**Name: - ARKAJYOTI NASKAR**

**Roll: - 002110501144**

**Class: - BCSE-II**

**sem: - first**

**session: - 2022-2023**

**Assignment 4 Q.1:** Design the class(es) for the following. Each account has account number and

balance amount. A list of account is to be maintained where one can add and find account,

display information of all accounts. While adding, account number must be unique. Withdraw

object has account number (must exist) and amount (will not exceed balance amount of

corresponding account). Withdraw object will update the balance of corresponding account in

the list. User will be able to search and view account, add account and withdraw money from

the account.

#include<bits/stdc++.h>

using namespace std;

class Account {

private:

    int accountNumber;

    double balance;

public:

    int getAccountNumber() { return accountNumber;}

    double getBalance() { return balance;}

void setAccountNumber(int accountNumber){this->accountNumber = accountNumber;}

    void setBalance(double balance) { this->balance = balance;}

};

class Withdraw {

private:

    int accountNumber;

    double amount;

public:

    Withdraw(int acc\_num, double amnt)

    {

        this->accountNumber=acc\_num;

        this->amount=amnt;

    }

    int getAccountNumber() { return accountNumber;}

    double getAmount() { return amount;}

};

class AccountList

{

    public:

    vector<Account> accountList;

    static int curr;

    static void update(){curr++;}

    void create\_account(int accountNumber, double balance);

    /\*creating an account with account number and balance and adding it in the accountList and updating the current no. of accounts in the bank. \*/

    bool findAccount(int accountNumber);

 /\*Searching in accountList, if any account with the given number exists returns true, otherwise returns false. \*/

    void displayAccounts();

/\* Traverses the accountList and displays all the existing account with their account number and current balance. \*/

    void updateBalance(Withdraw w);

         /\* if account with given account number is found then it withdraws the given amount and update the balance of that account, if account balance is low then withdraw is unsuccessful. \*/

class Bank

{

    public:

      static void menu() {

          int choice; int acc\_num; double balance; AccountList AL;

         do{

        cout << "1. Add Account" << endl;

        cout << "2. Find Account" << endl;

        cout << "3. Display Account" << endl;

        cout << "4. Withdraw Balance" << endl;

        cout << "5. Exit" << endl;

        cout << "Enter your choice: "; cin >> choice;

        switch (choice) {

            case 1:{

              cin>>acc\_num; cin>>balance;

                AL.create\_account(acc\_num,balance); break;}

            case 2:{

                cin>>acc\_num;

                if(AL.findAccount(acc\_num)) cout<<"Account Found"<<endl;

                else cout<<"Invalid Account Number"<<endl;

                break;}

            case 3:{

                AL.displayAccounts(); break;}

            case 4:{

              cin>>acc\_num; cin>>balance;

                Withdraw w(acc\_num, balance);

                AL.updateBalance(w);

                break;}

            case 5:{exit(1); break;}

            default:{ cout << "Invalid choice" << endl;   break;}

        }

         }while(choice!=5);

     }

};

int AccountList:: curr =0;

int main()

{

    Bank::menu();

    return 0;

}

**Assignment 5 Q.2:** In a library, for each book book-id, serial number (denotes copy number of a

book), title, author, publisher and price are stored. Book-id and serial number together will be

unique identifier for a book. Members are either student or faculty. Each member has unique

member-id. Name, e-mail, address are also to be stored. For any transaction (book issue or

return), members are supposed to place transactions slip. User will submit member-id, book-id,

and serial number (only for book return). While processing a transaction, check the validity of

the member. While issuing, availability of a copy of the book is to be checked. While returning a

book, it is to be checked whether this copy was issued to the member or not. A student member

can have 2 books issued at a point of time. For faculty members it is 10. Transaction information

is to be stored like date of transaction, member-id, book-id, serial number, returned or not. An

entry is made when book is issued and updated when the book is returned.

For list consider memory data structure.

#include<bits/stdc++.h>

using namespace std;

class Book {

int bookId; int serialNum; string title; string author; string publisher; double price; int no\_of\_copies;

public:

    Book(int Id, string title, string auth, string pub, double price, int copies){

        this->bookId=Id;  this->title=title ; this->author=auth ; this->publisher=pub; this->price=price;this->no\_of\_copies=copies;

    }

    vector<pair<int ,bool>> all\_copies;

    int getBookId() { return bookId;}

    string getTitle(){ return title;}

    string getAuthor(){ return author;}

    string getPublisher(){ return publisher;}

    double getPrice(){ return price;}

    int getcopies(){ return no\_of\_copies;}

    void store\_copies(int no\_of\_copies);

/\*It creates a pair of serial number and a Boolean value representing they are available or not, when adding a book, all copies get added with unique serial number and all are made available first.\*/

};

class BookList{

    public:

    static vector<Book> books;

    static void addBook(Book book){ books.push\_back(book); }

    static int checkAvailability(int bookId);

       /\*It checks if any copy of the book with the given book id is available or not, if available then it is returning a serial number else returning 0. \*/

    static void display\_book();

/\*It displays all the books along with their book id, author, publisher and price. \*/

    static void change\_availability(int bookId,int serial,bool k);

/\*In the function of issuing or returning book , the book with the given book id and serial number made available after returning or made unavailable after issuing. \*/

}; vector<Book> BookList::books;

class Member {

int memberId; string name; string email; string address; int no\_of\_books; bool Student;

static int maxBooks\_for\_student; static int maxBooks\_for\_faculty;

public:

    Member(int memberId, string name, string email, string address, bool k){

      this->memberId=memberId; this-> name=name ; this->email=email ; this->address= address; this->Student=k; this->no\_of\_books=0; }

    int getMaxBooks(bool Student) { if(Student) return maxBooks\_for\_student; else return maxBooks\_for\_faculty; }

    int getMemberId(){ return memberId;}

    bool isStudent(){ return Student; }

    int get\_no\_of\_books\_issued(){ return no\_of\_books;}

    void set\_no\_of\_books\_issued(int i);

/\*It increases or decreases the no. of books issued by a member depending upon situation.\*/

};

int Member:: maxBooks\_for\_student=2;

int Member:: maxBooks\_for\_faculty=10;

class MemberList{

    public:

    static vector<Member> members;

    static void addMember(Member member){ members.push\_back(member);}

    static bool checkValidity(int memberId);

/\*It searches the member with the given member id and if the no. of issued books is less than that of max books that member can issue then it is returning true, otherwise printing an error message and returns false. \*/

    static void edit\_member\_log(int id,int i);

        /\*It finds the member with id and increases or decreases the no. of issued books. \*/

}; vector<Member> MemberList::members;

class Transaction {

    int memberId; int bookId; int serialNum; string issue\_date; bool returned; string return\_date;

public:

    Transaction(int mId, int bId, int sNum, string d, bool returned){

       this->memberId=mId; this->bookId=bId; this->serialNum=sNum ; this->issue\_date= d; this->returned= returned; this->return\_date=" "; }

    int getMemberId(){ return memberId;}

    int getBookId(){ return bookId;}

    int getSerialNum(){ return serialNum;}

    string getIssueDate(){ return issue\_date;}

    bool getReturned(){return returned;}

    void setReturned(bool r){ this->returned=r;}

    void setReturnDate(string d){this->return\_date=d;}

};

class Library {

public:

    vector<Transaction> transactions;

    map<pair<int,int> ,int> pending;

void addTransaction(Transaction transaction) { transactions.push\_back(transaction); }

    bool issueBook(int memberId, int bookId,string date);

/\*it checks if the member is valid or not , checks the availability of the book with the given id, if both condition satisfies make a Transaction, add it to the transaction history, change the availability of the book, edit the member's no of issued books, and prints the transaction slip.\*/

    bool returnBook(int memberId, int bookId, int serialNum, string date);

/\*If the book with given id and serial number is issued to that member in pending then it deletes the pending record, changes the transaction state of returning and includes the returning date, made the book available and decreases the no of issued books of that member. \*/

};

class Library\_Manager{

    public:

     static void menu() {

           int choice; BookList BL; MemberList ML; Library L;

         do{

        cout << "1. Add Book" << endl;

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

        cout << "3. Display All books"<<endl;

        cout << "4. Issue book" << endl;

        cout << "5. Return book" << endl;

        cout << "6. Exit" << endl;

        cout << "Enter your choice: ";cin >> choice;

        switch (choice) {

            case 1: //Book is added to the book list.

            case 2: //Member is added to the member list.

            case 3: //Displaying all the books.

            case 4: //Issued the book.

            case 5: //Book returning operation is done here.

            case 6: {exit(1); break;}

            default:

                cout << "Invalid choice" << endl; break;

        }

         }while(choice!=6);}

};

int main(){

    Library\_Manager::menu();

    return 0;

}