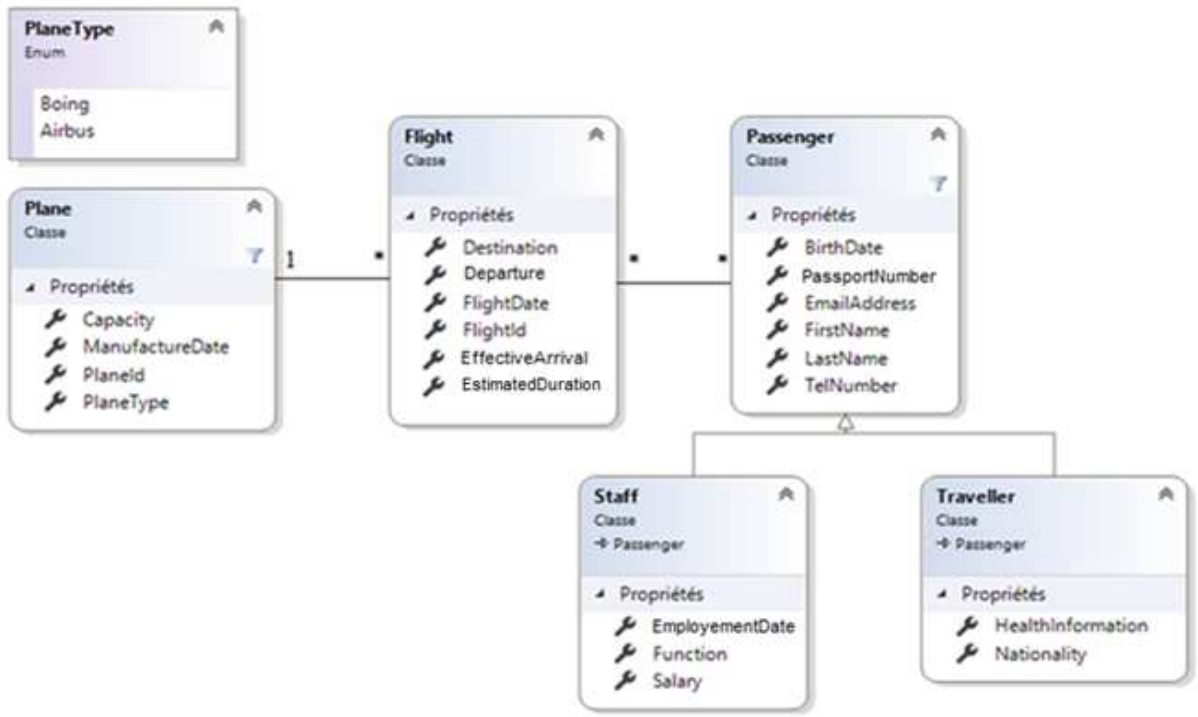
TP AirportManagement

Partie 1: Les principes de l'orienté objet

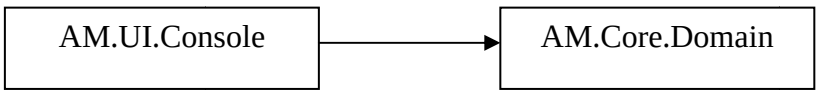
Pour tous les TPs, nous utiliserons l'implémentation .NET 6 du framework .Net.

On se propose de réaliser une application de gestion des activités d'un aéroport, définie par le diagramme de casses ci-dessous.



I Implémentation de la Couche de Domaine

1. Créer une Solution nommée « **AirportManagement** » ayant l'architecture logique suivante :



avec : **AM.UI.Console** : Projet de type application Console

AM.Core.Domain : Projet de type Bibliothèque de classes

2. Dans le projet **AM.Core.Domain**, créer les différentes classes du diagramme de classes ci-dessus.
3. Représenter l'héritage entre la classe **Passenger** et les deux classes **Staff** et **Traveller**.
4. Implémenter les propriétés qui représentent les différents attributs et leurs accesseurs.

5. Représenter les relations au biais des propriétés de navigation.

   Par exemple, la relation 1-* entre **Plane** et **Flight** sera représentée par les propriétés de navigation suivantes ;

   - Une propriété de type **IList<Flight>** dans la classe **Plane**
   - Une propriété de type **Plane** dans la classe **Flight**

6. Ré-implémenter la méthode ToString() pour toutes les classes. Ainsi cette méthode permettra de renvoyer une chaine de caractères représentant toutes les propriétés simples de la classe et leurs valeurs.

II Instanciation des objets

7. Dans le projet **Console**, créer un objet non initialisé de type **Plane** en utilisant le constructeur par défaut de la classe, puis initialiser ses attributs à travers leurs propriétés.
8. Dans la classe **Plane**, créer le constructeur paramétré suivant :

public Plane (PlaneType pt, int capacity, DateTime date)

Dans le projet **Console**, créer une autre instance en utilisant ce constructeur.

9. Instancier un autre avion en utilisant les initialiseurs d'objet.
10. Que remarquez-vous ?

III Le Polymorphisme

11. Surcharge des méthodes : Polymorphisme par Signature

Dans l'entité **Passenger**, créer les trois méthodes **bool CheckProfile(…)** suivantes :

   a. Une méthode pour vérifier le profile en utilisant deux paramètres: nom du passager et prénom du passager.
   b. Une méthode pour vérifier le profile en utilisant trois paramètres: nom du passager, prénom du passager et email du passager.
   c. Est-il possible de créer une méthode qui remplace les deux à la fois ?

12. Redéfinition des méthodes : Polymorphisme par héritage

   a. Créer la méthode **string GetPassengerType()** qui renvoie :

- « **I am a passenger»** : dans le cas où l'instance déclenchant la méthode est de type **Passenger**.
- « **I am a passenger I am a Staff Member»** : dans le cas où l'instance déclenchant la méthode est de type **Staff**
- « **I am a Traveller»** : dans le cas où l'instance déclenchant la méthode est de type **Traveller**.

b. Dans le projet **Console**, tester la méthode **GetPassengerType()** pour 3 instances de types **Passenger, Staff** et **Traveller**.

VI Passage par valeur / Passage par référence

13. Dans la classe **Passenger**, créer les deux méthodes suivantes qui permettent de calculer un âge à partir d'une date de naissance :
    a. **void GetAge(DateTime birthDate, int calculatedAge)**
    b. **void GetAge(Passenger aPassenger)** ; ajouter la propriété **Age** dans la classe **Passenger** qui contiendra l'âge calculée.
    c. Dans le projet **Console**, tester les deux méthodes et vérifier si les nouvelles valeurs sont conservées dans les paramètres d'entrée.
    d. Utiliser le mot clé **ref** pour forcer le passage par référence pour la première méthode, et tester.

V Encapsulation

14. Modifier la propriété **Age** de la classe **Passenger** pour qu'elle :

- soit en lecture seule et,
- permet de renvoyer l'âge d'un passager.