**CA 3: Experiential Learning**

**Group Members:**

|  |  |  |  |
| --- | --- | --- | --- |
| Sr. No. | PRN | Name of Student | Mail id |
| 1 | 23070122170 | Raghav Dhoot | raghav.dhoot.btech2023@sitpune.edu.in |
| 2 | 23070122174 | Atharva Rale | atharva.rale.btech2023@sitpune.edu.in |
| 3 | 24070122507 | Malhar Borse | malhar.borse.btech2023@sitpune.edu.in |

**Problem Statement:** Ticket booking system

**Brief Explanation:**

**1. Introduction**

This report summarizes the implementation of a ticket booking system in C++ that allows users to book train and flight tickets. The system uses inheritance and polymorphism to handle different types of tickets in a unified manner. The code is designed to handle multiple bookings, display detailed ticket information, and manage memory effectively, demonstrating the practical application of OOP concepts like inheritance and polymorphism in a real-world scenario. The structure of this code improves flexibility, maintainability, and scalability of the system, allowing for easy addition of new ticket types in the future.

**2. Code Explanation**

The ticket booking system is implemented with a user-friendly menu that allows the booking of either train or flight tickets. Below is a breakdown of how the system works:

Ticket Booking Process:

* The ticketBookingSystem() function is the central control point of the program. It displays a menu for the user to select whether they want to book train or flight tickets.
* After choosing the type of ticket, the user is prompted to enter how many tickets they want to book in that transaction, with a limit of 5 tickets per transaction.
* A loop is used to collect details for each ticket (such as passenger name, age, source, and destination). Depending on whether the ticket is a train or flight ticket, additional information is collected (train number and seat class for trains; flight number and seat number for flights).

Multiple tickets in one transaction:

* The system allows booking up to 5 tickets in one transaction. If a user tries to book more than 5 tickets, they are informed that the maximum limit is 5, and they are asked to re-enter the number of tickets.
* Once the tickets are booked, the user can choose to book another transaction or exit the system.

Display and Memory Management:

* After booking tickets, the system displays the total number of tickets booked, along with the detailed information for each ticket.
* Dynamically allocated memory for the tickets is cleaned up at the end of the process to avoid memory leaks.

**3. Inheritance**

Inheritance allows the creation of specialized classes (TrainTicket and FlightTicket) from a more general class (Ticket). The ticket booking system demonstrates the use of inheritance to create different types of tickets while maintaining a shared structure for common attributes and behaviors.

Base Class: Ticket

The Ticket class acts as a blueprint for all types of tickets. It contains attributes such as:

* passengerName: The name of the passenger.
* age: The age of the passenger.
* source: The starting point of the journey (station or airport).
* destination: The ending point of the journey (station or airport).

This class also declares two pure virtual functions (bookTicket() and displayTicket()), making it an abstract class. These functions must be implemented by any derived class.

Derived Classes: TrainTicket and FlightTicket

The TrainTicket and FlightTicket classes inherit the common attributes from the Ticket class and provide specific implementations for booking and displaying tickets.

TrainTicket adds:

* trainNumber: The number of the train.
* seatClass: The class of the seat (e.g., Sleeper, AC).

FlightTicket adds:

* flightNumber: The number of the flight.
* seatNumber: The seat number in the flight.

By using inheritance, we avoid duplicating the common properties (passenger name, age, etc.) in both classes, ensuring code reusability and maintainability.

**4. Polymorphism**

Polymorphism is achieved through the use of virtual functions, allowing the system to work with different types of tickets dynamically at run-time. This makes the system flexible and adaptable to different types of tickets using a common interface.

Virtual Functions:

* In the Ticket class, the bookTicket() and displayTicket() functions are declared as pure virtual functions. This requires derived classes (TrainTicket and FlightTicket) to implement these functions according to their specific needs.
* The bookTicket() function is overridden in both derived classes to handle ticket-specific details (such as collecting seat class for trains and seat number for flights).
* Similarly, the displayTicket() function is overridden to display the relevant ticket information.

Dynamic Dispatch via Base Class Pointer:

* A key aspect of polymorphism in the program is the use of a base class pointer (Ticket\* ticket). Depending on the user’s choice, this pointer is assigned either a TrainTicket or FlightTicket object.
* At run-time, the system uses dynamic dispatch to determine which version of the bookTicket() and displayTicket() methods should be called, based on the actual type of the object (TrainTicket or FlightTicket) that the base class pointer refers to. This flexibility allows the system to handle different types of tickets without needing to know their exact type at compile time.

Benefits of Polymorphism:

The system can easily be extended to support new ticket types (e.g., BusTicket) by creating a new derived class that implements the bookTicket() and displayTicket() methods. The core logic of the system (menu, booking flow, etc.) does not need to be modified, as it operates on the common Ticket interface.

**5. Benefits of Using Inheritance and Polymorphism**

Code Reusability:

The Ticket class defines attributes and methods that are common to all ticket types. These are inherited by TrainTicket and FlightTicket, avoiding duplication and promoting clean, maintainable code.

Extensibility:

The system is easy to extend. If a new ticket type (e.g., BusTicket) needs to be added, it can simply inherit from Ticket and provide its own implementations of the bookTicket() and displayTicket() methods. No changes are needed in the existing code for handling different ticket types.

Flexibility:

Through polymorphism, the program can handle different ticket types using a common interface (the Ticket\* pointer). This allows the program to remain flexible and adaptable to changes or new requirements.

Dynamic Behavior:

The program dynamically adapts at run-time based on the user’s choice (train or flight ticket). The correct version of the ticket booking and display functions is automatically selected depending on the actual type of ticket, showcasing the power of run-time polymorphism.

**6. Conclusion**

This ticket booking system efficiently demonstrates the use of inheritance and polymorphism in C++. Inheritance is used to share common code across ticket types, while polymorphism enables the program to handle different ticket types dynamically at run-time. This design makes the system both flexible and scalable, allowing for easy extension and maintenance, which are key advantages of object-oriented programming.

**Code snippets:**

#include <iostream>

#include <string>

#include <vector>

#include <limits>

using namespace std;

class Ticket {

protected:

    string passengerName;

    int age;

    string source;

    string destination;

public:

    virtual void bookTicket() = 0;

    virtual void displayTicket() const = 0;

    virtual ~Ticket() {}

};

string validateStringInput(const string& prompt) {

    string input;

    do {

        cout << prompt;

        getline(cin, input);

        if (input.empty()) {

            cout << "Invalid input. Please enter a valid non-empty string.\n";

        }

    } while (input.empty());

    return input;

}

int validateIntegerInput(const string& prompt) {

    int input;

    while (true) {

        cout << prompt;

        cin >> input;

        if (cin.fail() || input <= 0) {

            cout << "Invalid input. Please enter a valid positive integer.\n";

            cin.clear();

            cin.ignore(numeric\_limits<streamsize>::max(), '\n');

        } else {

            break;

        }

    }

    cin.ignore(numeric\_limits<streamsize>::max(), '\n');

    return input;

}

class TrainTicket : public Ticket {

private:

    string trainNumber;

    string seatClass;

public:

    void bookTicket() override {

        passengerName = validateStringInput("Enter passenger name: ");

        age = validateIntegerInput("Enter passenger age: ");

        source = validateStringInput("Enter source station: ");

        destination = validateStringInput("Enter destination station: ");

        trainNumber = validateStringInput("Enter train number: ");

        seatClass = validateStringInput("Enter seat class (e.g., Sleeper, AC): ");

        cout << "Train ticket booked successfully!\n";

    }

    void displayTicket() const override {

        cout << "Train Ticket Details:\n";

        cout << "Passenger Name: " << passengerName << "\n";

        cout << "Age: " << age << "\n";

        cout << "Source: " << source << "\n";

        cout << "Destination: " << destination << "\n";

        cout << "Train Number: " << trainNumber << "\n";

        cout << "Seat Class: " << seatClass << "\n";

    }

};

class FlightTicket : public Ticket {

private:

    string flightNumber;

    string seatNumber;

public:

    void bookTicket() override {

        passengerName = validateStringInput("Enter passenger name: ");

        age = validateIntegerInput("Enter passenger age: ");

        source = validateStringInput("Enter source airport: ");

        destination = validateStringInput("Enter destination airport: ");

        flightNumber = validateStringInput("Enter flight number: ");

        seatNumber = validateStringInput("Enter seat number: ");

        cout << "Flight ticket booked successfully!\n";

    }

    void displayTicket() const override {

        cout << "Flight Ticket Details:\n";

        cout << "Passenger Name: " << passengerName << "\n";

        cout << "Age: " << age << "\n";

        cout << "Source: " << source << "\n";

        cout << "Destination: " << destination << "\n";

        cout << "Flight Number: " << flightNumber << "\n";

        cout << "Seat Number: " << seatNumber << "\n";

    }

};

void ticketBookingSystem() {

    vector<Ticket\*> tickets;

    int choice;

    char bookMore;

    int ticketCount;

    do {

        Ticket\* ticket = nullptr;

        while (true) {

            cout << "\nWelcome to the Ticket Booking System\n";

            cout << "1. Book Train Ticket\n";

            cout << "2. Book Flight Ticket\n";

            cout << "Enter your choice: ";

            cin >> choice;

            if (choice != 1 && choice != 2) {

                cout << "Invalid choice. Please enter 1 for Train Ticket or 2 for Flight Ticket.\n";

                cin.clear();

                cin.ignore(numeric\_limits<streamsize>::max(), '\n');

            } else {

                break;

            }

        }

        while (true) {

            cout << "How many tickets do you want to book? (Maximum 5): ";

            cin >> ticketCount;

            if (cin.fail() || ticketCount <= 0 || ticketCount > 5) {

                cout << "Invalid input. You can only book up to 5 tickets in a single transaction.\n";

                cin.clear();

                cin.ignore(numeric\_limits<streamsize>::max(), '\n');

            } else {

                break;

            }

        }

        cin.ignore(numeric\_limits<streamsize>::max(), '\n');

        for (int i = 0; i < ticketCount; ++i) {

            cout << "\nBooking Ticket " << i + 1 << ":\n";

            if (choice == 1) {

                ticket = new TrainTicket();

            } else if (choice == 2) {

                ticket = new FlightTicket();

            }

            ticket->bookTicket();

            tickets.push\_back(ticket);

        }

        cout << "\nDo you want to book another transaction? (y/n): ";

        cin >> bookMore;

    } while (bookMore == 'y' || bookMore == 'Y');

    cout << "\n-------------------------\n";

    cout << "Total Tickets Booked: " << tickets.size() << "\n";

    cout << "Ticket Details:\n";

    for (size\_t i = 0; i < tickets.size(); ++i) {

        cout << "\nTicket " << i + 1 << ":\n";

        tickets[i]->displayTicket();

    }

    for (Ticket\* t : tickets) {

        delete t;

    }

}

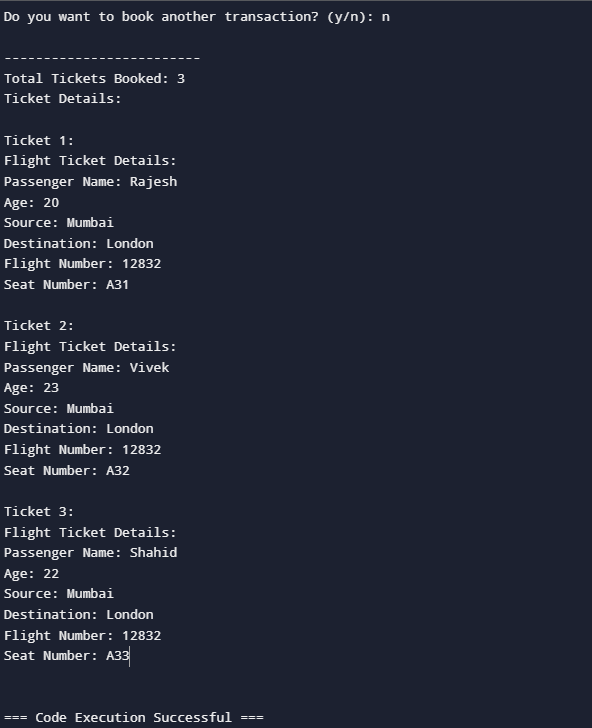
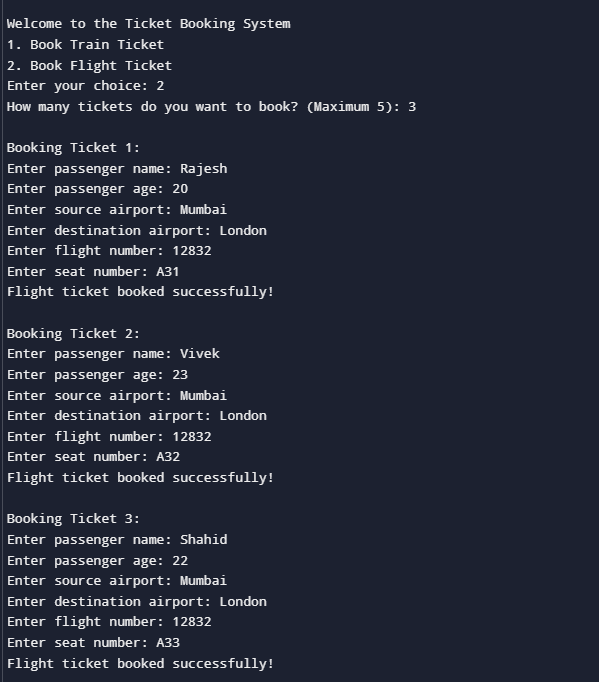
int main() {

    ticketBookingSystem();

    return 0;

}

**Input/Output:**



Github repository link: