Library Management System (Simplified)

Using Singleton + Factory Only

This is a clean, simple, interview-friendly implementation.

✅Only two design patterns: - **Singleton Pattern** → Only one Library instance - **Factory Pattern** → Creates Books and Members

---

✅Project Folder Structure

```
LibrarySystem/
├── Book.h
├── Book.cpp
├── Member.h
├── Member.cpp
├── Library.h
├── Library.cpp
├── Factory.h
├── main.cpp
├── tasks.json (VS Code)
├── launch.json (VS Code)
├── README.md
```

✅Book.h

```
#pragma once
#include <string>

class Book {
private:
    std::string title;
    std::string author;
    bool isBorrowed;

public:
    Book(const std::string& t, const std::string& a);
```

```
    void borrowBook();
    void returnBook();
    bool isBorrowedStatus() const;
    std::string getTitle() const;
    std::string getAuthor() const;
};
```

✅Book.cpp

```
#include "Book.h"

Book::Book(const std::string& t, const std::string& a)
    : title(t), author(a), isBorrowed(false) {}

void Book::borrowBook() { isBorrowed = true; }
void Book::returnBook() { isBorrowed = false; }

bool Book::isBorrowedStatus() const { return isBorrowed; }
std::string Book::getTitle() const { return title; }
std::string Book::getAuthor() const { return author; }
```

✅Member.h

```
#pragma once
#include <string>
#include <vector>
#include "Book.h"

class Member {
private:
    std::string name;
    int id;
    std::vector<Book*> borrowed;

public:
    Member(const std::string& n, int id);

    void borrowBook(Book* b);
    void returnBook(Book* b);
    void showBorrowedBooks() const;
```

```
    int getId() const;
    std::string getName() const;
};
```

✅Member.cpp

```
#include "Member.h"
#include <iostream>

Member::Member(const std::string& n, int id) : name(n), id(id) {}

void Member::borrowBook(Book* b) {
    if (b->isBorrowedStatus()) {
        std::cout << "Book already borrowed!
";
        return;
    }
    borrowed.push_back(b);
    b->borrowBook();
}

void Member::returnBook(Book* b) {
    for (auto it = borrowed.begin(); it != borrowed.end(); ++it) {
        if (*it == b) {
            borrowed.erase(it);
            b->returnBook();
            return;
        }
    }
    std::cout << "This book was not borrowed by this member.
";
}

void Member::showBorrowedBooks() const {
    if (borrowed.empty()) {
        std::cout << "No borrowed books.
";
        return;
    }
    std::cout << "Borrowed Books:
";
    for (auto b : borrowed) {
        std::cout << "- " << b->getTitle() << " by " << b->getAuthor() << "
```

```
";
    }
}

int Member::getId() const { return id; }
std::string Member::getName() const { return name; }
```

✅Factory.h (Factory Pattern)

```
#pragma once
#include "Book.h"
#include "Member.h"

class Factory {
public:
    static Book* createBook(const std::string& title, const std::string&
author) {
        return new Book(title, author);
    }

    static Member* createMember(const std::string& name, int id) {
        return new Member(name, id);
    }
};
```

✅Library.h (Singleton)

```
#pragma once
#include <vector>
#include "Book.h"
#include "Member.h"

class Library {
private:
    std::vector<Book*> books;
    std::vector<Member*> members;

    Library() {}  // private constructor

public:
```

```
    static Library& getInstance();

    void addBook(Book* b);
    void addMember(Member* m);

    Book* findBook(const std::string& title);
    Member* findMember(int id);

    void borrowBook(int memberId, const std::string& title);
    void returnBook(int memberId, const std::string& title);

    void showBooks() const;
};
```

✅Library.cpp

```
#include "Library.h"
#include <iostream>

Library& Library::getInstance() {
    static Library instance;
    return instance;
}

void Library::addBook(Book* b) { books.push_back(b); }
void Library::addMember(Member* m) { members.push_back(m); }

Book* Library::findBook(const std::string& title) {
    for (auto b : books) if (b->getTitle() == title) return b;
    return nullptr;
}

Member* Library::findMember(int id) {
    for (auto m : members) if (m->getId() == id) return m;
    return nullptr;
}

void Library::borrowBook(int memberId, const std::string& title) {
    Member* m = findMember(memberId);
    Book* b = findBook(title);

    if (!m || !b) {
        std::cout << "Member or Book not found!
";
```

```
        return;
    }

    m->borrowBook(b);
}

void Library::returnBook(int memberId, const std::string& title) {
    Member* m = findMember(memberId);
    Book* b = findBook(title);

    if (!m || !b) {
        std::cout << "Member or Book not found!
";
        return;
    }

    m->returnBook(b);
}

void Library::showBooks() const {
    if (books.empty()) {
        std::cout << "No books in library.
";
        return;
    }

    for (auto b : books) {
        std::cout << "- " << b->getTitle() << " by " << b->getAuthor();
        if (b->isBorrowedStatus()) std::cout << " (Borrowed)";
        std::cout << "
";
    }
}
```

✅main.cpp

```
#include <iostream>
#include "Library.h"
#include "Factory.h"

int main() {
    Library& lib = Library::getInstance();

    lib.addBook(Factory::createBook("C++ Basics", "Bjarne"));
```

```
    lib.addBook(Factory::createBook("UML Made Easy", "Fowler"));

    lib.addMember(Factory::createMember("John", 101));
    lib.addMember(Factory::createMember("Alice", 102));

    int choice;
    do {
        std::cout << "
1. Show Books
2. Borrow Book
3. Return Book
4. Exit
Choice: ";
        std::cin >> choice;

        if (choice == 1) lib.showBooks();
        else if (choice == 2) {
            int id; std::string title;
            std::cout << "Member ID: "; std::cin >> id; std::cin.ignore();
            std::cout << "Book Title: "; std::getline(std::cin, title);
            lib.borrowBook(id, title);
        }
        else if (choice == 3) {
            int id; std::string title;
            std::cout << "Member ID: "; std::cin >> id; std::cin.ignore();
            std::cout << "Book Title: "; std::getline(std::cin, title);
            lib.returnBook(id, title);
        }

    } while (choice != 4);

    return 0;
}
```

✅README.md (Short)

```
## Library Management System (C++)
### Design Patterns Used
- Singleton → Only one Library instance
- Factory → Creates Book and Member objects

### Features
- Add Books
- Add Members
```

```
- Borrow / Return Books
- Show All Books

### Build (VS Code)
Use: C/C++ Extension + MinGW
```bash
g++ *.cpp -o app.exe
./app.exe
```

```

---

# ✅Done!

This is a **clean, simple, interview-ready** version using ONLY: ✅Singleton ✅Factory

If you want next: ✅UML diagram (Class + Sequence)
✅Add file handling
✅Add exceptions
✅Add smart pointers (unique_ptr)