

# Java<sup>SE 7</sup> 技術手冊

- 涵蓋 OCP/JP (原 SCJP) 考試範圍
- Coin 專案、JSR166y、JDBC 4.1、NIO.2 等 Java SE 7 新功能介紹
- JDK 基礎與 IDE 操作交相對照
- 提供 Lab 檔案與操作錄影教學

# CHAPTER 2

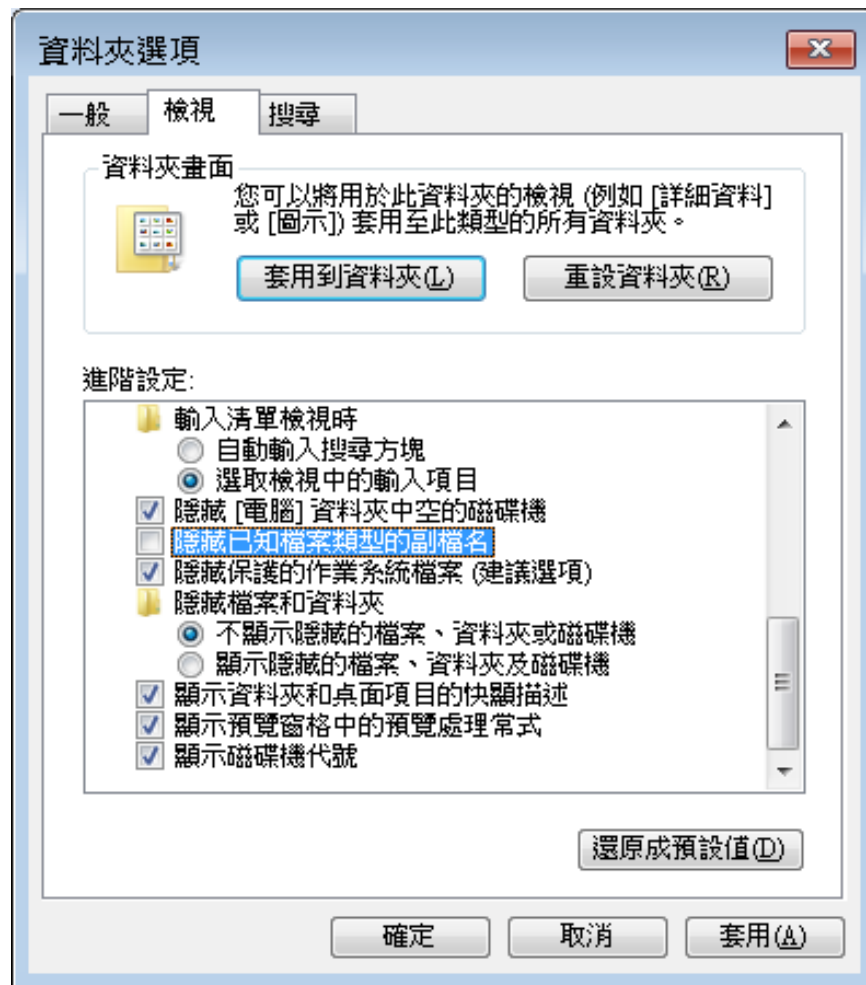
- 從JDK到IDE



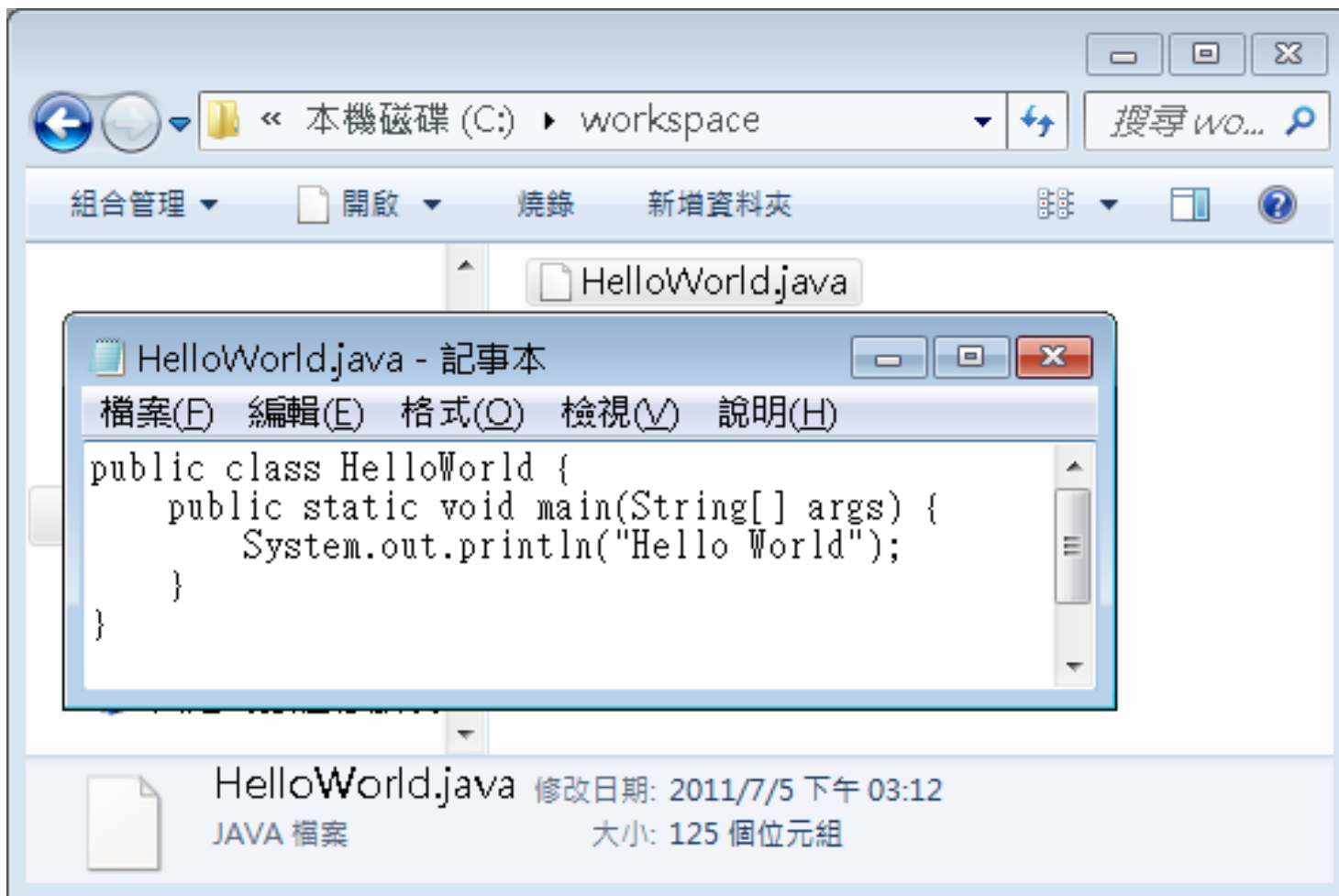
## 學習目標

- 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



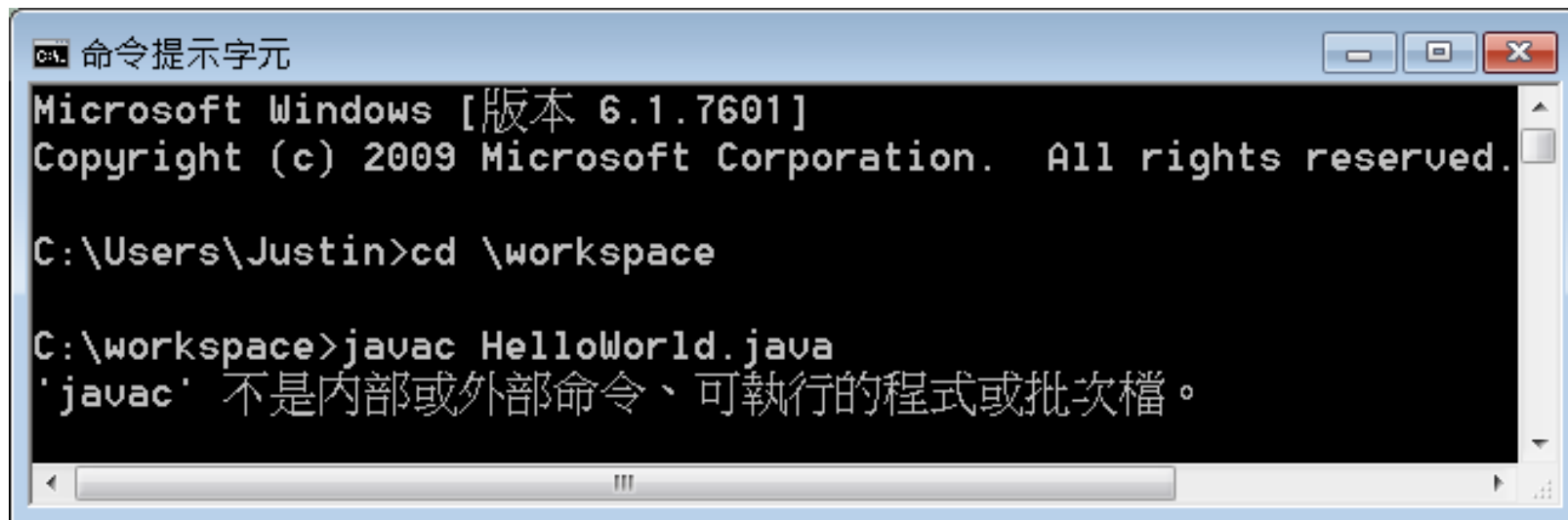
# 撰寫Java原始碼



# 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？

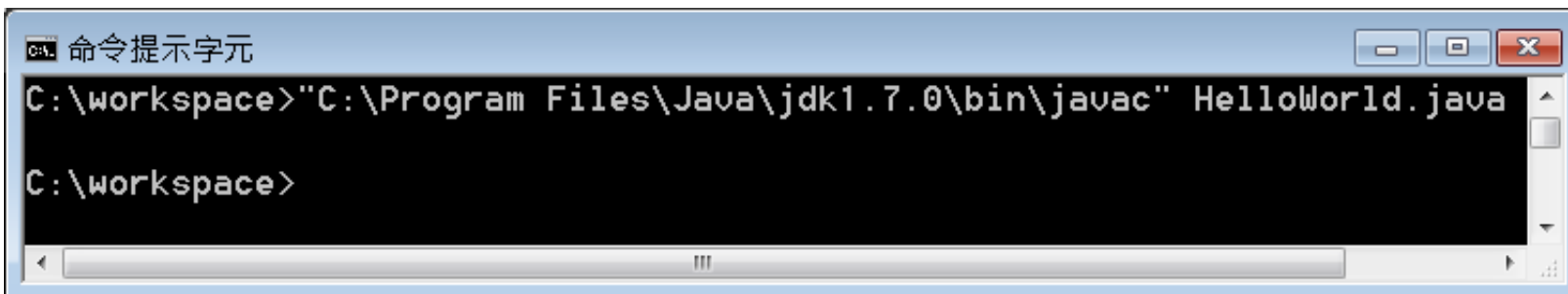


A screenshot of a Windows Command Prompt window titled "命令提示字元". The window shows the following text:

```
Microsoft Windows [版本 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\Justin>cd \workspace

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

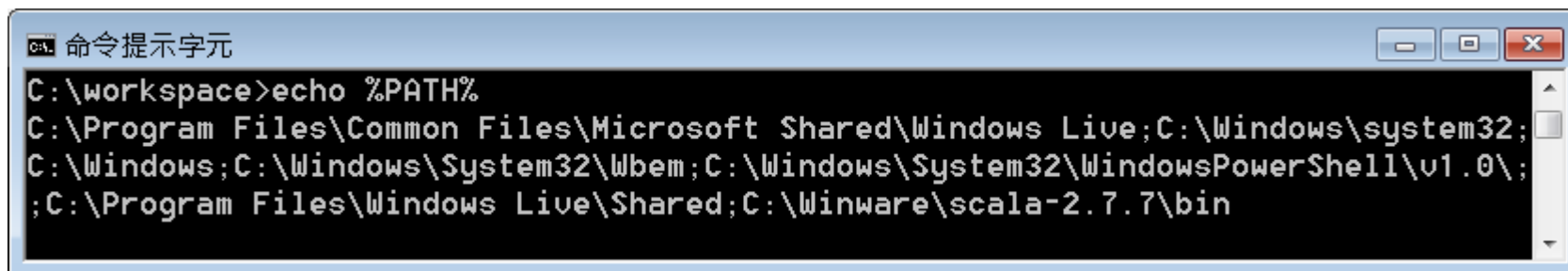


A screenshot of a Windows Command Prompt window titled "命令提示字元". The window shows the following text:

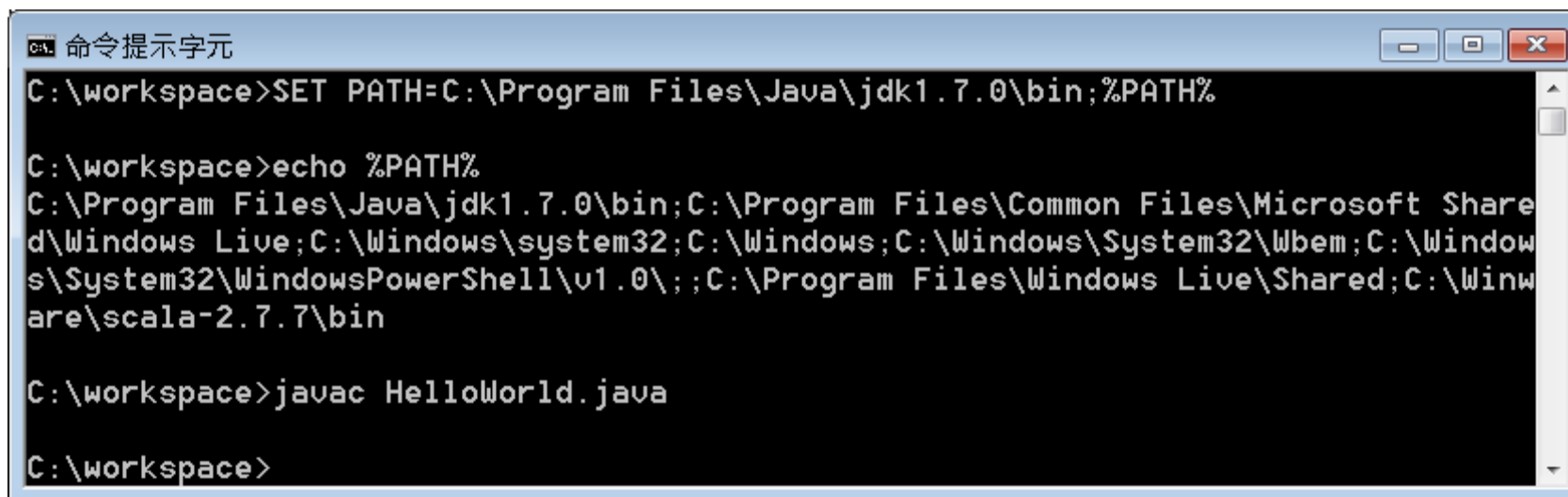
```
C:\workspace>"C:\Program Files\Java\jdk1.7.0\bin\javac" HelloWorld.java

C:\workspace>
```

# PATH是什麼？



```
命令提示字元
C:\workspace>echo %PATH%
C:\Program Files\Common Files\Microsoft Shared\Windows Live;C:\Windows\system32;
C:\Windows;C:\Windows\System32\Wbem;C:\Windows\System32\WindowsPowerShell\v1.0\;
;C:\Program Files\Windows Live\Shared;C:\Winware\scala-2.7.7\bin
```



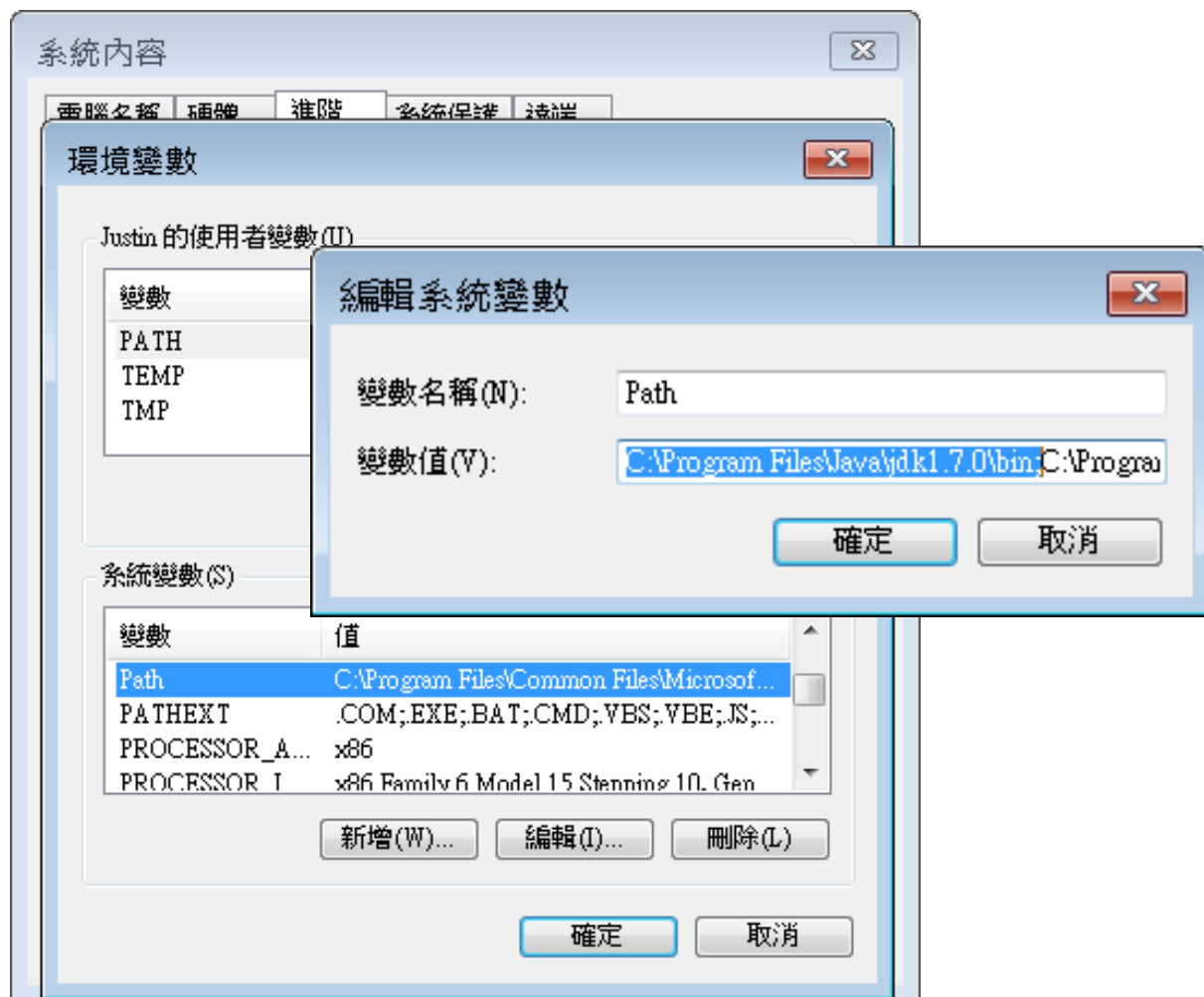
```
命令提示字元
C:\workspace>SET PATH=C:\Program Files\Java\jdk1.7.0\bin;%PATH%

C:\workspace>echo %PATH%
C:\Program Files\Java\jdk1.7.0\bin;C:\Program Files\Common Files\Microsoft Share
d\Windows Live;C:\Windows\system32;C:\Windows;C:\Windows\System32\Wbem;C:\Window
s\System32\WindowsPowerShell\v1.0\;;C:\Program Files\Windows Live\Shared;C:\Winw
are\scala-2.7.7\bin

C:\workspace>javac HelloWorld.java

C:\workspace>
```

# PATH是什麼？

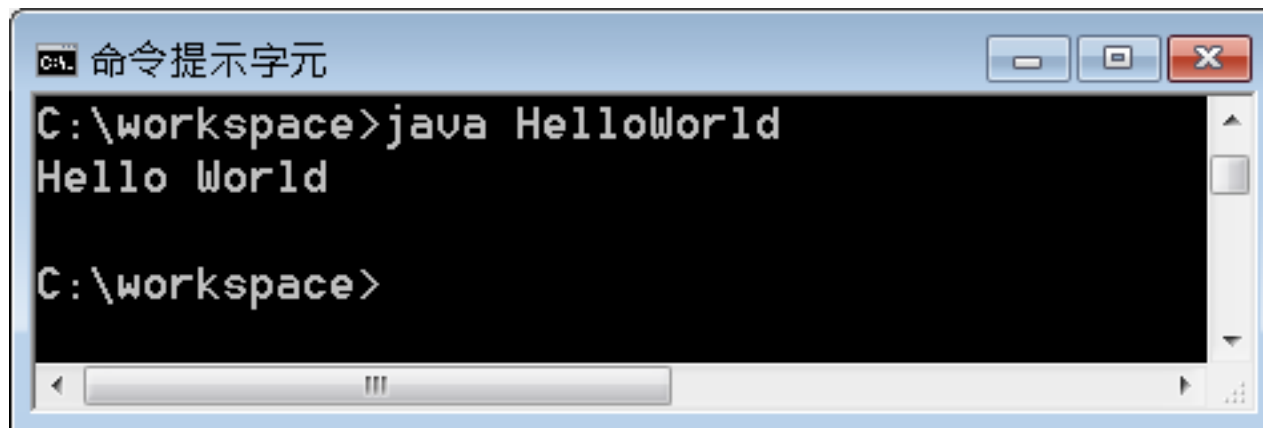




# PATH是什麼？

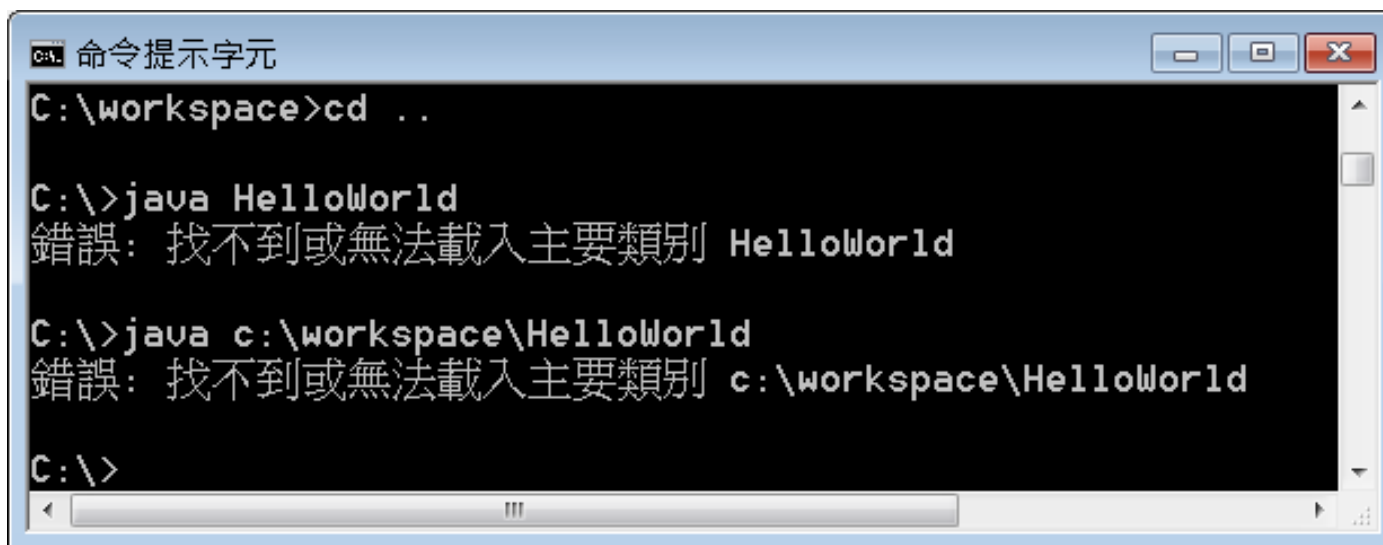
- 建議將JDK的bin路徑放在Path變數的最前方
  - 因為系統搜尋Path路徑時，會從最前方開始，如果路徑下找到指定的工具程式就會直接執行
- 若系統中安裝兩個以上JDK時，Path路徑中設定的順序，將決定執行哪個JDK下的工具程式
- 在安裝了多個JDK或JRE的電腦中，確定執行了哪個版本的JDK或JRE非常重要，確定PATH資訊是一定要作的動作

# 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

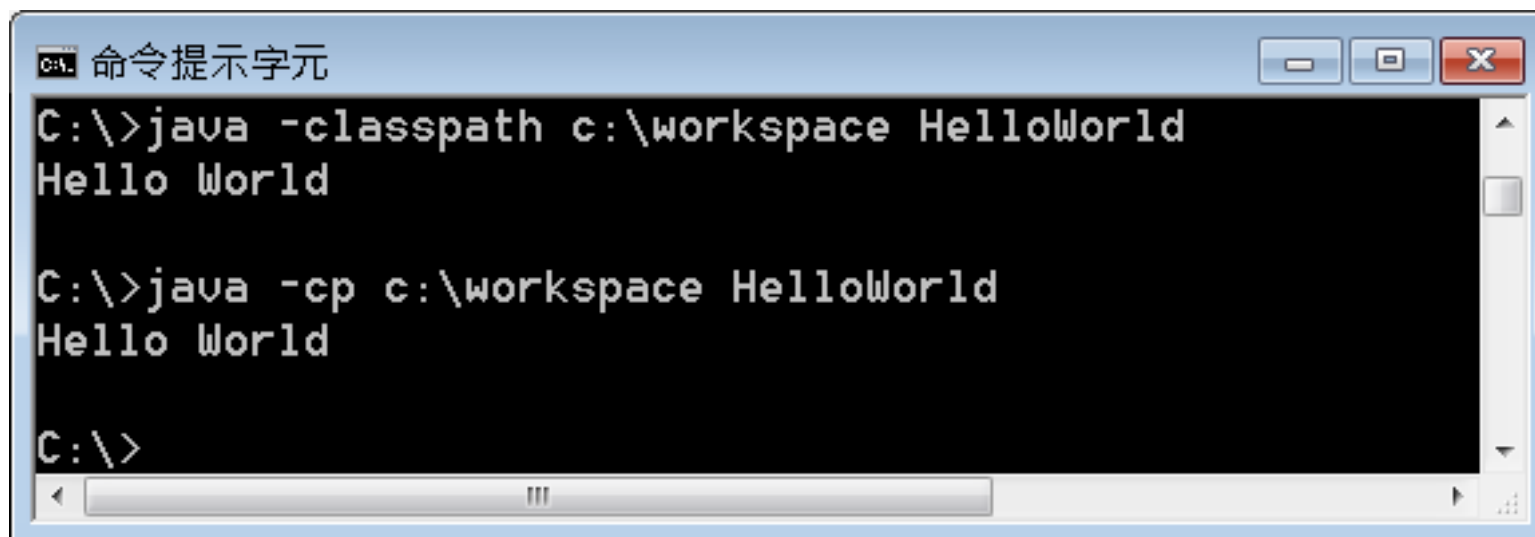
C:\>
```

# 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



```
命令提示字元
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



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

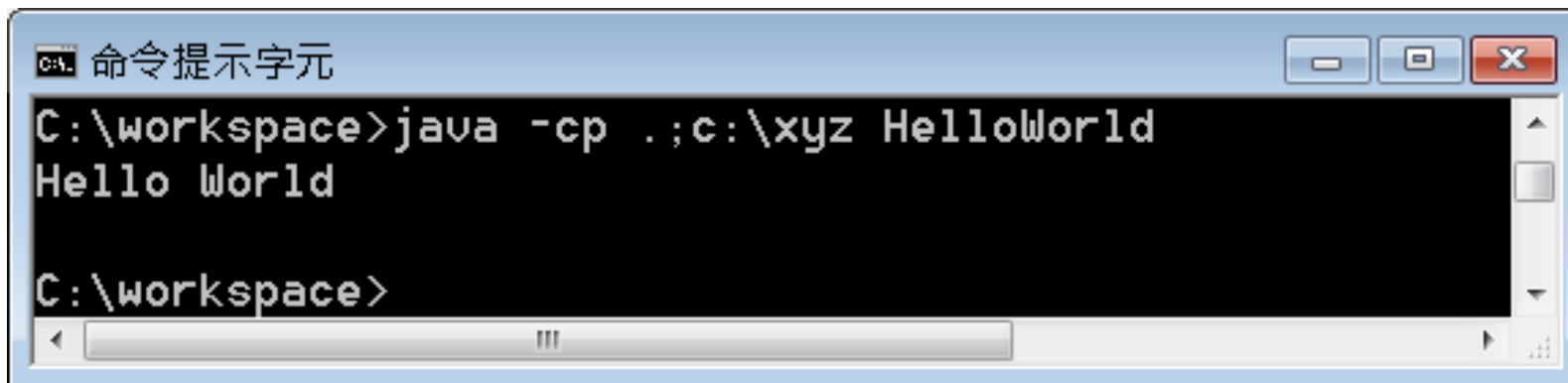
```
C:\>cd \workspace

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



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

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

The command executed is `java -cp .;c:\xyz HelloWorld`, which runs the `HelloWorld` class using the current directory and `c:\xyz` as the classpath. The output is `Hello World`.

# 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，執行時若要使用JAR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：

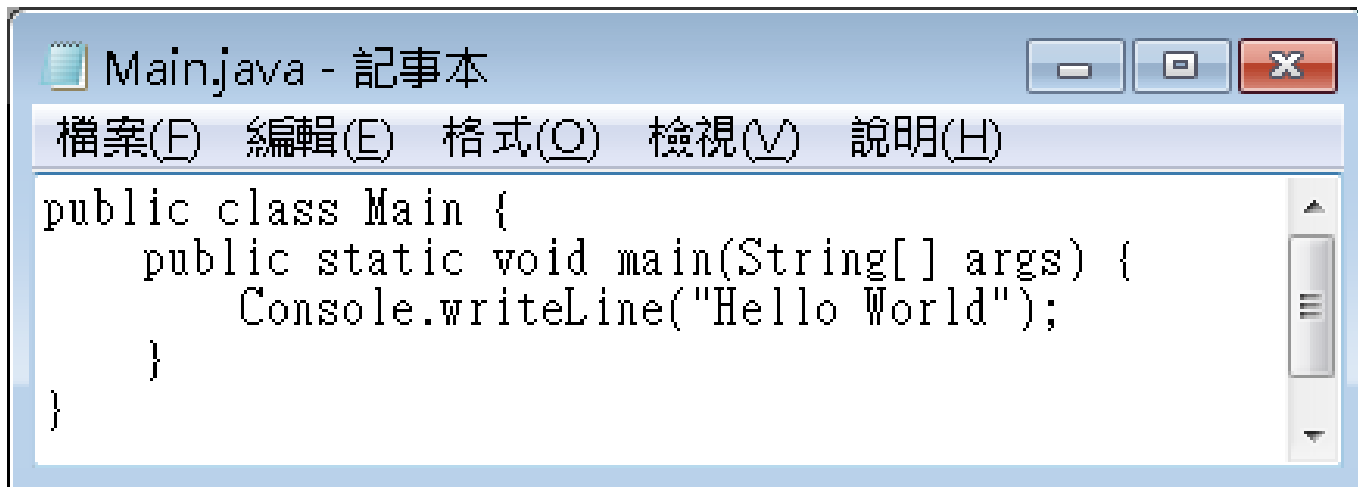
```
java -cp .;C:\jars\* cc.openhome.JNotePad
```

# 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

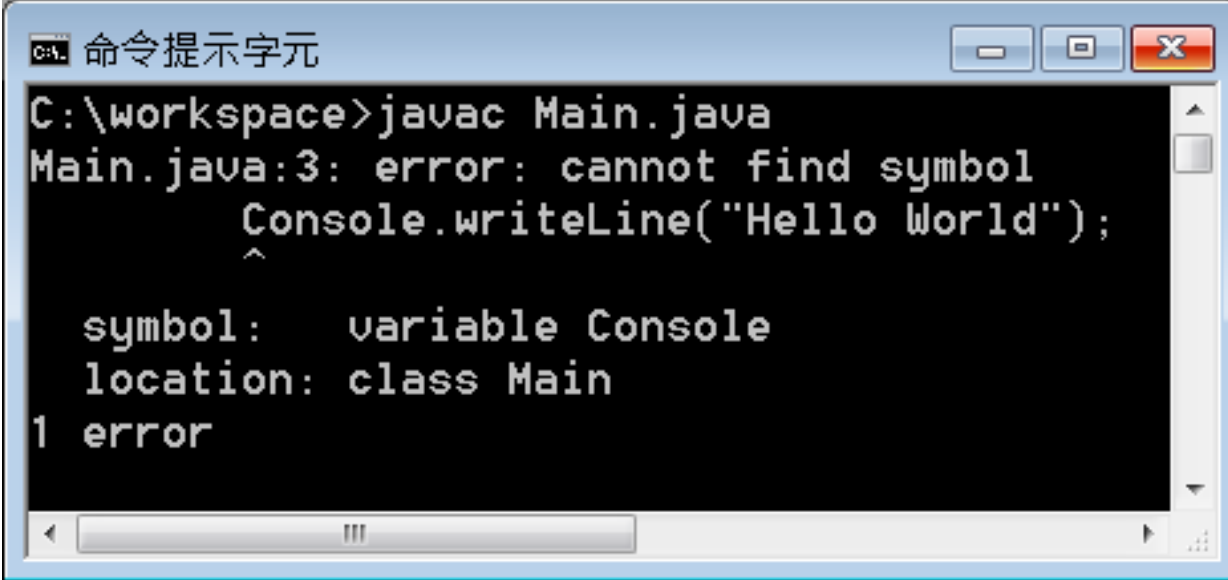
# 編譯器（ javac ）與CLASSPATH

- 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");  
    }  
}
```

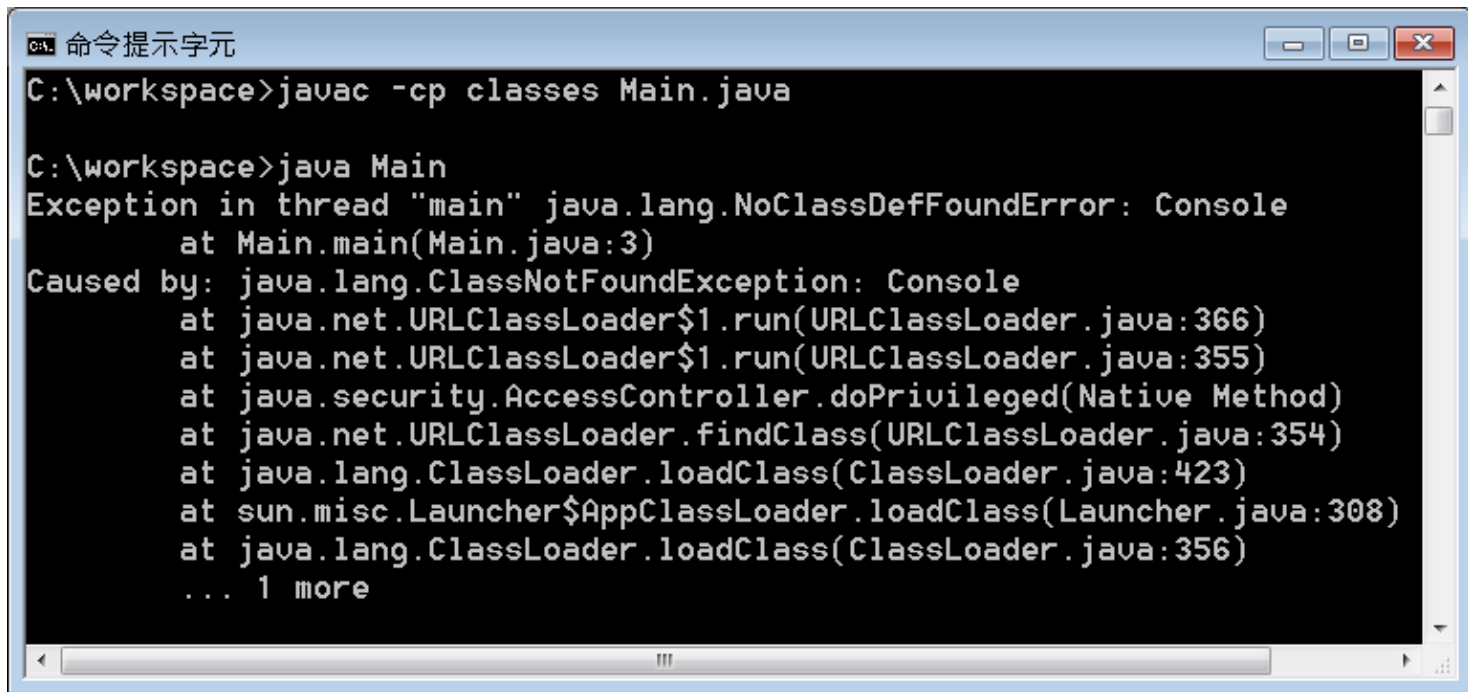
# 編譯器（javac）與CLASSPATH



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` and `location: class Main`. The window also shows `1 error` at the bottom.

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

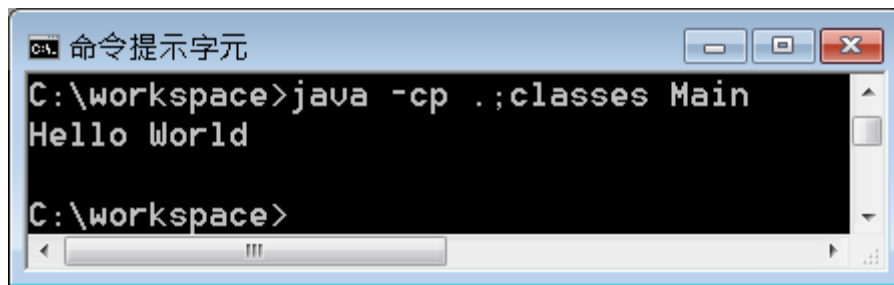
# 編譯器（javac）與CLASSPATH



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

```
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:366)
    at java.net.URLClassLoader$1.run(URLClassLoader.java:355)
    at java.security.AccessController.doPrivileged(Native Method)
    at java.net.URLClassLoader.findClass(URLClassLoader.java:354)
    at java.lang.ClassLoader.loadClass(ClassLoader.java:423)
    at sun.misc.Launcher$AppClassLoader.loadClass(Launcher.java:308)
    at java.lang.ClassLoader.loadClass(ClassLoader.java:356)
    ... 1 more
```



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

```
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

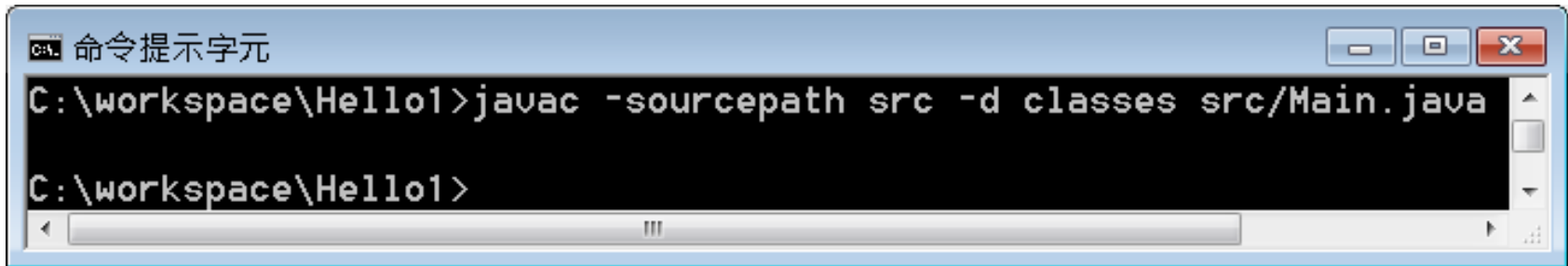
# 編譯器（ javac ）與SOURCEPATH

- 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

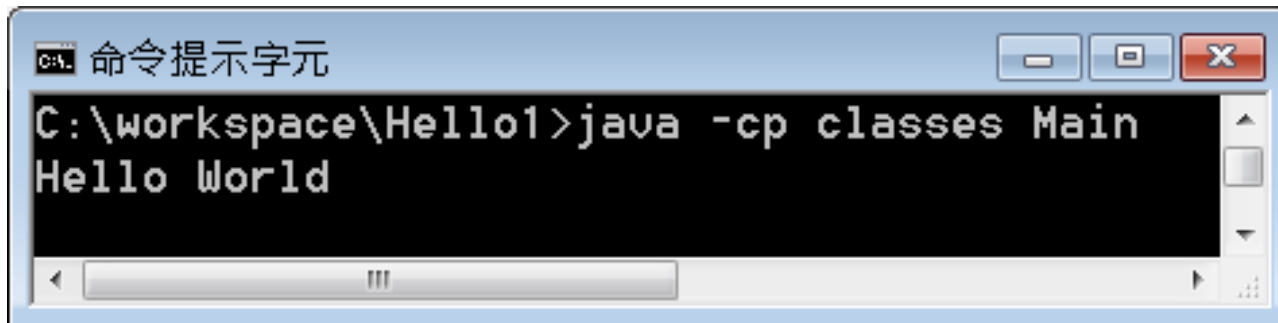
- src資料夾將用來放置原始碼檔案，而編譯好的位元碼檔案，希望能指定存放至classes資料夾



```
命令提示字元
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



```
命令提示字元
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 31ms] ①
[search path for source files: src] ②
[search path for class files: C:\Program Files\Java\jdk1.7.0\jre\lib\resources.jar,C:\Program Files\Java\jdk1.7.0\jre\lib\rt.jar,C:\Program Files\Java\jdk1.7.0\jre\lib\sunrsasign.jar,C:\Program Files\Java\jdk1.7.0\jre\lib\jsse.jar,C:\Program Files\Java\jdk1.7.0\jre\lib\jce.jar,C:\Program Files\Java\jdk1.7.0\jre\lib\charsets.jar,C:\Program Files\Java\jdk1.7.0\jre\classes,C:\Program Files\Java\jdk1.7.0\jre\lib\ext\dnsns.jar,C:\Program Files\Java\jdk1.7.0\jre\lib\ext\localedata.jar,C:\Program Files\Java\jdk1.7.0\jre\lib\ext\sunec.jar,C:\Program Files\Java\jdk1.7.0\jre\lib\ext\sunjc_provider.jar,C:\Program Files\Java\jdk1.7.0\jre\lib\ext\sunmscapi.jar,C:\Program Files\Java\jdk1.7.0\jre\lib\ext\sunpkcs11.jar,C:\Program Files\Java\jdk1.7.0\jre\lib\ext\zipfs.jar,...]
[loading ZipFileIndexFileObject[C:\Program Files\Java\jdk1.7.0\lib\ct.sym(META-INF/sym/rt.jar/java/lang/Object.class)]]
[loading ZipFileIndexFileObject[C:\Program Files\Java\jdk1.7.0\lib\ct.sym(META-INF/sym/rt.jar/java/lang/String.class)]]
[checking Main]
[loading ZipFileIndexFileObject[C:\Program Files\Java\jdk1.7.0\lib\ct.sym(META-INF/sym/rt.jar/java/lang/AutoCloseable.class)]]
[loading RegularFileObject[src\Console.java]]
[parsing started RegularFileObject[src\Console.java]]
[parsing completed 0ms]
[wrote RegularFileObject[classes/Main.class]] ③
[checking Console]
[loading ZipFileIndexFileObject[C:\Program Files\Java\jdk1.7.0\lib\ct.sym(META-INF/sym/rt.jar/java/lang/System.class)]]
[loading ZipFileIndexFileObject[C:\Program Files\Java\jdk1.7.0\lib\ct.sym(META-INF/sym/rt.jar/java/io/PrintStream.class)]]
[loading ZipFileIndexFileObject[C:\Program Files\Java\jdk1.7.0\lib\ct.sym(META-INF/sym/rt.jar/java/io/FilterOutputStream.class)]]
[loading ZipFileIndexFileObject[C:\Program Files\Java\jdk1.7.0\lib\ct.sym(META-INF/sym/rt.jar/java/io/OutputStream.class)]]
[wrote RegularFileObject[classes\Console.class]] ④
[total 265ms]
```

# 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

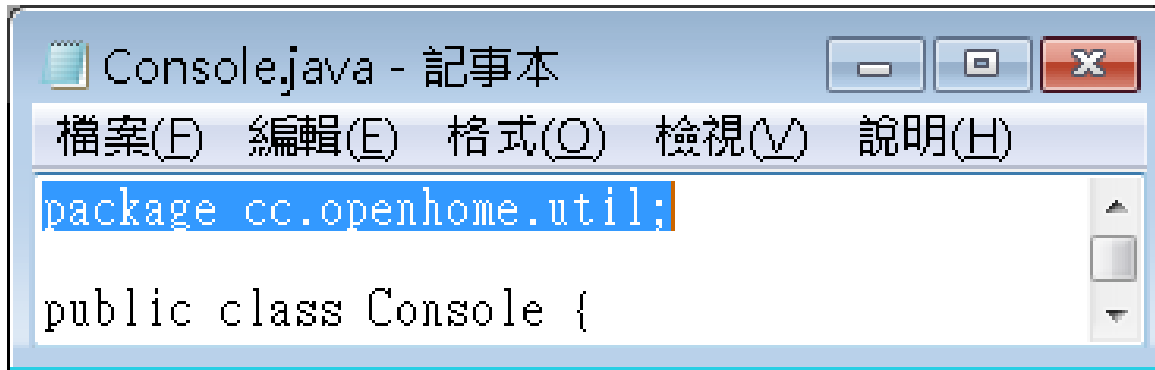
```
C:\workspace\Hello1>javac -verbose -sourcepath src -cp classes -d classes src/Main.java
[parsing started RegularFileObject[src\Main.java]]
[parsing completed 15ms]
[search path for source files: src]
[search path for class files: C:\Program Files\Java\jdk1.7.0\jre\lib\resources.jar,C:\Program Files\Java\jdk1.7.0\jre\lib\rt.jar,C:\Program Files\Java\jdk1.7.0\jre\lib\sunrsasign.jar,C:\Program Files\Java\jdk1.7.0\jre\lib\jsse.jar,C:\Program Files\Java\jdk1.7.0\jre\lib\jce.jar,C:\Program Files\Java\jdk1.7.0\jre\lib\charsets.jar,C:\Program Files\Java\jdk1.7.0\jre\classes,C:\Program Files\Java\jdk1.7.0\jre\lib\ext\dnsns.jar,C:\Program Files\Java\jdk1.7.0\jre\lib\ext\localedata.jar,C:\Program Files\Java\jdk1.7.0\jre\lib\ext\sunec.jar,C:\Program Files\Java\jdk1.7.0\jre\lib\ext\sunec_provider.jar,C:\Program Files\Java\jdk1.7.0\jre\lib\ext\sunmscapi.jar,C:\Program Files\Java\jdk1.7.0\jre\lib\ext\sunpkcs11.jar,C:\Program Files\Java\jdk1.7.0\jre\lib\ext\zipfs.jar,classes]
[loading ZipFileIndexFileObject[C:\Program Files\Java\jdk1.7.0\lib\ct.sym(META-INF/sym/rt.jar/java/lang/Object.class)]]
[loading ZipFileIndexFileObject[C:\Program Files\Java\jdk1.7.0\lib\ct.sym(META-INF/sym/rt.jar/java/lang/String.class)]]
[checking Main]
[loading ZipFileIndexFileObject[C:\Program Files\Java\jdk1.7.0\lib\ct.sym(META-INF/sym/rt.jar/java/lang/AutoCloseable.class)]]
[loading RegularFileObject[classes\Console.class]]
[wrote RegularFileObject[classes>Main.class]]
[total 188ms]
```

# 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;

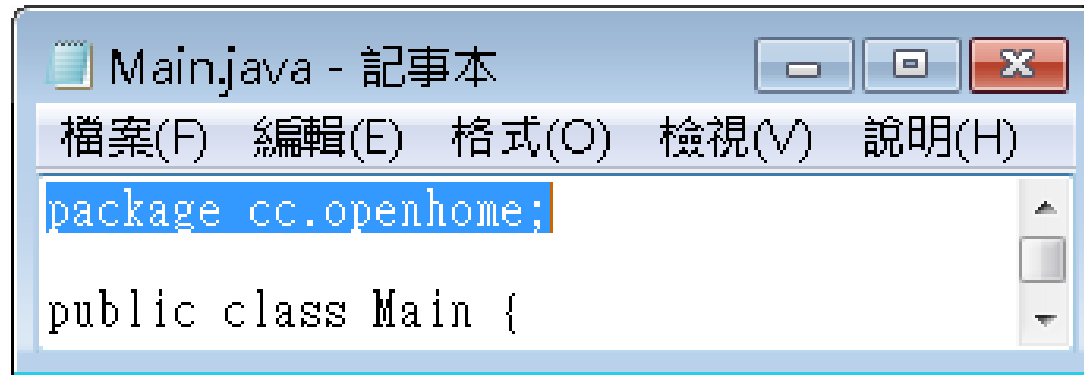
public class Console {
```

- 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Console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

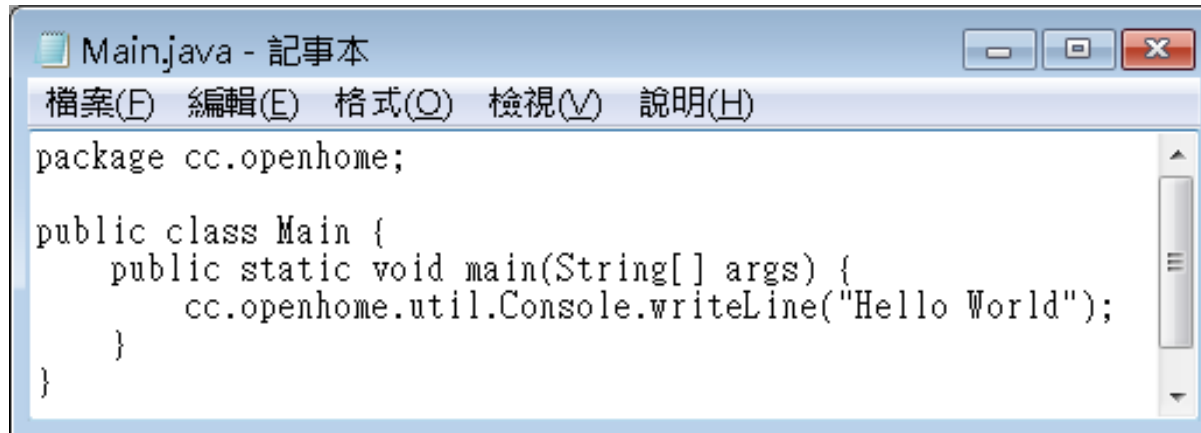
- 由於Console類別使用package定義在cc.openhome.util套件下，所以Console.java必須放在src資料夾中的cc/openhome/util資料夾
- Main類別使用package定義在cc.openhome套件下，所以Main.java必須放在src資料夾中的cc/openhome資料夾

## 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;

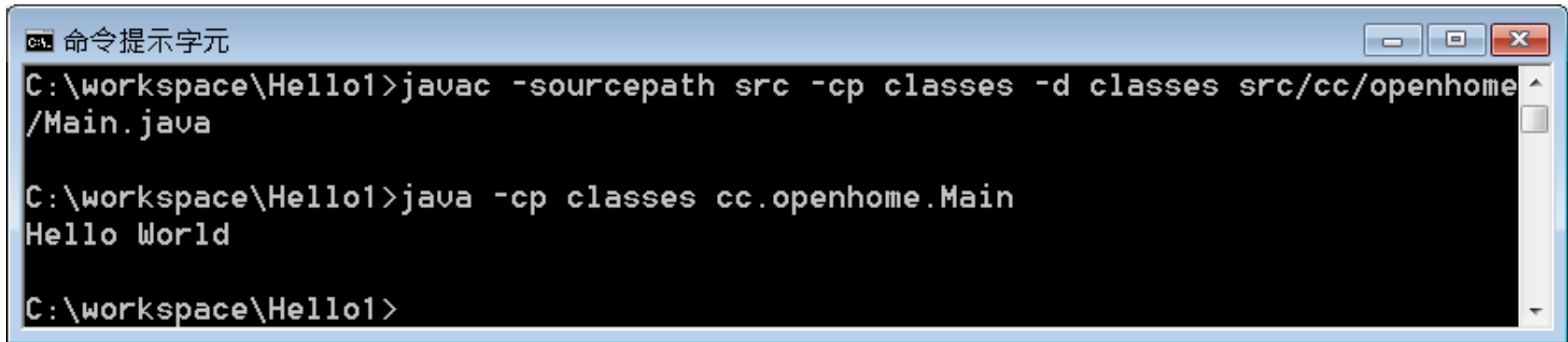
public class Main {
    public static void main(String[] args) {
        cc.openhome.util.Console.WriteLine("Hello World");
    }
}
```

# 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

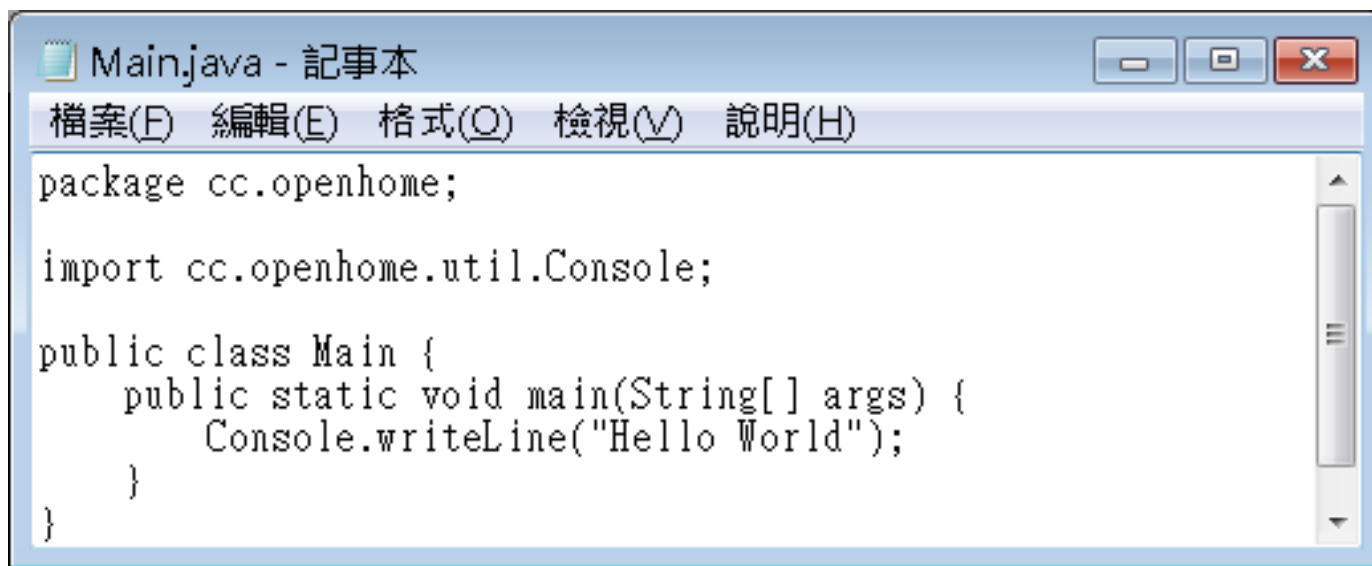
- 在編譯時若有使用-d指定位元碼的存放位置，就會自動建立出對應套件階層的資料夾，並將編譯出來的位元碼檔案放置至應有的位置



```
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：

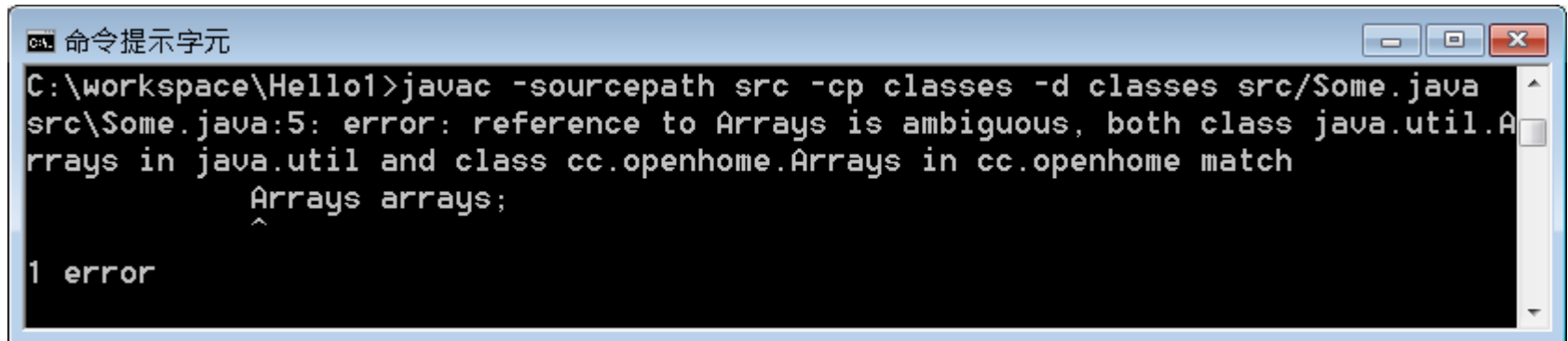
```
package cc.openhome;  
public class Arrays {  
    ...  
}
```

- 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?



```
命令提示字元
C:\workspace\Hello1>javac -sourcepath src -cp classes -d classes src/Some.java
src\Some.java:5: error: reference to Arrays is ambiguous, both class java.util.A
rrays in java.util and class cc.openhome.Arrays in cc.openhome match
    Arrays arrays;
    ^
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;
        ...
    }
}
```

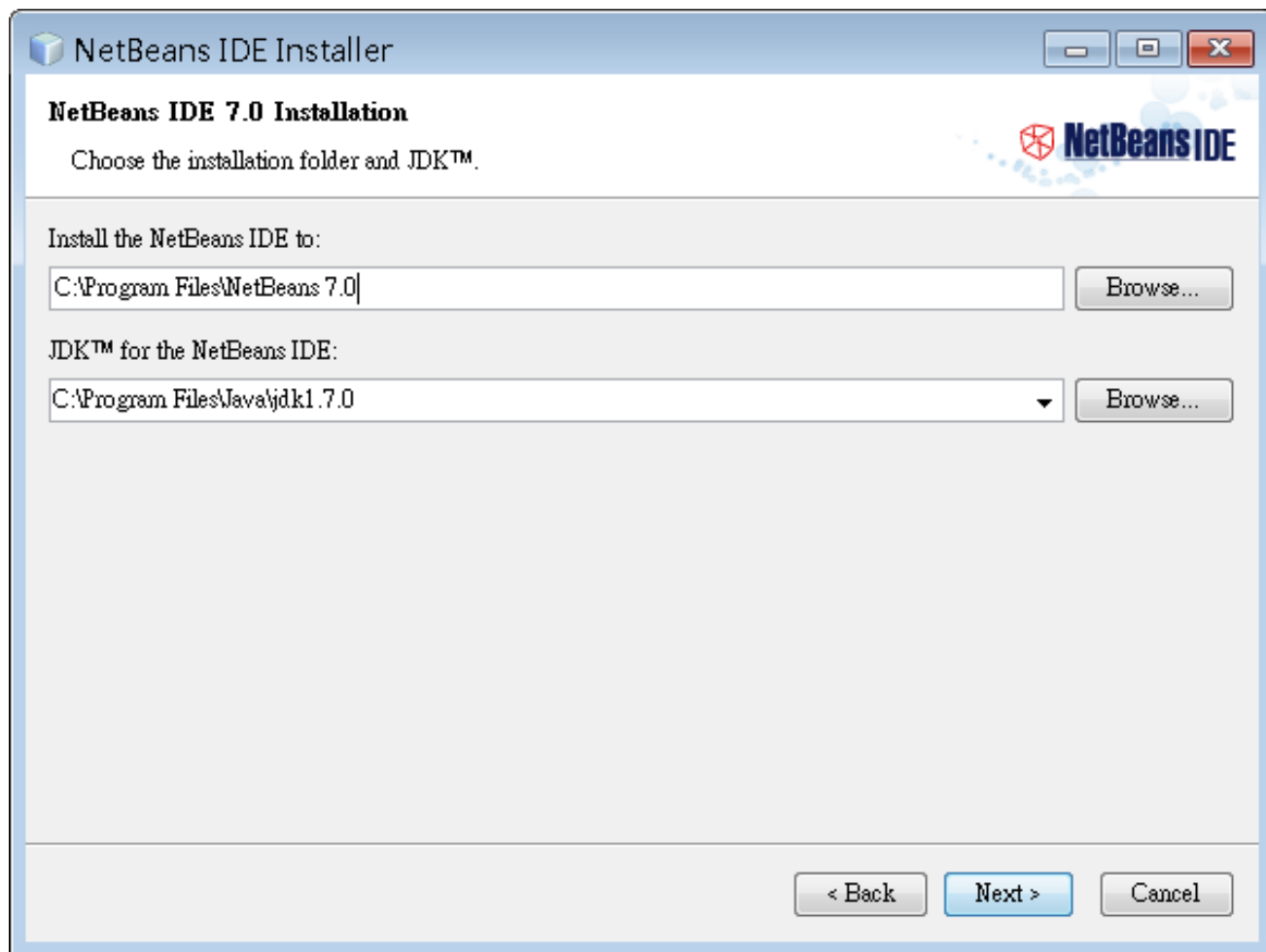
# 使用import偷懶

- 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

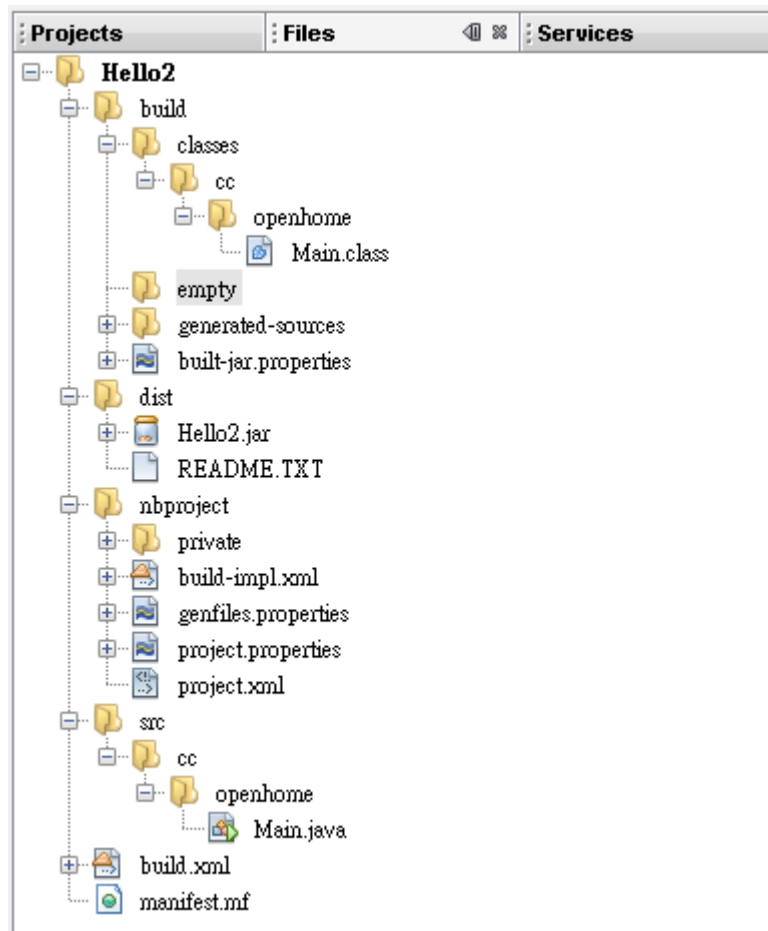
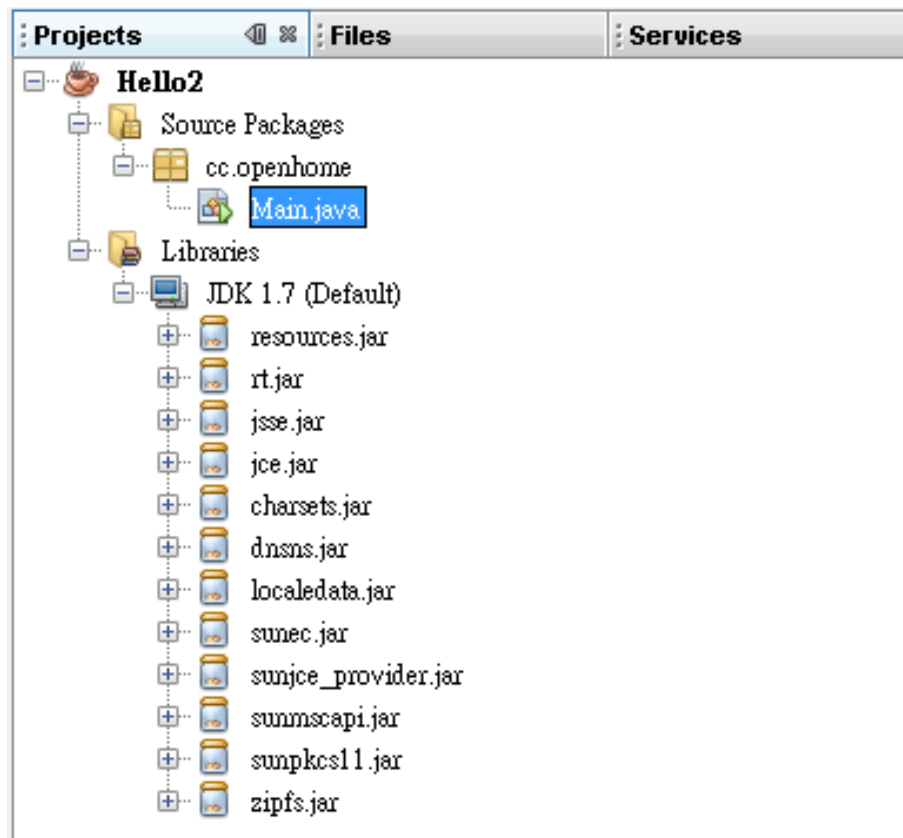
- 當編譯器看到一個沒有套件管理的類別名稱，會先在同一套件中尋找類別，如果找到就使用，若沒找到，再試著從import陳述進行比對
- `java.lang`可視為預設就有import，沒有寫任何import陳述時，也會試著比對`java.lang`組合，看看是否能找到對應類別

# IDE專案管理基礎

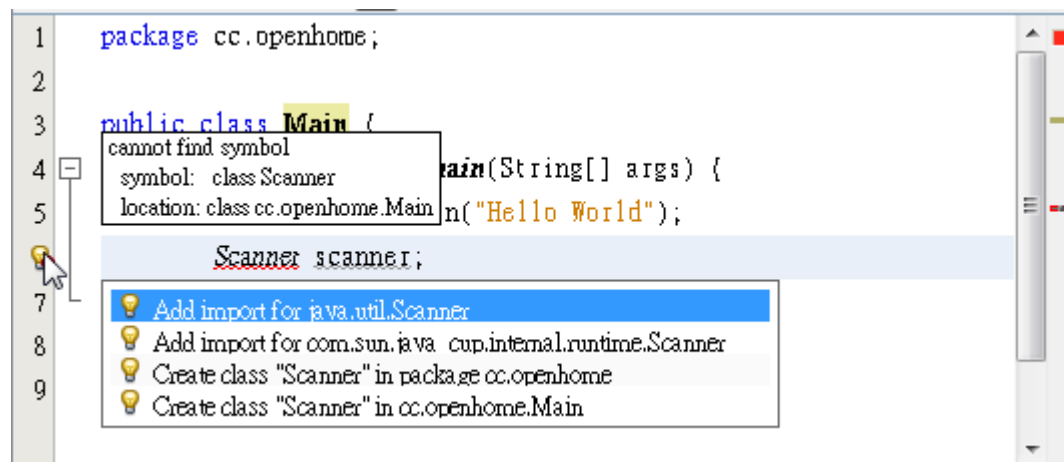
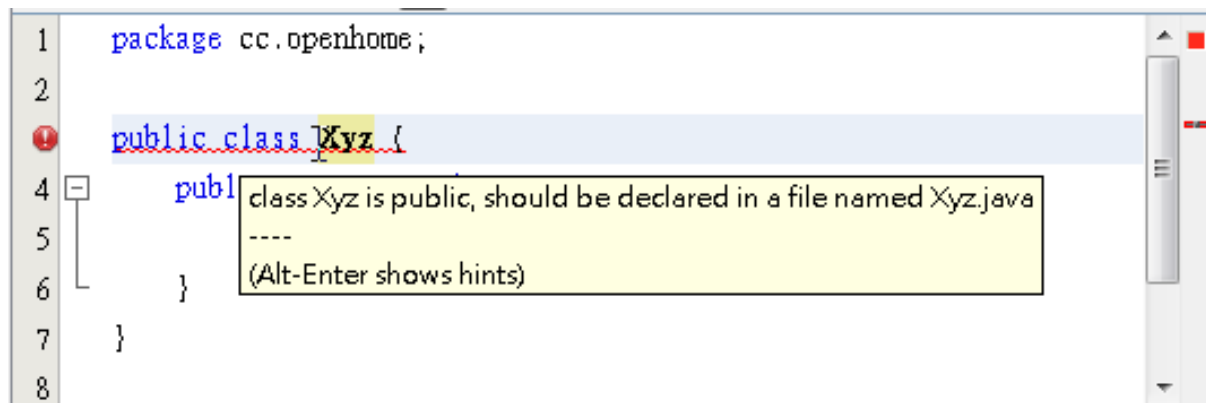




# IDE專案管理基礎



# 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，可以附帶一個-version引數，這可以顯示執行的JRE版本

## 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，有可能會發生以下的錯誤訊息...

命令提示字元

```
C:\workspace>javac -version
javac 1.7.0
```

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

```
C:\workspace>set PATH=C:\Program Files\Java\jdk1.6.0\bin;%PATH%
```

```
C:\workspace>java -version
java version "1.6.0_26"
Java(TM) SE Runtime Environment (build 1.6.0_26-b03)
Java HotSpot(TM) Client VM (build 20.1-b02, mixed mode, sharing)
```

```
C:\workspace>java HelloWorld
Exception in thread "main" java.lang.UnsupportedClassVersionError: HelloWorld :
Unsupported major.minor version 51.0
    at java.lang.ClassLoader.defineClass1(Native Method)
    at java.lang.ClassLoader.defineClassCond(ClassLoader.java:631)
    at java.lang.ClassLoader.defineClass(ClassLoader.java:615)
    at java.security.SecureClassLoader.defineClass(SecureClassLoader.java:14
1)
    at java.net.URLClassLoader.defineClass(URLClassLoader.java:283)
    at java.net.URLClassLoader.access$000(URLClassLoader.java:58)
    at java.net.URLClassLoader$1.run(URLClassLoader.java:197)
    at java.security.AccessController.doPrivileged(Native Method)
    at java.net.URLClassLoader.findClass(URLClassLoader.java:190)
    at java.lang.ClassLoader.loadClass(ClassLoader.java:306)
    at sun.misc.Launcher$AppClassLoader.loadClass(Launcher.java:301)
    at java.lang.ClassLoader.loadClass(ClassLoader.java:247)
Could not find the main class: HelloWorld.  Program will exit.
```

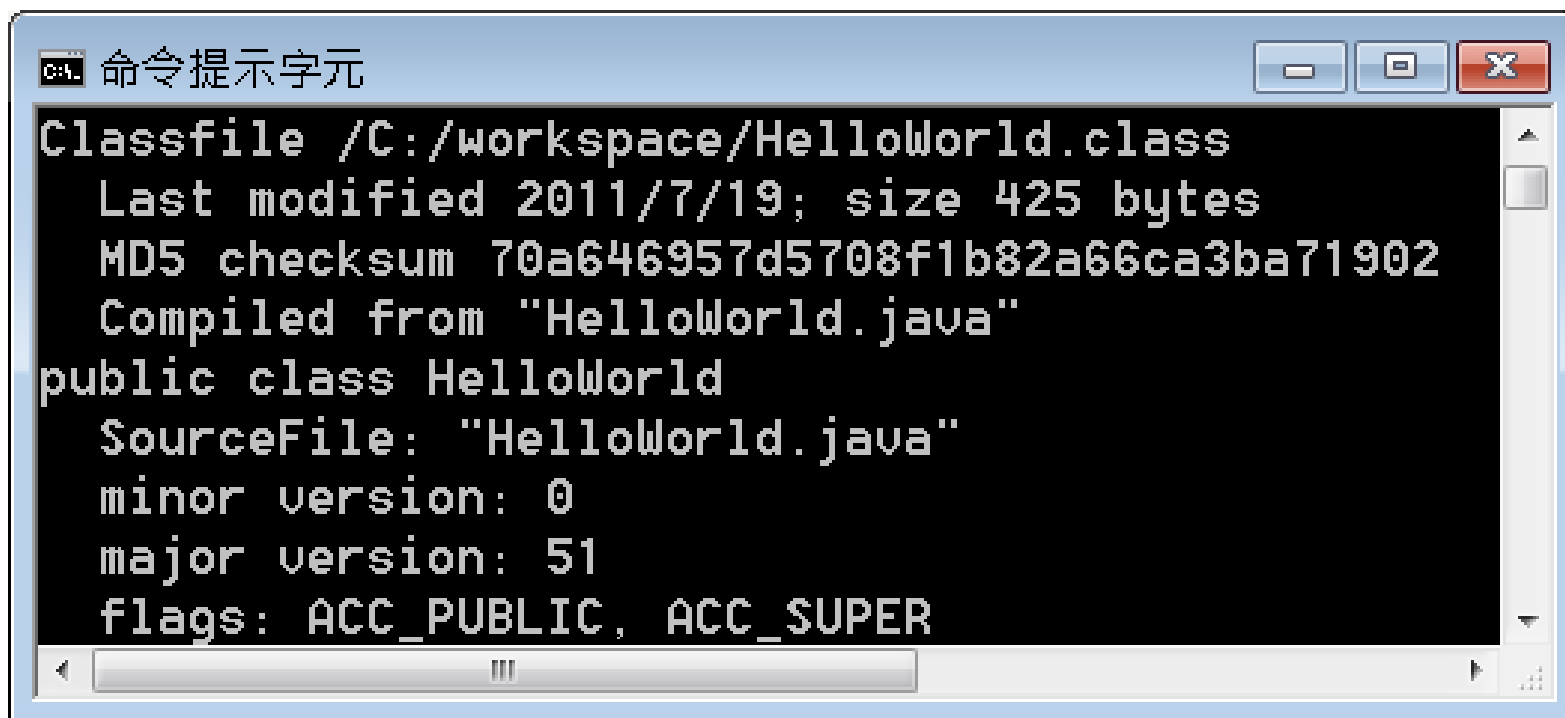
# 類別檔案版本

- 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：



```
命令提示字元
Classfile /C:/workspace/HelloWorld.class
  Last modified 2011/7/19; size 425 bytes
  MD5 checksum 70a646957d5708f1b82a66ca3ba71902
  Compiled from "HelloWorld.java"
public class HelloWorld
  SourceFile: "HelloWorld.java"
  minor version: 0
  major version: 51
  flags: ACC_PUBLIC, ACC_SUPER
```

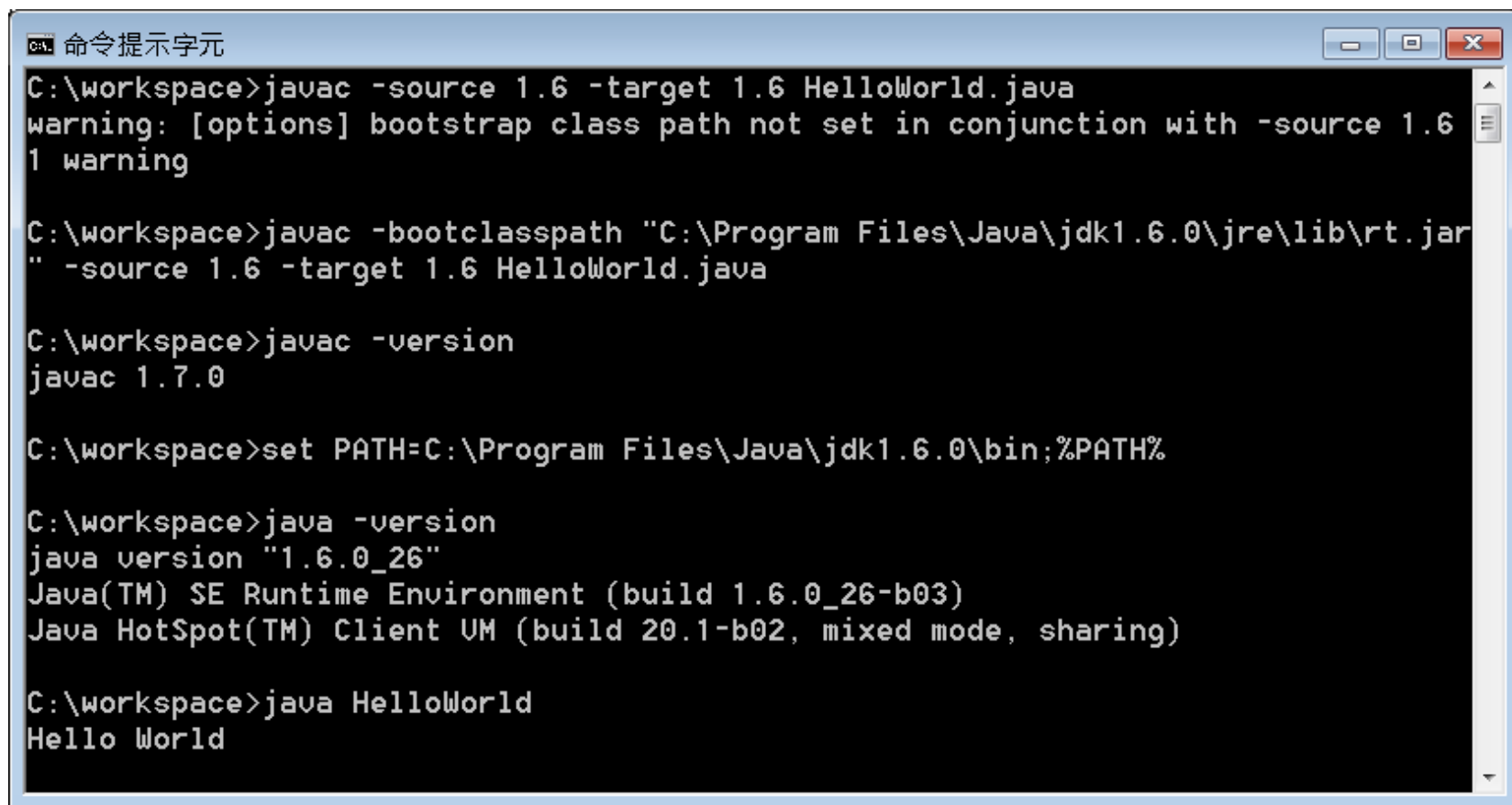
## 類別檔案版本

- `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



```
命令提示字元
C:\workspace>javac -source 1.6 -target 1.6 HelloWorld.java
warning: [options] bootstrap class path not set in conjunction with -source 1.6
1 warning

C:\workspace>javac -bootclasspath "C:\Program Files\Java\jdk1.6.0\jre\lib\rt.jar" -source 1.6 -target 1.6 HelloWorld.java

C:\workspace>javac -version
javac 1.7.0

C:\workspace>set PATH=C:\Program Files\Java\jdk1.6.0\bin;%PATH%

C:\workspace>java -version
java version "1.6.0_26"
Java(TM) SE Runtime Environment (build 1.6.0_26-b03)
Java HotSpot(TM) Client VM (build 20.1-b02, mixed mode, sharing)

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
- 例如**JDK7**預設的-target與-source都是**1.7**
- -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1.7的rt.jar（也就是Java SE 7 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...

命令提示字元

```
C:\workspace>javac -version
javac 1.7.0
```

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

```
C:\workspace>java -version:1.6 HelloWorld
```

```
Exception in thread "main" java.lang.UnsupportedClassVersionError: HelloWorld :
Unsupported major.minor version 51.0
```

```
    at java.lang.ClassLoader.defineClass1(Native Method)
    at java.lang.ClassLoader.defineClassCond(Unknown Source)
    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$000(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)
```

```
Could not find the main class: HelloWorld.  Program will exit.
```

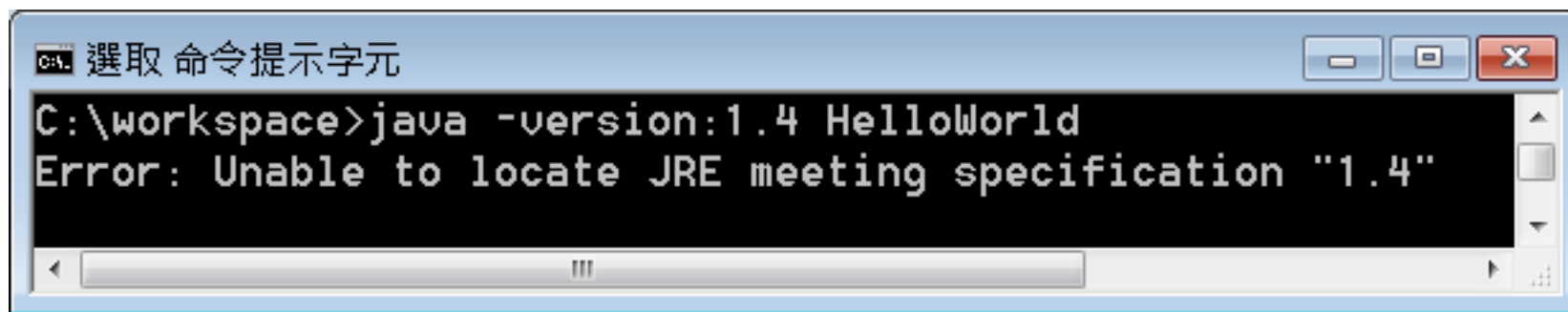
```
C:\workspace>javac -bootclasspath "C:\Program Files\Java\jdk1.6.0\jre\lib\rt.jar"
    -source 1.6 -target 1.6 HelloWorld.java
```

```
C:\workspace>java -version:1.6 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 '選取 命令提示字元' (Select Command Prompt) on the left and standard window controls (minimize, maximize, close) on the right. The command prompt area has a black background with white text. The first line shows the command 'C:\workspace>java -version:1.4 HelloWorld'. The second line shows the error message 'Error: Unable to locate JRE meeting specification "1.4"'. A horizontal scrollbar is visible at the bottom of the command prompt area.

```
C:\workspace>java -version:1.4 HelloWorld
Error: Unable to locate JRE meeting specification "1.4"
```

# 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...

