

INSTITUT SUPERIEUR DE GENIE ELECTRIQUE DU BURKINA FASO



Filière : Réseaux des Systèmes des Télécommunications

RAPPORT DE PROJET

**Système de gestion des vols**

**Groupe :**

FORGO Ibrahim

ILBOUDO P.Marco Cesaro

LENGLENGUE Kiswendsida Farida

Professeur : M. YAMEOGO Botozou Antoine

Année Académique 2024 – 2025

# Introduction

Ce rapport présente le projet de développement d'une application de gestion des vols, réalisée en C++ selon le paradigme de la Programmation Orientée Objets (POO). L'objectif principal était de concevoir un logiciel simulant les opérations d'une compagnie aérienne, incluant la gestion des passagers, des vols et des réservations. Ce projet a permis d'appliquer concrètement les concepts de la POO pour créer une solution modulaire, robuste et conviviale.

## 1. Outils Utilisés

Le développement de cette application a reposé sur des outils standards du développement C++, choisis pour leur efficacité et leur adéquation avec les principes de la POO.

|  |  |
| --- | --- |
| **Outils** | **Rôle dans le Projet** |
| **Langage C++** | Langage principal, choisi pour son support robuste de la POO (classes, objets, encapsulation). |
| **Paradigm Orienté Objets (POO)** | Approche de conception structurant le code en entités autonomes et interconnectées, favorisant la modularité et la réutilisabilité. |
| **Environnement de Développement Intégré (IDE)** | Facilite l'écriture, la compilation et le débogage du code (Visual Studio Code). |
| **Compilateur (g++, MinGW)** | Traduit le code source C++ en un programme exécutable. |
| **Fichiers .h et .cpp** | Assurent la modularité du code en séparant les déclarations des implémentations de classes. |
| **Fichiers .txt** | Utilisés pour la persistance des données, permettant de sauvegarder et charger l'état de l'application. |

## 2. Présentation des Classes Principales

La conception de notre système est basée sur **six (6) classes** principales, chacune ayant une responsabilité claire et unique, conformément aux principes de la POO.

|  |  |  |
| --- | --- | --- |
| **Nom de la Classe** | **Responsabilité** | **Justification de l'Utilité** |
| Vol | Représente un vol avec ses détails (numéro, capacité, dates, destination) et gère ses passagers. | Encapsule toutes les données et comportements liés à un vol, assurant cohérence et modularité. |
| Passager | Gère les informations personnelles d'un passager (nom, passeport, âge, etc.). | Centralise les données des passagers, facilite leur identification et leur validation. |
| Reservation | Crée un lien entre un Vol et un Passager, et gère le statut de confirmation. | Gère la relation entre vols et passagers, et le suivi de l'état de chaque réservation. |
| Date | Gère les informations temporelles (jour, mois, année, heure, minute) et leur validation. | Isole la logique de gestion des dates, assurant réutilisabilité et validation cohérente (ex: "Contrôle Date"). |
| Destination | Définit les villes de départ et d'arrivée d'un vol. | Regroupe les informations de destination, améliorant la clarté et la maintenabilité du code. |
| SystemeDeGestion | Orchestre toutes les interactions entre les autres classes et gère les collections de données (vols, réservations). | Agit comme un contrôleur central, exposant les fonctionnalités principales à l'utilisateur et gérant le flux de l'application. |

## 3. Fonctionnalités Clés

L'application offre un ensemble de fonctionnalités essentielles pour la gestion des vols.

|  |  |
| --- | --- |
| **Fonctionnalités** | **Descriptions** |
| **Ajout de vol** | Enregistre un nouveau vol avec ses caractéristiques (numéro, capacité, prix, dates, destination). |
| **Affichage des vols** | Affiche la liste complète de tous les vols disponibles. |
| **Annulation de vol** | Supprime un vol existant du système, annulant également les réservations associées. |
| **Modification de vol** | Permet de mettre à jour la date de départ d'un vol. |
| **Réservation de vol** | Enregistre une nouvelle réservation pour un passager sur un vol spécifique. |
| **Confirmation de réservation** | Change le statut d'une réservation de "non confirmée" à "confirmée". |
| **Affichage des passagers d'un vol** | Liste tous les passagers enregistrés pour un vol donné. |
| **Recherche de vol** | Vérifie l'existence d'un vol par son numéro et renvoie sa position. |
| **Recherche de réservation** | Vérifie l'existence d'une réservation par passeport et numéro de vol. |
| **Contrôle de la date** | Empêche la planification de vols à des dates déjà passées. |
| **Persistance des données** | Sauvegarde et charge toutes les données (vols, réservations) depuis un fichier .txt. |
| **Export de rapport lisible** | Génère un fichier texte formaté pour une lecture facile des données clés du système. |

## 4. Explication de Quelques Méthodes Importantes

Pour comprendre le fonctionnement interne, voici un aperçu de méthodes clés.

### bool Vol::ajouterPassager(const Passager& p)

Cette méthode est cruciale pour enregistrer un passager sur un vol. Avant d'ajouter le passager, elle effectue deux vérifications :

* Elle s'assure que le vol n'est pas complet.
* Elle vérifie que le passager (identifié par son numéro de passeport) n'est pas déjà enregistré sur ce vol. Si ces conditions sont respectées, le passager est ajouté à la liste du vol, et le nombre de places réservées est mis à jour.

### bool Date::estDansLePasse() const

Cette méthode de la classe Date est essentielle pour la validation des horaires. Elle compare la date et l'heure de l'objet Date avec l'heure actuelle du système. Si la date est antérieure à l'instant présent, la méthode retourne true, empêchant ainsi la création ou la modification de vols avec des horaires irréalistes. C'est le mécanisme derrière le "Contrôle Date".

### void Passager::saisir()

Cette méthode permet la saisie interactive des informations d'un passager via la console. Son importance réside dans sa **validation robuste des entrées**. Pour chaque champ (nom, âge, etc.), elle utilise des boucles et des fonctions de gestion d'erreurs (std::cin.fail(), std::cin.clear(), std::cin.ignore()) pour s'assurer que l'utilisateur fournit des données valides. L'utilisateur est invité à ressaisir l'information jusqu'à ce qu'elle soit correcte, garantissant la qualité des données.

### void SystemeDeGestion::exporterLisible(const std::string& chemin) const

Cette méthode génère un rapport des vols et réservations dans un fichier texte (.txt) facile à lire. Contrairement à la sauvegarde interne, qui est compacte pour le programme, cette fonction formate les informations de manière claire et structurée avec des en-têtes et des détails pertinents. Elle permet une consultation rapide de l'état du système sans avoir à exécuter l'application.

## 5. Défis Rencontrés et Solutions Apportées

Le développement a présenté plusieurs défis, dont la résolution a renforcé la robustesse de l'application.

|  |  |
| --- | --- |
| **Défi Rencontré** | **Solution Apportée** |
| **Formatage des données pour l'export** | Utilisation intensive des manipulateurs de flux (std::setw, std::setfill, std::left/std::right) de la bibliothèque <iomanip> pour créer des sorties tabulaires alignées et lisibles dans les fichiers exportés. |
| **Gestion du temps réel (validation des dates)** | Implémentation de la méthode Date::estDansLePasse() utilisant la bibliothèque <ctime> pour comparer les dates saisies avec l'heure système, empêchant la planification de vols passés. |
| **Prévention des doublons dans les réservations** | Mise en place de vérifications avant l'ajout d'une réservation : le vol ne doit pas être complet, et le passager ne doit pas déjà avoir de réservation pour ce vol. |
| **Traitement robuste des entrées utilisateurs** | Intégration de boucles de validation avec std::cin.fail(), std::cin.clear(), et std::cin.ignore() dans toutes les méthodes de saisie pour gérer les entrées invalides et guider l'utilisateur. |

## 6. Petites Améliorations UX Ajoutées

Au-delà des fonctionnalités de base, des améliorations ont été intégrées pour une meilleure expérience utilisateur.

* **Interface Console Lisible e :** L'affichage en console a été soigné avec des tableaux bien alignés, des bordures ASCII et des messages clairs, rendant l'application plus agréable et intuitive à utiliser.
* **Possibilité de Charger Automatiquement à l’Ouverture :** L'application charge automatiquement les données sauvegardées au démarrage, assurant la persistance de l'état du système entre les sessions sans intervention manuelle.
* **Menu Dynamique et Interactif :** Le système de menu est réactif et guide l'utilisateur à travers les options disponibles avec des invites claires, rendant les opérations complexes plus accessibles.

# Conclusion

Ce projet de gestion des vols représente une application C++ complète, stable et bien structurée, répondant aux exigences initiales. L'approche orientée objets a permis de créer une architecture r modulaire et maintenable. Les défis techniques ont été surmontés par des solutions ciblées, démontrant une capacité à résoudre des problèmes complexes et à garantir l'intégrité du système. Les innovations en matière d'expérience utilisateur, telles que l'interface soignée et la gestion automatique des données, améliorent significativement la qualité globale du logiciel. Ce projet a été une excellente opportunité d'appliquer les concepts de la POO et de développer des compétences essentielles en ingénierie logicielle.

## 8. Annexe : Code Source Complet et Commenté

Le code source complet de l'application est fourni ci-dessous, organisé par classe avec les fichiers d'en-tête (.h) et les fichiers source (.cpp) correspondants. Chaque fichier est abondamment commenté pour faciliter la compréhension de la logique, des responsabilités des classes et du but des fonctions, conformément aux exigences du projet :

# Vol.h

#ifndef VOL\_H

#define VOL\_H

#include <iostream>

#include <string>

#include <vector>

#include "Date.h"

#include "Destination.h"

#include "Passager.h"

/\*\*

\* @brief Représente un vol d’une compagnie aérienne.

\* \* Contient les informations essentielles pour gérer un vol :

\* - numéro, dates, prix, destination

\* - liste des passagers

\* - méthodes de réservation, d’affichage et de modification.

\*/

class Vol {

private:

std::string numeroVol;                 // Identifiant du vol

int nbPlacesMax;                       // Capacité totale

int nbPlacesReservees;                // Nombre de réservations faites

float prix;                            // Prix d’une place

Date dateDepart;                       // Date/heure de départ

Date dateArrivee;                      // Date/heure d’arrivée

Destination destination;               // Ville de départ/arrivée

std::vector<Passager> passagers;       // Liste des passagers

public:

// ==== Constructeurs ====

/\*\*

\* @brief Construit un vol par défaut avec des valeurs initiales.

\*/

Vol();

/\*\*

\* @brief Construit un vol avec les informations complètes.

\* @param numero Numéro du vol

\* @param maxPlaces Capacité maximale du vol

\* @param prix Prix d'une place sur ce vol

\* @param depart Date et heure de départ

\* @param arrivee Date et heure d'arrivée

\* @param dest Destination du vol

\*/

Vol(const std::string& numero,

int maxPlaces,

float prix,

const Date& depart,

const Date& arrivee,

const Destination& dest);

// ==== Réservation / Contrôle ====

/\*\*

\* @brief Vérifie si le vol est complet.

\* @return true si le nombre de places réservées atteint la capacité maximale, false sinon.

\*/

bool estComplet() const;

/\*\*

\* @brief Ajoute un passager au vol.

\* @param p L'objet Passager à ajouter.

\* @return true si le passager a été ajouté avec succès, false si le vol est complet ou le passager déjà enregistré.

\*/

bool ajouterPassager(const Passager& p);

/\*\*

\* @brief Vérifie si un passager avec un numéro de passeport donné est déjà enregistré sur ce vol.

\* @param passeport Le numéro de passeport à vérifier.

\* @return true si le passager est trouvé, false sinon.

\*/

bool contientPassager(const std::string& passeport) const;

// ==== Affichage ====

/\*\*

\* @brief Affiche les détails complets du vol dans un format lisible.

\*/

void afficher() const;

/\*\*

\* @brief Affiche la liste des passagers enregistrés pour ce vol dans un format tabulaire.

\*/

void afficherPassagers() const;

// ==== Modification ====

/\*\*

\* @brief Modifie la date de départ du vol.

\* @param nouvelleDate La nouvelle date de départ.

\*/

void modifierDate(const Date& nouvelleDate);

/\*\*

\* @brief Modifie l'heure et les minutes de la date de départ du vol.

\* @param heure La nouvelle heure.

\* @param minute Les nouvelles minutes.

\*/

void modifierHeure(int heure, int minute);

// ==== Accesseurs ====

/\*\*

\* @return Le numéro du vol.

\*/

std::string getNumeroVol() const;

/\*\*

\* @return Le nombre de places disponibles sur le vol.

\*/

int getNbPlacesDispo() const;

// ==== Export fichier ====

/\*\*

\* @brief Sérialise les informations du vol dans une ligne de texte pour la sauvegarde.

\* @return Chaîne de caractères au format CSV compatible avec les fichiers.

\*/

std::string toString() const;

};

#endif // VOL\_H

### Vol.cpp

#include "Vol.h"

#include <iostream>

#include <iomanip>

#include <algorithm> // Pour std::find\_if ou d'autres algorithmes si besoin

// === Constructeurs ===

Vol::Vol()

: numeroVol("UNKNOWN"), nbPlacesMax(0), nbPlacesReservees(0), prix(0.0f),

dateDepart(), dateArrivee(), destination() {}

Vol::Vol(const std::string& numero, int maxPlaces, float prix,

const Date& depart, const Date& arrivee, const Destination& dest)

: numeroVol(numero), nbPlacesMax(maxPlaces), nbPlacesReservees(0), prix(prix),

dateDepart(depart), dateArrivee(arrivee), destination(dest) {}

// === Réservation / Passagers ===

bool Vol::estComplet() const {

return nbPlacesReservees >= nbPlacesMax;

}

bool Vol::ajouterPassager(const Passager& p) {

if (estComplet()) {

std::cout << "Le vol est complet. Impossible d'ajouter un passager.\n";

return false;

}

if (contientPassager(p.getPasseport())) {

std::cout << "Ce passager est déjà enregistré sur ce vol.\n";

return false;

}

passagers.push\_back(p);

nbPlacesReservees++;

return true;

}

bool Vol::contientPassager(const std::string& passeport) const {

for (const auto& p : passagers) {

if (p.getPasseport() == passeport)

return true;

}

return false;

}

// === Affichage d’un vol ===

void Vol::afficher() const {

std::cout << "\n--------------------------------------------------\n";

std::cout << "VOL N°           : " << numeroVol << "\n";

std::cout << "DESTINATION      : "; destination.afficher(); std::cout << "\n";

std::cout << "DÉPART           : "; dateDepart.afficher(); std::cout << "\n";

std::cout << "ARRIVÉE          : "; dateArrivee.afficher(); std::cout << "\n";

std::cout << "PLACES RÉSERVÉES : " << nbPlacesReservees << " / " << nbPlacesMax << "\n";

std::cout << "PRIX PAR PLACE   : " << std::fixed << std::setprecision(2) << prix << " €\n";

std::cout << "--------------------------------------------------\n";

}

// === Affichage tabulaire des passagers ===

void Vol::afficherPassagers() const {

std::cout << "\n═══════════════════════════════════════════════════════\n";

std::cout << "PASSAGERS ENREGISTRÉS POUR LE VOL " << numeroVol << "\n";

std::cout << "═══════════════════════════════════════════════════════\n";

if (passagers.empty()) {

std::cout << "Aucun passager enregistré sur ce vol.\n";

return;

}

std::cout << std::left

<< std::setw(4)  << "N°"       << " | "

<< std::setw(6)  << "Titre"    << " | "

<< std::setw(14) << "Nom"      << " | "

<< std::setw(14) << "Prénom"   << " | "

<< std::setw(4)  << "Âge"      << " | "

<< "Passeport" << "\n";

std::cout << std::string(70, '-') << "\n";

int index = 1;

for (const auto& p : passagers) {

std::cout << std::left

<< std::setw(4)  << index++           << " | "

<< std::setw(6)  << p.getTitre()      << " | "

<< std::setw(14) << p.getNom()        << " | "

<< std::setw(14) << p.getPrenom()     << " | "

<< std::setw(4)  << p.getAge()        << " | "

<< p.getPasseport() << "\n";

}

std::cout << "═══════════════════════════════════════════════════════\n";

}

// === Modifications ===

void Vol::modifierDate(const Date& nouvelleDate) {

dateDepart = nouvelleDate;

}

void Vol::modifierHeure(int heure, int minute) {

dateDepart.setHeure(heure);

dateDepart.setMinute(minute);

}

// === Accesseurs ===

std::string Vol::getNumeroVol() const {

return numeroVol;

}

int Vol::getNbPlacesDispo() const {

return nbPlacesMax - nbPlacesReservees;

}

// === Sérialisation (sauvegarde) ===

std::string Vol::toString() const {

return numeroVol + ";" +

std::to\_string(nbPlacesMax) + ";" +

std::to\_string(nbPlacesReservees) + ";" +

std::to\_string(prix);

}

### Passager.h

#ifndef PASSAGER\_H

#define PASSAGER\_H

#include <iostream>

#include <string>

/\*\*

\* @brief Représente un passager d’un vol.

\* \* Contient les informations personnelles et le numéro de passeport

\* servant à l’identification lors des réservations.

\*/

class Passager {

private:

std::string nom;

std::string prenom;

std::string titre;             // "M", "Mme", ou "Mlle"

std::string numeroPasseport;

int age;

public:

// === Constructeurs ===

/\*\*

\* @brief Construit un passager par défaut avec des valeurs initiales.

\*/

Passager();

/\*\*

\* @brief Construit un passager avec les informations complètes.

\* @param nom Nom du passager

\* @param prenom Prénom du passager

\* @param titre Civilité ("M", "Mme", "Mlle")

\* @param numeroPasseport Numéro de passeport

\* @param age Âge du passager

\*/

Passager(const std::string& nom,

const std::string& prenom,

const std::string& titre,

const std::string& numeroPasseport,

int age);

// === Saisie & Affichage ===

/\*\*

\* @brief Saisie interactive des données du passager (avec validation).

\*/

void saisir();

/\*\*

\* @brief Affiche les informations du passager dans un format lisible.

\*/

void afficher() const;

// === Accesseurs ===

/\*\*

\* @return Le numéro de passeport du passager.

\*/

std::string getPasseport() const;

/\*\*

\* @return Le nom complet formaté du passager.

\*/

std::string getNomComplet() const;

/\*\*

\* @return Le nom du passager.

\*/

std::string getNom() const;

/\*\*

\* @return Le prénom du passager.

\*/

std::string getPrenom() const;

/\*\*

\* @return Le titre (civilité) du passager.

\*/

std::string getTitre() const;

/\*\*

\* @return L'âge du passager.

\*/

int getAge() const;

// === Export fichier ===

/\*\*

\* @brief Sérialise les informations du passager dans une ligne de texte.

\* @return Chaîne de caractères au format CSV compatible avec les fichiers.

\*/

std::string toString() const;

};

#endif // PASSAGER\_H

### Passager.cpp

#include "Passager.h"

#include <iostream>

#include <limits>    // pour std::numeric\_limits

#include <algorithm> // pour std::transform

#include <iomanip>   // pour setw, left

// === Constructeurs ===

Passager::Passager()

: nom("Inconnu"), prenom("Inconnu"), titre("M"),

numeroPasseport("000000"), age(0) {}

Passager::Passager(const std::string& nom, const std::string& prenom, const std::string& titre,

const std::string& numeroPasseport, int age)

: nom(nom), prenom(prenom), titre(titre),

numeroPasseport(numeroPasseport), age(age) {}

// === Saisie utilisateur avec validation ===

void Passager::saisir() {

// Titre

std::cout << "Titre (M / Mme / Mlle) : ";

std::getline(std::cin >> std::ws, titre);

std::transform(titre.begin(), titre.end(), titre.begin(), ::tolower);

titre = std::toupper(titre);

while (titre!= "M" && titre!= "Mme" && titre!= "Mlle") {

std::cout << "Titre invalide. Choisissez parmi : M, Mme, Mlle\nRéessayez : ";

std::getline(std::cin >> std::ws, titre);

std::transform(titre.begin(), titre.end(), titre.begin(), ::tolower);

titre = std::toupper(titre);

}

// Nom

std::cout << "Nom : ";

std::getline(std::cin >> std::ws, nom);

while (nom.empty()) {

std::cout << "Nom requis. Réessayez : ";

std::getline(std::cin >> std::ws, nom);

}

// Prénom

std::cout << "Prénom : ";

std::getline(std::cin >> std::ws, prenom);

while (prenom.empty()) {

std::cout << "Prénom requis. Réessayez : ";

std::getline(std::cin >> std::ws, prenom);

}

// Âge

std::cout << "Âge : ";

std::cin >> age;

while (std::cin.fail() |

| age <= 0 |

| age > 120) {

std::cout << "Âge invalide. Réessayez : ";

std::cin.clear();

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

std::cin >> age;

}

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

// Passeport

std::cout << "Numéro de passeport : ";

std::getline(std::cin >> std::ws, numeroPasseport);

while (numeroPasseport.empty()) {

std::cout << "Champ requis. Réessayez : ";

std::getline(std::cin >> std::ws, numeroPasseport);

}

}

// === Affichage formaté (aligné) ===

void Passager::afficher() const {

std::cout << std::left

<< std::setw(6)  << titre

<< std::setw(15) << nom

<< std::setw(15) << prenom

<< std::setw(5)  << age

<< "Passeport : " << numeroPasseport << "\n";

}

// === Accesseurs utiles ===

std::string Passager::getPasseport() const {

return numeroPasseport;

}

std::string Passager::getNomComplet() const {

return titre + " " + nom + " " + prenom;

}

std::string Passager::getNom() const {

return nom;

}

std::string Passager::getPrenom() const {

return prenom;

}

std::string Passager::getTitre() const {

return titre;

}

int Passager::getAge() const {

return age;

}

// === Export fichier ===

std::string Passager::toString() const {

return titre + ";" + nom + ";" + prenom + ";" +

std::to\_string(age) + ";" + numeroPasseport;

}

### SystemeDeGestion.h

#ifndef SYSTEME\_DE\_GESTION\_H

#define SYSTEME\_DE\_GESTION\_H

#include <vector>

#include <string>

#include "Vol.h"

#include "Reservation.h"

/\*\*

\* @brief Système global de gestion de vols pour une compagnie aérienne.

\* \* Gère les opérations de création de vols, de réservations,

\* de confirmation, de recherche et de persistance dans un fichier texte.

\*/

class SystemeDeGestion {

private:

std::vector<Vol> vols;                  ///< Liste de tous les vols

std::vector<Reservation> reservations;  ///< Liste des réservations (confirmées ou non)

public:

// ==== Constructeur ====

/\*\*

\* @brief Initialise un gestionnaire vide.

\*/

SystemeDeGestion();

// ==== GESTION DES VOLS ====

/\*\*

\* @brief Saisie et ajout d’un nouveau vol.

\*/

void ajouterVol();

/\*\*

\* @brief Affiche tous les vols disponibles.

\*/

void afficherVols() const;

/\*\*

\* @brief Annule un vol existant.

\* @param numeroVol Numéro du vol à annuler

\*/

void annulerVol(const std::string& numeroVol);

/\*\*

\* @brief Modifie la date de départ d’un vol.

\* @param numeroVol Numéro du vol concerné

\*/

void modifierVol(const std::string& numeroVol);

// ==== GESTION DES RÉSERVATIONS ====

/\*\*

\* @brief Réserve un vol pour un nouveau passager.

\*/

void reserverVol();

/\*\*

\* @brief Confirme une réservation existante (non confirmée).

\*/

void confirmerReservation();

/\*\*

\* @brief Affiche les passagers d’un vol donné.

\* @param numeroVol Numéro du vol

\*/

void afficherPassagersVol(const std::string& numeroVol) const;

// ==== RECHERCHE ====

/\*\*

\* @brief Recherche un vol par son numéro.

\* @param numeroVol Numéro du vol

\* @return Index dans la liste ou -1 si non trouvé

\*/

int rechercherVol(const std::string& numeroVol) const;

/\*\*

\* @brief Recherche une réservation par passeport et numéro de vol.

\* @param passeport Passeport du passager

\* @param numeroVol Vol concerné

\* @return Index dans la liste ou -1 si non trouvé

\*/

int rechercherReservation(const std::string& passeport, const std::string& numeroVol) const;

// ==== SAUVEGARDE / CHARGEMENT ====

/\*\*

\* @brief Enregistre les vols et réservations dans un fichier texte (compact).

\* @param chemin Nom du fichier cible

\*/

void enregistrerFichier(const std::string& chemin) const;

/\*\*

\* @brief Charge les données depuis un fichier texte (compact).

\* @param chemin Nom du fichier source

\*/

void chargerFichier(const std::string& chemin);

// ==== EXPORT LISIBLE POUR RAPPORT FINAL ====

/\*\*

\* @brief Génère un export lisible des données (vols + réservations) pour les professeurs ou les rapports.

\* @param chemin Nom du fichier.txt généré

\*/

void exporterLisible(const std::string& chemin) const;

};

#endif // SYSTEME\_DE\_GESTION\_H

### SystemeDeGestion.cpp

#include "SystemeDeGestion.h"

#include <iostream>

#include <fstream>   // Pour std::ofstream et std::ifstream

#include <sstream>   // Pour std::istringstream

#include <iomanip>   // Pour std::setw, std::fixed, std::setprecision

#include <limits>    // Pour std::numeric\_limits

SystemeDeGestion::SystemeDeGestion() {}

// ==== AJOUT DE VOL ====

void SystemeDeGestion::ajouterVol() {

std::string numero;

int maxPlaces;

float prix;

Date depart, arrivee;

Destination dest;

std::cout << "\n╔══════════════════════════════════════╗\n";

std::cout <<   "║         AJOUT D’UN NOUVEAU VOL       ║\n";

std::cout <<   "╚══════════════════════════════════════╝\n";

std::cout << "Numéro du vol            : ";

std::getline(std::cin >> std::ws, numero); // std::ws pour ignorer les espaces blancs précédents

// Validation de la saisie numérique pour maxPlaces

std::cout << "Nombre de places max     : ";

while (!(std::cin >> maxPlaces) |

| maxPlaces <= 0) {

std::cout << "Saisie invalide. Entrez un nombre entier positif : ";

std::cin.clear(); // Efface les drapeaux d'erreur

std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n'); // Ignore le reste de la ligne

}

// Validation de la saisie numérique pour prix

std::cout << "Prix par place (Fcfa)    : ";

while (!(std::cin >> prix) |

| prix < 0) {

std::cout << "Saisie invalide. Entrez un nombre positif : ";

std::cin.clear();

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

}

std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n'); // Nettoyer le buffer après la saisie numérique

std::cout << "\n--- DATE DE DEPART ---\n";

depart.saisir();

std::cout << "\n---- DATE D'ARRIVEE ----\n";

arrivee.saisir();

std::cout << "\n---- DESTINATION ----\n";

dest.saisir();

if (depart.estDansLePasse()) {

std::cout << "Erreur : la date de départ est dans le passé. Vol non ajouté.\n";

return;

}

// Vérifier si le numéro de vol existe déjà

if (rechercherVol(numero)!= -1) {

std::cout << "Erreur : Un vol avec ce numéro existe déjà. Vol non ajouté.\n";

return;

}

Vol vol(numero, maxPlaces, prix, depart, arrivee, dest);

vols.push\_back(vol);

std::cout << "Vol ajouté avec succès ✅.\n";

}

// ==== AFFICHAGE DES VOLS ====

void SystemeDeGestion::afficherVols() const {

std::cout << "\n╔══════════════════════════════════════╗\n";

std::cout <<   "║           LISTE DES VOLS             ║\n";

std::cout <<   "╚══════════════════════════════════════╝\n";

if (vols.empty()) {

std::cout << "Aucun vol enregistré.\n";

} else {

for (const auto& v : vols)

v.afficher();

}

}

// ==== ANNULATION ====

void SystemeDeGestion::annulerVol(const std::string& numeroVol) {

int index = rechercherVol(numeroVol);

if (index!= -1) {

// Supprimer toutes les réservations associées à ce vol

reservations.erase(std::remove\_if(reservations.begin(), reservations.end(),

[&](const Reservation& r){ return r.getNumeroVol() == numeroVol; }),

reservations.end());

vols.erase(vols.begin() + index);

std::cout << "Vol annulé avec succès ✅. Les réservations associées ont également été annulées.\n";

} else {

std::cout << "Vol introuvable.\n";

}

}

// ==== MODIFICATION ====

void SystemeDeGestion::modifierVol(const std::string& numeroVol) {

std::cout << "\n╔══════════════════════════════════════╗\n";

std::cout <<   "║     MODIFICATION D’UN VOL EXISTANT   ║\n";

std::cout <<   "╚══════════════════════════════════════╝\n";

int index = rechercherVol(numeroVol);

if (index!= -1) {

std::cout << "Nouvelle date de départ! \n";

Date nouvelleDate;

nouvelleDate.saisir();

if (nouvelleDate.estDansLePasse()) {

std::cout << "Erreur : la nouvelle date de départ est dans le passé. Modification annulée.\n";

return;

}

vols[index].modifierDate(nouvelleDate);

std::cout << "Date du vol modifiée avec succès .\n";

} else {

std::cout << "Vol introuvable.\n";

}

}

// ==== RÉSERVATION ====

void SystemeDeGestion::reserverVol() {

std::string numero;

std::cout << "\n╔══════════════════════════════════════╗\n";

std::cout <<   "║        RÉSERVATION D’UN VOL          ║\n";

std::cout <<   "╚══════════════════════════════════════╝\n";

std::cout << "Numéro du vol à réserver : ";

std::getline(std::cin >> std::ws, numero);

int index = rechercherVol(numero);

if (index == -1) {

std::cout << "Vol introuvable.\n";

return;

}

// Vérifier si le vol est complet avant de saisir les infos passager

if (vols[index].estComplet()) {

std::cout << "Désolé, ce vol est complet. Impossible de réserver.\n";

return;

}

Passager p;

std::cout << "\n--- Informations du passager ---\n";

p.saisir();

// Vérifier si le passager a déjà une réservation pour ce vol

if (rechercherReservation(p.getPasseport(), numero)!= -1) {

std::cout << "Ce passager a déjà une réservation pour ce vol. Réservation non effectuée.\n";

return;

}

if (vols[index].ajouterPassager(p)) { // ajouterPassager incrémente nbPlacesReservees

Reservation r(p, numero);

reservations.push\_back(r);

std::cout << "Réservation enregistrée (non confirmée).\n";

} else {

// Ce cas est déjà géré par ajouterPassager (vol complet ou passager déjà enregistré)

std::cout << "Impossible d'effectuer la réservation pour une raison inconnue.\n";

}

}

// ==== CONFIRMATION ====

void SystemeDeGestion::confirmerReservation() {

std::string passeport, numero;

std::cout << "\n╔══════════════════════════════════════╗\n";

std::cout <<   "║   CONFIRMATION D’UNE RÉSERVATION     ║\n";

std::cout <<   "╚══════════════════════════════════════╝\n";

std::cout << "Numéro du vol : ";

std::getline(std::cin >> std::ws, numero);

std::cout << "Numéro de passeport : ";

std::getline(std::cin >> std::ws, passeport);

int pos = rechercherReservation(passeport, numero);

if (pos!= -1) {

if (reservations[pos].estConfirmeeReservation()) {

std::cout << "Cette réservation est déjà confirmée.\n";

} else {

reservations[pos].confirmer();

std::cout << "Réservation confirmée .\n";

}

} else {

std::cout << "Réservation introuvable pour ce vol et ce passeport.\n";

}

}

// ==== AFFICHAGE DES PASSAGERS ====

void SystemeDeGestion::afficherPassagersVol(const std::string& numeroVol) const {

int index = rechercherVol(numeroVol);

if (index!= -1) {

vols[index].afficherPassagers();

} else {

std::cout << "Vol introuvable.\n";

}

}

// ==== RECHERCHES ====

int SystemeDeGestion::rechercherVol(const std::string& numeroVol) const {

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

if (vols[i].getNumeroVol() == numeroVol)

return static\_cast<int>(i);

}

return -1;

}

int SystemeDeGestion::rechercherReservation(const std::string& passeport, const std::string& numeroVol) const {

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

if (reservations[i].getPasseport() == passeport && reservations[i].getNumeroVol() == numeroVol)

return static\_cast<int>(i);

}

return -1;

}

// ==== SAUVEGARDE / CHARGEMENT ====

void SystemeDeGestion::enregistrerFichier(const std::string& chemin) const {

std::ofstream fichier(chemin);

if (!fichier) {

std::cout << "Erreur d’ouverture du fichier pour écriture : " << chemin << "\n";

return;

}

for (const auto& v : vols)

fichier << "VOL:" << v.toString() << "\n";

for (const auto& r : reservations)

fichier << "RES:" << r.toString() << "\n";

fichier.close();

std::cout << "Données sauvegardées dans '" << chemin << "'.\n";

}

void SystemeDeGestion::chargerFichier(const std::string& chemin) {

std::ifstream fichier(chemin);

if (!fichier) {

// Si le fichier n'existe pas, ce n'est pas une erreur grave au démarrage

// std::cout << "Fichier de données '" << chemin << "' introuvable. Démarrage avec des données vides.\n";

return;

}

vols.clear(); // Vider les données actuelles avant de charger

reservations.clear();

std::string ligne;

while (std::getline(fichier, ligne)) {

if (ligne.rfind("VOL:", 0) == 0) { // Si la ligne commence par "VOL:"

ligne = ligne.substr(4); // Supprimer le préfixe "VOL:"

std::istringstream iss(ligne);

std::string numero;

int max, reserves;

float prix;

char delim; // Pour consommer les délimiteurs ';'

// Lire les attributs du vol

std::getline(iss, numero, ';');

iss >> max >> delim >> reserves >> delim >> prix;

// Note: Les objets Date et Destination ne sont pas sérialisés/désérialisés ici.

// Cela nécessiterait d'ajouter leurs toString() et des méthodes de parsing.

// Pour cet exemple, ils sont initialisés par défaut.

Vol v(numero, max, prix, Date(), Date(), Destination());

vols.push\_back(v);

} else if (ligne.rfind("RES:", 0) == 0) { // Si la ligne commence par "RES:"

ligne = ligne.substr(4); // Supprimer le préfixe "RES:"

std::istringstream iss(ligne);

std::string titre, nom, prenom, ageStr, passeport, volNum, confStr;

// Lire les attributs du passager et de la réservation

std::getline(iss, titre, ';');

std::getline(iss, nom, ';');

std::getline(iss, prenom, ';');

std::getline(iss, ageStr, ';');

std::getline(iss, passeport, ';');

std::getline(iss, volNum, ';');

std::getline(iss, confStr, ';');

int age = std::stoi(ageStr);

bool confirme = (confStr == "1");

Passager p(nom, prenom, titre, passeport, age);

Reservation r(p, volNum);

if (confirme) r.confirmer();

reservations.push\_back(r);

// Ajouter le passager au vol correspondant si le vol existe déjà en mémoire

int i = rechercherVol(volNum);

if (i!= -1) {

// Vérifier si le passager n'est pas déjà ajouté au vol pour éviter les doublons lors du chargement

if (!vols[i].contientPassager(p.getPasseport())) {

vols[i].ajouterPassager(p); // Cela incrémentera nbPlacesReservees

}

}

}

}

fichier.close();

std::cout << "Données chargées depuis '" << chemin << "'.\n";

}

// ==== EXPORT LISIBLE ====

void SystemeDeGestion::exporterLisible(const std::string& chemin) const {

std::ofstream fichier(chemin);

if (!fichier) {

std::cout << "Impossible de créer le fichier : " << chemin << "\n";

return;

}

fichier << "\n========= RAPPORT DES VOLS =========\n\n";

if (vols.empty()) {

fichier << "Aucun vol disponible.\n";

} else {

for (const auto& v : vols) {

fichier << "- Vol : " << v.getNumeroVol() << "\n";

fichier << "  Places restantes : " << v.getNbPlacesDispo() << "\n";

// Extraction du prix de la chaîne toString() est une solution temporaire si prix n'est pas directement accessible

// Idéalement, Vol::getPrix() serait mieux.

std::string volString = v.toString();

size\_t lastSemi = volString.find\_last\_of(';');

if (lastSemi!= std::string::npos) {

fichier << "  Prix : " << volString.substr(lastSemi + 1) << " €\n\n";

} else {

fichier << "  Prix : N/A\n\n";

}

}

}

fichier << "\n========= RÉSERVATIONS =========\n\n";

if (reservations.empty()) {

fichier << "Aucune réservation.\n";

} else {

for (const auto& r : reservations) {

fichier << "- Vol : " << r.getNumeroVol()

<< ", Passeport : " << r.getPasseport()

<< ", Statut : " << (r.estConfirmeeReservation()? "Confirmée" : "Non confirmée") << "\n";

}

}

fichier.close();

std::cout << "Rapport exporté dans '" << chemin << "'.\n";

}

### Date.h

#ifndef DATE\_H

#define DATE\_H

/\*\*

\* @brief Représente une date complète avec heure.

\*

\* Utilisée pour les horaires de départ et d’arrivée dans la gestion des vols.

\*/

class Date {

private:

int jour;

int mois;

int annee;

int heure;

int minute;

public:

// ==== Constructeurs ====

/\*\*

\* @brief Initialise une date par défaut (01/01/2025 00:00).

\*/

Date();

/\*\*

\* @brief Initialise une date personnalisée.

\* @param j Jour

\* @param m Mois

\* @param a Année

\* @param h Heure (défaut = 0)

\* @param min Minute (défaut = 0)

\*/

Date(int j, int m, int a, int h = 0, int min = 0);

// ==== Méthodes principales ====

/\*\*

\* @brief Saisie utilisateur de la date (format guidé JJ/MM/AAAA HH:MM).

\*        Effectue une validation stricte des entrées.

\*/

void saisir();

/\*\*

\* @brief Affiche la date au format JJ/MM/AAAA HH:MM avec zéro-padding.

\*/

void afficher() const;

/\*\*

\* @brief Indique si la date est antérieure à la date système (au moment de l’exécution).

\* @return true si la date est dans le passé, false sinon.

\*/

bool estDansLePasse() const;

/\*\*

\* @brief Compare deux dates (utile pour tri, vérification, etc.).

\* @param autre L’autre date à comparer.

\* @return true si la date courante est antérieure à l’autre.

\*/

bool operator<(const Date& autre) const;

// ==== Setters ====

/\*\*

\* @brief Modifie l’heure (si valide).

\* @param h La nouvelle heure (0-23).

\*/

void setHeure(int h);

/\*\*

\* @brief Modifie les minutes (si valide).

\* @param m Les nouvelles minutes (0-59).

\*/

void setMinute(int m);

};

#endif // DATE\_H

### Date.cpp

#include "Date.h"

#include <iostream>

#include <ctime>     // Pour std::time, std::localtime, std::tm

#include <limits>    // Pour std::numeric\_limits

#include <iomanip>   // pour setw et setfill

// === Constructeurs ===

Date::Date() : jour(1), mois(1), annee(2025), heure(0), minute(0) {}

Date::Date(int j, int m, int a, int h, int min)

: jour(j), mois(m), annee(a), heure(h), minute(min) {}

// === Validation de date ===

bool Date::estDansLePasse() const {

std::time\_t t = std::time(nullptr);      // Obtenir le temps actuel en secondes depuis l'époque

std::tm\* now = std::localtime(&t);      // Convertir en structure tm pour les composants date/heure

// Comparaison année

if (annee < now->tm\_year + 1900) return true;

if (annee > now->tm\_year + 1900) return false;

// Si même année, comparer mois

if (mois < now->tm\_mon + 1) return true;

if (mois > now->tm\_mon + 1) return false;

// Si même année et mois, comparer jour

if (jour < now->tm\_mday) return true;

if (jour > now->tm\_mday) return false;

// Si même année, mois et jour, comparer heure

if (heure < now->tm\_hour) return true;

if (heure > now->tm\_hour) return false;

// Si même année, mois, jour et heure, comparer minute

if (minute < now->tm\_min) return true;

// Si minute est égale ou supérieure, ce n'est pas dans le passé

return false;

}

// === Saisie utilisateur ===

void Date::saisir() {

char slash1, slash2; // Pour consommer les '/'

std::cout << "Entrez la date (JJ/MM/AAAA) : ";

while (true) {

std::cin >> jour >> slash1 >> mois >> slash2 >> annee;

// Vérification des erreurs de saisie et du format

if (!std::cin.fail() && slash1 == '/' && slash2 == '/' &&

jour >= 1 && jour <= 31 && // Simplifié, une validation plus robuste des jours par mois serait nécessaire

mois >= 1 && mois <= 12 &&

annee >= 2024) { // Année minimale pour les vols futurs

break; // Sortir de la boucle si la saisie est valide

}

std::cout << "Format ou valeur de date invalide. Exemple : 27/05/2025\nNouvelle tentative : ";

std::cin.clear(); // Efface les drapeaux d'erreur

std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n'); // Ignore le reste de la ligne

}

std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n'); // Nettoyer le buffer après la saisie de la date

char colon; // Pour consommer le ':'

std::cout << "Entrez l’heure (HH:MM) : ";

while (true) {

std::cin >> heure >> colon >> minute;

// Vérification des erreurs de saisie et du format

if (!std::cin.fail() && colon == ':' &&

heure >= 0 && heure <= 23 &&

minute >= 0 && minute <= 59) {

break; // Sortir de la boucle si la saisie est valide

}

std::cout << "Format ou valeur d'heure invalide. Exemple : 14:30\nNouvelle tentative : ";

std::cin.clear();

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

}

std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n'); // Nettoyer le buffer après la saisie de l'heure

}

// === Affichage formaté ===

void Date::afficher() const {

std::cout << std::setfill('0') << std::setw(2) << jour << "/"

<< std::setw(2) << mois << "/"

<< annee << " "

<< std::setw(2) << heure << ":"

<< std::setw(2) << minute;

}

// === Comparaison (tri ou test) ===

bool Date::operator<(const Date& autre) const {

if (annee!= autre.annee) return annee < autre.annee;

if (mois!= autre.mois) return mois < autre.mois;

if (jour!= autre.jour) return jour < autre.jour;

if (heure!= autre.heure) return heure < autre.heure;

return minute < autre.minute;

}

// === Setters heure/minute ===

void Date::setHeure(int h) {

if (h >= 0 && h <= 23)

heure = h;

// else: gérer l'erreur ou ignorer

}

void Date::setMinute(int m) {

if (m >= 0 && m <= 59)

minute = m;

// else: gérer l'erreur ou ignorer

}

### Destination.h

#ifndef DESTINATION\_H

#define DESTINATION\_H

#include <string>

#include <iostream> // Pour std::cout dans afficher()

/\*\*

\* @brief Représente une destination de vol avec une ville de départ et une ville d'arrivée.

\*/

class Destination {

private:

std::string villeDepart;

std::string villeArrivee;

public:

// ==== Constructeurs ====

/\*\*

\* @brief Initialise une destination par défaut.

\*/

Destination();

/\*\*

\* @brief Initialise une destination avec des villes spécifiques.

\* @param depart Nom de la ville de départ.

\* @param arrivee Nom de la ville d'arrivée.

\*/

Destination(const std::string& depart, const std::string& arrivee);

// ==== Saisie & Affichage ====

/\*\*

\* @brief Saisie interactive des villes de départ et d'arrivée.

\*        Effectue une validation pour s'assurer que les champs ne sont pas vides.

\*/

void saisir();

/\*\*

\* @brief Affiche la destination au format "Ville Départ -> Ville Arrivée".

\*/

void afficher() const;

// ==== Méthodes utilitaires / Accesseurs ====

/\*\*

\* @brief Sérialise la destination dans une chaîne de caractères.

\* @return Chaîne au format "VilleDepart;VilleArrivee".

\*/

std::string toString() const;

/\*\*

\* @return Le nom de la ville de départ.

\*/

std::string getDepart() const;

/\*\*

\* @return Le nom de la ville d'arrivée.

\*/

std::string getArrivee() const;

};

#endif // DESTINATION\_H

### Destination.cpp

#include "Destination.h"

#include <iostream>

#include <limits> // Pour std::numeric\_limits

// === Constructeurs ===

Destination::Destination()

: villeDepart("Inconnue"), villeArrivee("Inconnue") {}

Destination::Destination(const std::string& depart, const std::string& arrivee)

: villeDepart(depart), villeArrivee(arrivee) {}

// === Saisie utilisateur ===

void Destination::saisir() {

// std::ws est utilisé avec getline pour ignorer les caractères blancs restants dans le buffer d'entrée

// après une saisie numérique précédente (par exemple, après un std::cin >> int).

// Cela évite que getline ne lise une ligne vide immédiatement.

std::cout << "Ville de départ   : ";

std::getline(std::cin >> std::ws, villeDepart);

while (villeDepart.empty()) { // Validation pour s'assurer que le champ n'est pas vide

std::cout << "Champ requis. Réessayez : ";

std::getline(std::cin >> std::ws, villeDepart);

}

std::cout << "Ville d'arrivée   : ";

std::getline(std::cin >> std::ws, villeArrivee);

while (villeArrivee.empty()) { // Validation pour s'assurer que le champ n'est pas vide

std::cout << "Champ requis. Réessayez : ";

std::getline(std::cin >> std::ws, villeArrivee);

}

}

// === Affichage de la destination ===

void Destination::afficher() const {

std::cout << villeDepart << " -> " << villeArrivee;

}

// === Méthodes utilitaires ===

std::string Destination::toString() const {

return villeDepart + ";" + villeArrivee;

}

std::string Destination::getDepart() const {

return villeDepart;

}

std::string Destination::getArrivee() const {

return villeArrivee;

}

### Reservation.h

#ifndef RESERVATION\_H

#define RESERVATION\_H

#include <iostream>

#include <string>

#include "Passager.h" // La réservation contient un objet Passager

/\*\*

\* @brief Représente une réservation pour un passager sur un vol.

\* \* Contient les informations du passager, le numéro du vol concerné,

\* et l’état de confirmation de la réservation.

\*/

class Reservation {

private:

Passager passager;         ///< Informations du passager

std::string numeroVol;     ///< Numéro du vol concerné

bool estConfirmee;         ///< Statut de confirmation (true si confirmée, false sinon)

public:

// ==== Constructeurs ====

/\*\*

\* @brief Crée une réservation vide avec des valeurs par défaut.

\*/

Reservation();

/\*\*

\* @brief Initialise une réservation avec un passager et un numéro de vol.

\*        La réservation est initialement non confirmée.

\* @param p L'objet Passager associé à la réservation.

\* @param vol Le numéro du vol pour lequel la réservation est faite.

\*/

Reservation(const Passager& p, const std::string& vol);

// ==== Opérations ====

/\*\*

\* @brief Marque la réservation comme confirmée.

\*/

void confirmer();

/\*\*

\* @brief Affiche les informations détaillées de la réservation, y compris le statut.

\*/

void afficher() const;

// ==== Accesseurs ====

/\*\*

\* @brief Vérifie si la réservation est confirmée.

\* @return true si confirmée, false sinon.

\*/

bool estConfirmeeReservation() const;

/\*\*

\* @brief Retourne le numéro de passeport du passager associé à cette réservation.

\* @return Le numéro de passeport.

\*/

std::string getPasseport() const;

/\*\*

\* @brief Retourne le numéro du vol concerné par cette réservation.

\* @return Le numéro du vol.

\*/

std::string getNumeroVol() const;

// ==== Sérialisation ====

/\*\*

\* @brief Sérialise les informations de la réservation dans une ligne de texte pour la sauvegarde.

\* @return Chaîne de caractères au format CSV compatible avec les fichiers.

\*/

std::string toString() const;

};

#endif // RESERVATION\_H

### Reservation.cpp

#include "Reservation.h"

#include <iostream>

// === Constructeurs ===

Reservation::Reservation()

: passager(), numeroVol("UNKNOWN"), estConfirmee(false) {}

Reservation::Reservation(const Passager& p, const std::string& vol)

: passager(p), numeroVol(vol), estConfirmee(false) {}

// === Opérations ===

void Reservation::confirmer() {

estConfirmee = true;

}

void Reservation::afficher() const {

std::cout << "\n--------------------------------------------------\n";

std::cout << "RÉSERVATION POUR LE VOL : " << numeroVol << "\n";

passager.afficher(); // Affiche les détails du passager

std::cout << "STATUT : " << (estConfirmee? "CONFIRMÉE " : "NON CONFIRMÉE") << "\n";

std::cout << "--------------------------------------------------\n";

}

// === Accesseurs ===

bool Reservation::estConfirmeeReservation() const {

return estConfirmee;

}

std::string Reservation::getPasseport() const {

return passager.getPasseport();

}

std::string Reservation::getNumeroVol() const {

return numeroVol;

}

// === Sérialisation ===

std::string Reservation::toString() const {

// Sérialise les informations du passager, puis le numéro de vol et le statut de confirmation

return passager.toString() + ";" + numeroVol + ";" + (estConfirmee? "1" : "0");

}