UML Class Diagram

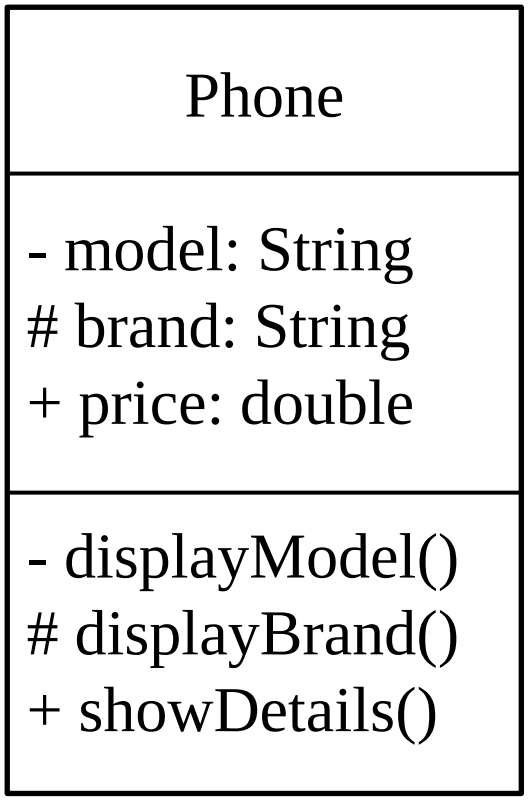
Shin-Jie Lee (李信杰)
Associate Professor
Dept. of CSIE
National Cheng Kung University

Class

- A class is a definition of the behavior of an object, and contains a complete description of the following:
  - The data elements (variables) the object contains
  - The operations the object can do
  - The way these variables and operations can be accessed
- *Objects are instances of classes*
- Creating instances of a class is called *instantiation.*

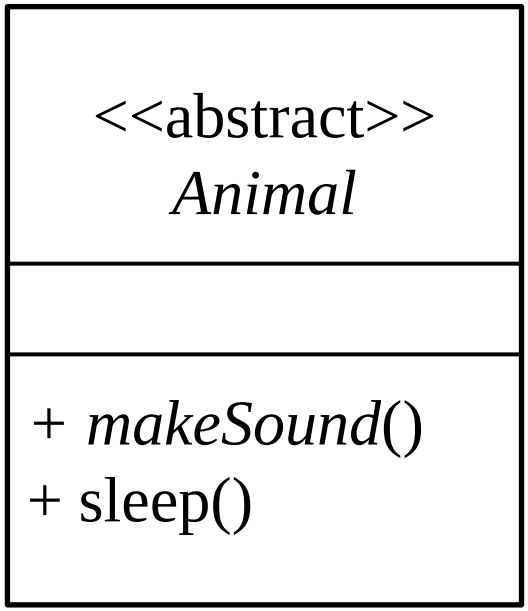
Class Notation

- ❑ - 代表private
- ❑ # 代表protected
- ❑ + 代表public

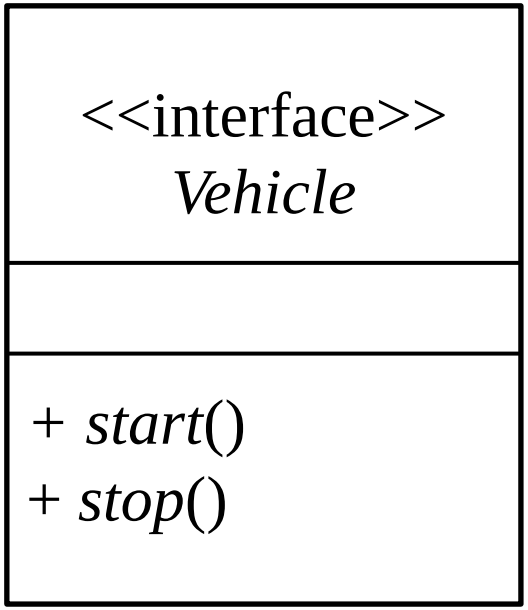


Abstract Class

- 通常Abstract Class Name與Abstract Method以斜體字表示，但有些UML工具仍以正體字表示並加上<<abstract>>字樣



Interface

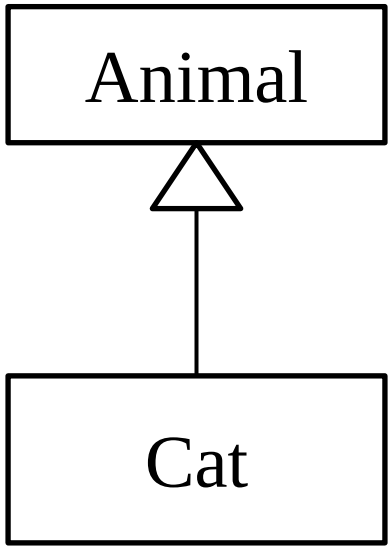


Relationship

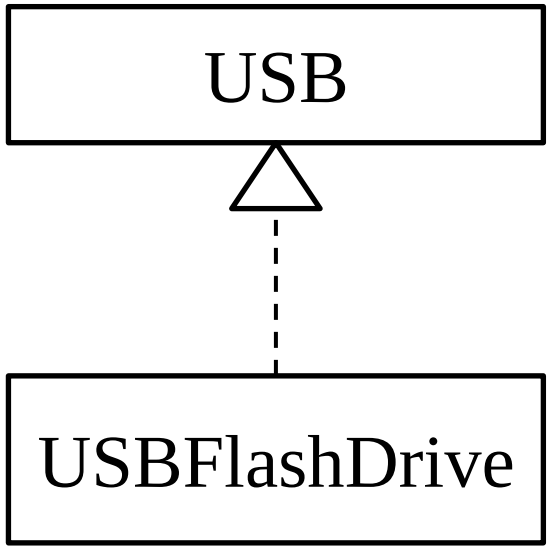
- Inheritance
- Implementation
- Dependency
- Association
  - Aggregation
  - Composition

Generalization

- 層次結構的關係
  - 口訣：檢查兩個Class間是否存在「is-a」關係



Inheritance



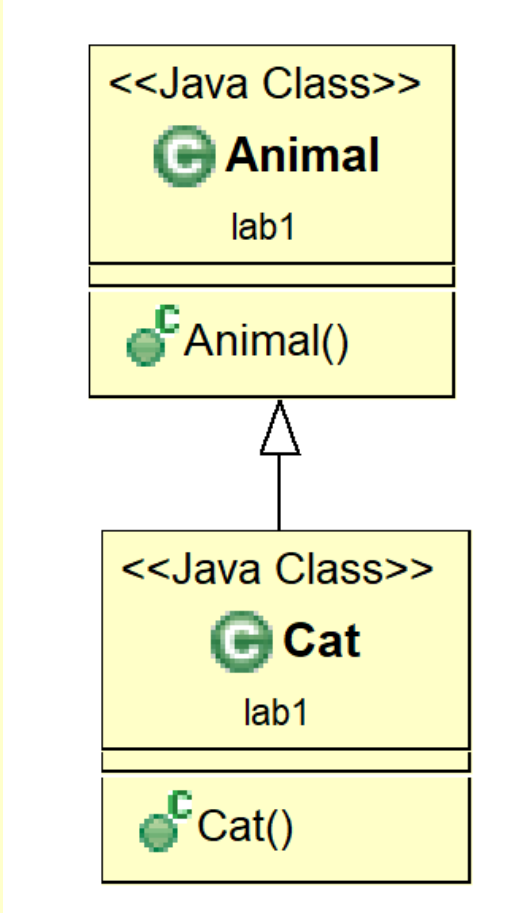
Implementation

Lab

❑ 請繪製以下Class Diagram

```
public class Animal {

}
```

```
public class Cat extends Animal{

}
```

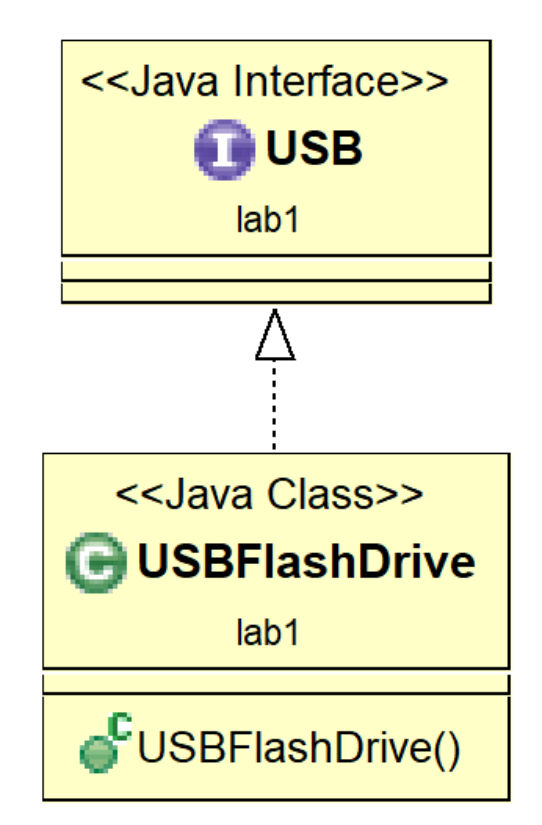


Lab

- 請繪製以下Class Diagram

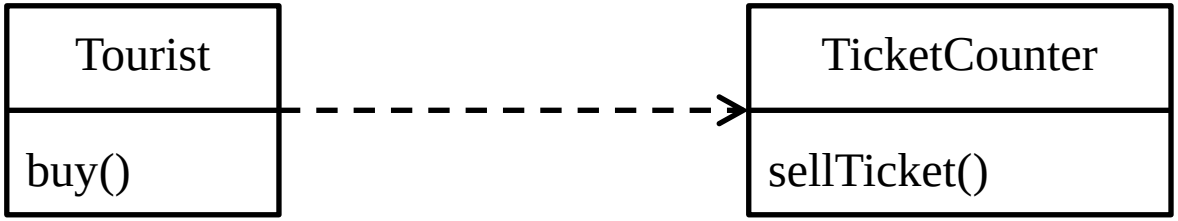
```
public interface USB {

}
```

```
public class USBFlashDrive implements USB{

}
```



Dependency

- 短期、臨時性的關係
  - 程式碼通常以Method的Parameters或Local Variable表示
  - 口訣：檢查兩個Class間是否存在「uses-a」關係




```
public class Tourist {

  public void buy(TicketCounter tc) {
    tc.sellTicket();
  }
}
```

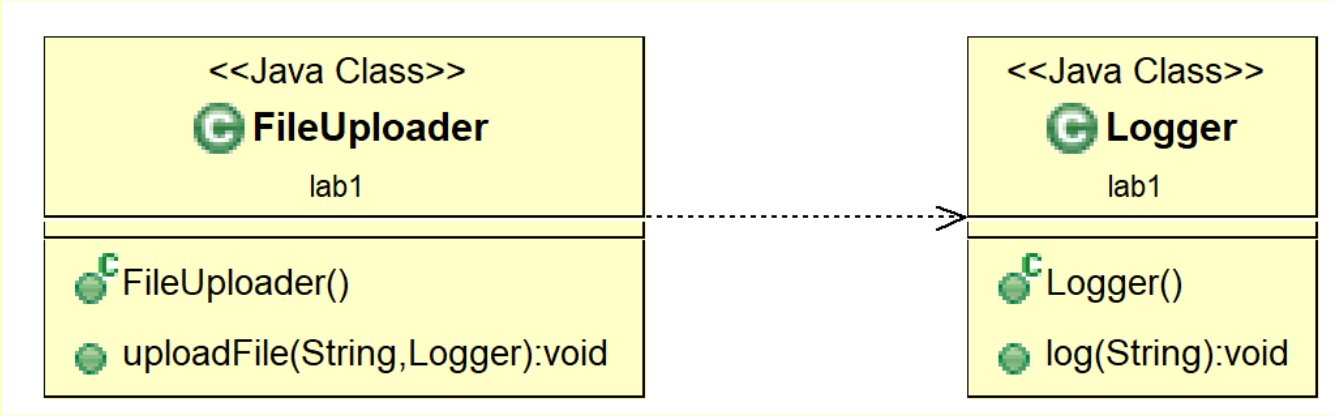
```
public class TicketCounter {

  public String sellTicket() {
    return "Random Ticket No.";
  }
}
```

Lab

- 請繪製以下Class Diagram

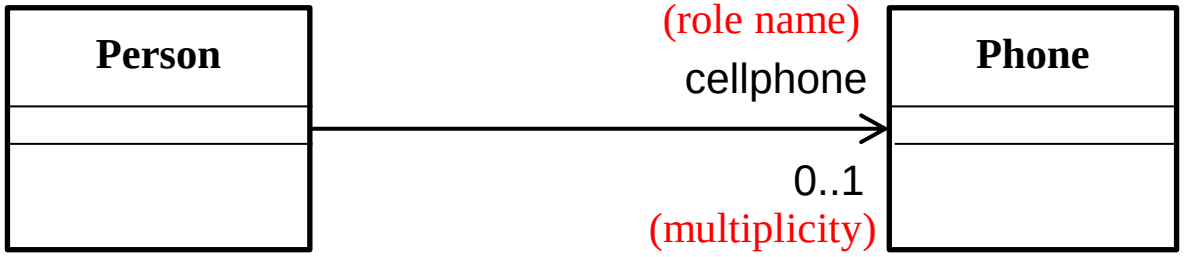
```java
public class FileUploader {

  public void uploadFile(String fileName, Logger logger) {
    logger.log(fileName + " has been uploaded.");
  }
}
```

```java
public class Logger {

  public void log(String message) {
    System.out.println("Log: " + message);
  }
}
```



Association

- 長期、結構性的關係
  - 程式碼通常以Attribute表示
  - 口訣：檢查兩個Class間是否存在「has-a」關係




```java
public class Person {

  private Phone cellphone;

  public void talkTo(String no) {
    cellphone.dial(no);
  }
}
```

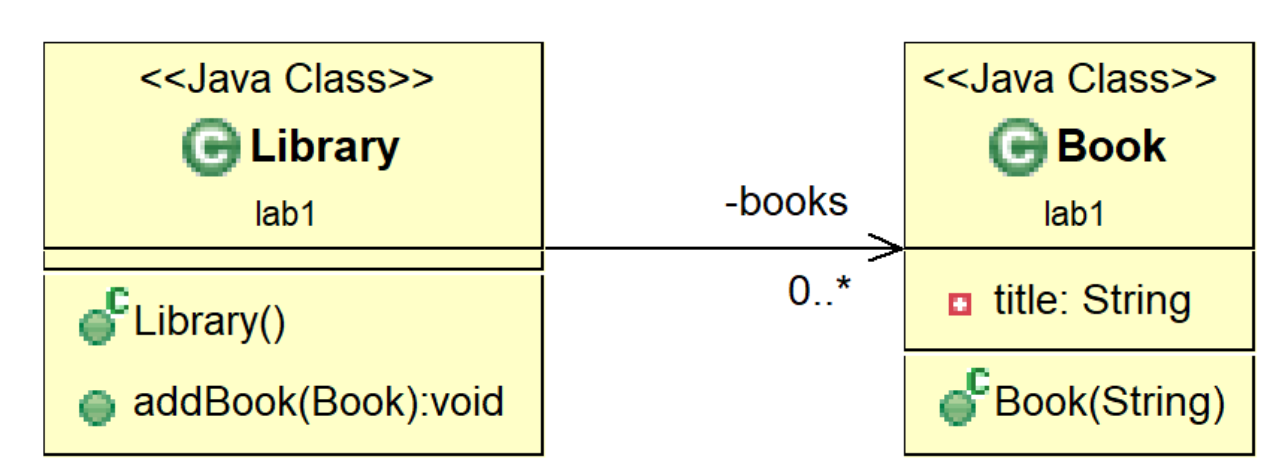
```java
public class Phone {

  public void dial(String no) {
    // Dial someone
  }
}
```

Lab

- 請繪製以下Class Diagram

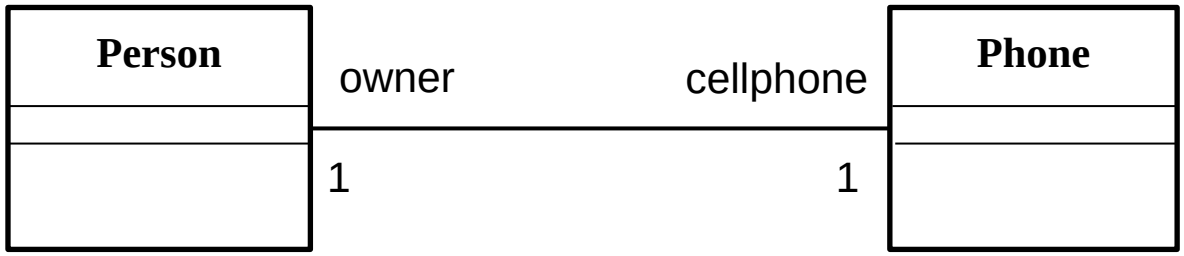
```java
import java.util.ArrayList;
public class Library {
  private ArrayList<Book> books = new ArrayList<Book>();

  public void addBook(Book book) {
    books.add(book);
  }
}
```

```java
public class Book {
  private String title;
}
```



雙向關係

- 雙向關係(Bidirectional Association)通常化簡為一條無箭頭的直線
  - 注意：但自動化工具可能仍只支援繪製出兩條箭頭直線




```java
public class Person {

  private Phone cellphone;

  public void talkTo(String no) {
    cellphone.dial(no);
  }
}
```

```java
public class Phone {

  private Person owner;

  public void dial(String no) {
    // Dial someone
  }
}
```

many-to-many雙向關係

- 會有什麼缺點?

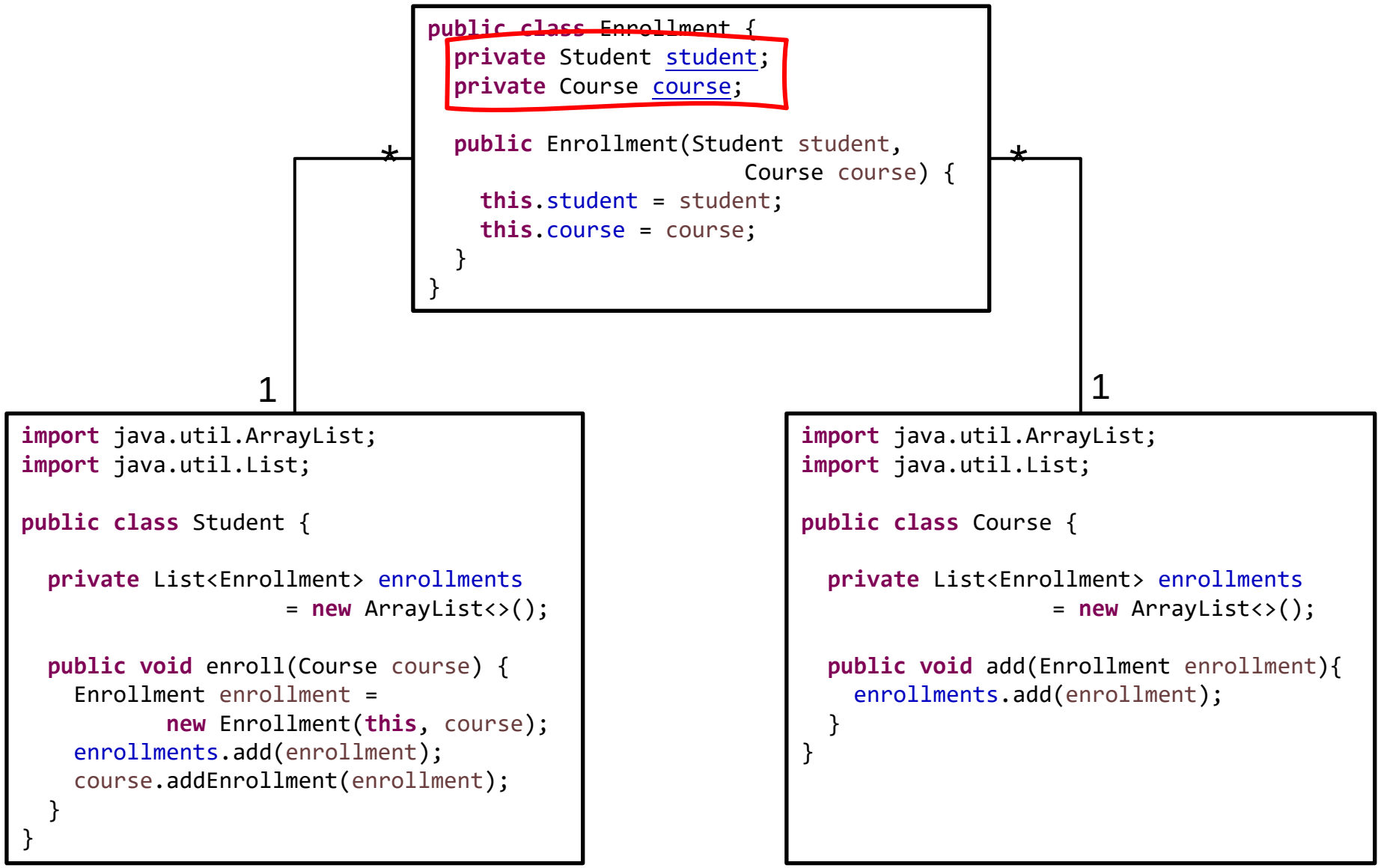

```java
import java.util.ArrayList;
import java.util.List;

public class Student {

  private List<Course> courses =
                    new ArrayList<>();

  public void enroll(Course course) {
    courses.add(course);
  }
}
```

```java
import java.util.ArrayList;
import java.util.List;

public class Course {

  private List<Student> students =
                    new ArrayList<>();

  public void add(Student student) {
    students.add(student);
  }
}
```

Lab

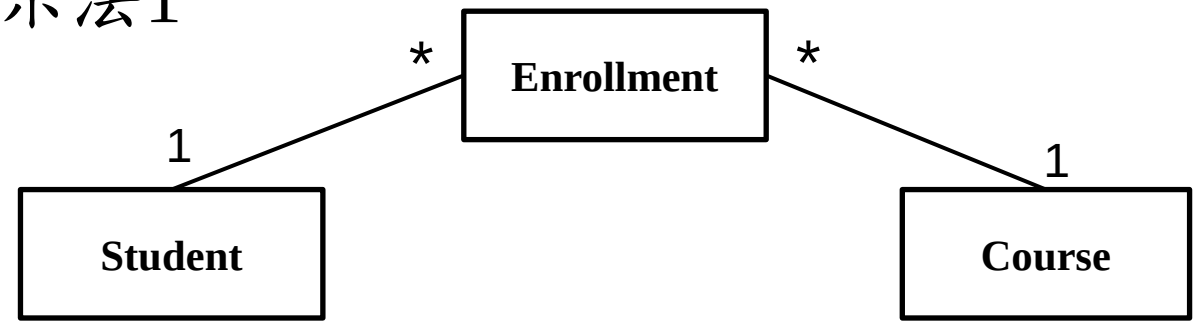
- 當要記錄修課成績時該怎麼辦?
  - 請修改上頁程式碼達成此需求

Association Class
(many-to-many關係的改良呈現方式)

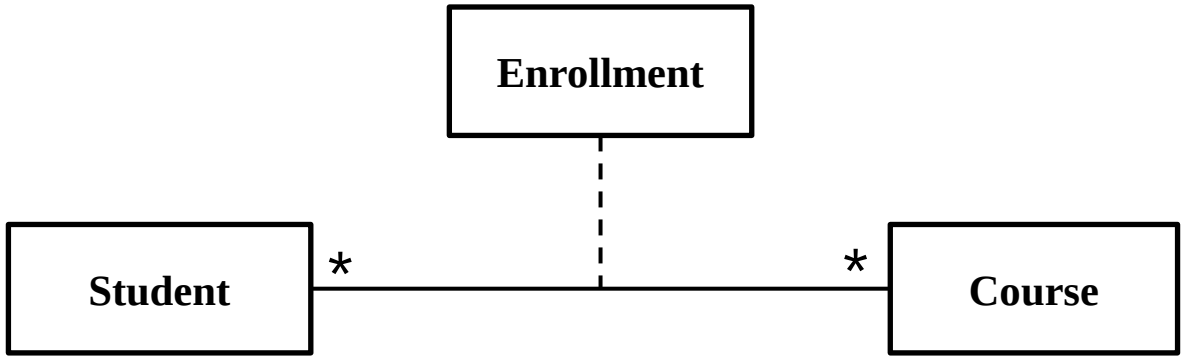


Association Class

- 表示法1

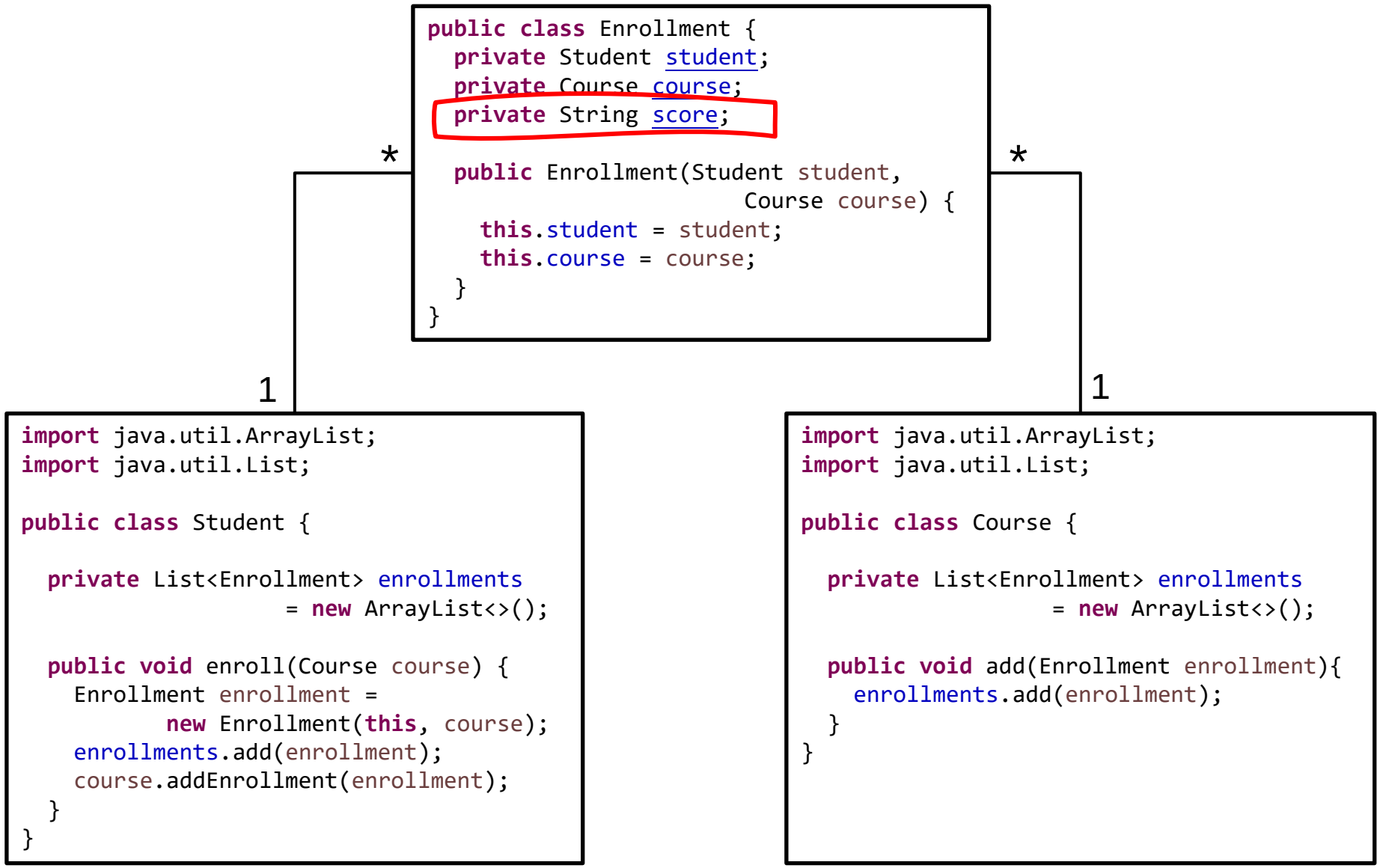


- 表示法2



Association Class優點

- 允許你在關聯上面增加屬性。

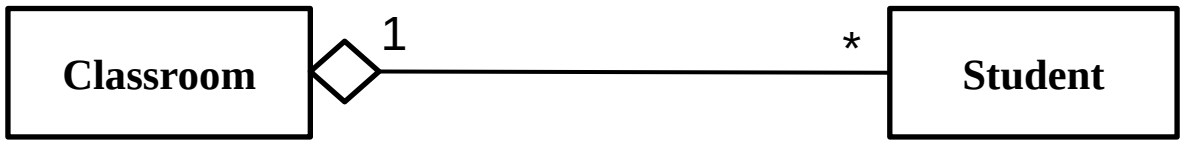


一定得用Association Class嗎?

- 不一定
  - 若不需要額外的多對多關係的管理。比如，Product 和 Category 的關係，每個產品可以屬於多個類別，而每個類別也可以包含多個產品。如果只需要查詢關聯，而不需要存儲其他資訊(Attributes)，則可以不使用 Association Class。

Aggregation

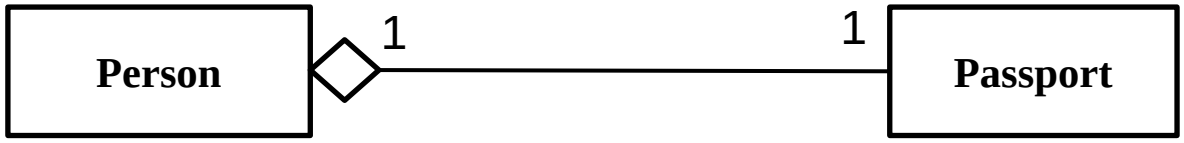
- Aggregation是一種特殊的Association，表示一個類別「包含/擁有」另一個類別
- Whole物件消失，Part物件仍可繼續存在，所以是「較弱」的聚合(Whole-Part)關係
- 以「空心菱形」表示



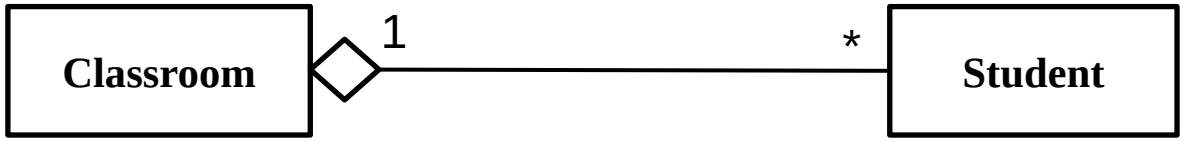
即使班級不存在，學生也能在其他班級中繼續存在

Aggregation (Multiplicity)

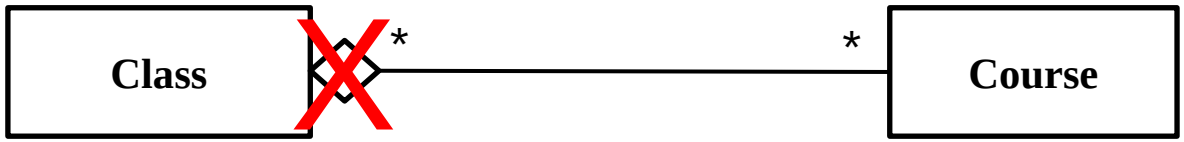
- 1對1



- 1對多 (最常見)



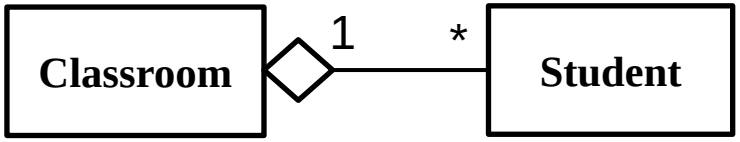
- ~~多對多~~ (通常直接使用Association，不加菱形)



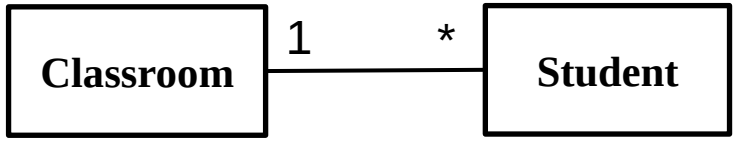
開發者很少使用 Aggregation？

- Aggregation大多僅使用在**設計階段**明確表達某個類別「包含/擁有」其他類別，以**促進口語化的溝通理解**
- 但**實作階段**時，Aggregation 和 Association幾乎一樣
- 結論：可用Aggregation，也可不用

溝通時看到此設計圖會唸出：
「一個Classroom包含許多學生」



溝通時看到此設計圖會唸出：
「一個Classroom關連到許多學生」




```
class Classroom {
  private List<Student> students;
}
```

實作時卻無差別

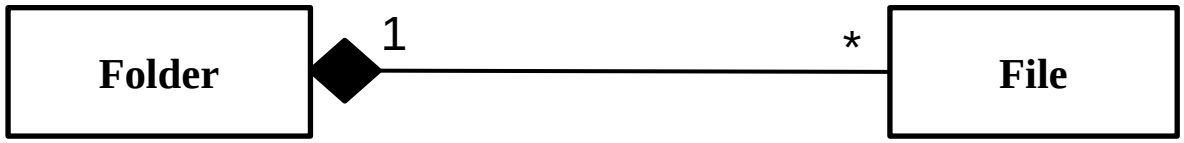
```
class Classroom {
  private List<Student> students;
}
```

Lab

- 請舉出Aggregation的例子

Composition

- Composition也是一種特殊的Association，表示一個類別「強烈」「包含/擁有」另一個類別
- Whole物件消失，Part物件則不可繼續存在
- 以「實心菱形」表示



一個Folder物件不存在，其包含的File也不能繼續存在

Composition實作

- 實作的方式並沒有標準規範，只需滿足Composition的概念即可
- 常見的實作方式
  - 將Part類別封裝在Whole類別內
  - 不提供getter，確保其他物件不持有 Part物件的Reference
  - 使用WeakReference

Composition實作1

- 將Part類別封裝在Whole類別內


```
class Car {
  private Engine engine;

  public Car() {
    this.engine = new Engine();
  }

  private class Engine {
  }
}
```

Composition實作2

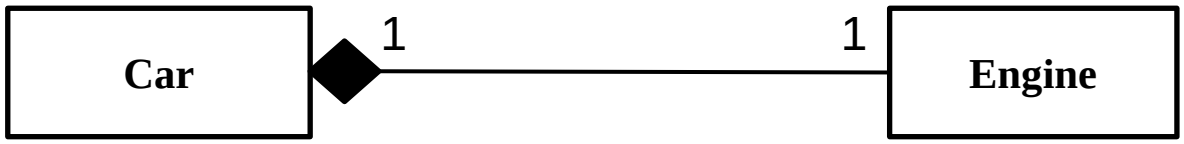
- 不提供getter，確保其他物件不持有 Part物件的 Reference


```
class Car {
  private Engine engine;
  public Car() {
    this.engine = new Engine();
  }
  public Engine getEngine() {
    return engine;
  }
}
```

```
class Engine {

}
```

Composition實作3

- 使用WeakReference




```
import java.lang.ref.WeakReference;

class Car {
  private Engine engine;

  public Car() {
    this.engine = new Engine();
  }

  public WeakReference<Engine> getEngineReference() {
    return new WeakReference<>(engine);
  }
}
```

```
class Engine {

}
```

Tips

- Code-to-UML同步工具
  - 由於Aggregation與Composition在實作上並沒有標準寫法，所以Code-to-UML同步工具很難由Code判定為Aggregation與Composition，大多以Association來取代
- UML-to-Code同步工具
  - 但若你是採用UML-2-Code工具，建議還是將Aggregation、Composition與Association區隔開來

Lab

- 請將以下程式碼以ObjectAid轉換為Class Diagram，看看是否繪製出Association (取代Composition)?

```
import java.lang.ref.WeakReference;

class Car {
  private Engine engine;

  public Car() {
    this.engine = new Engine();
  }

  public WeakReference<Engine> getEngineReference() {
    return new WeakReference<>(engine);
  }
}
```

```
class Engine {

}
```

ObjectAid可繪製出的Relationship

- Inheritance
- Implementation
- Dependency
- Association
  - ~~Aggregation~~
  - ~~Composition~~

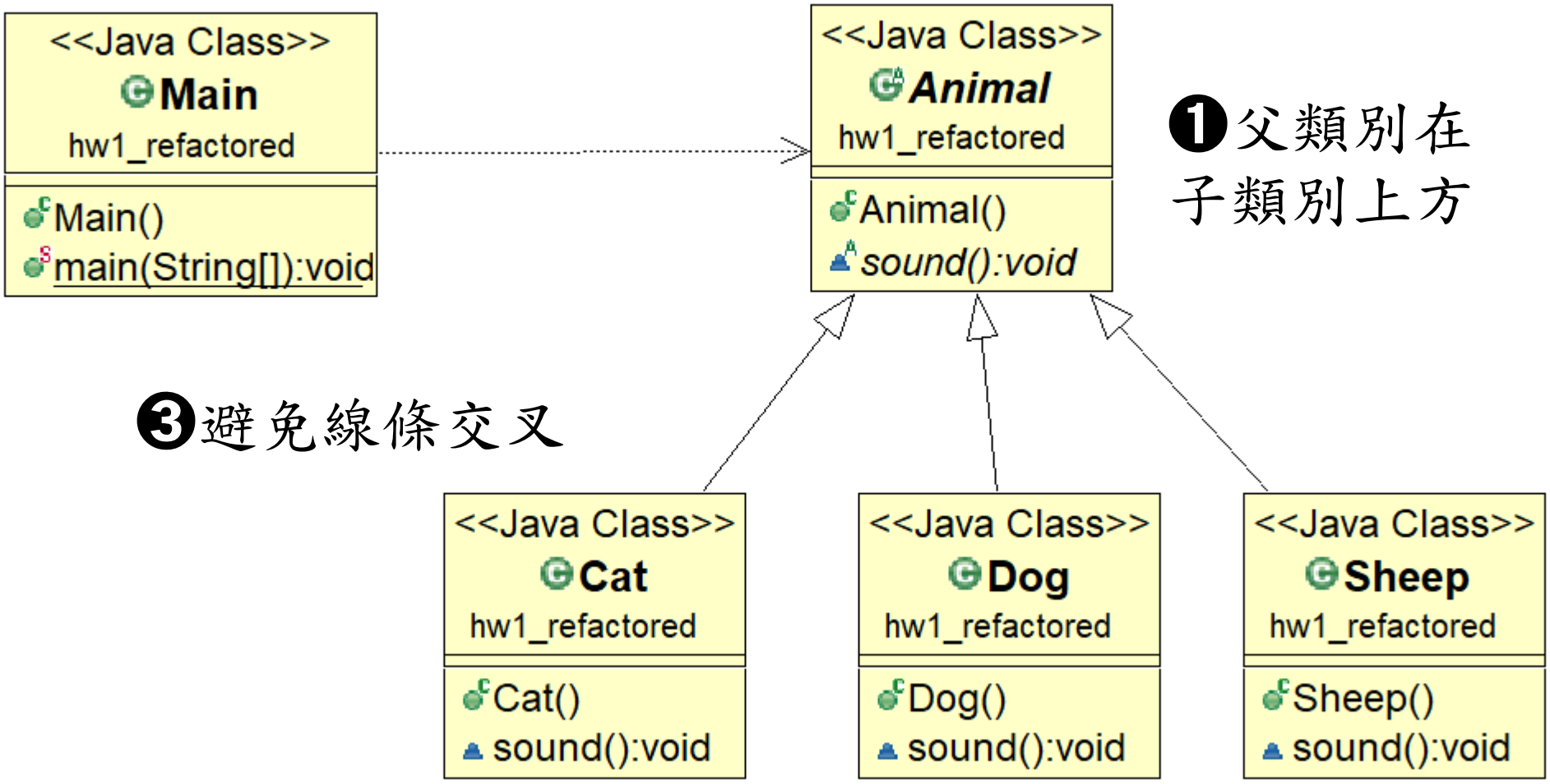
Lab

請繪製出以下Class Diagram

- A country has a capital city.
- A dining philosopher is using a fork.
- A file is an ordinary file or a directory file.
- Files contain records.
- A polygon is composed of points.
- A drawing object is text, a geometrical object, or a group.

Layout Guideline

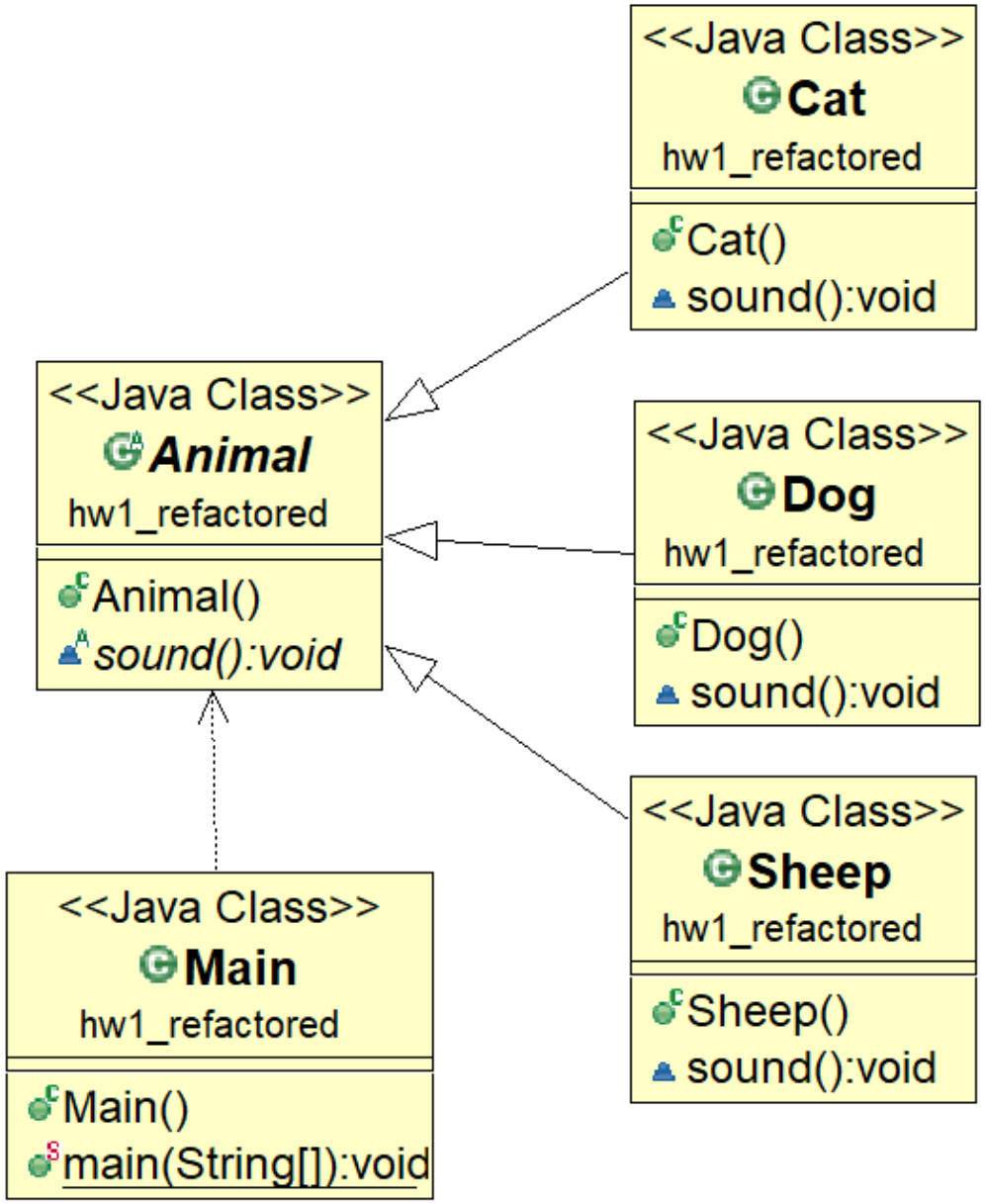
❷盡量主類別在左側，
被依賴/關聯在右側



❶父類別在
子類別上方

❸避免線條交叉

Layout反例


<<Java Class>>
Cat
hw1_refactored
Cat()
sound():void
<<Java Class>>
Animal
hw1_refactored
Animal()
sound():void
<<Java Class>>
Dog
hw1_refactored
Dog()
sound():void
<<Java Class>>
Sheep
hw1_refactored
Sheep()
sound():void
<<Java Class>>
Main
hw1_refactored
Main()
main(String[]):void

Lab (AI協助產生Class Diagram)

- 一個圖書館希望建立一個管理系統，以便有效地管理書籍、讀者和借閱記錄。系統需要能夠跟蹤書籍的可用性、讀者的資訊及其借閱歷史
- 書籍（Book）每本書應有標題、作者、ISBN編號、出版年份和可用數量。書籍可以被借出或歸還。
- 讀者（Reader）每位讀者應有姓名、讀者ID、註冊日期和聯絡電話。讀者可以借出書籍，並且每位讀者最多可以同時借出三本書。
- 借閱記錄（Borrowing Record）每筆借閱記錄應包含借出日期、歸還日期及與之相關的書籍和讀者資訊。
- 系統應能顯示每位讀者的借閱歷史。圖書館（Library）圖書館擁有多本書籍和多位讀者。圖書館可以管理書籍的借出和歸還流程。

Lab (AI協助產生Class Diagram)

- 請使用AI工具協助將上述需求轉換為Class Diagram
  1. 先生成各類別屬性與方法
  2. 再生成Java 程式碼
  3. 再使用ObjectAid繪製出Class Diagram
  4. 調整Class Diagram的Layout