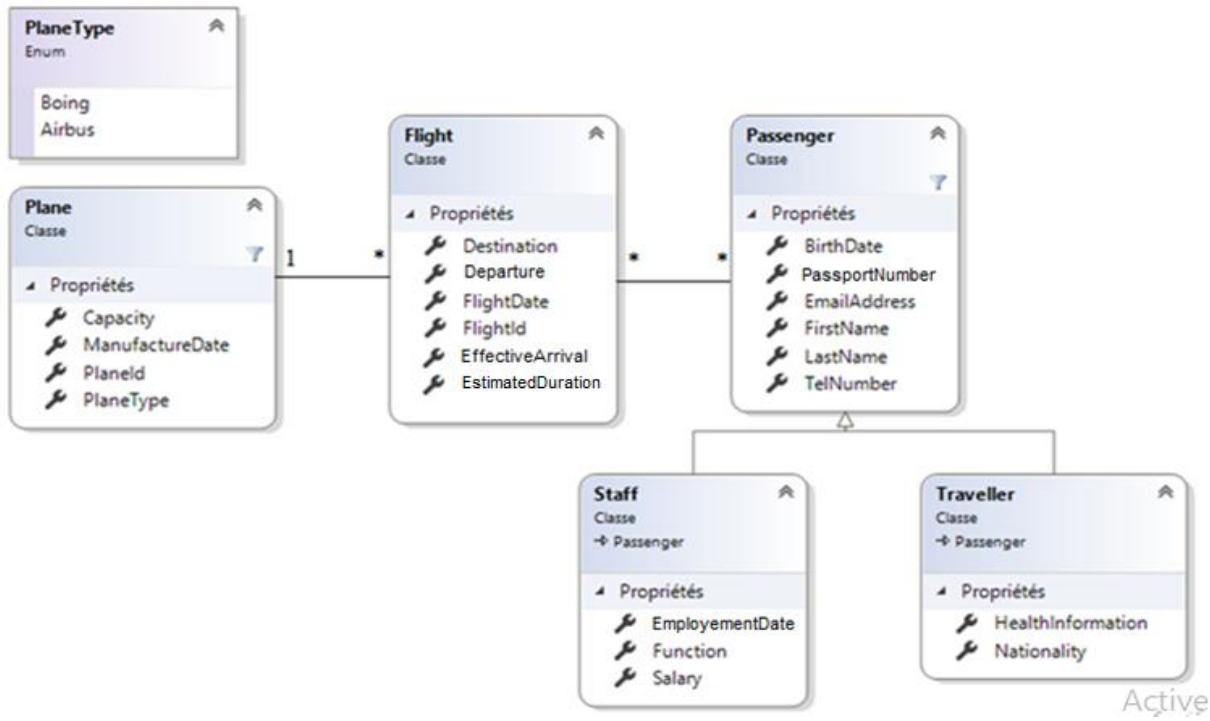
Correction TP AirportManagement

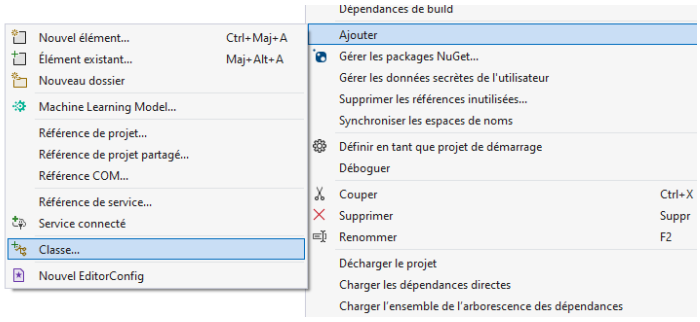
Partie 1: Les principes de l'orienté objet

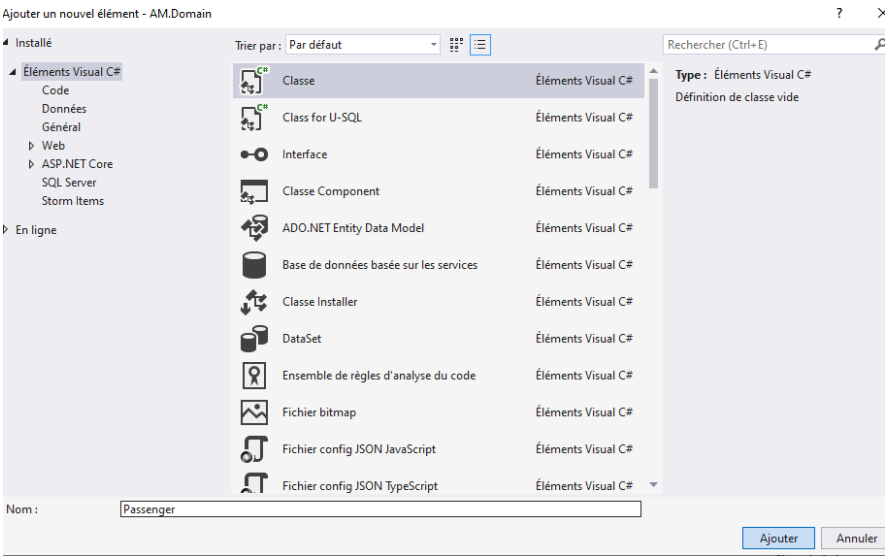
On se propose de réaliser une application de gestion des activités d'un aéroport, définie par le diagramme de classes ci-dessous.



I Implémentation de la Couche de Données

1. Créer une Solution nommée **AirportManagement** qui contient les deux projets suivants
   AM.UI.Console : Projet de type application Console (.NET 6.0)
   AM.ApplicationCore : Projet de type Bibliothèque de classe (.NET 6.0)
   Notez bien que le projet **Console** doit référencer le projet **ApplicationCore**.
2. Sous le projet **AM.ApplicationCore**, créer le dossier **Domain** et y implémenter 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 objets de navigation.
    a. Par exemple, la relation 1-* entre **Plane** et **Flight** sera représentée par les objets de navigation suivants ;
        i. – Une propriété de type **ICollection<Flight>** dans la classe **Plane**
        ii. – Une propriété de type **Plane** dans la classe **Flight**

```
public class Passenger
   {
      public string PassportNumber { get; set; }
      public string FirstName { get; set; }

      public string LastName { get; set; }

      public DateTime BirthDate { get; set; }
      public int TelNumber { get; set; }
      public string EmailAddress { get; set; }

      public virtual List<Flight> Flights { get; set; }
   }
```

```
public class Traveller : Passenger
   {
      public string HealthInformation { get; set; }
      public string Nationality { get; set; }
   }
```

```
public class Staff : Passenger
   {
      public string Function { get; set; }
      public DateTime EmployementDate { get; set; }
      public float Salary { get; set; }
   }
```

```
public class Flight
   {
      public int FlightId { get; set; }
      public DateTime FlightDate { get; set; }
      public int EstimatedDuration { get; set; }
      public DateTime EffectiveArrival { get; set; }
      public string Departure { get; set; }
```

```
        public string Destination { get; set; }
        public virtual List<Passenger> Passengers { get; set; }
        public virtual Plane Plane { get; set; }

    }
```

```
public enum PlaneType
    {
        Boing,
        Airbus
    }
    public class Plane
    {
        public int PlaneId { get; set; }

        public PlaneType PlaneType { get; set; }

        public DateTime ManufactureDate { get; set; }

        public int Capacity { get; set; }

        public virtual List<Flight> Flights { get; set; }
    }
```

6. Réimplémenter la méthode ToString() pour toutes les classes.

```
public override string ToString()
        {
            return "FirstName: "+ FirstName + " LastName: " + LastName + " date of
            Birth: "+ BirthDate;
        }
```

II Instanciation des objets

7. Créer un objet non initialisé de type **Plane** en utilisant le constructeur non paramétré de la classe, puis initialiser ses attributs à travers leurs propriétés.

```
Plane plane = new Plane();
plane.PlaneType = PlaneType.Airbus;
plane.Capacity = 200;
plane.ManufactureDate = new DateTime(2018, 11, 10);
```

8. Créer le constructeur suivant pour la classe **Plane**

```
    public Plane(PlaneType pt, int capacity, DateTime date)
        {
            PlaneType = pt;
            Capacity = capacity;
            ManufactureDate = date;

        }
```

Puis créer un autre avion en utilisant ce constructeur.

```
Plane plane2 = new Plane(PlaneType.Boing, 300, DateTime.Now);
```

9. Supprimer le constructeur crée précédemment et instancier un autre avion en utilisant les initialiseurs d'objet.

```
Plane plane3 = new Plane { PlaneType = PlaneType.Airbus, Capacity = 150, ManufactureDate = new
```

```
DateTime(2015, 02, 03) };
```

Que remarquez vous ?

C'est plus simple et intuitif d'utiliser les initialiseurs d'objets.

On n'est pas contraint de respecter la signature du constructeur.

On peut initialiser autant d'attribut qu'on décide.

III Le Polymorphisme

10. 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.

```
public bool CheckProfile(string firstName, string lastName)
    {
      return FirstName==firstName && LastName== lastName;
    }
```

b. Une méthode pour vérifier le profile en utilisant trois paramètres: nom du passager, prénom du passager et email du passager.

```
public bool CheckProfile(string firstName, string lastName, string email)
    {
      return FirstName==firstName && LastName==lastName && EmailAddress==email;
    }
```

c. Une méthode pour remplacer à la fois les deux méthodes précédentes.

```
public bool CheckProfile(string firstName, string lastName, string email=null)
   { if(email != null)
           return FirstName==firstName && LastName==lastName &&
EmailAddress==email;
       else
               return FirstName == firstName && LastName == lastName;}
```

d. Tester les méthodes à chaque fois

11. Polymorphysme par héritage

a- Dans la classe **Passenger**, implémenter la méthode **PassengerType** qui affiche « **I am a passenger»**

```
Public virtual void PassengerType()
    {
      Console.WriteLine("I am a Passenger");
    }
```

b- Dans la classe **Staff** et **Traveller**, réimplémenter la même méthode pour qu'elle affiche respectivement « **I am a passenger and I am a Staff Member**» et « **I am a passenger and I am a traveller»**.

Utiliser l'implémentation de la méthode de la classe mère dans les 2 méthodes précédentes.

```
public override void PassengerType()
    {
      base.PassengerType();
      Console.WriteLine("and I'm a staff member");
    }
```

```
public override void PassengerType()
    {
      base.PassengerType();
      Console.WriteLine("and I'm a traveller");
    }
```

c- Tester la méthode **PassengerType** dans le projet console pour 3 instances de types **Passenger, Staff et Traveller**.