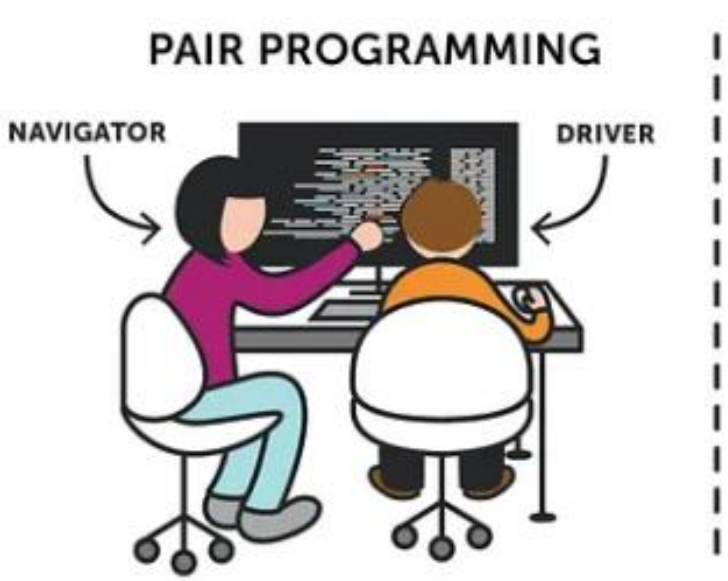
TAIPEI TECH 國立臺北科技大學 National Taipei University of Technology

Object Oriented Programming

Code Section: Encapsulation

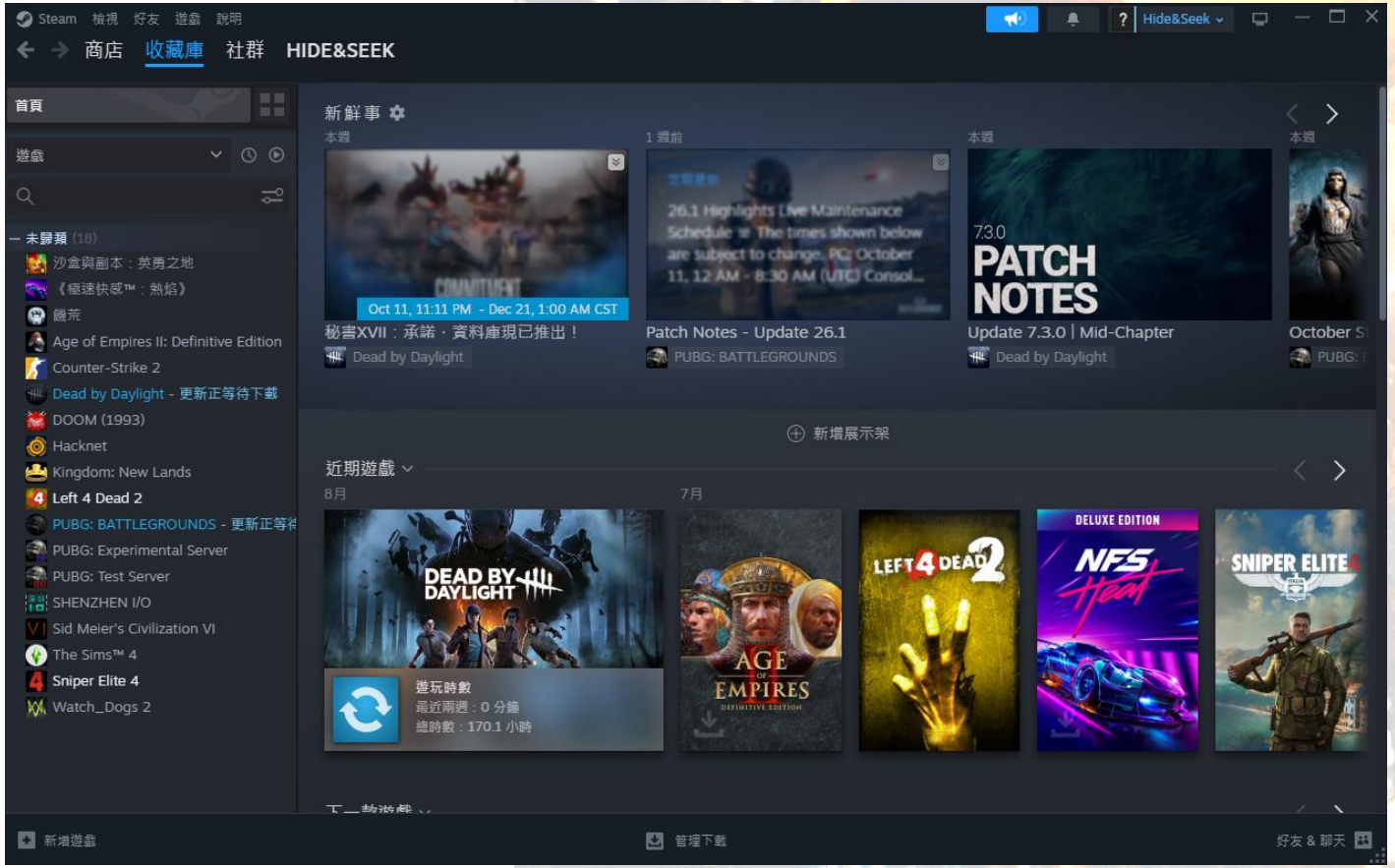
Sun Chin-Yu（孫勤昱）
cysun@ntut.edu.tw
2023/10/16



Information Security LAB

Agenda

- 創建一個小型遊戲庫系統，可以讓使用者管理、搜索和查看他們的遊戲集合
  - 遊戲的結構
  - 遊戲庫的類別
  - 類別中的方法實作
  - 主函式



目標(一)：實現遊戲的結構

- 遊戲的結構
  - 包含遊戲的名稱、玩家玩過的時間和遊戲的價格
    - Elden Ring（艾爾登法環）, 100（小時）, 1790（台幣）
    - PUBG（絕地求生）,3000（小時）, 799（台幣）
  - 使用Struct架構

Game.hpp

```cpp
#ifndef GAME_HPP
#define GAME_HPP

#include <string>

struct Game {
    std::string title;
    float playTime;
    int price;
};

#endif
```

目標(二)：設計API

- 我們希望Library能夠完成哪些基本操作?
  - 儲存遊戲（使用Vector）
- 有哪些功能是使用者最可能用到的?
  - 添加遊戲
  - 搜尋遊戲
  - 印出遊戲庫中所有的遊戲

Library.hpp

- <initializer_list>
  - std::initializer_list 是一種容器，它使用大括號 {} 初始化一個給定類型的陣列
  - std::initializer_list 通常用於容器、陣列或類別的初始化，使得我們可以使用一個乾淨、簡潔的語法來初始化物件
- <optional>
  - std::optional 是C++17中一個非常有用的功能，它讓Function的回傳值多了一個選擇：
    - 有值
    - 沒有值（nullopt狀態）
  - 常用於在不使用特殊值或拋出異常的情況下表示操作的失敗

```cpp
#ifndef LIBRARY_HPP
#define LIBRARY_HPP

#include <initializer_list>
#include <optional>
#include <string>
#include <vector>
#include "Game.hpp"

class Library {
public:

};
#endif
```

https://en.cppreference.com/w/cpp/utility/initializer_list
https://en.cppreference.com/w/cpp/utility/optional

補充：std::

- 表示正在使用來自標準庫（Standard Library）的 vector、string 和 optional 類別
  - 沒有 std::，編譯器就不知道vector、string或optional是什麼
  - 除非使用了像這樣的聲明：using namespace std;

- 這樣的聲明會將std名稱空間中的所有名稱引入到當前的名稱空間，允許在不加 std:: 前綴的情況下使用標準庫的名稱

- 許多開發者和團隊**避免**使用using namespace std;
  - 因為這樣做會在較大的專案或者多模塊專案中帶來名稱衝突的風險，例如
    - 如果你自定義了一個名為vector的類別或函數
    - 而你（或他人）使用了using namespace std
    - 那麼你自己的vector程式碼和標準庫中的名稱就會發生衝突
    - 這可能導致編譯錯誤或者更難以追踪的運行時錯誤

目標(三)：類別中的方法實作

```
class Library {
public:
    Library(std::initializer_list<Game> games = {});
    ~Library();

    void AddGame(Game game);
    std::optional<Game> FindGameByName(std::string title);
    std::vector<Game> GetGames();

private:
    std::vector<Game> m_Games;
};
#endif
```

①

②

③

④

⑤

實作 ① ② ③ ④ ⑤

目標(四)：設計主函式

- 給定一個main.cpp
  1. 建立一個遊戲庫
     - 包含兩款遊戲
  2. 在遊戲庫中搜尋一款購買過的遊戲
     - 印出 “Game found”
     - 顯示遊戲庫中所有的遊戲名稱
  3. 在遊戲庫中搜尋一款未購買的遊戲
     - 印出 “Game not found”
     - 添加新遊戲進入遊戲庫中
     - 顯示遊戲庫中所有的遊戲名稱

Code Review

討論：FindGameByIndex

- Library.hpp

```
27      std::optional<Game> FindGameByIndex(std::size_t index);
```

- Library.cpp

```
27  std::optional<Game> Library::FindGameByIndex(std::size_t index) {
28      if (index >= m_Games.size()) {
29          return std::nullopt;
30      }
31
32      return m_Games[index];
33  }
```

Leaky abstraction

- 如果我們從"封裝"與"抽象"的角度來看，Library類別應該由一個或多個API組成
  - 使用者不需要知道底層如何存儲這些遊戲數據
- 當我們提供一個方法如 FindGameByIndex
  - 其實透露了一些底層的實現細節：也就是遊戲對象是按照某種順序存儲在一個可索引的資料結構中（例如，一個陣列或一個std::vector）
- 破壞「封裝」
  - 正確的封裝應該能夠隱藏內部的實現細節，只通過一個精心設計的API與外界交互
  - （惡意）使用者可能開始根據這個具體的實現來試圖通過直接計算索引來查找或操作遊戲對象

如何解決?

- 設計FindGameByName的API
- 還有沒有其他更好的方式?
  - 不使用index number
  - 又可以降低搜尋時間複雜度

討論：std::unordered_map

- 元素間沒有順序

Key (鍵)	Value (值)
ABC	Game{“ABC", 999, 199}
PUBG	Game{"PUBG", 3000, 799}
Elden Ring	Game{"Elden Ring", 100, 1790}

- 這個資料結構支持高效的查找（.find）、插入（.insert）、和刪除（.erase）操作
  - O(1) 時間複雜度（avg. time）

- https://en.cppreference.com/w/cpp/container/unordered_map

討論：封裝

- Class Library 的定義和實現
  - Public/Private
  - Getter/Setter
- 設計API時，您應該優先考慮使用者的需求和期望。
  - 最佳的API設計應該是直觀的，不需要使用者了解背後的實現細節就能使用
  - FindGameByName/FindGameByIndex (Leaky abstraction)
- 資料結構
  - std::vector是有順序的
  - std::unordered_map 這樣的無順序結構時，「Index」的概念就變得模糊了