

SEQUENCE 2 – ASSOCIATION

SEANCES 3 ET 4 : CABINET MEDICAL

OBJECTIFS

- Manipuler des classes liées par la notion d'association
- Revoir les notions de base d'une classe

CONSIGNES

1. Au sein de votre répertoire « **Sequence_2_Association** » Créez un projet « **CabinetMedical** » dans une solution « **Seances_CabinetMedical** ».

L'agenda gère l'ensemble des consultations d'un cabinet médical. Une consultation concerne un patient. Elle est réalisée par un docteur. Vous allez définir les classes suivantes en suivant les consignes :

Agenda
Classe

▶ Champs
 ▲ Propriétés
 🔧 LesConsultations { get; set; } : List<Consultatio...
 ▶ Méthodes

Consultation
Classe

▶ Champs
 ▲ Propriétés
 🔧 ADomicile { get; set; } : bool
 🔧 EstDocteurTraitant { get; set; } : bool
 🔧 UnDocteur { get; set; } : Docteur
 🔧 UnJourUnHoraire { get; set; } : DateTime
 🔧 UnPatient { get; set; } : Patient
 ▶ Méthodes

Docteur
Classe

▶ Champs
 ▲ Propriétés
 🔧 UneCategorie { get; set; } : CategorieDocteur
 🔧 UnNom { get; set; } : string
 ▶ Méthodes

CategorieDocteur
Enum

Generaliste
 Specialiste
 Specialiste_Complexe
 Specialiste_Tres_Complexe

Patient
Classe

▶ Champs
 ▲ Propriétés
 🔧 Age { get; } : int
 🔧 DateNaissance { get; set; } : DateTime
 🔧 Nom { get; set; } : string
 🔧 Prenom { get; set; } : string
 ▶ Méthodes

2. Définissez la classe Docteur ainsi que l'énumération `CategorieDocteur`, en fait, il existe 4 catégories. Placez l'enum dans le même fichier que `Docteur`, juste au-dessus de la classe `Docteur`.

```
public enum CategorieDocteur { Generaliste = 0 , Specialiste = 1 ,  
Specialiste_Complexe = 2, Specialiste_Tres_Complexe =3 };
```

Docteur
Classe

▲ Champs

- TARIFS_CATEGORIES_DOCTEUR : double[]
- uneCategorie : CategorieDocteur
- unNom : string

▲ Propriétés

- UneCategorie { get; set; } : CategorieDocteur
- UnNom { get; set; } : string

▲ Méthodes

- Docteur()
- Docteur(CategorieDocteur uneCategorie, string unNom)
- DonneSonTarif() : double
- Equals(object obj) : bool
- GetHashCode() : int
- operator !=(Docteur left, Docteur right) : bool
- operator ==(Docteur left, Docteur right) : bool
- ToString() : string

CategorieDocteur
Enum

Generaliste
Specialiste
Specialiste_Complexe
Specialiste_Tres_Complexe

3. Stockez les tarifs dans `TARIFS_CATEGORIES_DOCTEUR` :

```
public static readonly double[] TARIFS_CATEGORIES_DOCTEUR = new double[] { 25,  
30, 46, 60 };
```

Catégories	Tarif
Generaliste	25
Specialiste	30
Specialiste_Complexe	46
Specialiste_Tres_Complexe	60




4. Définissez la méthode `DonneSonTarif`. Soyez malin, ne testez pas les 4 cas avec des if.
Aide : il est possible de caster en int un valeur de l'énumération. Ex :
(int)CategorieDocteur.Generaliste équivaut à 0 car 1^{ère} valeur dans la liste par défaut 0
5. Instanciez au moins 2 docteurs en utilisant l'énumération dans le main, le top serait 4, (pas de saisies utilisateurs) pour tester `DonneTarif` mais aussi `ToString`. Ex :

```
Docteur docGeneraliste = new Docteur(CategorieDocteur.Generaliste, "Oddou");
```





6. Définissez la classe Patient. Remarques :
- Age est une propriété sans « setter » uniquement un « getter » qui contiendra le code pour calculer et retourner l'âge.
 - Il y a une surcharge du constructeur afin de créer un patient sans avoir à créer un DateTime. Vous utiliserez la méthode Parse de la classe DateTime.

Patient
Classe









▲ Champs

 dateNaissance : DateTime
 nom : string
 prenom : string

▲ Propriétés

 Age { get; } : int
 DateNaissance { get; set; } : DateTime
 Nom { get; set; } : string
 Prenom { get; set; } : string

▲ Méthodes

 Equals(object obj) : bool
 GetHashCode() : int
 operator !=(Patient left, Patient right) : bool
 operator ==(Patient left, Patient right) : bool
 Patient()
 Patient(string nom, string prenom, DateTime dateNaissance)
 Patient(string nom, string prenom, string dateNaissance)
 ToString() : string

7. Instanciez au moins 2 patients (pas de saisies utilisateurs), un qui a 20 ans (anniversaire passé. Ex : 02/01/2004) , et un qui a 19 ans mais qui aura 20 ans avant la fin de l'année (ex : 25/11/2004) puis tester ToString mais aussi leurs âges.

8. Définissez la classe Consultation . Remarques :

a. Il y a 8 constantes :

- Pour stocker les suppléments : 5 euros le supplément très jeune enfant, 3 euros le supplément jeune enfant, 20 euros pour les dimanches et 10 euros pour un déplacement.
- Pour stocker les âges max : 2 pour très jeune enfant, 6 pour jeune enfant
- Pour stocker les taux de remboursement par la sécu : 0.70 le taux de remboursement pour une consultation chez un docteur traitant, 0.30 le taux de remboursement pour un autre docteur.

```
public const double AGE_MAX_TRES_JEUNE_ENFANT = 2;
public const double AGE_MAX_JEUNE_ENFANT = 5;
public const double SUPP_TRES_JEUNE_ENFANT = 5;
public const double SUPP_JEUNE_ENFANT = 3;
public const double SUPP_DIMANCHE = 20;
public const double SUPP_DEPLACEMENT = 10;
public const double TAUX_REMBOURSEMENT_DOC_TRAITANT = 0.7;
public const double TAUX_REMBOURSEMENT_DOC_NON_TRAITANT = 0.3;
public const string MONNAIE = "€";
```

b. La méthode CalculePrixConsultation : doit calculer le prix en fonction :

- du docteur, du patient
- du jour de consultation
- du fait que la consultation ait lieu à domicile ou non (ce qui engendre le supplément de déplacement ou non)

c. La méthode CalculeRemboursementSecu : doit calculer le remboursement : le taux de remboursement varie si la consultation est réalisée par le docteur traitant ou non.

Consultation

Classe

Champs

aDomicile : bool

AGE_MAX_JEUNE_ENFANT : double

AGE_MAX_TRES_JEUNE_ENFANT : double

estDocteurTraitant : bool

SUPP_DEPLACEMENT : double

SUPP_DIMANCHE : double

SUPP_JEUNE_ENFANT : double

SUPP_TRES_JEUNE_ENFANT : double

TAUX_REMBOURSEMENT_DOC_NON_TRAITANT : double

TAUX_REMBOURSEMENT_DOC_TRAITANT : double

unDocteur : Docteur

unJourUnHoraire : DateTime

unPatient : Patient

Propriétés

ADomicile { get; set; } : bool

EstDocteurTraitant { get; set; } : bool

UnDocteur { get; set; } : Docteur

UnJourUnHoraire { get; set; } : DateTime

UnPatient { get; set; } : Patient

Méthodes

CalculePrixConsultation() : double

CalculeRemboursementSecu() : double

Consultation()

Consultation(bool aDomicile, bool estDocteurTraitant, CategorieDocteur catDocteur, string nomDoc, string nomPatient, string prenomPatie...

Consultation(bool aDomicile, bool estDocteurTraitant, Docteur unDocteur, Patient unPatient, DateTime unJour)

Consultation(bool aDomicile, bool estDocteurTraitant, Docteur unDocteur, Patient unPatient, string unJour)

ToString() : string

9. Pour tester correctement CalculePrixConsultation : il faut un jeu de 48 tests. Explications :

Une consultation sans aucun supplément enfant et sans aucun autre supplément pour chaque catégorie de docteur : soit 4 tests

Une consultation sans aucun supplément enfant mais supplément dimanche pour chaque catégorie de docteur : soit 4 tests

Une consultation sans aucun supplément enfant mais supplément déplacement pour chaque catégorie de docteur : soit 4 tests

Une consultation sans aucun supplément enfant mais supplément dimanche et déplacement pour chaque catégorie de docteur : soit 4 tests

Soit 16 tests à renouveler avec le cas de figure enfant très jeune, et enfant jeune : Soit 48 tests en tout.

Nous verrons plus tard comment automatiser ces tests.

Vous vous contenterez d'instancier 12 consultations à partir du fichier « lesConsultations.json ». Voici les prix que vous devez obtenir :

```
// POUR UN GENERALISTE
// jeux de test pour un patient non enfant
1. un jour en semaine au cabinet => 25 €
2. un jour en semaine à domicile => 35 €
3. un dimanche au cabinet => 45 €
4. un dimanche et déplacement (à domicile) => 55 €

// jeux de test pour un jeune enfant
5. un jour en semaine au cabinet
6. un jour en semaine à domicile
7. un dimanche au cabinet
8. un dimanche et déplacement (à domicile)

// jeux de test pour un très jeune enfant
9. un jour en semaine au cabinet
10. un jour en semaine à domicile
11. un dimanche au cabinet
12. un dimanche et déplacement (à domicile)
```

```
Consultation 1 : 25€
Consultation 2 : 35€
Consultation 3 : 45€
Consultation 4 : 55€
Consultation 5 : 28€
Consultation 6 : 38€
Consultation 7 : 48€
Consultation 8 : 58€
Consultation 9 : 30€
Consultation 10 : 40€
Consultation 11 : 50€
Consultation 12 : 60€
```

```
private static List<T> ChargeJson<T>(String pathName)
{
    List<T> liste = null;
    try
    {
        String contenuFichier = File.ReadAllText(pathName);
        liste = JsonConvert.DeserializeObject<List<T>>(contenuFichier);
    }
    catch (Exception e) { throw; }
    return liste;
}

private static void SauvegardeJson<T>(List<T> liste, String pathName)
{
    try
    {
        string result = JsonConvert.SerializeObject(liste, Formatting.Indented);
        File.WriteAllText(pathName, result);
    }
    catch (Exception e) { throw; }
}
```

Console.OutputEncoding =
System.Text.Encoding.UTF8;

10. Testez aussi CalculeRemboursement : voici le résultat attendu pour les 4^{ème} consultations

```

Consultation 1 : 25€
Docteur traitant - taux de remboursement : 0,7
Remboursement 1 : 17,5€
-----
Consultation 2 : 35€
Docteur non traitant - taux de remboursement : 0,3
Remboursement 2 : 10,5€
-----
Consultation 3 : 45€
Docteur non traitant - taux de remboursement : 0,3
Remboursement 3 : 13,5€
-----
Consultation 4 : 55€
Docteur traitant - taux de remboursement : 0,7
Remboursement 4 : 38,5€
-----

```

11. Définissez la classe Agenda.

Agenda
 Classe

▲ Champs

- lesConsultations : List<Consultation>
- pathFichierData : string

▲ Propriétés

- LesConsultations { get; set; } : List<Consultatio...
- PathFichierData { get; set; } : string

▲ Méthodes

- ChargeJson<T>() : List<T>
- ConsultationsDuJour() : List<Consultation>
- ConsultationsDuJour(DateTime jour) : List<Co...
- NbConsultations() : int
- SauvegardeJson<T>(List<T> liste) : void
- TriParDate() : void
- TriParDocteur() : void
- TriParPrix() : void

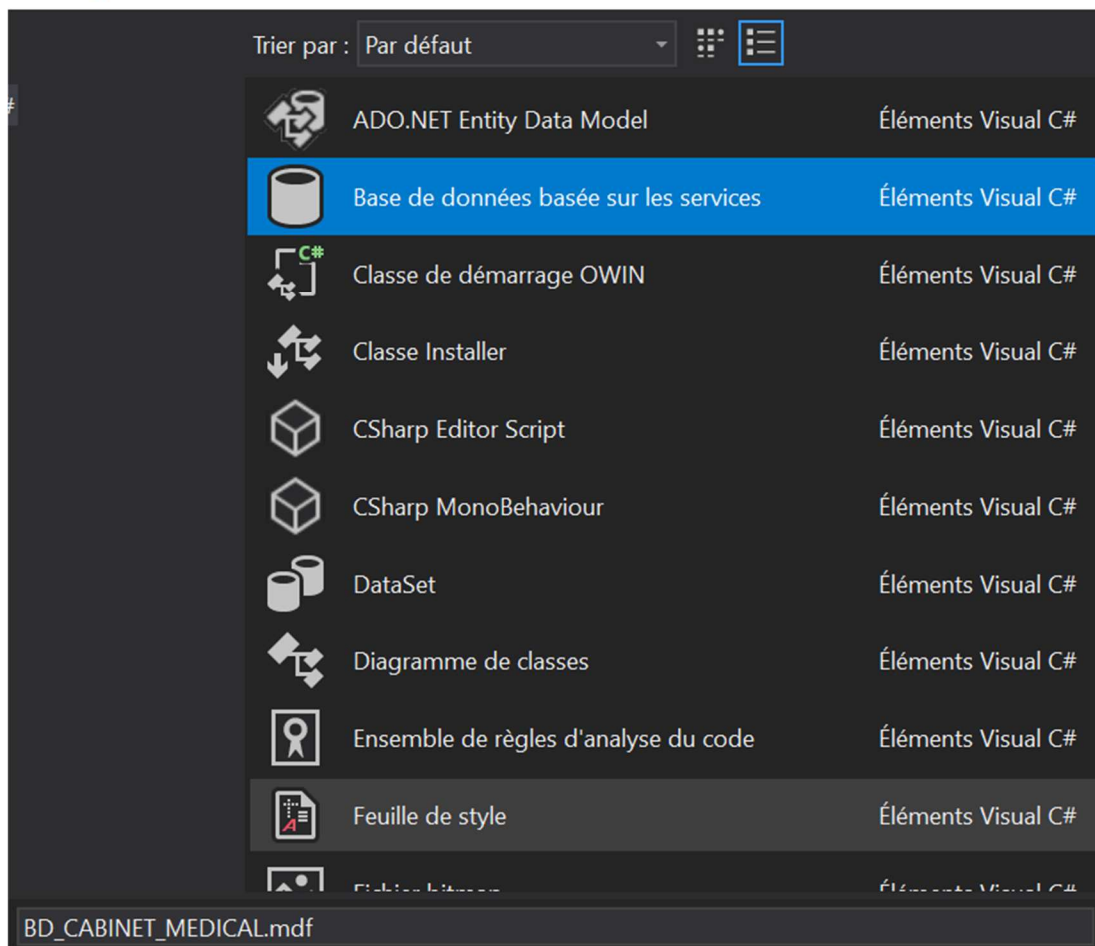
12. Définissez le menu ci-dessous et programmez ses différentes options de menu

```
Console.WriteLine("0. Quitter et sauvegarder");
Console.WriteLine("1. Voir toutes consultations ");
Console.WriteLine("2. Ajouter une consultation");
Console.WriteLine("3. Voir les consultations du jour");
Console.WriteLine("4. Voir les Consultations d'un docteur ");
Console.WriteLine("5. Voir les Consultations Triées par date ");
Console.WriteLine("6. Voir les Consultations Triées par prix ");
Console.WriteLine("7. Voir les Consultations Triées par docteur ");
```

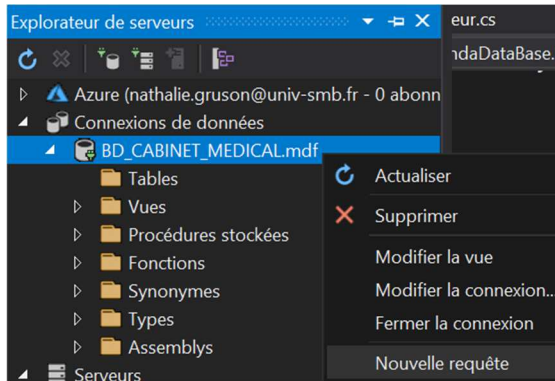
POUR LES + RAPIDES : BASE DE DONNEES

1. Ajouter un nouveau projet « AVEC_BD » à votre solution. Ajoutez une référence au projet précédent.
2. Création de la base de données **locale** : Sur votre projet, *Ajouter > Nouvel élément* : Ajouter une base de données basée sur les services, nommée BD_CABINET_MEDICAL.mdf :

ment - AgendaDataBase



3. Double-cliquer sur le fichier BD_CABINET_MEDICAL.mdf dans l'Explorateur de solution. L'explorateur de serveur s'affiche. Vous pouvez créer des tables



4. Cliquer avec le bouton droit de la souris sur la base de données puis « Nouvelle requête ». Copier-coller le script suivant :

```
Create table Patient(
    id int identity not null primary key,
    nom varchar(50),
    prenom varchar(50) ,
    dateNaissance datetime
)

Create table Docteur(
    id int identity not null primary key,
    categorie varchar(50),
    nom varchar(50)
)

Create table Consultation(
    id int identity not null primary key,
    horaire datetime,
    aDomicile bit,
    estDocteurTraitant bit,
    idDocteur int,
    idPatient int,
    FOREIGN KEY (idDocteur) REFERENCES Docteur(id),
    FOREIGN KEY (idPatient) REFERENCES Patient(id),
)

insert into Patient (nom, prenom, dateNaissance) values ('Rao', 'Lucie', '2000-10-15');
insert into Patient (nom, prenom, dateNaissance) values ('Blaze', 'Max', '2020-10-15');
insert into Patient (nom, prenom, dateNaissance) values ('Rati', 'Léo', '2023-01-01');

insert into Docteur(categorie,nom, prenom) values ('Generaliste', 'Oddou');

insert into Consultation( horaire, aDomicile , estDocteurTraitant , idDocteur , idPatient )
values ('2024-01-31T10:30:00',0,1,1,1);
insert into Consultation( horaire, aDomicile , estDocteurTraitant , idDocteur , idPatient )
values ('2024-01-30T11:30:00',1,1,1,1);
insert into Consultation( horaire, aDomicile , estDocteurTraitant , idDocteur , idPatient )
values ('2024-01-14T15:00:00',0,1,1,1);
insert into Consultation( horaire, aDomicile , estDocteurTraitant , idDocteur , idPatient )
values ('2024-01-14T11:30:00',1,1,1,1);

insert into Consultation( horaire, aDomicile , estDocteurTraitant , idDocteur , idPatient )
values ('2024-01-30T11:30:00',0,1,1,2);
insert into Consultation( horaire, aDomicile , estDocteurTraitant , idDocteur , idPatient )
values ('2024-02-02T13:30:00',1,1,1,2);
insert into Consultation( horaire, aDomicile , estDocteurTraitant , idDocteur , idPatient )
values ('2024-01-28T13:00:00',0,1,1,2);
```



```
insert into Consultation( horaire, aDomicile , estDocteurTraitant , idDocteur , idPatient )
values ('2024-01-28T15:30:00',1,1,1,2);
```

```
insert into Consultation( horaire, aDomicile , estDocteurTraitant , idDocteur , idPatient )
values ('2024-02-06T11:00:00',0,1,1,3);
insert into Consultation( horaire, aDomicile , estDocteurTraitant , idDocteur , idPatient )
values ('2024-02-06T12:00:00',1,1,1,3);
insert into Consultation( horaire, aDomicile , estDocteurTraitant , idDocteur , idPatient )
values ('2024-02-04T13:30:00',0,1,1,3);
insert into Consultation( horaire, aDomicile , estDocteurTraitant , idDocteur , idPatient )
values ('2024-02-04T15:00:00',1,1,1,3);
```

5. Exécuter. Vous pouvez afficher depuis les tables les données. Exemple :

Nombre maximal de lignes : 1000				
	id	nom	prenom	dateNaissance
	1	Rao	Lucie	15/10/2000 00:...
	2	Blaze	Max	15/10/2018 00:...
	3	Rati	Léo	01/01/2021 00:...
	NULL	NULL	NULL	NULL

6. Récupérez la classe DataAccess : pensez à ajouter le package :
System.Data.SqlClient. Elle vous facilitera la vie :
 - a. SetData : pour faire des requêtes de type insert,update,delete
 - b. GetData : pour faire des requêtes de type select

DataAccess
 Classe

▲ Champs

- sqlConnection : SqlConnection

▲ Méthodes

- CloseConnection() : void
- GetData(string getQuery) : DataTable
- OpenConnection() : bool
- SetData(string setQuery) : int

7. Modifiez le chemin de connexion dans DataAccess en le récupérant dans les propriétés de la base de données

Dépendances

- BD_CABINET_MEDICAL.mdf
- BD_CABINET_MEDICAL_log.ldf
- C# DataAccess.cs
- C# Program.cs

Explorateur de solutions Modifications Git

Propriétés

BD_CABINET_MEDICAL.mdf Connexion

(Nom)	C844EFC64B8CE84D499DCF3A103F4169_ASSOCIATION\TD3_TP3_CABINET_
Chaîne de connexion	Data Source=(LocalDB)\MSSQLLocalDB;AttachDbFilename="D:\IUT-ACV\mesf

8. Faites une requête pour récupérer toutes les consultations et recréer une liste de Consultation et ainsi alimenter agenda. Exemple de requête :

```
static void Main(string[] args)
{
    Agenda monAgenda = null;
    try
    {
        monAgenda = new Agenda();
    }
    catch (Exception e)
    {
        Console.WriteLine(e);
        Environment.Exit(-1);
    }

    DataAccess data = new DataAccess();
    DataTable res = data.GetData("select * from patient;");
    foreach (DataRow row in res.Rows)
    {
        Console.WriteLine(row["nom"] + " " + row["prenom"]);
    }
}
```