Airline Management System

Cet ´enonc´e comporte [4](#_bookmark1) pages.

# Objectif

L’objectif de ce devoir est de mettre en oeuvre les bonnes pratiques et patrons de conceptions en program- mation orient´ee-objet. Il s’agit de r´ealiser une application de gestion de syst`eme de gestion a´eroportuaire, dont le comportement global est r´eparti dans un ensemble de classes, pr´esent´ees dans la figure [1.](#_bookmark0)

L’application doit permettre `a un client de cr´eer des a´eroports, des compagnies a´eriennes ainsi que des vols. Le point d’entr´ee de l’application est le SystemManager. Chaque compagnie Airline est associ´e `a un ensemble de vols (Flights). Un vol a un a´eroport de d´epart (origin) et un a´eroport d’arriv´ee (destination). L’origine et la destination ne peuvent ˆetre les mˆemes.

Chaque vol comprend des classes tarifaires (e.g., premi`ere classe, business) appel´ees flight section.

Chaque classe tarifaire comprend des si`eges organis´es en rangs (ligne et colonne).

* 1. Consigne

Vous devez faire fonctionner votre application sur le jeu minimal d’instructions fourni dans le programme principal `a la fin de ce document. Vous devez ´egalement utiliser un framework de test (JUnit, TestNG), pour fournir des jeux de test pour les fonctions critiques de l’application que vous identifierez et dont vous justifierez le caract`ere critique (pas de tests sur les getters/setters !).

# SystemManager

C’est le point d’entr´ee de l’application. Les clients interagissent avec l’application en appelant les op´era- tions offertes par SystemManager. Ce dernier est reli´e aux a´eroports et compagnies a´eriennes dans l’appli- cation. A sa cr´eation, le SystemManager ne poss`ede aucun a´eroport ou compagnie a´erienne. Pour les cr´eer les op´erations createAirport() et createAirline() doivent ˆetre invoqu´ees.

Système manager est l’unique point d’entrée de l’application, il est donc nécessaire de n’avoir qu’une seule instance de Système manager. Utiliser le design pattern approprié et justifier votre choix.

Le SystemManager contient ´egalement les op´erations pour cr´eer les classes tarifaires, trouver les vols dis- ponibles entre deux a´eroports, et r´eserver des si`eges sur un vol. Pour afficher toute l’information concernant les a´eroports les compagnies et les vols, classes tarifaires et si`eges, on invoque l’op´eration displaySystem- Details().

— createAirport(String n) : cr´ee un objet de type Airport et le lie au SystemManager. L’a´eroport doit avoir un code n, dont la longueur est exactement ´egale `a 3. Deux a´eroports diff´erents ne peuvent avoir le mˆeme code.

Utilisez un test unitaire pour vérifier qu’il n’y a aucun doublon sur le code. Si c’est le cas, renvoyez un message d’erreur. Sinon, créer l’aéropot. Tous les aéroports seront stockés dans une collection de votre choix.

**Airline**

- name: String

+createFlight: Flight

+ﬁndFlight: Flight

+createSection()

+getAvailableFlights(): ArrayList<Flight>

+bookFlight()

**Flight**

* ﬂightDate: Calendar
* ﬂightID: String

+getAirline(): Airline

+getID(): String

+getOrig(): Airport

+getDest(): Airport

+getDate(): Calendar

+hasSection():Boolean

+hasSeats(): Boolean

+createSection(): Boolean

+ﬁndSection(): FlightSection

+bookSeat()

**FlightSection**

- section: SeatClass

+getSeatClass: SeatClass

+hasAvailableSeats: Boolean

+bookSeat(): Boolean

name doit avoir au

plus 5 caractères

\*

airlines

1

airline 1

ﬂights

\*

\*

1

destination

1

\*

origin

1

ﬂightID doit avoir au

plus 5 caractères

sections \*

1

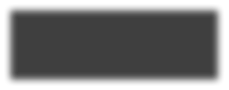
1..\*

seats

code doit avoir exactement 3 caractères

|  |  |
| --- | --- |
| **SystemManager** | |
|  | |
| +createAirport()  +createAirline()  +createFlight()  +createSection()  +ﬁndAvailableFlights()  +bookSeat()  +displaySystemDetails() | |
| airports | 1  \* |
| **Airport** | |
| - code: String | |
| +getName(): String | |

|  |
| --- |
| **Seat** |
| * seatNum: SeatID * isBooked: Boolean |
| +getSeatNum(): SeatID  +getStatus(): Boolean |



**«enumeration»**

**SeatClass**

+ FIRST ("First")

+ BUSI ("Business")

+ ECO ("Economy")

- name: String

|  |
| --- |
| **SeatID** |
| * row: Integer * column: Char |
|  |

## Figure 1 – Description architecturale de Airline Management System

* createAirline(String n) : cr´ee une compagnie a´erienne et la lie au SystemManager. Le nom n d’une compagnie doit ˆetre de longueur au plus ´egale `a 5. Deux compagnies diff´erentes ne peuvent avoir le mˆeme nom.

Utilisez un test unitaire pour vérifier qu’il n’y a aucun doublon sur le nom. Si c’est le cas, renvoyez un message d’erreur. Sinon, créer la compagnie aérienne. Toutes les compagnies seront stockées dans une collection de votre choix.

* createFlight(String n, String orig, String dest, int year, int month, int day, String id) : cr´ee un vol pour une compagnie n, en invoquant l’op´eration createFlight sur la classe Airline.

Tous les vols seront stockés dans une collection de votre choix implémentée dans la classe Airline.

* createSection(String air, String flID, int rows, int cols, SeatClass s) : cr´ee une sec- tion tarifaire, de classe s, pour un vol identifi´e par flID, associ´e `a une compagnie air, en invoquant l’op´eration createSection() de la classe Airline. La section contiendra le nombre de lignes et de colonnes.
* findAvailableFlights(String orig, Strin dest) : trouve tous les vols pour lesquels il existe encore des si`eges disponibles, entre les a´eroports de d´epart et d’arriv´ee.
* bookSeat(String air, String fl, SeatClass s, int row, char col) : r´eserve le si`ege dont la

position est indiqu´ee par row et col (e.g. 15A), sur le vol fl de la compagnie air.

— displaySystemDetails() : affiche toute l’information concernant tous les objets (e.g. a´eroports, com- pagnies, vols, si`eges, ...) dans le syst`eme.

Vous utiliserez la méthode toString().

# Airport

Un objet de cette classe repr´esente un a´eroport. Il poss`ede un nom, de longueur 3 caract`eres.

# Flight

Un objet de cette classe représente un vol. Il possède une date, un ID, un aéroport d’origine et de destination. Il appartient à une et une seule compagnie aérienne.

* +hasSection():Boolean : renvoie true si au moins une section tarifaire a été défini pour ce vol
* +hasSeats(): Boolean : renvoie true si il reste des sieges disponibles
* +createSection(int rows, int cols, SeatClass s): Boolean : créer une section tarifaire de classes, pour le vol. la section contiendra le nombre de lignes et de colonnes
* +ﬁndSection(): FlightSection ????

# Airline

Cette classe d´efinit une compagnie a´erienne. Une compagnie poss`ede z´ero ou plusieurs vols en cours. A la cr´eation d’un objet de ce type, il n’y a initialement aucun vol. Chaque vol doit avoir un identifiant unique.

* Flight createFlight(String orig, String dest, Calendar date, String id) : cr´ee un vol pour une compagnie a´erienne.
* Flight findFlight(String n) : trouve un vol dont l’identifiant est n.
* createSection(String flID, int rows, int cols, SeatClass s) : cr´ee une section tarifaire de classe s, pour un vol dont l’identifiant est flID. La section contiendra le nombre de lignes et de colonnes.
* ArrayList<Flight> getAvailableFlights(Airport orig, Airport dest) : trouve tous les vols sur lesquels il existe encore des si`eges disponibles, entre les a´eroports de d´epart et d’arriv´ee.
* bookFlight(String fl, SeatClass s, int row, char col) : r´eserve un si`ege dont la position est indiqu´ee par row et col (e.g. 15A) dans la section tarifaire s, sur le vol fl.

# FlightSection

Cette classe d´efinit une classe (ou section) tarifaire. Chaque section poss`ede une classe (premi`ere, affaires, ou ´economique) et au moins 1 si`ege. Une FlightSection poss`ede des attributs nombre de rows et nombre de columns, afin de savoir combien de si`eges elle contient et le calcul du nombre de si`eges disponibles.

Une section tarifaire contient au plus 100 rang´ees de si`eges et au plus 10 si`eges par rang´ee.

* hasAvailableSeats() renvoie vrai si et seulement si la section poss`ede encore des si`eges disponibles (non r´eserv´es).
* bookSeat() r´eserver le premier si`ege disponible. Son utilisation est conditionn´ee `a celle de hasAvai- lableSeats().
* boolean bookSeat(SeatID sId) r´eserver le si`ege `a l’emplacement d´esign´e par le param`etre sID, si ce si`ege est disponible.

Utilisez une collection pour stocker les sièges.

# Seat

Cette classe d´efinit un si`ege. Un si`ege poss`ede un identificateur, qui indique sa rang´ee et sa colonne (caract`ere allant de A `a J). Il poss`ede ´egalement un statut qui indique s’il est r´eserv´e ou pas.

# Client de l’application

Un exemple de client de cette application est fourni dans la classe ClientAMS. Ce client appelle des op´erations de la classe SystemManager.

Vous ˆetes invit´es `a ´etendre cette classe client avec d’autres invocations pour tester le comportement attendu de votre application.

1. **public class** Client AMS {
2. **public s t a t i c void** main ( String [] args ) {
3. System Manager res = **new** System Manager ();
4. *// A i r p o r t s*
5. res . create Airport ( **"DEN"** );
6. res . create Airport ( **"DFW"** );
7. res . create Airport ( **"LON"** );
8. res . create Airport ( **"DEN"** );
9. res . create Airport ( **"CDG"** );
10. res . create Airport ( **"JPN"** );
11. res . create Airport ( **"DEN"** ); *// Pb d ’ u n i c i t e*
12. res . create Airport ( **"DE"** ); *// I n v a l i d e*
13. res . create Airport ( **"DEH"** );
14. res . create Airport ( **" DRIrdn3 "** ); *// I n v a l i d e*

15

1. *// A i r l i n e s*
2. res . create Airline ( **"DELTA"** );
3. res . create Airline ( **" AIRFR"** );
4. res . create Airline ( **"AMER"** );
5. res . create Airline ( **" JET"** );
6. res . create Airline ( **"DELTA"** );
7. res . create Airline ( **"SWEST"** );
8. res . create Airline ( **"FRONTIER"** ); *// I n v a l i d e*

24

25 *// F l i g h t s*

26 res . create Flight ( **"DELTA"** , **"DEN"** , **"LON"** , 2008 , 11 , 12 , **" 123 "** );

27 res . create Flight ( **"DELTA"** , **"DEN"** , **"DEH"** , 2009 , 8 , 9 , **" 567 "** );

28 res . create Flight ( **"DELTA"** , **"DEN"** , **"NCE"** , 2010 , 9 , 8 , **" 567 "** ); *//*

*I n v a l i d e*

29

1. *// S e c t i o n s*
2. res . create Section ( **" JET"** , **" 123 "** , 2 , 2 , Seat Class . economy );
3. res . create Section ( **" JET"** , **" 123 "** , 1 , 3 , Seat Class . economy );
4. res . create Section ( **" JET"** , **" 123 "** , 2 , 3 , Seat Class . first );
5. res . create Section ( **"DELTA"** , **" 123 "** , 1 , 1 , Seat Class . business );
6. res . create Section ( **"DELTA"** , **" 123 "** , 1 , 2 , Seat Class . economy );
7. res . create Section ( **"SWSERTT"** , **" 123 "** , 5 , 5 , Seat Class . economy ); *// I n v a l i d e*

37

38 res . display System Details ();

39

40 res . find Available Flights ( **"DEN"** , **"LON"** );

41

1. res . book Seat ( **"DELTA"** , **" 123 "** , Seat Class . business , 1 , **’A ’** );
2. res . book Seat ( **"DELTA"** , **" 123 "** , Seat Class . economy , 1 , **’A ’** );
3. res . book Seat ( **"DELTA"** , **" 123 "** , Seat Class . economy , 1 , **’B ’** );
4. res . book Seat ( **"DELTA"** , **" 123 "** , Seat Class . business , 1 , **’A ’** ); *// Deja r e s e r v e*

46

47 res . display System Details ();

48

49 res . find Available Flights ( **"DEN"** , **"LON"** );

50 }

51 }