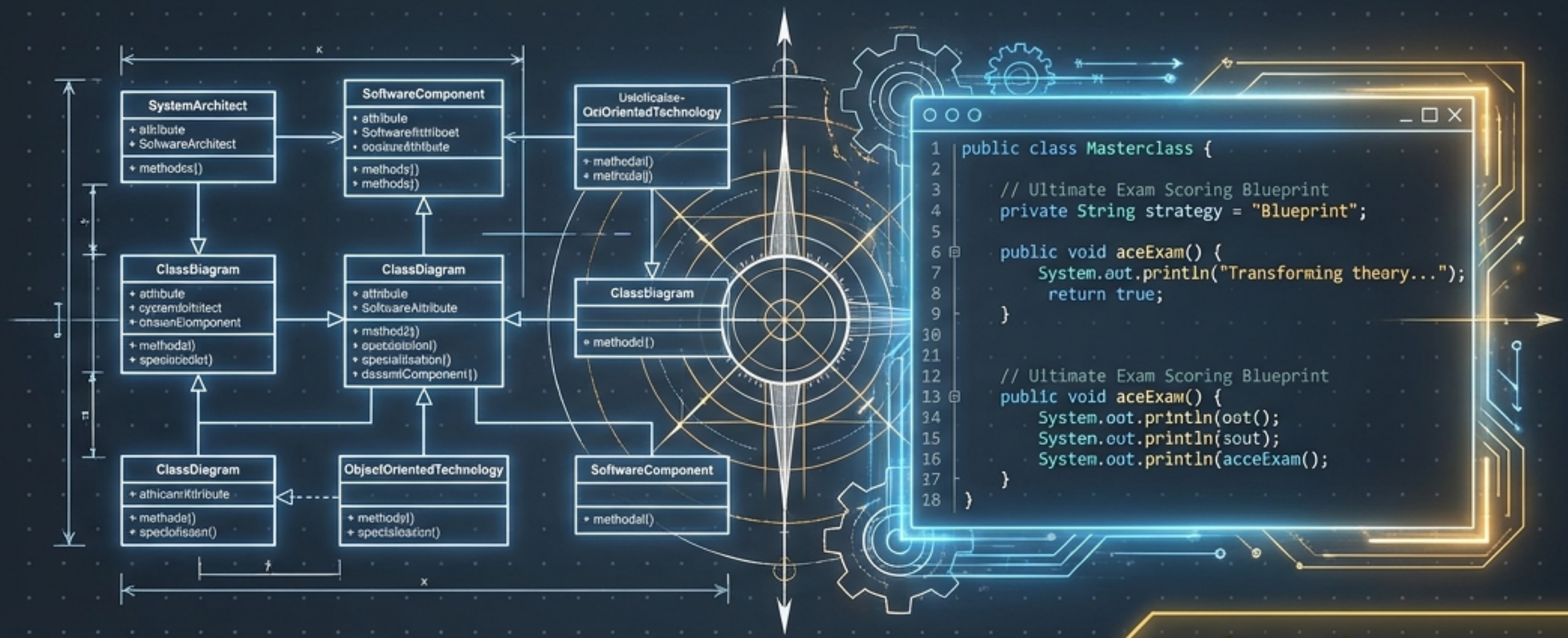
OOT Masterclass 從零到精通的考試攻略

Blueprint for Acing Object-Oriented Technology



Target: ITP4909
Object-Oriented Technology

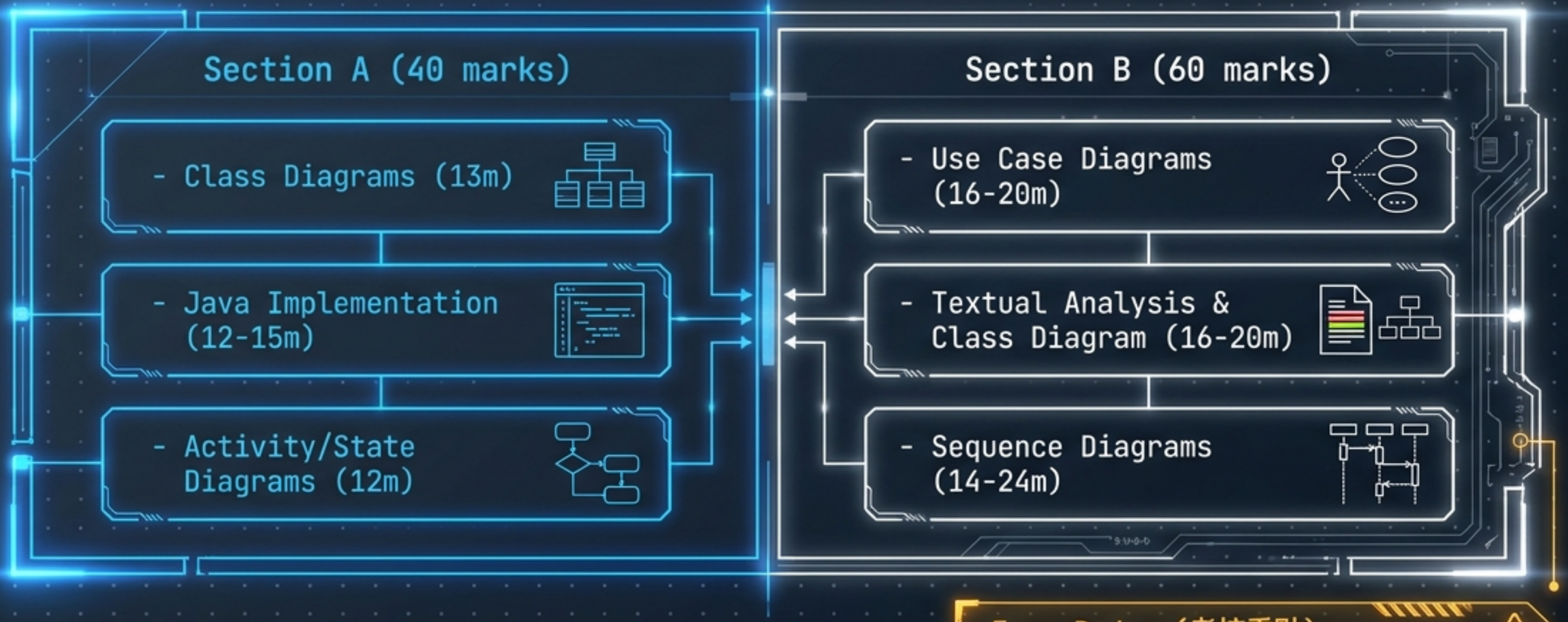
By *Professor OOT*

Professor's Note:

Welcome to the Masterclass! We will transform complex theory into your ultimate exam scoring blueprint.

Decoding the ITP4909 Exam Anatomy

知己知彼，百戰百勝。這張藍圖展示了考試的分數分佈。Section A 測試你的核心圖表與代碼實作能力；Section B 則是綜合應用大題。



Exam Radar（考核重點）

Master the UML-to-Java translation, and you've already secured a massive portion of Section A! Don't lose easy marks here.

Textual Analysis（文本分析）尋找系統類別

教你如何在長篇大論的題目中，精準找出有用的名詞作為系統類別（Candidate Classes）。不要憑空想像，一切皆來自文本！

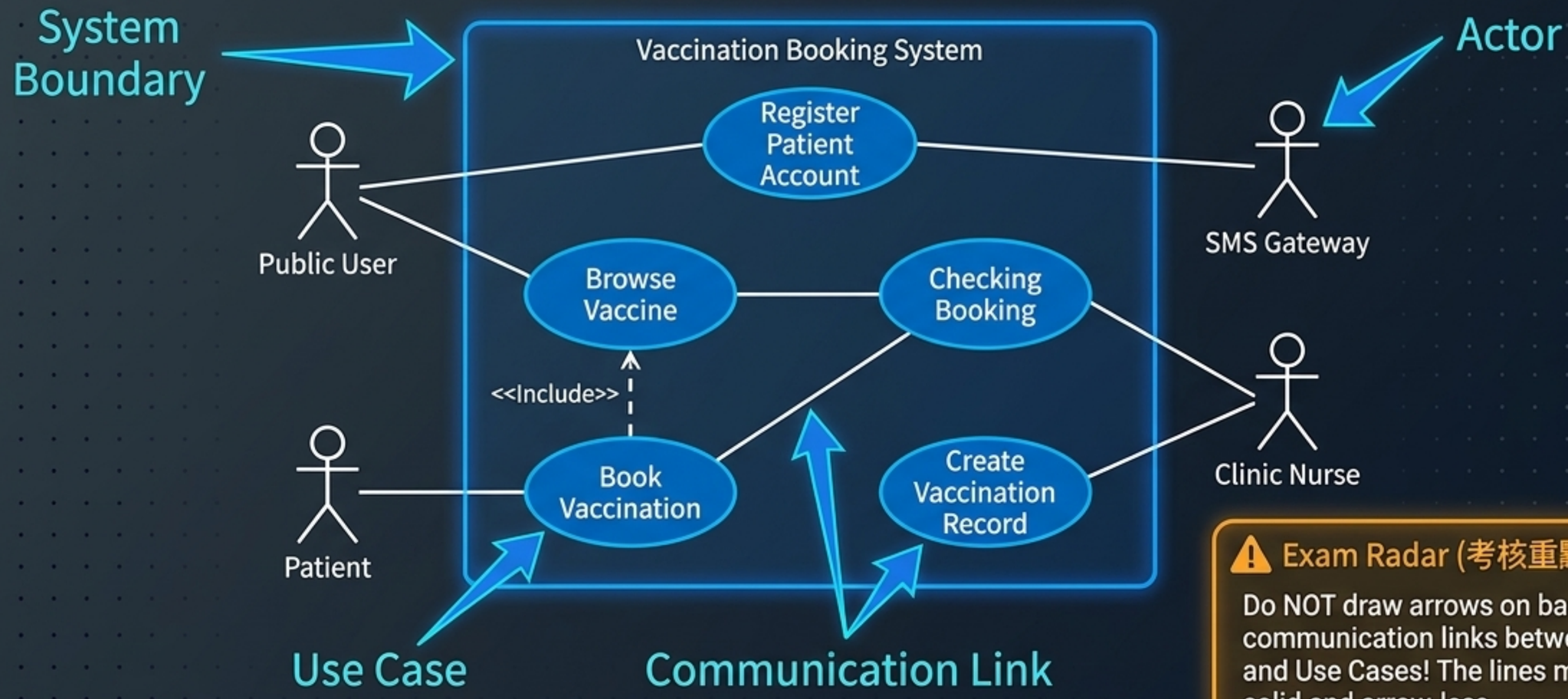
Keyword (Noun)	Candidate Class	Reason / Category
Patient / Clinic Nurse	Patient / Clinic Nurse	Role Play
Vaccination Centre / Vaccine	Vaccination Centre / Vaccine	Tangible Thing
Booking	Booking	Conceptual / Event
SMS Gateway	SMS Gateway	External System

Exam Radar（考核重點）

Always use the exact English categories to score full marks: Role Play, Tangible Thing, Conceptual / Event, External System.

Use Case Diagrams (用例圖) 定義系統邊界

用例圖代表系統的「需求」(What it does)，而非內部運作。定義系統邊界 (System Boundary)，並將使用者 (Actor) 放在外部。



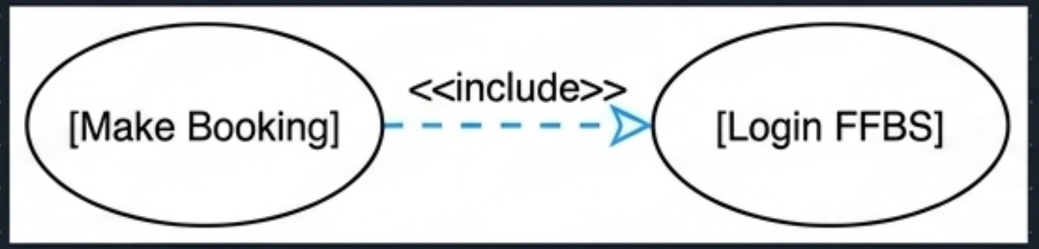
⚠ Exam Radar (考核重點)

Do NOT draw arrows on basic communication links between Actors and Use Cases! The lines must be solid and arrow-less.

Decoding Use Case Relationships: 包含與擴展

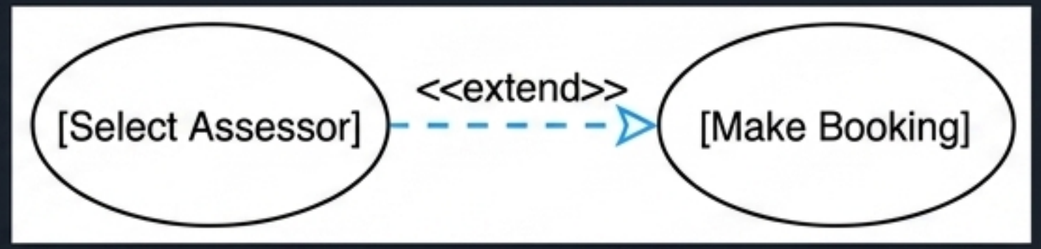
這兩個虛線箭頭是同學最常失分的地方。理解 Base Use Case (基礎用例) 的執行條件是關鍵。

<<include>> (Mandatory)



The base use case CANNOT finish without this.
Direction: Base -> Included.

<<extend>> (Optional)

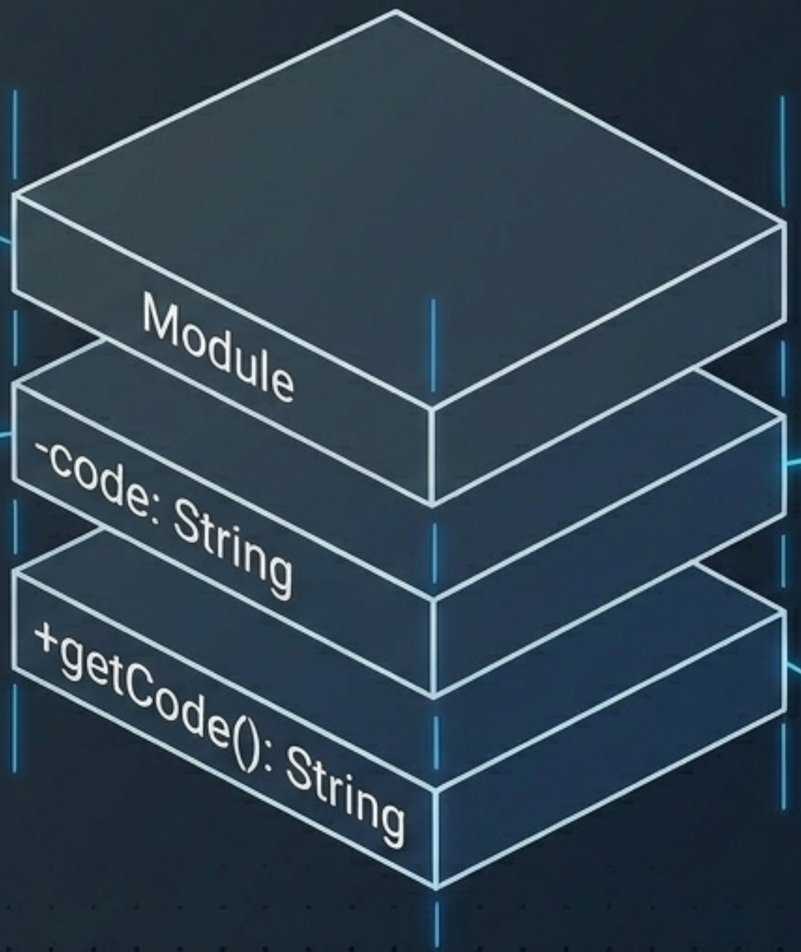


Extra features under certain conditions.
Direction: Extending -> Base.
(注意箭頭方向是反過來的！)

Exam Radar (考核重點)
Ensure you use dashed lines (虛線) with open arrowheads, and correctly spell the stereotypes in guillemets: <<include>> and <<extend>>.

Class Diagrams (類別圖) 系統的核心骨架

類別圖是 OOT 的靈魂。考試必考！你需要找出類別名稱、屬性 (Attributes)、方法 (Methods)，以及物件數量的對應關係。



Class Name (Capitalized)

Attributes (Variables)

Methods (Operations)

Multiplicities

1 (Exactly one)

0..* (Zero or more)

1..* (One or more)

Visibility:
+ Public
- Private
Protected

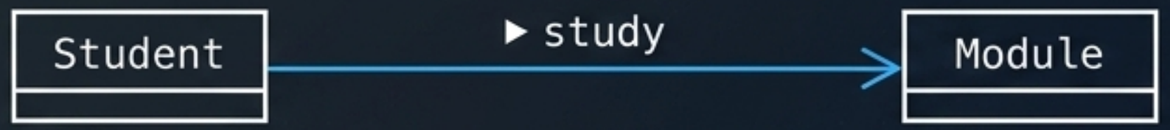
⚠ Exam Radar (考核重點)

In Section A1, you MUST include attributes and exact multiplicities to get full marks.

Navigating Class Relationships: 關聯與繼承

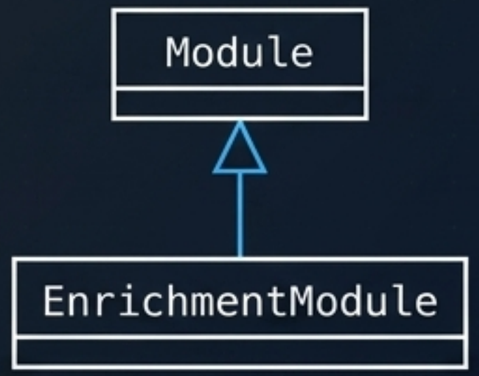
正確分辨 "Has-a" (擁有) 與 "Is-a" (屬於) 的關係，決定了你畫出的線條種類。

Association (關聯) - The 'Has-a' relationship



Must label the line with a descriptive name and navigation arrow.

Generalization (繼承) - The 'Is-a' relationship



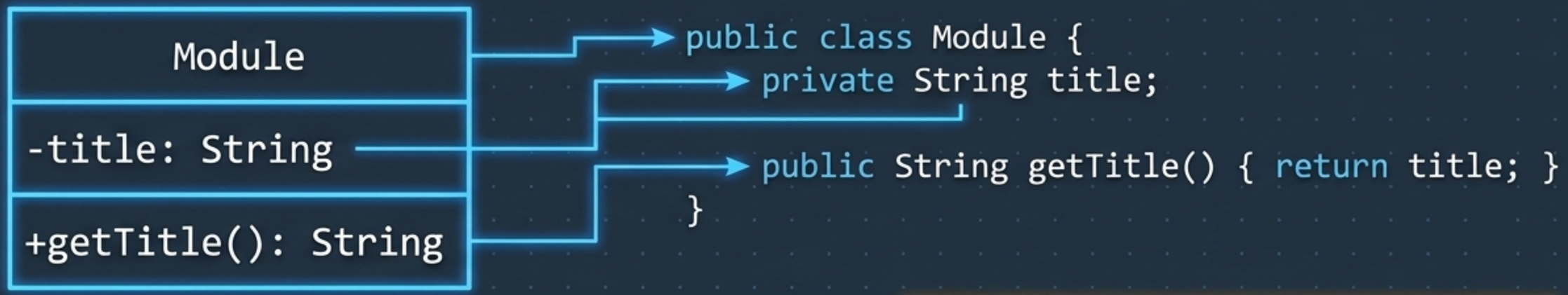
Superclass (Base) at the top, Subclass (Derived) at the bottom.

Exam Radar (考核重點)

Do not confuse the hollow triangle of Inheritance with a basic Association arrow!

UML to Java Translation (翻譯演算法)

這是搶分重地 (Section A2)。不需要創意，只需要將你畫的（或題目給的）UML圖表「翻譯」成 Java 程式碼。

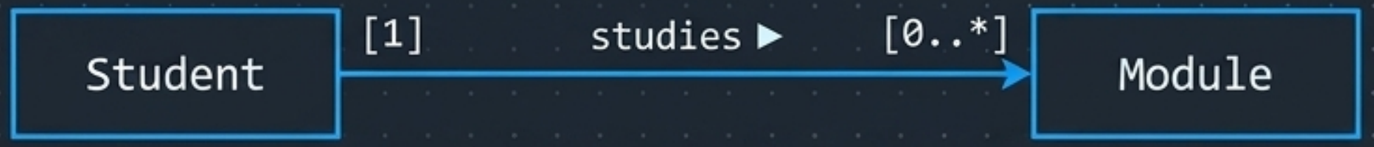


⚠ Exam Radar（考核重點）

Minus sign - always becomes private.
Plus sign + always becomes public.
Data types switch order (e.g., title: String becomes String title).

Coding 1-to-Many Associations in Java

當一個類別擁有多個另一類別的物件時 (例如 1..* 或 0..*)，我們在 Java 必須使用 Vector 或陣列 (Array) 來實作關聯。




```
// Declaration
private Vector _modules;

// Inside Constructor: Initialization
_modules = new Vector();

// Inside Method
public void addModule(Module module) { _modules.add(module); }
```

Exam Radar (考核重點)

Always remember to initialize the Vector inside the constructor using `new Vector()`. Without this, your code will crash with a `NullPointerException`!

Implementing Inheritance & Abstract Classes

繼承的語法及抽象類別的定義是考試的進階得分點。熟記 extends, super(), 和 abstract 的用法。

Abstract Class Syntax

```
public abstract class Plan {
  ...
  public abstract double getPlanCharge();
}
```

Note: No {} body!

Subclass Implementation

```
public class DailyPlan extends Plan {
  public DailyPlan(String planId...) {
    super(planId, planName);
  }
}
```

MUST BE FIRST LINE

⚠ Exam Radar (考核重點)

When creating a subclass constructor, the super() call to the parent constructor must be the absolutely first line of code inside the block!

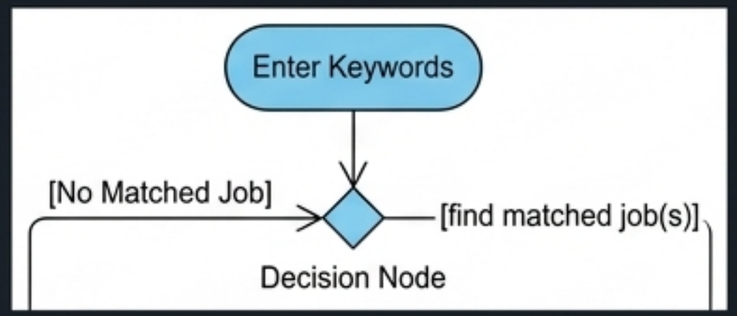
Behavioral Diagnostics: Activity vs State Machine

同學最常混淆這兩種圖表 (Section A3)。Activity Diagram 像是流程圖，注重「動作與決策」；State Machine Diagram 則專注於單一物件的「狀態轉變」。

Activity Diagram (流程主導)

Shapes: Ovals are Actions (e.g., Enter Keywords). Diamond is Decision Node.

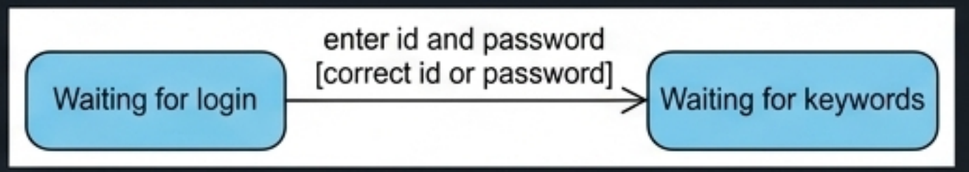
Flow: Control Flow arrows mapping the steps.



State Machine Diagram (狀態主導)

Shapes: Rounded rectangles are States (e.g., Waiting for keywords).

Flow: Transitions triggered by Events. Uses Guard Conditions like [correct login].

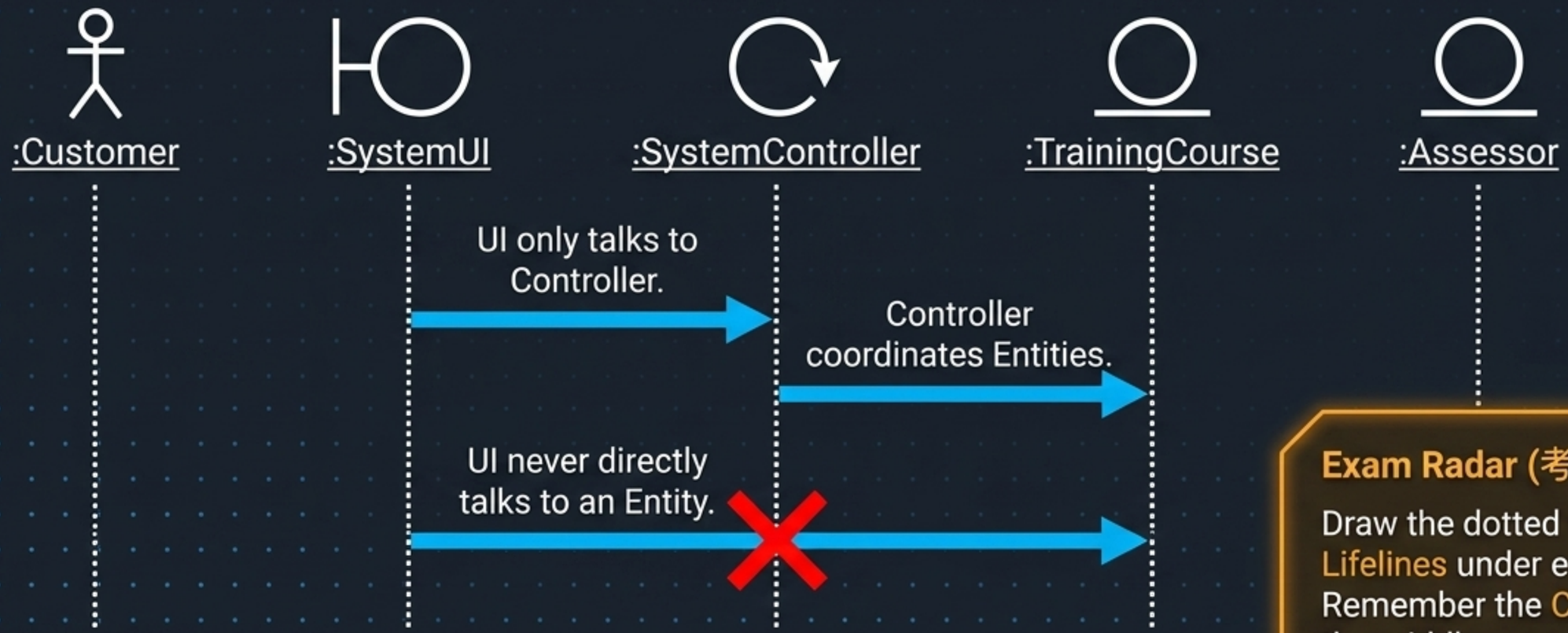


Exam Radar (考核重點)

In State Machine Diagrams, conditions must be enclosed in brackets [ ] (Guard Conditions), and you must clearly label the Event triggering the line.

Sequence Diagrams (順序圖) 揭密 MVC 架構

順序圖展示物件之間隨時間發生的訊息傳遞。考試長題目必定要求 Model-View-Controller (MVC) 三層架構。嚴格遵守溝通規則！

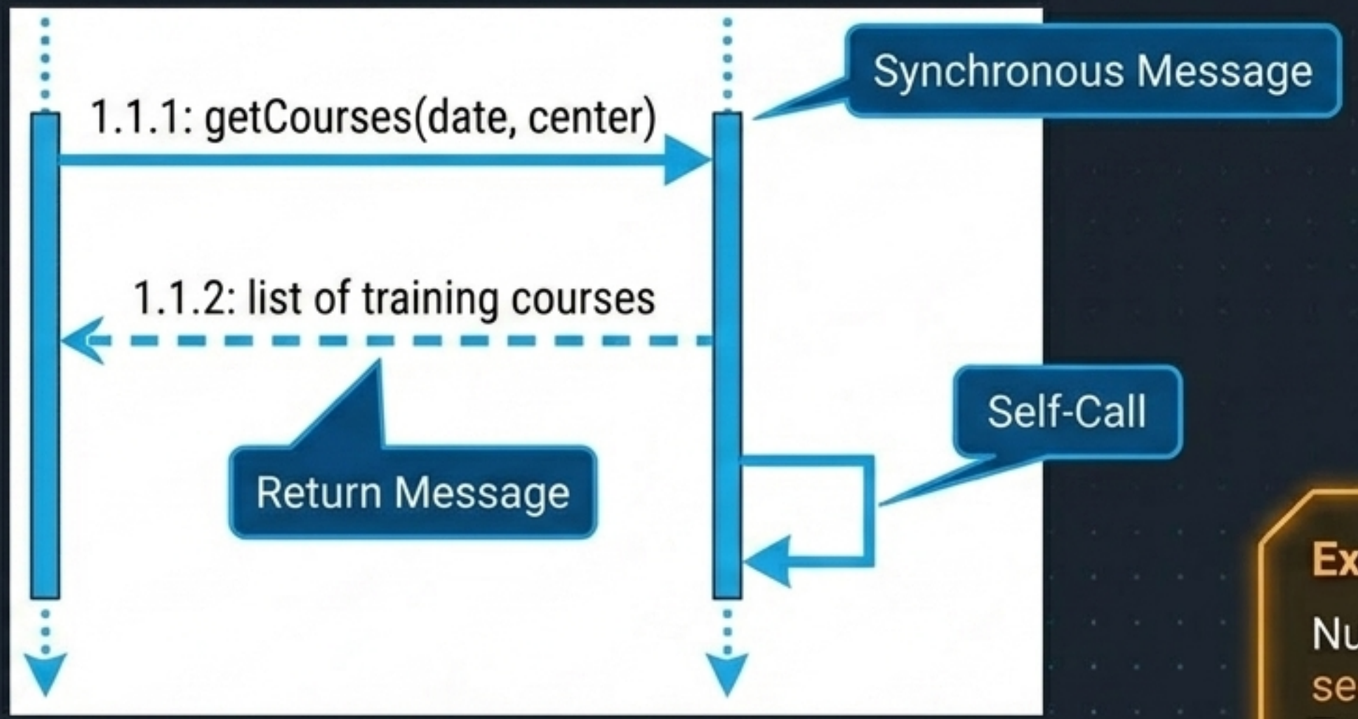


Exam Radar (考核重點)

Draw the dotted vertical Lifelines under every object. Remember the Controller is the middle-man.

Mapping Flow of Events to Messages

順序圖的時間軸由上而下。逐行閱讀題目的 Flow of Events（事件流程），每一個用戶動作都是傳給 UI 的訊息，每一次系統回應都是 Return Message。

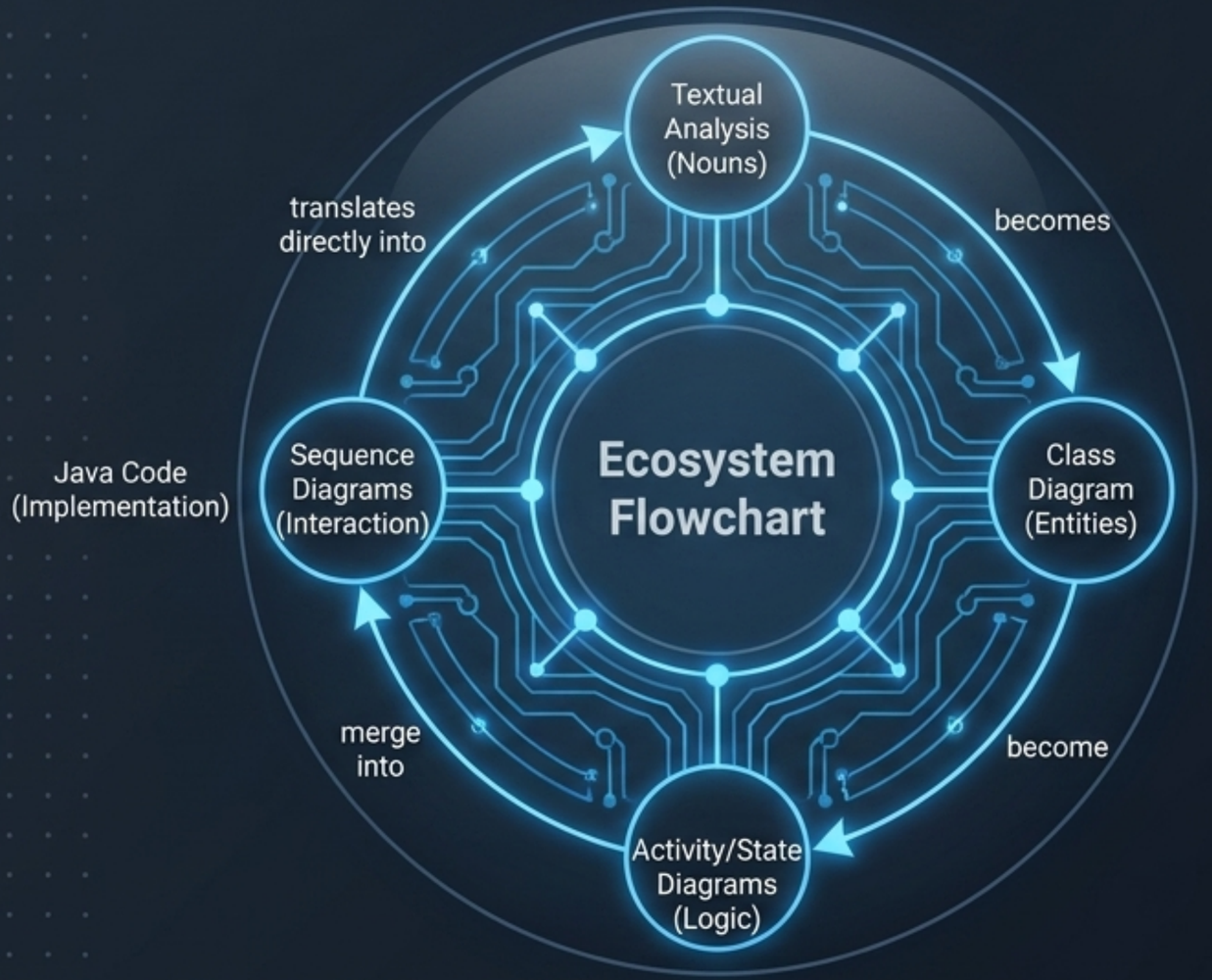


Exam Radar (考核重點)

Number your messages sequentially (1, 1.1, 1.1.1). Follow the explicit flow provided in the exam text word-for-word.

Synthesis: The OOT Blueprint Connected

這些圖表不是獨立的！它們從不同的維度 (Dimension) 描述同一個軟體系統。
名詞變成類別，動詞變成方法。



Professor's Note

Software engineering is just translating human needs into structured diagrams, and diagrams into code.

Professor OOT's Final Exam Checklist

進入考場前，在腦海中快速過濾這份清單，確保不再犯低級錯誤，穩拿 A 級分數！祝考試順利！

- [ ] **Multiplicities:** Are **1..*** / **0..*** included on EVERY Class Diagram association?
- [ ] **Guard Conditions:** Are conditions enclosed in **[ ]** in State Diagrams?
- [ ] **Java Inheritance:** Is **super()** used correctly as the first line in Java subclass constructors?
- [ ] **Vectors:** Did I use **new Vector()** in my Java constructor for 1-to-Many associations?
- [ ] **System Boundaries:** Is the box drawn and are Actors placed OUTSIDE for Use Cases?

Exam Radar (考核重點)

Print this framework in your mind. You are now ready to design and code like a true Software Architect.