**JAYPEE INSTITUTE OF INFORMATION TECHNOLOGY**

****

**SMART LIBRARY MANAGEMENT SYSTEM**

**BATCH: F-19**

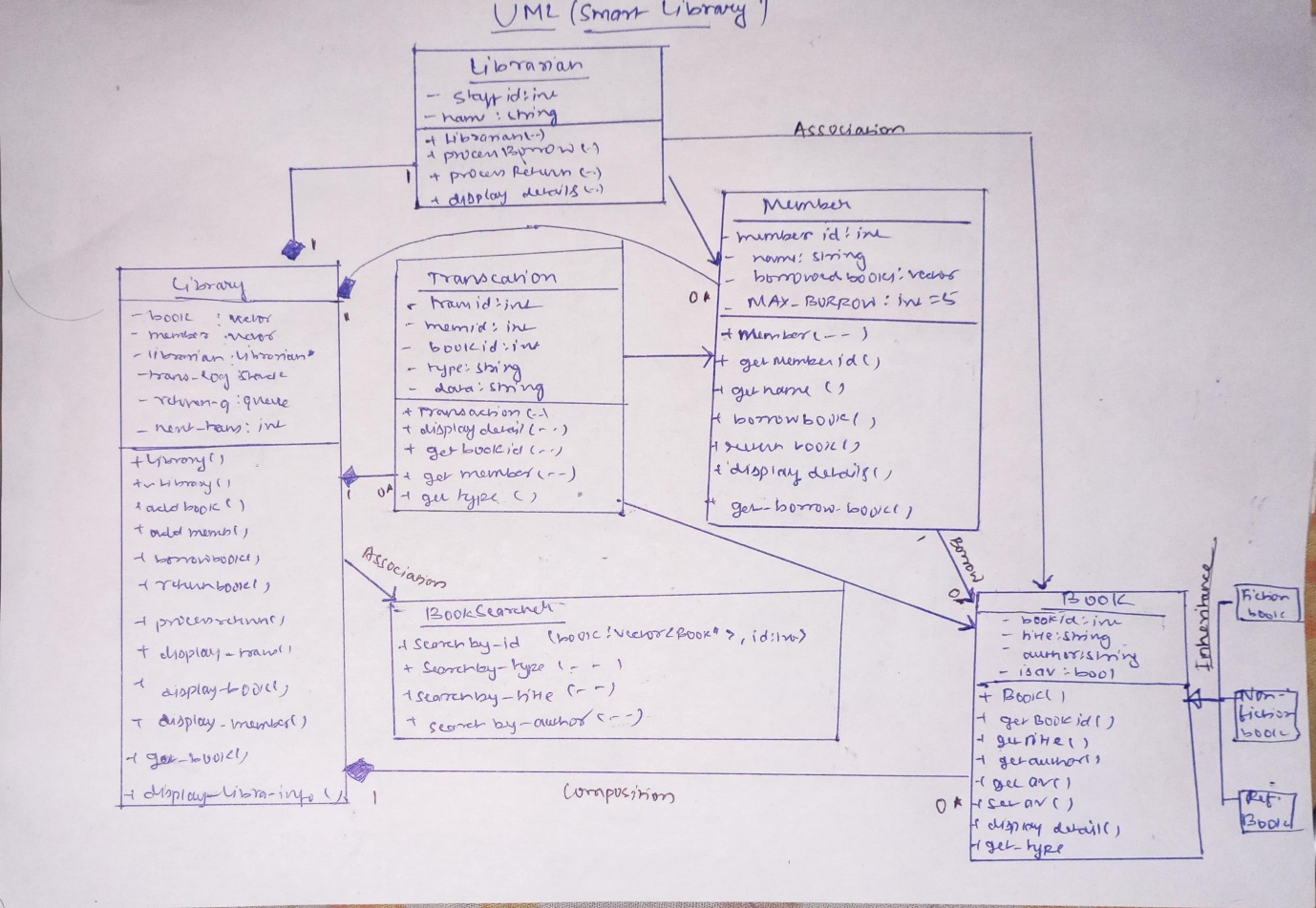
**Submitted by:**

**Shashwat Anand (992401040113)**

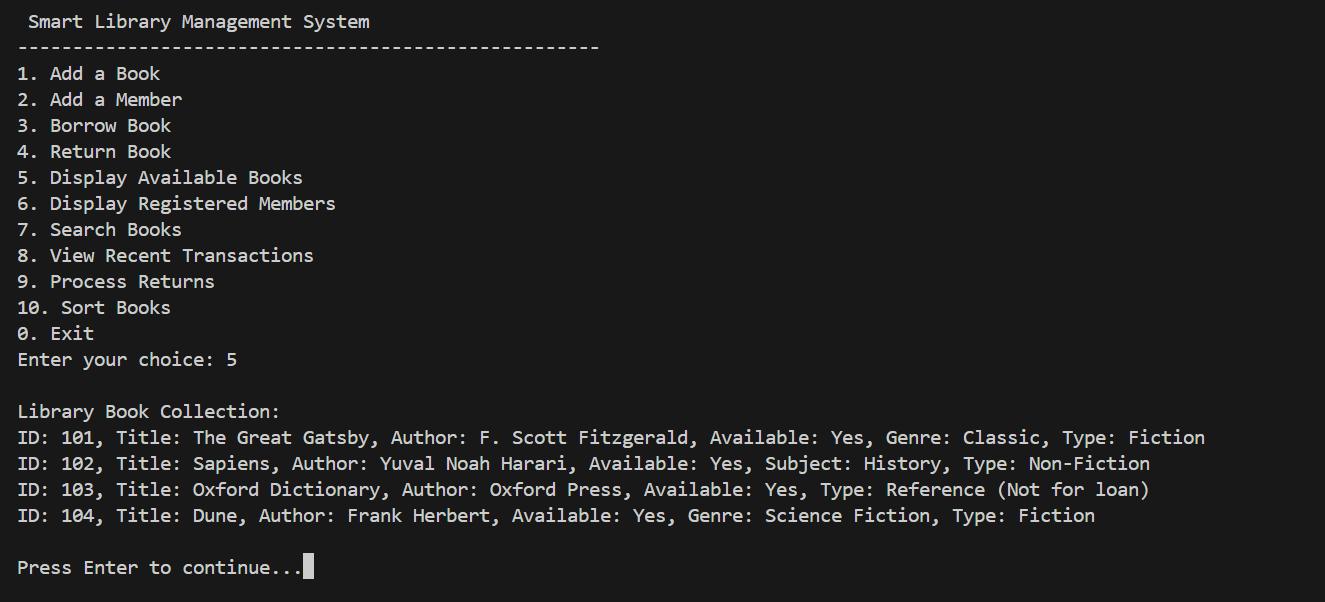
**Submitted to:**

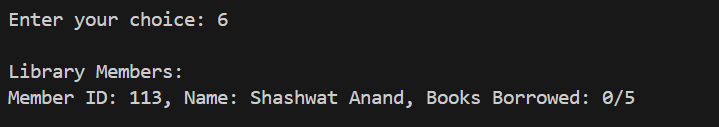
**Dr. Nivedita Batra**

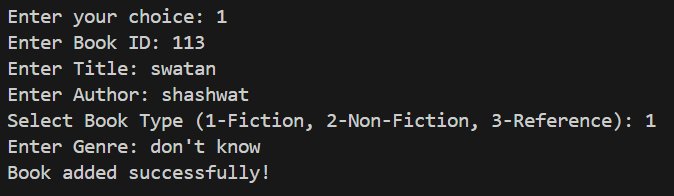
**UML DIG:**

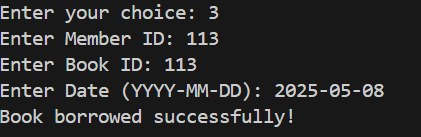


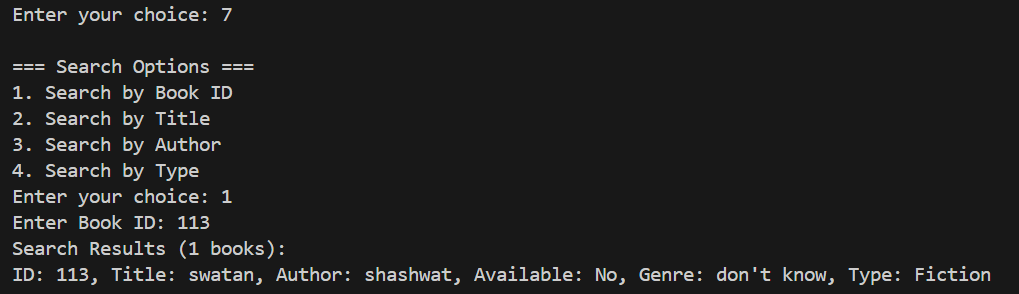
**OUTPUT**

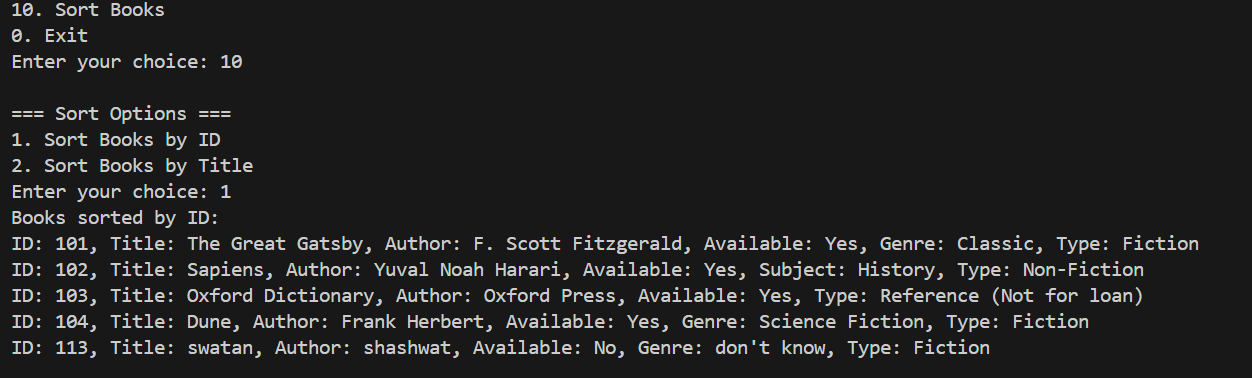


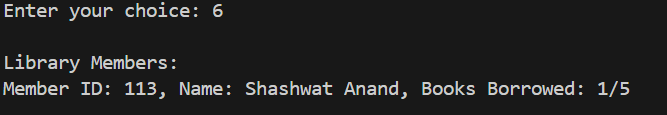


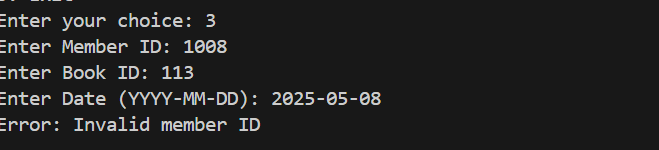


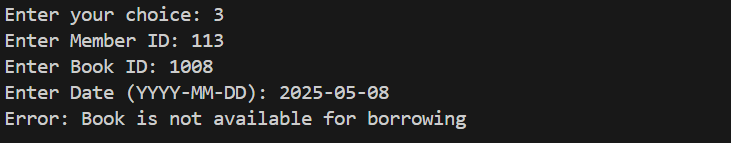


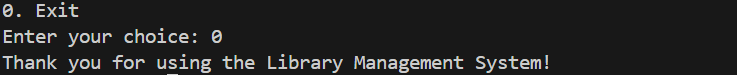












**READ ME:**

*LIBRARY MANAGEMENT SYSTEM*

*README DOCUMENTATION*

*1. SYSTEM OVERVIEW*

*The Library Management System is a C++ console application designed to manage book inventory, member registrations, and borrowing/returning transactions in a library. It implements object-oriented programming principles with classes for books, members, transactions, and library operations.*

*2. FEATURES*

*- Book Management:*

*• Add new books (Fiction, Non-Fiction, Reference)*

*• Display all available books*

*• Search books by ID, title, author, or type*

*• Sort books by ID or title*

*- Member Management:*

*• Register new members*

*• Track borrowed books*

*• Display member information*

*• Enforce borrowing limits (max 5 books per member)*

*- Transaction Processing:*

*• Borrow books with date tracking*

*• Return books with automatic availability update*

*• View recent transactions*

*• Process returns queue*

*3. CLASS STRUCTURE*

*3.1 BOOK CLASSES*

*• Book (Base Class)*

*- Attributes: bookID, title, author, isAvailable*

*- Methods: getters/setters, displayDetails(), getType()*

*• FictionBook (Derived)*

*- Additional: genre*

*• NonFictionBook (Derived)*

*- Additional: subject*

*• ReferenceBook (Derived)*

*- Special: Cannot be borrowed*

*3.2 MEMBER CLASS*

*- Attributes: memberID, name, borrowedBooks list*

*- Methods: borrowBook(), returnBook(), displayDetails()*

*3.3 TRANSACTION CLASS*

*- Attributes: transactionID, memberID, bookID, type, date*

*- Methods: displayDetails()*

*3.4 LIBRARIAN CLASS*

*- Handles: processBorrow(), processReturn()*

*3.5 BOOKSEARCHER CLASS*

*- Static methods for searching: by ID, title, author, type*

*3.6 LIBRARY CLASS (Main Controller)*

*- Manages collections of books and members*

*- Processes all transactions*

*- Maintains transaction log and return queue*

*4. HOW TO USE*

*4.1 COMPILATION*

*1. Save all code in a single .cpp file (e.g., LibrarySystem.cpp)*

*2. Compile using: g++ LibrarySystem.cpp -o LibrarySystem*

*3. Run the executable: ./LibrarySystem*

*4.2 MENU OPTIONS*

*1. Add Book - Register new books with details*

*2. Add Member - Create new library members*

*3. Borrow Book - Check out books to members*

*4. Return Book - Check in returned books*

*5. Display Books - Show all available books*

*6. Display Members - List all registered members*

*7. Search Books - Find books by various criteria*

*8. View Transactions - See recent borrowing history*

*9. Process Returns - Update availability of returned books*

*10. Sort Books - Organize books by ID or title*

*0. Exit - Close the application*

*5. SAMPLE DATA*

*The system comes pre-loaded with sample data:*

*- Books:*

*• Fiction: The Great Gatsby, Dune*

*• Non-Fiction: Sapiens*

*• Reference: Oxford Dictionary*

*- Members: Shashwat Anand (ID: 113)*

*6. ERROR HANDLING*

*The system includes validation for:*

*- Duplicate book borrowing*

*- Exceeding borrowing limits*

*- Invalid member/book IDs*

*- Attempting to borrow unavailable books*

*7. TECHNICAL NOTES*

*- Uses STL containers: vector, stack, queue*

*- Implements polymorphism for book types*

*- Follows OOP principles with encapsulation*

*- Includes proper memory management*

*- Used linked list*

*8. LIMITATIONS*

*- Console-based interface (no GUI)*

*- Data not persistent between runs*

*- Basic error handling with C-strings*

*9. FUTURE ENHANCEMENTS*

*- Database integration for persistent storage*

*- Graphical user interface*

*- Advanced reporting features*

*- Fine management for late returns*

*- User authentication system*

*10. AUTHOR*

*Shashwat Anand*

**CODE:**

#include <iostream>

#include <string>

#include <stack>

#include <queue>

#include <vector>

#include <algorithm>

#include <cstring>

using namespace std;

// Base Book Class

class Book {

protected:

    int bookID;

    string title;

    string author;

    bool isAvailable;

public:

    Book(int id, string t, string a) {

        this->bookID = id;

        this->title = t;

        this->author = a;

        this->isAvailable = true;

    }

    virtual ~Book() {}

    int getBookID() const { return this->bookID; }

    string getTitle() const { return this->title; }

    string getAuthor() const { return this->author; }

    bool getAvailability() const { return this->isAvailable; }

    void setAvailability(bool status) { this->isAvailable = status; }

    virtual void displayDetails() const {

        cout << "ID: " << this->bookID << ", Title: " << this->title

             << ", Author: " << this->author << ", Available: "

             << (this->isAvailable ? "Yes" : "No");

    }

    virtual string getType() const { return "Generic Book"; }

};

// Derived Book Classes

class FictionBook : public Book {

    string genre;

public:

    FictionBook(int id, string t, string a, string g) : Book(id, t, a) {

        this->genre = g;

    }

    void displayDetails() const override {

        Book::displayDetails();

        cout << ", Genre: " << this->genre << ", Type: Fiction" << endl;

    }

    string getType() const override { return "Fiction"; }

};

class NonFictionBook : public Book {

    string subject;

public:

    NonFictionBook(int id, string t, string a, string s) : Book(id, t, a) {

        this->subject = s;

    }

    void displayDetails() const override {

        Book::displayDetails();

        cout << ", Subject: " << this->subject << ", Type: Non-Fiction" << endl;

    }

    string getType() const override { return "Non-Fiction"; }

};

class ReferenceBook : public Book {

public:

    ReferenceBook(int id, string t, string a) : Book(id, t, a) {}

    void displayDetails() const override {

        Book::displayDetails();

        cout << ", Type: Reference (Not for loan)" << endl;

    }

    string getType() const override { return "Reference"; }

};

// Member Class

class Member {

private:

    int memberID;

    string name;

    vector<int> borrowedBooks;

    static const int MAX\_BORROW\_LIMIT = 5;

public:

    Member(int id, string n) {

        this->memberID = id;

        this->name = n;

    }

    int getMemberID() const { return this->memberID; }

    string getName() const { return this->name; }

    void borrowBook(int bookID) {

        if (this->borrowedBooks.size() >= MAX\_BORROW\_LIMIT) {

            throw "Maximum borrowing limit reached";

        }

        for (int id : this->borrowedBooks) {

            if (id == bookID) {

                throw "Member has already borrowed this book";

            }

        }

        this->borrowedBooks.push\_back(bookID);

    }

    void returnBook(int bookID) {

        bool found = false;

        for (auto it = this->borrowedBooks.begin(); it != this->borrowedBooks.end(); ++it) {

            if (\*it == bookID) {

                this->borrowedBooks.erase(it);

                found = true;

                break;

            }

        }

        if (!found) {

            throw "Member hasn't borrowed this book";

        }

    }

    void displayDetails() const {

        cout << "Member ID: " << this->memberID << ", Name: " << this->name

             << ", Books Borrowed: " << this->borrowedBooks.size() << "/"

             << MAX\_BORROW\_LIMIT << endl;

    }

    const vector<int>& getBorrowedBooks() const { return this->borrowedBooks; }

};

// Transaction Class

class Transaction {

private:

    int transactionID;

    int memberID;

    int bookID;

    string type;

    string date;

public:

    Transaction(int tid, int mid, int bid, string t, string d) {

        this->transactionID = tid;

        this->memberID = mid;

        this->bookID = bid;

        this->type = t;

        this->date = d;

    }

    void displayDetails() const {

        cout << "Transaction #" << this->transactionID << ": Member " << this->memberID

             << " " << this->type << " Book " << this->bookID << " on " << this->date << endl;

    }

    int getBookID() const { return this->bookID; }

    int getMemberID() const { return this->memberID; }

    string getType() const { return this->type; }

};

// Librarian Class

class Librarian {

private:

    int staffID;

    string name;

public:

    Librarian(int id, string n) {

        this->staffID = id;

        this->name = n;

    }

    void processBorrow(Member& member, Book& book) {

        if (!book.getAvailability()) {

            throw "Book is not available for borrowing";

        }

        member.borrowBook(book.getBookID());

        book.setAvailability(false);

    }

    void processReturn(Member& member, Book& book) {

        member.returnBook(book.getBookID());

        book.setAvailability(true);

    }

    void displayDetails() const {

        cout << "Librarian ID: " << this->staffID << ", Name: " << this->name << endl;

    }

};

// Search Book Class

class BookSearcher {

public:

    static vector<Book\*> searchByID(const vector<Book\*>& books, int id) {

        vector<Book\*> results;

        for (Book\* book : books) {

            if (book->getBookID() == id) {

                results.push\_back(book);

            }

        }

        return results;

    }

    static vector<Book\*> searchByType(const vector<Book\*>& books, const string& type) {

        vector<Book\*> results;

        for (Book\* book : books) {

            if (book->getType() == type) {

                results.push\_back(book);

            }

        }

        return results;

    }

    static vector<Book\*> searchByTitle(const vector<Book\*>& books, const string& title) {

        vector<Book\*> results;

        for (Book\* book : books) {

            if (book->getTitle().find(title) != string::npos) {

                results.push\_back(book);

            }

        }

        return results;

    }

    static vector<Book\*> searchByAuthor(const vector<Book\*>& books, const string& author) {

        vector<Book\*> results;

        for (Book\* book : books) {

            if (book->getAuthor().find(author) != string::npos) {

                results.push\_back(book);

            }

        }

        return results;

    }

};

// Library Class

class Library {

private:

    vector<Book\*> books;

    vector<Member\*> members;

    Librarian\* librarian;

    stack<Transaction> transactionLog;

    queue<int> returnQueue;

    int nextTransactionID;

    Book\* findBook(int bookID) {

        for (Book\* book : this->books) {

            if (book->getBookID() == bookID) {

                return book;

            }

        }

        return nullptr;

    }

    Member\* findMember(int memberID) {

        for (Member\* member : this->members) {

            if (member->getMemberID() == memberID) {

                return member;

            }

        }

        return nullptr;

    }

public:

    Library() {

        this->nextTransactionID = 1;

        this->librarian = new Librarian(1, "Admin Librarian");

    }

    ~Library() {

        for (Book\* book : this->books) delete book;

        for (Member\* member : this->members) delete member;

        delete this->librarian;

    }

    void addBook(Book\* book) {

        this->books.push\_back(book);

    }

    void addMember(Member\* member) {

        this->members.push\_back(member);

    }

    void borrowBook(int memberID, int bookID, const string& date) {

        Member\* member = this->findMember(memberID);

        Book\* book = this->findBook(bookID);

        if (!member) throw "Invalid member ID";

        if (!book) throw "Book not found in records";

        this->librarian->processBorrow(\*member, \*book);

        Transaction t(this->nextTransactionID++, memberID, bookID, "borrowed", date);

        this->transactionLog.push(t);

    }

    void returnBook(int memberID, int bookID, const string& date) {

        Member\* member = this->findMember(memberID);

        Book\* book = this->findBook(bookID);

        if (!member) throw "Invalid member ID";

        if (!book) throw "Book not found in records";

        this->librarian->processReturn(\*member, \*book);

        Transaction t(this->nextTransactionID++, memberID, bookID, "returned", date);

        this->transactionLog.push(t);

        this->returnQueue.push(bookID);

    }

    void processReturns() {

        cout << "Processing returns queue:" << endl;

        while (!this->returnQueue.empty()) {

            int bookID = this->returnQueue.front();

            this->returnQueue.pop();

            Book\* book = this->findBook(bookID);

            if (book) {

                cout << "Book ID " << bookID << " (" << book->getTitle()

                     << ") is now available" << endl;

            }

        }

    }

    void displayRecentTransactions(int count = 5) {

        stack<Transaction> temp = this->transactionLog;

        cout << "Recent " << count << " transactions:" << endl;

        for (int i = 0; i < count && !temp.empty(); i++) {

            temp.top().displayDetails();

            temp.pop();

        }

    }

    void displayAllBooks() const {

        cout << "\nLibrary Book Collection:" << endl;

        for (const Book\* book : this->books) {

            book->displayDetails();

        }

    }

    void displayAllMembers() const {

        cout << "\nLibrary Members:" << endl;

        for (const Member\* member : this->members) {

            member->displayDetails();

        }

    }

    vector<Book\*> getBooks() const { return this->books; }

    void displayLibrarianInfo() const {

        cout << "\nLibrary Administrator:" << endl;

        this->librarian->displayDetails();

    }

};

// Comparison functions for sorting

bool compareBooksByID(Book\* a, Book\* b) {

    return a->getBookID() < b->getBookID();

}

bool compareBooksByTitle(Book\* a, Book\* b) {

    return a->getTitle() < b->getTitle();

}

void displayMenu() {

    cout << "\n Smart Library Management System" << endl;

    cout<<"-----------------------------------------------------"<<endl;

    cout << "1. Add a Book" << endl;

    cout << "2. Add a Member" << endl;

    cout << "3. Borrow Book" << endl;

    cout << "4. Return Book" << endl;

    cout << "5. Display Available Books" << endl;

    cout << "6. Display Registered Members" << endl;

    cout << "7. Search Books" << endl;

    cout << "8. View Recent Transactions" << endl;

    cout << "9. Process Returns" << endl;

    cout << "10. Sort Books" << endl;

    cout << "0. Exit" << endl;

    cout << "Enter your choice: ";

}

void searchMenu(Library& library) {

    cout << "\n=== Search Options ===" << endl;

    cout << "1. Search by Book ID" << endl;

    cout << "2. Search by Title" << endl;

    cout << "3. Search by Author" << endl;

    cout << "4. Search by Type" << endl;

    cout << "Enter your choice: ";

    int choice;

    cin >> choice;

    cin.ignore();

    vector<Book\*> results;

    if (choice == 1) {

        int id;

        cout << "Enter Book ID: ";

        cin >> id;

        results = BookSearcher::searchByID(library.getBooks(), id);

    }

    else if (choice == 2) {

        string title;

        cout << "Enter Title (or part of it): ";

        getline(cin, title);

        results = BookSearcher::searchByTitle(library.getBooks(), title);

    }

    else if (choice == 3) {

        string author;

        cout << "Enter Author (or part of name): ";

        getline(cin, author);

        results = BookSearcher::searchByAuthor(library.getBooks(), author);

    }

    else if (choice == 4) {

        string type;

        cout << "Enter Type (Fiction/Non-Fiction/Reference): ";

        getline(cin, type);

        results = BookSearcher::searchByType(library.getBooks(), type);

    }

    else {

        cout << "Invalid choice!" << endl;

        return;

    }

    if (results.empty()) {

        cout << "No books found matching your criteria." << endl;

    } else {

        cout << "Search Results (" << results.size() << " books):" << endl;

        for (Book\* book : results) {

            book->displayDetails();

        }

    }

}

void sortMenu(Library& library) {

    cout << "\n=== Sort Options ===" << endl;

    cout << "1. Sort Books by ID" << endl;

    cout << "2. Sort Books by Title" << endl;

    cout << "Enter your choice: ";

    int choice;

    cin >> choice;

    vector<Book\*> books = library.getBooks();

    if (choice == 1) {

        sort(books.begin(), books.end(), compareBooksByID);

        cout << "Books sorted by ID:" << endl;

    }

    else if (choice == 2) {

        sort(books.begin(), books.end(), compareBooksByTitle);

        cout << "Books sorted by Title:" << endl;

    }

    else {

        cout << "Invalid choice!" << endl;

        return;

    }

    for (Book\* book : books) {

        book->displayDetails();

    }

}

int main() {

    Library library;

    // Add some sample data

    library.addBook(new FictionBook(101, "The Great Gatsby", "F. Scott Fitzgerald", "Classic"));

    library.addBook(new NonFictionBook(102, "Sapiens", "Yuval Noah Harari", "History"));

    library.addBook(new ReferenceBook(103, "Oxford Dictionary", "Oxford Press"));

    library.addBook(new FictionBook(104, "Dune", "Frank Herbert", "Science Fiction"));

    library.addMember(new Member(113, "Shashwat Anand"));

    int choice;

    do {

        displayMenu();

        cin >> choice;

        cin.ignore();

        try {

            if (choice == 1) {

                int id, typeChoice;

                string title, author, extra;

                cout << "Enter Book ID: ";

                cin >> id;

                cin.ignore();

                cout << "Enter Title: ";

                getline(cin, title);

                cout << "Enter Author: ";

                getline(cin, author);

                cout << "Select Book Type (1-Fiction, 2-Non-Fiction, 3-Reference): ";

                cin >> typeChoice;

                cin.ignore();

                Book\* newBook = nullptr;

                if (typeChoice == 1) {

                    cout << "Enter Genre: ";

                    getline(cin, extra);

                    newBook = new FictionBook(id, title, author, extra);

                }

                else if (typeChoice == 2) {

                    cout << "Enter Subject: ";

                    getline(cin, extra);

                    newBook = new NonFictionBook(id, title, author, extra);

                }

                else if (typeChoice == 3) {

                    newBook = new ReferenceBook(id, title, author);

                }

                else {

                    cout << "Invalid type selection!" << endl;

                    continue;

                }

                library.addBook(newBook);

                cout << "Book added successfully!" << endl;

            }

            else if (choice == 2) {

                int id;

                string name;

                cout << "Enter Member ID: ";

                cin >> id;

                cin.ignore();

                cout << "Enter Member Name: ";

                getline(cin, name);

                library.addMember(new Member(id, name));

                cout << "Member registered successfully!" << endl;

            }

            else if (choice == 3) {

                int memberID, bookID;

                string date;

                cout << "Enter Member ID: ";

                cin >> memberID;

                cout << "Enter Book ID: ";

                cin >> bookID;

                cin.ignore();

                cout << "Enter Date (YYYY-MM-DD): ";

                getline(cin, date);

                library.borrowBook(memberID, bookID, date);

                cout << "Book borrowed successfully!" << endl;

            }

            else if (choice == 4) {

                int memberID, bookID;

                string date;

                cout << "Enter Member ID: ";

                cin >> memberID;

                cout << "Enter Book ID: ";

                cin >> bookID;

                cin.ignore();

                cout << "Enter Date (YYYY-MM-DD): ";

                getline(cin, date);

                library.returnBook(memberID, bookID, date);

                cout << "Book returned successfully!" << endl;

            }

            else if (choice == 5) {

                library.displayAllBooks();

            }

            else if (choice == 6) {

                library.displayAllMembers();

            }

            else if (choice == 7) {

                searchMenu(library);

            }

            else if (choice == 8) {

                library.displayRecentTransactions();

            }

            else if (choice == 9) {

                library.processReturns();

            }

            else if (choice == 10) {

                sortMenu(library);

            }

            else if (choice == 0) {

                cout << "Thank you for using the Library Management System!" << endl;

            }

            else {

                cout << "Invalid choice. Please try again." << endl;

            }

        } catch (const char\* msg) {

            cerr << "Error: " << msg << endl;

        }

        if (choice != 0) {

            cout << "\nPress Enter to continue...";

            cin.ignore();

        }

    } while (choice != 0);

    return 0;

}