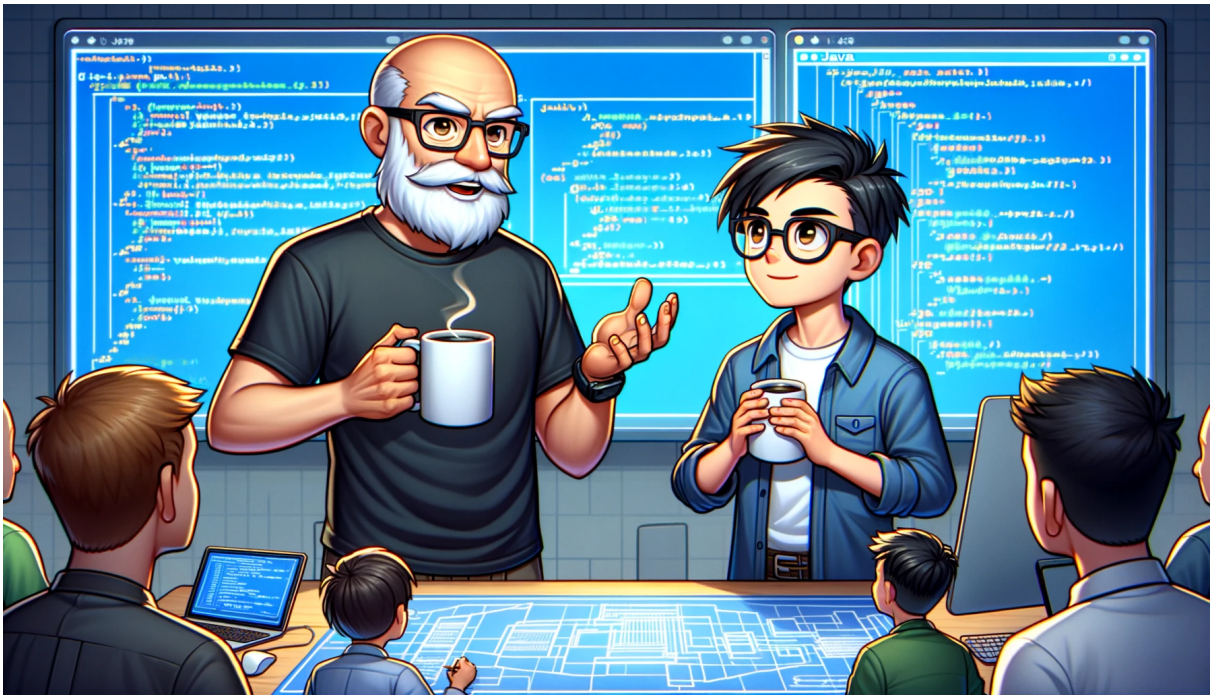
PD2 Lecture 1 - Introduction to JAVA



Generated by DALL.E

Java History and Version

▼ Introduction to Java History and Java Version

PD2 Lecture 1 - Introduction to Java History and Java 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，而無需支付授權費用，但必須在分發JDK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鼓勵社群參與和貢獻。
- **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、Debugger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Java bytecode。

JRE是運行Java應用程序所需的環境。它包含Java虛擬機（JVM）和Java class library。JVM是Java應用程序運行的虛擬機器，它可以解釋和執行Java bytecode。Java library包含了許多用於開發Java應用程序的class和method。

Java平台還包括了一個廣泛的standard library，提供了許多用於開發各種應用程序的class和method，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、Networking和database access等。

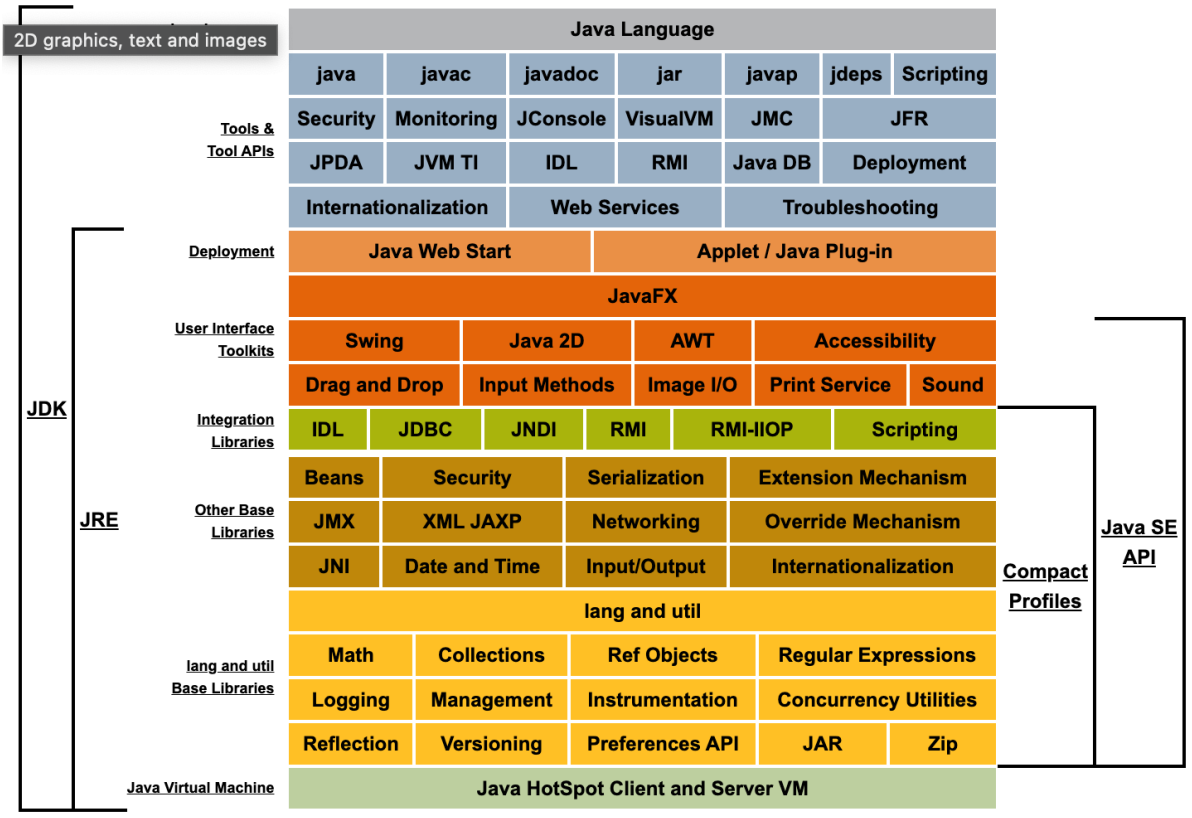
Java平台還支持眾多的framework，可以幫助開發人員更容易地構建各種應用程序。一些著名的Java框架包括Spring、Hibernate和Apache Struts框架等。這些框架提供了開發人員所需的組件和模塊，使開發工作更加高效和便捷。

除了基本功能和library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、debug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class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元件，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資料流操作的class，如讀寫文件、串流操作、對象序列化等。在Java中，文件和資料流操作是非常常見的任務，而`java.io`提供了許class和方法，使得文件和資料串流的操作變得簡單和方便。
- `java.net`：這個Package提供了用於網路通信的類和接口，如Socket、URL等，可以用於創建網路連接、進行數據傳輸等。在當今互聯網時代，網路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。

- `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Non-blocking I/O操作、`java.sql`用於database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。

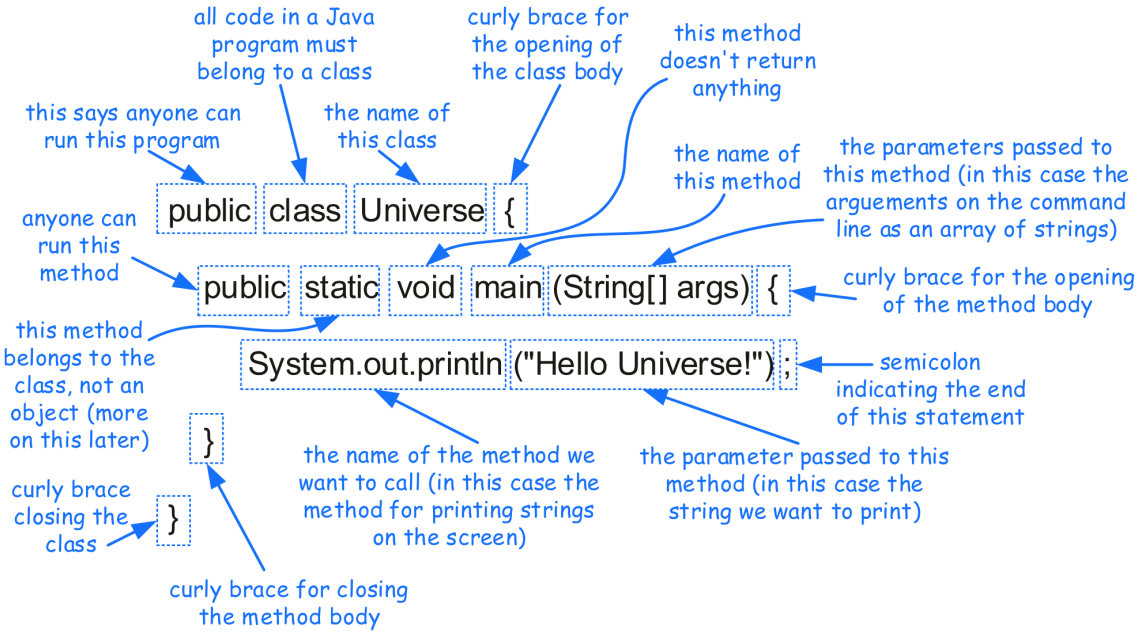
總之，Java中的主要Package提供了各種功能和工具，使得Java成為一個強大且功能豐富的language。開發人員可以利用這些Package來實現各種應用程序，從基本的數據處理到複雜的圖形界面，Java都提供了相應的Package來支持開發工作。

Note - 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. **Compilation and Interpretation**：C++是一種compiled language，需要使用編譯器將原始碼轉換為machine code，然後再執行。而Java是一種interpreted language，它使用Java虛擬機（JVM）來執行Java bytecode。
2. **Memory Management**：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memory leak等問題。Java使用自動內存管理，開發人員不需要手動處理memory，由JVM負責進行GC（Garbage Collection）。
3. **Platform Dependence**：C++程式碼通常是platform-specific，也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，同一個code也不一定能在不同平台執行。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class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（相融於C）。
5. **執行效能**：由於C++是直接編譯為machine code，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hot-spot code snippet動態編譯為機器碼，提高執行效能。

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 already have Java Development Kit (JDK) installed on your computer before executing the Java code. You can download the JDK from the official Oracle website.

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。基本會使用的包含java.io。
- `public class MyClass { ... }`：定義了一個名為`MyClass`的public class。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private整數型別的variable `myField`。variable是class的屬性Attribute，封裝在這個MyClass中。有時我們就會直接稱之為

Attribute。

- `public MyClass(int field) { ... }`：定義了一個public的建構子Constructor `MyClass`，它接受一個整數參數 `field`。建構子用於初始化這個object。
- `public void myMethod() { ... }`：定義了一個公開的方法 `myMethod`，它沒有參數和返回值。Method是Class的可操作行為，用於執行特定的操作。
- `public static void main(String[] args) { ... }`：定義了一個public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class類別、Variable、Constructor、Method以及 `main` 方法。你可以根據需求在類別中添加更多的Variable和method，並在 `main` 方法中new出對應的object。

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code block中（例如，method必須位於類別中），並使用大括號 `{}` 來定義block的範圍。每個語句必須以分號結束。

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class：

```
// 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 MyList {
    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();
    }
}
```

```
graph TB
  A[Start]
  B{Create LinkedList object}
  C[AddNode method: Create new Node]
  D{Is head null?}
  E[Assign new Node to head]
  F[Traverse to end of list]
  G[Assign new Node to current Node's next]
  H[PrintList method: Traverse list and print data]
  I[End]
  A --> B
  B --> C
  C --> D
  D -- Yes --> E
  E --> H
  D -- No --> F
  F --> G
  G --> H
  G -.->|repeat adding nodes|C
  H --> I
```

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，`MyList` 是主程式class，用於啟動執行。

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。

概念清晰 - Class versus Object

在Java中，類別`class`可被視為一個藍圖或模板Template，它定義了可以創建的物件（`object`）的型態和行為。一個`class`可以包含Variable（或稱為`Attribute`）和Method。當你根據這個藍圖創建一個具體實例時，那個實例就是一個`object`。每個對象都可以擁有自己的`Attribute`，但它們共享同一套`class`定義的Method。

關係和差異

- **關係**：Class和Object之間的關係是模板和實例的關係。Class提供了創建物件的具體細節，而物件是根據這些細節創建的實例。
- **差異**：
  - Class是`抽象的`，它不能直接用來表示具體的事物。
  - Object是`具體的`，它是根據Class藍圖創建的，代表了具體的實例。有時我們會說Object是一種instance of class。

下面是簡單示意，表示前一個例子中的node跟不同的node object之間的關係：

```
classDiagram
    NodeClass <|-- NodeObject A: 實例化instance
    NodeClass <|-- NodeObject B: 實例化instance
    NodeClass <|-- NodeObject C: 實例化instance
    NodeClass : +data
    NodeClass : +next
    class NodeObject A {
        data: "1"
        next: "NodeObjectB"
    }
    class NodeObject B {
        data: "2"
        next: "NodeObjectC"
    }
    class NodeObject C {
        data: "3"
```

```
        next: "null"
    }
```

這個圖展示了Class是如何被實例化成一個或多個Object，每個Object都具有從Class繼承來的屬性和方法，但具體的屬性值會根據不同的對象而有所不同。

我們會在之後的課程說明OOP的細節，請同學現階段先了解這些基本的差異即可。

Java Variable Scope

在Java中，變數的作用域（Scope）和存取修飾符（Access Modifiers）決定了變數可以被access的範圍和方式。以下是主要的存取修飾符和其他關鍵字（如static或final）對變數作用域的影響說明：

存取修飾符

1. **public**: 可以被任何其他類別access。若一個變數被標記為public，那麼它可以在任何地方被access，包括不同的Package。
2. **private**: 只能在定義它的class內部access。若一個變數被標記為private，那麼只有同一個類別的Method法可以access它。
3. **protected**: 可以被同一package內的其他class以及不同package中的自己的子類別access。protected提供了一種折衷的access級別（較不常用）。
4. **(Default)**: 只能被同一package內的其他class access。也就是如果不指定任何asscess modifiers，則變數將為Package-private。

Static修飾符

- **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修飾符

- **final**: 表示常量，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，該變數成為常量；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。

影響Summary

- 使用**public**修飾符的變數可以被任何其他類別access，這提供了最大的visibility。
- 使用**private**修飾符的變數只能在其所屬的class中被access，這有助於封裝該class的內部狀態。
- 使用**protected**修飾符的變數可以被同一package內的類別或任何子類別訪問。
- **static**變數屬於class本身，所有的實例共享這一靜態變數，這常用於實現class-level的變數或方法。
- **final**變數一旦初始化後，其值就不可改變，常用於定義常量。

這些修飾符可以組合使用來達到更精細的控制，例如，你可以有一個 `public static final` 變數，這會創建一個公開的、靜態的、不可變的class-level常量。

Public範例

```
public class Car {
    public int year;

    public Car(int year) {
        this.year = year;
    }

    public void displayYear() {
        System.out.println("Year: " + year);
    }
}
```

在這個範例中， `year` 變數和 `displayYear()` 方法都被標記為 `public` ，這意味著它們可以從任何其他類別訪問。

Private範例

```
public class Person {
    private String name;

    public Person(String name) {
        this.name = name;
    }
```

```
    public String getName() {
        return name;
    }
}
```

這裡，`name`變數是`private`的，這表示它只能在`Person`類別內部被訪問。外部類別需要通過`getName()`方法來獲取`name`變數的值。

Protected範例

```
package family;

public class Parent {
    protected String familyName;

    public Parent(String familyName) {
        this.familyName = familyName;
    }
}

class Child extends Parent {
    public Child(String familyName) {
        super(familyName);
    }

    public void displayFamilyName() {
        System.out.println("Family Name: " + familyName);
    }
}
```

在這個範例中，`familyName`變數是`protected`的，這意味著它可以被同一package內的其他類別以及所有子類別訪問。`Child`類別作為`Parent`的子類，可以訪問`familyName`變數。

Static範例

```
public class Calculator {
    public static final double PI = 3.14159;
```

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

在這個範例中，`PI`變數和`add()`方法都是`static`的，這意味著它們屬於`Calculator`類別，可以不需要創建類的實例來訪問。例如，`Calculator.add(5, 3)`或`Calculator.PI`。

Final範例

```
public class Configuration {
    public static final String VERSION = "1.0.0";
}
```

這裡，`VERSION`變數被標記為`final`，這表示它的值一旦被賦值（在這裡是`"1.0.0"`），就不能被修改。這是一種定義常量的常見方式。

這些範例展示了如何在Java中使用存取修飾符和`static`、`final`關鍵字來控制變數的作用域和行為。

It doesn't work..... why?
It works..... why?