Introduction to **Gradle**

N.L. Hsueh

Feng Chia University

Gradle 介紹

- Gradle 是一個建構自動化工具
  - 協助自動化完成 **編譯、測試、檢查程式碼、打包 等工作**
- 使用 Groovy DSL(領域特定語言) 作為設定檔語言
  - 比起 XML, 縮減自 Groovy 語言的 Groovy DSL 有更高的可讀性, 並且可以執行更複雜的工作。
- 需要先安裝 Java
- 類似的工具, 如:ant、maven
  - 使用 XML 語言
- 許多開源軟體或企業使用中
  - ex: Netflix、Google 都將 Gradle 作為使用 Java 語言時的官方建置工具

領域特定語言:
「針對特定應用領域的程式語言」
可以全新設計, 也可以是現有程式語言"瘦身"之後的不同版本。

Gradle 和 Maven 的差異

	設定檔檔名	設定檔語法	設定檔寫法	建構速度	靈活性
gradle	build.gradle	Groovy DSL	較精簡、易懂	較快	自定義的彈性更大且便利
maven	pom.xml	XML	較繁瑣	較慢	自定義較複雜

Groovy DSL 範例

build.gradle

```
plugins {
    id 'java'
    id 'checkstyle' version '2.12.1'
}

version ='1.0'

repositories {
    mavenCentral()
}

dependencies {
    testImplementation 'junit:junit:4.13.2'

    implementation 'com.google.guava:guava:31.0.1-jre'
}
```

XML 範例

pom.xml

```xml
    <version>1.0</version>
    <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-checkstyle-plugin</artifactId>
        <version>2.12.1</version>
        <executions>
            <execution>
                <configuration>
                    <configLocation>config/checkstyle/checkstyle.xml</configLocation>
                    <consoleOutput>true</consoleOutput>
                    <failsOnError>true</failsOnError>
                </configuration>
                <goals>
                    <goal>check</goal>
                </goals>
            </execution>
        </executions>
        </plugin>
    </plugin>
    <dependencies>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.13.2</version>
        </dependency>
        <dependency>
            <groupId>com.google.guava</groupId>
            <artifactId>guava</artifactId>
            <version>31.1-jre</version>
        </dependency>
    </dependencies>
    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>2.3.2</version>
            </plugin>
        </plugins>
    </build>
```

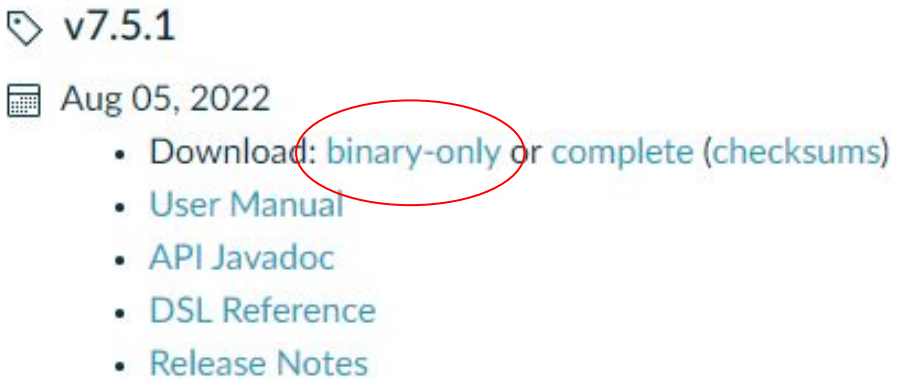
安裝 Gradle

安裝 Gradle (1/4)

- 確認 Java 環境版本為 1.8 以上

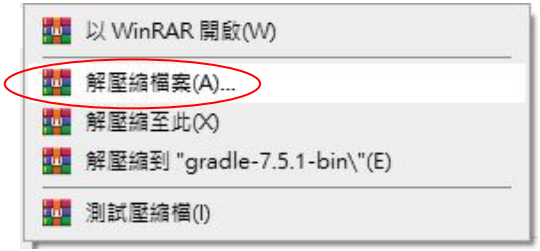
```
$ java -version
java version "1.8.0_121"
```

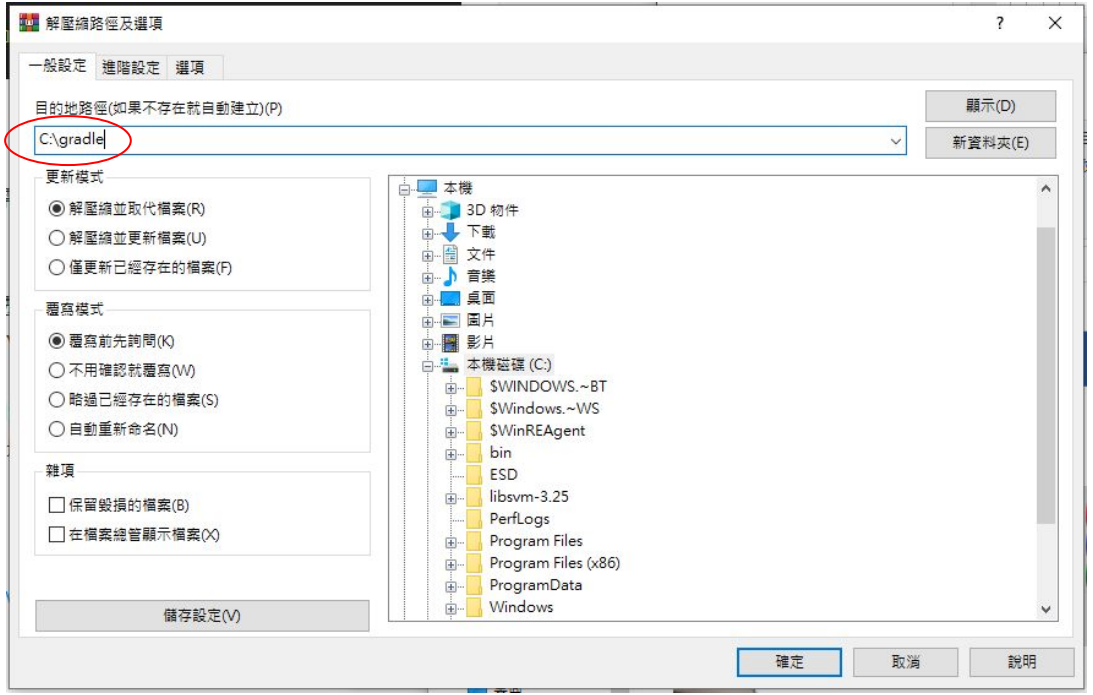
- 下載 Gradle (7.5.1 版本) 的二進位檔
  - https://gradle.org/releases/



安裝 Gradle (2/4)

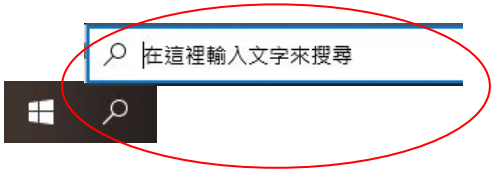
- 解壓縮 zip 檔案 至 **C:\gradle**





安裝 Gradle (3/4)

- 設定環境變數
  - 開啟左下角的系統內建搜尋

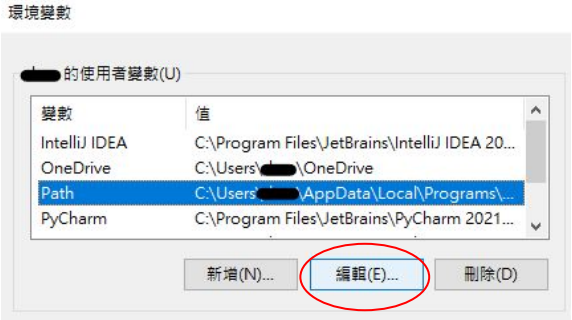


  - 搜尋

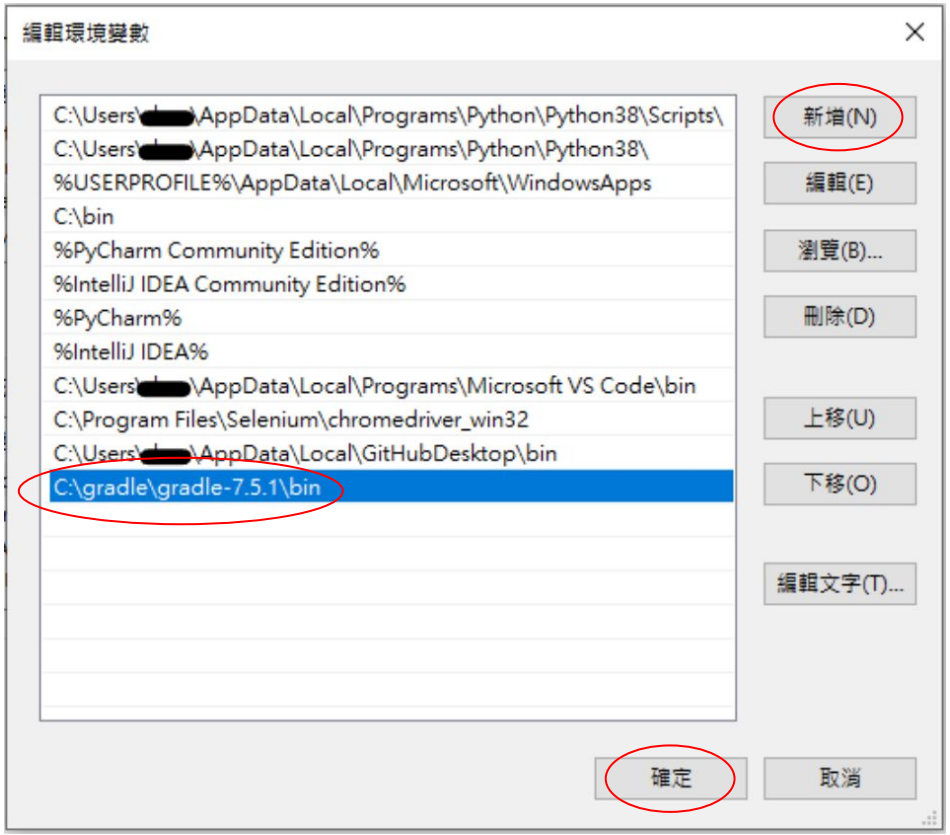
  - 點選 **環境變數**, 進行修改

安裝 Gradle (4/4)

- 設定環境變數
  - 點選使用者變數的 **Path**, 按下**編輯**



  - 點選**新增**
    輸入解壓縮後的資料夾路徑
    加上 \gradle-7.5.1\bin
    (C:\gradle\gradle-7.5.1\bin)



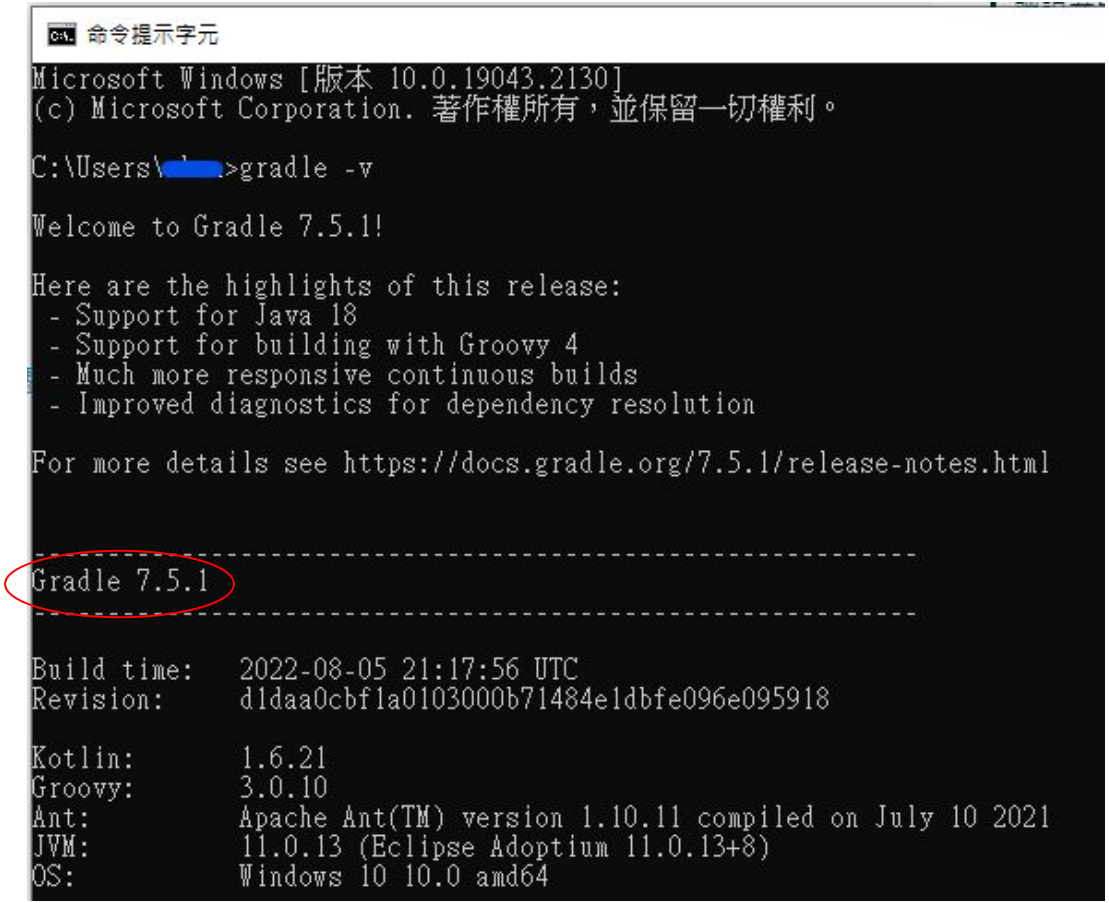
確認 gradle 安裝完成

- 開啟系統內建搜尋
- 搜尋並開啟

- 在命令提示字元中

  輸入 **gradle -v**

  查看 gradle 的版本



設定檔意義

- **build.gradle** 位於專案根目錄, 用以調整建構所需的設定

  ex: 定義專案所需的插件、libraries ...

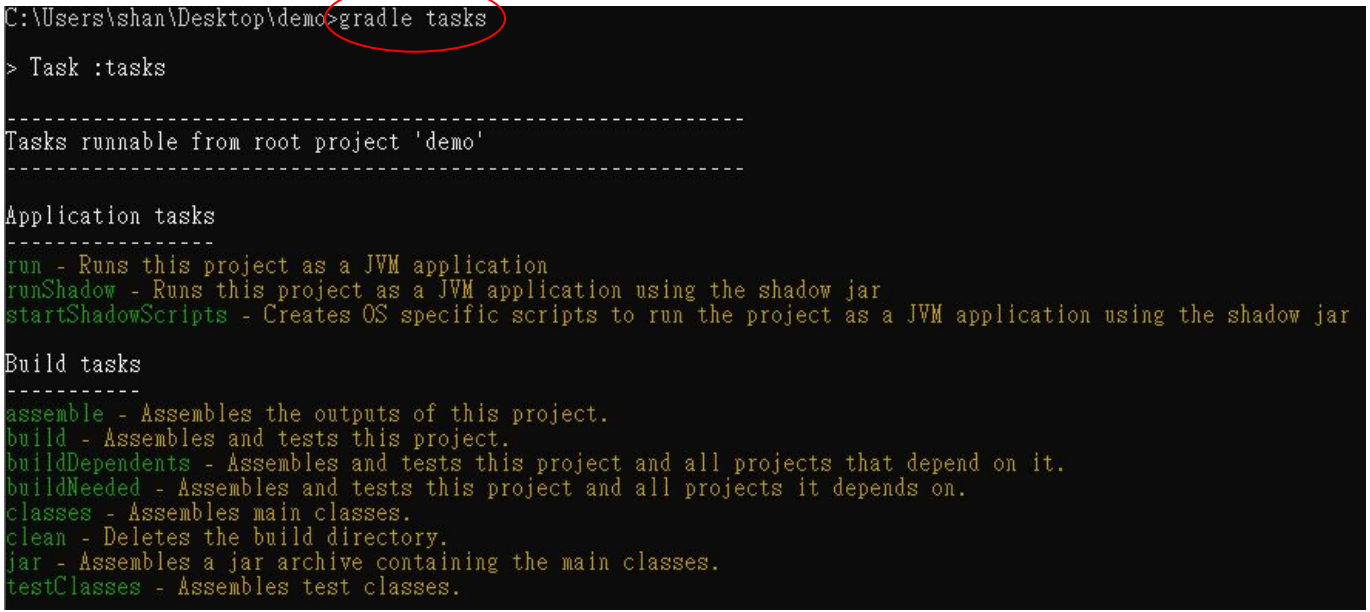
  - **plugins**
    - 插件可以增加 Gradle 的功能
      提供額外的指令控制第三方
      軟體的行為
    - ex: 可以加入 JaCoCo Plugins 將
      覆蓋率報告包含在建置流程中
  - **repositories**
    - 程式 libraries 的遠端儲存庫
  - **dependencies**
    - 設定 libraries 和版本
  - **application**
    - 程式客製化設定
      ex: main (程式入口點)

官方定義的 Maven 儲存庫
託管開源的 library 供 Java 專案使用

```
plugins {
    id 'java'
    id 'application'
}
repositories {
    mavenCentral()
}
dependencies {
    implementation 'group:name:version'
}
application {
    mainClass = 'demo.App'
}
```

Gradle 中的任務

- task：專案建構過程中的任務，表示一項工作
- 開啟命令提示字元，並輸入 **gradle tasks**，查看所有**預設的任務種類**

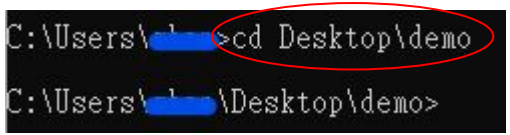


實際演練

初始化 Java 專案

使用 Gradle 初始化 Java 專案 (1/4)

- 新增一個 demo 資料夾
- 在命令提示字元中, 進入該 demo 資料夾 **(cd 資料夾路徑)**



- 輸入 **gradle init**, 並選擇要產生的專案類型 **(2:application)**
  - basic: 空的專案 / application: 有簡單內容的應用程式

```
C:\Users\[redacted]\Desktop\demo>gradle init
Starting a Gradle Daemon, 1 incompatible and 1 stopped Daemons could not be reused, use --status for details

Select type of project to generate:
  1: basic
  2: application
  3: library
  4: Gradle plugin
Enter selection (default: basic) [1..4] 2
```

使用 Gradle 初始化 Java 專案 (2/4)

- 選擇語 **(3:Java)**
  - 能以不同語言建立 application

```
Select implementation language:
  1: C++
  2: Groovy
  3: Java
  4: Kotlin
  5: Scala
  6: Swift
Enter selection (default: Java) [1..6] 3
```

- 選擇 "不" 拆分多個子專案 **(1:no)**
  - 2: 包含多個子專案 (資料夾) 的範例, 整體專案結構較複雜

```
Split functionality across multiple subprojects?:
  1: no - only one application project
  2: yes - application and library projects
Enter selection (default: no - only one application project) [1..2] 1
```

使用 Gradle 初始化 Java 專案 (3/4)

- 選擇 Groovy 作為腳本的語言 **(1: Groovy)**

```
Select build script DSL:
  1: Groovy
  2: Kotlin
Enter selection (default: Groovy) [1..2] 1
```

- 不使用新版 API 產生建構 **(保持預設選項, 直接按 Enter)**

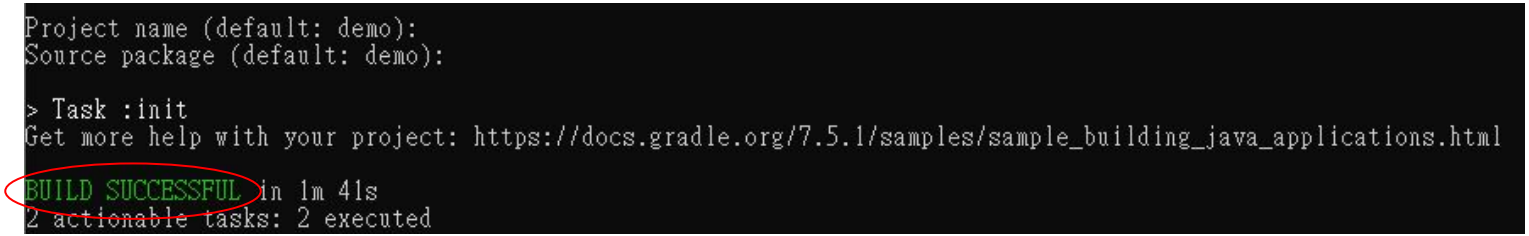
```
Generate build using new APIs and behavior (some features may change in the next minor release)? (default: no) [yes, no]
```

- 選擇單元測試框架 **(4: JUnit Jupiter)**

```
Select test framework:
  1: JUnit 4
  2: TestNG
  3: Spock
  4: JUnit Jupiter
Enter selection (default: JUnit Jupiter) [1..4] 4
```

使用 Gradle 初始化 Java 專案 (4/4)

- 專案名稱和 package 使用預設名稱, 直接**按 Enter**

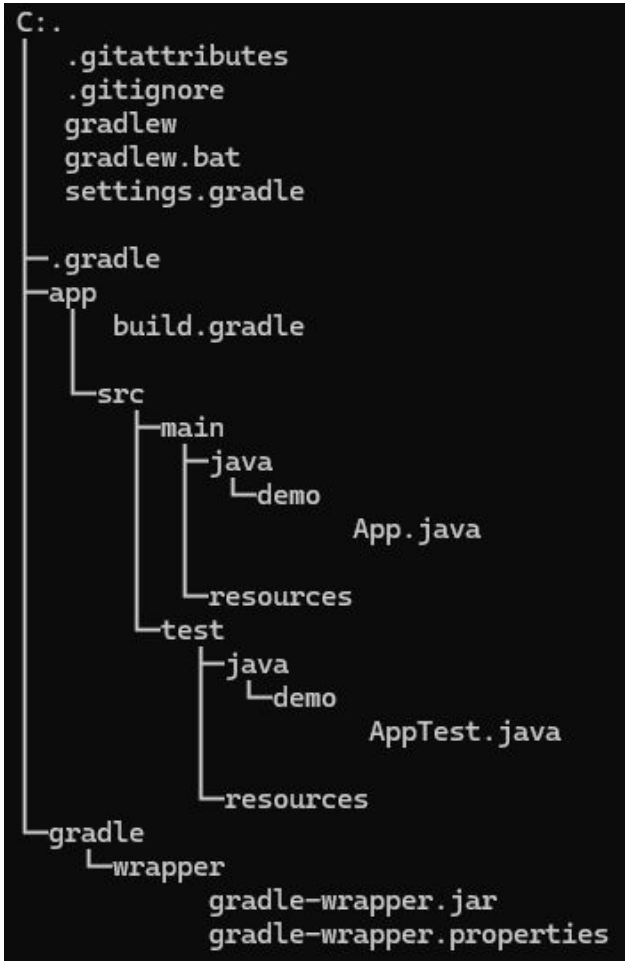


- 看到 **BUILD SUCCESSFUL**, 表示已經成功初始化專案

Gradle 專案結構

- **原始碼路徑**: src/main/java/*
- **測試程式碼路徑**: src/test/java/*
- **專案建置腳本:** build.gradle
- **其他 gradle 設定**:

  專案根目錄, 與 src 資料夾在同一層



設定檔說明 (build.gradle)

- demo\app\build.gradle
  - **plugins**
    - 套用 application 插件
  - **repositories**
    - 使用 maven 儲存庫
  - **dependencies**
    - 應用的 libraries
      寫法: **'group:name:version'**
  - **application**
    - 定義程式進入點 (main)

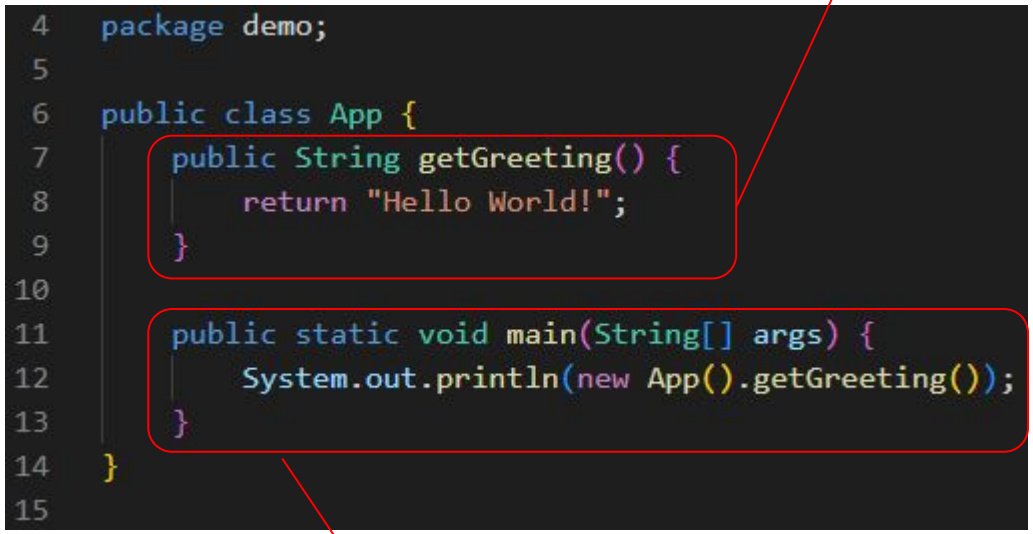
```
plugins {
    id 'application'
}
repositories {
    mavenCentral()
}
dependencies {
    testImplementation 'junit:junit:4.13.2'
    implementation 'com.google.guava:guava:31.0.1-jre'
}
application {
    mainClass = 'demo.App'
}
```

group name version

主程式說明 (App.java)

- demo\app\src\main\java\demo\App.java

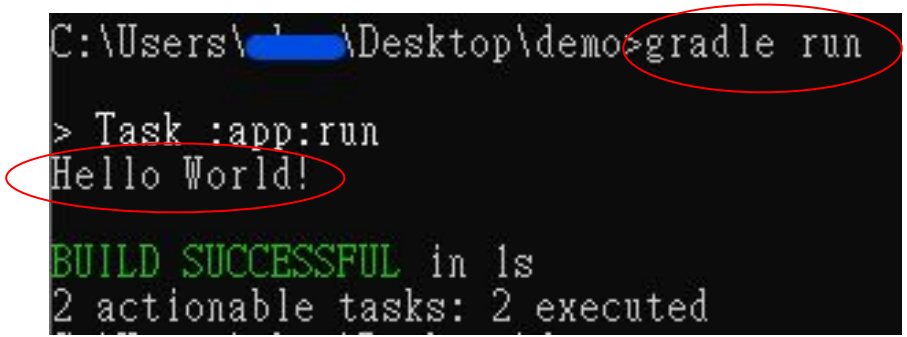
回傳 Hello World 的函式



呼叫函式, 並印出 Hello World

執行專案

- 任務 **run**：編譯 java 檔並產生 class 檔，接著執行class 檔
- 在命令提示字元中，輸入 **gradle run**



- 成功印出 Hello World!
  且 **BUILD SUCCESSFUL**

建置並封裝程式

- 任務 **build**:

  先編譯 java 檔, 接著將編譯產生的 class 檔打包成 jar 檔案

- 輸入 **gradle build**

```
C:\Users\[redacted]\Desktop\demo>gradle build

BUILD SUCCESSFUL in 2s
7 actionable tasks: 6 executed, 1 up-to-date
```

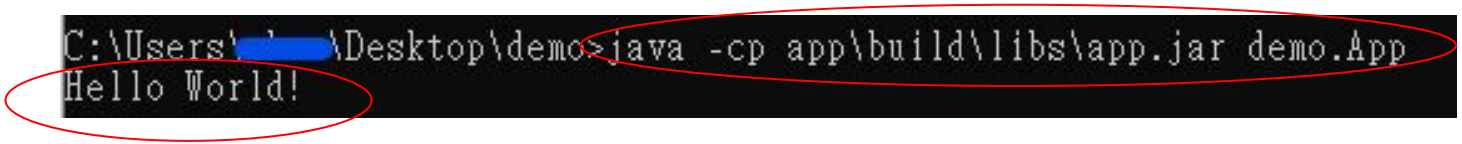
build 不包含執行程式
因此沒有結果輸出

- 成功建置後, 查看 **demo\app\build\libs** 資料夾
- Gradle 將產生 **app.jar 檔案**

執行 jar 檔

- 輸入 **java -cp app\build\libs\app.jar demo.App**

  以執行 jar 檔
- 印出 Hello World! 表示成功



實際演練 II

新增 plugin, library

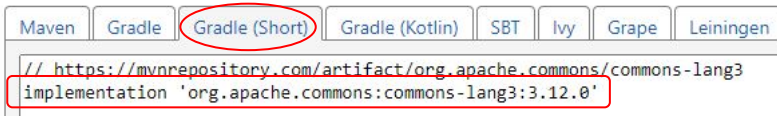
新增 Commons Lang library 判斷字串 (1/2)

- 引用在 Maven 儲存庫中的 Apache Commons Lang package

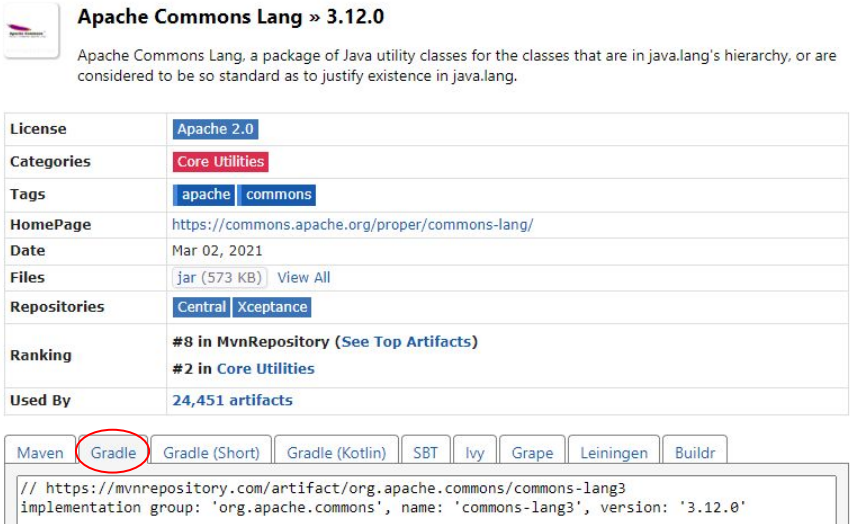
- 選擇最新版本: **3.12.0**

新增 Commons Lang library 判斷字串 (2/2)

- 選擇 **Gradle (Short)**, 查看寫法



- 複製新增 lib 的寫法
- 加在 **build.gradle** 的 dependencies { }




```
dependencies {
    testImplementation 'junit:junit:4.13.2'

    implementation 'com.google.guava:guava:31.0.1-jre'
    implementation 'org.apache.commons:commons-lang3:3.12.0'
}
```

設定 shadowJar 套件

預設的 build 方法只會打包編譯後的 class 檔, 由於程式引入 libraries, 因此需要 **shadowJar 套件**, 將外部的 **libraries 一起打包到 jar 檔**裡面。

- 在 **build.gradle** 中的 **plugins {}** 中新增 shadowJar 套件

```
plugins {
    id 'application'
    id 'com.github.johnrengelman.shadow' version '5.1.0'
}
```

- 在 **build.gradle** 的最後, 新增下方程式碼
  - 設定 shadowJar 任務
  - 設定 mainClass 名稱

```
shadowJar {
    mainClassName  = 'demo.App'
}
```

修改主程式 判斷字串是否為數字

- 在 demo\app\src\main\java\demo\App.java 中，import StringUtils 類別

```
import org.apache.commons.lang3.StringUtils;
```

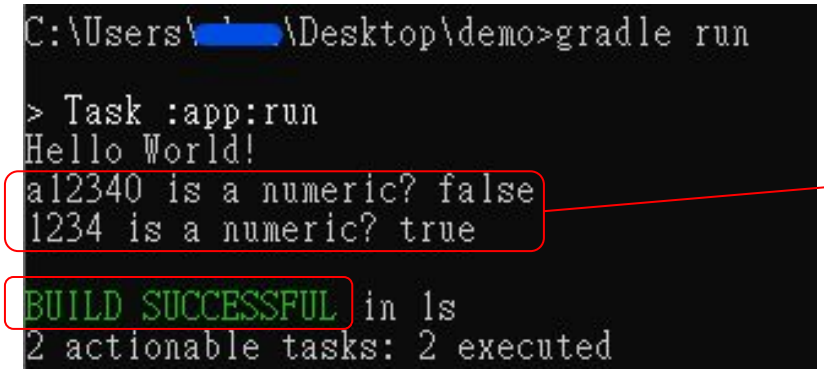
- 在 main 區塊新增以下程式碼，以判斷 text1、text2 字串是否為數字

```
public class App {
    public static void main(String[] args) {
        String text1 = "a12340";
        String text2 = "1234";

        boolean result1 = StringUtils.isNumeric(text1);
        boolean result2 = StringUtils.isNumeric(text2);

        System.out.println(text1 + " is a numeric? " + result1);
        System.out.println(text2 + " is a numeric? " + result2);
    }
}
```

執行程式

修改程式和設定檔後, 需要重新執行程式, 確保正確

- 任務 **run**:編譯 java 檔並產生 class 檔, 接著執行class 檔
- 在命令提示字元中, 輸入 **gradle run**



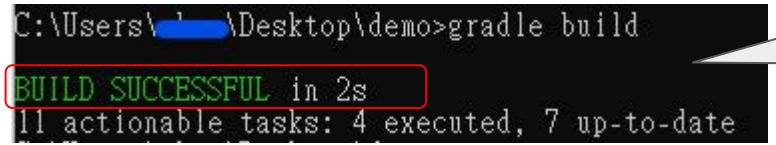
成功判斷字串

建置並封裝程式

- 任務 **build**:

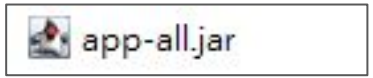
  先編譯 java 檔, 接著將編譯產生的 class 檔和 libraries 打包成 jar 檔案

- 在命令提示字元中, 輸入 **gradle build**



build 不包含執行程式
因此沒有結果輸出

- 成功後, **demo\app\build\libs** 下產生 **app-all.jar 檔案**



執行 jar 檔

- 輸入 **java -cp app\build\libs\app-all.jar demo.App** 以執行 jar 檔

- 能夠成功執行, 並印出結果