**Assignment NO 4:**

**Reg no: BCS233127**

#include <iostream>

#include <string>

#include <vector>

#include <map>

class Item {

protected:

std::string itemID;

std::string title;

bool availability;

public:

Item(std::string id, std::string title, bool available)

: itemID(id), title(title), availability(available) {}

virtual ~Item() {}

virtual void getItemDetails() const = 0;

virtual bool checkAvailability() const = 0;

virtual void checkOut() = 0;

virtual void checkIn() = 0;

};

**2. Derived Classes: Book and Journal**

class Book : public Item {

private:

std::string author;

std::string ISBN;

public:

Book(std::string id, std::string title, bool available, std::string author, std::string ISBN)

: Item(id, title, available), author(author), ISBN(ISBN) {}

void getItemDetails() const override {

std::cout << "Book ID: " << itemID << "\nTitle: " << title << "\nAuthor: " << author << "\nISBN: " << ISBN << "\nAvailable: " << (availability ? "Yes" : "No") << std::endl;

}

bool checkAvailability() const override {

return availability;

}

void checkOut() override {

if (availability) {

availability = false;

std::cout << "Book checked out successfully." << std::endl;

}

else {

std::cout << "Book is not available." << std::endl;

}

}

void checkIn() override {

availability = true;

std::cout << "Book checked in successfully." << std::endl;

}

};

**Journal Class**

class Journal : public Item {

private:

std::string publisher;

int issueNumber;

public:

Journal(std::string id, std::string title, bool available, std::string publisher, int issueNumber)

: Item(id, title, available), publisher(publisher), issueNumber(issueNumber) {}

void getItemDetails() const override {

std::cout << "Journal ID: " << itemID << "\nTitle: " << title << "\nPublisher: " << publisher << "\nIssue Number: " << issueNumber << "\nAvailable: " << (availability ? "Yes" : "No") << std::endl;

}

bool checkAvailability() const override {

return availability;

}

void checkOut() override {

if (availability) {

availability = false;

std::cout << "Journal checked out successfully." << std::endl;

}

else {

std::cout << "Journal is not available." << std::endl;

}

}

void checkIn() override {

availability = true;

std::cout << "Journal checked in successfully." << std::endl;

}

};

**3. Class: Member**

class Member {

private:

std::string memberID;

std::string name;

std::string contactInfo;

int loanLimit;

std::vector<std::string> loanHistory;

public:

Member(std::string id, std::string name, std::string contact, int limit)

: memberID(id), name(name), contactInfo(contact), loanLimit(limit) {}

void borrowItem(Item\* item) {

if (loanHistory.size() < loanLimit && item->checkAvailability()) {

item->checkOut();

loanHistory.push\_back(item->getItemDetails());

}

else {

std::cout << "Cannot borrow item. Loan limit reached or item unavailable." << std::endl;

}

}

void returnItem(Item\* item) {

item->checkIn();

}

void getLoanHistory() const {

std::cout << "Loan History for " << name << ":" << std::endl;

for (const auto& history : loanHistory) {

std::cout << history << std::endl;

}

}

};

**4. Class: Loan**

class Loan {

private:

std::string loanID;

Item\* item;

Member\* member;

std::string dueDate;

std::string actualReturnDate;

public:

Loan(std::string id, Item\* item, Member\* member, std::string dueDate)

: loanID(id), item(item), member(member), dueDate(dueDate), actualReturnDate("") {}

void returnItem(std::string returnDate) {

actualReturnDate = returnDate;

item->checkIn();

}

double calculateFine() const {

int overdueDays = 0;

return overdueDays > 0 ? overdueDays \* 1.0 : 0.0;

}

};

**Additional Features**

std::map<std::string, std::vector<Member\*>> reservationSystem;

void reserveItem(Item\* item, Member\* member) {

if (!item->checkAvailability()) {

reservationSystem[item->getItemDetails()].push\_back(member);

std::cout << "Item reserved successfully." << std::endl;

}

else {

std::cout << "Item is available; no need to reserve." << std::endl;

}

}

**Search Functionality**

std::vector<Item\*> libraryItems;

std::vector<Item\*> searchByTitle(const std::string& title) {

std::vector<Item\*> results;

for (Item\* item : libraryItems) {

if (item->title.find(title) != std::string::npos) {

results.push\_back(item);

}

}

return results;

}

**Report Generation**

void generateOverdueReport(const std::vector<Loan\*>& loans) {

std::cout << "Overdue Items Report:" << std::endl;

for (const Loan\* loan : loans) {

if (loan->calculateFine() > 0) {

std::cout << "Loan ID: " << loan->loanID << " - Fine: $" << loan->calculateFine() << std::endl;

}

}

}