Airflight Booking System – Encapsulation

Purpose of work

Gain practical skills in implementing encapsulation principles in object-oriented programming. The goal is to create self-sufficient classes with clearly defined responsibilities and minimal coupling.

Task

Develop a console-based airplane ticket booking system. The system should read input configuration from a text file and allow users to book tickets via a command-line user interface. Emphasis should be placed on encapsulating the data and behavior within well-organized classes.

!Attention!
The config file during passing the assignment will be different to what is mentioned in this document. New config file will contain more flights, rows inside a flight, different prices for different rows etc.

Requirements

1. Class Decomposition:
    - Create a class for representing an `Airplane` with attributes like the number of seats, seat availability, and any other relevant information.
    - Create a class for representing a `Ticket` with attributes like passenger name, seat number, flight information, and booking status.
    - Encapsulate the data members in these classes using appropriate access specifiers (e.g., private or protected).
    - Define clear responsibilities for each class. For example, the `Airplane` class should manage seat availability and bookings, while the `Ticket` class should handle ticket-related information.
2. Input Configuration (see example below):
    - Read the input configuration (e.g., airplane details, flight data) from a text file.
    - Encapsulate the logic for reading and processing this configuration in a separate class to keep the main program clean.
3. Command-line Interface:
    - Develop a user interface that allows users to perform the following commands:
        - Check seat availability.
        - Book a ticket.
        - Return a ticket.
        - View booked tickets.
    - Each of these commands should be implemented as methods in the main program, which interact with the `Airplane` and `Ticket` classes.
4. Encapsulation:
    - Ensure that the `Airplane` and `Ticket` classes encapsulate their data members and provide public methods to interact with them. For instance, booking a ticket

should be done through a method in the `Airplane` class that ensures the encapsulated seat availability data is updated correctly.

5. Data Validation:
   - Implement data validation within the classes. For example, when booking a ticket, validate that the requested seat is available.
6. File Handling:
   - Encapsulate file reading operations in a separate class (e.g., `FileReader`) to promote encapsulation and separation of concerns, also to allow working with different file formats in the future by replacing only 1 class in the code.
7. Error Handling:
   - Implement error handling to gracefully handle situations like invalid input or file not found errors.
8. Documentation / theory learning:
   - Include clear comments and documentation to explain how encapsulation principles are applied within the project.

Command-line interface

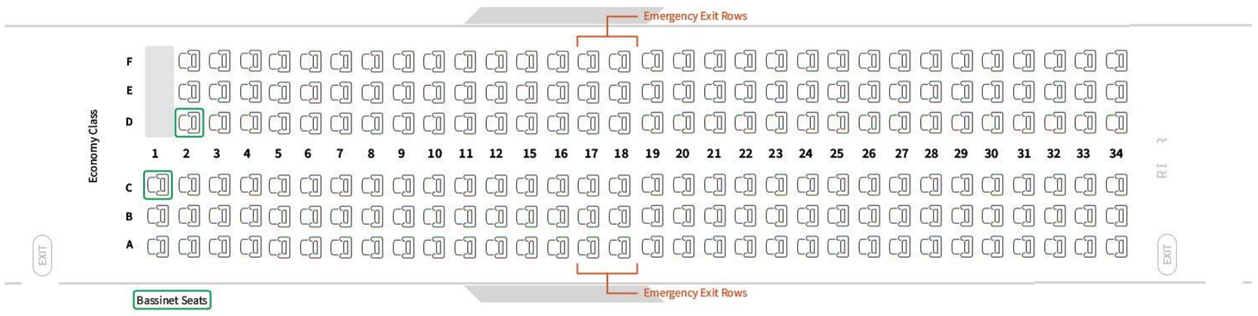
Command	**Parameters**	**Description**	**Result**	**Example**
check	Date, FlightNo	Check available places for the flight	List of free spaces with prices	> check 1.12.23 FR12 > 1A 100$, 2B 100 $
book	Date, FlightNo, place, Username	Buy a ticket for the flight	Booking confirmation ID	> book 1.12.23 FR12 1A AdamSmith > Confirmed with ID 12345
return	ID	Return ticket with refund	Confirmation	> return 12345 > Confirmed 100$ refund for AdamSmith
view	ID	View the booking confirmation info	Flight number, date, seat, and the ticket’s price	> view 12345 > Flight FR12, 1.12.23, seat 1A, price 100$, AdamSmith
view	Username	View all booked tickets for a particular user	For each flight: flight number, date, seat, and the ticket’s price	> view username AdamSmith > 1. Flight FR12, 1.12.23, seat 1A, price 100$ 2. Flight FQ11, 5.12.23, seat 11B, price 10$
view	Date, FlightNo	View all booked tickets for the particular flight	For the chosen flight: seat, user name, price	> view flight FR12 1.12.23 > 1A AdamSmith 100$ 1B MarySmith 100$

Example of the input configuration file (// please ignore comments!!!):

```
2 // number of records
11.12.2022 FQ12 6 1-20 100$ 21-40 50$ // date, flight number, number of seats per row (6 means A B C D E F), price for rows 1-20 is 100$, price for rows 21-40 is 50$
11.12.2022 HJ114 9 1-10 10$ 11-50 20$ // date, flight number, number of seats per row (6 means A B C D E F G H I), price for rows 1-10 is 10$, price for rows 11-50 is 20$
```

!Attention!
The config file during passing the assignment will be different to what is mentioned in this document. New config file will contain more flights, rows inside a flight, different prices for different rows etc.

Explanation of a plane seat map (just as example):



Task milestones

1. Design your system on the paper or by using some online tools, please use squares and arrows to draw classes and their relations, no specific notation is needed in this task, think in advance how would your system look like.
2. Implement the I/O system to parse input file, create a class for user input processing
3. Implement logic to book the ticket, create appropriate classes and relations between them, consider effective data structures
4. Implement logic to view booking results
5. Test the solution, develop your own input data files

Code examples

In this example, the Student class encapsulates the name and age data members and provides getter and setter methods to access and modify them, respectively.

```
#include <iostream>
#include <string>

class Student {
private:
    std::string name;
    int age;
```

```cpp
public:
    // Constructor to initialize the object
    Student(const std::string& n, int a) : name(n), age(a) {}

    // Getter method for the name
    std::string getName() const {
        return name;
    }

    // Setter method for the age
    void setAge(int a) {
        if (a >= 0 && a <= 120) {
            age = a;
        }
        else {
            std::cout << "Invalid age!" << std::endl;
        }
    }
};

int main() {
    // Creating a Student object
    Student student("Alice", 20);

    // Accessing and modifying private data using getter and setter
    std::cout << "Student's name: " << student.getName() << std::endl;
    student.setAge(22);
    std::cout << "Student's age: " << student.getName() << std::endl;

    // Trying to set an invalid age
    student.setAge(150); // Outputs "Invalid age!"

    return 0;
}
```

This example shows encapsulation within an inheritance hierarchy. The Person class encapsulates the name attribute, and the Student class extends Person and encapsulates the age attribute.

```cpp
#include <iostream>
#include <string>

class Person {
private:
    std::string name;

public:
    Person(const std::string& n) : name(n) {}

    std::string getName() const {
        return name;
    }
};

class Student : public Person {
private:
    int age;
public:
    Student(const std::string& n, int a) : Person(n), age(a) {}

    int getAge() const {
        return age;
    }
};

int main() {
    Student student("Bob", 18);

    std::cout << "Student's name: " << student.getName() << std::endl;
    std::cout << "Student's age: " << student.getAge() << std::endl;

    return 0;
}
```

Control questions

1. Explain data abstraction and encapsulation programming technique.
2. What is an interface?
3. Which access specifiers do you know?
4. What is the difference between struct and class in C++?
5. What is **this** pointer?
6. How can you initialize fields in the object?
7. What is constructor?
8. What is destructor?
9. What is a friend class?
10. What is const member function?
11. What does mutable data member?
12. What is objects composition?

Evaluation

Practice

Encapsulation and data abstraction principles usage	3
Correct view functionality and optimal data structures usage	3
Effective memory usage	1
System design (test description, photo of your notes or some diagrams)	1
Git (3+ meaningful commits)	1
Extra: RAII wrapper around system file calls (winapi/posix)	1

Total	9 + 1

Theory

Test in Moodle	1

Please be aware that evaluation points may be reduced due to incorrect answers, program output, or logic issues. To ensure full credit, make sure to submit all materials in PDF format to Moodle before the deadline and present your results to the teaching assistant, demonstrating that the work is your own. All git submissions must be done before the deadline.

Links

C++ Primer 5th Edition: Chapter 7