دليل شامل - شرح ملفات Library

🎯 الفكرة العامة

ملفات Library مسؤولة عن بناء وإدارة المكتبة الكاملة. المكتبة عبارة عن شبكة معقدة من الكتب المترابطة:

- **كتب مداخل** (نقاط البداية) **4**
- **كتاب وسطي** (المسارات المتنوعة) **10-15**
- **كتاب نهائي واحد** (الهدف النهائي)

كل الكتب دي متربطة مع بعضها بطريقة ذكية عشان تخلق تجربة لعب متنوعة!

📂 ملف Library.h (ملف الهيدر)

1️⃣ المتغيرات الخاصة (Private)

الكتب الأساسية

```cpp
BookNode* entranceBooks[4];     // مصفوفة من 4 مؤشرات لكتب المداخل
vector<BookNode*> middleBooks;  // قائمة ديناميكية للكتب الوسطية
BookNode* finalBook;            // مؤشر للكتاب النهائي الوحيد
```

رسم توضيحي للبنية:

```
[كتاب مدخل 1]
[كتاب مدخل 2]
[كتاب مدخل 3]  ──→  [كتب وسطية 10-15]  ──→  [كتاب نهائي]
[كتاب مدخل 4]
```

لماذا استخدمنا Vector للكتب الوسطية؟

- العدد متغير (10-15) فنحتاج مرونة
- نقدر نضيف ونمسح بسهولة
- أسهل في التعامل من المصفوفات الثابتة

الألغاز المتاحة

```cpp
```

```
vector<Puzzle> allPuzzles;    // قائمة كل الألغاز المتاحة
int nextPuzzleIndex;          // مؤشر يتتبع اللغز الجاي
```

الفكرة: بدل ما نكرر نفس الألغاز، المكتبة بتوزعهم بالترتيب على الكتب.

---

2 الدوال الخاصة (Private Functions)

هذه الدوال داخلية تستخدمها المكتبة لبناء نفسها:

```
cpp

void createEntranceBooks();     // إنشاء 4 كتب مداخل
void createMiddleBooks();       // إنشاء 10-15 كتاب وسطي
void createFinalBook();         // إنشاء الكتاب النهائي
void linkAllBooks();            // ربط كل الكتب ببعضها
void addPuzzleToBook(BookNode* book, int puzzleCount);  // إضافة ألغاز
```

لماذا هذه الدوال private؟

- المستخدم لا يحتاج استدعاءها مباشرة
- تُستدعى تلقائياً في الـ Constructor
- تحمي المكتبة من التعديلات الخاطئة

---

3 الدوال العامة (Public Functions)

البناء والتدمير

```
cpp

Library(vector<Puzzle> puzzles);  // ينشئ المكتبة ويبنيها
~Library();                       // يمسح كل الكتب من الذاكرة
```

دوال الوصول للبيانات

```
cpp

BookNode** getEntranceBooks();    // يرجع مؤشر لمصفوفة المداخل
BookNode* getFinalBook();         // يرجع الكتاب النهائي
int getMiddleBooksCount();        // يرجع عدد الكتب الوسطية
```

دوال الاختبار والتحقق

```cpp
void printStructure();          // يطبع بنية المكتبة بالكامل
bool checkIfValid();            // يتحقق من صحة الروابط
```

📝 ملف Library.cpp (ملف التنفيذ)

1 البناء (Constructor)

```cpp
Library::Library(vector<Puzzle> puzzles) {
    // خطوة 1: حفظ الألغاز
    allPuzzles = puzzles;
    nextPuzzleIndex = 0;

    // خطوة 2: تهيئة المؤشرات بـ null
    finalBook = nullptr;
    for (int i = 0; i < 4; i++) {
        entranceBooks[i] = nullptr;
    }

    // خطوة 3: بناء المكتبة بالكامل
    createEntranceBooks();  // ✓ إنشاء المداخل
    createMiddleBooks();    // ✓ إنشاء الوسطية
    createFinalBook();      // ✓ إنشاء النهائي
    linkAllBooks();         // ✓ ربط الكل
}
```

الترتيب مهم جداً!

- نبدأ بالمداخل والوسطية قبل الربط
- الربط يحتاج كل الكتب موجودة أولاً

مثال استخدام:

```cpp
vector<Puzzle> myPuzzles;
// ... إضافة ألغاز للقائمة
Library lib(myPuzzles);  // تتبني المكتبة تلقائياً
```

---

2 المدمر (Destructor)

cpp

```cpp
Library::~Library() {
    // مسح كتب المداخل (مصفوفة عادية)
    for (int i = 0; i < 4; i++) {
        if (entranceBooks[i] != nullptr) {
            delete entranceBooks[i];
            entranceBooks[i] = nullptr;
        }
    }

    // مسح الكتب الوسطية (vector)
    for (int i = 0; i < middleBooks.size(); i++) {
        if (middleBooks[i] != nullptr) {
            delete middleBooks[i];
        }
    }
    middleBooks.clear();  // تفضية الـ vector

    // مسح الكتاب النهائي
    if (finalBook != nullptr) {
        delete finalBook;
        finalBook = nullptr;
    }
}
```

لماذا هذا مهم؟

- نمنع تسرب الذاكرة (Memory Leak)
- كل كتاب تم إنشاؤه بـ `new` يجب حذفه بـ `delete`
- نضع `nullptr` بعد الحذف لتجنب الـ Dangling Pointers

---

3 إنشاء كتب المداخل

cpp

```cpp
void Library::createEntranceBooks() {
    for (int i = 0; i < 4; i++) {
        // اختيار صعوبة عشوائية (50% سهل، 50% صعب)
        BookDifficulty diff = (randomNumber(0, 1) == 0) ? EASY : HARD;

        // إنشاء الكتاب بمعرف يبدأ من 100
        BookNode* book = new BookNode(100 + i, ENTRANCE, diff);

        // إضافة لغز واحد لكل كتاب مدخل
        addPuzzleToBook(book, 1);

        // حفظه في المصفوفة
        entranceBooks[i] = book;
    }
}
```

نتيجة التنفيذ:

```
entranceBooks[0] = كتاب #100 (عشوائي EASY/HARD)
entranceBooks[1] = كتاب #101 (عشوائي EASY/HARD)
entranceBooks[2] = كتاب #102 (عشوائي EASY/HARD)
entranceBooks[3] = كتاب #103 (عشوائي EASY/HARD)
```

---

4 إنشاء الكتب الوسطية

cpp

```cpp
void Library::createMiddleBooks() {
    // اختيار عدد عشوائي بين 10-15
    int count = randomNumber(10, 15);

    for (int i = 0; i < count; i++) {
        // صعوبة عشوائية
        BookDifficulty diff = (randomNumber(0, 1) == 0) ? EASY : HARD;

        // إنشاء كتاب بمعرف يبدأ من 200
        BookNode* book = new BookNode(200 + i, INTERMEDIATE, diff);

        // الكتب السهلة: لغزين | الصعبة: لغز واحد
        int puzzleCount = (diff == EASY) ? 2 : 1;
        addPuzzleToBook(book, puzzleCount);

        // إضافة للـ vector
        middleBooks.push_back(book);
    }
}
```

الفرق في عدد الألغاز:

- **كتاب سهل (EASY)**: لغزين - عشان يستاهل المسارين
- **كتاب صعب (HARD)**: لغز واحد - عشان المسار الواحد

مثال نتيجة:

```
middleBooks[0] = كتاب #200 (EASY - لغزين)
middleBooks[1] = كتاب #201 (HARD - لغز واحد)
...
middleBooks[12] = كتاب #212 (EASY - لغزين)
```

---

5 إنشاء الكتاب النهائي

```cpp
```

```cpp
void Library::createFinalBook() {
    // الكتاب النهائي دائماً HARD
    finalBook = new BookNode(999, FINAL, HARD);

    // لغز نهائي واحد صعب
    addPuzzleToBook(finalBook, 1);
}
```

لماذا دائماً HARD؟

- عشان يكون تحدي نهائي للاعب
- ما فيش مسارات بعده، فمش محتاجين EASY

---

6 إضافة لغز لكتاب

```cpp
void Library::addPuzzleToBook(BookNode* book, int puzzleCount) {
    // تجهيز مساحة للألغاز في الكتاب
    book->setupClues(puzzleCount);

    for (int i = 0; i < puzzleCount; i++) {
        // لو وصلنا آخر لغز، نرجع للأول (دائري)
        if (nextPuzzleIndex >= allPuzzles.size()) {
            nextPuzzleIndex = 0;
        }

        // جلب اللغز الحالي
        Puzzle p = allPuzzles[nextPuzzleIndex];

        // إضافته للكتاب
        book->addClue(i, p.riddle, p.answer);

        // الانتقال للغز التالي
        nextPuzzleIndex++;
    }
}
```

مثال توزيع الألغاز:

لو عندنا 20 لغز:

لغز 0 ← كتاب #100

لغز 1 ← كتاب #101

لغز 2 ← كتاب #102

لغز 3، لغز 4 ← كتاب #200 (EASY)

لغز 5 ← كتاب #201 (HARD)

وهكذا ...

---

7 ربط كل الكتب (أهم دالة!)

cpp

```cpp
void Library::linkAllBooks() {
    int middleCount = middleBooks.size();

    // ===== ربط المداخل بالكتب الوسطية: الجزء 1 =====
    for (int i = 0; i < 4; i++) {
        // المسار الأول: كتاب وسطي عشوائي من أول 5
        int randomIndex = randomNumber(0, min(4, middleCount - 1));
        entranceBooks[i]->next1 = middleBooks[randomIndex];

        // لو المدخل سهل، نعطيه مسار ثاني
        if (entranceBooks[i]->difficulty == EASY) {
            int randomIndex2 = randomNumber(0, min(4, middleCount - 1));

            // نتأكد المسار الثاني مختلف عن الأول
            if (randomIndex2 == randomIndex && middleCount > 1) {
                randomIndex2 = (randomIndex + 1) % middleCount;
            }

            entranceBooks[i]->next2 = middleBooks[randomIndex2];
        }
    }
}
```

رسم توضيحي:

```
مدخل #100 (EASY) ──next1──→ كتاب #200
                 └─next2──→ كتاب #203

مدخل #101 (HARD) ──next1──→ كتاب #201
                 └─next2──→ NULL
```

الجزء 2: ربط الكتب الوسطية

```cpp
    // ===== الجزء 2: ربط الكتب الوسطية =====
    for (int i = 0; i < middleCount; i++) {
        BookNode* current = middleBooks[i];

        // آخر 3 كتب يروحوا للكتاب النهائي مباشرة
        if (i >= middleCount - 3) {
            current->next1 = finalBook;

            if (current->difficulty == EASY) {
                current->next2 = finalBook;
            }
        }
        // باقي الكتب تروح لكتب وسطية أخرى
        else {
            // المسار الأول: كتاب لاحق عشوائي
            int nextIndex = randomNumber(i + 1, middleCount - 1);
            current->next1 = middleBooks[nextIndex];

            // لو الكتاب سهل، نعطيه مسار ثاني
            if (current->difficulty == EASY) {
                int nextIndex2 = randomNumber(i + 1, middleCount - 1);

                // نتأكد المسار مختلف
                if (nextIndex2 == nextIndex && nextIndex > i + 1) {
                    nextIndex2 = nextIndex - 1;
                }

                current->next2 = middleBooks[nextIndex2];
            }
        }
    }
}
```

استراتيجية الربط:

1. **الكتب الوسطية الأولى** (إلى 0 count-4):
    - تروح لكتب وسطية أخرى أبعد منها
    - تخلق مسارات متشعبة
2. **آخر 3 كتب وسطية**:
    - تروح مباشرة للكتاب النهائي

- تضمن وصول اللاعب للنهاية

مثال رسم:

```
كتاب #200 ──→ كتاب #205 ──→ كتاب #210 ──→ [كتاب نهائي #999]
                  └──→ كتاب #207 ──→ كتاب #211 ──→ [كتاب نهائي #999]
                                          └──→ [كتاب نهائي #999]
```

---

8 دوال الوصول (Getters)

cpp

```cpp
BookNode** Library::getEntranceBooks() {
    return entranceBooks;  // يرجع مؤشر للمصفوفة
}

BookNode* Library::getFinalBook() {
    return finalBook;  // يرجع مؤشر الكتاب النهائي
}

int Library::getMiddleBooksCount() {
    return middleBooks.size();  // عدد العناصر في الـ vector
}
```

الاستخدام:

cpp

```cpp
Library lib(puzzles);

// الحصول على المداخل
BookNode** entries = lib.getEntranceBooks();
cout << "أول مدخل: " << entries[0]->id << endl;

// الحصول على النهائي
BookNode* final = lib.getFinalBook();
cout << "الكتاب النهائي: " << final->id << endl;

// عدد الكتب
cout << "عدد الكتب الوسطية: " << lib.getMiddleBooksCount() << endl;
```

---

9 طباعة بنية المكتبة

cpp

```cpp
void Library::printStructure() {
    // ===== المداخل =====
    cout << "ENTRANCE BOOKS (4):" << endl;
    for (int i = 0; i < 4; i++) {
        cout << "  Book #" << entranceBooks[i]->id;
        cout << " (" << entranceBooks[i]->getDifficultyString() << ")";
        cout << " -> Book #" << entranceBooks[i]->next1->id;

        if (entranceBooks[i]->next2 != nullptr) {
            cout << " & Book #" << entranceBooks[i]->next2->id;
        }
        cout << endl;
    }

    // ===== الوسطية =====
    cout << "\nINTERMEDIATE BOOKS (" << middleBooks.size() << "):" << endl;
    for (int i = 0; i < middleBooks.size(); i++) {
        BookNode* book = middleBooks[i];
        cout << "  Book #" << book->id;
        cout << " (" << book->getDifficultyString() << ")";

        if (book->next1 != nullptr) {
            cout << " -> Book #" << book->next1->id;
        }
        if (book->next2 != nullptr) {
            cout << " & Book #" << book->next2->id;
        }
        cout << endl;
    }

    // ===== النهائي =====
    cout << "\nFINAL BOOK:" << endl;
    cout << "  Book #" << finalBook->id << endl;
}
```

مثال خرج:

```
==========================================
      LIBRARY STRUCTURE
==========================================

ENTRANCE BOOKS (4):
  Book #100 (EASY) -> Book #200 & Book #201
  Book #101 (HARD) -> Book #202
  Book #102 (EASY) -> Book #200 & Book #203
  Book #103 (HARD) -> Book #201

INTERMEDIATE BOOKS (12):
  Book #200 (EASY) -> Book #205 & Book #207
  Book #201 (HARD) -> Book #206
  ...
  Book #209 (EASY) -> Book #999 & Book #999
  Book #210 (HARD) -> Book #999
  Book #211 (EASY) -> Book #999 & Book #999

FINAL BOOK:
  Book #999

==========================================
```

---

10 التحقق من صحة المكتبة

```cpp

```

```cpp
bool Library::checkIfValid() {
    // التحقق من كتب المداخل
    for (int i = 0; i < 4; i++) {
        if (entranceBooks[i] == nullptr) return false;
        if (entranceBooks[i]->next1 == nullptr) return false;
    }

    // التحقق من الكتاب النهائي
    if (finalBook == nullptr) return false;

    // التحقق من الكتب الوسطية
    for (int i = 0; i < middleBooks.size(); i++) {
        BookNode* book = middleBooks[i];

        if (book == nullptr) return false;
        if (book->next1 == nullptr) return false;

        // الكتب السهلة يجب أن يكون لها next2
        if (book->difficulty == EASY && book->next2 == nullptr) {
            return false;
        }
    }

    return true;  // كل شيء صحيح!
}
```

الاستخدام:

```cpp
Library lib(puzzles);

if (lib.checkIfValid()) {
    cout << "✓ المكتبة سليمة وجاهزة!" << endl;
} else {
    cout << "✗ خطأ في بنية المكتبة!" << endl;
}
```

🎮 مثال استخدام كامل

```cpp
```

```cpp
#include "Library.h"
#include <iostream>
using namespace std;

int main() {
    // ===== خطوة 1: تجهيز الألغاز =====
    vector<Puzzle> puzzles;

    Puzzle p1;
    p1.riddle = "ما الذي له عين ولا يرى؟";
    p1.answer = "الإبرة";
    puzzles.push_back(p1);

    Puzzle p2;
    p2.riddle = "شيء يمشي بلا أرجل؟";
    p2.answer = "الوقت";
    puzzles.push_back(p2);

    // ... إضافة المزيد من الألغاز

    // ===== خطوة 2: إنشاء المكتبة =====
    Library* myLibrary = new Library(puzzles);

    // ===== خطوة 3: التحقق من الصحة =====
    if (myLibrary->checkIfValid()) {
        cout << "المكتبة جاهزة!" << endl;
    }

    // ===== خطوة 4: عرض البنية =====
    myLibrary->printStructure();

    // ===== خطوة 5: بدء اللعبة =====
    BookNode** entrances = myLibrary->getEntranceBooks();

    cout << "\n(3-0) اختر نقطة البداية: ";
    int choice;
    cin >> choice;

    BookNode* currentBook = entrances[choice];
    currentBook->displayInfo();

    // ... منطق اللعبة

    // ===== خطوة 6: التنظيف =====
    delete myLibrary;
```

```
    return 0;
}
```

🔑 نقاط مهمة جداً

✅ إدارة الذاكرة الصحيحة

- كل `new` له `delete` مقابل
- المدمر ينظف كل شيء تلقائياً
- بعد الحذف `nullptr` استخدام

✅ العشوائية الذكية

- الصعوبات عشوائية (50/50)
- عدد الكتب الوسطية عشوائي (10-15)
- الروابط عشوائية لكن منطقية

✅ الروابط المنطقية

- المداخل تروح لأول 5 كتب وسطية
- الكتب الوسطية تتقدم للأمام فقط
- آخر 3 كتب تروح للنهائي مباشرة

✅ التنوع في التجربة

- كل لعبة مختلفة (عشوائية)
- مسارات متعددة للوصول للنهاية
- توزيع عادل للألغاز

🎯 استخدامات المكتبة

1. **لعبة مغامرات**:
    - اللاعب يختار مدخل
    - يحل الألغاز
    - يختار المسارات
2. **نظام تعليمي**:

- كل كتاب درس
- الألغاز تمارين
- المسارات مستويات صعوبة

3. **قصة تفاعلية**:
    - كل كتاب فصل
    - الألغاز اختيارات
    - المسارات نهايات مختلفة

📊 إحصائيات المكتبة

عدد الكتب الكلي: 15-20 كتاب
- مداخل: 4 كتب (ثابت)
- وسطية: 10-15 كتاب (متغير)
- نهائي: 1 كتاب (ثابت)

المسارات المتاحة:
- كتاب EASY: مساران (2 خيارات)
- كتاب HARD: مسار واحد (خيار واحد)

الألغاز:
- مدخل: لغز واحد
- وسطي EASY: لغزان
- وسطي HARD: لغز واحد
- نهائي: لغز واحد

🐛 نصائح التصحيح

لو المكتبة مش شغالة:

1. يعطيك فكرة عن المشكلة - `checkIfValid()` تحقق من
2. لرؤية الروابط - `printStructure()` استخدم
3. تأكد من وجود ألغاز كافية في القائمة
4. `Utils.h` موجودة في `randomNumber()` تحقق من دالة