Correction TP AirportManagement

Partie 2: Interrogation des données

1. Sous le projet **AM.ApplicationCore**, créer les deux dossiers **Interfaces** et **Services**.
2. Créer l'interface **IFlightMethods** dans le dossier **Interfaces** et la classe **FlightMethods** dans le dossier **Services**.
3. Dans la classe **FlightMethods**, créer la propriété **public List<Flight> Flights {get ; set ;}** et l'initialiser à une liste vide.

Nous formulerons les requêtes de ce TP en se basant sur cette liste comme source de données.

N'oubliez pas de mettre à chaque fois la signature de chaque méthode dans l'interface **IFlightMethods.**

I Préparation des données de test

4. Ajouter dans le dossier **Domain** la classe statique **TestData** qui contient les données statiques de test du tableau suivant.

Planes		
PlaneType	**Capacity**	**ManufactureDate**
Boing	150	03/02/2015
Airbus	250	11/11/2020

Staff					
FirstName	**LastName**	**EmailAddress**	**BirthDate**	**EmploymentDate**	**Salary**
captain	captain	Captain.captain@gmail.com	01/01/1965	01/01/1999	99999
hostess1	hostess1	hostess1.hostess1@gmail.com	01/01/1995	01/01/2020	999
hostess2	hostess2	hostess2.hostess2@gmail.com	01/01/1996	01/01/2020	999

Travellers					
FirstName	**LastName**	**EmailAddress**	**BirthDate**	**HealthInformation**	**Nationality**
Traveller1	Traveller1	Traveller1. Traveller1@gmail.com	01/01/1980	No troubles	American
Traveller2	Traveller2	Traveller2. Traveller2@gmail.com	01/01/1981	Some troubles	French

Traveller3	Traveller3	Traveller3. Traveller3@gmail.com	01/01/1982	No troubles	Tunisian
Traveller4	Traveller4	Traveller4. Traveller4@gmail.com	01/01/1983	Some troubles	American
Traveller5	Traveller5	Traveller5. Traveller5@gmail.com	01/01/1984	Some troubles	Spanish

Flights					
FlightDate	**Destination**	**EffectiveArrival**	**Plane**	**EstimatedDuration**	**Passengers**
01/01/2022 15:10:10	Paris	01/01/2022 17:10:10	Airbus	110	All created travellers
01/02/2022 21:10:10	Paris	01/02/2022 23:10:10	Boing	105	
01/03/2022 5:10:10	Paris	01/03/2022 6:40:10	Boing	100	
01/04/2022 6:10:10	Madrid	01/04/2022 8:10:10	Boing	130	
01/05/2022 17:10:10	Madrid	01/05/2022 18:50:10	Boing	105	
01/06/2022 20:10:10	Lisbonne	01/06/2022 22:30:10	Airbus	200	

Dans la même classe, créer la liste statique **List<Flight> listFlights** et l'initialiser avec tous les vols crées précédemment.

```
public static class TestData
    {
       public static Plane BoingPlane = new Plane { PlaneType = PlaneType.Boing,
Capacity = 150, ManufactureDate = new DateTime(2015, 02, 03) };
        public static Plane Airbusplane = new Plane { PlaneType = PlaneType.Airbus,
Capacity = 250, ManufactureDate = new DateTime(2020, 11, 11) };
        // Staffs
        public static Staff captain = new Staff { FirstName = "captain", LastName =
"captain", EmailAddress = "captain.captain@gmail.com", BirthDate = new DateTime(1965,
01, 01), EmployementDate = new DateTime(1999, 01, 01), Salary = 99999 };
        public static Staff hostess1 = new Staff { FirstName = "hostess1", LastName =
"hostess1", EmailAddress = "hostess1.hostess1@gmail.com", BirthDate = new
DateTime(1995, 01, 01), EmployementDate = new DateTime(2020, 01, 01), Salary = 999 };
        public static Staff hostess2 = new Staff { FirstName = "hostess2", LastName =
"hostess2", EmailAddress = "hostess2.hostess2@gmail.com", BirthDate = new
DateTime(1996, 01, 01), EmployementDate = new DateTime(2020, 01, 01), Salary = 999 };
        // Travellers
        public static Traveller traveller1 = new Traveller { FirstName = "traveller1",
LastName = "traveller1", EmailAddress = "traveller1.traveller1@gmail.com", BirthDate =
new DateTime(1980, 01, 01), HealthInformation = "no troubles", Nationality =
"American" };
        public static Traveller traveller2 = new Traveller { FirstName = "traveller2",
LastName = "traveller2", EmailAddress = "traveller2.traveller2@gmail.com", BirthDate =
```

```
new DateTime(1981, 01, 01), HealthInformation = "Some troubles", Nationality =
"French" };
        public static Traveller traveller3 = new Traveller { FirstName = "traveller3",
LastName = "traveller3", EmailAddress = "traveller3.traveller3@gmail.com", BirthDate =
new DateTime(1982, 01, 01), HealthInformation = "no troubles", Nationality =
"Tunisian" };
        public static Traveller traveller4 = new Traveller { FirstName = "traveller4",
LastName = "traveller4", EmailAddress = "traveller4.traveller4@gmail.com", BirthDate =
new DateTime(1983, 01, 01), HealthInformation = "Some troubles", Nationality =
"American" };
        public static Traveller traveller5 = new Traveller { FirstName = "traveller5",
LastName = "traveller5", EmailAddress = "traveller5.traveller5@gmail.com", BirthDate =
new DateTime(1984, 01, 01), HealthInformation = "Some troubles", Nationality =
"Spanish" };
          // Flights
     // Affect all passengers to flight1
     public static Flight flight1 = new Flight { FlightDate = new DateTime(2022, 01, 01,15,10,10), Destination =
"Paris", EffectiveArrival = new DateTime(2022, 01, 01,17,10,10),EstimatedDuration=110 ,  Passengers= new
List<Passenger> { captain, hostess1, hostess2, traveller1, traveller2, traveller3, traveller4, traveller5 }
    ,Plane= Airbusplane };
     public static Flight flight2 = new Flight { FlightDate = new DateTime(2022, 02, 01,21,10,10), Destination =
"Paris", EffectiveArrival = new DateTime(2022, 02, 01, 23, 10, 10), EstimatedDuration =105 , Plane = BoingPlane };
     public static Flight flight3 = new Flight { FlightDate = new DateTime(2022, 03, 01, 5, 10, 10), Destination =
"Paris", EffectiveArrival = new DateTime(2022, 03, 01, 6, 40, 10), EstimatedDuration = 100, Plane = BoingPlane };
     public static Flight flight4 = new Flight { FlightDate = new DateTime(2022, 04, 01, 6, 10, 10), Destination =
"Madrid", EffectiveArrival = new DateTime(2022, 04, 01, 8, 10, 10), EstimatedDuration =130 , Plane = BoingPlane
};
     public static Flight flight5 = new Flight { FlightDate = new DateTime(2022, 05, 01, 17, 10, 10), Destination =
"Madrid", EffectiveArrival = new DateTime(2022, 05, 01, 18, 50, 10), EstimatedDuration =105 , Plane = BoingPlane
};
     public static Flight flight6 = new Flight { FlightDate = new DateTime(2022, 06, 01, 20, 10, 10), Destination =
"Lisbonne", EffectiveArrival = new DateTime(2022, 06, 01, 22, 30, 10), EstimatedDuration =200 , Plane =
Airbusplane };
        //test list
        public static List<Flight> listFlights = new List<Flight> { flight1, flight2,
flight3, flight4, flight5, flight6 };
    }
```

5. Dans le projet console, créer une instance de la classe **FlightMethods** puis affecter **listFlights** à la propriété **Flights** de cette classe service.

```
FlightMethods fm = new FlightMethods();
fm.Flights = TestData.listFlights;
```

Tester toutes les méthodes qui suivent en se basant sur ces données de test.

II Les structures itératives

6. En utilisant la boucle For, implémenter la méthode **GetFlightDates (string destination)** qui retourne la liste des dates de vols d'une destination passée en paramètre.

```
public class FlightMethods: IFlightMethods
  {   public List<Flight> Flights { get; set; } = new List<Flight>();

     public List<DateTime> GetFlightDates(string destination)
     {
        List<DateTime> ls = new List<DateTime>();
```

```
        for (int j = 0; j < Flights.Count; j++)
            if (Flights[j].Destination.Equals(destination))
                ls.Add(Flights[j].FlightDate);
        return ls;
    }}
```

7. Reformuler la requête avec **foreach**.

```
List<DateTime> ls = new List<DateTime>();
foreach(Flight f in Flights)
    if (f.Destination.Equals(destination))
            ls.Add(f.FlightDate);
return ls;
```

8. Implémenter la méthode **GetFlights(string filterType, string filterValue)** qui affiche les vols en fonction de type de filtre et sa valeur. Le type de filtre représente un attribut simple (ce n'est pas une instance de classe ou une liste) de la classe **Flight**.

   Par exemple **GetFlights("Destination", "Paris")** permettra d'afficher les vols dont la valeur de **Destination** est Paris.

```
public void GetFlights(string filterType, string filterValue)
    {
      switch (filterType)
      {
        case "Destination":
          foreach (Flight f in Flights)
          {
            if (f.Destination.Equals(filterValue))
              Console.WriteLine(f);
          }
          break;
        case "FlightDate":
          foreach (Flight f in Flights)
          {
            if (f.FlightDate == DateTime.Parse(filterValue))

              Console.WriteLine(f);

          }
          break;
         case "EffectiveArrival":
          foreach (Flight f in Flights)
          {
            if (f.EffectiveArrival == DateTime.Parse(filterValue))
              Console.WriteLine(f);
          }
          break;          }
    }
```

III Le langage LINQ

Implémenter les méthodes suivantes et les tester à chaque fois dans le projet Console

9. Reformuler la méthode **GetFlightDates** en utilisant une requête LINQ.

```
IEnumerable<DateTime> req = from f in Flights
```

```
                    where f.Destination.Equals(destination)
                    select f.FlightDate;
return req.ToList();
```

10. Afficher les dates et les destinations des vols d'un avion passé en paramètre

    i. **ShowFlightDetails(Plane plane)**

```
public void ShowFlightDetails(Plane plane)
    {
      var req = from f in Flights
            where f.Plane == plane
            select new{f.FlightDate, f.Destination};
      foreach(var v in req)
      Console.WriteLine("Flight Date; "+v.FlightDate+" Flight destination: "+v.Destination);
    }
```

    ii.

11. Retourner le nombre de vols programmés pour une semaine (7jours) à partir d'une date donnée

    i. **ProgrammedFlightNumber**(DateTime startDate)

```
public int ProgrammedFlightNumber(DateTime startDate)
    {
      var req = from f in Flights
            where DateTime.Compare(f.FlightDate,startDate)>0 && (f.FlightDate - startDate).TotalDays < 7
            select f;
       return req.Count();

    }
```

12. Retourner la moyenne de durées estimées des vols d'une destination donnée

    i. **DurationAverage(string destination)**

```
public double DurationAverage(string destination)
    {
      return (from f in Flights
            where f.Destination.Equals(destination)
            select f.EstimatedDuration).Average();

     }
```

13. Retourner les Vols ordonnés par EstimatedDuration du plus long au plus court

    i. **OrderedDurationFlights()**

```
public IEnumerable<Flight> OrderedDurationFlights()
    {
      var req = from f in Flights
            orderby f.EstimatedDuration descending
            select f;

      return req;
    }
```

14. Retourner les nationalités des 3 passagers, de type traveller, les plus âgés d'un vol

    i. **SeniorTravellers(Flight flight)**

```
public IEnumerable<String> SeniorTravellers(Flight f)
      {

         var oldTravellers = from p in f.Passengers.OfType<Traveller>()
                        orderby p.BirthDate
                        select p.Nationality;
         return oldTravellers.Take(3);
         //if we want to skip 3
         //return oldTravellers.Skip(3);
      }
```

15. Retourner les vols groupés par destination et les afficher sous ce format :

Destination Paris
Décollage : 03/05/2022 12 : 10 :00
Décollage : 05/05/2022 23 : 00 :00
Décollage : 10/05/2022 21 : 15 :00
Destination Madrid
Décollage : 01/05/2022 10 : 10 :00
Décollage : 02/05/2022 13 : 10 :00

i. **DestinationGroupedFlights()**

```
public IEnumerable<IGrouping<string, Flight>> DestinationGroupedFlights()
    {
       var req = from f in Flights
             group f by f.Destination;
       foreach (var g in req)
       { Console.WriteLine("Destination: " + g.Key);
          foreach (var f in g)
          Console.WriteLine("Décollage: " +f.FlightDate);
       }
       return req;
    }
```

ii.

IV Expressions Lambda / Les méthodes LINQ prédéfinies

16. Créer dans la classe **FlightMethods** les deux délégués suivants :

FlightDetailsDel : pointe vers une méthode qui prend un objet **Plane** en paramètre et ne retourne rien.

DurationAverageDel : pointe vers une méthode qui prend une chaine de caractère en paramètre et qui retourne un réel

```
public Action<Plane> FlightDetailsDel;
public Func<string,double> DurationAverageDel;
```

17. Dans le constructeur de la classe « FlightMethods », affecter respectivement les deux méthodes **ShowFlightDetails** et **DurationAverage** aux deux délégués crées précédemment et tester le résultat.

```
public FlightMethods()
      {
         DurationAverageDel = DurationAverage;
         FlightDetailsDel = ShowFlightDetails;
      }
```

Tester

18. Affecter aux délégués le même traitement des méthodes **ShowFlightDetails** et **DurationAverage**, mais à travers des méthodes anonymes (expressions Lambda).

```
// DurationAverageDel = DurationAverage;
        DurationAverageDel = dest =>
        {
           return (from f in Flights
                   where f.Destination.Equals(dest)
                   select f.EstimatedDuration).Average();

        };
```

```
// FlightDetailsDel = ShowFlightDetails;
        FlightDetailsDel = p =>
        {
           var req = from f in Flights
                     where f.Plane == p
                     select new { f.FlightDate, f.Destination };
           foreach (var v in req)
              Console.WriteLine("Flight Date; " + v.FlightDate + " Flight destination: " + v.Destination);
        };
```

Retester.

19. Reformuler toutes les méthodes de la section II en se basant sur les méthodes de requêtes prédéfinies de la bibliothèque System.LINQ.

V Les méthodes d'extension

Dans le dossier **Services**, créer la classe **PassengerExtension** qui étend la classe **Passenger** et qui contient la méthode d'extension **UpperFullName**() qui met en majuscule la première lettre du nom et du prénom d'un passager.

```
public static class PassengerExtension
   {
      public static void UpperFullName(this Passenger p)
      {
         p.FirstName = p.FirstName[0].ToString().ToUpper() + p.FirstName.Substring(1);
         p.LastName = p.LastName[0].ToString().ToUpper() + p.LastName.Substring(1);
      }
   }
```