PD2 Lecture 2 - Introduction to JAVA Programming - Foundation

Generated by DALL.E

Course Introduction

在本堂課程中，你可以學習到包含以下Java的基本語法：

1. **Java的第一個程式**：介紹如何撰寫並執行Java的第一個程式，包括Java基本程式組成元素（類別class、方法Method、語句Statement）和主函式（main method）的使用。

2. **變數Variable**：講解如何聲明和使用變數，包括變數的資料型別（整數、浮點數、字元、布林值等）、變數的命名規則和命名慣例，以及變數的初始化和賦值。
3. **表達式Expression**：介紹表達式的概念和使用，包括算術運算Operator（加減乘除、餘數等）、比較運算（大於、小於、等於等）、邏輯運算（and、or、not等）和遞增/遞減運算。
4. **流程控制Flow Control**：解釋if、switch、for和do-while等流程控制語句的使用，以及條件判斷和迴圈的應用場景。講解if-else語句的使用、switch語句的多重分支、for迴圈的使用和do-while迴圈的先執行後判斷特性。
5. **陣列Array**：介紹陣列的概念和使用，包括如何聲明和初始化陣列、access陣列元素、以及陣列的長度和iterate等操作。
6. **Java String：**Java的String類別的基本概念、創建方式、不可變性、操作和方法、其他資料型態的轉換的簡介。其中，包括了如何使用concat()、equals()、length()、substring()、toUpperCase()和toLowerCase()等方法，以及如何進行數值與String之間的轉換。

📖 **First Java Program**: An introduction on how to write and execute the first Java program, including basic elements of Java programming (classes, methods, statements) and the use of the main function (main method).

Variables: Explains how to declare and use variables, including data types of variables (integers, floating point numbers, characters, boolean values, etc.), variable naming rules and conventions, as well as variable initialization and assignment.

Expressions: Introduces the concept and use of expressions, including arithmetic operators (addition, subtraction, multiplication, division, remainder, etc.), comparison operators (greater than, less than, equal to, etc.), logical operators (and, or, not, etc.) and increment/decrement operators.

Flow Control: Explains the use of flow control statements such as if, switch, for, and do-while, as well as the application scenarios of conditional judgment and loops. It discusses the use of if-else statements, multiple branches of switch statements, the use of for loops, and the perform-first-judge-later characteristic of do-while loops.

Arrays: An introduction to the concept and use of arrays, including how to declare and initialize arrays, access array elements, and operations such as the length of arrays and iteration.

Java String: Basic concepts of Java's String class, methods of creation, immutability, operations and methods, and an introduction to conversions of other data types. This includes how to use methods like concat(), equals(), length(), substring(), toUpperCase(), and toLowerCase(), as well as how to convert between numbers and Strings.

解說HelloWorld

在Java中，我們可以撰寫並執行我們的第一個程式。下面是一個簡單的範例，介紹了Java基本程式組成元素和main function的使用：

```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello, World! My name is "+args
[0]); //count starting from 0
    }
}
```

這個程式由三個主要部分組成：

1. `public class HelloWorld`：這是我們的類別定義，使用關鍵字`class`來定義`類別名稱`為`HelloWorld`。類別名稱必須與檔案名稱相同，以確保程式能夠正確執行。
   a. You should name the file as ‘HelloWorld.java’
   b. Try to name as ‘helloworld.java’ or ‘hw.java’. What happen?
2. `public static void main(String[] args)`：這是我們的主函式（main method），是程式的進入點。它是程式的起點，程式從這裡開始執行。在這個範例中，我們只有一個簡單的statement。
3. `System.out.println("Hello, World!")`：這是一個輸出語句，使用`System.out.println`來輸出一個字串。在這個範例中，我們輸出了一個簡單的問候語句"Hello, World!"。

這個程式的執行結果是將"Hello, World!"輸出到控制台。 Try it!

📖 Three components:

1. `public class HelloWorld`: This is our class definition, using the keyword `class` to define the class name as `HelloWorld`. The class name must be the same as the file name to ensure the program runs correctly.
   a. You should name the file as 'HelloWorld.java'
   b. Try to name as 'helloworld.java' or 'hw.java'. What happen?
2. `public static void main(String[] args)`: This is our main function (main method), the entry point of the program. It is the starting point of the program, and the program starts executing from here. In this example, we only have a simple statement.
3. `System.out.println("Hello, World!")`: This is an output statement, using `System.out.println` to output a string. In this example, we output a simple greeting statement "Hello, World!".

The result of running this program is to output "Hello, World!" to the console.

📌 在Java中，檔案名稱必須與公共類別（public class）名稱`大小寫完全一致`，這是Java語言的規範。

主要原因：

1. **可讀性和一致性**：當開發者查看一個Java檔案時，他們可以直接從檔案名稱知道檔案中的public class名稱。這使得開發者更容易找到和理解程式碼。
2. **編譯和執行**：Java編譯器（javac）和JRE依賴於這個規範來定位和載入class。當你試圖編譯或執行一個Java檔案時，Java編譯器和JRE會期望檔案名稱與公共類別名稱一致。

因此，如果檔案名稱與公共類別名稱不一致，Java編譯器將無法正確地編譯該檔案，並可能導致編譯錯誤。

The main function in Java is the entry point of a program. When executing a Java program, the system automatically searches for and executes the main function with a specific signature. The main function is where the program starts its execution. In the main function, we can write the main logic of the program.

In Java, the signature of the main function must follow the following specification:

```
public static void main(String[] args) {
    // Code of the main function
}
```

- `public`: This is the access modifier of the main function, indicating that the main function is publicly accessible. This is necessary because the JVM needs to access the `main()` method from outside the program, which is the JVM itself, when executing the program.
- `static`: This is the modifier of the main function, indicating that it is a static method. Static methods can be called directly using the class name, without instantiating the class. This is also necessary because the JVM does not create any instances of classes when starting the program.
- `void`: This is the return type of the main function, indicating that it doesn't return any value.
- `main`: This is the name of the main function. The Java system automatically searches for and executes a method with this name.
- `String[] args`: This is the parameter list of the main function, which accepts an array of type String as a parameter. This parameter is often used to receive command-line arguments.

Command line parameters: The `args` parameter of the `main()` method is a string array that can accept arguments from the command line. For example, if you run a Java application from the command line, you can pass arguments like this:

```
java HelloWorld arg0 arg1... -> start from 0
```

The code inside the main function can be written as needed. In the main function, you can call other functions, declare and initialize variables, and

perform various operations.

In general, the `main()` method is the starting point of a Java program, and its existence is crucial for the program to be executed. Through this method, developers can control the execution flow of the program, and it is where simple output printing to complex application logic processing can begin.

Java的基本程式組成元素

Java的基本程式組成元素包括類別（class）、方法（method）和語句（statement）。

- 類別是Java程式的基本結構，包含了一組相關的member data和method。method是類別中的一個執行單元，用於執行特定的任務。語句是程式的基本組成部分，用於執行特定的操作。
- 主函式（main method）是Java程式的進入點。當我們執行一個Java程式時，系統會自動尋找並執行具有特定signature的主函式。主函式是程式的起點，程式從這裡開始執行。在主函式中，我們可以撰寫程式的啟始邏輯。
- `System.out.println`是Java的輸出語句(statement)，用於將內容輸出到控制台。在這個範例中，我們使用它來輸出一個問候語句"Hello, World!"。

Java Statement

Java的語句(Statement)是用來表達某一種操作的，可以包括聲明、賦值、方法呼叫、流程控制等。以下是一些相關的規則：

- 每個語句結尾必須有一個分號（;），代表該語句的結束。
- **聲明語句**：用來聲明一個變數或者常數。例如：

```
int myNumber;
```

- **賦值語句**：用來給一個變數賦值。例如：

```
myNumber = 5;
```

❓ 變數初始化：

在Java中，如果一個區域變數在使用前並未被賦值，會導致編譯錯誤。Java語言要求變數在使用前必須被初始化，以確保變數在任何情況下都有確定的值。如果一個變數未被賦值就嘗試去使用，Java編譯器將無法確定該變數的值，因此會拋出一個編譯錯誤，提示變數可能尚未被初始化。

- **Method呼叫語句**：用來呼叫一個方法。例如：

```
System.out.println("Hello, World!");
```

- **流程控制語句**：用來進行流程控制，例如條件語句、迴圈語句等。例如：

```
if (myNumber > 0) {
    System.out.println("The number is positive.");
} else {
    System.out.println("The number is negative or zero.");
}
```

- **迴圈語句**：迴圈語句用於重複執行某段程式碼，直到指定條件不成立為止。例如，`for`迴圈和`while`迴圈：

```
for(int i = 0; i < 10; i++){
    System.out.println(i);
}

int i = 0;
while(i < 10){
    System.out.println(i);
    i++;
}
```

- **Switch語句**：Switch語句用於根據某個變數的值選擇執行不同的操作：

```
int month = 2;
switch (month) {
    case 1:
        System.out.println("January");
        break;
```

```
    case 2:
        System.out.println("February");
        break;
    //...other months
    default:
        System.out.println("Invalid month");
}
```

> 💡 如果你沒有用 `break` statement會怎麼樣呢?
>
> 在Java的 `switch` 語句中，如果某個 `case` match成功後沒有遇到 `break` 指令，程序就會繼續執行下一個 `case` 的程式碼，而不會檢查下一個 `case` 的condition條件。這種行為稱為"穿透"（fall-through）。
>
> 在上面的程式中，如果去掉 `break` 指令，當 `month` 的值為2時，程序會先match到 `case 2:` 並執行 `System.out.println("February");` ，然後由於缺少 `break` ，它不會停止執行，而是繼續執行下一個 `case` 的程式碼，直到遇到一個 `break` 或者 `switch` 語句的結尾。
>
> 例如，如果我們移除 `case 2:` 後面的 `break` ，且程式中有對應3月到12月的 `case` 程式碼（假設每個月份都有類似的輸出和 `break` ），則當 `month` 為2時，它將會輸出"February"後繼續執行，如果沒有其他 `break` ，理論上會輸出所有月份直到遇到 `break` 或 `default` 結束。

- **Try-catch語句**：Try-catch語句用於處理可能會產生錯誤的程式碼區塊：

  在Java程式設計中，我們經常需要處理那些可能出現錯誤的程式碼區塊。這時，我們就需要用到try-catch語句。當程式執行try區塊中的程式碼時，如果出現異常，Java會立即停止當前的程式碼執行，並跳到相對應的catch區塊中去執行。這就是基本的錯誤處理機制。

```
try {
   BufferedReader reader = new BufferedReader(new FileReader
(args[0]));
   String line;
   while ((line = reader.readLine()) != null) {
       System.out.println(line);
   }
   reader.close();
} catch (IOException e) {
```

```
    e.printStackTrace();
}
```

在這個範例中，如果try區塊中的程式碼出現`IOException`，則會執行catch區塊的程式碼。

> 📌 後續我們會再特別說明Try-catch用法，現階段你可以依IDE告知你那些Exception要處理來撰寫你的程式。

- **Return語句**：Return語句用於指定一個方法返回的結果：

```
public int add(int a, int b) {
    return a + b;
}
```

在這個範例中，`add`方法將兩個整數相加，然後返回結果。

- **Break與Continue語句**：`break`與`continue`語句用於控制迴圈的執行。`break`會立即結束迴圈，而`continue`則會跳過當前迴圈剩餘的部分，直接進行下一次迴圈。例如：

```
for (int i = 0; i < 10; i++) {
    if (i == 5) {
        break; // 當i等於5時，結束迴圈
    }
    System.out.println(i);
}

for (int i = 0; i < 10; i++) {
    if (i % 2 == 0) {
        continue; // 當i為偶數時，跳過迴圈的剩餘部分
    }
    System.out.println(i);
}
```

- **運算表達式**：運算表達式可以是一個值，一個變數，或者一個運算符號與兩個運算元的組合。例如：

```
int a = 5; // a是一個運算表達式
int b = a + 1; // a + 1是一個運算表達式
```

- **Block語句**：Block語句是由一對大括號`{}`包圍的一或多個語句。這通常用於創建一個statement set，例如在`if`、`for`和`while`等語句中（但不一定要是這些statement才可以create Block)：

```
if (myNumber > 0) {
    System.out.println("The number is positive.");
    myNumber--;
}
```

在這個例子中，`if`語句中的兩行程式碼就被包含在一個statement block中。

- **空語句**：空語句是一個沒有任何效果的語句，它只包含一個分號`;`。這種語句在某些情況下可以用來作為語句的Placeholder，例如在迴圈中：

```
for (int i = 0; i < 10; i++); // 空語句，迴圈不執行任何操作
```

💡 **Placeholder:**

在程式設計中，Placeholder通常指的是在程式碼中暫時代替某個元素或值的符號或變數。它在我們還不確定要放什麼，或者在我們需要先留下一個位置以便之後再填入實際內容時，會非常有用。

在Java中，我們可以使用一個未初始化的變數作為Placeholder。例如：

```
int placeholder; // 宣告一個變數作為Placeholder

// 在我們知道實際值之前，我們可以在程式碼中使用這個變數
for (int i = 0; i < 10; i++) {
    placeholder = i; // 可以先將placeholder assign一
個值
    System.out.println(placeholder); // 使用placehol
der
}
```

在這個例子中，`placeholder`就是一個Placeholder，我們先在程式碼中預留了一個位置給它，然後在迴圈中給它賦值，最後再使用它。

所以，Java中的空語句（只有一個分號`;`的語句）也可以看作是一種Placeholder。例如在下面的迴圈中：

```
for (int i = 0; i < 10; i++); // 空語句，迴圈不執行任何
操作
```

這裡的空語句就是一個Placeholder，代表迴圈裡面可以執行的操作。雖然它現在沒有做任何事情，但我們可以在之後需要的時候，將它替換為實際的操作。

Java語法基本要求

- **檔案名稱和類別名稱**：在Java中，檔案名稱必須與類別名稱相同，並且以`.java`作為副檔名。類別名稱的首字母通常使用大寫，並遵循駝峰式大小寫命名規則。例如，一個名為`MyClass`的類別，對應的檔案名稱應為`MyClass.java`。
- **關鍵字**：Java有一組保留的關鍵字，這些關鍵字不能用作變數名稱或其他識別符號。這些關鍵字是用於特定目的的，例如定義類別、方法、條件語句等。關鍵字通常使用小寫。

- **變數和方法名稱**：Java中的變數和方法名稱通常使用Camel case大小寫命名規則。即使變數名稱只有一個單詞，也推薦使用Camel case駝峰式命名。例如，`myVariable` 或 `myMethod`。
- **大小寫敏感**：Java是區分大小寫的。這意味著 `myVariable` 和 `myvariable` 被視為不同的變數。因此，在Java中使用正確的大小寫非常重要。 — Same as C++ and Python

📎 Java has some basic differences in English capitalization, indentation, etc. compared to other programming languages. Here are a few key differences:

- **File names and class names**: In Java, the file name must be the same as the class name and have the `.java` file extension. The first letter of the class name is usually capitalized and follows the camel case naming convention. For example, a class named `MyClass` should have a corresponding file name of `MyClass.java`.
- **Keywords**: Java has a set of reserved keywords that cannot be used as variable names or other identifiers. These keywords are used for specific purposes, such as defining classes, methods, conditional statements, etc. Keywords are typically written in lowercase.
- **Variable and method names**: Variable and method names in Java usually follow the camel case naming convention. Even if the variable name consists of only one word, it is recommended to use camel case. For example, `myVariable` or `myMethod`.
- **Case sensitivity**: Java is case sensitive. This means that `myVariable` and `myvariable` are considered as different variables. Therefore, using the correct capitalization is crucial in Java. — Same as C++ and Python

Indention Requirement - It is not necessary but is important

在Java中，縮排不是強制要求的，但它在程式碼的可讀性方面非常重要。Java使用大括號 `{}` 來定義程式碼區塊，而縮排則是一種常見的編碼風格，用於使程式碼區塊更容易閱讀和理解。儘管Java不會強制要求縮排，但建議遵循一致的縮排風格，以提高程式碼的可讀性。通常，Java社群遵循類似於以下的縮排風格：

📖 In Java, indentation is not a mandatory requirement, but it is very important for the readability of the code. Java uses the braces `{}` to define code blocks, and indentation is a common coding style used to make code blocks easier to read and understand. Although Java does not enforce indentation, it is recommended to follow a consistent indentation style to enhance the readability of the code. Typically, the Java community follows an indentation style similar to the following:

```
public class Example {
    public static void main(String[] args) {
        if (condition1) {
            // 單一縮排
            // 更多程式碼
        } else if (condition2) {
            // 單一縮排
            // 更多程式碼
        } else {
            // 單一縮排
            // 更多程式碼
        }

        for (int i = 0; i < 5; i++) {
            // 單一縮排
            // 更多程式碼
        }
    }
}
```

這種縮排風格不是強制性的，但它幫助提高程式碼的可讀性，讓程式開發者更容易理解程式碼的結構和邏輯。當然，您和您的團隊可以根據自己的偏好選擇不同的縮排風格，但一致性在整個項目中很重要。

📖 This indentation style is not mandatory, but it helps to improve the readability of the code, making it easier for developers to understand the structure and logic of the code. Of course, you and your team can choose different indentation styles according to your own preferences, but consistency is important throughout the project.

Java Coding Style

以下是Java編碼標準和最佳實踐的詳細說明，包括命名慣例、縮排和對齊、注釋慣例、語句結構和錯誤處理的指南，每項都附有程式範例。

命名慣例

1. 類別和介面命名 - Class and Interface

- **Class與Interface名**應該使用大寫開頭的駝峰式命名法（CamelCase）。

Class names and **interface names** should use CamelCase naming convention starting with a capital letter.

```
public class Car {
}

public interface Vehicle {
}
```

2. Method方法和Variable變數命名

- **方法**和**變數**應該使用小寫開頭的駝峰式命名法。

```
public class Calculator {
    private int result;

    public void addNumbers(int number1, int number2) {
        this.result = number1 + number2;
    }
}
```

3. Constant常數命名

- **常數constant**應全部大寫，單詞之間用下劃線分隔。

```
public class Constants {
    public static final String COMPANY_NAME = "Tech Innovatio
ns";
}
```

縮排和對齊

1. 縮排使用4個空格

- 在Java中建議避免使用Tab，改用空格進行縮排。

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

2. 換行和對齊

- 當一行程式碼過長時，應該換行並適當對齊以提高可讀性。

```
public class Example {
    public void longMethodWithLotsOfParameters(String paramet
erOne,
                                               int parameterT
wo,
                                               boolean parame
terThree,
                                               double paramet
erFour) {
        // 方法內容
    }
}
```

注釋慣例

1. 多行和單行注釋

- 使用`//`進行單行注釋，使用`/* ... */`進行多行注釋。

```
// 這是一個單行注釋
/*
這是一個多行注釋
可以跨越多行。
*/
```

2. 文檔注釋

- 使用`/** ... */`進行文檔注釋，適用於類別、方法和變數，以生成JavaDoc。

```
/**
 * 這是一個計算器類。
 */
public class Calculator {
    /**
     * 將兩個數字相加。
     * @param number1 第一個數字
     * @param number2 第二個數字
     * @return 兩數之和
     */
    public int add(int number1, int number2) {
        return number1 + number2;
    }
}
```

語句結構慣例

1. 避免過長的方法

- 每個方法應該只做一件事情，並保持精簡 - 一個Method超過200行不見得是好事。

  Each method should only do one thing and keep it lean - it's not necessarily a good thing for a method to exceed 200 lines.

```
public class Utility {
    public void performAction() {
        // 做某事
    }

    public void performAnotherAction() {
        // 做另一件事
    }
```

2. 使用空行區分程式碼區塊

- 使用空行來分隔程式Block，增加可讀性。

```
public void multipleTasks() {
    // 第一個任務
    doFirstThing();
                        //有一行空行
    // 第二個任務
    doSecondThing();
}
```

錯誤處理

1. 異常處理

- 適當地捕獲和處理異常，避免使用過於通用的異常捕獲。

```
try {
    //

 可能拋出異常的程式碼
} catch (SpecificExcepti
on ex) {
    // 處理特定異常
} catch (AnotherExceptio
n ex) {
    // 處理另一種特定異常
}
```

💡 Java Exception將在Java OOP內容再詳細說明

以上是基於Java編碼標準和最佳實踐的概覽，這些慣例有助於維持程式碼的清晰性、一致性和可維護性。不同的項目或團隊可能有自己的特定標準，因此在實際應用中應當遵循相關指導原則。

Good Reference:

- https://www.cs.cornell.edu/courses/JavaAndDS/JavaStyle.html
- https://google.github.io/styleguide/javaguide.html

Java變數介紹

在Java程式設計中，變數的使用佔了很重要的一部分，它是在程式中儲存和操作資訊的基本方式。以下是關於Java變數的一些基本說明：

變數聲明與使用：在Java中，我們必須先聲明變數才能使用它。變數的聲明包含了變數的類型和名稱，例如：

```
int number;
```

在這裡，`int`是變數的類型，`number`是變數的名稱。

資料型別：Java支持多種基本資料型別(Primitive Data Type)，包括整數（如`int`、`byte`、`short`、`long`）、浮點數（如`float`、`double`）、字元（`char`）和布林值（`boolean`）等。每種資料型別都有其特定的大小和值範圍。選擇適當的資料型別可以幫助節省記憶體空間，並提高程式的效率。

變數命名規則與慣例：變數名稱應該以小寫字母開始，並遵守駝峰式命名法。變數名稱可以包含字母、數字和下劃線（_），但不能以數字開頭。變數名稱應該具有描述性，以便其他讀者理解其用途。例如，`int studentCount;`和`double averageScore;`，這些變數名稱就很直觀地反映了他們的用途。

變數的初始化與賦值：在聲明變數後，我們可以使用等號（`=`）來初始化或賦值給變數。例如：

```
int count = 10;
count = 20;
```

在上面的程式碼中，我們先聲明了一個`int`型別的變數`count`，並初始化它的值為10。然後，我們將`count`的值更改為20。這兩步操作都是將資料賦值給變數，只不過一個是在聲明變數的同時進行（稱為初始化），另一個是在變數聲明後進行。

變數初始預設議題

Class Member

成員變數如果沒有明確初始化，Java會為其提供一個預設值。這個預設值依賴於變數的數據類型：

- 數字型（整數和浮點數）的預設值為`0`或`0.0`。
- 布爾型的預設值為`false`。
- Reference類型（例如，Object、Array）的預設值為`null`。

```
public class MyClass {
    int myInt; // 預設值為0
```

```
    double myDouble; // 預設值為0.0
    boolean myBool; // 預設值為false
    String myString; // 預設值為null

    public void printValues() {
        System.out.println("Int: " + myInt);
        System.out.println("Double: " + myDouble);
        System.out.println("Boolean: " + myBool);
        System.out.println("String: " + myString);
    }

    public static void main(String[] args) {
        MyClass obj = new MyClass();
        obj.printValues();
    }
}
```

局部變數Local Variable

對於局部變數，Java要求在使用前必須明確初始化。如果嘗試使用未初始化的局部變數，會導致compiler error。

```
public class Test {
    public void myMethod() {
        int myLocalInt; // 局部變數未初始化
        // System.out.println(myLocalInt); // 這行程式碼會導致編譯錯誤，因為局部變數在使用前必須初始化
    }
```

在上面的例子中，如果對`myLocalInt`取消註解，會導致compiler error，提示變數`myLocalInt`可能尚未初始化。

> 📎 總結：Java中的成員變數會有預設值，而局部變數則必須在使用前明確賦值。這是為了避免程序在運行時出現不確定的狀態。

除了Primitive Data Type基本資料型別之外，Java還支援類別（Class）和介面（Interface），這些都可以用來宣告變數。例如，我們可以建立一個

> 💡 在OOP中將詳細介紹

Class `Student`，然後使用這個Class來宣告變數，如 `Student aStudent;`。這種變數就是物件（Object）。

保留字

Java保留字（關鍵字）是在Java語言中有特殊含義的單詞，不能用作識別identifier（例如，變量名、函數名或類名）。以下是Java保留字的列表：

abstract	assert	boolean	break	byte
case	catch	char	class	const
continue	default	do	double	else
enum	extends	final	finally	float
for	goto	if	implements	import
instanceof	int	interface	long	native
new	package	private	protected	public
return	short	static	strictfp	super
switch	synchronized	this	throw	throws
transient	try	void	volatile	while

注意：

- `goto`和`const`是保留字，但在Java中實際上並未使用。
- Java 5引入了`enum`作為保留字。
- 這份列表不包括Java 5之後新增的關鍵字，如`var`（Java 10中引入作為局部變量類型推斷的保留類型名稱）。
- `true`，`false`，`null`是Literal字，均不能做為變數使用

這些保留字涵蓋了Java語言的基本結構和概念，從數據類型到控制結構，再到異常處理等等。在編寫Java程式碼時，開發者需要避免使用這些保留字作為任何形式的identifier。

Unicode支援

Java支援Unicode用作識別符號（例如，變量名、函數名或類名）。以下是一個使用Unicode作為變量名的範例：

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

```
        int 變量 = 5;
        System.out.println(變量);
    }
}
```

📎 在上面的程式碼中，"變量" 是一個有效的變量名，當你運行這個程序，它將輸出數字5。然而，請注意，雖然這是語法上允許的，但在實際的編碼實踐中，使用非英文字母作為識別符號可能會導致可讀性和相容性問題，因此並不推薦。

常數Constant

在Java中，我們可以使用 `final` 關鍵字來定義一個常量，這表示該變數的值一旦被賦值後就不能再改變。以下是如何定義和使用常量的範例：

```
public class Test {
    public static void main(String[] args) {
        final int MAX_COUNT = 10;
        System.out.println(MAX_COUNT);
        // MAX_COUNT = 20; // 這行程式碼會導致編譯錯誤，因為我們不能改變一個常量的值
    }
}
```

在上面的程式碼中，我們定義了一個常量 `MAX_COUNT`，並給它賦值為10。由於 `MAX_COUNT` 是一個常量，所以我們不能再次給它賦值，否則會導致編譯錯誤。

📎 常數在程式設計中非常有用。例如，我們可以使用常數來表示一些固定的值，如數學中的圓周率、物理中的重力加速度等。此外，使用常數也可以提高程式的可讀性和維護性。當我們需要改變一個值時，只需改變常數的定義，而不需要在整個程式中尋找和修改該值。

💡 **觀念Recap**

static修飾字

- **static**: 表示class-level的變數，而不是instance-level的。這意味著該變數屬於class本身，所有instances將共享同一個variable。
  - Static variables can be accessed directly through the class name, without the need to create an instance of the class.

final修飾字

- **final**: 表示常數，一旦被賦值後，其值就不能被修改。final修飾符可以用於變數、Method和class。當用於變數時，該變數成為常數；用於Method時，該Method不能被extended class覆寫；用於class時，該class別不能被繼承。

Java 變數 Data Type

Primitive Data Type

Java擁有八種基本資料型別（Primitive Data Types），如下所示：

資料型態	描述	記憶體空間
byte	Java的8位元有符號整數，範圍從-128到127	1位元組(位元組=byte=8bits)
short	Java的16位元有符號整數，範圍從-32768到32767	2位元組
int	Java的32位元有符號整數，範圍大約從-2的31次方（-2147483648）到2的31次方減1（2147483647）	4位元組
long	Java的64位元有符號整數，範圍大約從-2的63次方到2的63次方減1	8位元組
float	Java的32位元單精度浮點數	4位元組
double	Java的64位元雙精度浮點數	8位元組
boolean	Java的布林型態，只有兩個可能的值：`true` 或 `false`	不在Language規範boolean的實際記憶體大小，而是依賴於JVM實現以及可能的情境（如

		陣列、物件欄位等）。通常，一個 `boolean` 變數本身在記憶體中的大小可能超過1位元組，因為JVM可能會為了效能或對齊的原因採用更大的儲存單位。
char	Java的16位元Unicode字元，範圍從U+0000到U+FFFF	2位元組

📎 這些基本資料型別的大小都是固定的，不論在任何操作系統或平台上，都不會變更。這是Java語言“一次寫作，處處運行”的一部分。

Reference Data Type

在Java中，Reference Data Type（參考資料型態）是指非原始（primitive）資料型別的變數，包括類別（classes）、陣列（arrays）、介面（interfaces）和列舉（enums）。不同於原始資料型態直接儲存值，參考資料型態變數儲存的是對記憶體中物件或陣列的引用（即記憶體地址）。當你操作這些變數時，實際上是在操作物件的參考，而非物件本身。

💡 將於上到Object Oriented Programming內容時，會再詳細說明

範例

假設我們有一個 `Book` 類別，如下所示：

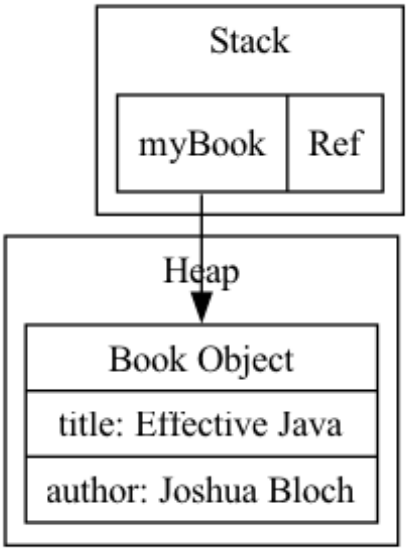
```
class Book {
    String title;
    String author;

    Book(String title, String author) {
        this.title = title;
        this.author = author;
    }
}
```

接下來，我們創建一個 `Book` 類別的實例：

```
Book myBook = new Book("Effective Java", "Joshua Bloch");
```

在這個例子中，`myBook`是一個參考資料型態(reference)的變數，它存儲了對`Book`實例的引用。這個object instance被儲存在JVM的（heap）記憶體區域，而`myBook`變數本身儲存在（stack）記憶體區域。

這個圖表展示變數`myBook`（位於stack）如何通過參考reference指向heap上的`Book`物件instance。



JVM Memory Management: Heap and Stack

在 Java 的記憶體管理機制中，堆積（Heap）和堆疊（Stack）是用於儲存資料的兩個主要區域，它們各自有不同的用途和管理方式：

- **堆積（Heap）**：用於存放 Java program中創建的物件（Object）和類別（Class）實例instance。無論是哪個部分創建的物件，這些物件都在heap中配置記憶體。heap的記憶體release是由Java 的GC自動處理的。
- **堆疊（Stack）**：用於存放基本類型的變數（如 int、short、long、byte、float、double、boolean、char）和物件reference變數（物件的參考地址），這些變數被定義在方法中。

Example

以下是一個簡單的 Java 程序示例，說明哪些數據會被放在heap和stack中：

```
public class MemoryExample {
    public static void main(String[] args) {
        int number = 10; // Stack
        String text = "Hello World"; // Heap (text is a refer
ence stored in Stack)
        MyClass myObject = new MyClass(); // Heap (myObject i
s a reference stored in Stack)

        calculate(number); // 'number' is passed by value to
the method
    }

    public static void calculate(int num) {
        int result = num * 2; // Stack
        System.out.println("Result: " + result);
    }
}

class MyClass {
    int value; // This instance variable is stored in Heap as
part of MyClass object
}
```

💡 在這個例子中：

- 基本類型變數 `number` 和 `calculate` 方法中的 `num` 及 `result` 都是在stack中，因為它們是Method中的local變數。
- 字串 `text` 和物件 `myObject` 的reference參考（地址）是在stack中存儲的，但實際的物件內容（即 String 物件和 MyClass 的實例）是在heap中分配的。
- `MyClass` 中的member data `value` 會隨著 `MyClass` 的物件一起存儲在heap中。

Java運算式Expression介紹

在Java中，表達式（Expression）是由變數、運算Operator和Method組合而成的構造，該構造能夠產生一個結果值。以下介紹幾種常用的運算Operator：

- **算術運算**Operator：這些運算Operator用於實現基本的數學運算：+;-;*;/;%

```
int a = 10;
int b = 20;
int sum = a + b; // 30
int diff = a - b; // -10
int product = a * b; // 200
int quotient = b / a; // 2
int remainder = b % a; // 0
```

- **比較運算**Operator：這些運算Operator用於比較兩個值。例如，`>`（大於）、`<`（小於）、`==`（等於）、`!=`（不等於）、`>=`（大於等於）和`<=`（小於等於）等。

```
int a = 10;
int b = 20;
boolean result1 = a > b; // false
boolean result2 = a < b; // true
boolean result3 = a == b; // false
boolean result4 = a != b; // true
boolean result5 = a >= b; // false
boolean result6 = a <= b; // tru
```

- **邏輯運算**Operator：這些運算Operator用於實現邏輯運算。例如，`&&`（與）、`||`（或）和`!`（非）等。

```
boolean a = true;
boolean b = false;
boolean result1 = a && b; // false
boolean result2 = a || b; // true
boolean result3 = !a; // false
```

- **遞增/遞減運算**Operator：這些運算Operator用於對變數進行遞增（`++`）或遞減（`--`）操作。

```
int a = 10;
a++; // a的值現在是11
a--; // a的值現在又回到了10
```

這些運算Operator提供了Java中基本運算的實現方式，並且它們可以在各種情況下組合使用，以實現更複雜的計算和邏輯判斷。

Operator Precedence

在Java中，運算子的優先順序是一個重要的概念，它決定了在一個包含多個運算子的表達式中，哪個運算子會先被執行。如果你沒有正確理解和應用這個規則，可能會導致程式的結果與你預期的不同。

在Java中，運算子的優先順序如下（由高到低）：

1. 圓括號`()`：最高的優先級，會首先執行圓括號內的運算。
2. 單元運算子，如`++`，`-`，`!`：例如，`++`和`-`運算子用於增加或減少變數的值，`!`運算子用於對布林值進行否定操作。
3. 算術運算子，如`*`，`/`，`%`，`+`，`-`：這些運算子用於進行基本的數學運算，如乘法、除法、取餘數、加法和減法。
4. 比較運算子，如`<`，`<=`，`>`，`>=`，`==`，`!=`：這些運算子用於進行比較操作，例如，檢查一個數是否大於、小於、等於或不等於另一個數。
5. 邏輯運算子，如`&&`，`||`：邏輯運算子用於進行邏輯運算，例如，檢查兩個布林表達式是否都為真（和），或者至少有一個為真（或）。
6. 條件運算子，如`?:`：這是Java的三元運算子，它允許在一個表達式中進行條件選擇。
7. 賦值運算子，如`=`，`+=`，`-=`，`*=`，`/=`，`%=`：這些運算子用於給變數賦值。

以下是一個運算子優先順序的範例：

```
int result = 1 + 2 * 3; // 結果是7，而不是9，因為*的優先順序高於+
```

這裡，由於乘法運算子`*`的優先級高於加法運算子`+`，所以`2 * 3`會先被計算，然後再加上1。

還有一些常見的易犯錯誤的範例，如下：

```
boolean result = 3 + 2 < 5 && 10 / 2 > 3; // 結果是false，因為算術運算子的優先順序高於比較運算子，比較運算子的優先順序高於邏輯運算子
```

在這個例子中，算術運算（`3 + 2`和`10 / 2`）首先被執行，然後才是比較運算（`<`和`>`），最後執行邏輯運算`&&`。所以，原表達式相當於`((3 + 2) < 5) && ((10 / 2) > 3)`，即`5 < 5 && 5 > 3`，因此結果為`false && true`，即`false`。

為了避免由於優先順序引起的混淆和錯誤，我們應該在表達式中使用括號來明確指定運算的順序。例如，如果我們想先進行加法運算，然後再進行乘法運算，我們可以寫成 `(1 + 2) * 3`。這種方法不僅可以避免錯誤，而且可以使表達式更容易理解。

Data Type Conversion

在Java中，有兩種主要的類型轉換方法：

1. **自動類型轉換（Automatic Type Conversion）**：這種類型轉換發生在當一個資料類型的變數被賦予另一種資料類型的值時，並且當目標資料類型大於來源資料類型時。這種轉換被稱為自動類型提升(Automatic Type Promotion)。例如：

```
int myInt = 150;
long myLong = myInt;  // 自動類型轉換從int到long
```

1. **強制類型轉換（Casting）**：當我們需要從一種更大的資料類型轉換到一種更小的資料類型，或者從非相容的資料類型轉換時，我們需要使用強制類型轉換。這種轉換需要在變數前加上目標類型。例如：

```
double myDouble = 25.6;
int myInt = (int) myDouble;  // 強制類型轉換從double到int
```

```
public class Main {
  public static void main(String[] args) {
    double myDouble = 9.78d; //在這裡，後綴 d 表示這是一個 double 型
                             //在Java中，浮點數默認是 double 型別，
                             //指定這是一個 double 型別的數字，而不是
                             //9.78f就是後綴float
    int myInt = (int) myDouble; // Explicit casting: double to i

    System.out.println(myDouble);
    System.out.println(myInt);
  }
}
```

這裡的一個常見錯誤是假設強制類型轉換會四捨五入，但實際上，強制類型轉換會直接去除小數點後的所有數字。例如，上述的強制類型轉換結果為 `25`，而不是 `26`。

另一個常見的錯誤是在沒有足夠的空間存儲原始值時進行強制類型轉換。例如：

```
int myInt = 300;
byte myByte = (byte) myInt;  // 強制類型轉換從int到byte
```

這裡，`myInt` 的值 `300` 超過了 `byte` 類型的最大值 `127`。結果，`myByte` 的值將會是 `44`，這是通過將 `300` 除以 `256`（`byte` 類型的範圍）並取得的餘數。這稱為整數溢出(Integer Overflow)，通常會導致錯誤的結果。

其它容易犯錯的Type Conversion情況

- **整數除法導致的精度問題：**
  當使用整數進行除法運算時，結果也會是整數，這可能導致初學者混淆。

```
int i1 = 5;
int i2 = 2;
double result = i1 / i2; // 這裡先進行整數除法，結果為2，然後將2轉換為double
System.out.println("Result: " + result); // 輸出：2.0，而不是預期的2.5
```

為了得到正確的結果，至少有一個操作數需要被顯式或隱式地轉換為浮點型：

```
double correctResult = (double)i1 / i2; // 強制將i1轉換為double後進行除法
System.out.println("Correct Result: " + correctResult); // 正確輸出：2.5
```

- **字串與基本數據類型間的轉換錯誤**

在將字串轉換為基本數據類型（如 `int`、`double` 等）時，如果字串不能正確解析為目標類型的值，將會拋出 `NumberFormatException`。

範例：

```
String s = "100";
int i = Integer.parseInt(s); // 正確轉換
System.out.println("Int value: " + i); // 輸出：100

String s2 = "100.5";
// int j = Integer.parseInt(s2); // 這行將拋出NumberFormatException，因為"100.5"不能解析為int
```

```
double d = Double.parseDouble(s2); // 正確的轉換方式
System.out.println("Double value: " + d); // 輸出：100.5
```

💡 In summary,

there are two types of casting in Java:

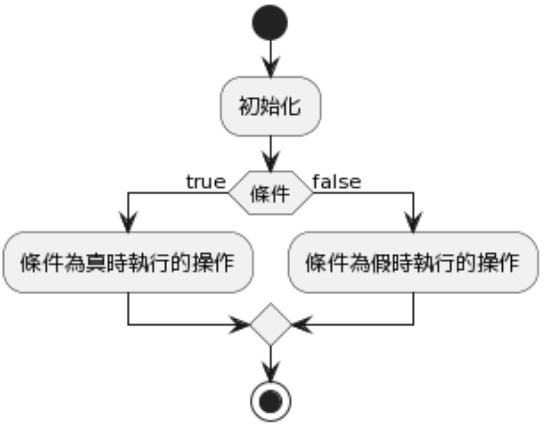
- **Widening Casting** (automatically) - converting a smaller type to a larger type size `byte` → `short` → `char` → `int` → `long` → `float` -> `double`
- **Narrowing Casting** (manually) - converting a larger type to a smaller size type `double` → `float` → `long` → `int` → `char` → `short` → `byte`

Java流程控制

Java的流程控制語句是用來根據特定條件來控制程序執行流程的。它們是任何程式設計語言中不可或缺的部分，使得程式能夠根據不同的條件執行不同的程式碼區塊，或者重複執行某段程式碼直到滿足特定條件。Java中的主要流程控制結構包括條件判斷（if-else和switch）和迴圈（for、while和do-while）。

if-else語句

if-else語句用於基於條件表達式的結果來執行不同的程式碼區塊。如果條件表達式為true，則執行if後面的程式碼區塊；如果為false，則執行else後面的程式碼區塊（如果有的話）。



範例：

```
int score = 75;
if (score >= 60) {
    System.out.println("Passed");
} else {
```

```
    System.out.println("Failed");
}
```

switch語句

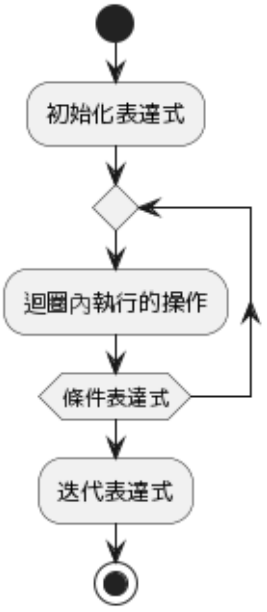
switch語句用於基於一個表達式的值來進行多重分支的選擇。它與一系列的case語句配合使用，每個case後面跟著一個要匹配的值和一段程式碼。

範例：

```
int month = 8;
switch (month) {
    case 1:
        System.out.println("January");
        break;
    case 2:
        System.out.println("February");
        break;
    // 更多月份...
    default:
        System.out.println("Invalid month");
}
```

for迴圈

for迴圈用於當你需要執行一段程式碼特定次數時。它將初始化表達式、條件表達式和迭代表達式組合在一起。



範例：

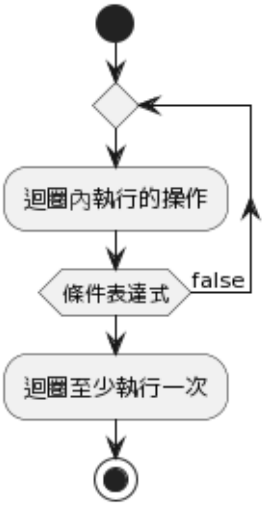
```
for (int i = 1; i <= 5; i++) {
    System.out.println("Value of i: " + i);
}
```

do-while迴圈

do-while迴圈至少執行一次程式碼區塊，然後再根據條件表達式的結果決定是否繼續執行。它是一種"先執行後判斷"的迴圈結構。

範例：

```
int count = 1;
do {
    System.out.println("Cou
nt is: " + count);
    count++;
} while (count <= 5);
```



Break and Continue用法

在 Java 中，有兩種語句可以用來更改循環的流程：`break` 和 `continue`。

- **`break` 語句**：`break` 語句用於立即結束當前正在運行的循環，並繼續執行循環之後的程式碼。在 `for`，`while` 和 `do-while` 循環中都可以使用 `break` 語句。`break` 語句也可以用於在 `switch` 語句中結束一個 `case`。

以下是使用 `break` 語句的一個例子：

```
for (int i = 1; i <= 10; i++) {
    if (i == 6) {
        break; // 當 i 等於 6 時，結束循環
    }
```

```
    System.out.println(i); // 印出 1 到 5
}
```

- **`continue` 語句**：`continue` 語句用於跳過當前正在運行的循環的剩餘部分，並立即開始下一次循環。`continue` 語句可以用於 `for`，`while` 和 `do-while` 循環。

以下是使用 `continue` 語句的一個例子：

```
for (int i = 1; i <= 10; i++) {
    if (i == 6) {
        continue; // 當 i 等於 6 時，跳過循環的剩餘部分
    }
    System.out.println(i); // 印出 1 到 10，但不包括 6
}
```

在這個例子中，當 `i` 等於6時，`continue` 語句將跳過 `System.out.println(i)`，並立即開始下一個循環。

> 📖 請注意，不適當的使用 `break` 和 `continue` 語句可能會使你的程式變得難以理解和維護。因此，除非必要，否則應該儘量避免使用它們。

Do Exercise

- 請判斷是否輸入的三邊長是否能組成三角形

```
import java.io.*;

public class Deltaic {

  public static void main(String[] argv)
        throws IOException { //將IOException往外丟

    BufferedReader br = new
        BufferedReader(new InputStreamReader(System.in));

    int i,j,k;

    // 輸入第 1 邊邊長
    System.out.println("請輸入三角形的三邊長：");
    System.out.print("邊長 1 →");
```

```
    String str = br.readLine();
    i = Integer.parseInt(str);

    // 輸入第 2 邊邊長
    System.out.print("邊長 2 →");
    str = br.readLine();
    j = Integer.parseInt(str);

    // 輸入第 3 邊邊長
    System.out.print("邊長 3 →");
    str = br.readLine();
    k = Integer.parseInt(str);

    if ((i+j) > k)     // 判斷第 1, 2 邊的和是否大於第 3 邊
      if ((i+k) > j)   // 判斷第 1, 3 邊的和是否大於第 2 邊
        if ((j+k) > i) // 判斷第 2, 3 邊的和是否大於第 1 邊
          System.out.println("可以為三角形的三邊長。");
        else
          System.out.println("第 2、3邊的和應大於第 1 邊");
      else
        System.out.println("第 1、3邊的和應大於第 2 邊");
    else
      System.out.println("第 1、2邊的和應大於第 3 邊");
  }
}
```

Java 陣列(Array)

在Java中，陣列是一種資料結構，可以存儲多個同類型的值。它是一種靜態的資料結構，也就是說，您在聲明陣列時必須指定其大小，並且一旦創建，其大小就不能改變。

以下是一些關於如何在Java中使用陣列的基本概念：

- **聲明陣列**：您可以使用以下語法聲明陣列：

```
dataType[] arrayName;  // 推薦
dataType arrayName[];  // 也可以，但不推薦
```

例如，要聲明一個可以存儲整數的陣列，您可以這麼寫：

```
int[] myArray;
```

- **初始化陣列**：聲明陣列後，您需要使用`new`關鍵字來創建陣列。此時，您需要指定陣列的大小。例如：

```
myArray = new int[10];
```

您也可以在聲明陣列的時候就初始化它：

```
int[] myArray = new int[10];
```

或者，您可以在初始化陣列的時候就指定它的元素：

```
int[] myArray = {1, 2, 3, 4, 5};
```

- **訪問陣列元素**：您可以使用陣列index來訪問陣列的元素。陣列索引是從0開始的。例如，要獲取上面陣列的第一個元素，您可以這麼寫：

```
int firstElement = myArray[0];
```

要修改陣列的元素，您也可以使用陣列index。例如，要把陣列的第一個元素改為10，您可以這麼寫：

```
myArray[0] = 10;
```

- **陣列的長度**：您可以使用`length`屬性來獲取陣列的長度（即它包含的元素數量）。例如：

```
int length = myArray.length;
```

- Iterate**陣列**：您可以使用for迴圈或者for-each迴圈來迭代陣列的元素。例如：

```
// 使用for迴圈
for (int i = 0; i < myArray.length; i++) {
    System.out.println(myArray[i]);
}

// 使用for-each迴圈
for (int element : myArray) {
```

```
    System.out.println(element);
}
```

📖 需要注意的是，如果您嘗試訪問超出陣列範圍的索引（例如，對於一個長度為10的陣列，索引範圍應為0到9），則會拋出 `ArrayIndexOutOfBoundsException` 異常。

📖 如果你只有寫

```
int[] myArray;
```

而沒有new這個array，則myArray將指向 `null`。

📖 在Java中，當我們 `new` 了一個陣列但沒有初始化內容，每個元素會有一個預設的初始值。這個初始值取決於陣列的數據類型：

- 對於數字型別（如 `byte`、`short`、`int`、`long`、`float`、`double`），初始值為0或0.0。
- 對於 `char` 型別，初始值為空字符號（'\u0000'）。
- 對於 `boolean` 型別，初始值為 `false`。
- 對於引用型別（如物件或陣列），初始值為 `null`。

Example of Array Use

```
public class ArrayInitWithExpr {
  public static void main(String[] argv) {
    int i = 4,j = 8;
    int[] a = {4, i, i + j};  // 使用運算式

    for(int k = 0;k < a.length;k++) {
      System.out.println("a[" + k + "] : " + a[k]);
    }
  }
}
```

```
public class DiagonalSum {
    public static void main(String[] args) {
        int[][] matrix = {
```

```
            {1, 2, 3},
            {4, 5, 6},
            {7, 8, 9}
        };

        int sum = 0;
        for (int i = 0; i < matrix.length; i++) {
            sum += matrix[i][i];
        }

        System.out.println("The sum of the diagonal elements
is: " + sum);
    }
}
```

多維陣列及不規則多維陣列

在Java中，您可以創建多維陣列，這是由兩個或多個維度的數組組成的。最常見的多維陣列是二維陣列，它可以被視為一個陣列的陣列，或者可以想像為一個表格，有行和列。

以下是一個創建和初始化二維陣列的示例：

```
int[][] myArray = {
  {1, 2, 3},
  {4, 5, 6},
  {7, 8, 9}
};
```

1	2	3
4	5	6
7	8	9

在這個例子中，`myArray`是一個二維陣列，它包含三個一維陣列。每個一維陣列中都有三個元素。

要訪問二維陣列的元素，您需要提供兩個索引：一個表示行，一個表示列。例如：

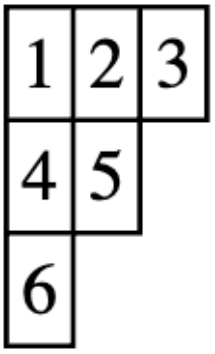
```
int firstElement = myArray[0][0];  // 取得第一行第一列的元素
```

您也可以使用巢狀for迴圈來iterate二維陣列的所有元素：

```
for (int i = 0; i < myArray.length; i++) {
  for (int j = 0; j < myArray[i].length; j++) {
    System.out.println(myArray[i][j]);
  }
}
```

除了規則的多維陣列外，Java還支援不規則的多維陣列（也稱為"鋸齒陣列"）。在不規則的多維陣列中，每個子陣列可以有不同的長度。例如：

```
int[][] myArray = {
  {1, 2, 3},
  {4, 5},
  {6}
};
```



在這個例子中，`myArray`是一個不規則的二維陣列，它的第一個子陣列有三個元素，第二個子陣列有兩個元素，第三個子陣列只有一個元素。

要訪問和iterate不規則多維陣列的元素，其方式與規則多維陣列相同。

- 以下是一個使用`new`創建和初始化規則二維陣列的示例：

```
int[][] myArray = new int[3][3]; // 創建一個3x3的二維陣列

// 初始化陣列
for (int i = 0; i < 3; i++) {
    for (int j = 0; j < 3; j++) {
        myArray[i][j] = i * j;
    }
}
```

- 對於不規則的二維陣列，您可以首先創建一個一維陣列，然後對每個元素單獨分配一個陣列，如下所示：

```
int[][] myArray = new int[3][]; // 創建一個包含3個一維陣列的二維陣列
```

```
// 分別為每個一維陣列分配不同的長度
myArray[0] = new int[3];
myArray[1] = new int[2];
myArray[2] = new int[1];

// 初始化陣列
for (int i = 0; i < myArray.length; i++) {
    for (int j = 0; j < myArray[i].length; j++) {
        myArray[i][j] = i * j;
    }
}
```

📌 Array不是Primitive Data Type

在Java中，陣列並不被視為基本資料型態（Primitive Data Type），而是被視為物件（Object）。基本資料型態包括`byte`、`short`、`int`、`long`、`float`、`double`、`boolean`和`char`。

然而，陣列在Java中被當作物件來處理，每個陣列都是`java.lang.Object`類別的實例。這表示陣列可以有方法和屬性，例如可以使用`length`屬性來獲取陣列的長度。

所以，以下的程式的行為會是如何呢?

```
int[] A = {1, 2, 3};
int[] B = A;
A[0]=0;
System.out.println(B[0]); //B[0]是1還是0？
```

在Java中，你可以使用`System.arraycopy()`方法來複製陣列的內容。以下是一個示例：

```
int[] originalArray = {1, 2, 3, 4, 5};
int[] copiedArray = new int[5];

System.arraycopy(originalArray, 0, copiedArray, 0, originalAr
ray.length);
```

在上述範例中，`System.arraycopy()`方法的參數分別代表：

- `originalArray`：源陣列，即我們要複製的陣列。

- `0`：源陣列中的起始位置。
- `copiedArray`：目標陣列，即我們要複製到的陣列。
- `0`：目標陣列中的起始位置。
- `originalArray.length`：要複製的元素數量。

此外，你也可以使用`Arrays.copyOf()`或者`Arrays.copyOfRange()`方法來複製陣列的內容。

示範題目：成績統計和分析系統

任務描述：
你需要寫一個Java程式來管理一個班級的學生成績。這個程式應該能夠執行以下功能：

1. **輸入學生成績**：允許用戶輸入班級學生的數學成績。假設班級中有10名學生，使用陣列來存儲這些成績。
2. **計算平均成績**：計算並顯示全班學生的數學平均成績。
3. **找出最高和最低成績**：顯示全班中最高和最低的數學成績。
4. **成績分布**：根據成績顯示成績分布情況，如90分以上為A級，80至89分為B級，70至79分為C級，60至69分為D級，60分以下為E級。顯示每個等級的學生人數。
5. **資料驗證**：輸入的成績應該在0到100之間，否則顯示錯誤訊息並要求重新輸入。

技術要求：

- 使用陣列來存儲學生成績。
- 運用if或switch語句進行成績等級的判斷。
- 使用for或do-while迴圈進行成績的輸入和統計計算。
- 應適當使用變數來存儲計算結果，如平均成績、最高成績、最低成績等。

```
graph LR
A[開始] --> B{輸入學生成績}
B -->|輸入正確| C[計算平均成績]
B -->|輸入錯誤| B
C --> D[找出最高和最低成績]
D --> E[成績分布]
E --> F[結束]
```

示例輸出：

```
請輸入10名學生的數學成績：
第1名學生成績：95
第2名學生成績：82
...
平均成績為：XX.X
最高成績為：XX
最低成績為：XX
成績分布：
A級學生人數：X
B級學生人數：X
...
```

這個題目要求學生綜合運用變數宣告、表達式計算、流程控制語句以及陣列操作的知識，同時也要求對於數據驗證和邏輯判斷的能力。

以下是根據上述作業要求的一個Java程式參考答案。這個程式實現了成績的輸入、計算平均成績、找出最高和最低成績、顯示成績分布以及輸入驗證的功能。

```
import java.util.Scanner;

public class GradeStatistics {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int[] scores = new int[10];
        int sum = 0, max = 0, min = 100;
        int[] gradeDistribution = new int[5]; // A, B, C, D,
E

        // 輸入學生成績
        for (int i = 0; i < scores.length; i++) {
            System.out.print("請輸入第" + (i + 1) + "名學生的數學
成績：");
            int score = scanner.nextInt();

            // 驗證成績是否在0到100之間
            while (score < 0 || score > 100) {
                System.out.println("錯誤！成績必須在0到100之
間。");
                System.out.print("請重新輸入第" + (i + 1) + "名
學生的數學成績：");
```

```
                score = scanner.nextInt();
            }

            scores[i] = score;
            sum += score;
            if (score > max) max = score;
            if (score < min) min = score;

            // 成績分布
            if (score >= 90) gradeDistribution[0]++;
            else if (score >= 80) gradeDistribution[1]++;
            else if (score >= 70) gradeDistribution[2]++;
            else if (score >= 60) gradeDistribution[3]++;
            else gradeDistribution[4]++;
        }

        // 計算平均成績
        double average = (double) sum / scores.length;

        // 顯示結果
        //%.1f：指定小數點後顯示的位數，這裡是1位小數
        //如果average是123.456，則output "123.5"
        System.out.println("平均成績為：" + String.format("%.1
f", average));
        System.out.println("最高成績為：" + max);
        System.out.println("最低成績為：" + min);
        System.out.println("成績分布：");
        System.out.println("A級學生人數：" + gradeDistribution
[0]);
        System.out.println("B級學生人數：" + gradeDistribution
[1]);
        System.out.println("C級學生人數：" + gradeDistribution
[2]);
        System.out.println("D級學生人數：" + gradeDistribution
[3]);
        System.out.println("E級學生人數：" + gradeDistribution
[4]);

        scanner.close();
```

```
    }
}
```

這段程式碼首先創建一個 `Scanner` 物件來讀取用戶輸入的成績，並使用一個整數陣列來存儲這些成績。程式通過一個 `for` 迴圈讀取10名學生的成績，並使用 `while` 迴圈確保輸入的成績在0到100之間。程式計算總成績、最高成績、最低成績，並根據成績範圍更新成績分布的統計。最後，程式計算平均成績並輸出結果以及成績分布情況。

Java String

在Java中，`String` 是一個代表一系列character的類別。`String` 類別提供了許多方法來操作character set，例如連接、比較、獲取長度、擷取子character set、轉換為大寫或小寫等。

- **創建字串**：在Java中，您可以使用以下兩種方式來創建字串：

```
String str1 = "Hello";  // 直接賦值
String str2 = new String("Hello");  // 使用new關鍵字
```

- **不可變性**：`String` 是不可變的，這意味著一旦創建了一個 `String` 對象，就不能改變它。如果您試圖改變一個 `String` 對象，Java會創建一個新的 `String` 對象來存儲新的character set，而原來的 `String` 對象將不會被修改。

💡 如果兩個 `String` 物件是直接賦值創建的，假設內容都是"Hello"，那麼他們會指向一個記憶體位置。這稱為字串池（String Pool）機制。但如果是通過 `new` 關鍵字創的 `String` 物件，即使內容相同，他們也會在記憶體中有不同的位置。

String Pool是一個特殊的存儲區域，在JVM的heap內存中。這個機制是為了優化減少Java字串空間。當你直接使用雙引號創建字串時，JVM首先檢查字符串池中是否存在相同內容的字符串。如果存在，Java會返回引用那個已有的字串對象，而不是創建一個新的object。這種方式有助於節省空間，因為它減少了相同內容的字串數量。

```
String s1 = "Hello";
String s2 = "Hello";
```

在這個例子中，`s1` 和 `s2` 都指向String Pool中的同一個字串對象"Hello"，因為它們的內容相同，並且都是直接使用雙引號創建的。

然而，當你使用 `new` 關鍵字創建字串object時，Java會在heap中創建一個新的字串object，即使String Pool中已經存在內容相同的字串。

```
String s3 = new String("Hello");
String s4 = new String("Hello");
```

在這個例子中，即使 `s3` 和 `s4` 的內容相同，它們也會指向heap中的兩個不同的object。這是因為 `new` 關鍵字的使用強制在heap分配新的記憶體空間，並創建新的串object。

這個差別的主要原因是String Pool的設計目的在於優化記憶體使用和提高性能。通共享相同的字串字面值，Java可以減少內存的佔用和降低GC執行的頻率。而 `new` 關鍵字的使用則提供了一種方式來明確地創建獨立的字串Object，即使這樣做可能不麼高效。在需要確保字串對象身份唯一性時，這種能力是有用的。

範例思考

```
String token1 = new String("abc123");
String token2 = new String("abc123");
String token3 = "abc123";
String token4 = "abc123";

System.out.println(token1 == token2); //what is the output
System.out.println(token3 == token4); //what is the output
```

- **操作和方法**：`String`類別提供了許多方法來操作character set：
  - `concat(String str)`：將指定的character set連接到此character set的結尾。

```
String str1 = "Hello";
String str2 = " World";
System.out.println(str1.concat(str2));  // 輸出"Hello Worl
d"
```

  - `equals(Object obj)`：將此string與指定的對象比較。如果指定的對象是一個string，那麼該方法將返回`true`，if and only if這兩個string包含相同的character sequence。

```
String str1 = "Hello";
String str2 = new String("Hello");
System.out.println(str1.equals(str2));  // 輸出true
```

  - `length()`：返回此string的長度。

```
String str = "Hello";
System.out.println(str.length());  // 輸出5
```

  - `substring(int beginIndex)`和`substring(int beginIndex, int endIndex)`：返回一個新的string，這個string是此string的一個子string。

```
String str = "Hello";
System.out.println(str.substring(1));  // 輸出"ello"
System.out.println(str.substring(1, 4));  // 輸出"ell"
```

  - `toUpperCase()`：將所有在此string中的character轉換為大寫。

```
String str = "Hello";
System.out.println(str.toUpperCase());  // 輸出"HELLO"
```

  - `toLowerCase()`：將所有在此字串中的character轉換為小寫。

```
String str = "Hello";
System.out.println(str.toLowerCase());  // 輸出"hello"
```

- **其他資料型態的轉換**：您可以使用`String`類別的`valueOf()`方法來將其他資料型態轉換為string，例如：

```
int num = 123;
String str = String.valueOf(num);
System.out.println(str);  // 輸出"123"
```

相反地，如果您需要將一個string轉換為其他資料型態，如`int`或`double`，您可以使用`Integer.parseInt(String s)`或`Double.parseDouble(String s)`方法。

```
String str = "123";
int num = Integer.parseInt(str);
System.out.println(num);  // 輸出123
```

觀念釐清

在Java中，主函式（main function）的返回值類型必須為void，意即該函式不返回任何值。主函式的標準signature為`public static void main(String[] args)`。所以如果你這樣寫main function：

```
public class HelloWorld {
    public static int main(String[] args) {
        return 1;
    }
}
```

則你可能會遇到compile success但runtime會有錯誤的message：

```
Main method must return a value of type void in class HelloWorlc
define the main method as: public static void main(String[] args
```

consigliere
@moyodre
大家會怎麼在自己的履歷上寫「我換了一個燈泡」？
M
@MuyiwaSaka
在沒有造成任何成本超支以及安全事故的情況下，獨力成功管理了環境照明系統的升級與安裝。