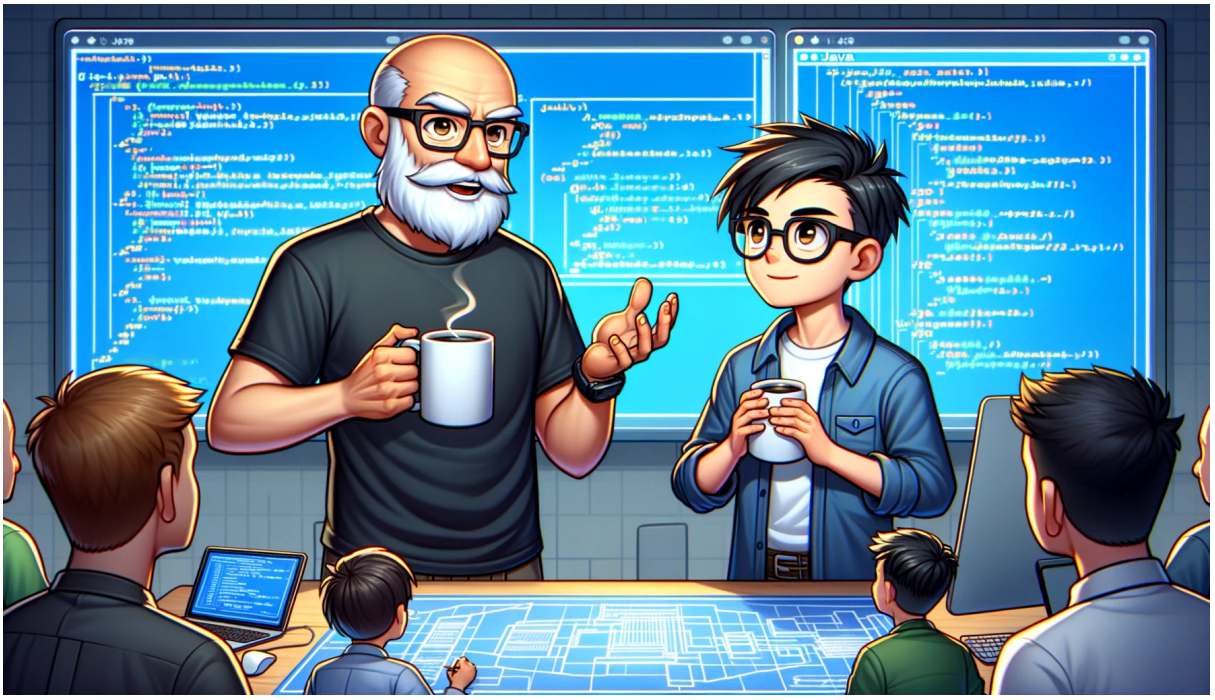
PD2 Lecture 1 - Introduction to JAVA



Generated by DALL.E

Java History and Version

▼ Introduction to Java History and Version

PD2 Lecture 1 - Introduction to Java History and Version

Remind of Java Licenses

OpenJDK

- **授權**：OpenJDK使用的是GNU General Public License, version 2, with the Classpath Exception（`GPLv2+CPE`）。這種授權允許任何人自由地使用、修改和分發代碼，而無需支付授權費用，但必須在分發修改後的版本時公開源代碼。

  GPLv2+CPE

- **開源**：作為一個完全開源的實現，OpenJDK鼓勵社區參與和貢獻。
- **更新和支持**：由於是開源項目，OpenJDK的更新和支持來自於社區和各大廠商。Oracle和其他公司如Red Hat、Amazon、Azul等，都提供OpenJDK的長期支持（LTS）版本。

Oracle JDK

- **授權**：從Java 11開始，Oracle JDK採用了Oracle Technology Network License Agreement for Oracle Java SE，這是一種商業授權。這意味著對於所有商業用途，使用Oracle JDK都需要支付授權費用。對於個人使用、開發、測試和演示是免費的。
- **商業特性和支持**：Oracle JDK包括一些OpenJDK中不提供的商業特性和工具，並提供付費的商業支持。這包括性能、安全性和穩定性的增強，以及專門的監控和管理工具。
- **更新頻率和支持**：Oracle JDK提供更頻繁的更新和更長期的支持選項，但這些通常需要根據Oracle的商業授權來購買。

💡 Java/C/Python已是世界前三大開發程式語言，然後目前Oracle掌握Java的開發進程，請了解到如果你有「商業化用途」或「長期維護需求」，需要對Oracle授權議題特別注意。

而對一般非商業應用則無需考量。

> 當然，你如果採用OpenJDK則無需擔心商業化用途，但長期維護則由開源社群來支持。

Java/C/Python are the top three development languages in the world, and currently, Oracle controls the development process of Java with its dynamic licensing. Please understand that if you have "commercial use" or "long-term maintenance needs", you need to pay special attention to the issue of licensing.

For general non-commercial applications, there is no need to consider this.

> Of course, if you choose to use OpenJDK, you don't need to worry about commercial use, but long-term maintenance is supported by the open source community.

Java Framework

Java平台概觀

Java平台是一個廣泛使用的計算平台，用於開發和執行可移植的應用程序。它由兩個主要組件組成：Java開發工具包（JDK）和Java運行時環境（JRE）。

JDK是Java開發人員的工具，它包含了Java編譯器、調試器和其他開發工具。開發人員可以使用JDK來創建Java應用程序，並將其編譯為可運行的字節碼。

JRE是運行Java應用程序所需的環境。它包含Java虛擬機（JVM）和Java類庫。JVM是Java應用程序運行的虛擬機器，它可以解釋和執行Java字節碼。Java類庫包含了許多用於開發Java應用程序的類和函數。

Java平台還包括了一個廣泛的標準庫，提供了許多用於開發各種應用程序的類和函數。這些類和函數可以幫助開發人員加快應用程序的開發速度，並提供了許多常用的功能，如文件處理、網絡通信和數據庫訪問等。

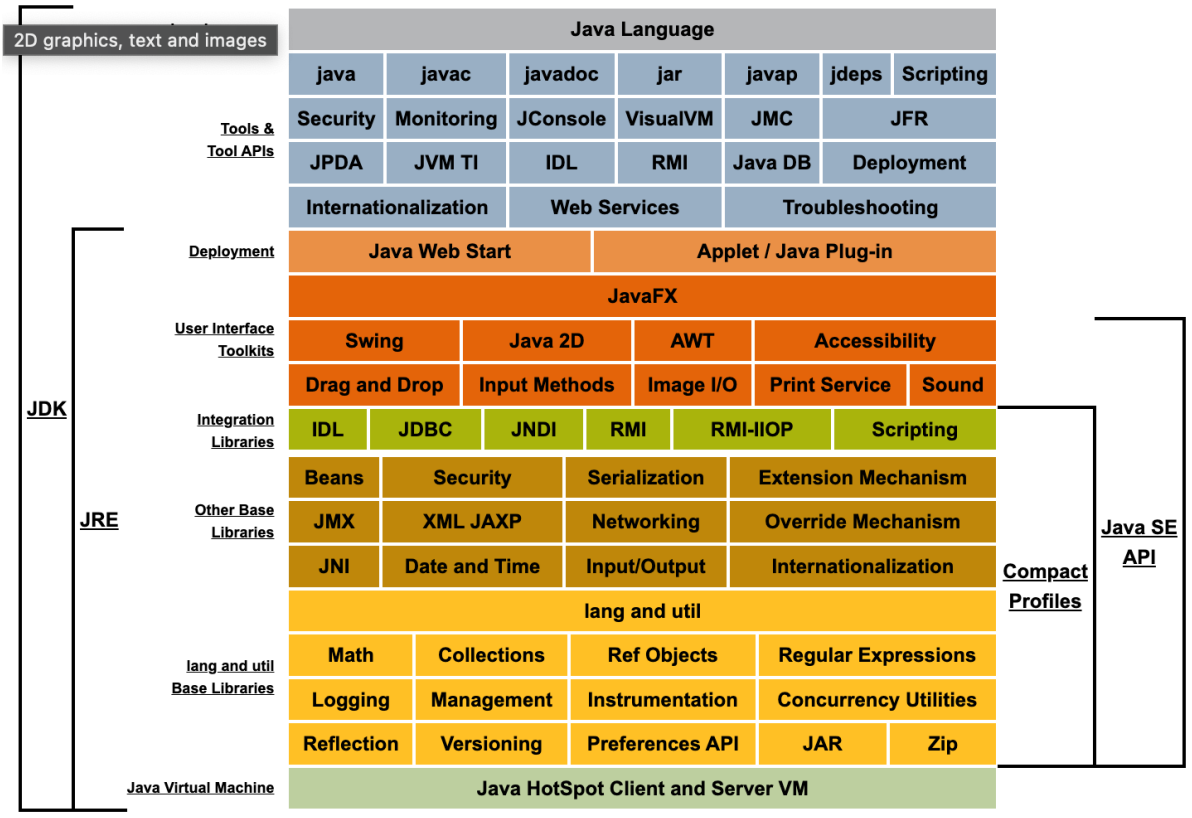
Java平台還支持眾多的框架和庫，這些框架和庫可以幫助開發人員更容易地構建各種應用程序。一些著名的Java框架包括Spring框架、Hibernate框架和Apache Struts框架等。這些框架提供了開發人員所需的組件和模塊，使開發工作更加高效和便捷。

除了基本功能和庫外，Java平台還提供了豐富的開發工具，如集成開發環境（IDE）和程式碼編輯器等。這些工具可以幫助開發人員編寫、調試和測試Java應用程序，提高開發效率和品質。

Java平台的特點之一是其可移植性。由於Java字節碼是運行在JVM上的，所以Java應用程序可以在不同的操作系統和硬體平台上運行，而不需要重新編寫。這使得Java成為開發跨平台應用程序的首選語言之一。

另外，Java平台還提供了豐富的安全特性和機制，用於保護應用程序免受安全漏洞和攻擊。這些特性包括內建的安全管理器、加密庫和安全通信協議等，可以有效地提高應用程序的安全性。

Java平台是一個功能豐富且廣泛使用的計算平台，它提供了豐富的開發工具、運行環境和庫，使開發人員能夠開發出高效、可移植和安全的應用程序。無論是開發桌面應用、網絡應用還是移動應用，Java都是一個強大的選擇。



Java Platform Standard Edition 8

Java中有許多主要的Package，以下是其中幾個常用的：

- `java.lang`：這是Java的核心Package，提供了一些基本的類和功能，如基本數據類型、字串、數學運算等。該Package中的類可以直接使用，無需額外的import語句。
- `java.util`：這個Package提供了各種實用的工具類，如集合框架（如ArrayList、HashMap等）、日期和時間處理、隨機數生成、正則表達式等。在Java開發中，使用`java.util`是非常常見的，因為它提供了許多常用的工具類和數據結構，方便開發人員進行各種操作和處理。
- `java.io`：這個Package提供了用於進行文件和流操作的類，如讀寫文件、串流操作、對象序列化等。在Java中，文件和流操作是非常常見的任務，而`java.io`提供了許多類和方法，使得文件和流的操作變得簡單和方便。
- `java.net`：這個Package提供了用於網絡通信的類和接口，如Socket、URL等，可以用於創建網絡連接、進行數據傳輸等。在當今互聯網時代，網絡通信是非常重要的一部分，而`java.net`提供了豐富的功能和工具，使得開發人員可以輕鬆地進行網絡通信的操作。
- `java.awt`：這個Package提供了用於創建圖形用戶界面（GUI）的類和接口，如窗口、按鈕、標籤等。在Java中，GUI應用程序是非常常見的，而`java.awt`提供了基

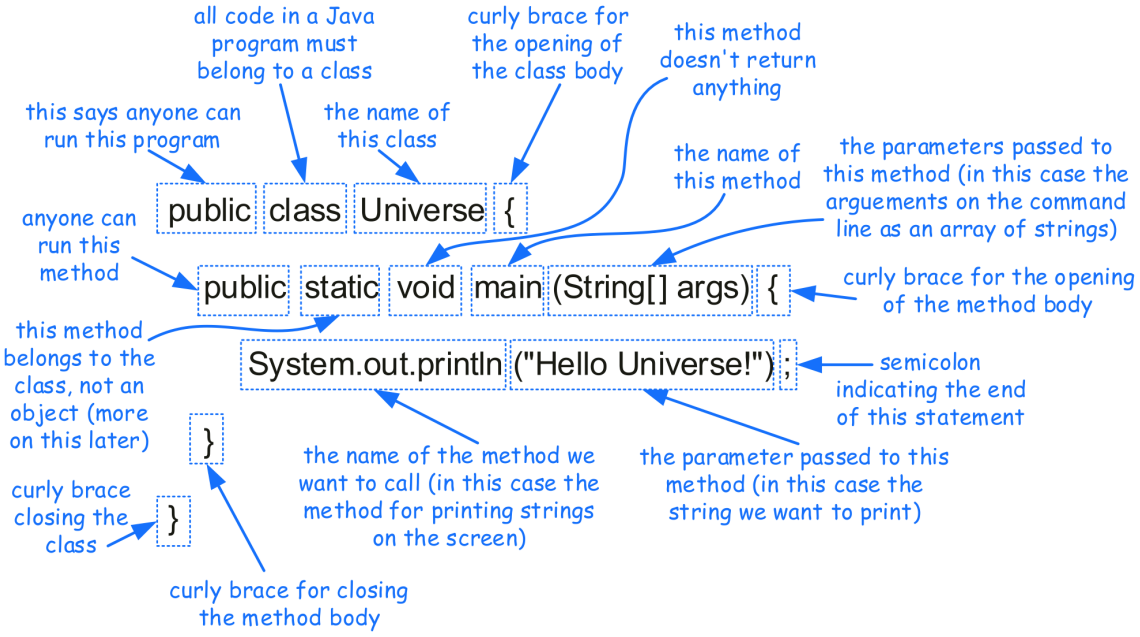
本的GUI元素和功能，使得開發人員可以輕鬆地創建和設計各種用戶界面。

- `javax.swing`：這個Package是Java的GUI工具包，提供了更多功能強大的GUI組件，如JFrame、JButton、JLabel等。相比於`java.awt`，`javax.swing`提供了更多的GUI組件和功能，使得開發人員可以創建更複雜和具有豐富交互性的用戶界面。

除了上述提及的Package外，Java還有許多其他重要的Package，如`java.math`用於數學運算、`java.nio`用於非阻塞I/O操作、`java.sql`用於數據庫操作等。這些Package提供了各種功能和工具，滿足了不同領域和應用的需求。

總之，Java中的主要Package提供了各種功能和工具，使得Java成為一個強大且功能豐富的編程語言。開發人員可以利用這些Package來實現各種應用程序，從基本的數據處理到複雜的圖形界面，Java都提供了相應的Package來支持開發工作。

Your First Java Example



Quoted from Data Structures and Algorithms in Java, 6/e

HelloWorld

```
public class HelloWorld {
    public static void main(String[] args) { //所有的程式都是從main開始
        System.out.println("Hello, World!");
```

```
    }
}
```

To execute the Java code, follow these steps:

1. Open a text editor and copy the Java code into a new file. Save the file with a `.java` extension, for example, `HelloWorld.java`.
2. Open a command prompt or terminal and navigate to the directory where you saved the Java file.
3. Compile the Java file by typing `javac HelloWorld.java` and press Enter. This will create a bytecode file with a `.class` extension, for example, `HelloWorld.class`.
4. Run the compiled Java program by typing `java HelloWorld` and press Enter. The program will execute, and you will see the output `Hello, World!` printed to the console.

Note: Make sure you have Java Development Kit (JDK) installed on your computer before executing the Java code. You can download the JDK from the official Oracle website.

What is the difference between Java and C++?

C++和Java是兩種常見的程式語言，它們在執行過程中有一些重要的不同之處：

1. **編譯與解釋**：C++是一種編譯語言，需要使用編譯器將原始碼轉換為機器碼，然後再執行。而Java是一種解釋語言，它使用Java虛擬機（JVM）來執行Java字節碼。
2. **記憶管理**：C++提供了手動的內存管理，開發人員需要負責分配和釋放內存。這使得C++更具靈活性，但也容易引起記憶體泄漏和段錯誤等問題。Java使用自動內存管理，開發人員不需要手動處理內存，由JVM負責進行垃圾回收。
3. **平台依賴性**：C++程式碼通常是特定平台的，需要對每個平台進行編譯。這意味著在不同的平台上運行C++程式碼可能需要重新編譯。Java程式碼則是平台無關的，只需在具有相應JVM的平台上執行即可。
4. **物件導向程式設計（OOP）**：C++和Java都支持物件導向程式設計。然而，Java是一種純粹的物件導向程式設計語言，所有的程式碼都必須位於類中。C++則更具靈活性，可以使用物件導向程式設計，也可以使用傳統的結構化程式設計。

5. **執行效能**：由於C++是直接編譯為機器碼，因此在執行效能上通常優於Java。然而，Java的JVM使用即時編譯（JIT）技術，可以將熱門的程式碼片段動態編譯為機器碼，提高執行效能。
6. **程式庫和生態系統**：C++有豐富的程式庫和工具，可以用於各種應用領域，包括圖形、數據庫、網絡等。Java也擁有龐大的生態系統，擁有豐富的程式庫和框架，尤其在企業應用領域非常流行。

總而言之，C++和Java是兩種不同的程式語言，它們在執行過程中有一些重要的不同之處。選擇使用哪種語言取決於項目的需求、開發人員的經驗和所選平台的特性。

Java Coding Structure

```
import java.io.*;

public class MyClass {
    // Fields
    private int myField;

    // Constructor
    public MyClass(int field) {
        myField = field;
    }

    // Methods
    public void myMethod() {
        System.out.println("Hello, world!");
    }

    // Main method
    public static void main(String[] args) {
        MyClass myObject = new MyClass(10);
        myObject.myMethod();
    }
}
```

在這個簡單的Java程式中，我們展示了Java的基本程式碼結構。以下是程式的各個部分的說明：

- `import java.*.`：使用那些package函式。
- `public class MyClass { ... }`：定義了一個名為`MyClass`的公開類別。類別名稱通常以大寫字母開頭，且與檔案名稱一致。
- `private int myField;`：定義了一個私有整數型別的欄位`myField`。欄位是類別的屬性，用於存儲對象的狀態。
- `public MyClass(int field) { ... }`：定義了一個公開的建構子`MyClass`，它接受一個整數參數`field`。建構子用於初始化對象的狀態。
- `public void myMethod() { ... }`：定義了一個公開的方法`myMethod`，它沒有參數和返回值。方法是類別的行為，用於執行特定的操作。
- `public static void main(String[] args) { ... }`：定義了一個公開的靜態方法`main`，它接受一個字串陣列參數`args`。`main`方法是Java程式的入口點，程式從這裡開始執行。

這個程式示範了如何定義類別、欄位、建構子、方法以及`main`方法。你可以根據需求在類別中添加更多的欄位和方法，並在`main`方法中創建對象並調用方法。

請注意，Java程式必須放在檔案中並以`.java`為副檔名保存。程式中的每個部分都必須位於適當的區塊中（例如，方法必須位於類別中），並使用大括號`{}`來定義區塊的範圍。每個語句必須以分號結束。

Here's a simple example to demonstrate the usage of the `import` statement in Java:

```
import java.util.ArrayList;

public class Example {
    public static void main(String[] args) {
        // Create an ArrayList object
        ArrayList<String> list = new ArrayList<>();

        // Add elements to the ArrayList
        list.add("Hello");
        list.add("World");

        // Print the elements of the ArrayList
```

```
        for (String element : list) {
            System.out.println(element);
        }
    }
}
```

In the above example, we are importing the `ArrayList` class from the `java.util` package using the `import` statement: `import java.util.ArrayList;`. This allows us to use the `ArrayList` class in our code without specifying the fully qualified name (`java.util.ArrayList`) every time.

By importing the `ArrayList` class, we can create an `ArrayList` object, add elements to it, and print the elements using a `for-each` loop.

Importing classes using the `import` statement helps in reducing the verbosity of code and makes it more readable. It allows us to use the simple name of the imported class directly in our code.

```
package exercise.netdb.ncku;

public class Exercise1 {
    public static void main(String[] args) {
        var m = 1000;
        var n = 495;

        while(n != 0) {
            var r = m % n;
            m = n;
            n = r;
        }
        System.out.printf("first test", m);
    }
}
```

Your First Exercise of Java Class

以下是一個實作連結串列（linked list）的簡單範例，使用了三個以上的類別：

```java
// 節點類別
class Node {
    int data;
    Node next;

    public Node(int data) {
        this.data = data;
        this.next = null;
    }
}

// 連結串列類別
class LinkedList {
    Node head;

    public LinkedList() {
        this.head = null;
    }

    // 在連結串列尾端新增節點
    public void addNode(int data) {
        Node newNode = new Node(data);

        if (head == null) {
            head = newNode;
        } else {
            Node current = head;
            while (current.next != null) {
                current = current.next;
            }
            current.next = newNode;
        }
    }

    // 列印連結串列中的節點資料
    public void printList() {
        Node current = head;
```

```
        System.out.print("連結串列內容: ");
        while (current != null) {
            System.out.print(current.data + " ");
            current = current.next;
        }
        System.out.println();
    }
}

// 主程式類別
public class Main {
    public static void main(String[] args) {
        LinkedList linkedList = new LinkedList();

        // 新增節點到連結串列
        linkedList.addNode(1);
        linkedList.addNode(2);
        linkedList.addNode(3);

        // 列印連結串列內容
        linkedList.printList();
    }
}
```

在這個範例中，我們建立了三個類別：`Node` 代表連結串列的節點，`LinkedList` 代表連結串列本身，`Main` 是主程式類別用於執行程式。

連結串列是一種常見的資料結構，由一連串的節點組成，每個節點包含資料和一個指向下一個節點的指標。在這個範例中，我們實作了連結串列的基本功能，包括新增節點和列印連結串列內容。

在主程式中，我們建立了一個 `LinkedList` 物件，並使用 `addNode` 方法新增了三個節點到連結串列中。最後，我們使用 `printList` 方法列印了連結串列的內容。

請注意，這只是一個簡單的範例，用於示範連結串列的基本操作。在實際應用中，連結串列可能包含更多的功能和方法，並且可以根據需求進行擴展和修改。