhome.User;  
import cc.openhome.Address;
```

- 你可以更偷懶一些，用以下的 `import` 語句

```
import cc.openhome.*;
```

使用 `import` 偷懶

- 偷懶也是有個限度，如果你自己寫了一個

Arrays：

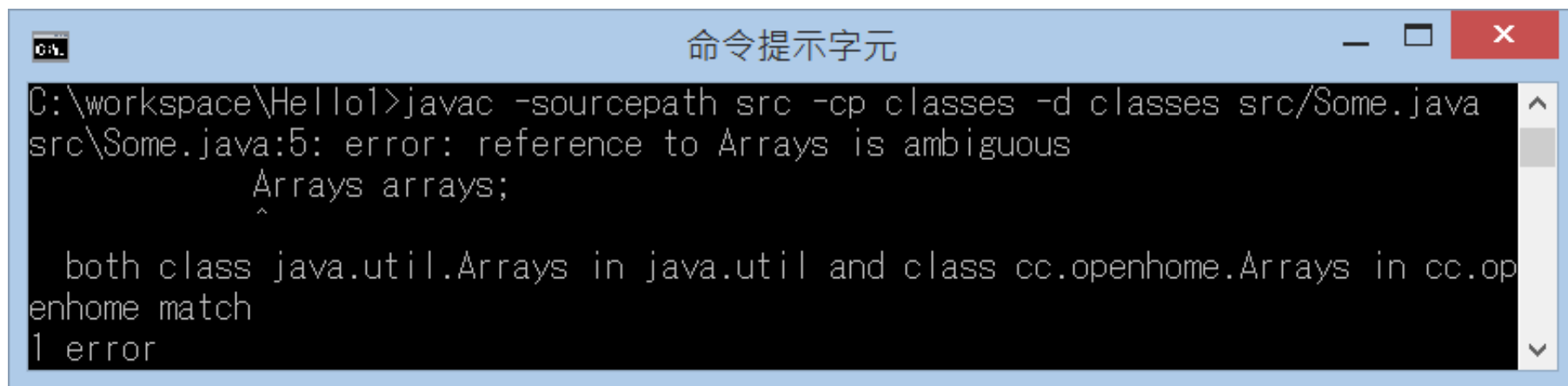
```
package cc.openhome;  
public class Arrays {  
    ...  
}
```

- 若在某個類別中撰寫有以下的程式碼：

```
import cc.openhome.*;  
import java.util.*;  
public class Some {  
    public static void main(String[] args) {  
        Arrays arrays;  
        ...  
    }  
}
```

使用 `import` 偷懶

- 底該使用 `cc.openhome.Arrays` 還是 `java.util.Arrays` ?



A screenshot of a Windows Command Prompt window titled "命令提示字元". The window shows the following text:

```
C:\workspace\Hello1>javac -sourcepath src -cp classes -d classes src/Some.java
src\Some.java:5: error: reference to Arrays is ambiguous
    Arrays arrays;
    ^
   both class java.util.Arrays in java.util and class cc.openhome.Arrays in cc.op
enhome match
1 error
```

使用 `import` 偷懶

- 遇到這種情況時，就不能偷懶了，你要使用哪個類別名稱，就得明確地逐字打出來：

```
import cc.openhome.*;
import java.util.*;
public class Some {
    public static void main(String[] args) {
        cc.openhome.Arrays arrays;
        ...
    }
}
```

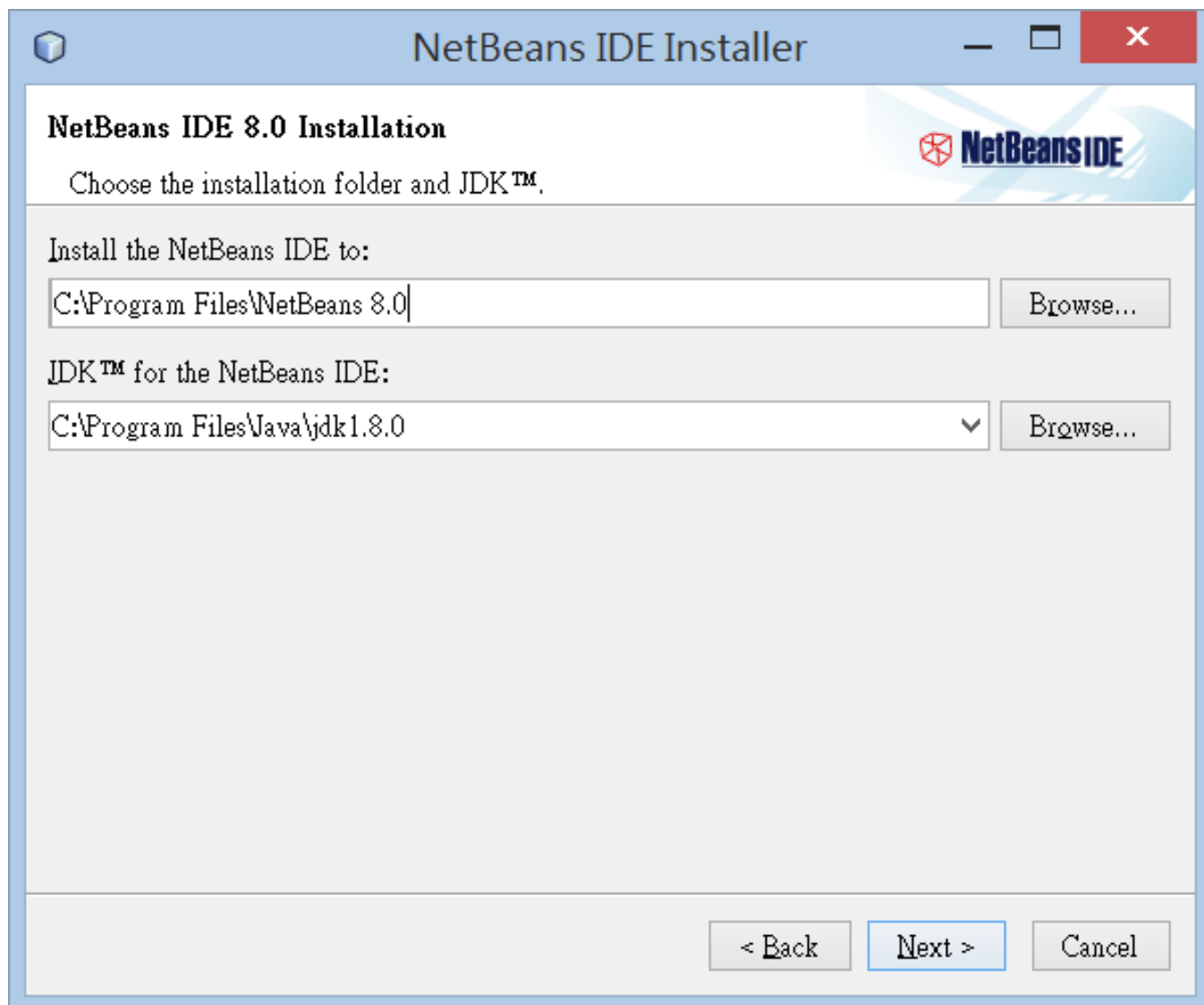
使用 `import` 偷懶

- 寫第一個 Java 程式時使用的 `System` 類別，其實也有使用套件管理，完整名稱其實是 `java.lang.System`
- 在 `java.lang` 套件下的類別由於很常用，不用撰寫 `import` 也可以直接使用 `class` 定義的名稱

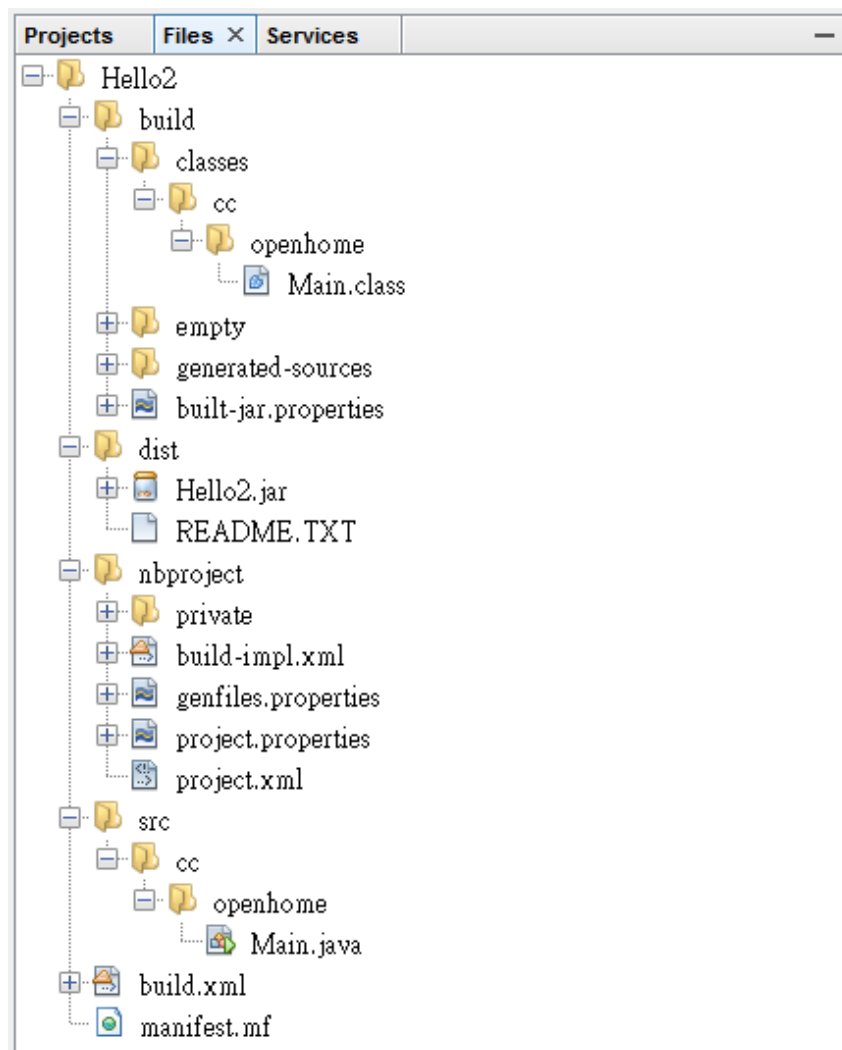
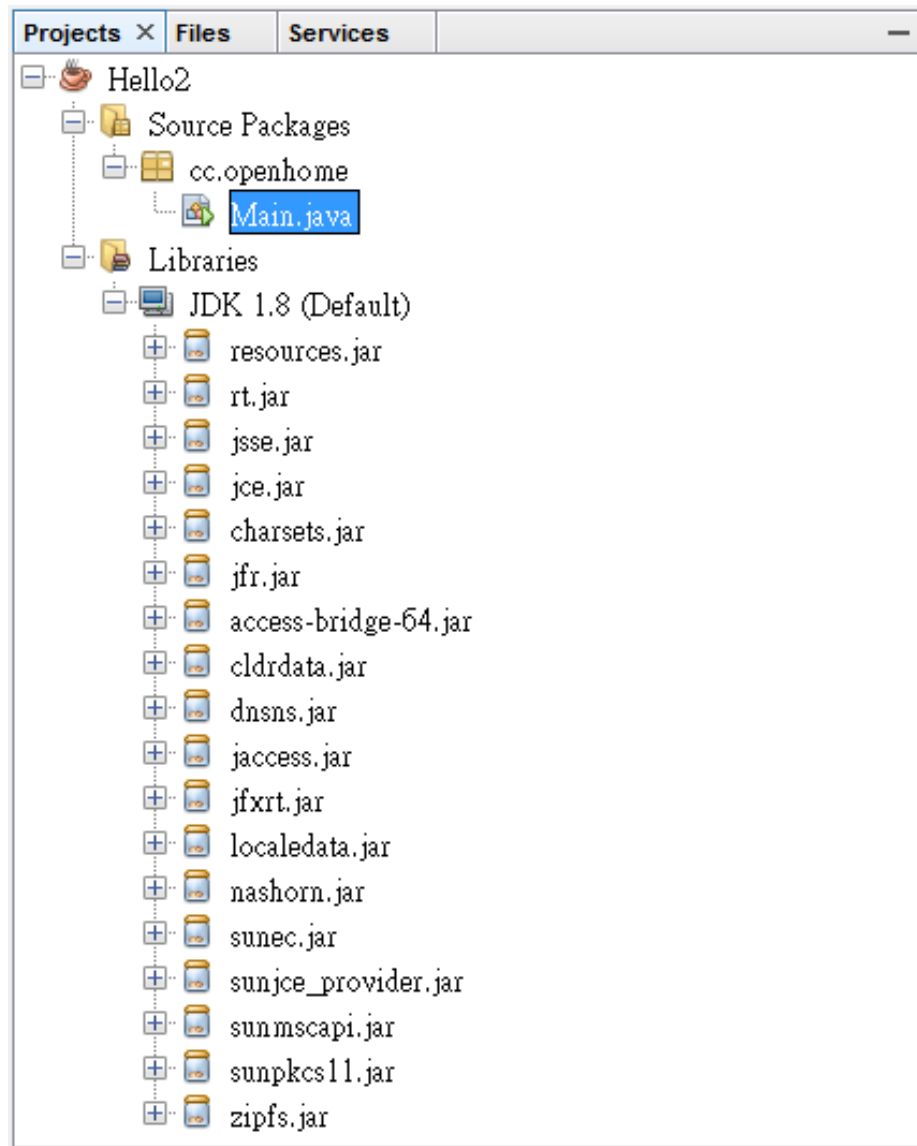
使用 `import` 偷懶

- 當編譯器看到一個沒有套件管理的類別名稱，會先在同一套件中尋找類別，如果找到就使用，若沒找到，再試著從 `import` 陳述進行比對
- `java.lang` 可視為預設就有 `import`，沒有寫任何 `import` 陳述時，也會試著比對 `java.lang` 組合，看看是否能找到對應類別

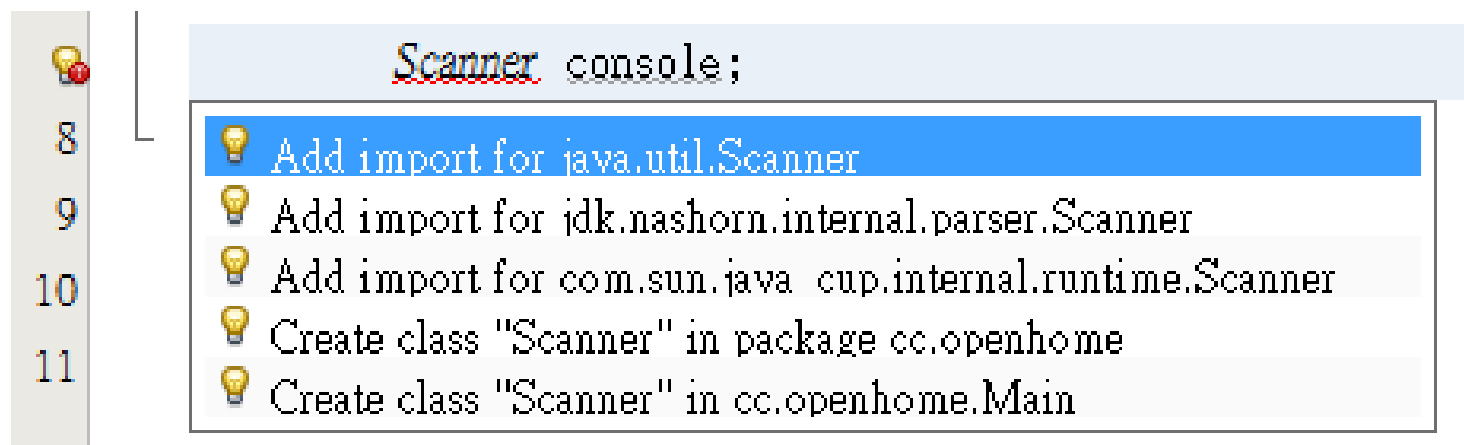
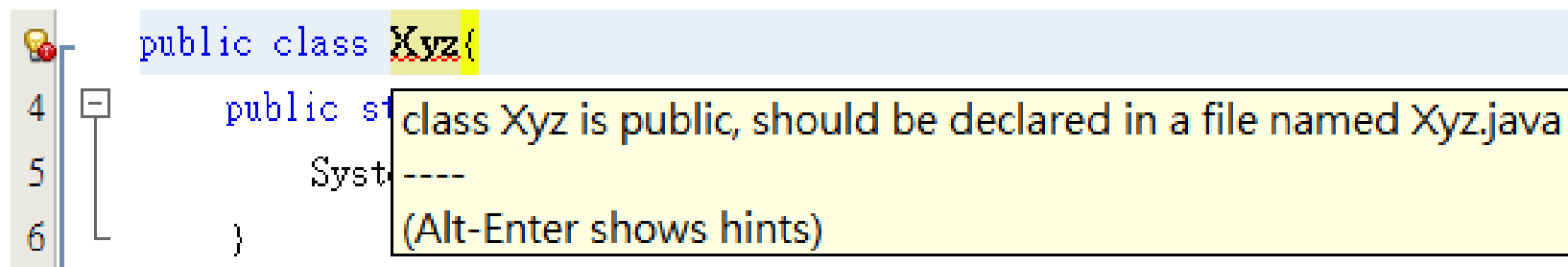
IDE 專案管理基礎



IDE 專案管理基礎



IDE 專案管理基礎

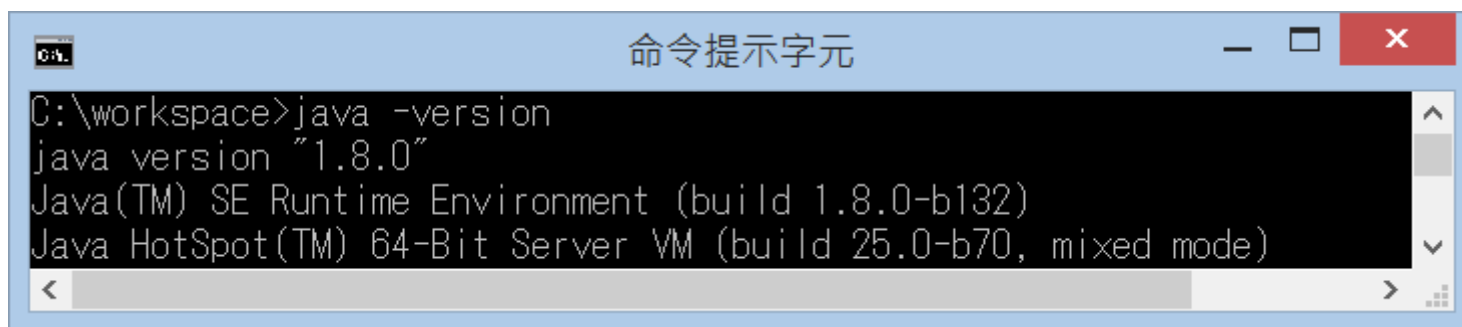


使用了哪個 JRE ？

- 因為各種原因，你的電腦中可能不只存在一套 JRE ！
- 在文字模式下鍵入 `java` 指令，如果設定了 `PATH`，會執行 `PATH` 順序下找到的第一個 `java` 可執行檔，這個可執行檔所啟動的是哪套 JRE ？

使用了哪個 JRE ？

- 當找到 java 可執行檔並執行時，會依照以下的規則來尋找可用的 JRE：
 - 可否在 java 可執行檔資料夾下找到相關原生（Native）程式庫
 - 可否在上一層目錄中找到 jre 目錄
- 在執行 java 指令時，可附帶一個 -version 引數，這可以顯示執行的 JRE 版本



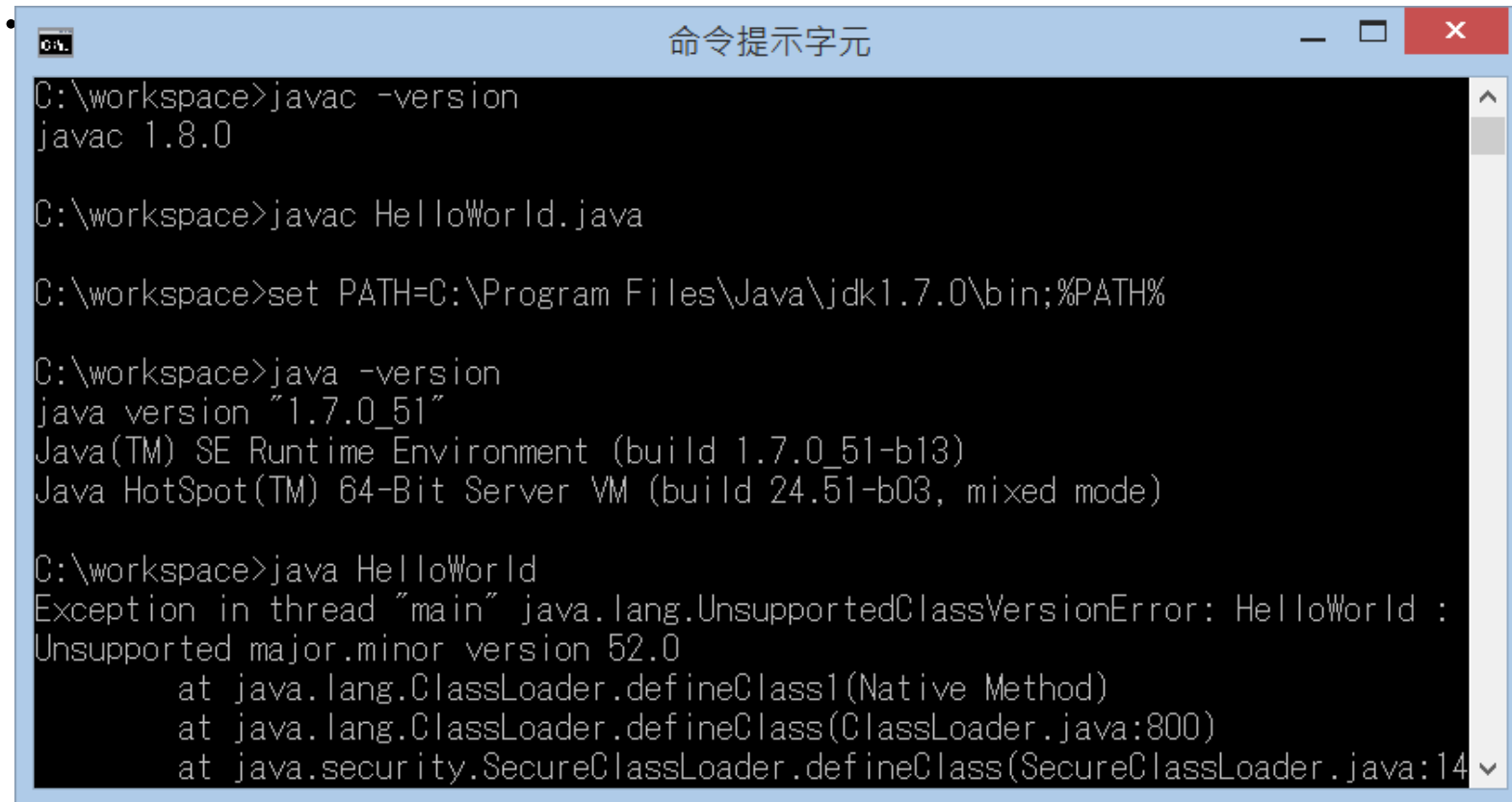
```
命令提示字元
C:\workspace>java -version
java version "1.8.0"
Java(TM) SE Runtime Environment (build 1.8.0-b132)
Java HotSpot(TM) 64-Bit Server VM (build 25.0-b70, mixed mode)
```

使用了哪個 JRE ？

- 如果有個需求是切換 JRE，文字模式下必須設定 PATH 順序中，找到的第一個 JRE 之 bin 資料夾是你想要的 JRE，而不是設定 CLASSPATH
- 如果使用 IDE 新增專案，你使用了哪個 JRE 呢？

類別檔案版本

- 如果使用新版本 JDK 編譯出位元碼檔案，在舊版本 JRE 上執行，可能發生以下錯誤訊息



```
命令提示字元

C:\workspace>javac -version
javac 1.8.0

C:\workspace>javac HelloWorld.java

C:\workspace>set PATH=C:\Program Files\Java\jdk1.7.0\bin;%PATH%

C:\workspace>java -version
java version "1.7.0_51"
Java(TM) SE Runtime Environment (build 1.7.0_51-b13)
Java HotSpot(TM) 64-Bit Server VM (build 24.51-b03, mixed mode)

C:\workspace>java HelloWorld
Exception in thread "main" java.lang.UnsupportedClassVersionError: HelloWorld :
Unsupported major.minor version 52.0
    at java.lang.ClassLoader.defineClass1(Native Method)
    at java.lang.ClassLoader.defineClass(ClassLoader.java:800)
    at java.security.SecureClassLoader.defineClass(SecureClassLoader.java:144)
```

類別檔案版本

- 編譯器會在位元碼檔案中標示主版本號與次版本號，不同的版本號，位元碼檔案格式可能有所不同
- JVM 在載入位元碼檔案後，會確認其版本號是否在可接受的範圍，否則就不會處理該位元碼檔案

類別檔案版本

- 可以使用 JDK 工具程式 `javap`，確認位元碼檔案的版本號：



```
C:\workspace>javap -v HelloWorld
Classfile /C:/workspace/HelloWorld.class
  Last modified 2014/3/8; size 425 bytes
  MD5 checksum 63e47f1d243e0eb6bc952df3f6ac0d5a
  Compiled from "HelloWorld.java"
public class HelloWorld
  SourceFile: "HelloWorld.java"
  minor version: 0
  major version: 52
```

類別檔案版本

- `System.getProperty("java.class.version")` 可取得 JRE 支援的位元碼版本號
- `System.getProperty("java.runtime.version")` 可取得 JRE 版本訊息

類別檔案版本

- 在編譯的時候，可以使用 `-target` 指定編譯出來的位元碼，必須符合指定平台允許的版本號
- 使用 `-source` 要求編譯器檢查使用的語法，不超過指定的版本

類別檔案版本



```
C:\workspace>javac -source 1.7 -target 1.7 HelloWorld.java
warning: [options] bootstrap class path not set in conjunction with -source 1.7
1 warning

C:\workspace>javac -bootclasspath "C:\Program Files\Java\jre7\lib\rt.jar" -source 1.7 -target 1.7 HelloWorld.java

C:\workspace>javac -version
javac 1.8.0

C:\workspace>set PATH=C:\Program Files\Java\jdk1.7.0\bin;%PATH%

C:\workspace>java -version
java version "1.7.0_51"
Java(TM) SE Runtime Environment (build 1.7.0_51-b13)
Java HotSpot(TM) 64-Bit Server VM (build 24.51-b03, mixed mode)

C:\workspace>java HelloWorld
Hello World
```

類別檔案版本

- 在不指定 `-target` 與 `-source` 的情況下，編譯器會有預設的 `-target` 值
- **JDK8 預設的 `-target` 與 `-source` 都是 1.8**
- **`-target` 在指定時，值必須大於或等於 `-source`**

類別檔案版本

- 如果只指定 `-source` 與 `-target` 進行編譯，會出現警示訊息，這是因為編譯時預設的 Bootstrap 類別載入器（Class loader）沒有改變
- 系統預設的類別載入器仍參考至 1.8 的 `rt.jar`（也就是 Java SE 8 API 的 JAR 檔案）
- 如果引用到一些舊版 JRE 沒有的新 API，就會造成在舊版 JRE 上無法執行
- 最好是編譯時指定 `-bootclasspath`，參考至舊版的 `rt.jar`，這樣在舊版 JRE 執行時比較不會發生問題

類別檔案版本

- 如果你已經安裝有舊版 JDK 或 JRE，可以在執行時使用 `-version` 引數並指定版本。例如…

```
C:\workspace>javac -version  
javac 1.8.0
```

```
C:\workspace>javac HelloWorld.java
```

```
C:\workspace>java -version:1.7 HelloWorld  
Exception in thread "main" java.lang.UnsupportedClassVersionError: HelloWorld :  
Unsupported major.minor version 52.0
```

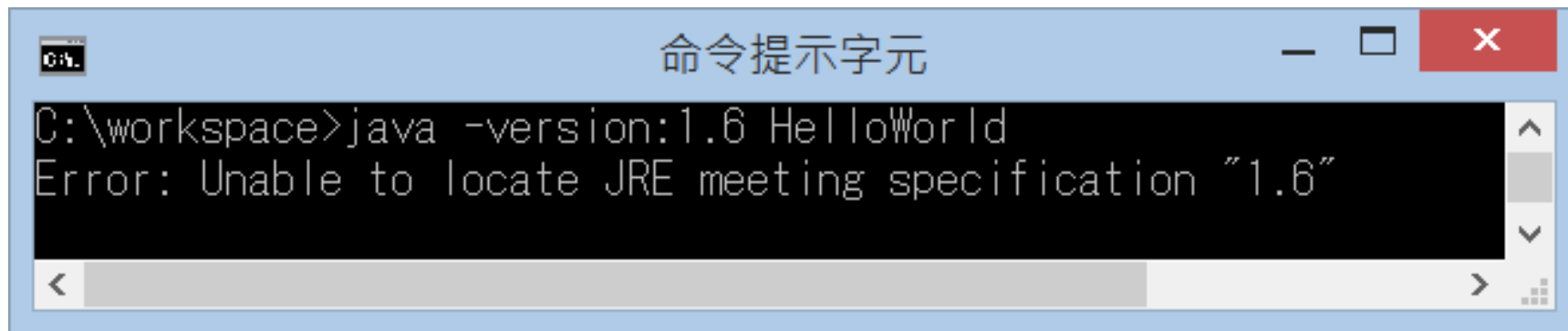
```
    at java.lang.ClassLoader.defineClass1(Native Method)  
    at java.lang.ClassLoader.defineClass(Unknown Source)  
    at java.security.SecureClassLoader.defineClass(Unknown Source)  
    at java.net.URLClassLoader.defineClass(Unknown Source)  
    at java.net.URLClassLoader.access$100(Unknown Source)  
    at java.net.URLClassLoader$1.run(Unknown Source)  
    at java.net.URLClassLoader$1.run(Unknown Source)  
    at java.security.AccessController.doPrivileged(Native Method)  
    at java.net.URLClassLoader.findClass(Unknown Source)  
    at java.lang.ClassLoader.loadClass(Unknown Source)  
    at sun.misc.Launcher$AppClassLoader.loadClass(Unknown Source)  
    at java.lang.ClassLoader.loadClass(Unknown Source)  
    at sun.launcher.LauncherHelper.checkAndLoadMain(Unknown Source)
```

```
C:\workspace>javac -bootclasspath "C:\Program Files\Java\jre7\lib\rt.jar" -source  
e 1.7 -target 1.7 HelloWorld.java
```

```
C:\workspace>java -version:1.7 HelloWorld  
Hello World
```


類別檔案版本

- 如果使用 `-version` 指定的版本，實際上無法在系統上找到已安裝的 JRE，則會出現以下錯誤：

A screenshot of a Windows Command Prompt window. The title bar is light blue and contains the text "命令提示字元" (Command Prompt) in the center, with standard window controls (minimize, maximize, close) on the right. The main area is black with white text. The first line shows the command `C:\workspace>java -version:1.6 HelloWorld`. The second line shows the error message `Error: Unable to locate JRE meeting specification "1.6"`. A horizontal scrollbar is visible at the bottom of the text area.

```
C:\workspace>java -version:1.6 HelloWorld
Error: Unable to locate JRE meeting specification "1.6"
```

類別檔案版本

- 那麼在 IDE 中如何設定 -source 與 -target 對應的選項呢？以 NetBeans 為例

...

