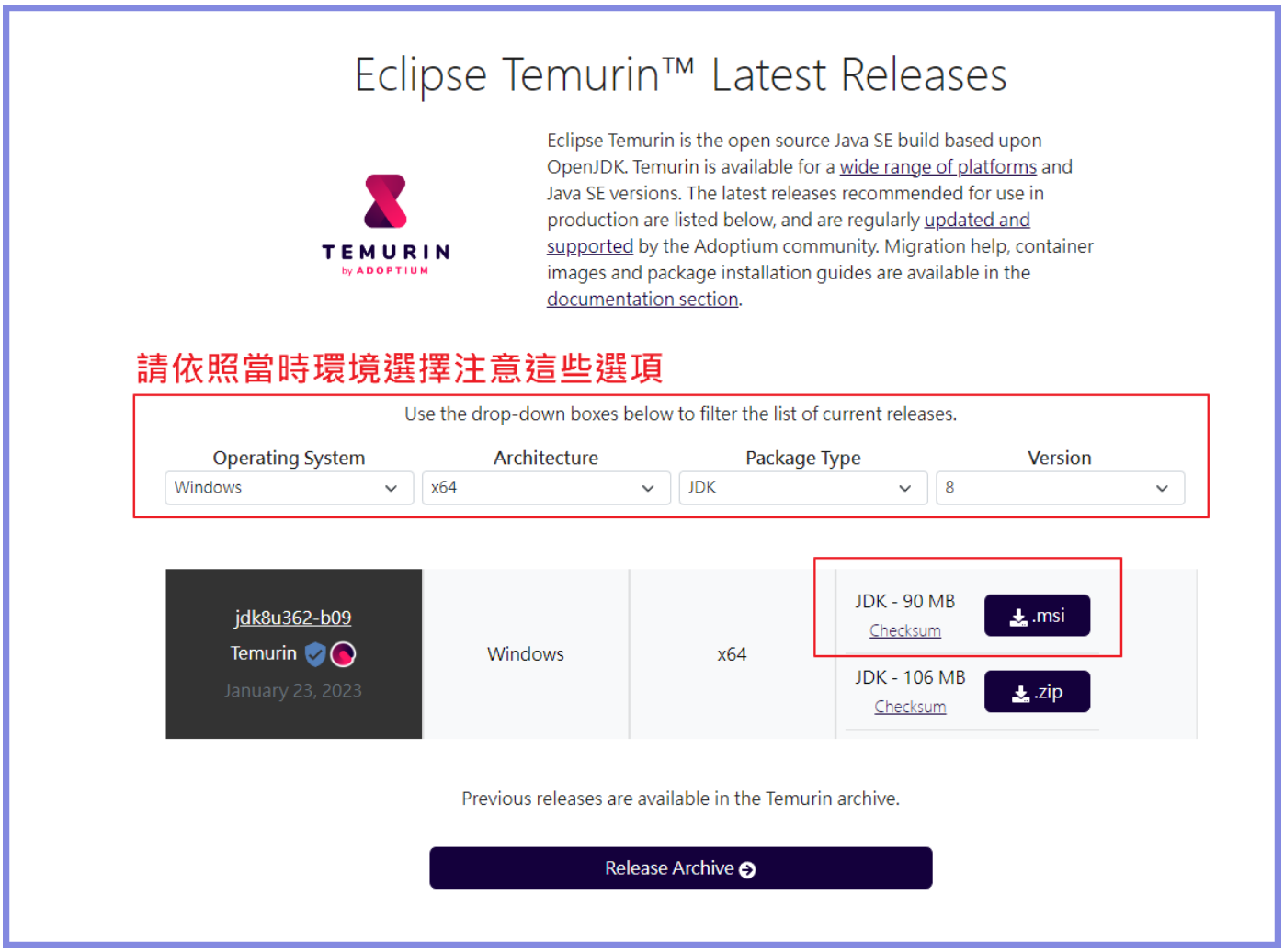
環境架設

Java

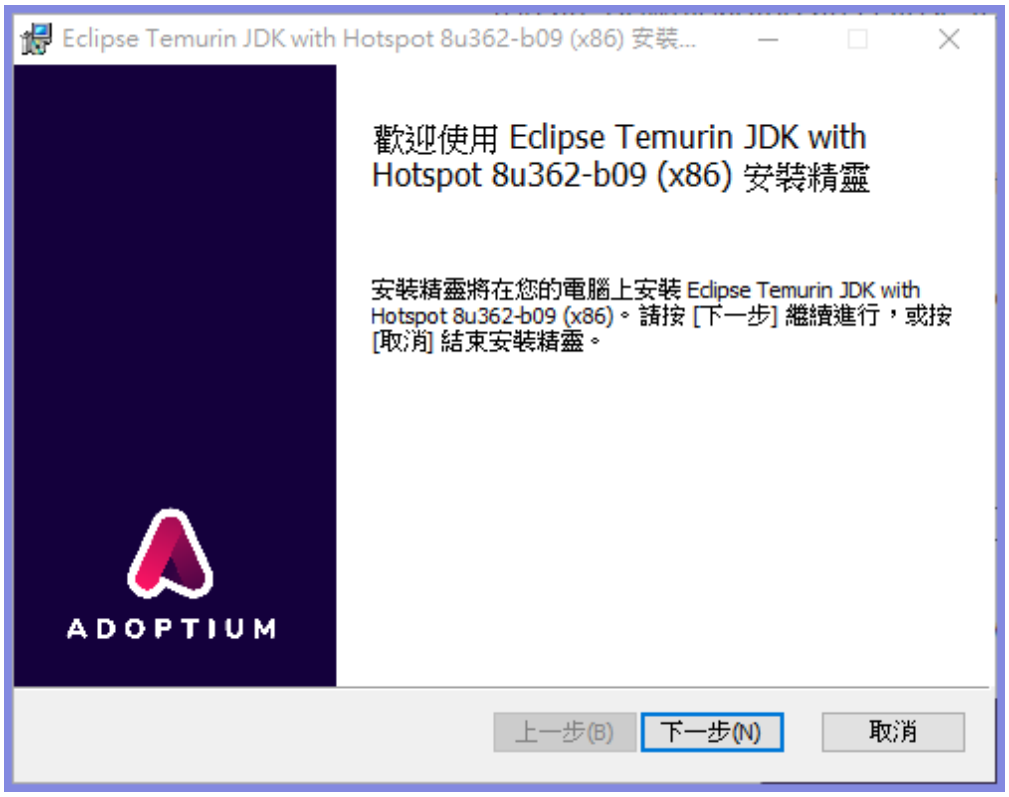
注意：因公司使用問題，本釋例為AdoptOpenJDK 8

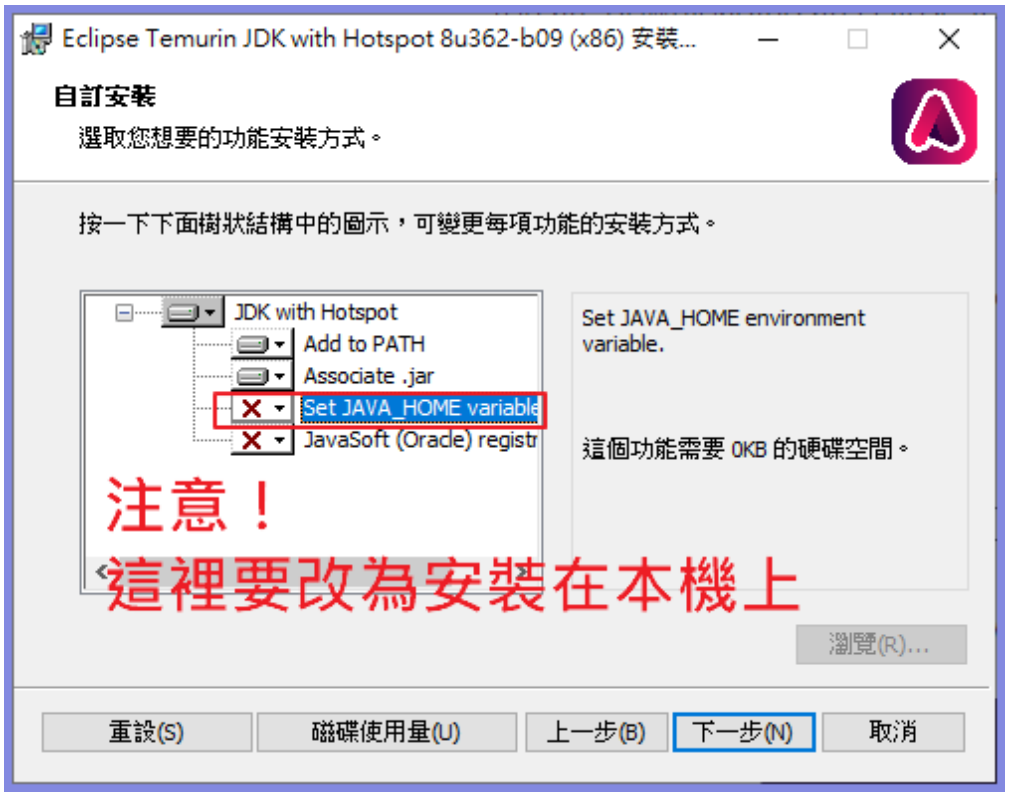
下載Java並安裝

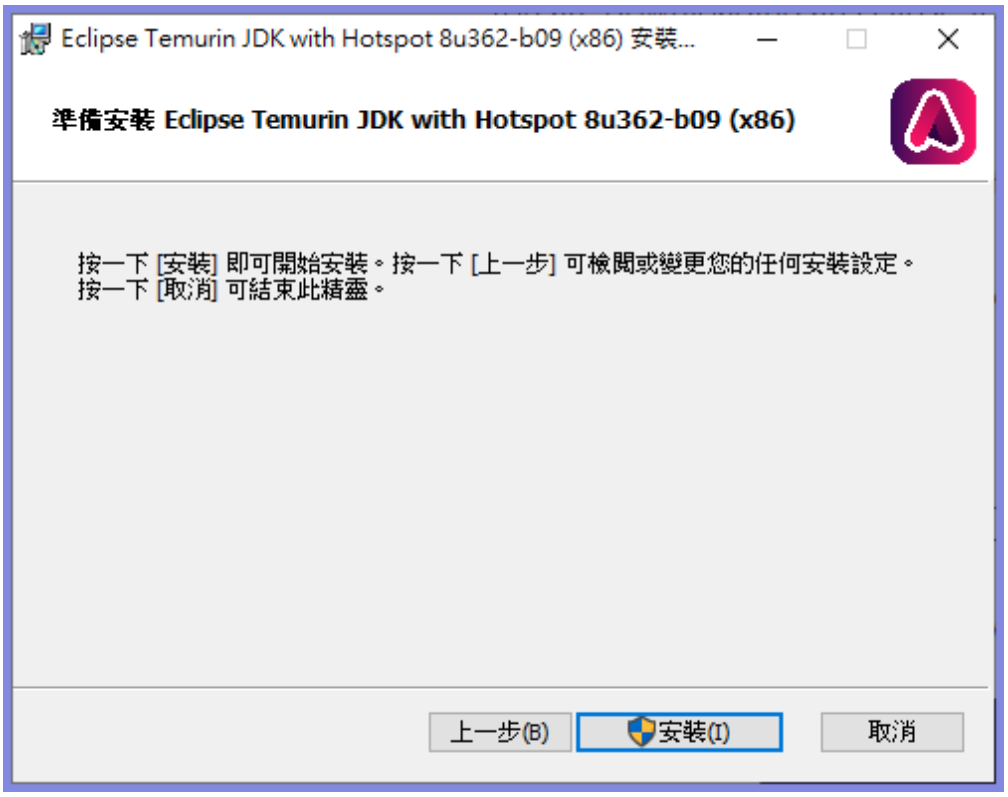
1. 點擊連結 https://adoptium.net/temurin/releases/?version=8



2. 下載後開啟


Eclipse Temurin JDK with Hotspot 8u362-b09 (x86) 安裝...
歡迎使用 Eclipse Temurin JDK with Hotspot 8u362-b09 (x86) 安裝精靈
安裝精靈將在您的電腦上安裝 Eclipse Temurin JDK with Hotspot 8u362-b09 (x86)。請按 [下一步] 繼續進行，或按 [取消] 結束安裝精靈。
ADOPTIUM
上一步(B)
下一步(N)
取消


Eclipse Temurin JDK with Hotspot 8u362-b09 (x86) 安裝...
自訂安裝
選取您想要的功能安裝方式。
按一下下面樹狀結構中的圖示，可變更每項功能的安裝方式。
JDK with Hotspot
Add to PATH
Associate .jar
Set JAVA_HOME variable
JavaSoft (Oracle) registr
Set JAVA_HOME environment variable.
這個功能需要 0KB 的硬碟空間。
注意！
這裡要改為安裝在本機上
瀏覽(R)...
重設(S)
磁碟使用量(U)
上一步(B)
下一步(N)
取消

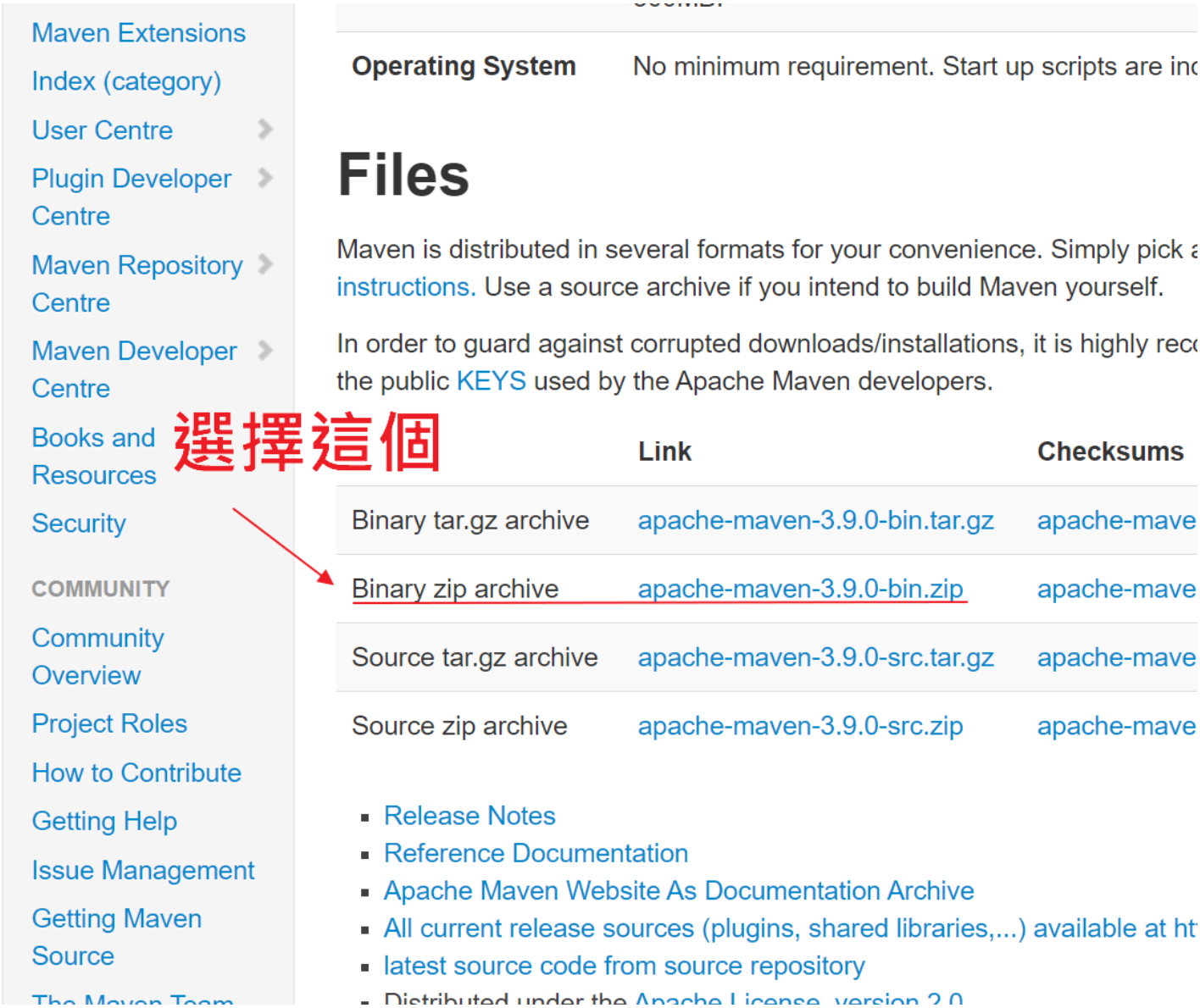


3. 確認安裝成功
    1. 開啟CMD
    2. 輸入 `java -version` 與 `javac` ，看到以下訊息便表示安裝成功

```
命令提示字元
Microsoft Windows [版本 10.0.19045.2486]
(c) Microsoft Corporation. 著作權所有，並保留一切權利。

C:\Users\Sander>java -version
openjdk version "1.8.0_362"
OpenJDK Runtime Environment (Temurin)(build 1.8.0_362-b09)
OpenJDK Client VM (Temurin)(build 25.362-b09, mixed mode)

C:\Users\Sander>javac
Usage: javac <options> <source files>
where possible options include:
  -g                         Generate all debugging info
  -g:none                    Generate no debugging info
  -g:{lines,vars,source}     Generate only some debugging info
  -nowarn                    Generate no warnings
  -verbose                   Output messages about what the compiler is doing
  -deprecation               Output source locations where deprecated APIs are used
  -classpath <path>          Specify where to find user class files and annotation processors
  -cp <path>                 Specify where to find user class files and annotation processors
  -sourcepath <path>         Specify where to find input source files
  -bootclasspath <path>      Override location of bootstrap class files
  -extdirs <dirs>            Override location of installed extensions
  -endorseddirs <dirs>       Override location of endorsed standards path
  -proc:{none,only}          Control whether annotation processing and/or compilation is done.
  -processor <class1>[,<class2>,<class3>...] Names of the annotation processors to run; bypasses default discovery process
  -processorpath <path>      Specify where to find annotation processors
  -parameters                Generate metadata for reflection on method parameters
  -d <directory>             Specify where to place generated class files
  -s <directory>             Specify where to place generated source files
  -h <directory>             Specify where to place generated native header files
  -implicit:{none,class}     Specify whether or not to generate class files for implicitly referenced files
  -encoding <encoding>       Specify character encoding used by source files
  -source <release>          Provide source compatibility with specified release
  -target <release>          Generate class files for specific VM version
  -profile <profile>         Check that API used is available in the specified profile
  -version                   Version information
  -help                      Print a synopsis of standard options
  -Akey[=value]              Options to pass to annotation processors
  -X                         Print a synopsis of nonstandard options
  -J<flag>                   Pass <flag> directly to the runtime system
  -Werror                    Terminate compilation if warnings occur
  @<filename>                Read options and filenames from file


C:\Users\Sander>
```

Maven

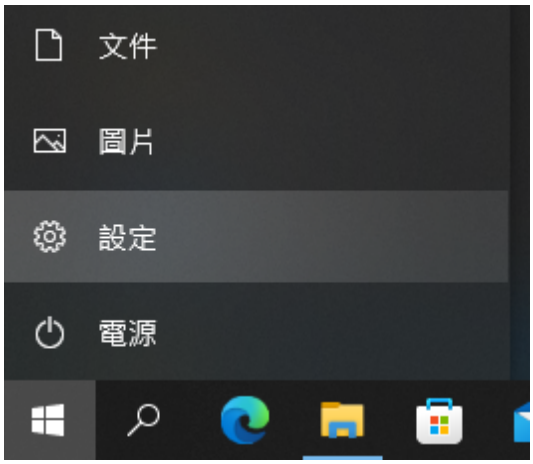
下載Maven並且解壓縮



選定一個資料夾解壓縮，請記住路徑

設定Maven Home

1. 點 設定 ，點 系統



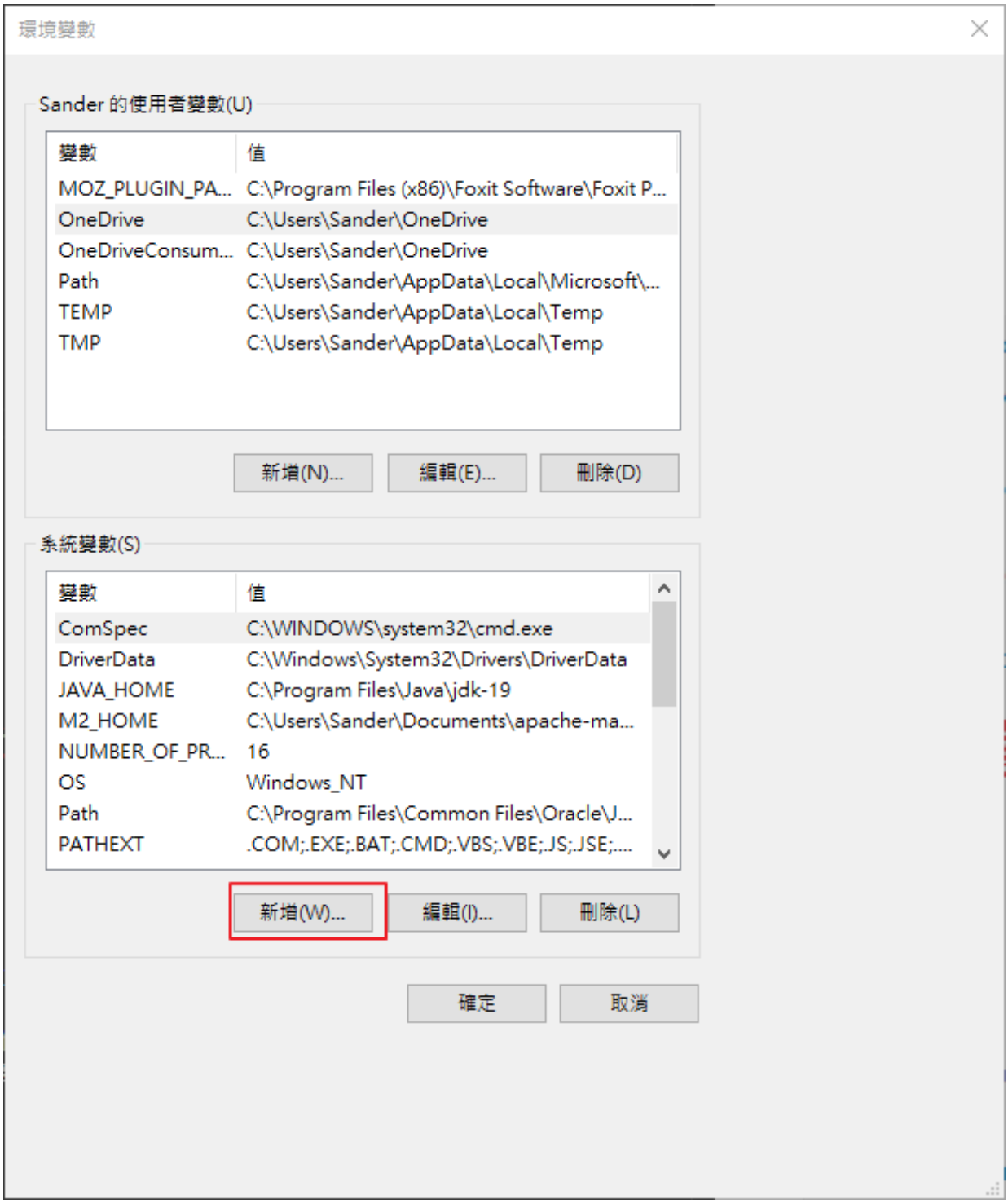
2. 點 關於 ，點 進階系統設定


設定
首頁
尋找設定
系統
顯示器
音效
通知與動作
專注輔助
電源與睡眠
儲存體
平板
多工
投影到此電腦
共用體驗
剪貼簿
遠端桌面
關於
1. 先點這個
關於
系統正在監控並保護您的電腦。
參閱 Windows 安全性中的詳細資訊
裝置規格
2. 再點這個
裝置名稱 DESKTOP-U2BJ55P
處理器 AMD Ryzen 7 5800X 8-Core Processor 3.80 GHz
已安裝記憶體(RAM) 64.0 GB
裝置識別碼 BF1A70EA-0C95-43CD-8F25-515141354A59
產品識別碼 00330-81486-40830-AA446
系統類型 64 位元作業系統，x64 型處理器
手寫筆與觸控 此顯示器不提供手寫筆或觸控式輸入功能
複製
重新命名此電腦
Windows 規格
版本 Windows 10 專業版
版本 22H2
安裝於 2022/12/15
OS 組建 19045.2486
體驗 Windows Feature Experience Pack 120.2212.4190.0
複製
變更產品金鑰或升級您的 Windows 版本
閱讀適用於我們的服務的 Microsoft 服務合約
閱讀 Microsoft 軟體授權條款
相關設定
BitLocker 設定
裝置管理員
遠端桌面
系統保護
進階系統設定
重新命名此電腦 (進階)
取得協助
提供意見反應

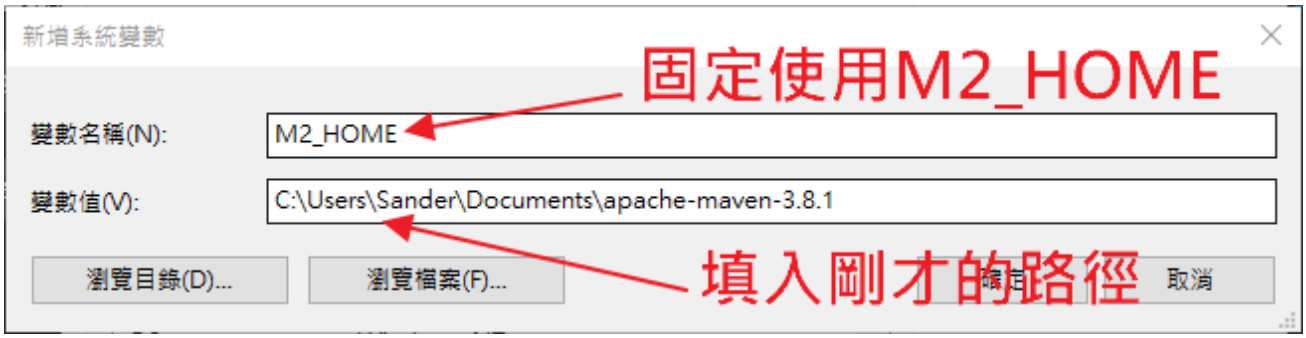
3. 點 環境變數


系統內容
電腦名稱
硬體
進階
系統保護
遠端
您必須以系統管理員的身分登入，才能變更這裡的大部分設定。
效能
視覺效果、處理器排程、記憶體使用量和虛擬記憶體
設定(S)...
使用者設定檔
關於您登入時的桌面設定
設定(E)...
啟動及修復
系統啟動、系統失敗、及偵錯資訊
設定(T)...
環境變數(N)...
確定
取消
套用(A)

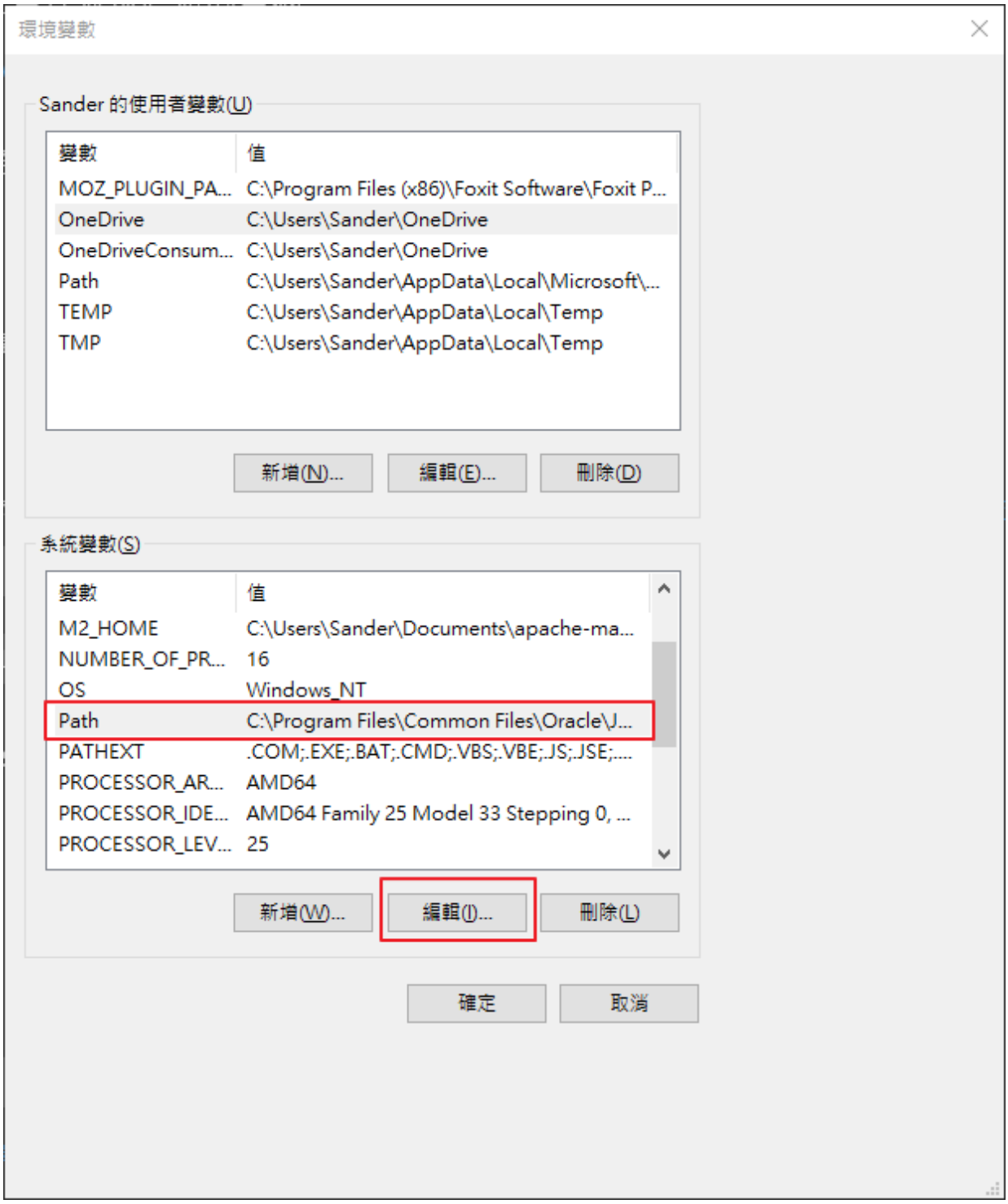
4. 點 新增



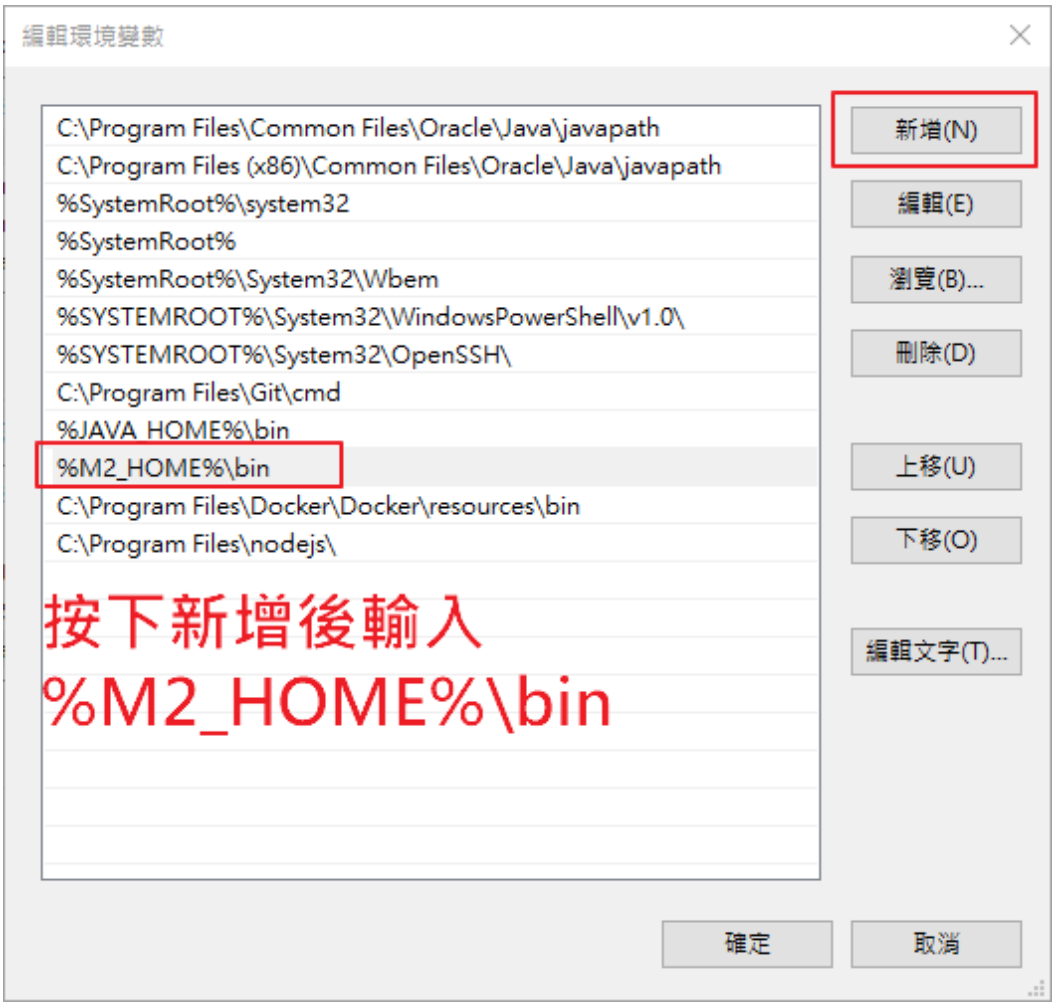
5. 變數名稱固定使用 `M2_HOME`



6. 編輯 `Path` 變數



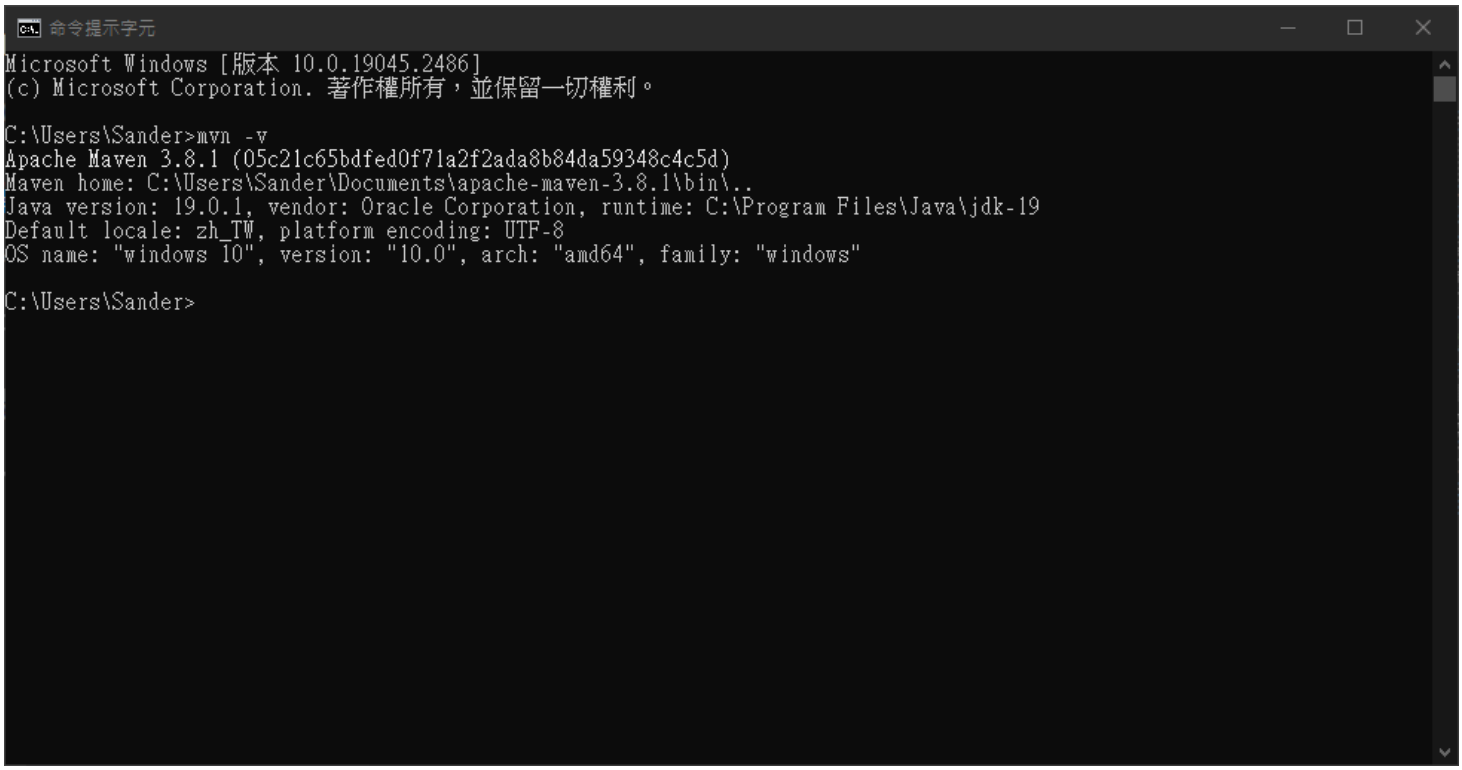
7. 點 新增 ， `%M2_HOME%\bin`



以上便設定完成。

確認安裝成功

1. 開啟CMD
2. 輸入 `mvn -v` ，看到以下訊息便表示安裝成功



Maven資料夾結構

```
project-name/
├─ src/
│  ├─ main/
│  │  ├─ java/
│  │  ├─ resource/
│  ├─ test/
│  │  ├─ java/
│  │  ├─ resource/
├─ target/
├─ pom.xml
```

src/main/java

主要放程式碼

src/main/resource

主要放配置檔案

src/test/java

主要放測試程式碼

`src/test/resource`

主要放測試用的配置檔案

`target`

放編譯/打包好的檔案，通常會列入 `.gitignore`

`pom.xml`

`Maven` 最重要的設定檔案，每個專案都藉由 `pom.xml` 進行依賴關係設定與打包的流程。

基礎語法

第一個程式

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

定義類別

> *撰寫 Java 程式通常都是由定義「類別」開始，"class" 是 Java 用來定義類別的關鍵字，範例中類別的名稱是 App，這與您編輯的檔案（App.java）主檔名必須相同，App 類別使用關鍵字 "public" 宣告。*
>
> *在編寫 Java 程式時，**一個檔案中可撰寫數個類別，但是只能有一個公開（public）類別，而且檔案主檔名必須與這個公開類別的名稱相同**，在定義類別名稱時，建議將類別首字母大寫，並在類別名稱上表明類別的作用。*
>
> *Java SE 6 技術手冊，第一個Java程式*

main()方法解說

main() 是 Java 程式的「進入點」（Entry point），程式的執行是由進入點開始的。

- public：表示此方法可以被外部呼叫。
- static: Main方法不需要實例化物件便可執行

- void：主方法是程式的起點，所以不需要任何的返回值。
- main: 系統規定好預設呼叫的方法名稱，執行時預設找到main方法名稱。

System.out.print

這裡不琢磨太多IO相關的問題，簡單來講就是調用了System類別中的out去把文字輸出。

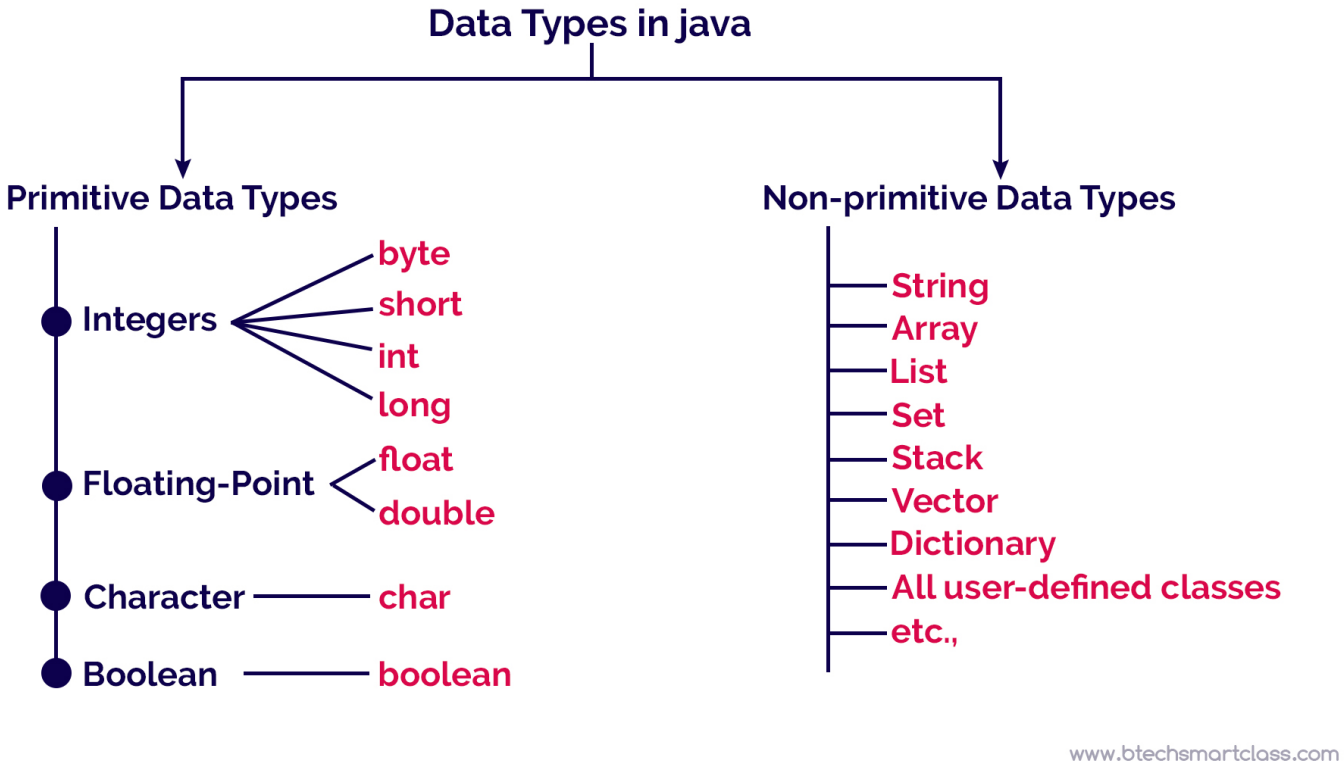
註解

跟大部分語言一樣，語法為 `//` ，或是 `/* */` ，要注意的是 `/** */` 為Javadoc，生成Java手冊用的工具，可以再查API的時候看到。

```
public class App {
    public static void main(String[] args) {

        // This is comment
        /*
         * Hello world!
         */
        /**
         * This is javadoc
         * @authoer  Sander Chen
         */
        System.out.println("Hello World!");
    }
}
```

基礎資料型態與運算子

基礎資料型態(Primitive Type)




```
short   數值範圍：32767 ~ -32768
int     數值範圍：2147483647 ~ -2147483648
long    數值範圍：9223372036854775807 ~ -9223372036854775808
byte    數值範圍：127 ~ -128
float   數值範圍：3.402823e+38 ~ 1.401298e-45
double  數值範圍：1.797693e+308 ~ 4.900000e-324
```

運算子

- 算數運算子

Arithmetic Operators

Operator	Meaning	Example	Result
+	Addition	10 + 2	12
-	Subtraction	10 – 2	8
*	Multiplication	10 * 2	20
/	Division	10 / 2	5
%	Modulus (remainder)	10 % 2	0
++	Increment	a++ (consider a = 10)	11
--	Decrement	a-- (consider a = 10)	9
+=	Addition Assignment	a += 10 (consider a = 10)	20
-=	Subtraction assignment	a -= 10 (consider a = 10)	0
*=	Multiplication assignment	a *= 10 (consider a = 10)	100
/=	Division assignment	a /= 10 (consider a = 10)	1
%=	Modulus assignment	a %= 10 (consider a = 10)	0

Startertutorials.com

- 邏輯運算子

Logical Operators

int a = 10, b = 2 for all examples below

Operator	Meaning	Example	Result
&	Logical AND	(a>10) & (b<3)	false
|	Logical OR	(a>10) & (b<3)	true
^	Logical exclusive OR (XOR)	(a>10) & (b<3)	true
!	Logical NOT	a>10	true
&&	Short-circuit AND	(a>10) & (b<3)	false
||	Short-circuit OR	(a>10) & (b<3)	true
==	Equal to	a == 10	True
!=	Not equal to	a != 10	False
&=	AND assignment	(a>10) & (b<3)	false
|=	OR assignment	(a>10) & (b<3)	true
^=	XOR assignment	(a>10) & (b<3)	true
?:	Ternary if-then-else	(a==10) ? true : false	true

Startertutorials.com

- 比較運算子

Relational Operators

int a = 10, b = 2 for all examples below

Operator	Meaning	Example	Result
==	Equals to	a == b	false
!=	Not equal to	a != b	true
<	Less than	a < b	false
<=	Less than or equal to	a <= b	false
>	Greater than	a > b	true
>=	Greater than or equal to	a >= b	true

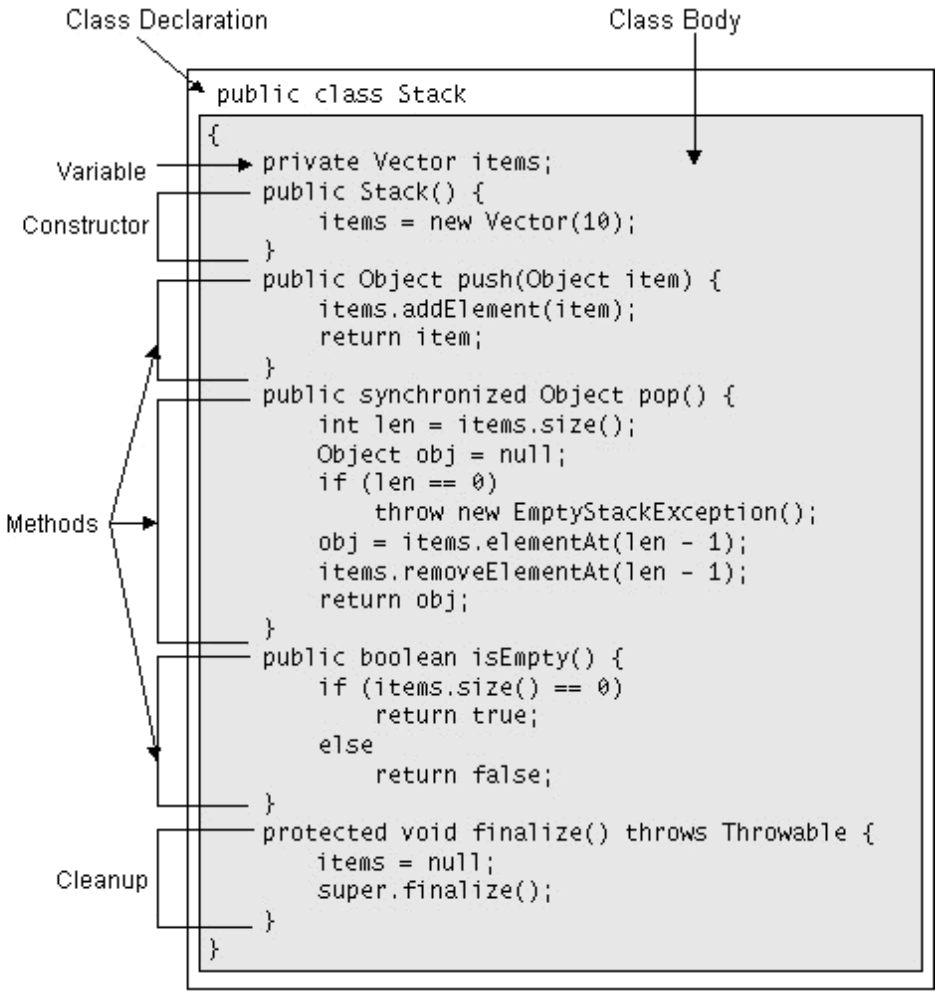
Startertutorials.com

流程控制

請參考FlowControl.java

物件基礎

定義類別 Define Class



類別宣告 The Class Declaration

第一行內我們可以分為幾個部分：

1. 權限修飾子 非必須

定義這個Class的Scope(可視範圍)，要注意與Field(成員變數)與Method的差別是沒有 `protected`

至於為什麼沒有可以參考這篇stackoverflow

限修飾子	在同一Class	在同一Package	不同Package
Private	Y	N	N

限修飾子	在同一Class	在同一Package	不同Package
Default(不寫)	Y	Y	N
Public	Y	Y	Y

2. `abstract` 抽象 非必須

這個關鍵字用在宣告Class的時候，這個Class便不能被實例化(instantiate)。

簡單講，就是無法用 `new` 去產生物件。

3. `final` 非必須

這個關鍵字用在宣告Class的時候，這個Class便不能被繼承。

簡單講，就是無法有子類別。

題外話， `final` 如果再成員變數上使用的話跟 `javascript` 的 `const` 很像，不能重複 `assign value`

4. `class {NameOfClass}` 必須

`class` 關鍵字讓 `compiler` 知道你要定義的是類別， `{NameOfClass}` 則是定義類別的名稱，規則之前有提到：**在定義類別名稱時，建議將類別首字母大寫**

5. `extends {SuperClass}` 非必須

`extends` 關鍵字代表繼承，被繼承者須為類別，這個我們之後會提到。這邊只要注意的是 **Java繼承只能使用一次(單一繼承)，與C++多重繼承不同。**

6. `implements {Interface}` 非必須

`implements` 關鍵字代表實作，被實作者須為介面，依樣之後會提到，這邊只要注意的是 **Java可以無限實作介面，與繼承只能一次不同**

類別體

由 `{}` 包覆，裡面會有包含以下幾個元素：

- 變數

主要有以下兩種

1. 成員變數
2. 類別變數

- 方法

1. 成員方法
2. 類別方法
3. 有個特別的 **類別同名方法**，我們稱為 **建構子(`constructor`)**

我們先從變數與宣告講起：

變數與宣告 Declaring Variables

```
private Vector items;
```

而宣告變數中也有幾個元素可以參考

accessLevel	Indicates the access level for this member.
static	Declares a class member.
final	Indicates that it is constant.
transient	This variable is transient.
volatile	This variable is volatile.
type name	The type and name of the variable.

1. 權限修飾子(access level) 非必須
   我們剛剛有提到類別的權限修飾子，而變數也有，這時候要注意多了一個 `protected` 。

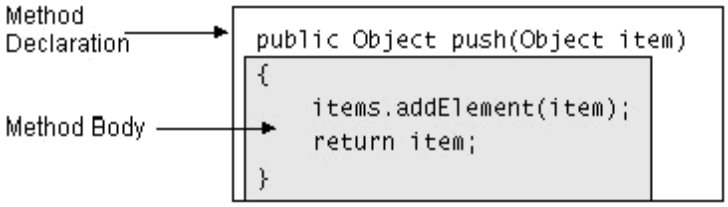
權限修飾子	在同一Class	在同一Package	不同Package但是子類別繼承	不同Package
Private	Y	N	N	N
Default(不寫)	Y	Y	N	N
Protected	Y	Y	Y	N
Public	Y	Y	Y	Y

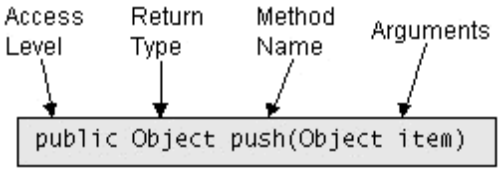
2. `static` 非必須
   加上了static之後就變成了 `類別變數` ，成員變數必須實例化之後才會被創造，個別實體的成員變數是獨立的。而類別變數則是共享的且無須等類別被實例化，所有實體都可以分享該類別變數。
3. `final` 非必須
   與類別的不能繼承不同，這裡指的是變數不能被re-assign value(類似 `js const`)
4. `transient` 非必須
   很少用 ，通常是用在序列化 `Object Serialization` 用，但Java現在已經很少再用甚至不被推薦使用，有興趣可以參考我之前寫的我在想甚麼時候該用Serializable

5. `volatile` 非必須
   進階Java才會用到，主要是解決多執行續的相關問題，這議題不再這個光速講義的範圍裡面。
6. `{type}` 必須
   型別，可以是基礎型別 `primitive type` 像是 `short` , `int` ...，或是類別型態(`Class type`)像是 `String` 等等。
7. `{name}` 必須
   參數名，只要是legal Java identifier 都可以。約定成俗的命名方式是 **開頭小寫**

方法 Method

分為兩個副分， 方法宣告`method declaration` 與 方法體`method body`





方法宣告method declaration

accessLevel	Access level for this method.
`static`	This is a class method.
`abstract`	This method is not implemented.
`final`	Method cannot be overridden.
`native`	Method implemented in another language.
synchronized	Method requires a monitor to run.
returnType methodName	The return type and method name.
(*paramList*)	The list of arguments.
`throws` *exceptions*	The exceptions thrown by this method.

1. 權限修飾子(access level) 非必須
   跟宣告變數的一樣，一樣注意多了一個 `protected` 。

權限修飾子	在同一Class	在同一Package	不同Package但是子類別繼承	不同Package
Private	Y	N	N	N
Default(不寫)	Y	Y	N	N
Protected	Y	Y	Y	N

權限修飾子	在同一Class	在同一Package	不同Package但是子類別繼承	不同Package
Public	Y	Y	Y	Y

2. `static` 非必須
   加上了static之後就變成了 類別方法 ，跟變數一樣，宣告了之後所有實體皆可調用。
3. `abstract` 非必須

這裡的抽象與類別的 `abstract` 相呼應，寫上之後便不用寫方法體，留給繼承的子類別去implement。

4. `final` 非必須

寫上之後繼承的子類別不能覆寫(Override)。

5. `native` 非必須

通常用於調用其他語言的方法，不在此討論。

6. `synchronized` 非必須

通常用於多執行序加鎖，不在此討論。

7. `returnType` 必須

回傳類型，可以是基礎型別 `primitive type` 或是複雜型別，如果不回傳則需使用 `void` 關鍵字。

8. `{methodName}` 必須
   方法名，只要是legal Java identifier 都可以，取名時要小心 `Overload` 問題
9. `(paramlist)` 必須
   傳入參數，如果不傳入參數則寫上 `()`
10. `[throws exceptions]` 非必須
    需要顯式寫出拋出受檢例外 `checked exception` 時須加上，這個等到 `Exception` 篇章再詳細討論

方法體method body

這裡有幾個重點要提：

1. `this`

通常用來給人類看，避免造成誤會，譬如成員變數與方法參數同名時，可以用 `this` 來表達該實體的參數，請參照以下範例：

```
class HSBColor {
    int hue, saturation, brightness;
    HSBColor (int hue, int saturation, int brightness) {
        this.hue = hue;
        this.saturation = saturation;
        this.brightness = brightness;
    }
}
```

2. 區域變數

如同其他語言一樣，方法結束則變數消失。

```
Object findObjectInArray(Object o, Object[] arrayOfObjects) {
    //方法結束後i則消失
    int i;      // local variable
    for (i = 0; i < arrayOfObjects.length; i++) {
        if (arrayOfObjects[i] == o)
            return o;
    }
    return null;
}
```

註：Overload與Override

Overload(多載)，是指在**同一 Class 內有同名方法但參數的個數、型態不同**，但要注意僅只有參數不同，整個方法簽名(Method Signature)改變不一定能稱為Overload。請參考以下範例：

- 原有方法

```
public void show(String message) {
    System.out.println(message);
}
```

- Overload - 參數個數不一

```
public void show(String message, boolean show) {
    System.out.println(message);
}
```

- Overload - 參數型態不一致

```
public void show(Integer message) {
    System.out.println(message);
}
```

- 非Overload - 改變回傳參數

```
public boolean show(String message) {
    System.out.println(message);
    return false;
}
```

Override(覆寫)，是指在**有繼承關係的 `Class` 內方法簽名(Method Signature)一致的同名方法**，請參考以下範例：

```
public class Animal {
    public void say(String message) {
        System.out.println(message);
    }
}

// 方法簽名一樣但是在不一樣的繼承類別
class Dog extends Animal {
    @Override
    public void say(String message) {
        System.out.println("I'm dog, bark!");
    }
}
```

運行看看...

```
public class App {
    public static void main( String[] args ) {
        Animal animal = new Dog();
        animal.say("I'm unknown animal");
    }
}


Output:
I'm dog, bark!
```

註：@Override可以不加，依然可以編譯通過並執行。但加了有兩個好處：

1. 編譯器會幫你檢查Overriding的方法是否與父類別的簽名保持一致，避免出錯。
2. 對於人類來讀程式碼的時候，會比較好理解。

Overload與Override都是**Java實現多型的方法**。具體差異如下：

Overriding	Overloading
執行時期多型	編譯時期多型
執行時期決定哪個方法應該被呼叫	編譯時期就決定呼叫哪個方法
存在與父子類別	存在於同一類別
以一樣的方法簽名	只有方法名字一樣，參數不同
出錯時，只有執行時期看的見	出錯可以在編譯時期就看見

建構子 Constructor

類別同名方法，所有Java類別都有建構子去實例化物件。因Java有Overload(多載)機制，所以建構子可以有多個。而呼叫時Java會判斷類型與參數數量去決定使用哪個建構子。不過建構子不一定要寫，如果compiler沒偵測到類別中有顯式寫出建構子的話，會自動提供預設建構子(default constructor)，但要注意的是裡面的成員類別因為沒有assigned value，所有的成員變數均為預設值(如int = 0, 自定義類別為null)。

而建構子也可以套用權限修飾子。

權限修飾子	作用
private	誰都不行實例化類別，通常用於Singleto Pattern
protected	只有繼承類別可以實例化
public	誰都可以實例化
default	只有同package下的類別可以實例化

封裝，繼承，多型

Exception

簡介與語法

> 程式中的臭蟲（Bug）總是無所不在，即使您認為程式中應該沒有錯誤了，臭蟲總會在某個時候鑽出來困擾您，面對程式中各種層出不窮的錯誤，Java 提供了「例外處理」機制來協助開發人員避開

> 可能的錯誤，「例外」（Exception）在 Java 中代表一個錯誤的實例，編譯器會幫您檢查一些可能產生例外（Checked exception）的狀況，並要求您注意並處理，而對於「執行時期例外」（Runtime exception），您也可以嘗試捕捉例外並將程式回復至正常狀況。
>
> *Java SE 6 技術手冊，第 10 章 例外處理（Exception Handling）*

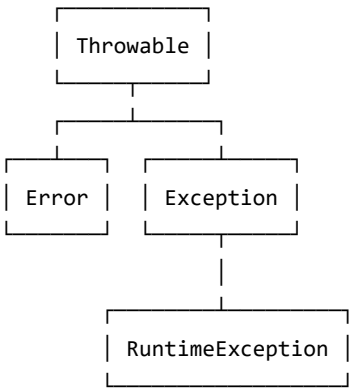
在 Java 中如果想嘗試捕捉例外，可以使用 "try"、"catch"、"finally" 三個關鍵字組合的語法來達到，其語法基本結構如下：

```
try {
    // 陳述句
}
catch(例外型態 名稱) {
    // 例外處理
}
finally {
    // 一定會處理的區塊
}
```

一個 "try" 語法所包括的區塊，必須有對應的 "catch" 區塊或是 "finally" 區塊，"try" 區塊可以搭配多個 "catch" 區塊，如果有設定 "catch" 區塊，則 "finally" 區塊可有可無，如果沒有定義 "catch" 區塊，則一定要有 "finally" 區塊。

受檢例外（Checked Exception）、執行時期例外（Runtime Exception）

架構圖



受檢例外

> 例如使用輸入輸出功能時，可能會由於硬體環境問題，而使得程式無法正常從硬體取得輸入或進行輸出，這種錯誤是可預期發生的，像這類的例外稱之為「受檢例外」（Checked Exception）
>
> *Java SE 6 技術手冊，第 10 章 例外處理（Exception Handling）*

簡單來說，就是一定要處理的Excpetion，不管繼續拋出去或是攔截處理。

執行時期例外

> 「執行時期例外」（Runtime exception），也就是例外是發生在程式執行期間，並不一定可預期它的發生，編譯器不要求您一定要處理，對於執行時期例外若沒有處理，則例外會一直往外丟，最後由 JVM 來處理例外，JVM 所作的就是顯示例外堆疊訊息，之後結束程式。
>
> *Java SE 6 技術手冊，第 10 章 例外處理（Exception Handling）*

簡單來說，不強制要處理

throw、throws

> 當程式發生錯誤而無法處理的時候，會丟出對應的例外物件，除此之外，在某些時刻，您可能會想要自行丟出例外，例如在捕捉例外並處理結束後，再將例外丟出，讓下一層例外處理區塊來捕捉；另一個狀況是重新包裝例外，將捕捉到的例外以您自己定義的例外物件加以包裝丟出。若想要自行丟出例外，您可以使用 "throw" 關鍵字，並生成指定的例外物件，例如：
>
> ```
> javathrow new ArithmeticException();
> ```

> 如果您在方法中會有例外的發生，而您並不想在方法中直接處理，而想要由呼叫方法的呼叫者來處理，則您可以使用 `throws` 關鍵字來宣告這個方法將會丟出例外，例如 `java.ioBufferedReader` 的 `readLine()` 方法就聲明會丟出 java.io.IOException。使用 `throws` 聲明丟出例外的時機，通常是工具類別的某個工具方法，因為作為被呼叫的工具，本身並不需要將處理例外的方式給定義下來，所以在方法上使用 `throws` 聲明會丟出例外，由呼叫者自行決定如何處理例外是比較合適的，您可以如下使用 `throws` 來丟出例外：

```
private void someMethod(int[] arr) throws
               ArrayIndexOutOfBoundsException,
               ArithmeticException {
    // 實作
}
```

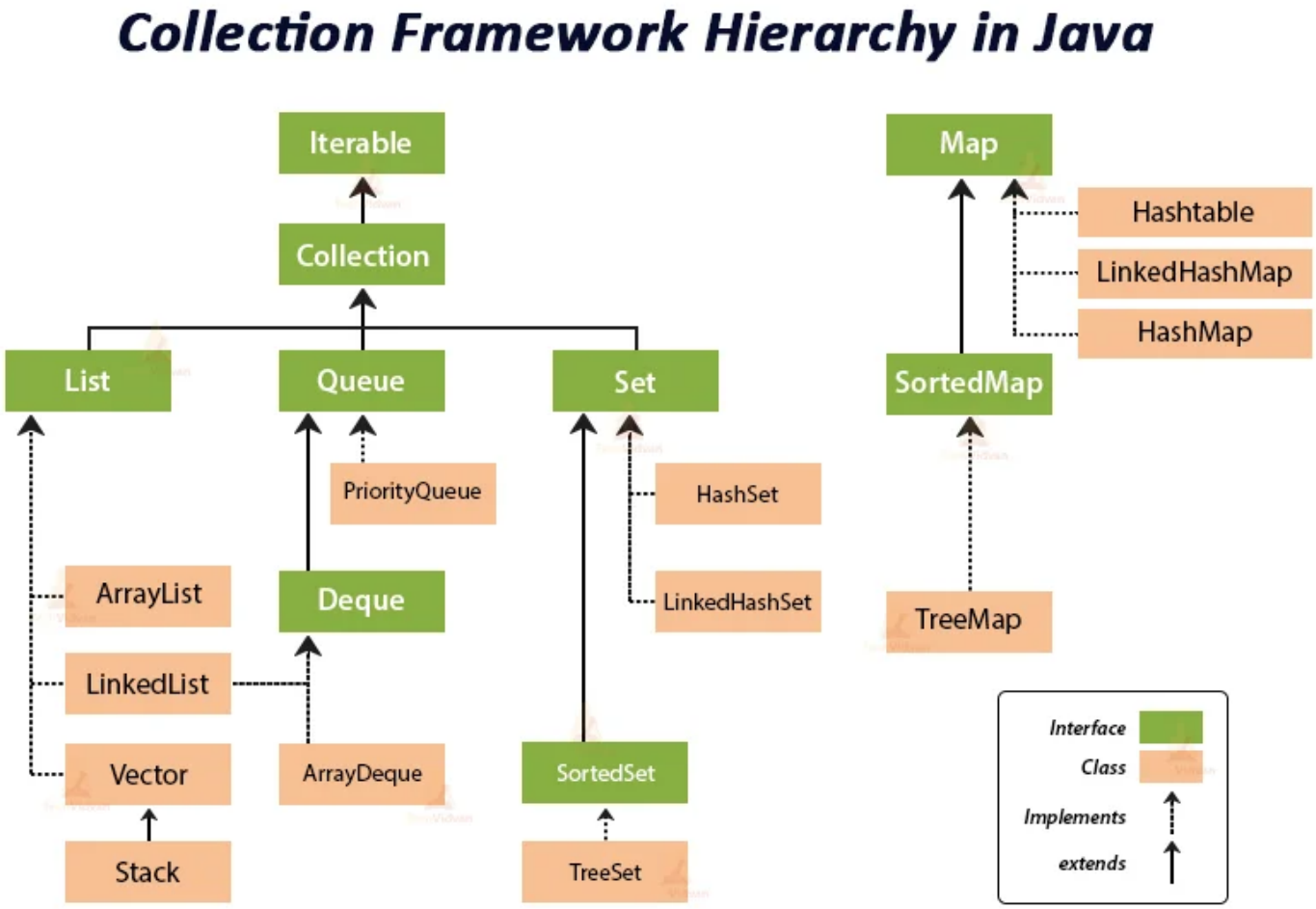
註：除非是受檢例外 `Checked Exception` 或是一些特別例外，不然 `Runtime Exception` 不應該 `throws` ，寫在 `javadoc` 裡面會比較好。有興趣可以參考這篇stackoverflow文章

`Assertions`

JDK 1.4加入，基本上不使用，與Junit的Assert是不一樣的東西。有興趣可以參考這篇stackoverflow文章

物件容器與泛型

基本上就是裝物件的容器，依照介面分為 `Collection` 與 `Map`



常用的介面有一些特行，請考下表

	List	Set	Map
重複物件	可以	不可以	不可以

	List	Set	Map
順序	依照插入順序	沒有順序	沒有順序
Nullable	可以加入Null	Null只能有一個	Key只能有一個， 但Value可以多個Null
實現	Array List, LinkedList.	HashSet, LinkedHashSet, and TreeSet.	HashMap, HashTable, TreeMap, ConcurrentHashMap, and LinkedHashMap.
用途	需要常用index存取	需要物件是唯一的	需要鍵值對的時候

詳細操作可以參考這篇教學。

泛型

因為教學篇幅的問題，這邊只提到怎麼使用以及大略概念。

> 在 J2SE 5.0 之後新增了泛型（Generic）的功能，使用物件容器時可以宣告將儲存的物件型態，如此您的物件在存入容器會被限定為您所宣告的型態，編譯器在編譯時期會協助您進行型態檢查，而取出物件時也不至於失去原來的型態資訊，這可以避免型態轉換時的問題。

語法

在容器宣告的後面寫上 `<>` ，裡面是類型名稱

```
List<String> list = new ArrayList<String>();
```

如果懶得化，也可以寫成這樣

```
List<String> list = new ArrayList<>();
```

Java 8 Lambda

參考：
https://www.cs.fsu.edu/~jtbauer/cis3931/tutorial/trailmap.html
https://docs.oracle.com/en/java/javase/17/
https://docs.oracle.com/javase/tutorial/java/javaOO/localclasses.html