IPEFA Sup Seraing - Verviers

Projet de développement SGDB

Notes de cours

Georgette Collard

2025-2026

Table des matières

Classe AccesControl

```
static public class AccesConsole
{
    // ------------------------------------------------------------
    // CreerEcran: effacer l'écran et afficher un titre
    // param: le titre de l'écran
    // sortie: aucune
    // ------------------------------------------------------------
    static public void CreerEcran(string s)
    {
        Console.Clear();
        Console.WriteLine(s);
        Console.WriteLine(new String('-', s.Length) + "\n");
    }
    // ------------------------------------------------------------
    // SaisirChaine: saisir une chaîne
    // param: un message d'introduction à afficher
    // sortie: une chaîne
    // ------------------------------------------------------------
    static public string SaisirChaine(string s)
    {
        Console.Write(s);
        return Console.In.ReadLine();
    }
    // ------------------------------------------------------------
    // SaisirDate: saisir un date
    // param: un message d'introduction à afficher
    // sortie: un objet de type DateTime
    // ------------------------------------------------------------
    static public DateTime SaisirDate(string s)
    {
        Console.Write(s);
        return Convert.ToDateTime(Console.In.ReadLine());
    }
```

```
    // ------------------------------------------------------------------------
    // SaisirInt: saisir un entier
    // param: un message d'introduction à afficher
    // sortie: un entier
    // ------------------------------------------------------------------------
    static public int SaisirInt(string s)
    {
        Console.Write(s);
        return Convert.ToInt32(Console.In.ReadLine());
    }
    // ------------------------------------------------------------------------
    // SaisirDouble: saisir un double
    // param: un message d'introduction à afficher
    // sortie: un double
    // ------------------------------------------------------------------------
    static public double SaisirDouble(string s)
    {
        Console.Write(s);
        return Convert.ToDouble(Console.In.ReadLine());
    }


    // ------------------------------------------------------------------------


    // ------------------------------------------------------------------------
    // Attendre: afficher un message puis Attendre la pression d'une touche
    // param: aucun
    // sortie: aucune
    // ------------------------------------------------------------------------
    static public void Attendre()
    {
        Console.Write("\nPressez une touche pour continuer...");
        Console.ReadKey();
        Console.WriteLine("\n");
    }
}
```

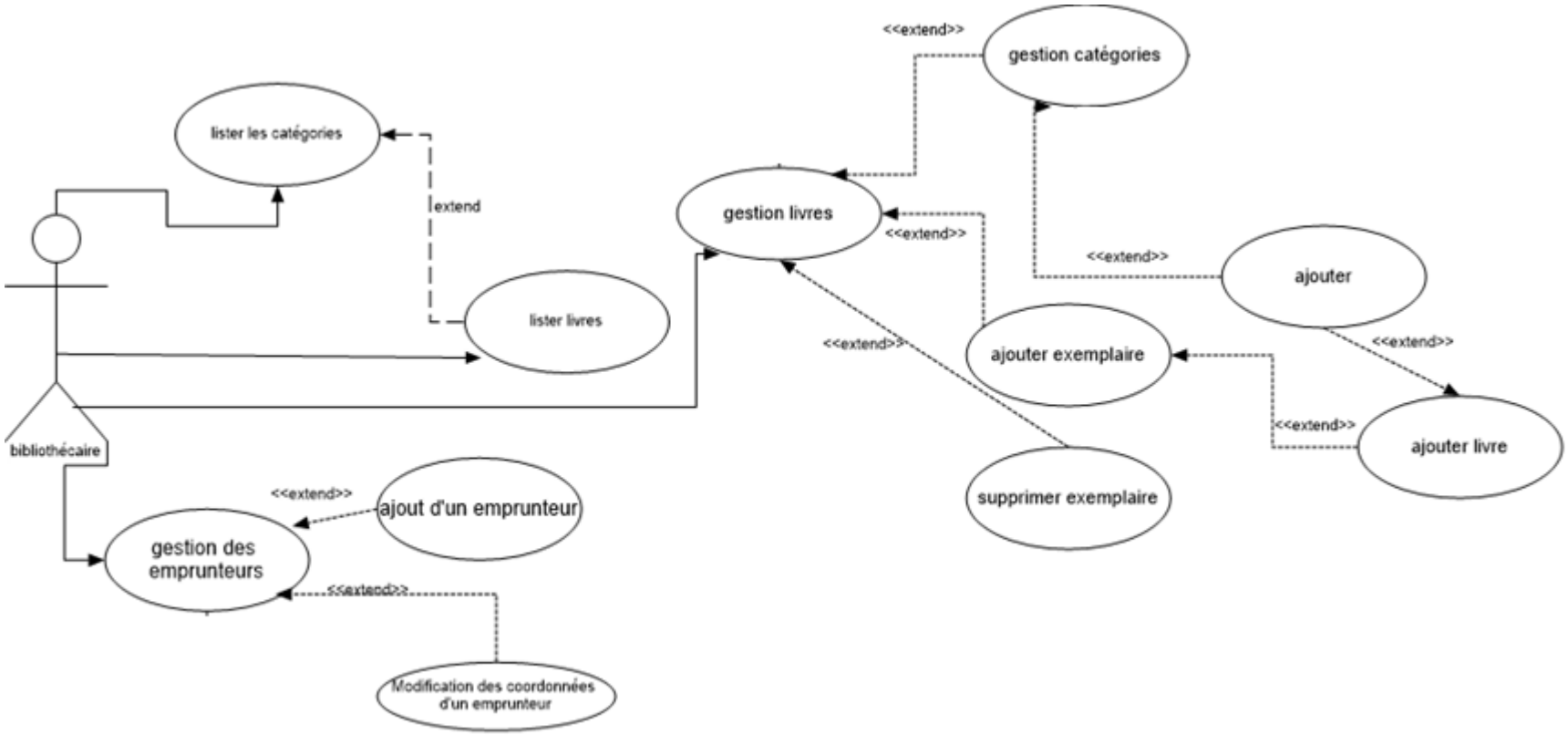

Chapitre 5 : PostgreSQL server et application console en C#

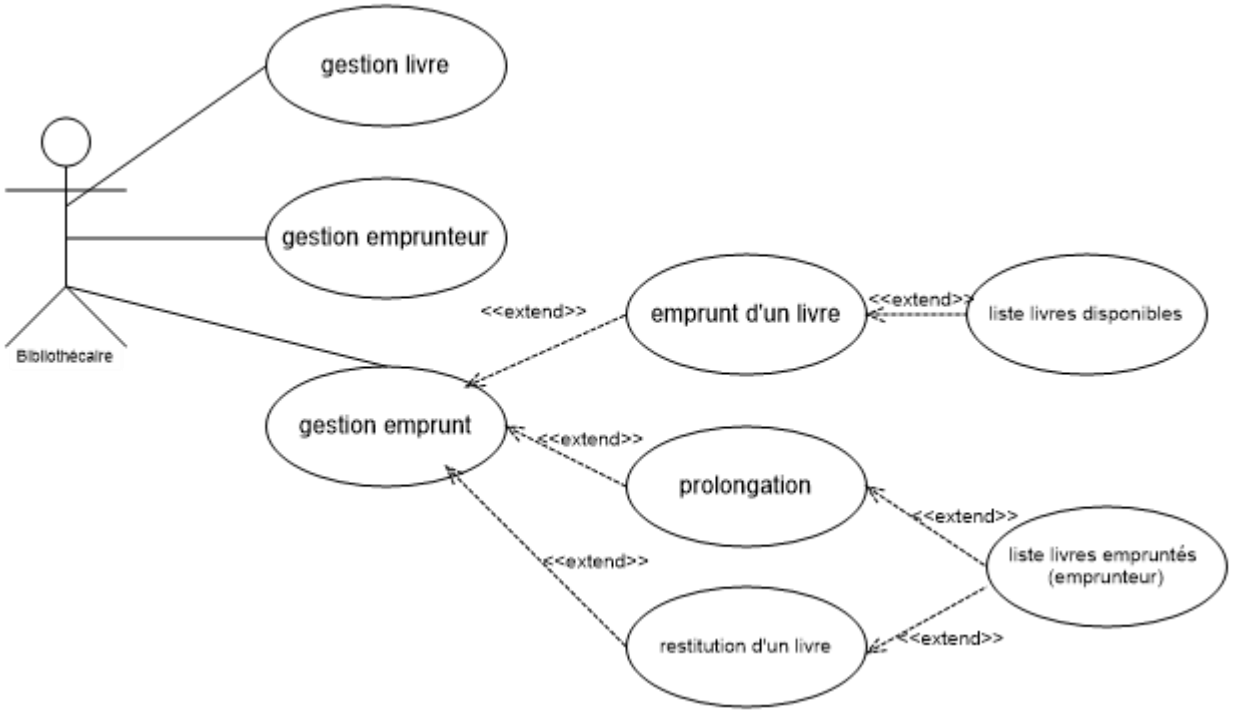
Aperçu de l'application finale

Apres avoir discuté avec un client fictif, voici les premières fonctionnalités qu'il veut obtenir dans l'application de *Bibliotheque* :

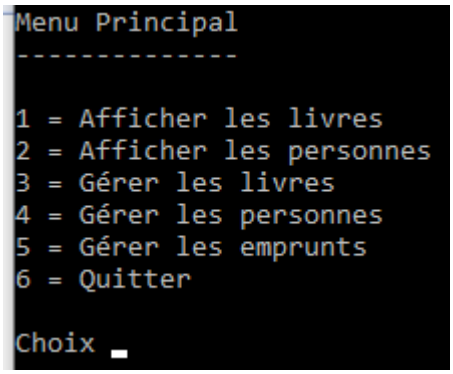
- Obtenir la liste des catégories
- Obtenir la liste des livres appartenant à une catégorie
- Obtenir les informations sur les exemplaires (identifiant, emprunté/disponible) dont le titre et l'auteur sont donnés
- Obtenir la liste des emprunteurs
- Obtenir les informations sur un emprunteur
- Ajout d'un emprunteur
- Ajout d'une catégorie
- Ajout d'un livre/exemplaire

- Supprimer un livre/exemplaire
- Modifier les coordonnées d'un emprunteur
- Faire en sorte qu'un exemplaire est emprunté par une personne
- Faire en sorte qu'un exemplaire est restitué
- Faire en sorte qu'un exemplaire est prolongé

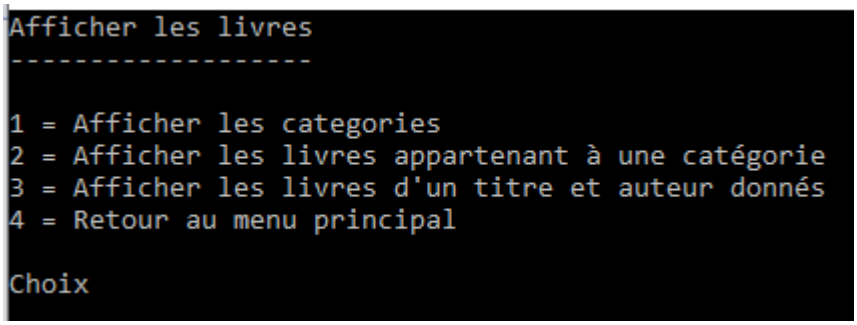

lister les catégories
extend
lister livres
bibliothécaire
gestion livres
<<extend>>
gestion catégories
<<extend>>
ajouter
<<extend>>
ajouter exemplaire
<<extend>>
ajouter livre
<<extend>>
supprimer exemplaire
<<extend>>
ajout d'un emprunteur
gestion des emprunteurs
<<extend>>
Modification des coordonnées d'un emprunteur


gestion livre
gestion emprunteur
Bibliothécaire
<<extend>>
emprunt d'un livre
<<extend>>
liste livres disponibles
gestion emprunt
<<extend>>
prolongation
<<extend>>
liste livres empruntés
(emprunteur)
<<extend>>
restitution d'un livre
<<extend>>

Menu principal



Quand on sélectionne 1 dans le menu principal, le l'écran *« Afficher les livres »* apparaît



Quand on sélectionne 1 dans le menu *Afficher les livres*, la liste des catégories stockées dans la base de données s'affiche

```
Afficher les catégorie
----------------------


 liste obtenu

idCategorie, nom, sous-catégorie:

9, bande dessinée, humoristique
8, Bande dessinée, historique
6, Bande dessinée,
7, Géographie,
5, Poesie,
2, roman, historique
3, roman, philosophique
4, roman, policier
1, roman, Science-fiction

Pressez une touche pour continuer...
```

Quand on sélectionne 2 dans le menu *Afficher les livres*, puis qu'on entre l'identifiant de la catégorie liste des livres appartenant à cette catégorie s'affichent

```
Lister les livres d'une catégorie donnée
----------------------------------------

Entrez l'identifiant de la catégorie : 4

 liste obtenu

isbn, titre, auteur,  edition:

9782253010210, Le Crime de l'Orient Express, Agatha Christie, Le livre de poche
9782702413937, Un cadavre dans la bibliothèque, Agatha Christie, Le livre de poche
9782253142921, Le chien jaune, Georges Simenon, Lgf

Pressez une touche pour continuer...
```

Quand on sélectionne 3 dans le menu *Afficher les livres*, puis qu'on entre le titre et l'auteur, la liste des exemplaires empruntés et disponibles correspondant à ce titre et cet auteur s'affichent

```
Afficher information d'un livre donné
-------------------------------------

Entrez le titre du livre : Le Crime de l'Orient Express
Entrez l'auteur du livre : Agatha Christie

 liste obtenu

8, Emprunte
15, Emprunte
18, Emprunte

Pressez une touche pour continuer...
```

Quand on sélectionne 2 dans le menu principal le menu « Afficher les personnes » s'affiche

```
Afficher les personnes
----------------------

1 = Afficher toutes les personnes
2 = Afficher les informations d'une personne donnée
3 = Retour au menu principal

Choix
```

Quand on sélectionne 1 dans le menu *Afficher les personne*, la liste des personnes stockée dans la base de données s'affichent

```
Afficher les personnes
----------------------


 liste obtenu

idpersonne, nom, prenom, gsm, (rue, cp, localite):

8, Beckers, Anne, 043987654, (rue de la Paix, 43, 4100,Seraing):
7, Bras, Bernard, 042123456, (Place du Marché, 4, 4000,Liège):
9, Bryers, Pierre, 087654321, (rue du Moulin, 109, 4800,Verviers):
4, Dupont, Quentin, 042123456, (rue de la Paix, 6, 4000,Liège):
5, Durant, Pierre, 056432187, (rue de la Place, 8, 4800,Verviers):
12, Egmont, Simon, 048765432, (Place Verte, 87, 4800,Verviers):
6, Legrand, Marie, 098765432, (rue de la Liberté, 98, 4100,Seraing):
10, Molders, Bernard, 042123456, (Place du Marché, 4, 4000,Liège):
11, Wyn, Anne, 043987654, (rue de la Paix, 79, 4100,Seraing):

Pressez une touche pour continuer...
```

Quand on sélectionne 2 dans le menu *Afficher les personnes*, puis on entre l'identifiant de la personne, les informations de cette personne sont affichée

```
Afficher information personne
-----------------------------

Entrez l'identifiant de la personne : 11
idpersonne, nom, prenom, gsm, (rue, cp, localite):

11, Wyn, Anne, 043987654, (rue de la Paix, 79, 4100,Seraing):

Pressez une touche pour continuer...
```

Quand on sélectionne 2 dans le menu principal le menu «*Gestion des livres* » s'affiche

```
Gérer les livres
----------------

1 = Ajouter une categorie
2 = Ajouter un livre
3 = Supprimer un livre
4 = Retour au menu principal

Choix
```

Quand on sélectionne 1 dans le menu *Gérer les livres* puis qu'on entre le nom de la catégorie et le nom de la sous-catégorie, cette catégorie est ajoutée dans la base de donnée.

```
Ajouter une catégorie
---------------------

Entrez le nom de la catégorie: informatique
Entrez le nom de la sous-catégorie: base de données

L'ajout s'est bien déroulé

Pressez une touche pour continuer...
```

Quand on sélectionne 2 dans le menu *Gérer les livres* puis qu'on entre l'isbn, si aucun livre avec cet isbn existe, on entre le titre, l'auteur, l'édition et l'identifiant de la catégorie. Une insertion dans la table livre et une insertion dans la table exemplaire sont faites. Si un livre avec cet isbn existe, alors un exemplaire est ajouté à la base de données.

```
Ajouter un livre
----------------

Entrez isbn: 2729872302

Premier exemplaire

Entrez le titre: Le langage Python
Entrez l'auteur: Pierre Puiseux
Entrez l'édition: Ellipses
Entrez l'identifiant de la catégorie: 40

L'ajout s'est bien déroulé

Pressez une touche pour continuer...
```

```
Ajouter un livre
----------------

Entrez isbn: 2729872302

L'ajout s'est bien déroulé

Pressez une touche pour continuer...
```

Ecran Gestion des personnes

```
Gérer les personnes
-------------------

 1 = Ajouter une personne
 2 = Modifier les coordonnées d'une personne
 3 = Retour au menu principal

 Choix
```

Ecran Gestion des Emprunt

```
Gérer les emprunts
------------------


1 = Ajouter un emprunt d'un livre par une personne
2 = Restituer un livre
3 = Prolonger un emprunt
4 = Retour au menu principal

Choix
```

La base de données

Voici le modèle entité-association

Le modèle relationnel est :

PERSONNE(idPersonne, nom, prenom, GSM, (rue, cp, localite))

CATEGORIE(idCategorie, nom, sous_categorie)

LIVRE(isbn, titre, auteur, edition,#idCategorie)

EXEMPLAIRE(idLivre, #isbn)

EMPRUNT(#idPersonne, #iidLivre, date_sortie, date_retour, date_prolongation)

Création de tables en postgreSQL

Définition d'un type composite : adresse

```
create type adresse as (
    rue varchar(35),
    cp  varchar(6),
    localite varchar(20));
```

Création des tables

```
create table PERSONNE(
    idPersonne SERIAL  primary key ,
    nom varchar(15),
    prenom varchar(15),
    date_naissance date,
    gsm varchar(12),
    adressePersonne adresse);
```

```
create table CATEGORIE(
    idCategorie SERIAL primary key,
    nom varchar(30),
    sousCategorie varchar(50));

create table LIVRE(
    isbn varchar(30) primary key,
    idCategorie integer,
    titre varchar(100),
    auteur varchar(50),
    edition varchar(30),
    foreign key (idCategorie) references CATEGORIE(idCategorie));
```

```
create table EXEMPLAIRE(
idLivre serial primary key,
isbn varchar(18),
    foreign key(isbn) references LIVRE(isbn) );

create table EMPRUNT(
    idPersonne integer,
    isbn varchar(30),
    date_sortie date,
    date_prolongation date,
    date_retour date,
    primary key (idPersonne, isbn, date_sortie));
```

Contraintes

```
alter table categorie alter column nom set not null;

alter table livre alter column titre set not null;

alter table livre alter column auteur set not null;

alter table livre alter column edition set not null;

alter table categorie add constraint nomPasChaineVide check (char_length(nom) > 0);

alter table livre add constraint titrePasChaineVide check (char_length(titre) > 0);

alter table livre add constraint  auteurPasChaineVide check (char_length(auteur) > 0);

alter table livre add constraint editionPasChaineVide check (char_length(edition) > 0);

alter table emprunt add constraint prolongationPosSortie
check (date_prolongation between date_sortie and (date_sortie + 28));

alter table emprunt add constraint retourPosSortieAndPosProlongation
    check ((date_retour >= date_sortie) and
           ((date_prolongation is null) or (date_retour >= date_prolongation)));

alter table personne add constraint nomAuMoins2Car check (char_length(nom) >= 2);

alter table personne add constraint prenomAuMoins2Car
    check (char_length(prenom) >= 2);

alter table personne add constraint dateNaissanceAntCurrent check (date_naissance < current_date);
```

Conception de l'application

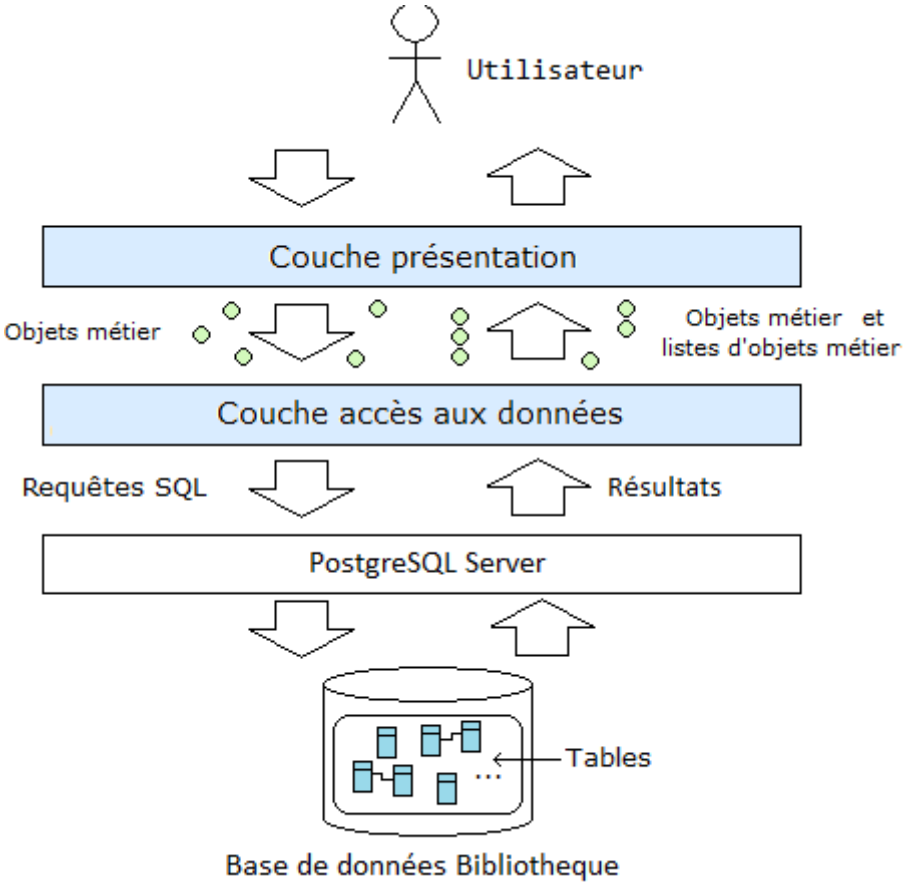
Architecture à deux couches

1) L'application conçue en C# de « *Bibliotheque* » que nous allons décrire dans ce document est constituée de 2 couches distinctes:

   Couche présentation (*presentation layer*): cette couche gère les accès à la console (clavier et écran) et effectue les traitements sur les données : tests, calculs, ... Dans notre application, il va s'agir d'une classe appelée *Presentation*.

2) **Couche accès aux données (data access layer)**: cette couche effectue les accès en lecture et en écriture dans la base de données.

Voici une figure montrant ou se situe chaque couche:



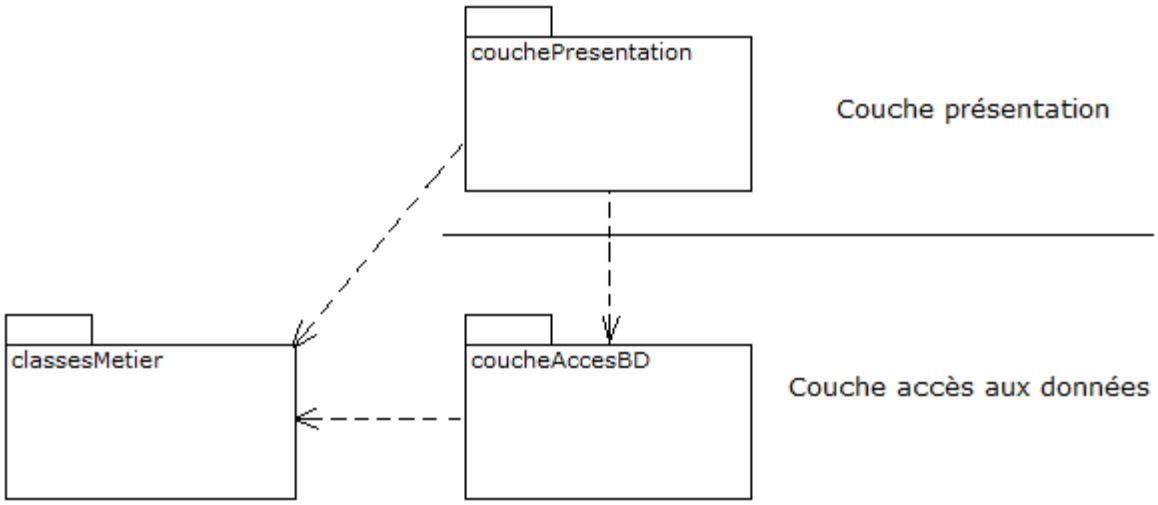
Pendant l'exécution de l'application, les objets des classes *Presentation* et *AccesBD* s'échangent des objets de classes directement construites à partir des tables de la base de données (CATEGORIE, LIVRE, PERSONNE, …). Ces objets sont appelés objets métier.

Pourquoi avoir divisé l'application en plusieurs couches différentes?

- On peut changer la nature de l'interface utilisateur (ex: passer d'une application console à une application fenêtrée), donc changer la couche présentation, sans avoir à modifier, ajouter ou supprimer la moindre ligne de code dans les autres classes.
- On peut changer la couche accès aux données pour qu'elle prenne, par exemple, en compte l'usage de procédures stockées, sans avoir à modifier, ajouter ou supprimer la moindre ligne de code dans les autres classes.

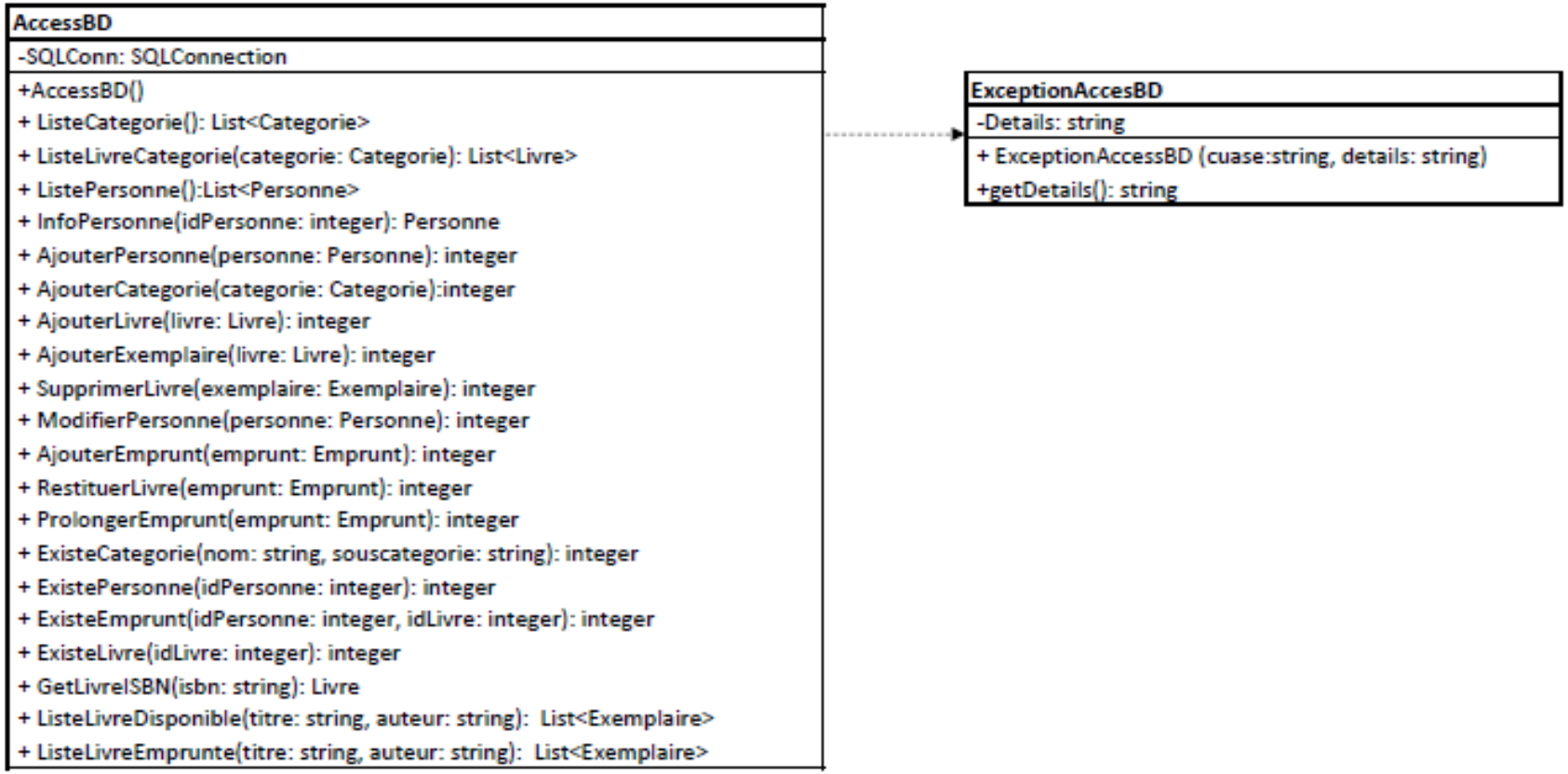
Espaces de noms

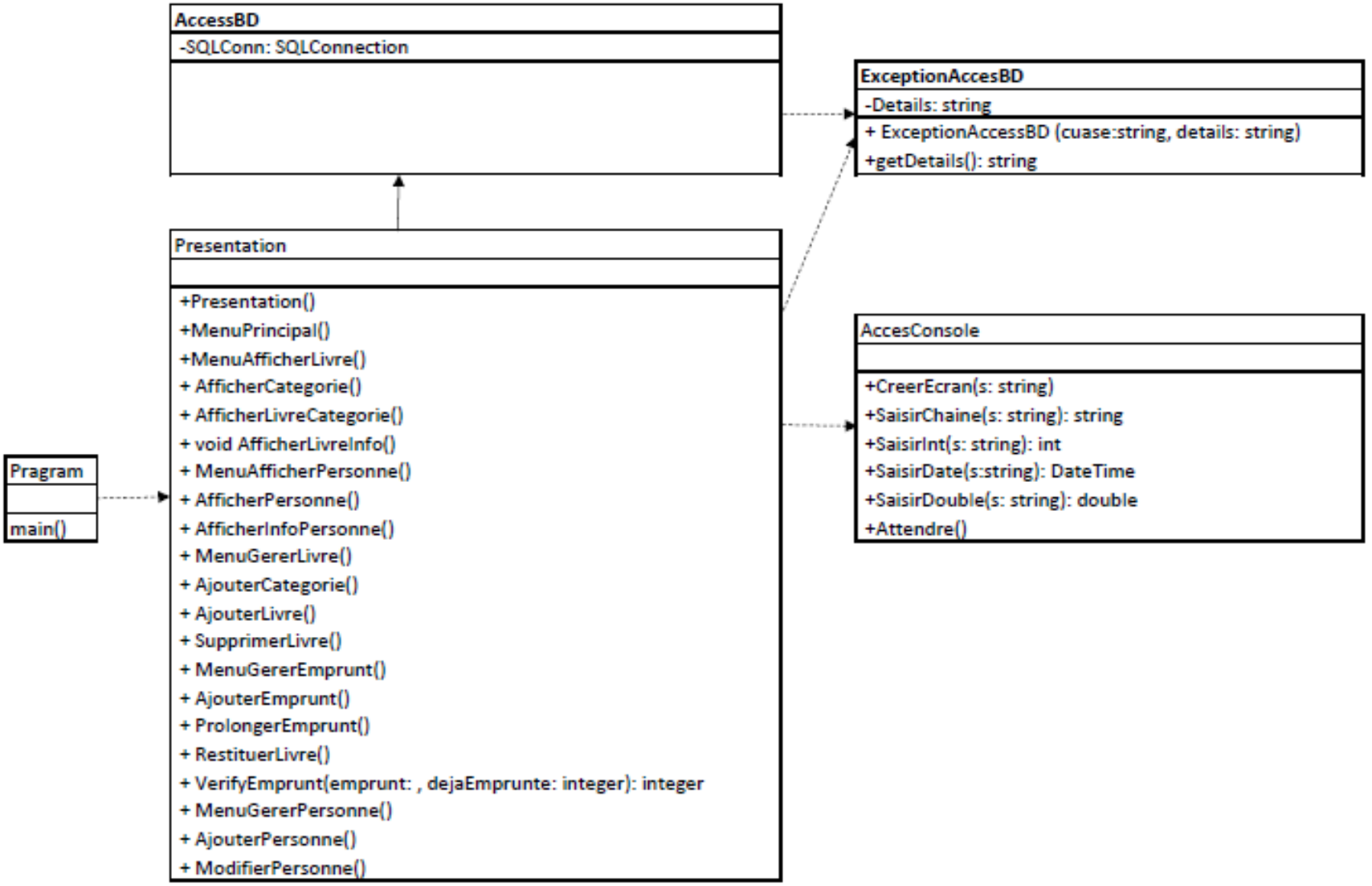
Sur base de l'architecture de l'application, voici les espaces de noms qui vont être créés dans l'application:



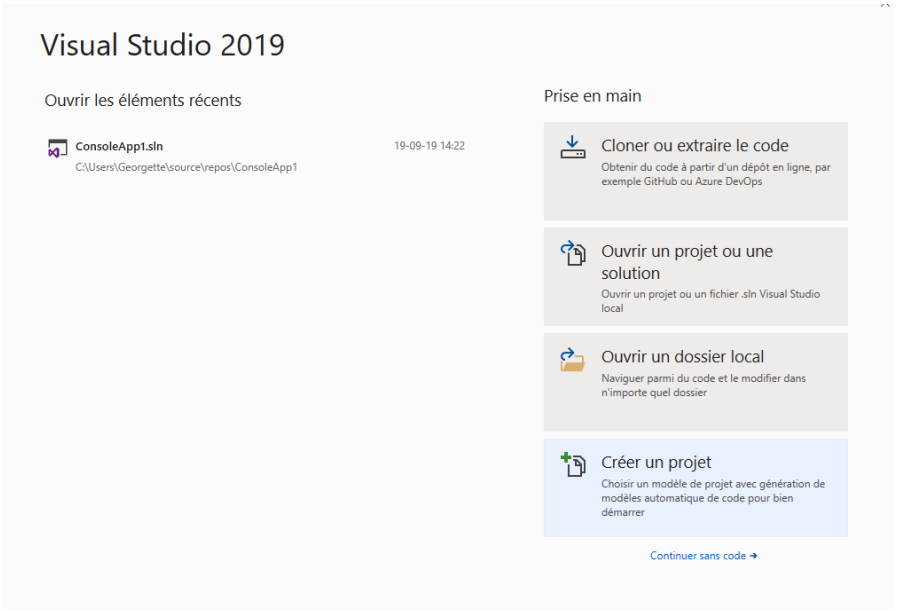
Trois espaces de noms vont être créés :

- L'espace de noms **couchePresentation** va stocker les classes d'accès à l'écran et au clavier.
- L'espace de noms **coucheAccesBD** va stocker les classes d'accès à la base de données.
- L'espace de noms **classesMetier** va stocker les classes métier.

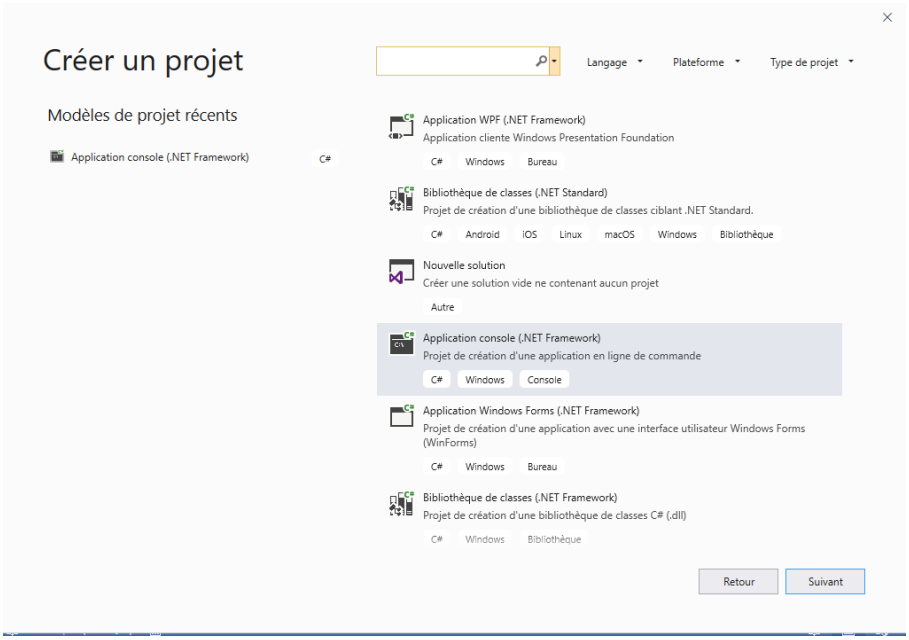

AccessBD
-SQLConn: SQLConnection
+AccessBD()
+ ListeCategorie(): List<Categorie>
+ ListeLivreCategorie(categorie: Categorie): List<Livre>
+ ListePersonne():List<Personne>
+ InfoPersonne(idPersonne: integer): Personne
+ AjouterPersonne(personne: Personne): integer
+ AjouterCategorie(categorie: Categorie):integer
+ AjouterLivre(livre: Livre): integer
+ AjouterExemplaire(livre: Livre): integer
+ SupprimerLivre(exemplaire: Exemplaire): integer
+ ModifierPersonne(personne: Personne): integer
+ AjouterEmprunt(emprunt: Emprunt): integer
+ RestituerLivre(emprunt: Emprunt): integer
+ ProlongerEmprunt(emprunt: Emprunt): integer
+ ExisteCategorie(nom: string, souscategorie: string): integer
+ ExistePersonne(idPersonne: integer): integer
+ ExisteEmprunt(idPersonne: integer, idLivre: integer): integer
+ ExisteLivre(idLivre: integer): integer
+ GetLivreISBN(isbn: string): Livre
+ ListeLivreDisponible(titre: string, auteur: string): List<Exemplaire>
+ ListeLivreEmprunte(titre: string, auteur: string): List<Exemplaire>
ExceptionAccesBD
-Details: string
+ ExceptionAccessBD (cuase:string, details: string)
+getDetails(): string


AccessBD
-SQLConn: SQLConnection
ExceptionAccesBD
-Details: string
+ ExceptionAccessBD (cuase:string, details: string)
+getDetails(): string
Presentation
+Presentation()
+MenuPrincipal()
+MenuAfficherLivre()
+ AfficherCategorie()
+ AfficherLivreCategorie()
+ void AfficherLivreInfo()
+ MenuAfficherPersonne()
+ AfficherPersonne()
+ AfficherInfoPersonne()
+ MenuGererLivre()
+ AjouterCategorie()
+ AjouterLivre()
+ SupprimerLivre()
+ MenuGererEmprunt()
+ AjouterEmprunt()
+ ProlongerEmprunt()
+ RestituerLivre()
+ VerifyEmprunt(emprunt: , dejaEmprunte: integer): integer
+ MenuGererPersonne()
+ AjouterPersonne()
+ ModifierPersonne()
AccesConsole
+CreerEcran(s: string)
+SaisirChaine(s: string): string
+SaisirInt(s: string): int
+SaisirDate(s:string): DateTime
+SaisirDouble(s: string): double
+Attendre()
Pragram
main()

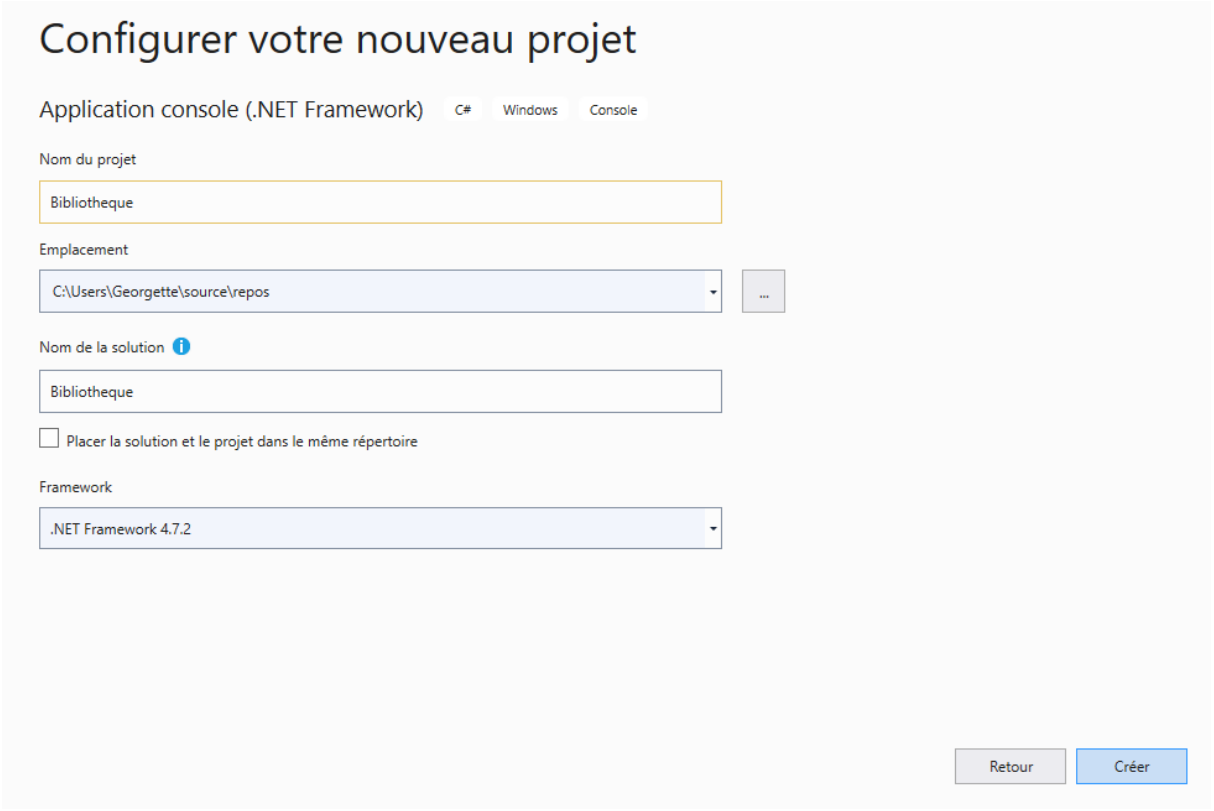
Chapitre 6 : Création du projet de l'application dans Visual C#



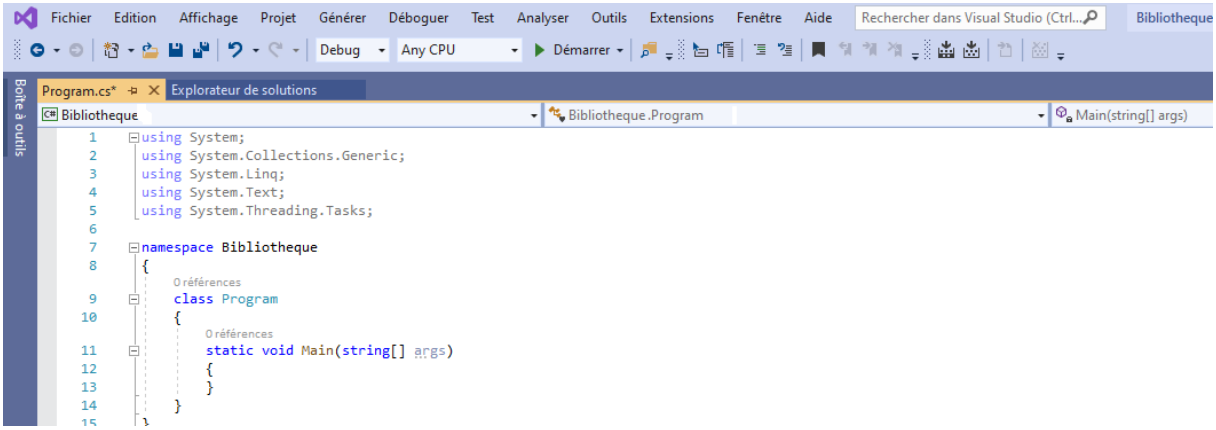
➢ Cliquer sur Créer un projet



➢ Choisir Application Console et cliquer sur Suivant



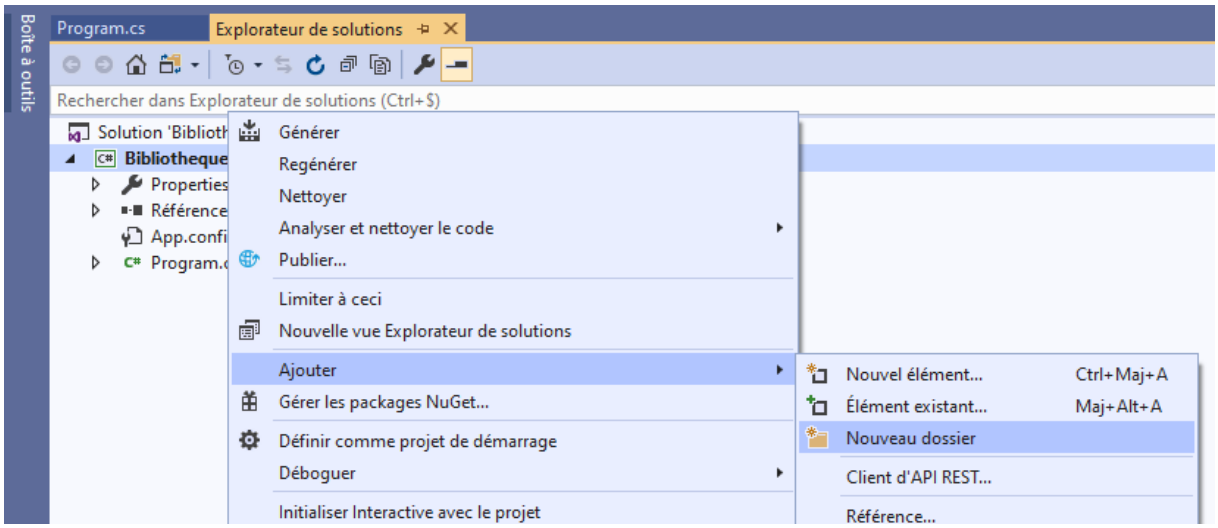
- Donner un nom à votre projet et cliquer sur Créer

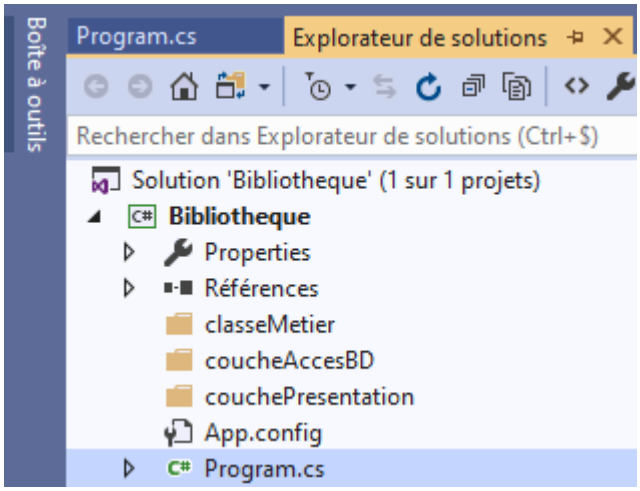


Le projet contient un fichier appelé *Program.cs*. Dans ce fichier se trouve comme contenu initial la classe *Program* contenant une méthode statique *Main*. Celle-ci sera la toute première méthode exécutée lorsque l'exécution du programme commencera.

Dans le projet, créez les dossiers : *classesMetier, coucheAccesBD* et *couchePresentation*.

Pour créer un dossier, clic droit sur le nom du *projet -> Ajouter -> Nouveau dossier*.





Signification des espaces de noms:

- Dans le dossier *classesMetier*, on va placer les fichiers qui vont stocker les classes metier : *Categorie, Livre, Personne, Emprunt.*

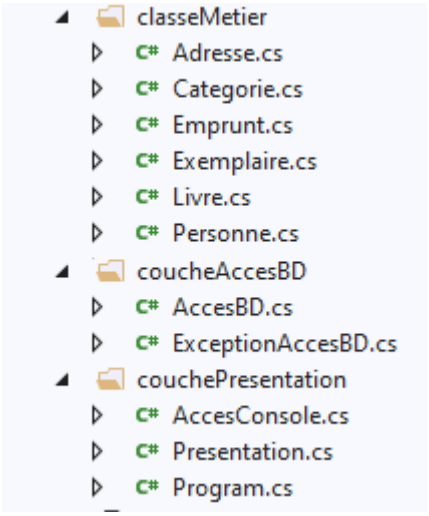
- Dans le dossier *coucheAccesBD*, on va placer les fichiers qui vont stocker les classes permettant les accès aux données : *AccesBD* et *ExceptionAccesBD*.

- Dans le dossier *couchePresentation*, on va placer les fichiers qui vont stocker les classes permettant les accès à la console : *AccesConsole* et *Presentation* ainsi que la classe P*rogram*.

Chaque dossier va correspondre à un espace de noms spécifique.

Aperçu du projet général terminé

Le fichier *Program.cs* a été déplacé dans le dossier *couchePresentation*.


classeMetier
Adresse.cs
Categorie.cs
Emprunt.cs
Exemplaire.cs
Livre.cs
Personne.cs
coucheAccesBD
AccesBD.cs
ExceptionAccesBD.cs
couchePresentation
AccesConsole.cs
Presentation.cs
Program.cs

Exécution de l'application

Voici comment les éléments de l'application vont fonctionner ensemble :

1. La classe *Program* contient la méthode statique *Main*. Cette méthode est automatiquement exécutée quand démarre le programme.

2. La méthode *Main* crée un objet de la classe *Presentation*.

3. Le constructeur de la classe *Presentation* crée un objet de la classe *AccesBD*.

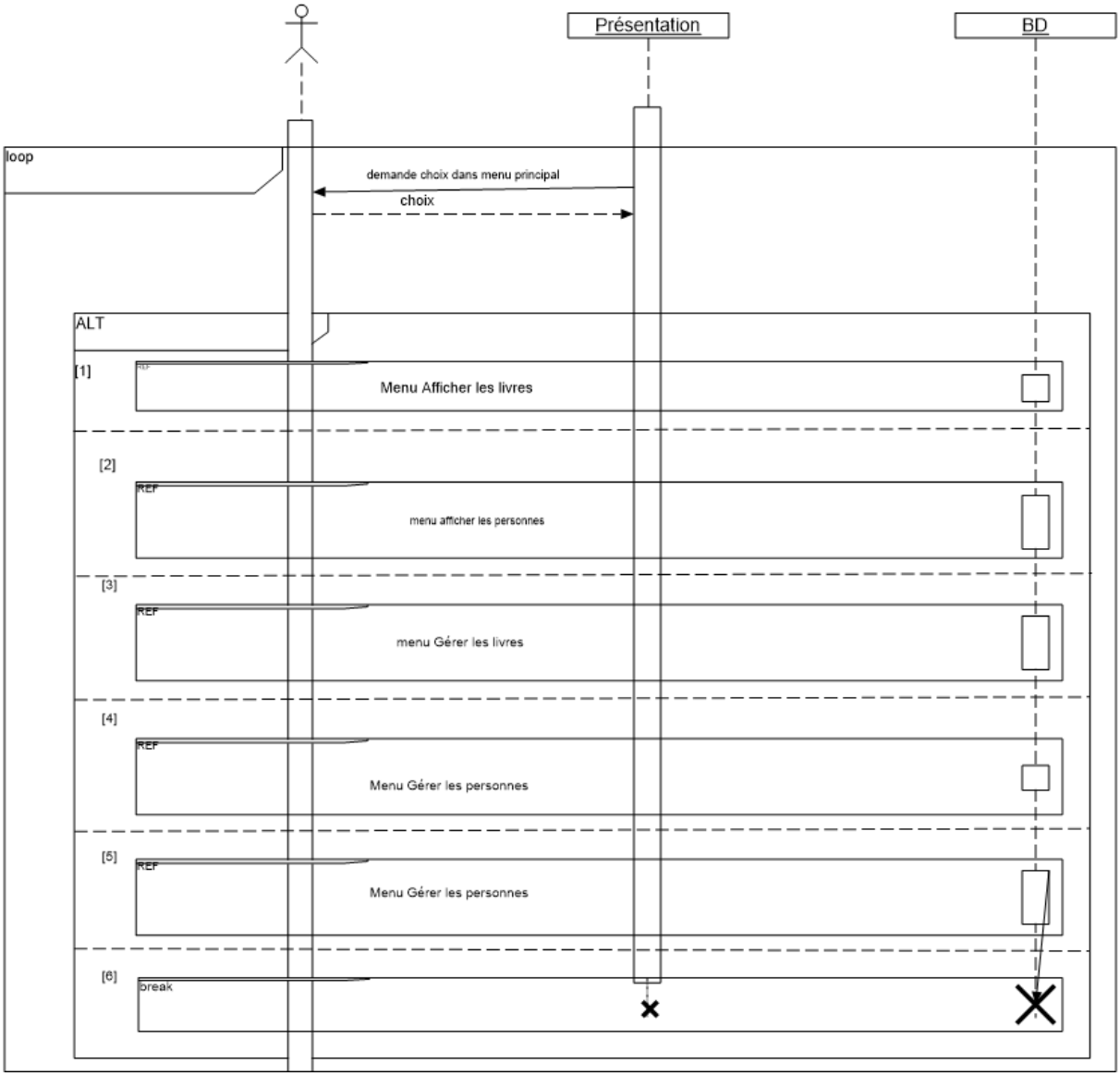
4. La méthode *Main* donne ensuite le contrôle à la méthode *menuPrincipa*l de l'objet de la classe *Presentation*.

5. La méthode *MenuPrincipa*l affiche le menu principal de l'application. Dès que l'utilisateur fait un choix, cette methode invoque la methode en relation avec le choix effectué par l'utilisateur. Par exemple, si l'utilisateur choisit 2, la methode *MenuAfficherPersonne()* est exécutée. Le menu Afficher Personne s'affiche, si l'utilisateur choisit 1, la méthode *AfficherPersonne()* est exécutée.

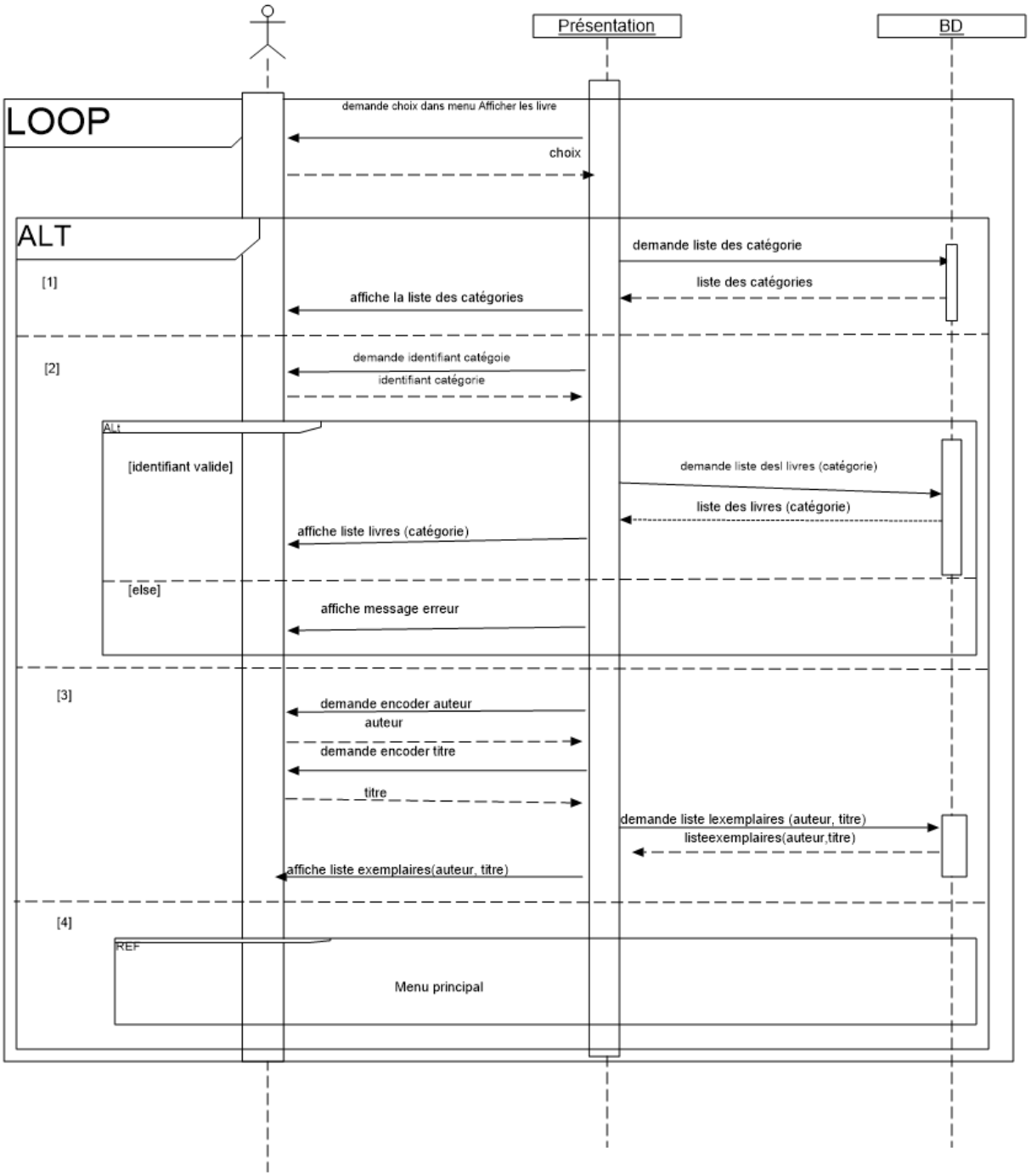
6. La methode *AfficherPersonne* de l'objet de la classe *Presentation* appelle la methode *ListePersonne* de l'objet de la classe *AccesBD* pour obtenir de la base de données la liste des personnes.

Diagramme de séquence

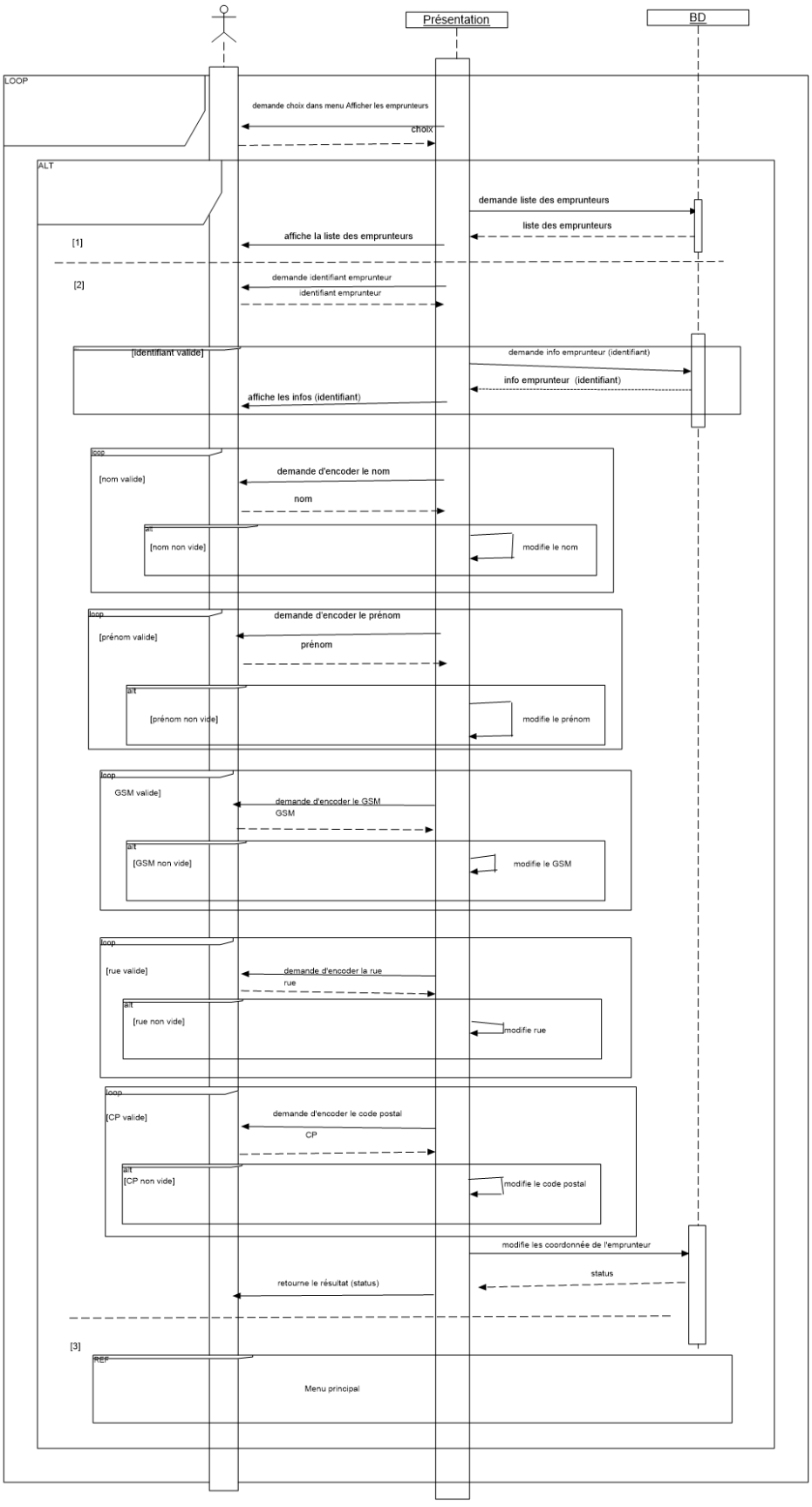
Menu principal


Présentation
BD
loop
demande choix dans menu principal
choix
ALT
[1]
Menu Afficher les livres
[2]
REF
menu afficher les personnes
[3]
REF
menu Gérer les livres
[4]
REF
Menu Gérer les personnes
[5]
REF
Menu Gérer les personnes
[6]
break

Menu Afficher les livres



Menu Afficher Personne

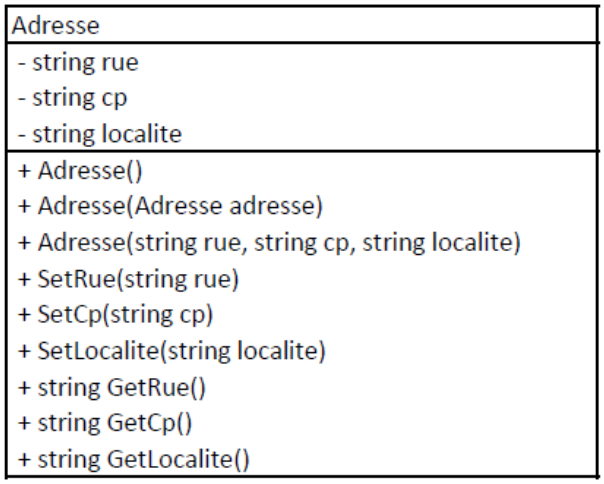


Chapitre 7 : Les classes métiers

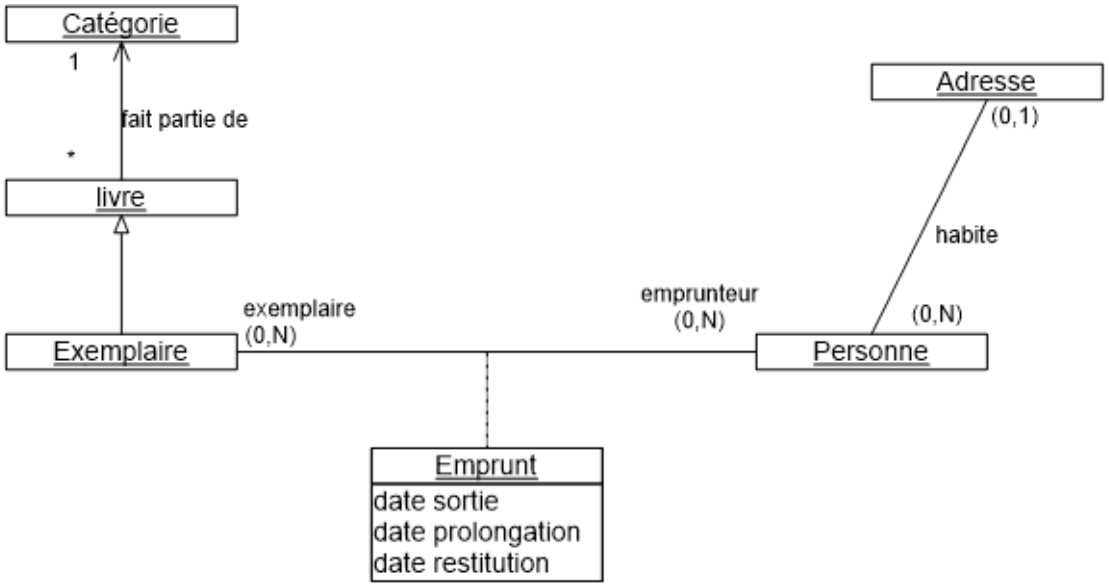
Tout comme les tables de la base de données, les **classes métier** décrivent des entités du métier (compte, client, transaction ..., pour une banque). Les instances des classes métier sont les **objets métier**.

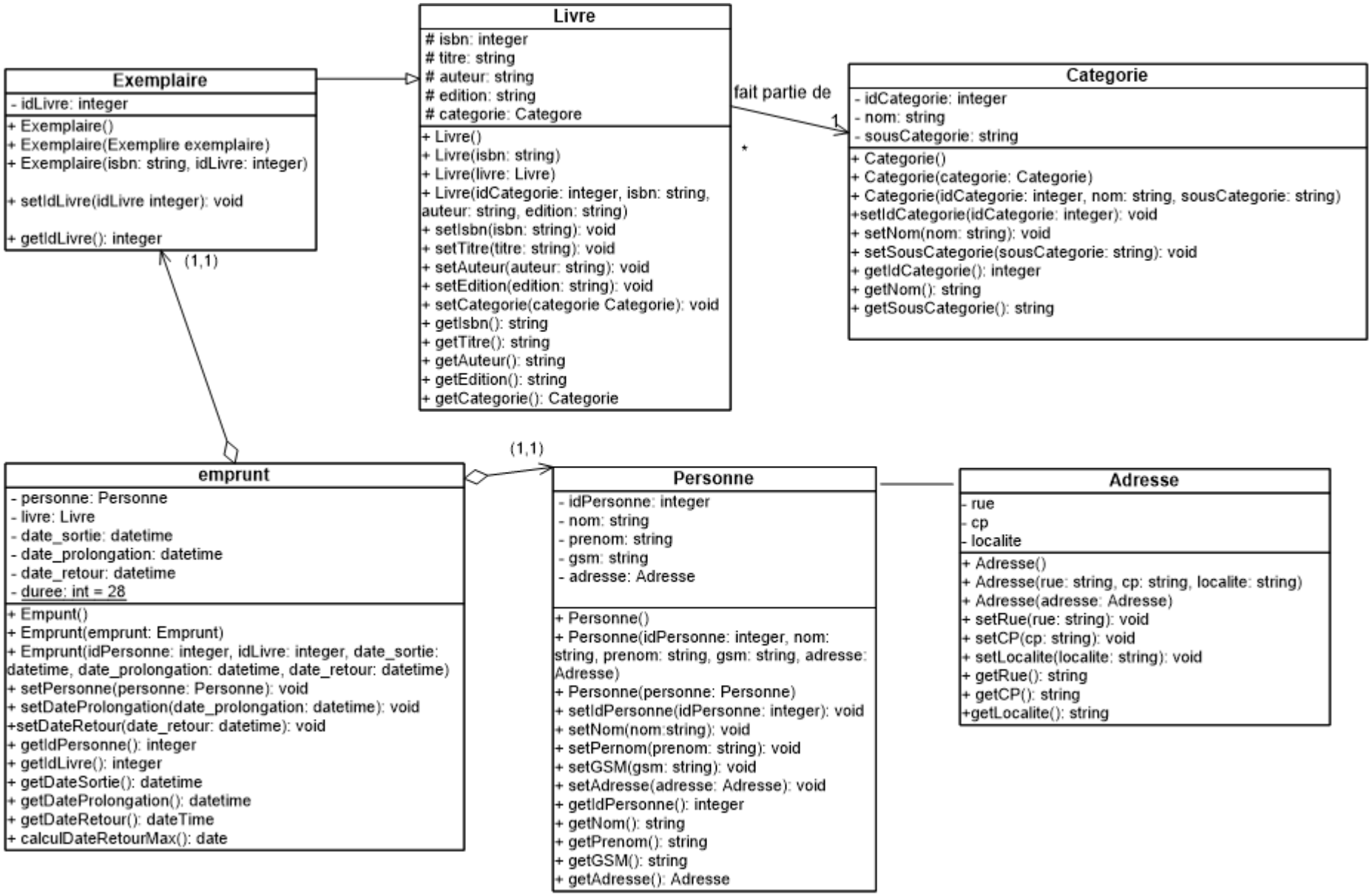
Diagrammes de classes

Nous allons convertir les tables *CATEGORIE, LIVRE, EXEMPLAIRE, PERSONNE, EMPRUNT* en les classes *Catégorie, Livre, Exemplaire, Personne, Emprunt*.



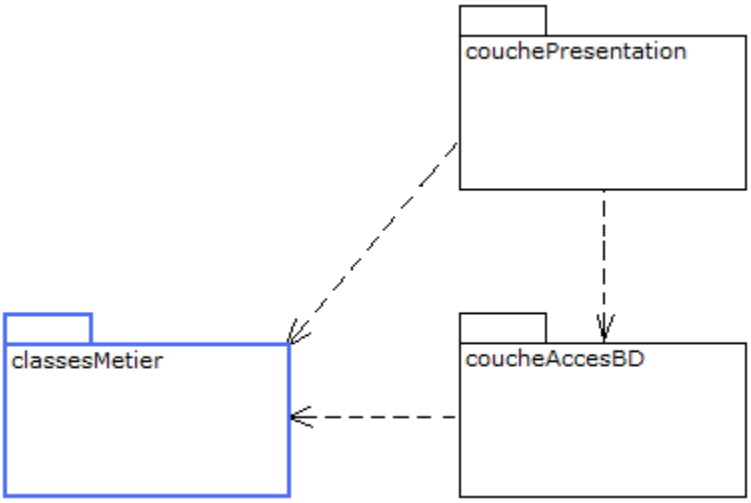
Représente le type composite *adresse* défini dans *postgreSQ*L. Ceci est un composant de la classe *Personne*




Livre
isbn: integer
titre: string
auteur: string
edition: string
categorie: Categore
+ Livre()
+ Livre(isbn: string)
+ Livre(livre: Livre)
+ Livre(idCategorie: integer, isbn: string, auteur: string, edition: string)
+ setIsbn(isbn: string): void
+ setTitre(titre: string): void
+ setAuteur(auteur: string): void
+ setEdition(edition: string): void
+ setCategorie(categorie Categorie): void
+ getIsbn(): string
+ getTitre(): string
+ getAuteur(): string
+ getEdition(): string
+ getCategorie(): Categorie
Exemplaire
- idLivre: integer
+ Exemplaire()
+ Exemplaire(Exemplire exemplaire)
+ Exemplaire(isbn: string, idLivre: integer)
+ setIdLivre(idLivre integer): void
+ getIdLivre(): integer
fait partie de
1
*
Categorie
- idCategorie: integer
- nom: string
- sousCategorie: string
+ Categorie()
+ Categorie(categorie: Categorie)
+ Categorie(idCategorie: integer, nom: string, sousCategorie: string)
+setIdCategorie(idCategorie: integer): void
+ setNom(nom: string): void
+ setSousCategorie(sousCategorie: string): void
+ getIdCategorie(): integer
+ getNom(): string
+ getSousCategorie(): string
(1,1)
(1,1)
emprunt
- personne: Personne
- livre: Livre
- date_sortie: datetime
- date_prolongation: datetime
- date_retour: datetime
- duree: int = 28
+ Empunt()
+ Emprunt(emprunt: Emprunt)
+ Emprunt(idPersonne: integer, idLivre: integer, date_sortie: datetime, date_prolongation: datetime, date_retour: datetime)
+ setPersonne(personne: Personne): void
+ setDateProlongation(date_prolongation: datetime): void
+setDateRetour(date_retour: datetime): void
+ getIdPersonne(): integer
+ getIdLivre(): integer
+ getDateSortie(): datetime
+ getDateProlongation(): datetime
+ getDateRetour(): dateTime
+ calculDateRetourMax(): date
Personne
- idPersonne: integer
- nom: string
- prenom: string
- gsm: string
- adresse: Adresse
+ Personne()
+ Personne(idPersonne: integer, nom: string, prenom: string, gsm: string, adresse: Adresse)
+ Personne(personne: Personne)
+ setIdPersonne(idPersonne: integer): void
+ setNom(nom:string): void
+ setPernom(prenom: string): void
+ setGSM(gsm: string): void
+ setAdresse(adresse: Adresse): void
+ getIdPersonne(): integer
+ getNom(): string
+ getPrenom(): string
+ getGSM(): string
+ getAdresse(): Adresse
Adresse
- rue
- cp
- localite
+ Adresse()
+ Adresse(rue: string, cp: string, localite: string)
+ Adresse(adresse: Adresse)
+ setRue(rue: string): void
+ setCP(cp: string): void
+ setLocalite(localite: string): void
+ getRue(): string
+ getCP(): string
+getLocalite(): string

Espace de noms *classesMetier*

Les classes décrites dans ce chapitre sont placées dans l'espace de noms *classesMetier*.



Implémentation en C#

Prenons le cas de la classe *Categorie* pour illustrer comment passer d'une table de la base de données a une classe en C#.

```
class Categorie
{
    protected int idcategorie;
    protected string nom { set; get; }
    protected string souscategorie { set; get; }

    public int Idcategorie
    {
        set { this.idcategorie = value; }
        get { return this.idcategorie; }
    }
    public string Nom
    {
        set { this.nom = value; }
        get { return this.nom; }
    }
    public string Souscategorie
    {
        set { this.souscategorie = value; }
        get { return this.souscategorie; }
    }


    public Categorie()
    {

    }

    public Categorie(int idcategorie)
    {
        this.idcategorie = idcategorie;

    }
    public Categorie(int idcategorie, string nom, string souscategorie)
    {
        this.idcategorie = idcategorie;
        this.nom = nom;
        this.souscategorie = souscategorie;
    }


    public Categorie(Categorie categorie)
    {
        this.idcategorie = categorie.idcategorie;
        this.nom = categorie.nom;
        this.souscategorie = categorie.souscategorie;
    }
}
```

Si une colonne de la table *Categorie* devient un attribut de la classe *Categorie*. Le type de données de cette colonne de la table doit être similaire à celui de l'attribut correspondant dans la classe.

On ajoute trois constructeurs:

- ***Categorie()*** : constructeur sans paramètre
- ***Categorie(Categorie categorie)*** : constructeur avec un paramètre de type *Categorie*, il copie l'état de cet objet dans l'objet en cours de création
- ***Categorie(int idcategorie, string nom, string souscategorie)*** : constructeur qui assigne une valeur à chaque attribut de l'objet en cours de création

On ajoute un constructeur supplémentaire car *livre* utilise cette classe et *idCategorie* sera éventuellement le seul champ spécifié

- ***Categorie (int idCategorie)***

Correspondances entre les types de données en PostgreSQL et les types de données en C#

Type postgreSQL	Type c#
Bigint	Int64
Boolean	Boolean
Box, Circle, Line, LSeg, Path, Point, Polygon	Object
Bytea	Byte[]
Date	DateTime
Double	Double
Integer	Int32
Numeric	Decimal
Real	Single
Smallint	Int16
Text	String
Time	DateTime
Timestamp	DateTime
Varchar	String
Array	Array
type composite	struct, objet

Classe Livre

Un livre appartient à une catégorie => attribut *categorie* (de type *Categorie*) dans la classe *Livre*

```
class Livre
{
    protected string isbn ;
    protected string titre;
    protected string auteur;
    protected string edition;
    public Categorie categorie {set; get;}

    public string Isbn
    {
        set {this.isbn = value; }
        get { return this.isbn; }
    }
    public string Titre
    {
        set
        {
            if (value.Length > 0)
                this.titre = value;
            else
                throw new Exception
                    ("Le titre doit au moins contenir un caractère");
        }
        get { return this.titre; }
    }
    public string Auteur
    {
        set
        {
            if (value.Length > 0)
                this.auteur = value;
            else
                throw new Exception
                    ("L'auteur doit au moins contenir un caractère");
        }
        get { return this.auteur; }
    }
    public string Edition
    {
        set
        {
            if (value.Length > 0)
                this.edition = value;
            else
                throw new Exception
                    ("L'édition doit au moins contenir un caractère");
        }
        get { return this.edition; }
    }
```

```
    public Livre()
    {
        this.categorie = new Categorie();
    }
    public Livre(string isbn)

    {

        this.isbn = isbn;

    }
    public Livre(int idCategorie, string isbn, string titre,
                 string auteur,  string edition)

    {
        this.isbn = isbn;

        this.titre = titre;
        this.auteur = auteur;
        this.edition = edition;
        this.categorie = new Categorie(idCategorie);

    }


    public Livre (Livre livre)
    {
        this.isbn = livre.isbn;
        this.titre = livre.titre;
        this.auteur = livre.auteur;
        this.edition = livre.edition;
        this.categorie = new Categorie(livre.categorie);

    }

}
```

Classe Exemplaire

```
class Exemplaire :Livre
{
    private int idLivre;
    public int IdLivre
    {
        set { this.idLivre = value; }
        get { return this.idLivre; }
    }

    public Exemplaire() : base()
    { }

    public Exemplaire (Exemplaire exemplaire):base(exemplaire.Isbn)

    {
        this.idLivre = exemplaire.idLivre;

    }

    public Exemplaire(string isbn, int idLivre):base(isbn)
    {
        this.idLivre = idLivre;
     }


}
```

Classe Personne

```
class Personne
{
    private int idPersonne;
    private string nom;
    private string prenom;
    private string gsm;

    public int IdPersonne
    {
        set { this.idPersonne = value; }
        get { return this.idPersonne; }
    }
    public string Nom
    {
        set
        {
            if (value.Length > 2)
                this.nom = value;
            else
                throw new Exception
                    ("Le nom doit contenir au moins deux caractère");
        }
        get { return this.nom; }
    }
    public string Prenom
    {
        set
        {
            if (value.Length > 2)
                this.prenom = value;
            else
                throw new Exception
                    ("Le prénom doit contenir au moins deux caractère");
        }
        get { return this.prenom; }
    }
    public string GSM
    {
        set { this.gsm = value; }
        get { return this.gsm; }
    }
    public Adresse adresse { set; get; }
    public Personne()
    { }
    public Personne(int idPersonne, string nom, string prenom, string gsm,
        Adresse adresse)
    {
        this.idPersonne = idPersonne;
        this.nom = nom;
        this.prenom = prenom;
        this.gsm = gsm;
        this.adresse = adresse;
    }

    public Personne(Personne personne)
    {
        this.idPersonne = personne.idPersonne;
        this.nom = personne.nom;
        this.prenom = personne.prenom;
        this.gsm = personne.gsm;
        this.adresse = personne.adresse;
    }
}
```

Classe Emprunt

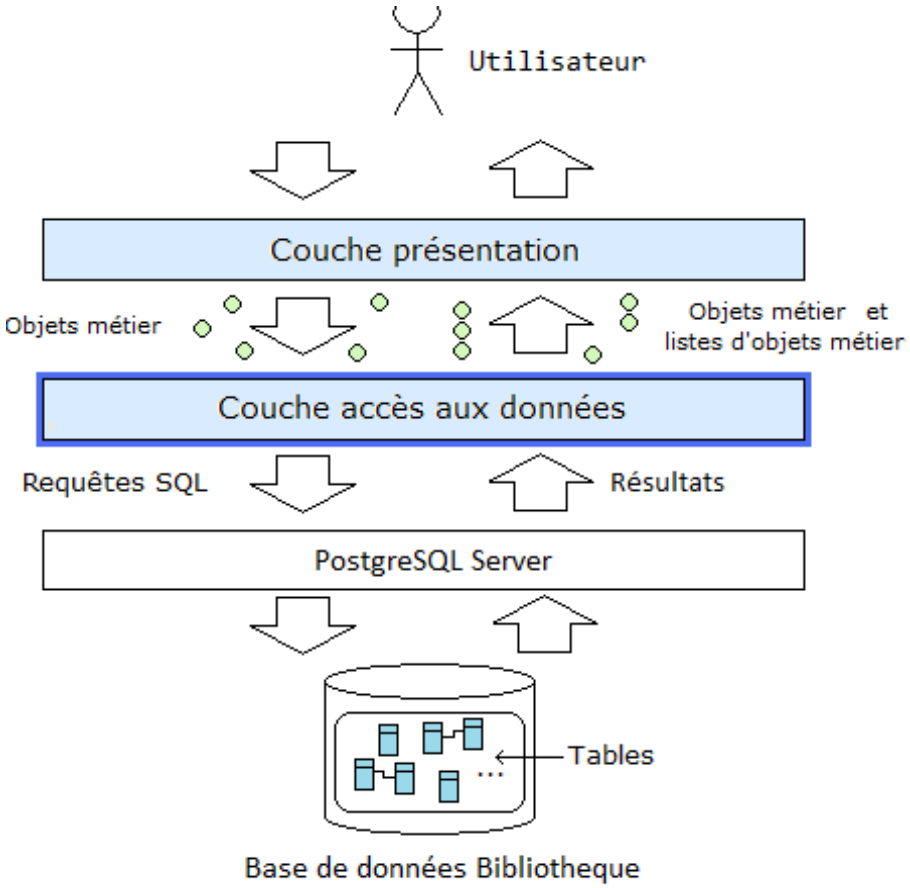
Emprunt comprend une personne et un exemplaire (composant faible – agrégation)

```
class Emprunt
{
    public Personne emprunteur;
    public Exemplaire livreEmprunte;
    public DateTime date_sortie { set; get; }
    public DateTime date_prolongation { set; get; }
    public DateTime date_retour { set; get; }
    private static int duree = 28;


    public Emprunt()
    {
        this.emprunteur = new Personne();
        this.livreEmprunte = new Exemplaire();
    }

    public Emprunt(int idPersonne, int idLivre, DateTime date_sortie,
        DateTime date_prolongation, DateTime date_retour)
    {
        this.emprunteur = new Personne();
        this.livreEmprunte = new Exemplaire();
        this.emprunteur.IdPersonne = idPersonne;
        this.livreEmprunte.IdLivre = idLivre;
        this.date_sortie = date_sortie;
        this.date_prolongation = date_prolongation;
        this.date_retour = date_retour;
    }

    public Emprunt(Emprunt emprunt)
    {
        this.emprunteur = new Personne();
        this.livreEmprunte = new Exemplaire();
        this.emprunteur.IdPersonne = emprunt.emprunteur.IdPersonne;
        this.livreEmprunte.IdLivre = emprunt.livreEmprunte.IdLivre;
        this.date_sortie = emprunt.date_sortie;
        this.date_prolongation = emprunt.date_prolongation;
        this.date_retour = emprunt.date_retour;
    }

      public DateTime calculDateRetourMax()
      {
          if (date_prolongation.Ticks != 0)
              return this.date_prolongation.AddDays(duree);
          else
              return this.date_sortie.AddDays(duree);
      }
  }
}
```

Classe Adresse – composant composite

```
class Adresse
{
    public string rue { get; set; }
    public string cp { get; set; }
    public string localite { get; set; }

    public Adresse()
    {

    }
    public Adresse(string rue, string codePostal, string localite)
    {
        this.rue = rue;
        this.cp = codePostal;
        this.localite = localite;
    }
    public Adresse(Adresse adresse)
    {
        this.rue = adresse.rue;
        this.cp = adresse.cp;
        this.localite = adresse.localite;

    }

}
```

Chapitre 8 : Création de la couche accès aux données



Concrètement, la couche accès aux données consiste en les classes qui vont :

1. Créer et stocker la connexion avec la base de données.

2. Transmettre les requêtes SQL a PostgreSQL Server.

3. Transformer toute entité stockée dans la base de données, c'est-à-dire toute ligne d'une table, en un objet métier et réciproquement.

Par exemple, dans la figure ci-dessous la couche accès aux données convertit une ligne de la table *Client* en un objet métier qui est une instance de la classe métier *Client*. Cet objet métier stocke les données de la ligne provenant de la table *Client*.

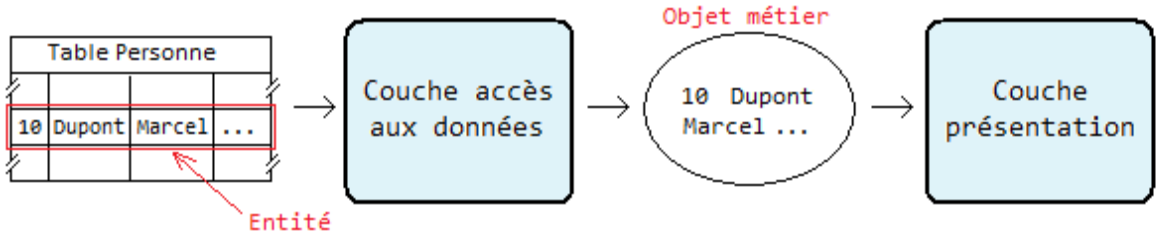
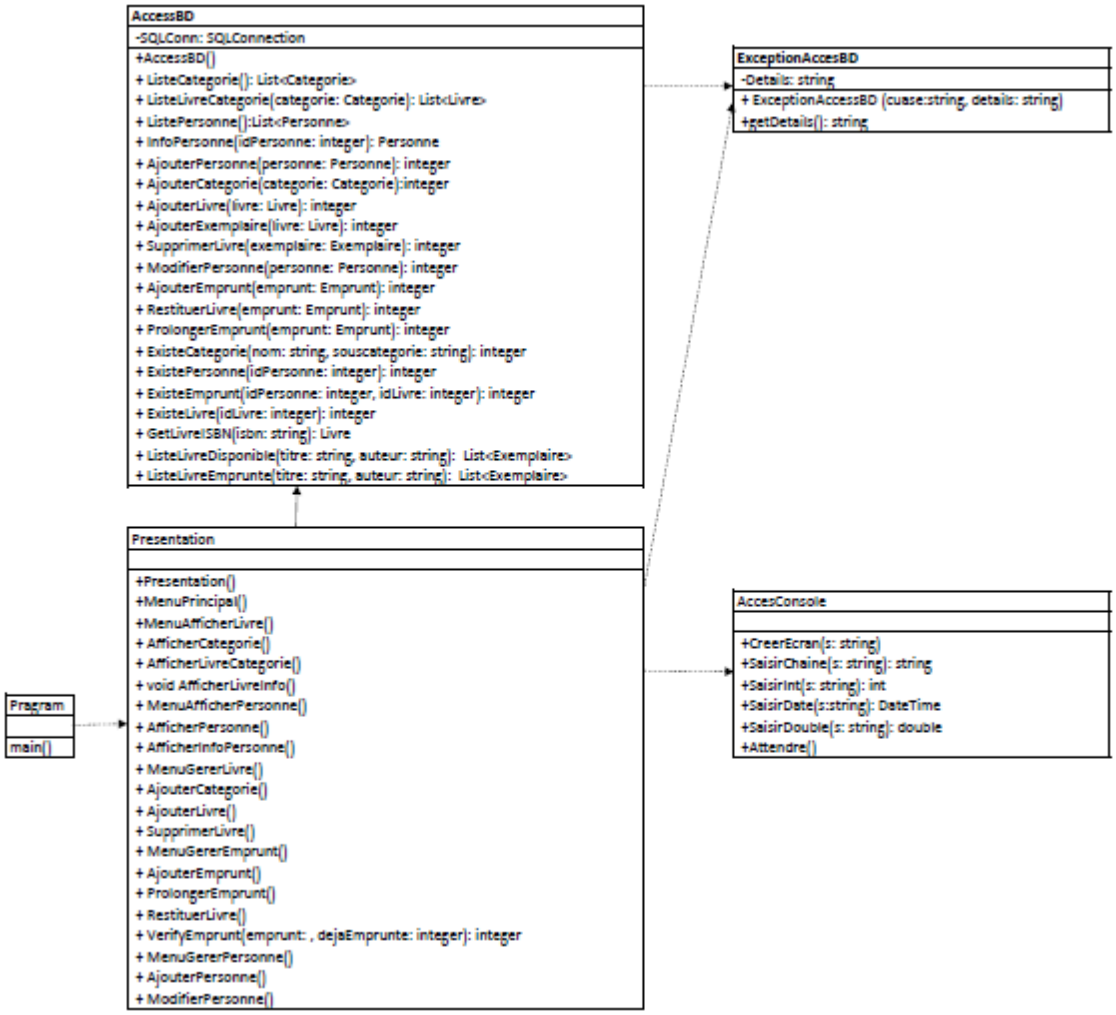


Diagramme de classes

Les classes *AccesBD* et *ExceptionAccesBD* se trouvent dans l'espace de noms *coucheAccesBD.*



La classe AccesBD

```
class AccesBD
{
    private NpgsqlConnection SqlConn;
    /**
    * Constructeur : créer une instance de la classe AccesBD

    public AccesBD()
    {
        try
        {

            this.SqlConn = new NpgsqlConnection("Server=localhost;" +
            "port=5432;" +
            "Database=Bibliotheque;" +
            "UserID=postgres;" +
            "Password = postgreSQL");


            this.SqlConn.Open();

        }
        catch (Exception e)
        {
            throw new ExceptionAccesBD("Connexion à la BD", e.Message);
        }
    }
```

```
    // Obtenir la liste des catégories

    public List<Categorie> ListeCategorie()
    {
        List<Categorie> liste = null;
        NpgsqlCommand sqlCmd = null;
        try
        {
            sqlCmd = new NpgsqlCommand("select idCategorie, nom, " +
                                       "sousCategorie from CATEGORIE " +
                                       "order by nom, sousCategorie", this.SqlConn);

            NpgsqlDataReader sqlreader = sqlCmd.ExecuteReader();

            if (sqlreader.Read())
            {
                liste = new List<Categorie>();

                do
                {
                    liste.Add(new Categorie(Convert.ToInt32(sqlreader["idCategorie"]),
                                            Convert.ToString(sqlreader["nom"]),
                                            Convert.ToString(sqlreader["sousCategorie"])));
                } while (sqlreader.Read());
            }
            sqlreader.Close();
        }
        catch (Exception e)
        {
            throw new ExceptionAccesBD(sqlCmd.CommandText, e.Message);
        }
        return liste;
    }
```

```
//Obtenir la liste des livres appartenant à une catégorie
//
// entrée : idCategorie (catégorie.idCategorie)
// sortie la liste des exemplaires ordonné par auteur et titre

public List<Livre> ListeLivreCategorie(Categorie categorie)
{
    List<Livre> liste = null;
    NpgsqlCommand sqlCmd = null;

    try
    {
        sqlCmd = new NpgsqlCommand("select  LIVRE.isbn, titre, auteur, " +
                                   " edition from LIVRE " +
                                   " where idCategorie =  :idCategorie" +
                                   " order by auteur, titre",

                                   this.SqlConn);

        // Ajoute les paramètres

        sqlCmd.Parameters.Add(new NpgsqlParameter("idCategorie",
            NpgsqlTypes.NpgsqlDbType.Integer));

        // Prepare la commande

        sqlCmd.Prepare();

        // Ajouter les valeurs aux paramètres

        sqlCmd.Parameters[0].Value = categorie.Idcategorie;

        NpgsqlDataReader sqlreader = sqlCmd.ExecuteReader();

        if (sqlreader.Read())
        {
            liste = new List<Livre>();

            do
            {
                liste.Add(new Livre(categorie.Idcategorie,
                    Convert.ToString(sqlreader["isbn"]),
                    Convert.ToString(sqlreader["titre"]),
                    Convert.ToString(sqlreader["auteur"]),
                    Convert.ToString(sqlreader["edition"])));
            } while (sqlreader.Read());
        }
        sqlreader.Close();

        return liste;
    }

    catch (Exception e)
    {
        throw new ExceptionAccesBD(sqlCmd.CommandText, e.Message);
    }
}
```

```
// Obtenir les informations d'une personne
//
// entrée: identifiant de la personne
// sortie: entité de la classe personne
public Personne InfoPersonne(int idpersonne)
{
    NpgsqlCommand sqlCmd = null;
    Personne personne = null;

    try
    {
        sqlCmd = new NpgsqlCommand("select nom, prenom, GSM, " +
                                   "(adressePersonne).rue as rue, " +
                                   "(adressePersonne).localite as localite, " +
                                   "(adressePersonne).cp as cp " +
                                   " from PERSONNE " +
                                   " where idPersonne =  :idPersonne",
                this.SqlConn);

        // Ajoute les paramètres

        sqlCmd.Parameters.Add(new NpgsqlParameter("idPersonne",
            NpgsqlTypes.NpgsqlDbType.Integer));


        // Prepare la commande

        sqlCmd.Prepare();

        // Ajouter les valeurs aux paramètres

        sqlCmd.Parameters[0].Value = idpersonne;

        NpgsqlDataReader sqlreader = sqlCmd.ExecuteReader();

          if (sqlreader.Read())
          {
              personne = new Personne(idpersonne,
                  Convert.ToString(sqlreader["nom"]),
                  Convert.ToString(sqlreader["prenom"]),
                  Convert.ToString(sqlreader["gsm"]),
                  new Adresse(Convert.ToString(sqlreader["rue"]),
                  Convert.ToString(sqlreader["cp"]),
                  Convert.ToString(sqlreader["localite"])));
          }
          sqlreader.Close();
          return personne;
      }
      catch (Exception e)
      {
          throw new ExceptionAccesBD(sqlCmd.CommandText, e.Message);
      }
  }
```

```
// Ajout d'une catégorie

public int AjouterCategorie(Categorie categorie)
{
    NpgsqlCommand sqlCmd = null;
    try
    {
        sqlCmd = new NpgsqlCommand("insert into " +
            "CATEGORIE(nom, sousCategorie) values " +
            "( :nom, :sousCategorie)", this.SqlConn);

        // Ajoute les paramètres

        sqlCmd.Parameters.Add(new NpgsqlParameter("nom",
            NpgsqlTypes.NpgsqlDbType.Varchar));
        sqlCmd.Parameters.Add(new NpgsqlParameter("sousCategorie",
            NpgsqlTypes.NpgsqlDbType.Varchar));

        // Prepare la commande

        sqlCmd.Prepare();

        // Ajouter les valeurs aux paramètres

        sqlCmd.Parameters[0].Value = categorie.Nom;
        sqlCmd.Parameters[1].Value = categorie.Souscategorie;

        // Execute la commande

        return (sqlCmd.ExecuteNonQuery());
    }
    catch (Exception e)
    {
        throw new ExceptionAccesBD(sqlCmd.CommandText, e.Message);
    }
}
```

```
public Livre GetLivreISBN(string isbn)
{
    NpgsqlCommand sqlCmd = null;
    Livre livre = null;

    try
    {
        sqlCmd = new NpgsqlCommand("select auteur, titre, edition, idCategorie " +
                                   " from LIVRE " +
                                   " where isbn = :isbn ",
                                   this.SqlConn);

        // Ajoute les paramètres

        sqlCmd.Parameters.Add(new NpgsqlParameter("isbn",
            NpgsqlTypes.NpgsqlDbType.Varchar));

        // Prepare la commande

        sqlCmd.Prepare();

        // Ajouter les valeurs aux paramètres

        sqlCmd.Parameters[0].Value = isbn;
          NpgsqlDataReader sqlreader = sqlCmd.ExecuteReader();

          if (sqlreader.Read())
          {
              livre = new Livre();
              livre.Isbn = isbn;
              livre.Auteur = Convert.ToString(sqlreader["auteur"]);
              livre.Titre = Convert.ToString(sqlreader["titre"]);
              livre.Edition = Convert.ToString(sqlreader["edition"]);

              livre.categorie.Idcategorie =
                  Convert.ToInt32(sqlreader["idCategorie"]);
          }

          sqlreader.Close();
          return livre;
      }
      catch (Exception e)
      {
          throw new ExceptionAccesBD(sqlCmd.CommandText, e.Message);
      }
}
```

```
// Ajouter un livre et le premier exemplaire

public int AjouterLivre(Livre livre)
{
    int result = 0;

    NpgsqlCommand sqlCmd = null;

    try
    {
        sqlCmd = new NpgsqlCommand("insert into " +
            "LIVRE(idCategorie, titre, auteur,  " +
            "edition, isbn) values " +
            "(:idCategorie, :titre, :auteur,  " +
            ":edition, :isbn)", this.SqlConn);

        // Ajoute les paramètres

        sqlCmd.Parameters.Add(new NpgsqlParameter("idCategorie",
            NpgsqlTypes.NpgsqlDbType.Integer));
        sqlCmd.Parameters.Add(new NpgsqlParameter("titre",
            NpgsqlTypes.NpgsqlDbType.Varchar));
        sqlCmd.Parameters.Add(new NpgsqlParameter("auteur",
            NpgsqlTypes.NpgsqlDbType.Varchar));
        sqlCmd.Parameters.Add(new NpgsqlParameter("edition",
            NpgsqlTypes.NpgsqlDbType.Varchar));
        sqlCmd.Parameters.Add(new NpgsqlParameter("isbn",
            NpgsqlTypes.NpgsqlDbType.Varchar));

        // Prepare la commande

        sqlCmd.Prepare();

        // Ajouter les valeurs aux paramètres

        sqlCmd.Parameters[0].Value = livre.categorie.Idcategorie;
        sqlCmd.Parameters[1].Value = livre.Titre;
        sqlCmd.Parameters[2].Value = livre.Auteur;
        sqlCmd.Parameters[3].Value = livre.Edition;
        sqlCmd.Parameters[4].Value = livre.Isbn;

        // Execute la commande

        result = sqlCmd.ExecuteNonQuery();

        if (result != 0)
            result = AjouterExemplaire(livre);

        return result;
    }
    catch (Exception e)
    {
        throw new ExceptionAccesBD(sqlCmd.CommandText, e.Message);
    }
}
```

```
// Ajouter un exemplaire et retourne le numéro de l'exemplaire

public int AjouterExemplaire(Livre livre)
{
    NpgsqlCommand sqlCmd = null;
    int idExemplaire = 0;
    try
    {
        sqlCmd = new NpgsqlCommand("insert into " +
            "EXEMPLAIRE(ISBN) values " +
            "(:isbn) returning idLivre", this.SqlConn);

        // Ajoute les paramètres

        sqlCmd.Parameters.Add(new NpgsqlParameter("isbn",
            NpgsqlTypes.NpgsqlDbType.Varchar));

        // Prepare la commande

        sqlCmd.Prepare();

        // Ajouter les valeurs aux paramètres

        sqlCmd.Parameters[0].Value = livre.Isbn;

        // Execute la commande

        NpgsqlDataReader sqlreader = sqlCmd.ExecuteReader();
        if (sqlreader.Read())
        {
            idExemplaire = Convert.ToInt32(sqlreader["idLivre"]);
        }

        sqlreader.Close();
        return idExemplaire;
    }
    catch (Exception e)
    {
        throw new ExceptionAccesBD(sqlCmd.CommandText, e.Message);
    }
}
```

```
// Supprimer un exemplaire,
// si dernier exemplaire, le supprimer également dans la table LIVRE


public int SupprimerLivre(Exemplaire exemplaire)
{
    NpgsqlCommand sqlCmd1 = null;
    NpgsqlCommand sqlCmd2 = null;
    NpgsqlTransaction sqltr = null;
    NpgsqlCommand sqlCmdCount = null;
    int nbr;
    string isbn;
    try
    {
        sqlCmdCount = new NpgsqlCommand("select count(*) as nbr, isbn " +
            " from exemplaire  " +
            " where isbn = " +
            "(select isbn from exemplaire where idLivre = :idLivre) " +
            "group by isbn ",
            this.SqlConn);

        // Ajout du paramètre

        sqlCmdCount.Parameters.Add(new NpgsqlParameter("idLivre",
            NpgsqlTypes.NpgsqlDbType.Integer));

        // Prepare la commande

        sqlCmdCount.Prepare();

        // Ajouter les valeurs aux paramètres
        sqlCmdCount.Parameters[0].Value = exemplaire.IdLivre;

        NpgsqlDataReader sqlreaderCount = sqlCmdCount.ExecuteReader();

        if (sqlreaderCount.Read())
        {
            nbr = Convert.ToInt32(sqlreaderCount["nbr"]);
            isbn = Convert.ToString(sqlreaderCount["isbn"]);
            sqlreaderCount.Close();

            sqlCmd1 = new NpgsqlCommand("delete from EXEMPLAIRE  " +
                "where idLivre = :idLivre ", this.SqlConn);

            // Ajoute les paramètres

            sqlCmd1.Parameters.Add(new NpgsqlParameter("idLivre",
                NpgsqlTypes.NpgsqlDbType.Integer));

            // Prepare la commande

            sqlCmd1.Prepare();
            // Ajouter les valeurs aux paramètres

            sqlCmd1.Parameters[0].Value = exemplaire.IdLivre;

            sqltr = this.SqlConn.BeginTransaction();

            sqlCmd1.Transaction = sqltr;
```

```
                sqlCmd1.ExecuteNonQuery();

                if (nbr == 1)
                {
                    sqlCmd2 = new NpgsqlCommand("delete from LIVRE  " +
                    "where isbn = :isbn ", this.SqlConn);

                    // Ajoute les paramètres

                    sqlCmd2.Parameters.Add(new NpgsqlParameter("isbn",
                        NpgsqlTypes.NpgsqlDbType.Varchar));

                    // Prepare la commande

                    sqlCmd2.Prepare();

                    // Ajouter les valeurs aux paramètres

                    sqlCmd2.Parameters[0].Value = isbn;

                    sqlCmd2.Transaction = sqltr;

                    sqlCmd2.ExecuteNonQuery();
                }

                // Termine la transaction

                sqltr.Commit();

                return (1);
            }
            else
            {
                sqlreaderCount.Close();
                return 0;
            }
        }

        catch (Exception e)
        {
            sqltr.Rollback();
            throw new ExceptionAccesBD(sqlCmd1.CommandText, e.Message);
        }
    }
```

```
    // ajouter une personne et retourne l'identifiant de la personne
public int AjouterPersonne(Personne personne)
{
    NpgsqlCommand sqlCmd = null;
    int idPersonne = 0;
    try
    {
        sqlCmd = new NpgsqlCommand("insert into " +
            "PERSONNE(nom, prenom, GSM, adressePersonne) " +
            " values " +
            "( :nom, :prenom, :gsm, (:rue, :cp, :localite))"+
            " returning idPersonne",
            this.SqlConn);

        // Ajoute les paramètres

        sqlCmd.Parameters.Add(new NpgsqlParameter("nom",
            NpgsqlTypes.NpgsqlDbType.Varchar));
        sqlCmd.Parameters.Add(new NpgsqlParameter("prenom",
            NpgsqlTypes.NpgsqlDbType.Varchar));
        sqlCmd.Parameters.Add(new NpgsqlParameter("gsm",
            NpgsqlTypes.NpgsqlDbType.Varchar));
        sqlCmd.Parameters.Add(new NpgsqlParameter("rue",
            NpgsqlTypes.NpgsqlDbType.Varchar));
        sqlCmd.Parameters.Add(new NpgsqlParameter("cp",
            NpgsqlTypes.NpgsqlDbType.Varchar));
        sqlCmd.Parameters.Add(new NpgsqlParameter("localite",
            NpgsqlTypes.NpgsqlDbType.Varchar));

        // Prepare la commande

        sqlCmd.Prepare();
```

```
            // Ajouter les valeurs aux paramètres

            sqlCmd.Parameters[0].Value = personne.Nom;
            sqlCmd.Parameters[1].Value = personne.Prenom;
            sqlCmd.Parameters[2].Value = personne.GSM;
            sqlCmd.Parameters[3].Value = personne.adresse.rue;
            sqlCmd.Parameters[4].Value = personne.adresse.cp;
            sqlCmd.Parameters[5].Value = personne.adresse.localite;

            // Execute la commande

            NpgsqlDataReader sqlreader = sqlCmd.ExecuteReader();


            if (sqlreader.Read())
            {
                idPersonne = Convert.ToInt32(sqlreader["idPersonne"]);
            }

            sqlreader.Close();
            return idPersonne;
        }
        catch (Exception e)
        {
            throw new ExceptionAccesBD(sqlCmd.CommandText, e.Message);
        }
    }
```

```
// modifier les coordonnées d'une personne

public int ModifierPersonne(Personne personne)
{
    NpgsqlCommand sqlCmd = null;
    try
    {
        sqlCmd = new NpgsqlCommand("update PERSONNE " +
            "set nom = :nom, prenom = :prenom, gsm = :gsm, " +
            " adressePersonne.rue = :rue," +
            " adressePersonne.cp = :cp," +
            " adressePersonne.localite = :localite " +
            "where idPersonne = :idPersonne",
            this.SqlConn);

        // Ajoute les paramètres

        sqlCmd.Parameters.Add(new NpgsqlParameter("nom",
            NpgsqlTypes.NpgsqlDbType.Varchar));
        sqlCmd.Parameters.Add(new NpgsqlParameter("prenom",
           NpgsqlTypes.NpgsqlDbType.Varchar));
        sqlCmd.Parameters.Add(new NpgsqlParameter("gsm",
           NpgsqlTypes.NpgsqlDbType.Varchar));
        sqlCmd.Parameters.Add(new NpgsqlParameter("rue",
           NpgsqlTypes.NpgsqlDbType.Varchar));
        sqlCmd.Parameters.Add(new NpgsqlParameter("cp",
           NpgsqlTypes.NpgsqlDbType.Varchar));
        sqlCmd.Parameters.Add(new NpgsqlParameter("localite",
           NpgsqlTypes.NpgsqlDbType.Varchar));
        sqlCmd.Parameters.Add(new NpgsqlParameter("idPersonne",
           NpgsqlTypes.NpgsqlDbType.Integer));

        // Prepare la commande

        sqlCmd.Prepare();

        // Ajouter les valeurs aux paramètres

        sqlCmd.Parameters[0].Value = personne.Nom;
        sqlCmd.Parameters[1].Value = personne.Prenom;
        sqlCmd.Parameters[2].Value = personne.GSM;
        sqlCmd.Parameters[3].Value = personne.adresse.rue;
        sqlCmd.Parameters[4].Value = personne.adresse.cp;
        sqlCmd.Parameters[5].Value = personne.adresse.localite;
        sqlCmd.Parameters[6].Value = personne.IdPersonne;

        // Execute la commande

        return (sqlCmd.ExecuteNonQuery());
    }
    catch (Exception e)
    {
        throw new ExceptionAccesBD(sqlCmd.CommandText, e.Message);
    }
}
```

```
// Retourne le nombre de livres emprunté par une personne donné

public int getNbreLivresEmpruntes(int idPersonne)
{
    int nombreLivresEmpruntes = 0;
    NpgsqlCommand sqlCmd = null;


    try
    {

        sqlCmd = new NpgsqlCommand("select count(*) as nombre " +
            " from EMPRUNT   " +
                                   " where ((date_retour is null) " +
                                   " and (idPersonne = :idPersonne)) ",

                                   this.SqlConn);

        // Ajoute les paramètres

        sqlCmd.Parameters.Add(new NpgsqlParameter("idPersonne",
            NpgsqlTypes.NpgsqlDbType.Integer));

        // Prepare la commande

        sqlCmd.Prepare();

        // Ajouter les valeurs aux paramètres

        sqlCmd.Parameters[0].Value = idPersonne;

        NpgsqlDataReader sqlreader = sqlCmd.ExecuteReader();
        if (sqlreader.Read())
        {
            nombreLivresEmpruntes = (Convert.ToInt32(sqlreader["nombre"]));

        }

        sqlreader.Close();

        return nombreLivresEmpruntes;
    }

    catch (Exception e)
    {
        throw new ExceptionAccesBD(sqlCmd.CommandText, e.Message);
    }
}
```

```
//Restituer un livre

public int RestituerLivre(Emprunt emprunt)
{
    NpgsqlCommand sqlCmd = null;
    try
    {
        sqlCmd = new NpgsqlCommand("update EMPRUNT " +
            "set date_retour = current_date " +
            "where idPersonne = :idPersonne " +
            "and idLivre = :idLivre",
            this.SqlConn);

        // Ajoute les paramètres

        sqlCmd.Parameters.Add(new NpgsqlParameter("idPersonne",
            NpgsqlTypes.NpgsqlDbType.Integer));
        sqlCmd.Parameters.Add(new NpgsqlParameter("idLivre",
           NpgsqlTypes.NpgsqlDbType.Integer));

        // Prepare la commande

        sqlCmd.Prepare();

        // Ajouter les valeurs aux paramètres
        sqlCmd.Parameters[0].Value = emprunt.emprunteur.IdPersonne;
        sqlCmd.Parameters[1].Value = emprunt.livreEmprunte.IdLivre;

        // Execute la commande

        return (sqlCmd.ExecuteNonQuery());
    }
    catch (Exception e)
    {
        throw new ExceptionAccesBD(sqlCmd.CommandText, e.Message);
    }
}
```

```
// Faire en sorte qu'un livre soit emprunt par une personne

public int AjouterEmprunt(Emprunt emprunt)
{
    NpgsqlCommand sqlCmd = null;
    try
    {
        sqlCmd = new NpgsqlCommand("insert into " +
            "EMPRUNT(idLivre, idPersonne, date_sortie) " +
            "values " +
            "( :idLivre, :idPersonne, current_date) ",
             this.SqlConn);

        // Ajoute les paramètres

        sqlCmd.Parameters.Add(new NpgsqlParameter("idLivre",
            NpgsqlTypes.NpgsqlDbType.Integer));
        sqlCmd.Parameters.Add(new NpgsqlParameter("idPersonne",
            NpgsqlTypes.NpgsqlDbType.Integer));

        // Prepare la commande

        sqlCmd.Prepare();

        // Ajouter les valeurs aux paramètres

        sqlCmd.Parameters[0].Value = emprunt.livreEmprunte.IdLivre;
        sqlCmd.Parameters[1].Value = emprunt.emprunteur.IdPersonne;

        // Execute la commande

        return (sqlCmd.ExecuteNonQuery());
    }
    catch (Exception e)
    {
        throw new ExceptionAccesBD(sqlCmd.CommandText, e.Message);
    }
}
```

```
// Prolonger un emprunt

public int ProlongerEmprunt(Emprunt emprunt)
{
    NpgsqlCommand sqlCmd = null;
    try
    {
        sqlCmd = new NpgsqlCommand("update EMPRUNT " +
            "set date_prolongation = current_date " +
            "where idPersonne = :idPersonne " +
            "and idLivre = :idLivre",
            this.SqlConn);

        // Ajoute les paramètres

        sqlCmd.Parameters.Add(new NpgsqlParameter("idPersonne",
            NpgsqlTypes.NpgsqlDbType.Integer));
        sqlCmd.Parameters.Add(new NpgsqlParameter("idLivre",
           NpgsqlTypes.NpgsqlDbType.Integer));

        // Prepare la commande

        sqlCmd.Prepare();

        // Ajouter les valeurs aux paramètres

        sqlCmd.Parameters[0].Value = emprunt.emprunteur.IdPersonne;
        sqlCmd.Parameters[1].Value = emprunt.livreEmprunte.IdLivre;

        // Execute la commande

        return (sqlCmd.ExecuteNonQuery());
    }
    catch (Exception e)
    {
        throw new ExceptionAccesBD(sqlCmd.CommandText, e.Message);
    }
}
```

```
// Obtenir la liste des livres (exemplaires) non empruntés pour un titre donné et
// un auteur donné

public List<Exemplaire> ListeLivreDisponible(string titre, string auteur)
{
    List<Exemplaire> liste = null;
    NpgsqlCommand sqlCmd = null;

    try
    {

        sqlCmd = new NpgsqlCommand("select idLivre, LIVRE.isbn " +
            " from EXEMPLAIRE inner join LIVRE " +
            " on EXEMPLAIRE.isbn = LIVRE.isbn " +
                                    " where idLivre not in " +
                                    " (select EXEMPLAIRE.idLivre " +
                                "from (EXEMPLAIRE inner join EMPRUNT " +
                                "on EXEMPLAIRE.idLivre = EMPRUNT.idLivre) " +
                                " inner join LIVRE on livre.isbn = exemplaire.isbn " +
                                " where (date_retour is null)) " +
                                " and titre = :titre and auteur = :auteur",

                                this.SqlConn);

        // Ajoute les paramètres

        sqlCmd.Parameters.Add(new NpgsqlParameter("titre",
            NpgsqlTypes.NpgsqlDbType.Varchar));

        sqlCmd.Parameters.Add(new NpgsqlParameter("auteur",
            NpgsqlTypes.NpgsqlDbType.Varchar));
        // Prepare la commande
         sqlCmd.Prepare();

         // Ajouter les valeurs aux paramètres

         sqlCmd.Parameters[0].Value = titre;
         sqlCmd.Parameters[1].Value = auteur;

         NpgsqlDataReader sqlreader = sqlCmd.ExecuteReader();


         if (sqlreader.Read())
         {
             liste = new List<Exemplaire>();

             do
             {
                 liste.Add(new Exemplaire(Convert.ToString(sqlreader["isbn"]),
                     Convert.ToInt32(sqlreader["idLivre"])));
             } while (sqlreader.Read());
         }

         sqlreader.Close();

         return liste;
     }

     catch (Exception e)
     {
         throw new ExceptionAccesBD(sqlCmd.CommandText, e.Message);
     }
 }
```

```
// Obtenir la liste des livres empruntés (exemplaires) pour un titre donné et
// un auteur donné

public List<Exemplaire> ListeLivreEmprunte(string titre, string auteur)
{
    List<Exemplaire> liste = null;
    NpgsqlCommand sqlCmd = null;

    try
    {

        sqlCmd = new NpgsqlCommand("select livre.isbn as isbn, " +
                                   "EMPRUNT.idLivre as idLivre  " +
                                  "from (EXEMPLAIRE inner join EMPRUNT " +
                                  "on EXEMPLAIRE.idLivre = EMPRUNT.idLivre) " +
                                  " inner join LIVRE on livre.isbn = exemplaire.isbn " +
                                  " where date_retour is null and " +
                                  " titre = :titre and auteur = :auteur",

                                  this.SqlConn);

        // Ajoute les paramètres

        sqlCmd.Parameters.Add(new NpgsqlParameter("titre",
            NpgsqlTypes.NpgsqlDbType.Varchar));

        sqlCmd.Parameters.Add(new NpgsqlParameter("auteur",
            NpgsqlTypes.NpgsqlDbType.Varchar));
        // Prepare la commande

        sqlCmd.Prepare();

         // Ajouter les valeurs aux paramètres

         sqlCmd.Parameters[0].Value = titre;
         sqlCmd.Parameters[1].Value = auteur;

         NpgsqlDataReader sqlreader = sqlCmd.ExecuteReader();

         if (sqlreader.Read())
         {
             liste = new List<Exemplaire>();

             do
             {
                 liste.Add(new Exemplaire(Convert.ToString(sqlreader["isbn"]),
                     Convert.ToInt32(sqlreader["idLivre"])));
             } while (sqlreader.Read());
         }

         sqlreader.Close();

         return liste;
    }

    catch (Exception e)
    {
        throw new ExceptionAccesBD(sqlCmd.CommandText, e.Message);
    }
}
```

```
// Obtenir la liste des personnes

public List<Personne> ListePersonne()
{
    List<Personne> liste = null;
    NpgsqlCommand sqlCmd = null;

    try
    {
        sqlCmd = new NpgsqlCommand("select idPersonne, nom, " +
                                   "prenom, gsm," +
                                   " (adressePersonne).rue as rue, " +
                                    " (adressePersonne).cp as cp, " +
                                     " (adressePersonne).localite as localite " +
                                     " from PERSONNE " +
                                   "order by nom, prenom", this.SqlConn);

        NpgsqlDataReader sqlreader = sqlCmd.ExecuteReader();
        if (sqlreader.Read())
        {
            liste = new List<Personne>();

            do
            {
                liste.Add(new Personne(
                          Convert.ToInt32(sqlreader["idPersonne"]),
                          Convert.ToString(sqlreader["nom"]),
                          Convert.ToString(sqlreader["prenom"]),
                          Convert.ToString(sqlreader["gsm"]),
                          new Adresse(Convert.ToString(sqlreader["rue"]),
                             Convert.ToString(sqlreader["cp"]),
                             Convert.ToString(sqlreader["localite"]))));
            } while (sqlreader.Read());
        }

        sqlreader.Close();
    }

    catch (Exception e)
    {
        throw new ExceptionAccesBD(sqlCmd.CommandText, e.Message);
    }

    return liste;
}
```

```
// Verifie si une catégorie existe
//
// entrée : le nom de la catégorie et la sous-catégorie
//
// retourne 1 si la catégorie existe sinon retourne 0
public int ExisteCategorie(string nom, string souscategorie)
{
    NpgsqlCommand sqlCmd = null;
    int trouve = 0;
    try
    {
        sqlCmd = new NpgsqlCommand(
            "select * from CATEGORIE " +
            "where nom = :nom and sousCategorie = :souscategorie",
            this.SqlConn);

        // Ajoute les paramètres

        sqlCmd.Parameters.Add(new NpgsqlParameter("nom",
            NpgsqlTypes.NpgsqlDbType.Varchar));

        sqlCmd.Parameters.Add(new NpgsqlParameter("souscategorie",
            NpgsqlTypes.NpgsqlDbType.Varchar));

        // Prepare la commande

        sqlCmd.Prepare();

        // Ajouter les valeurs aux paramètres

        sqlCmd.Parameters[0].Value = nom;
        sqlCmd.Parameters[1].Value = souscategorie;

        NpgsqlDataReader sqlreader = sqlCmd.ExecuteReader();

        if (sqlreader.Read())
        {
            trouve = Convert.ToInt32(sqlreader[0]);
        }

        sqlreader.Close();

        return (trouve);
    }

    catch (Exception e)
    {

        throw new ExceptionAccesBD(sqlCmd.CommandText, e.Message);
    }
}
```

```
// Verifie si une personne existe
//
// entrée : l'identifiant de la personne
//
// retourne 1 si la personne existe sinon retourne 0
public int ExistePersonne(int idPersonne)
{
    NpgsqlCommand sqlCmd = null;
    int trouve = 0;
    try
    {
        sqlCmd = new NpgsqlCommand(
            "select * from PERSONNE " +
            "where idPersonne = :idPersonne",
            this.SqlConn);

        // Ajoute les paramètres

        sqlCmd.Parameters.Add(new NpgsqlParameter("idPersonne",
            NpgsqlTypes.NpgsqlDbType.Integer));

        // Prepare la commande

        sqlCmd.Prepare();

        // Ajouter les valeurs aux paramètres

        sqlCmd.Parameters[0].Value = idPersonne;

        NpgsqlDataReader sqlreader = sqlCmd.ExecuteReader();
        if (sqlreader.Read())
        {
            trouve = 1;
        }

        sqlreader.Close();

        return (trouve);
    }

    catch (Exception e)
    {
        throw new ExceptionAccesBD(sqlCmd.CommandText, e.Message);
    }
}
// vérifie si l'exemplaire existe et si c'est le cas si cet exemplaire a été emrunté par la personne ou prolongé
// sortiee
//      0: exemplaire n'existe pas
//      1: exemplaire disponible
//      2: exemplaire emprunté par une autre personne
//      3: exemplaire emprunté par cette personne
//      4: exemplaire qui a déjà été prolongé par cette personne
```

```
public int VerifyEmprunt(Emprunt emprunt)
{
    NpgsqlCommand sqlCmd = null;
    int trouve = 0;
    int idLivre = emprunt.livreEmprunte.IdLivre;
    int idPersonne = emprunt.emprunteur.IdPersonne;

    try
    {
        sqlCmd = new NpgsqlCommand(
            "select * from EXEMPLAIRE " +
            "where idLivre = :idLivre ",
            this.SqlConn);

        // Ajoute les paramètres

        sqlCmd.Parameters.Add(new NpgsqlParameter("idLivre",
            NpgsqlTypes.NpgsqlDbType.Integer));


        // Prepare la commande

        sqlCmd.Prepare();

        // Ajouter les valeurs aux paramètres

        sqlCmd.Parameters[0].Value = idLivre;

        NpgsqlDataReader sqlreader = sqlCmd.ExecuteReader();
```

```
        if (sqlreader.Read())
        {
            // exemplaire existe
            sqlreader.Close();

            sqlCmd = new NpgsqlCommand(
            "select idPersonne, date_prolongation, date_sortie from EMPRUNT " +
            "where idLivre = :idLivre and date_retour is null ",
            this.SqlConn);

            // Ajoute les paramètres

            sqlCmd.Parameters.Add(new NpgsqlParameter("idLivre",
                NpgsqlTypes.NpgsqlDbType.Integer));

            // Prepare la commande

            sqlCmd.Prepare();

            // Ajouter les valeurs aux paramètres

            sqlCmd.Parameters[0].Value = idLivre;

            NpgsqlDataReader sqlreaderE = sqlCmd.ExecuteReader();
```

```
                if (sqlreaderE.Read())
                {
                    if (Convert.ToInt32(sqlreaderE["idPersonne"]) == idPersonne)
                    {
                        // livre emprunté par cette personne

                        //if (sqlreaderE.IsDBNull(1))

                        if (sqlreaderE.IsDBNull(sqlreaderE.GetOrdinal("date_prolongation")))
                            // livre emprunté et non prolongé par cette personne
                            trouve = 3;
                        else
                            // livre emprunté et prolongé par cette personne
                            trouve = 4;
                    }
                    else
                        // livre emprunté par une autre personne
                        trouve = 2;
                }
                else
                    // livre disponible
                    trouve = 1;
                sqlreaderE.Close();
            }
            else
                sqlreader.Close();

            return (trouve);
        }
            catch (Exception e)
            {
                throw new ExceptionAccesBD(sqlCmd.CommandText, e.Message);
            }
        }

    }
}
```

Pour chaque fonctionnalité de notre application qui requiert un accès à la base de données, nous avons ajouté une méthode à la classe *AccesBD*.

Fonctionnalités	**Méthode de la classe AccesBD**
Obtenir la liste des catégories	ListeCategorie(): List<Categorie>
Obtenir la liste des livres appartenant à une catégorie	ListeLivreCategorie(categorie: Categorie): List<Livre>
Obtenir les informations sur les livres dont le titre et l'auteur sont donnés	ListeLivreDisponible(titre: string, auteur: string): List<Exemplaire> ListeLivreEmprunte(titre: string, auteur: string): List<Exemplaire>
Obtenir la liste des personnes	ListePersonne():List<Personne>
Obtenir les informations sur une personne	InfoPersonne(idPersonne: integer): Personne
Ajout d'une personne	AjouterPersonne(personne: Personne): integer
Ajout d'une catégorie	AjouterCategorie(categorie: Categorie):integer
Ajout d'un livre	AjouterLivre(livre: Livre): integer AjouterExemplaire(livre: Livre): integer
Supprimer un livre	SupprimerLivre(exemplaire: Exemplaire): integer
Modifier les coordonnées d'une personne	ModifierPersonne(personne: Personne): integer
Faire en sorte qu'un livre est emprunté par une personne	AjouterEmprunt(emprunt: Emprunt): integer
Faire en sorte qu'un livre est restitué	RestituerLivre(emprunt: Emprunt): integer
Faire en sorte qu'un livre est prolongé	ProlongerEmprunt(emprunt: Emprunt): integer

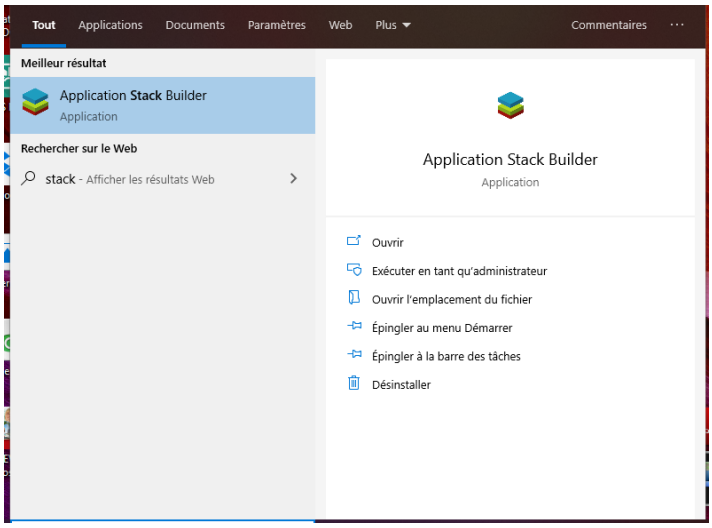
En plus de ces fonctionnalités, nous avons ajouté les méthodes suivantes :

- ExisteCategorie(nom: string, souscategorie: string): integer
  Vérifie si une catégorie existe
- ExistePersonne(idPersonne: integer): integer
  Vérifie si une personne avec un identifiant donné existe déjà
- VerifyEmprunt(emprunt: Emprunt): integer
  Vérifie si l'exemplaire existe, si exemplaire (n')est (pas) emprunté, (pas) prolongé par l'emprunteur
- GetLivreISBN(isbn: string): Livre
  Recherche les informations sur un livre dont l'ISBN est donné

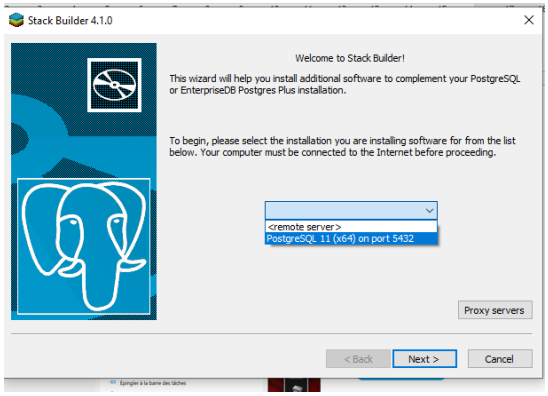
Les lignes using `Npgsql doit` figurer au début de ce fichier pour rendre accessibles les classes d'accès à une base de données PostgreSQL Server. Sans ces lignes, Visual C# indiquera ne pas connaitre les classes *NpgsqlConnection, NpgsqlCommand, ...*

Pour utiliser la librairie Npgsql.dll, il ajouter la référence au projet. L'installation de ce pilote se fait via l'application *Stack Builder*.

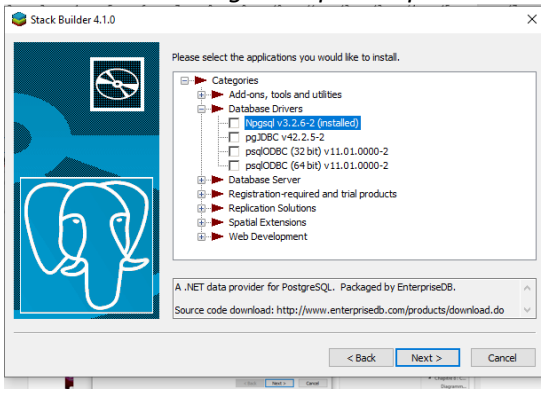
- Lancer *Stack Builder*

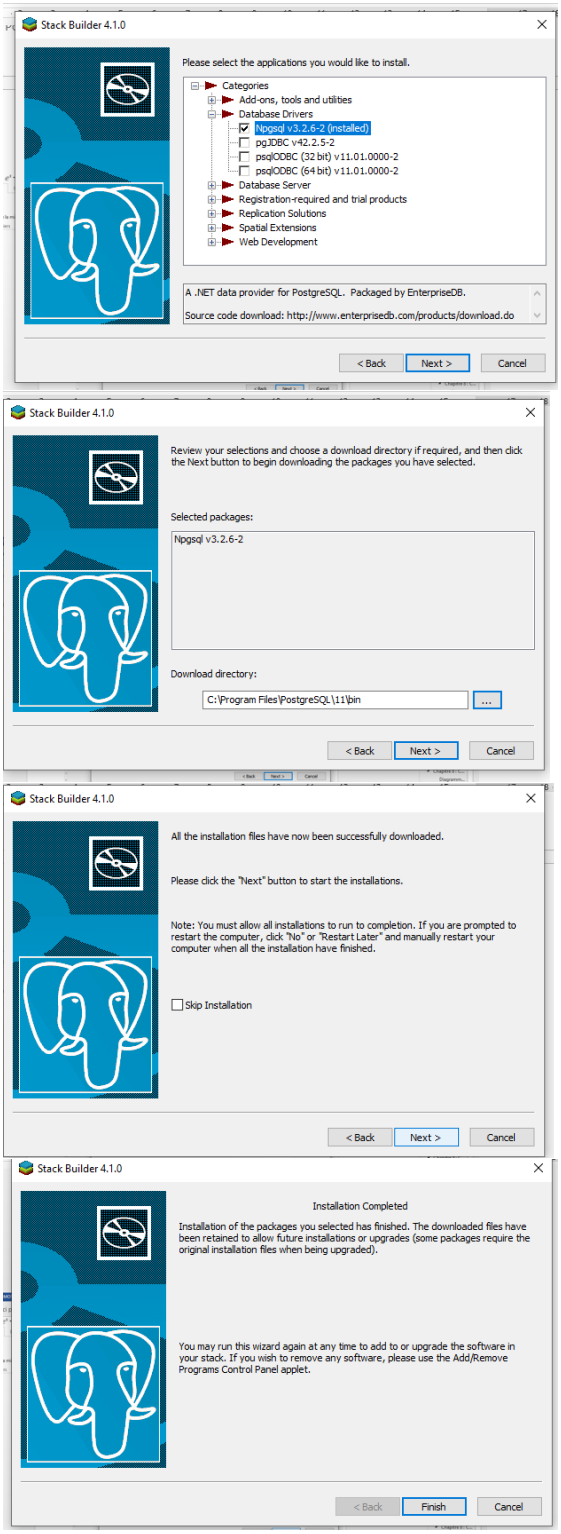


- *Démarrer stack builder*

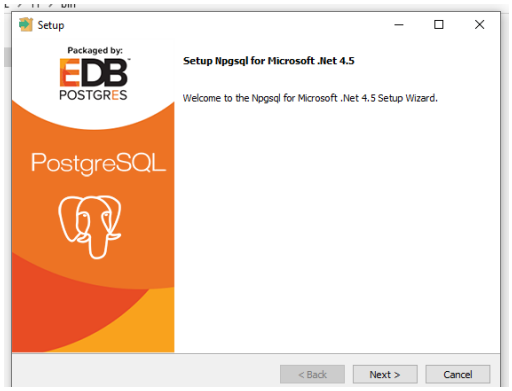


- *Choisir PosgreSQL puis cliquer sur next*

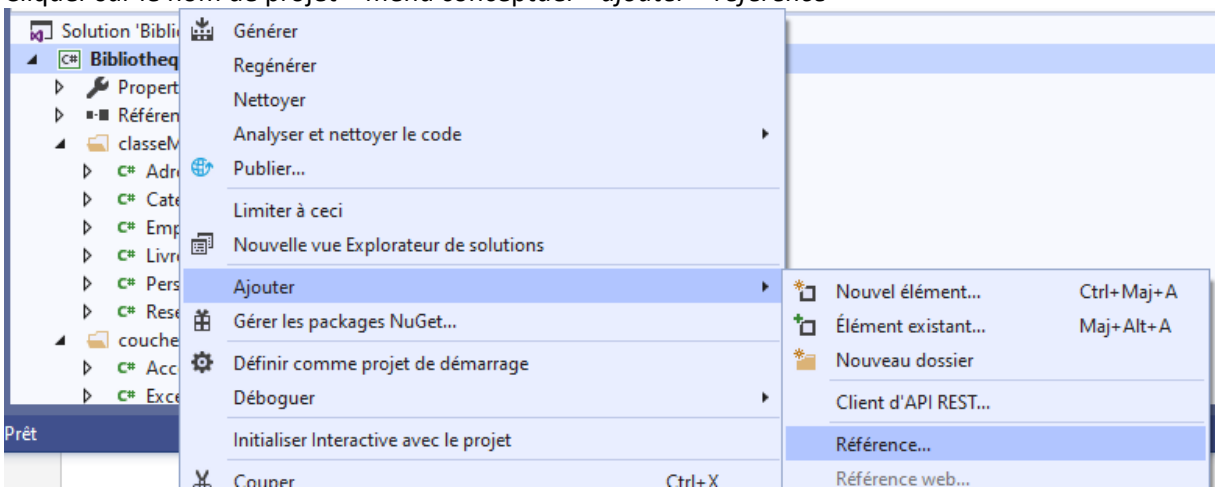


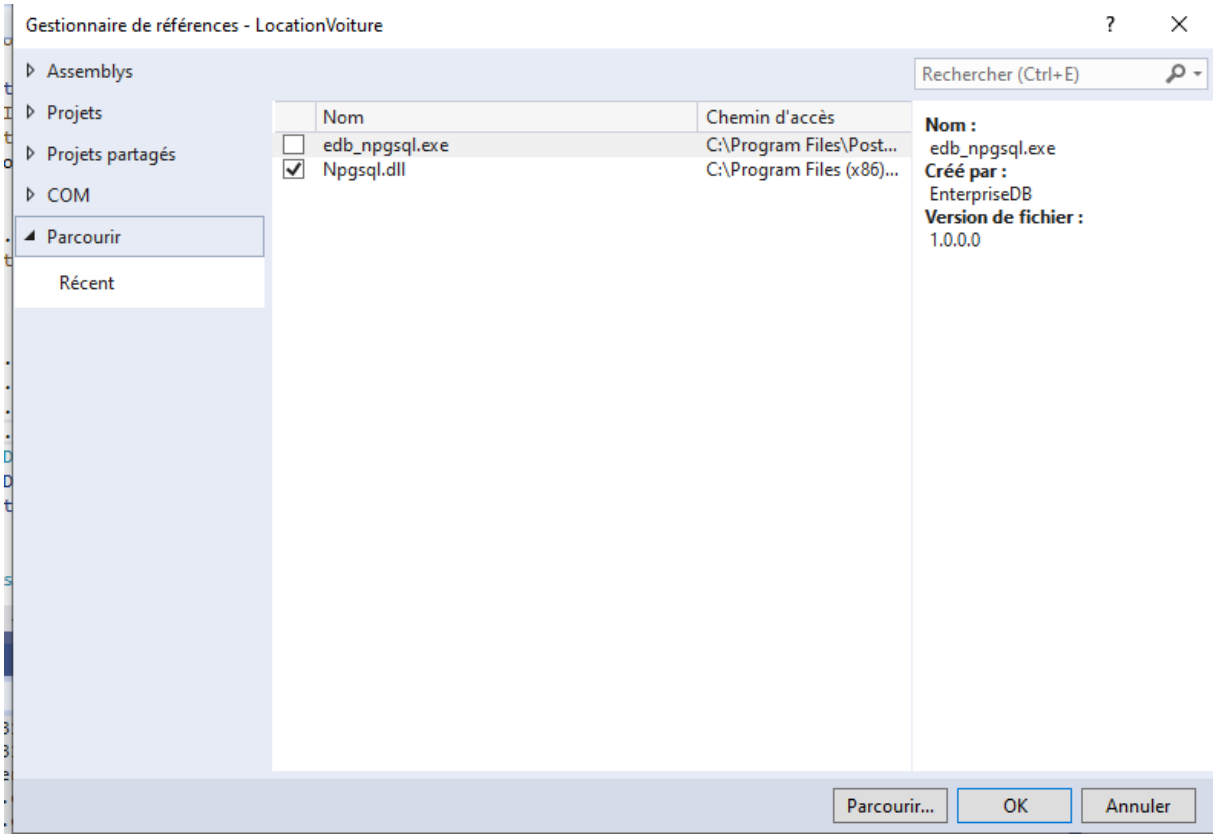


Executer ….exe



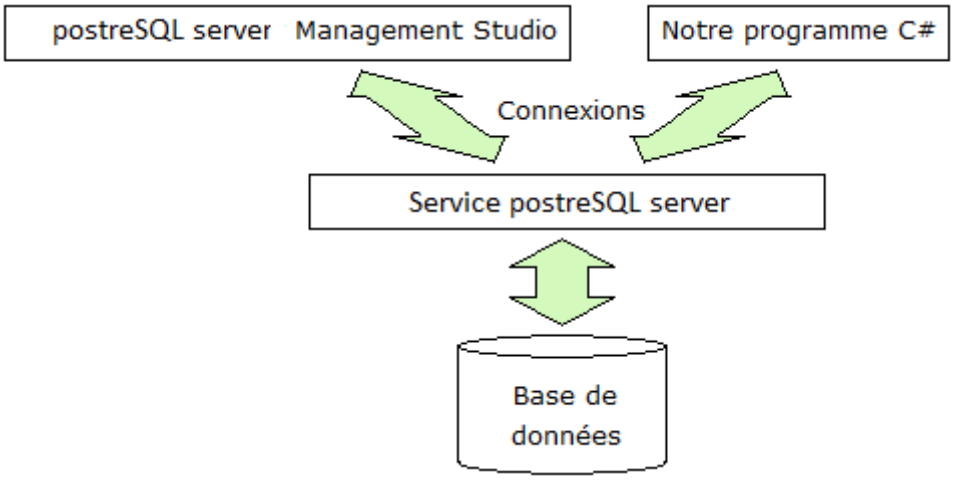
Dans c# pour se connecter à la base de donnée
Cliquer sur le nom de projet – menu conceptuel – *ajouter – référence*





Etablir une connexion avec la base de données

Pour accéder à une base de données, une application passe par le service postgreSQL server



Pour communiquer avec le service postgreSQL Server:

1. L'application établit une connexion avec le service SQL Server.
2. Une fois cette connexion établie, l'application peut transmettre ses requêtes SQL au service postgreSQL Server et recevoir en retour les informations de la base de données dont elle a besoin.
3. Le code suivant montre comment établir une connexion avec la base de données *Bibliotheque* :

```
using Npgsql;
```

```
public AccesBD()
{
    try
    {

        this.SqlConn = new NpgsqlConnection("Server=localhost;" +
        "port=5432;" +
        "Database=Bibliotheque;" +
        "UserID=postgres;" +
        "Password = postgreSQL");


        this.SqlConn.Open();
    }
    catch (Exception e)
    {
        throw new ExceptionAccesBD("Connexion à la BD", e.Message);
    }
}
```

*sqlConn**.open()*** : établit la connexion avec le service postreSQL server.

Requête SELECT

Constructeur de la classe ***NpgsqlCommand***

- ***NpgsqlCommand()*** : initialise une nouvelle instance de la classe ***NpgsqlCommand***
- ***NpgsqlCommand(string cmdText)*** : cmdText représente le texte de la requête
- ***NpgsqlCommand(string cmdText, NpgsqlConnection connection)*** :
  cmdText représente le texte de la requête
  connection : un objet NpgsqlConnection qui représente la connexion au postgreSQL server

NpgsqlDataReader *sqlReader =* ***sqlCmd.ExecuteReader()*** transmet à postgreSQL Server la requête et fournit en retour un objet de type **NpgsqlDataReader**. C'est via cet objet qu'on va récupérer chaque ligne de données.

Pour charger l'entièreté d'une ligne de données récupéré dans l'objet *sqlreader*, on utilise ***sqlReader.Read().*** Cette méthode retourne la valeur false quand il n'y a plus aucune ligne à lire, sinon elle retourne true.

Pour obtenir la valeur d'une colonne de la ligne de données chargée, on utilise ***sqlReader****["nom de la colonne"].*

Pour rendre compatible le type de données des colonnes des tables avec celui des valeurs en C#, voici ce qu'on utilise :

integer (postgreSQL) → Convert.ToInt32() → int (c#)
varchar (postgreSQL) → Convert.ToString() → string (c#)
char (postgreSQL) → Convert.ToString() → string (c#)
real (postgreSQL) → Convert.ToDouble() → double (c#)
double (postgreSQL) → Convert.ToDouble() → double (c#)
numeric (postgreSQL) →(decimal) Convert.ToDouble() → decimal (c#)
date (postgreSQL) → Convert.ToDateTime() → DateTime (c#)

….

La méthode ***sqlReader.Close()*** doit être appelée quand il n'y a plus aucune ligne à récupérer dans la base de données. Si cette méthode n'est pas exécutée, plus aucune requête via la connexion courante ne pourra être réalisée.

```
// Obtenir la liste des catégories

public List<Categorie> ListeCategorie()
{
    List<Categorie> liste = null;
    NpgsqlCommand sqlCmd = null;

    try
    {
        sqlCmd = new NpgsqlCommand("select idCategorie, nom, " +
                                   "sousCategorie from CATEGORIE " +
                                   "order by nom, sousCategorie", this.SqlConn);


        NpgsqlDataReader sqlreader = sqlCmd.ExecuteReader();

        if (sqlreader.Read())
        {
            liste = new List<Categorie>();

            do
            {
                liste.Add(new Categorie(Convert.ToInt32(sqlreader["idCategorie"]),
                                        Convert.ToString(sqlreader["nom"]),
                                        Convert.ToString(sqlreader["sousCategorie"])));
            } while (sqlreader.Read());

        }
        sqlreader.Close();

    }

    catch (Exception e)
    {
        throw new ExceptionAccesBD(sqlCmd.CommandText, e.Message);
    }

    return liste;
}
```

Requête contenant un ou des paramètre(s)

Dans le texte de la requête, les paramètres sont précédés de :

```
sqlCmd = new NpgsqlCommand("select  LIVRE.isbn, titre, auteur, " +
                           " edition from LIVRE " +
                           " where idCategorie = :idCategorie " +
                           " order by auteur, titre",

                           this.SqlConn);
```

Pour ajouter un paramètre et le type à une requête :

```
sqlCmd.Parameters.Add(new NpgsqlParameter("idCategorie",
                      NpgsqlTypes.NpgsqlDbType.Integer));
```

Pour preparer la requête:

```
sqlCmd.Prepare();
```

Pour ajouter les valeurs aux paramètres

```
sqlCmd.Parameters[0].Value = categorie.Idcategorie;
```

```
//Obtenir la liste des livres appartenant à une catégorie
//
// entrée : idCategorie (catégorie.idCategorie)
// sortie la liste des exemplaires ordonné par auteur et titre

public List<Livre> ListeLivreCategorie(Categorie categorie)
{
    List<Livre> liste = null;
    NpgsqlCommand sqlCmd = null;

    try
    {
        sqlCmd = new NpgsqlCommand("select  LIVRE.isbn, titre, auteur, " +
                                   " edition from LIVRE " +
                                   " where idCategorie =  :idCategorie" +
                                   " order by auteur, titre",

                                   this.SqlConn);

        // Ajoute les paramètres

        sqlCmd.Parameters.Add(new NpgsqlParameter("idCategorie",
            NpgsqlTypes.NpgsqlDbType.Integer));


        // Prepare la commande

        sqlCmd.Prepare();

        // Ajouter les valeurs aux paramètres

        sqlCmd.Parameters[0].Value = categorie.Idcategorie;

        NpgsqlDataReader sqlreader = sqlCmd.ExecuteReader();

        if (sqlreader.Read())
        {

            liste = new List<Livre>();

            do
            {
                liste.Add(new Livre(categorie.Idcategorie,
                    Convert.ToString(sqlreader["isbn"]),
                    Convert.ToString(sqlreader["titre"]),
                    Convert.ToString(sqlreader["auteur"]),
                    Convert.ToString(sqlreader["edition"])));
            } while (sqlreader.Read());

        }
        sqlreader.Close();


        return liste;
    }

    catch (Exception e)
    {

        throw new ExceptionAccesBD(sqlCmd.CommandText, e.Message);
    }

}
```

Commande INSERT – UPDATE – DELETE

La méthode ***ExecuteReader*** de la classe ***NpgsqlCommand*** est utilisée lors de l'exécution d'une commande SELECT et retourne les résultats de la requête en tant qu'objet *DataReader.*

La méthode ***ExecuteNonQuery*** de la classe ***NpgsqlCommand*** est utilisée pour des requêtes qui ne renvoient aucune donnée, tels que les commandes INSERT, DELETE, UPDATE. Cette méthode exécute la commande et retourne le nombre de lignes affectées.

```
// Ajout d'une catégorie

public int AjouterCategorie(Categorie categorie)
{
    NpgsqlCommand sqlCmd = null;
    try
    {
        sqlCmd = new NpgsqlCommand("insert into " +
            "CATEGORIE(nom, sousCategorie) values " +
            "( :nom, :sousCategorie)", this.SqlConn);

        // Ajoute les paramètres

        sqlCmd.Parameters.Add(new NpgsqlParameter("nom",
            NpgsqlTypes.NpgsqlDbType.Varchar));
        sqlCmd.Parameters.Add(new NpgsqlParameter("sousCategorie",
            NpgsqlTypes.NpgsqlDbType.Varchar));

        // Prepare la commande

        sqlCmd.Prepare();

        // Ajouter les valeurs aux paramètres

        sqlCmd.Parameters[0].Value = categorie.Nom;
        sqlCmd.Parameters[1].Value = categorie.Souscategorie;

        // Execute la commande

        return (sqlCmd.ExecuteNonQuery());
    }
    catch (Exception e)
    {
        throw new ExceptionAccesBD(sqlCmd.CommandText, e.Message);
    }
}
```

Transaction

Pour créer une transaction, on appelle la méthode *BeginTransaction()* de la classe *NpgsqlConnection*

sqltr = this.SqlConn.BeginTransaction();

this.SqlConn représente une connexion à la base de données

Pour ajouter une commande postgreSQL à une transaction

sqlCmd1.Transaction = sqltr;

sqlCmd1 représente une instance de la classe *pgsqlCommand*

Dans la méthode *SupprimerLivre(Exemplaire exemplaire)* , si c'est le dernier exemplaire, on doit également le supprimer dans la table livre. Ces deux commandes de suppression sont indivisibles, pour cela on crée une transaction qui comprendra ces deux commandes.

```
// Supprimer un exemplaire,
// si dernier exemplaire, le supprimer également dans la table LIVRE


public int SupprimerLivre(Exemplaire exemplaire)
{
    NpgsqlCommand sqlCmd1 = null;
    NpgsqlCommand sqlCmd2 = null;
    NpgsqlTransaction sqltr = null;
    NpgsqlCommand sqlCmdCount = null;
    int nbr;
    string isbn;
    try
    {
        sqlCmdCount = new NpgsqlCommand("select count(*) as nbr, isbn " +
            " from exemplaire  " +
            " where isbn = " +
            "(select isbn from exemplaire where idLivre = :idLivre) " +
            "group by isbn ",
            this.SqlConn);

        // Ajout du paramètre

        sqlCmdCount.Parameters.Add(new NpgsqlParameter("idLivre",
            NpgsqlTypes.NpgsqlDbType.Integer));

        // Prepare la commande

        sqlCmdCount.Prepare();

        // Ajouter les valeurs aux paramètres

        sqlCmdCount.Parameters[0].Value = exemplaire.IdLivre;

        NpgsqlDataReader sqlreaderCount = sqlCmdCount.ExecuteReader();
    if (sqlreaderCount.Read())
    {
        nbr = Convert.ToInt32(sqlreaderCount["nbr"]);
        isbn = Convert.ToString(sqlreaderCount["isbn"]);

        sqlreaderCount.Close();

        sqlCmd1 = new NpgsqlCommand("delete from EXEMPLAIRE  " +
            "where idLivre = :idLivre ", this.SqlConn);

        // Ajoute les paramètres

        sqlCmd1.Parameters.Add(new NpgsqlParameter("idLivre",
            NpgsqlTypes.NpgsqlDbType.Integer));

        // Prepare la commande

        sqlCmd1.Prepare();

        // Ajouter les valeurs aux paramètres

        sqlCmd1.Parameters[0].Value = exemplaire.IdLivre;

        sqltr = this.SqlConn.BeginTransaction();

        sqlCmd1.Transaction = sqltr;

        sqlCmd1.ExecuteNonQuery();

        if (nbr == 1)
        {
            sqlCmd2 = new NpgsqlCommand("delete from LIVRE  " +
            "where isbn = :isbn ", this.SqlConn);

            // Ajoute les paramètres

            sqlCmd2.Parameters.Add(new NpgsqlParameter("isbn",
                NpgsqlTypes.NpgsqlDbType.Varchar));

            // Prepare la commande

            sqlCmd2.Prepare();
```

```
                // Ajouter les valeurs aux paramètres

                sqlCmd2.Parameters[0].Value = isbn;

                sqlCmd2.Transaction = sqltr;

                sqlCmd2.ExecuteNonQuery();

            }

            // Termine la transaction

            sqltr.Commit();

            return (1);

        }
        else
        {
            sqlreaderCount.Close();
            return 0;
        }

    }

    catch (Exception e)
    {
        sqltr.Rollback();
        throw new ExceptionAccesBD(sqlCmd1.CommandText, e.Message);
    }
}
```

Si dans la base de donnée un champ est à *null*, cette valeur ne peut pas être assigné à une variable dans c#, pour savoir si le champ est *null*, utiliser la méthode *IsDBNull* de la classe *NpgsqlDataReader*

```
sqlCmd = new NpgsqlCommand(
"select idPersonne, date_prolongation, date_sortie from EMPRUNT " +
"where idLivre = :idLivre and date_retour is null ",
this.SqlConn);

// Ajoute les paramètres

sqlCmd.Parameters.Add(new NpgsqlParameter("idLivre",
    NpgsqlTypes.NpgsqlDbType.Integer));


// Prepare la commande

sqlCmd.Prepare();

// Ajouter les valeurs aux paramètres

sqlCmd.Parameters[0].Value = idLivre;

NpgsqlDataReader sqlreaderE = sqlCmd.ExecuteReader();

if (sqlreaderE.Read())
{
    if (Convert.ToInt32(sqlreaderE["idPersonne"]) == idPersonne)
    {
        // livre emprunté par cette personne

        //if (sqlreaderE.IsDBNull(1))

        if (sqlreaderE.IsDBNull(sqlreaderE.GetOrdinal("date_prolongation")))
            // livre emprunté et non prolongé par cette personne
            trouve = 3;
        else
            // livre emprunté et prolongé par cette personne
            trouve = 4;
    }
```

deuxième champ sélectionné

retourne 1 dans ce cas

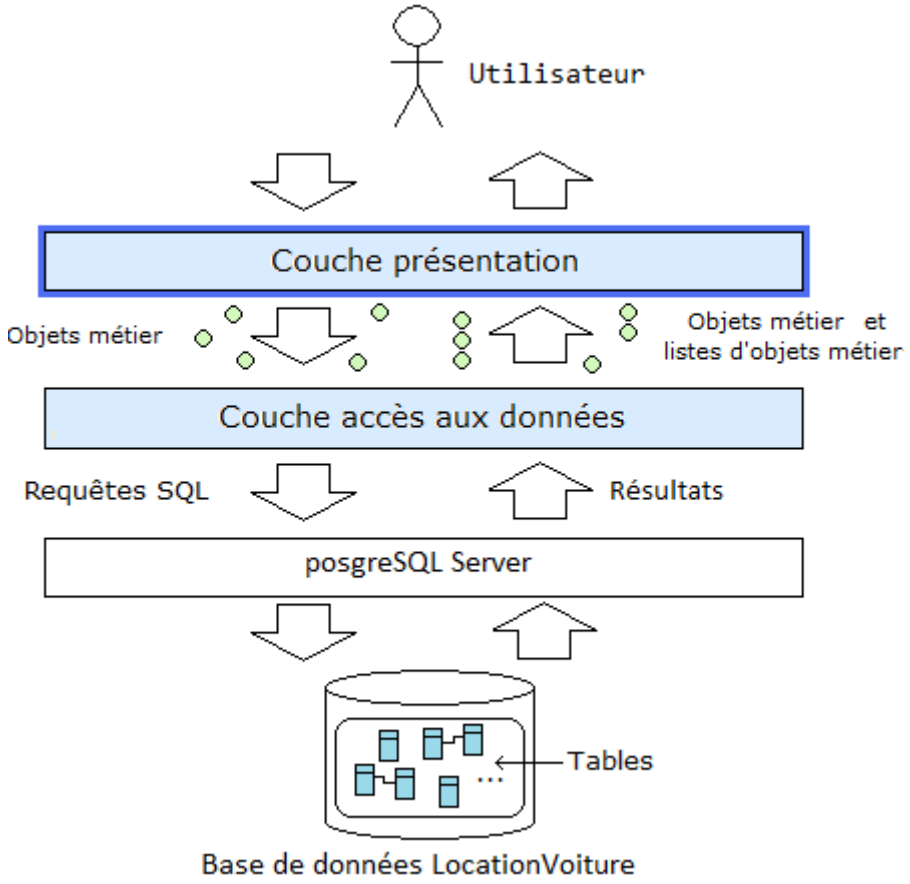
L'exception ExceptionAccesBD

Quand l'objet de la classe *AccesBD* rencontre un problème lors de l'accès à la base de données (impossible de créer une connexion, erreur pendant l'exécution d'une requête, ...), il déclenche une exception de type *ExceptionAccesBD*.

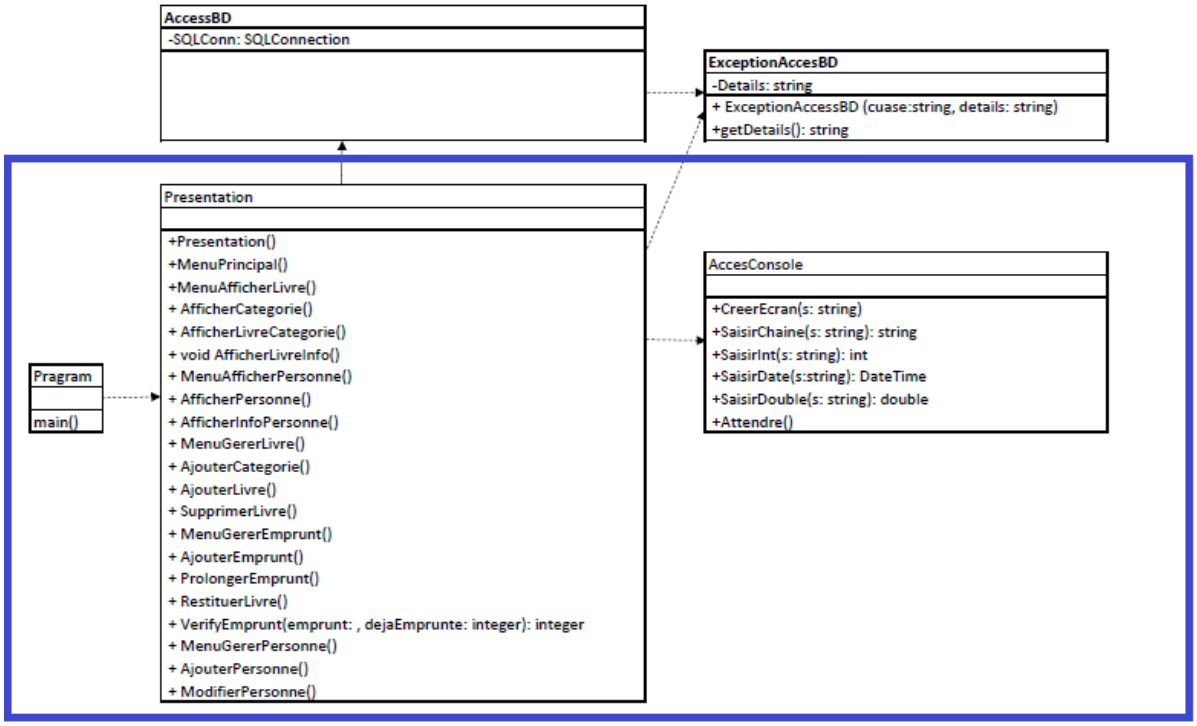
Cette classe contient l'attribut *Details* qui mémorise des informations détaillées sur la nature de l'erreur.

```
class ExceptionAccesBD : Exception
{
    2 références
    public string details { get; private set; }
    21 références
    public ExceptionAccesBD(string cause, string details) : base(cause)
    {
        this.details = details;
    }

}
```

Chapitre 9 : Création de la couche présentation



Les classes *Program, Presentation et AccesConsole* se trouvent dans l'espace de noms *couchePresentation*.



La classe Program

```
class Program
{

    static void Main(string[] args)
    {
        Presentation application = new Presentation();
        application.MenuPrincipal();
    }
}
```

La méthode *Main* est automatiquement exécutée quand démarre l'exécution du programme. Cette méthode crée un objet de type *Presentation*, puis elle donne le contrôle à la méthode *MenuPrincipa*l de cet objet.

La classe présentation

```
class Presentation
{
    private AccesBD monAccesBD;

    // Constructeur
    // Crée l'objet de type AccesBD (connection BD)

    public Presentation()
    {
        try
        {
            this.monAccesBD = new AccesBD();
        }
        catch (ExceptionAccesBD e)
        {
            Console.WriteLine("\nAccès à la BD impossible (" +
                e.Message + ")");
            System.Environment.Exit(0);
        }
    }
```

```
// Affiche le menu principal

public void MenuPrincipal()
{
    while (true)
    {

        AccesConsole.CreerEcran("Menu Principal");
        Console.WriteLine("1 = Afficher les livres" +
                          "\n2 = Afficher les personnes" +
                          "\n3 = Gérer les livres" +
                          "\n4 = Gérer les personnes" +
                          "\n5 = Gérer les emprunts" +
                          "\n6 = Quitter");
        try
        {
            switch (AccesConsole.SaisirInt("\nChoix "))
            {
                case 1:
                    MenuAfficherLivre();
                    break;

                case 2:
                    MenuAfficherPersonne();
                    break;

                case 3:
                    MenuGererLivre();
                    break;

                case 4:
                    MenuGererPersonne();
                    break;

                case 5:
                    MenuGererEmprunt();
                    break;

                case 6:
                    System.Environment.Exit(0);
                    break;
                default:
                    Console.WriteLine("\nSaisie incorrecte");
                    AccesConsole.Attendre();
                    break;
            }
        }
        catch (ExceptionAccesBD e)
        {
            Console.WriteLine("\n Menu principal ( " +
                e.Message + ")");
        }
        catch (Exception e)
        {
            Console.WriteLine("\nErreur rencontrée (" +
                e.Message + ")");
        }
        AccesConsole.Attendre();
    }
}
```

```
// Affiche le menu: Afficher les livres

public void MenuAfficherLivre()
{
    while (true)
    {

        AccesConsole.CreerEcran("Afficher les livres");
        Console.WriteLine("1 = Afficher les categories" +
                          "\n2 = Afficher les livres appartenant à une catégorie" +
                          "\n3 = Afficher les livres d'un titre et auteur donnés"  +
                          "\n4 = Retour au menu principal");
        try
        {
            switch (AccesConsole.SaisirInt("\nChoix "))
            {
                case 1:
                    AccesConsole.CreerEcran("Afficher les catégorie");
                    AfficherCategorie();
                    break;

                case 2:
                    AccesConsole.CreerEcran(
                        "Lister les livres d'une catégorie donnée");
                    AfficherLivreCategorie();
                    break;
                case 3:
                    AccesConsole.CreerEcran(
                        "Afficher information d'un livre donné");
                    AfficherLivreInfo();
                    break;
                case 4:

                    return;
                default:
                    Console.WriteLine("\nSaisie incorrecte");
                    AccesConsole.Attendre();
                    break;

            }
        }
        catch (ExceptionAccesBD e)
        {
            Console.WriteLine("\nErreur rencontrée BD ("
                + e.details + ")");
            throw e;
        }
        catch (Exception e)
        {
            Console.WriteLine("\nErreur rencontrée Menu ("
                + e.Message + ")");
        }
        AccesConsole.Attendre();
    }
}
```

```
// Affiche la liste des catégorie
//   pour chaque catégorie les informations suivantes sont affichées:
//    idCategorie, le nom de la catégorie, le nom de la sous-catégorie

public void AfficherCategorie()
{
    List<Categorie> liste = this.monAccesBD.ListeCategorie();

    Console.WriteLine("\n liste obtenu\n");

    if (liste != null)
    {
        Console.WriteLine("idCategorie, nom, sous-catégorie:\n");

        while (liste.Count > 0)
        {
            Console.WriteLine("{0}, {1}, {2}",
                liste[0].Idcategorie, liste[0].Nom,
                liste[0].Souscategorie);
            liste.RemoveAt(0);
        }
    }
    else
    {
        Console.WriteLine("\nIl n'y a aucune catégorie dans la BD!");
    }
}

// Demande d'encoder l'identifiant de la categorie
// Affiche la liste des livres appartenant à une catégorie donnée
//   pour chaque livre les informations suivantes sont affichées:
//    identifiant, titre, auteur, date_partion et edition

public void AfficherLivreCategorie()
{
    Categorie categorie = new Categorie();
    categorie.Idcategorie = AccesConsole.SaisirInt(
        "Entrez l'identifiant de la catégorie : ");

    List<Livre> liste = this.monAccesBD.ListeLivreCategorie(categorie);

    Console.WriteLine("\n liste obtenu\n");

    if (liste != null)
    {
        Console.WriteLine("isbn, titre, auteur,  edition:\n");

        while (liste.Count > 0)
        {
            Console.WriteLine("{0}, {1}, {2}, {3}",
                liste[0].Isbn, liste[0].Titre, liste[0].Auteur,
                liste[0].Edition);
            liste.RemoveAt(0);
        }
    }
    else
    {
        Console.WriteLine("\nIl n'y a aucun livre de cette catégorie dans la BD!");
    }
}
```

```
public void AfficherLivreInfo()
{

    string titre = AccesConsole.SaisirChaine(
        "Entrez le titre du livre : ");

    string auteur = AccesConsole.SaisirChaine(
        "Entrez l'auteur du livre : ");

    List<Exemplaire> listeDispo = this.monAccesBD.ListeLivreDisponible(titre, auteur);

    List<Exemplaire> listeEmprunte = this.monAccesBD.ListeLivreEmprunte(titre, auteur);

    Console.WriteLine("\n liste obtenu\n");

    if ((listeDispo == null) && (listeEmprunte == null))
    {
        Console.WriteLine("\nIl n'y a aucun livre avec ce titre et cet auteur dans la BD!");
    }
    else
    {
        if (listeDispo != null)
            while (listeDispo.Count > 0)
            {
                Console.WriteLine("{0}, Disponible",
                    listeDispo[0].IdLivre);
                listeDispo.RemoveAt(0);
            }

        if (listeEmprunte != null)
            while (listeEmprunte.Count > 0)
            {
                Console.WriteLine("{0}, Emprunte",
                    listeEmprunte[0].IdLivre);
                listeEmprunte.RemoveAt(0);
            }
    }
}
```

```
public void MenuAfficherPersonne()
{
    while (true)
    {

        AccesConsole.CreerEcran("Afficher les personnes");
        Console.WriteLine("1 = Afficher toutes les personnes" +
                          "\n2 = Afficher les informations d'une personne donnée" +
                          "\n3 = Retour au menu principal");
        try
        {
            switch (AccesConsole.SaisirInt("\nChoix "))
            {
                case 1:
                    AccesConsole.CreerEcran("Afficher les personnes");
                    AfficherPersonne();
                    break;

                case 2:
                    AccesConsole.CreerEcran(
                        "Afficher information personne");
                    AfficherInfoPersonne();
                    break;
                case 3:

                    return;
                default:
                    Console.WriteLine("\nSaisie incorrecte");
                    AccesConsole.Attendre();
                    break;
            }
        }
         catch (ExceptionAccesBD e)
         {
             Console.WriteLine("\nErreur rencontrée BD ("
                 + e.details + ")");
             throw e;
         }
         catch (Exception e)
         {
             Console.WriteLine("\nErreur rencontrée Menu ("
                 + e.Message + ")");
         }
         AccesConsole.Attendre();
     }
}
```

```
// Affiche la liste des personnes
//   pour chaque personne les informations suivantes sont affichées:
//    idPersonne, le nom, le prenom, le GSM, l'adresse

public void AfficherPersonne()
{
    List<Personne> liste = this.monAccesBD.ListePersonne();

    Console.WriteLine("\n liste obtenu\n");

    if (liste != null)
    {
        Console.WriteLine("idpersonne, nom, prenom, gsm, (rue, cp, localite):\n");

        while (liste.Count > 0)
        {
            Console.WriteLine("{0}, {1}, {2}, {3}, ({4}, {5},{6}): ",
                liste[0].IdPersonne, liste[0].Nom,
                liste[0].Prenom, liste[0].GSM, liste[0].adresse.rue,
                liste[0].adresse.cp, liste[0].adresse.localite);
            liste.RemoveAt(0);
        }
    }
    else
    {
        Console.WriteLine("\nIl n'y a aucune personne dans la BD!");
    }
}
```

```
// Demande d'encoder l'identifiant d'une personne
// affiche les informations de cette personne
//  nom, prénom, GSM, adresse

public void AfficherInfoPersonne()
{

    Personne personne = monAccesBD.InfoPersonne(AccesConsole.SaisirInt(
        "Entrez l'identifiant de la personne : "));
    if (personne == null)
    {
        Console.WriteLine(
     "Il n'existe pas de personne dans la BD avec cet identifiant");
    }
    else
    {
        Console.WriteLine("idpersonne, nom, prenom, gsm, " +
            "(rue, cp, localite):\n");

        Console.WriteLine("{0}, {1}, {2}, {3}, ({4}, {5},{6}): ",
            personne.IdPersonne, personne.Nom,
            personne.Prenom, personne.GSM, personne.adresse.rue,
            personne.adresse.cp, personne.adresse.localite);
    }
}
```

```
public void MenuGererLivre()
{
    while (true)
    {

        AccesConsole.CreerEcran("Gérer les livres");
        Console.WriteLine("1 = Ajouter une categorie" +
                          "\n2 = Ajouter un livre" +
                          "\n3 = Supprimer un livre" +
                          "\n4 = Retour au menu principal");
        try
        {
            switch (AccesConsole.SaisirInt("\nChoix "))
            {
                case 1:
                    AccesConsole.CreerEcran("Ajouter une catégorie");
                    AjouterCategorie();
                    break;

                case 2:
                    AccesConsole.CreerEcran(
                        "Ajouter un livre");
                    AjouterLivre();
                    break;
                case 3:
                    AccesConsole.CreerEcran(
                        "Suprimer un livre");
                    SupprimerLivre();
                    break;
                case 4:

                    return;
                default:
                    Console.WriteLine("\nSaisie incorrecte");
                    AccesConsole.Attendre();
                    break;
            }
        }
        catch (ExceptionAccesBD e)
        {
            Console.WriteLine("\nErreur rencontrée BD ("
                + e.details + ")");
            throw e;
        }
        catch (Exception e)
        {
            Console.WriteLine("\nErreur rencontrée Menu ("
                + e.Message + ")");
        }
        AccesConsole.Attendre();
    }
}
```

```
// Ajouter une categorie

public void AjouterCategorie()
{
    Categorie categorie = new Categorie();
    categorie.Nom = AccesConsole.SaisirChaine(
        "Entrez le nom de la catégorie: ");
    categorie.Souscategorie = AccesConsole.SaisirChaine(
        "Entrez le nom de la sous-catégorie: ");
    if (monAccesBD.ExisteCategorie(categorie.Nom,
        categorie.Souscategorie) != 0)
        Console.WriteLine("\nCette catégorie existe déjà");
    else
    {
        if (monAccesBD.AjouterCategorie(categorie) == 0)
            Console.WriteLine("\nL'ajout n'a pas eu lieu");
        else
            Console.WriteLine("\nL'ajout s'est bien déroulé");
    }
}
// Ajouter une livre

public void AjouterLivre()
{
    string isbn = AccesConsole.SaisirChaine("Entrez isbn: ");
    Livre livre = monAccesBD.GetLivreISBN(isbn);

    if (livre == null)
    {
        Console.WriteLine("\nPremier exemplaire\n");
        livre = new Livre();
        livre.Isbn = isbn;

        livre.Titre = AccesConsole.SaisirChaine("Entrez le titre: ");
        livre.Auteur = AccesConsole.SaisirChaine("Entrez l'auteur: ");

        livre.Edition = AccesConsole.SaisirChaine("Entrez l'édition: ");
        livre.categorie.Idcategorie = AccesConsole.SaisirInt(
        "Entrez l'identifiant de la catégorie: ");

        if (monAccesBD.AjouterLivre(livre) == 0)
        {
            Console.WriteLine("\nL'ajout du nouveau livre n'a pas eu lieu");
        }
    }
    else
    {
        if (monAccesBD.AjouterExemplaire(livre) == 0)
        {
            Console.WriteLine("\nL'ajout n'a pas eu lieu");
        }
        else
            Console.WriteLine("\nL'ajout s'est bien déroulé");
    }
}
```

```
// Supprimer un livre

public void SupprimerLivre()
{
    Exemplaire livre = new Exemplaire();
    livre.IdLivre = AccesConsole.SaisirInt(
    "Entrez l'identifiant du livre : ");

    if (monAccesBD.SupprimerLivre(livre) == 0)
        Console.WriteLine("\nLa suppression n'a pas eu lieu");
    else
        Console.WriteLine("\nLa suppression s'est bien déroulé");
}
public void MenuGererPersonne()
{
    while (true)
    {

        AccesConsole.CreerEcran("Gérer les personnes");
        Console.WriteLine("1 = Ajouter une personne" +
                          "\n2 = Modifier les coordonnées d'une personne" +
                          "\n3 = Retour au menu principal");
        try
        {
            switch (AccesConsole.SaisirInt("\nChoix "))
            {
                case 1:
                    AccesConsole.CreerEcran("Ajouter une personne");
                    AjouterPersonne();
                    break;

                case 2:
                    AccesConsole.CreerEcran(
                        "Modifier les coordonnées d'une personne");
                    ModifierPersonne();
                    break;
                case 3:
                    return;
                default:
                    Console.WriteLine("\nSaisie incorrecte");
                    AccesConsole.Attendre();
                    break;
            }
        }
         catch (ExceptionAccesBD e)
         {
             Console.WriteLine("\nErreur rencontrée BD ("
                 + e.details + ")");
             throw e;
         }
         catch (Exception e)
         {
             Console.WriteLine("\nErreur rencontrée Menu ("
                 + e.Message + ")");
         }
         AccesConsole.Attendre();
    }
}
```

```
public void AjouterPersonne()
{

    Personne personne = new Personne();
    personne.Nom = AccesConsole.SaisirChaine(
    "Entrez le nom : ");
    personne.Prenom = AccesConsole.SaisirChaine("Entrez le prénom: ");
    personne.GSM = AccesConsole.SaisirChaine("Entrez le GSM: ");
    personne.adresse = new Adresse(AccesConsole.SaisirChaine("Entrez la rue: "),
        AccesConsole.SaisirChaine("Entrez le code postal: "),
        AccesConsole.SaisirChaine("Entrez la localité: "));

    if (monAccesBD.AjouterPersonne(personne) == 0)
        Console.WriteLine("\nL'ajout n'a pas eu lieu");
    else
        Console.WriteLine("\nL'ajout s'est bien déroulé");
}

// Modifie une personne
// Seuls les paramètres à modifier doivent être encodés

public void ModifierPersonne()
{
    int idPersonne;

    idPersonne = AccesConsole.SaisirInt("Entrez idPersonne: ");

    Personne personne = monAccesBD.InfoPersonne(idPersonne);
    if (personne == null)
    {
        Console.WriteLine("\n Cette personne n'existe pas");
    }
    else
    {
        Console.WriteLine("\nLes coordonnées sont  (nom, prénom,GSM, adresse");
        Console.WriteLine("{0},{1}, {2}, {3} {4} {5}\n",
            personne.Nom, personne.Prenom, personne.GSM,
            personne.adresse.rue, personne.adresse.cp,
            personne.adresse.localite);
        string nom = AccesConsole.SaisirChaine("Entrez le nom: ");
        string prenom = AccesConsole.SaisirChaine(
            "Entrez le prenom: ");
        string gsm = AccesConsole.SaisirChaine(
            "Entrez le numéro de GSM: ");
        string rue = AccesConsole.SaisirChaine("Entrez la rue: ");
        string cp = AccesConsole.SaisirChaine(
            "Entrez le code postale: ");
        string localite = AccesConsole.SaisirChaine(
            "Entrez la localité: ");

        if (nom.Trim() != "")
            personne.Nom = nom;
        if (prenom.Trim() != "")
            personne.Prenom = prenom;

        if (gsm.Trim() != "")
            personne.GSM = gsm;

        if (rue.Trim() != "")
            personne.adresse.rue = rue;

        if (cp.Trim() != "")
            personne.adresse.cp = cp;
```

```
            if (localite.Trim() != "")
                personne.adresse.localite = localite;

            if (monAccesBD.ModifierPersonne(personne) == 0)
                Console.WriteLine("\nLa modification n'a pas eu lieu");
            else
                Console.WriteLine("\nla modification s'est bien déroulé");
        }
    }

    public void MenuGererEmprunt()
    {
        while (true)
        {

            AccesConsole.CreerEcran("Gérer les emprunts");
            Console.WriteLine("\n1 = Ajouter un emprunt d'un livre par une personne" +
                              "\n2 = Restituer un livre" +
                              "\n3 = Prolonger un emprunt" +
                              "\n4 = Retour au menu principal");
            try
            {
                switch (AccesConsole.SaisirInt("\nChoix "))
                {
                   case 1:
                        AccesConsole.CreerEcran(
                            "Ajouter un emprunt");
                        AjouterEmprunt();
                        break;
                    case 2:
                        AccesConsole.CreerEcran(
                            "RestituerLivre");
                        RestituerLivre();
                        break;
                    case 3:
                        AccesConsole.CreerEcran(
                            "Prolonger l'emprunt d'un livre");
                        ProlongerEmprunt();
                        break;
                    case 4:
                        return;
                    default:
                        Console.WriteLine("\nSaisie incorrecte");
                        AccesConsole.Attendre();
                        break;
                }
            }
            catch (ExceptionAccesBD e)
            {
                Console.WriteLine("\nErreur rencontrée BD ("
                    + e.details + ")");
                throw e;
            }
            catch (Exception e)
            {
                Console.WriteLine("\nErreur rencontrée Menu ("
                    + e.Message + ")");
            }
            AccesConsole.Attendre();
        }
    }
```

```
// Verifier si un emprunt exist
// procédure appelé par ....

// entrée : emprunt : idPersonne et idLivre
//          dejaEmprunte : si vaut 0 alors message si livre déjà emprunté
//
// sortie : 0 : le livre n'existe pas
//          1 : le livre et la personne existent et
//              le livre n'est pas encore emprunté
//          2 : le livre est déjà emprunté par une autre personne
//          3 : le livre est emprunté par cette personne et peut être prolongé
//              (dejaEmprunte = 1)
//          4 : le livre est emprunté par cette personne et ne peut pas être prolongé
//              (dejaEmprunte = 1)
//          5 : personne a dépassé son quota

// vérifie si l'exemplaire existe et si c'est le cas si cet exemplaire a été emrunté par la personne ou prolongé
// 0: exemplaire n'existe pas
// 1: exemplaire disponible
// 2: exemplaire emprunté par cette personne
// 3: exemplaire emprunté par une autre personne
// 4: exemplaire qui a déjà été prolongé par cette personne
private int VerifyEmprunt(Emprunt emprunt, int dejaEmprunte)
{
    //Emprunt emprunt = new Emprunt();
    emprunt.emprunteur.IdPersonne = AccesConsole.SaisirInt(
        "Entrez l'identifiant de la personne: ");
    int trouve = monAccesBD.ExistePersonne
        (emprunt.emprunteur.IdPersonne);
    if (trouve == 1)
    {
        if (dejaEmprunte == 0 && monAccesBD.getNbreLivresEmpruntes
                (emprunt.emprunteur.IdPersonne) == 10)
        {
            Console.WriteLine("\n Cette personne a déjà emprunté 10 livres");
            trouve = 5;
        }
        else
        {
            emprunt.livreEmprunte.IdLivre = AccesConsole.SaisirInt(
            "Entrez l'identifiant du livre: ");

            trouve = monAccesBD.VerifyEmprunt(emprunt);

            if (trouve == 0)
                Console.WriteLine("\n Ce livre n'existe pas");
            else
            {
                if ((trouve == 2 || trouve == 3) && dejaEmprunte == 0)
                    Console.WriteLine("\n Ce livre est déjà emprunté");
                if (trouve == 4)
                    Console.WriteLine("\n Ce livre a déjà été prolongé");
            }
        }
    }
    else
        Console.WriteLine("\n Cette personne n'existe pas");
    return trouve;
}
```

```
    // Ajouter un emprunt

    private void AjouterEmprunt()
    {
        Emprunt emprunt = new Emprunt();

        if (this.VerifyEmprunt(emprunt, 0) == 1)
        {
            if (monAccesBD.AjouterEmprunt(emprunt) == 0)
                Console.WriteLine("\nL'ajout n'a pas eu lieu");
            else
                Console.WriteLine("\nL'ajout s'est bien déroulé");
        }
    }

    // Restituer un livre revient à mette à jour la date_retour à la date du jour

    public void RestituerLivre()
    {
        Emprunt emprunt = new Emprunt();

        int verification = this.VerifyEmprunt(emprunt, 1);

        if (verification == 3 || verification == 4)
        {
            if (monAccesBD.RestituerLivre(emprunt) == 0)
                Console.WriteLine("\nLa suppression n'a pas eu lieu");
            else
                Console.WriteLine("\nLa suppression s'est bien déroulée");
        }
        else
            Console.WriteLine("\n Cet exemplaire ne peut pas être restitué par cette personne");
    }

    // Prolonge un emprunt (revient à mettre la date de prolongation au jour cournant)

    public void ProlongerEmprunt()
    {
        Emprunt emprunt = new Emprunt();
        if (this.VerifyEmprunt(emprunt, 1) == 3)
        {
            if (monAccesBD.ProlongerEmprunt(emprunt) == 0)
                Console.WriteLine("\nLa prolongation n'a pas eu lieu");
            else
                Console.WriteLine("\nLa prolongation s'est bien déroulée");
        }
        else
            Console.WriteLine("\n Cet exemplaire ne peut pas être prolongé par cette personne");
    }

}
```

Classe AccesControl

```
static public class AccesConsole
{
    // ------------------------------------------------------------
    // CreerEcran: effacer l'écran et afficher un titre
    // param: le titre de l'écran
    // sortie: aucune
    // ------------------------------------------------------------
    static public void CreerEcran(string s)
    {
        Console.Clear();
        Console.WriteLine(s);
        Console.WriteLine(new String('-', s.Length) + "\n");
    }
    // ------------------------------------------------------------
    // SaisirChaine: saisir une chaîne
    // param: un message d'introduction à afficher
    // sortie: une chaîne
    // ------------------------------------------------------------
    static public string SaisirChaine(string s)
    {
        Console.Write(s);
        return Console.In.ReadLine();
    }
    // ------------------------------------------------------------
    // SaisirDate: saisir un date
    // param: un message d'introduction à afficher
    // sortie: un objet de type DateTime
    // ------------------------------------------------------------
    static public DateTime SaisirDate(string s)
    {
        Console.Write(s);
        return Convert.ToDateTime(Console.In.ReadLine());
    }
```

```
    // ------------------------------------------------------------------------
    // SaisirInt: saisir un entier
    // param: un message d'introduction à afficher
    // sortie: un entier
    // ------------------------------------------------------------------------
    static public int SaisirInt(string s)
    {
        Console.Write(s);
        return Convert.ToInt32(Console.In.ReadLine());
    }
    // ------------------------------------------------------------------------
    // SaisirDouble: saisir un double
    // param: un message d'introduction à afficher
    // sortie: un double
    // ------------------------------------------------------------------------
    static public double SaisirDouble(string s)
    {
        Console.Write(s);
        return Convert.ToDouble(Console.In.ReadLine());
    }


    // ------------------------------------------------------------------------


    // ------------------------------------------------------------------------
    // Attendre: afficher un message puis Attendre la pression d'une touche
    // param: aucun
    // sortie: aucune
    // ------------------------------------------------------------------------
    static public void Attendre()
    {
        Console.Write("\nPressez une touche pour continuer...");
        Console.ReadKey();
        Console.WriteLine("\n");
    }
}
```