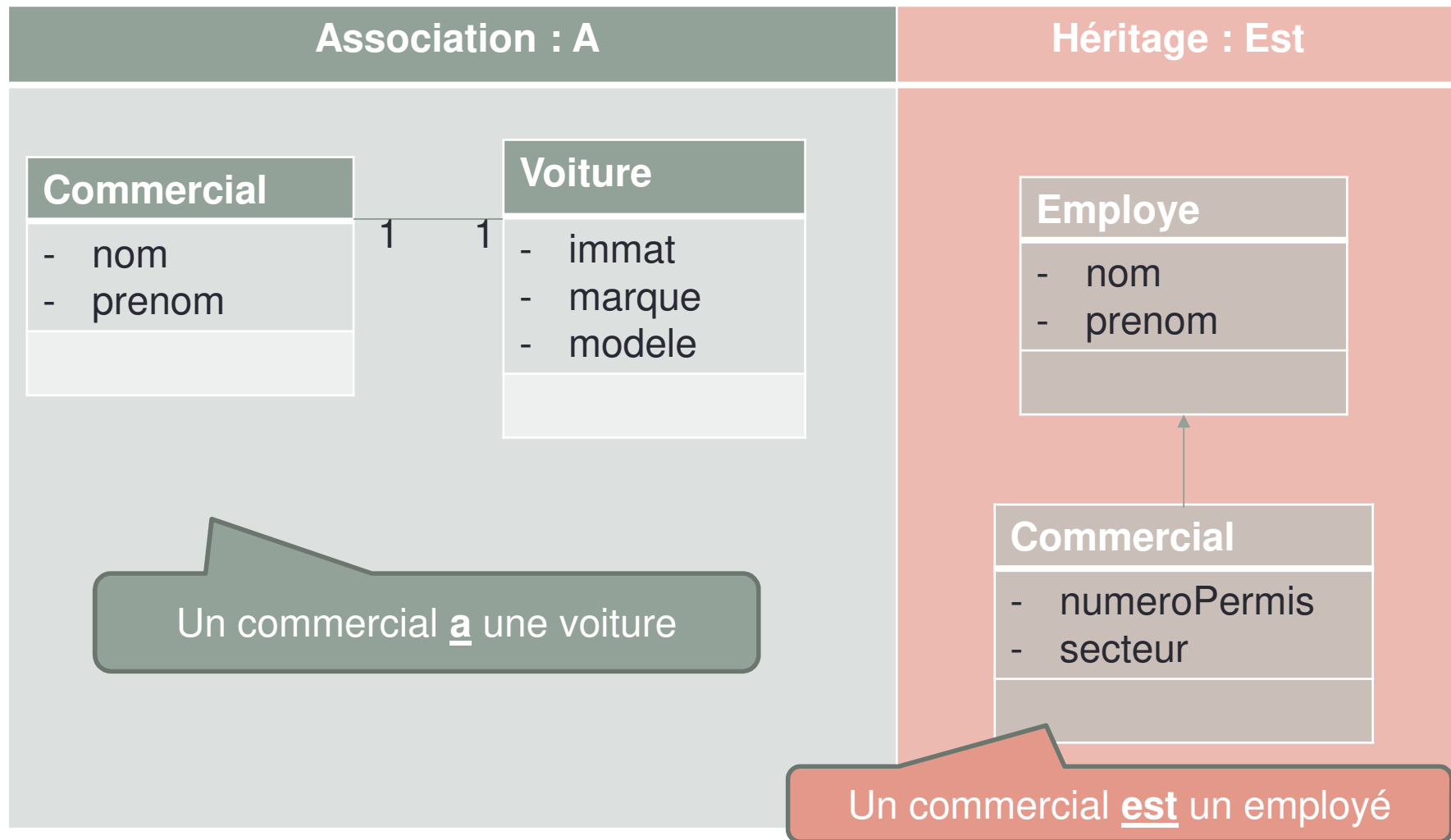


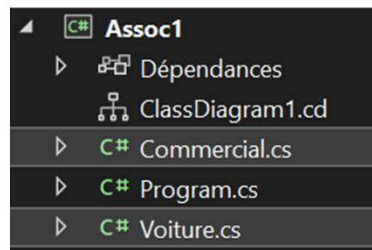
ASSOCIATION

Les 2 principaux liens entre classe



Association 1-1

- Un commercial utilise une voiture
- Une voiture est utilisée par un commercial



Association 1-1 en C#

- Pour cet exemple, on va donc coder 2 classes dans 2 fichiers différents au sein du projet :

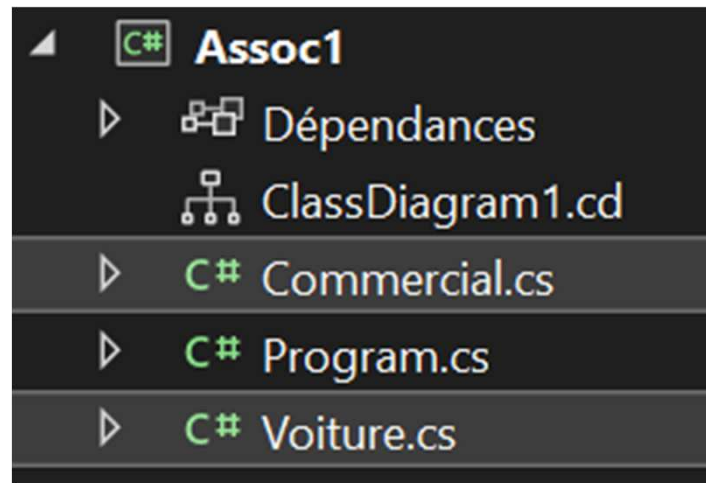
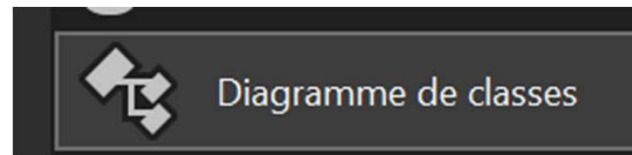


Diagramme de classe sous visual

- Pour avoir le diagramme de classe depuis Visual, depuis le projet, Ajouter un nouvel élément puis choisir :



- Ensuite faire glisser ses classes depuis l'explorateur vers la zone de dessin à gauche :

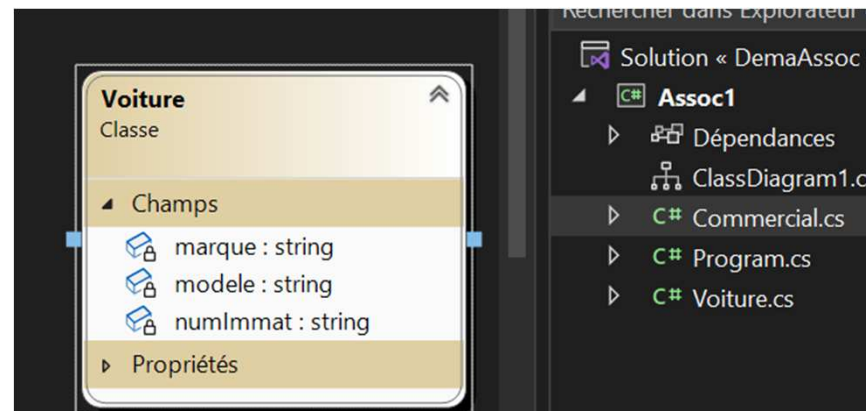
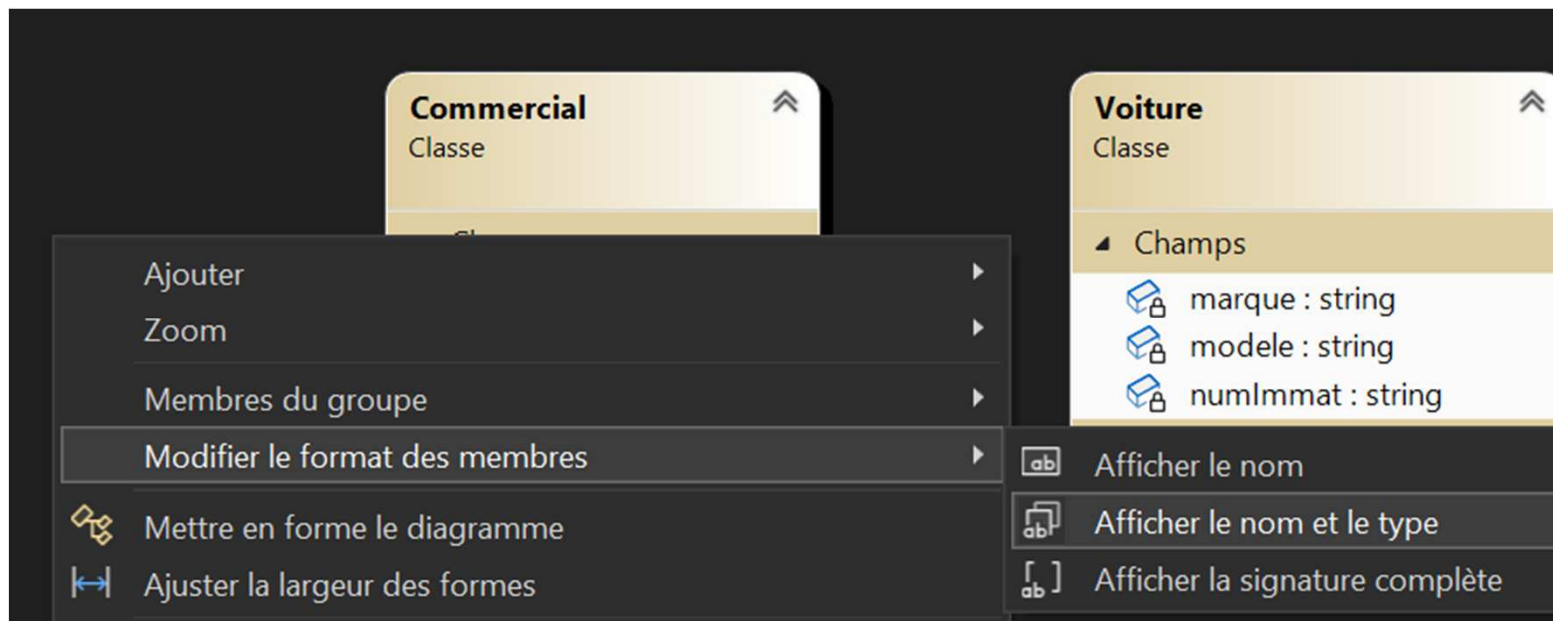


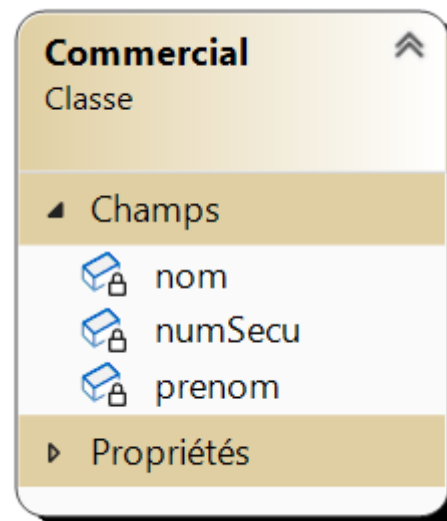
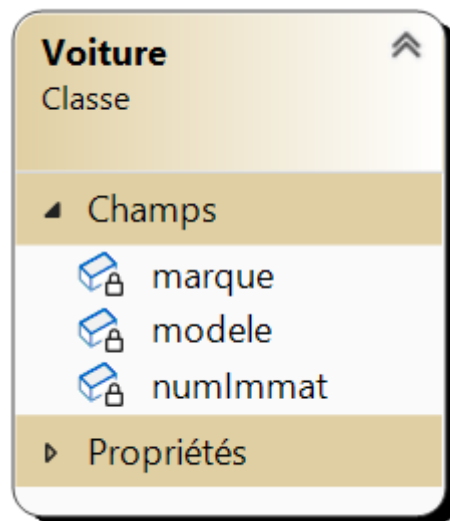
Diagramme de classe sous visual

- Puis clic droit dans la zone vide et choisir le paramétrage ci-dessous pour affiner l'affichage :



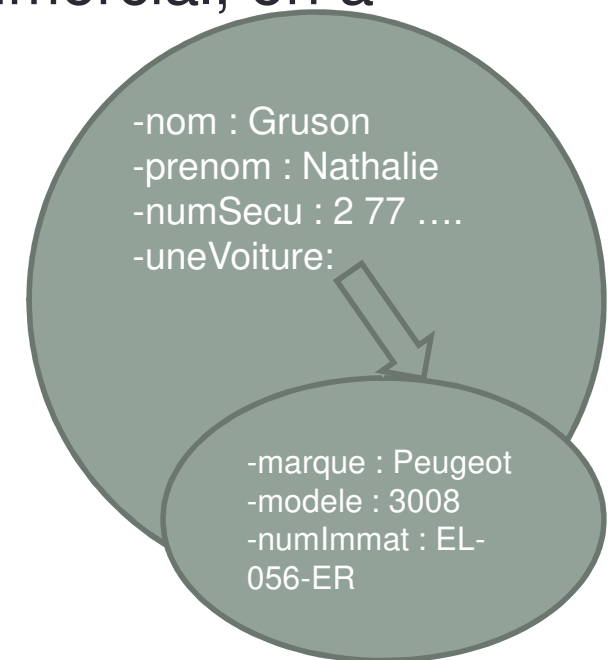
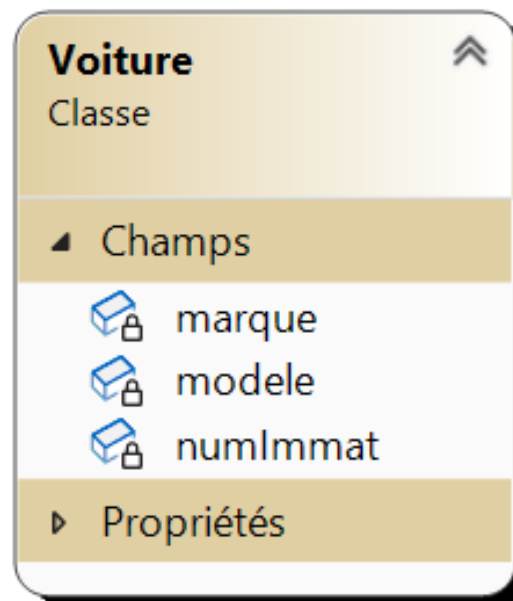
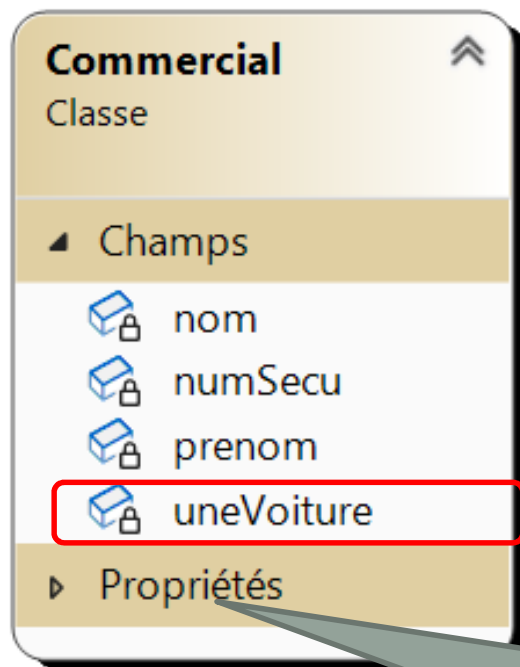
Association 1-1

Ainsi définies les classes et donc les objets n'ont aucun lien possible : on aura la possibilité de créer des voitures et des commerciaux mais rien pour faire le lien !



Association 1-1

Ici, on a une navigabilité depuis le commercial pour atteindre la voiture, au sein d'un objet commercial, on a une référence à un objet voiture

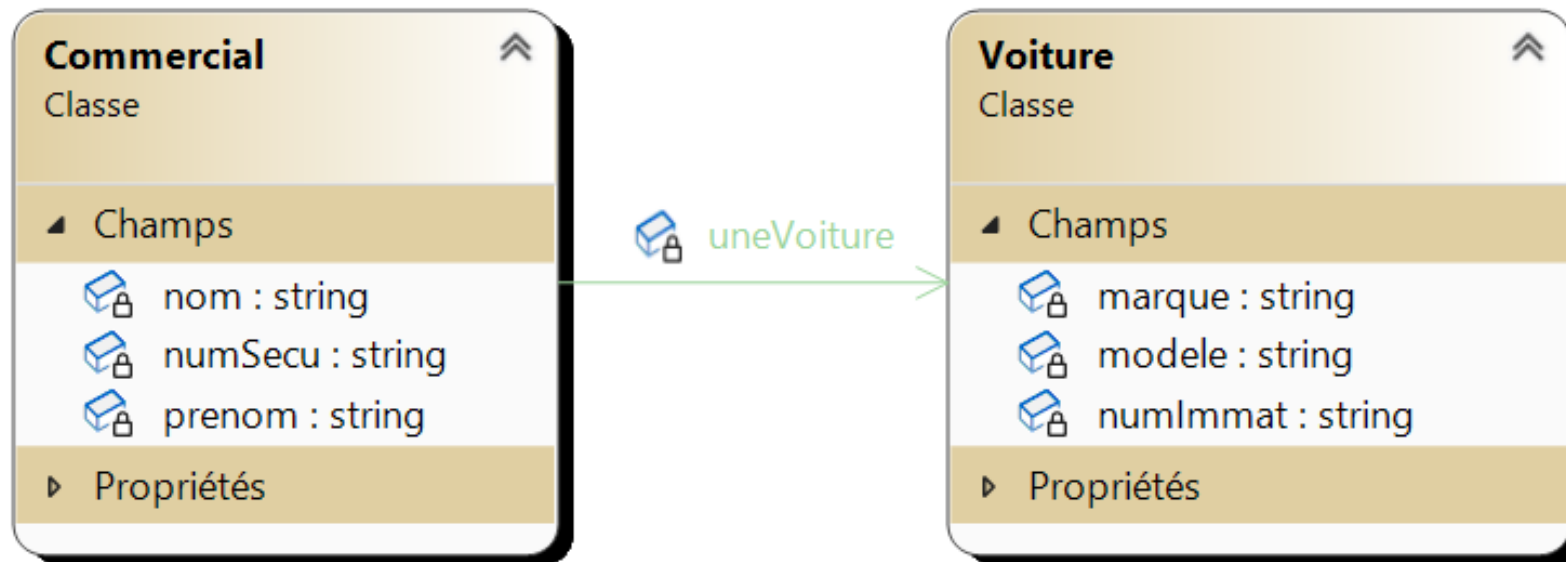
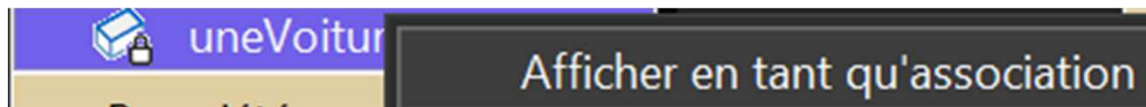


Préfixée par « une » pour que la propriété générée soit « UneVoiture » et non « Voiture » : ambiguïté avec la classe



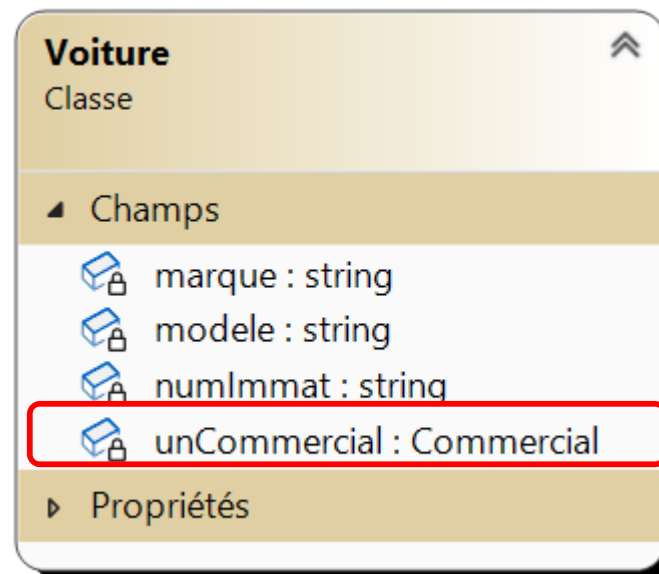
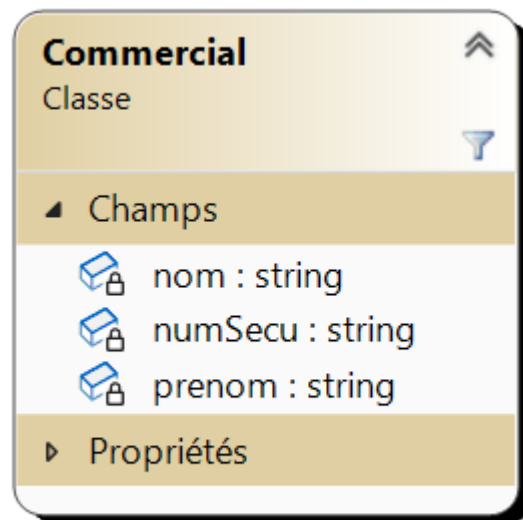
Association 1-1

Il est possible de représenter le champ en tant qu'association, mais il reste un champ au niveau du code :



Association 1-1

Ici, on a une navigabilité depuis la voiture pour atteindre le commercial, au sein d'un objet voiture, on a une référence à un objet commercial



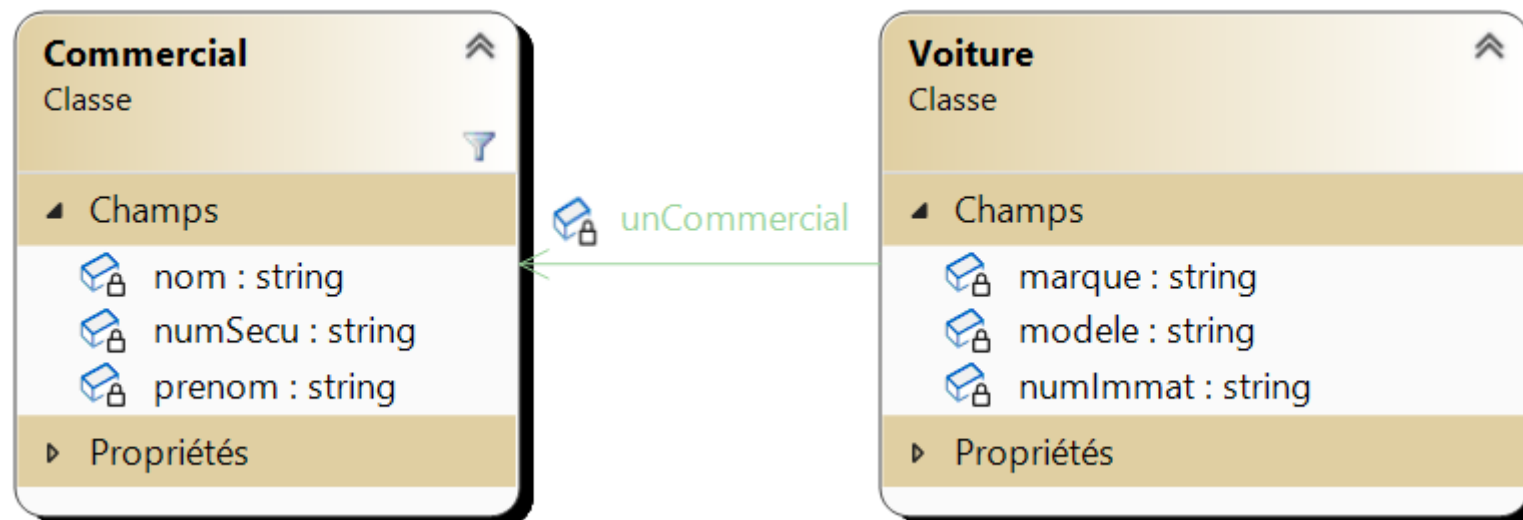
-marque : Peugeot
-modele : 3008
-numImmat : EL-056-ER
-unCommercial

-nom : Gruson
-prenom : Nathalie
-numSecu : 2 77
....
-uneVoiture:



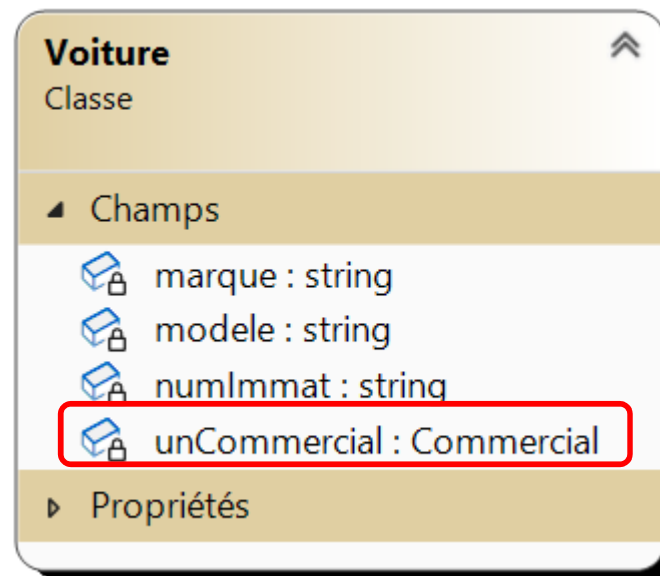
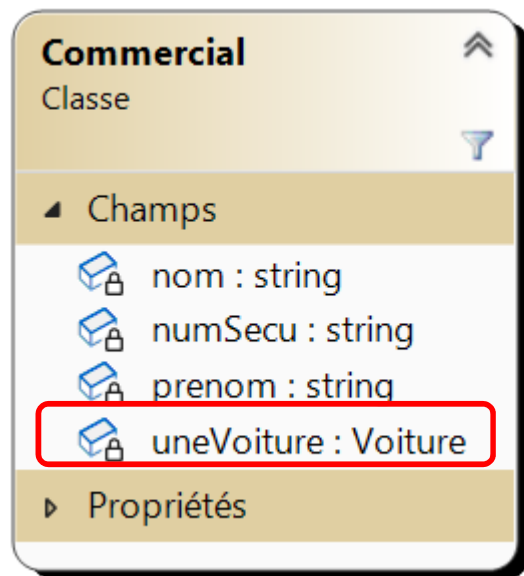
Association 1-1

Représentation du champ en tant qu'association :



Association 1-1 :: double navigabilité

On peut aussi décider d'avoir une double navigabilité.
Le choix de la navigabilité dépend des besoins applicatifs



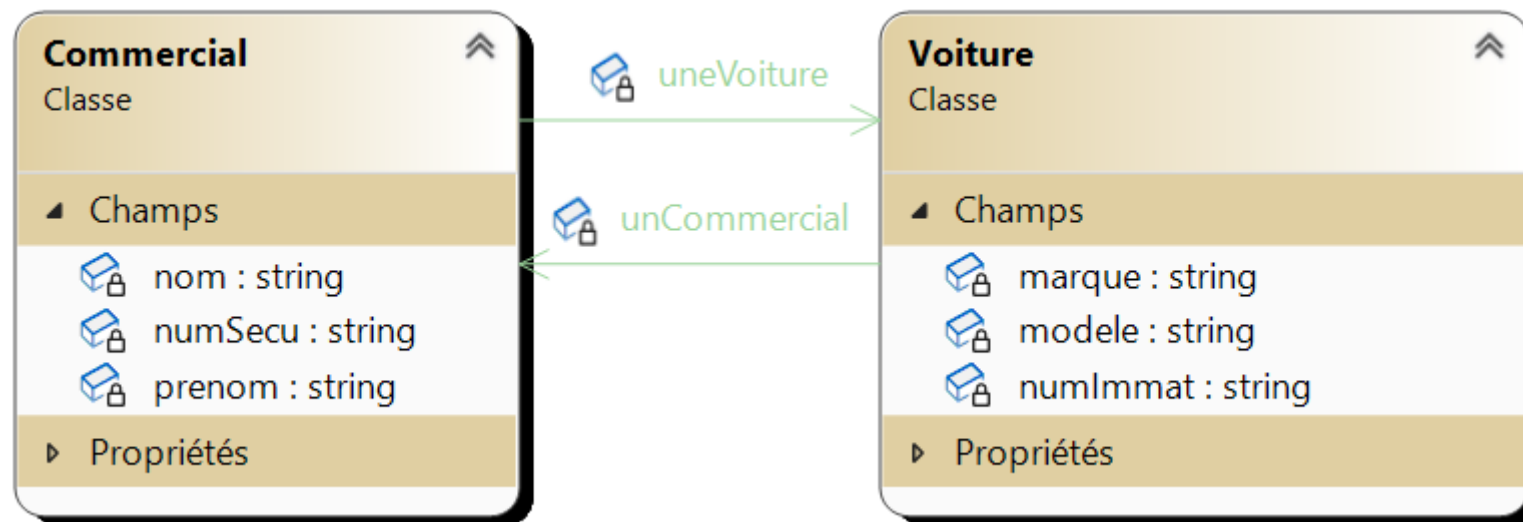
-marque : Peugeot
-modele : 3008
-numImmat : EL-056-ER
-unCommercial :

-nom : Gruson
-prenom : Nathalie
-numSecu : 2 77
-uneVoiture:



Association 1-1 : double navigabilité

Représentation des champs en tant qu'association :



Association 1-1 en code C#

- Le constructeur de Commercial attend donc un objet de classe Voiture

```
public Commercial(string numSecu, string nom, string prenom, Voiture uneVoiture)
{
    this.NumSecu = numSecu;
    this.Nom = nom;
    this.Prenom = prenom;
    this.UneVoiture = uneVoiture;
}
```

- Lorsqu'on instancie un commercial, il faut avant tout instancier une voiture

```
Voiture voiture = new Voiture("EL-05-3RE", "Peugeot", "3008");
Commercial commercial = new Commercial("2771244", "Gruson", "Nathalie", voiture);
```

- Possible en une ligne :

```
Commercial commercial = new Commercial("2771244", "Gruson", "Nathalie", new Voiture("EL-05-3RE", "Peugeot", "3008"));
```

Association 1-1

- Surcharge du constructeur possible pour créer la voiture à la création du commercial et ne pas avoir à créer la voiture avant :

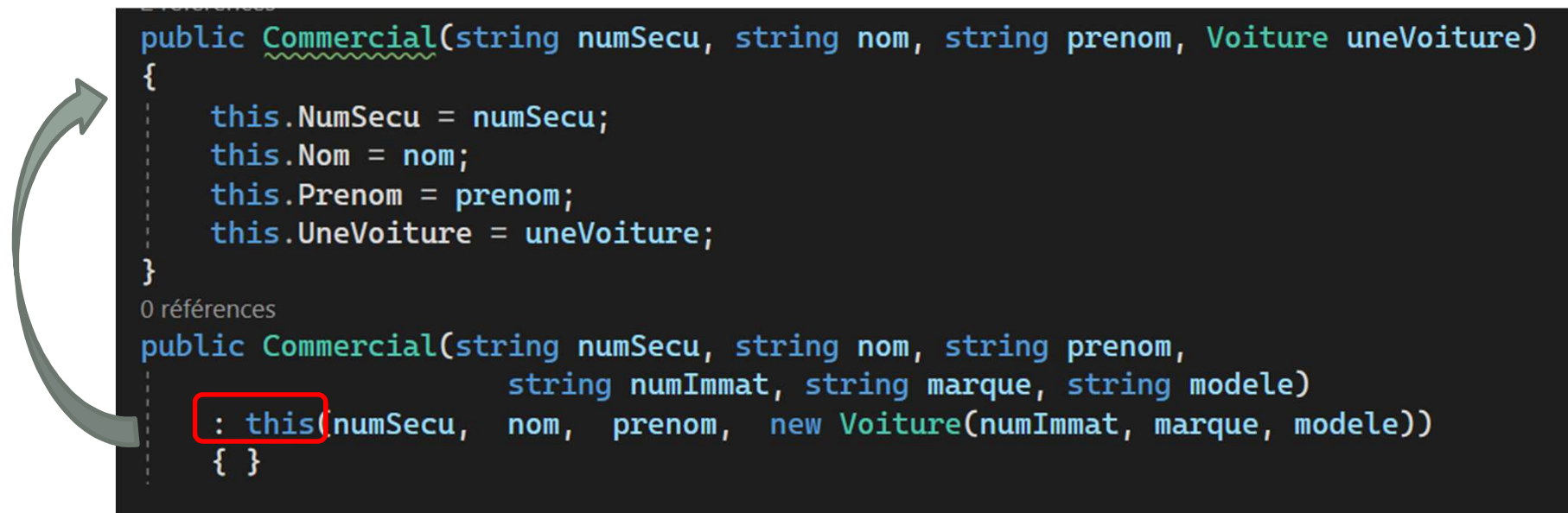
```
public Commercial(string numSecu, string nom, string prenom,  
                  string numImmat, string marque, string modele)  
{  
    this.NumSecu = numSecu;  
    this.Nom = nom;  
    this.Prenom = prenom;  
    this.UneVoiture = new Voiture(numImmat, marque, modele);  
}
```

- Pas besoin d'instancier de voiture avant :

```
Commercial commercial = new Commercial("2771244", "Gruson", "Nathalie", "EL-05-3RE", "Peugeot", "3008");
```

Association 1-1

- Surcharge du constructeur avec factorisation de code :
mot clef **this**



The diagram illustrates code reuse in C++ using the `this` keyword. It shows two constructor definitions for the `Commercial` class. The first constructor takes four parameters: `numSecu`, `nom`, `prenom`, and `Voiture uneVoiture`. The second constructor takes five parameters: `numSecu`, `nom`, `prenom`, `numImmat`, `marque`, and `modele`. The second constructor is a specialization that reuses the logic of the first by calling `this(numSecu, nom, prenom, new Voiture(numImmat, marque, modele))`. A red box highlights the `this` keyword in the second constructor, and a curved arrow points from it to the first constructor, indicating the call to the base constructor.

```
public Commercial(string numSecu, string nom, string prenom, Voiture uneVoiture)
{
    this.NumSecu = numSecu;
    this.Nom = nom;
    this.Prenom = prenom;
    this.UneVoiture = uneVoiture;
}

0 références
public Commercial(string numSecu, string nom, string prenom,
                  string numImmat, string marque, string modele)
: this(numSecu, nom, prenom, new Voiture(numImmat, marque, modele))
{ }
```


Association 1-1

- Au sein de la classe Commercial, on peut appeler les méthodes ou les propriétés de la classe Voiture par l'intermédiaire de la propriété UneVoiture

```
public override string? ToString()
{
    return Nom + " " + Prenom + "\n"
        + NumSecu + "\n"
        + UneVoiture.ToString();
}
```

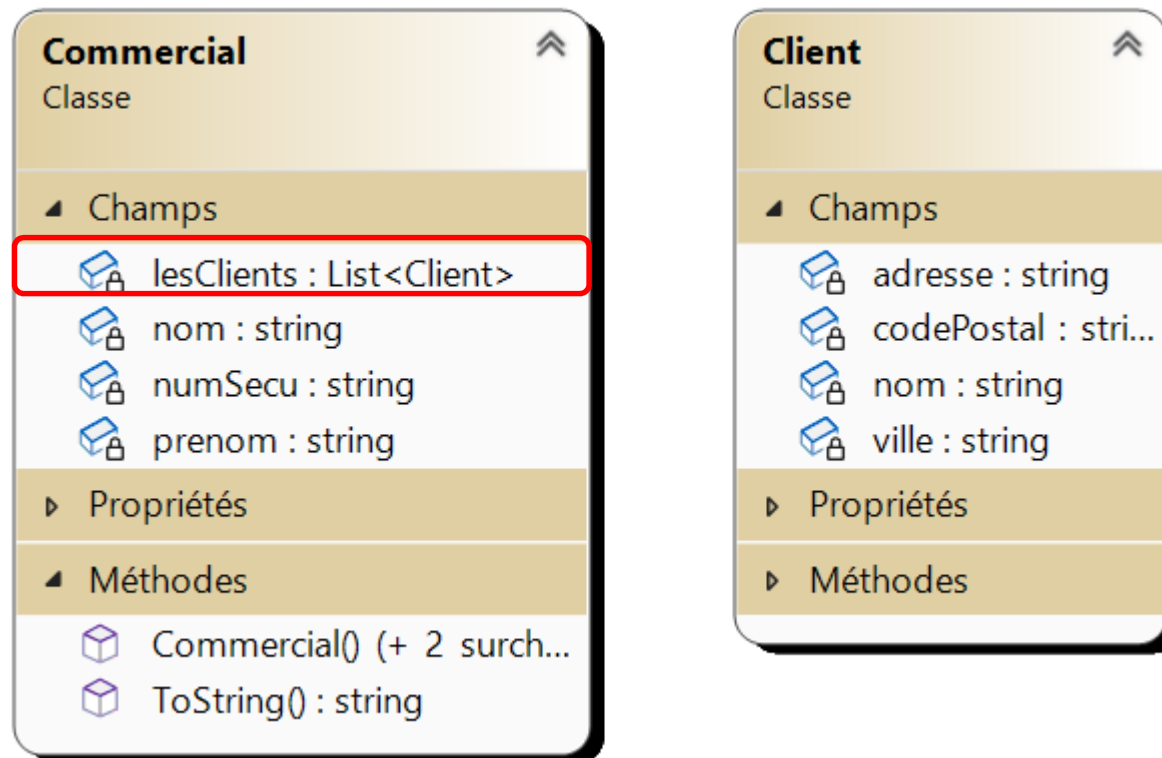
Association 1-N

Exemple : Un commercial visite plusieurs clients pour renouveler ses ventes.



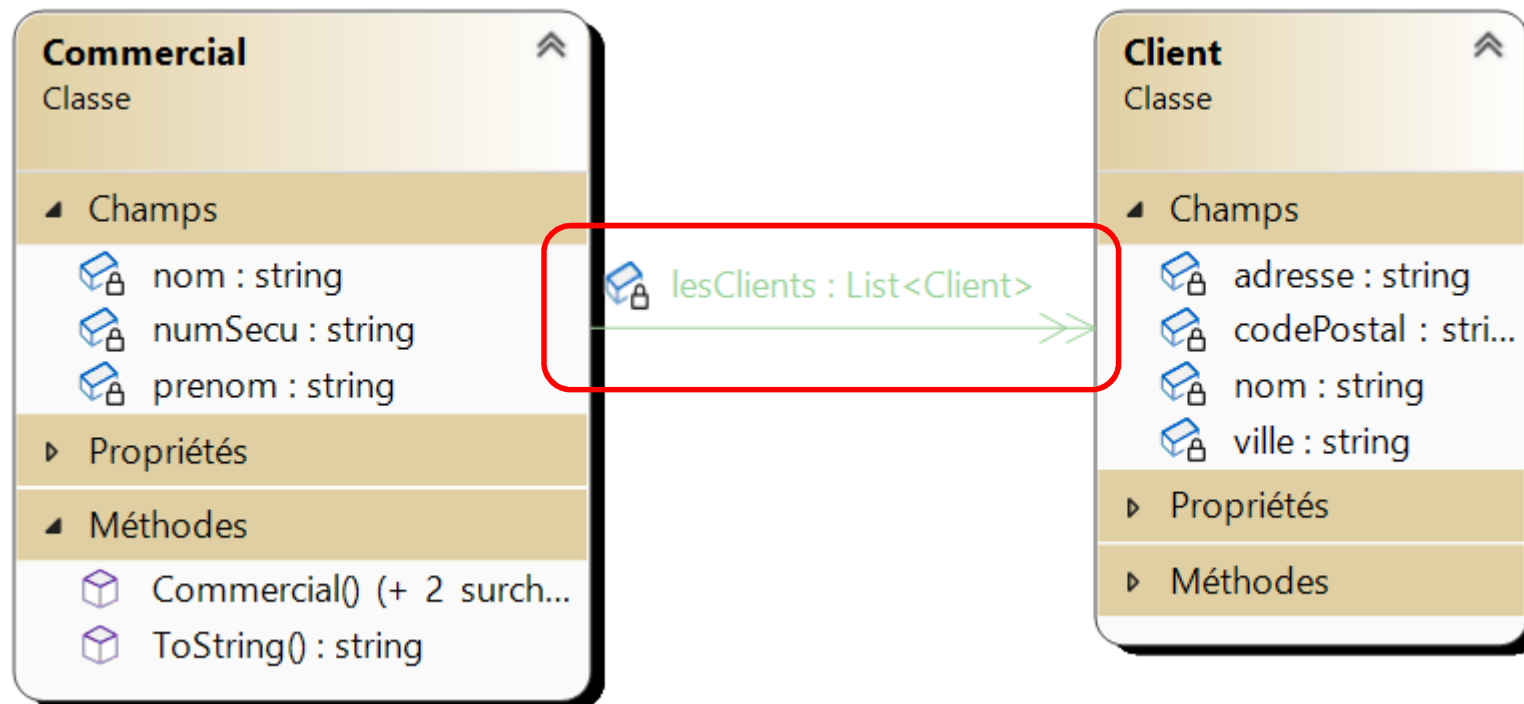
Association 1-N

Ici, on a une navigabilité depuis le commercial pour atteindre les clients, au sein d'un objet commercial, on a une référence à une liste de Clients



Association 1-N

On peut représenter ce champs en tant qu'association :



Association 1-N en code C#

- Le constructeur de Commercial attend donc une **liste** de Clients

```
public Commercial(string numSecu, string nom, string prenom,  
                  Voiture uneVoiture, List<Client> lesClients)  
{  
    this.NumSecu = numSecu;  
    this.Nom = nom;  
    this.Prenom = prenom;  
    this.UneVoiture = uneVoiture;  
    this.LesClients = lesClients;  
}
```

- Lorsqu'on instancie un commercial, il faut avant tout instancier une liste :

```
Voiture voiture = new Voiture("EL-05-3RE", "Peugeot", "3008");  
List<Client> clients = new List<Client>();  
clients.Add(new Client("Auchan", "zone grand Epagny", "74330", "Epagny"));  
clients.Add(new Client("Carrefour", "rue du P rim tre", "74000", "Annecy"));  
Commercial commercial = new Commercial("2771244", "Gruson", "Nathalie", voiture, clients);
```

Association 1-N en code C#

- On fera alors une surcharge sans liste. Attention : pensez à initialiser la liste :

```
public Commercial(string numSecu, string nom, string prenom, Voiture uneVoiture)
{
    this.NumSecu = numSecu;
    this.Nom = nom;
    this.Prenom = prenom;
    this.UneVoiture = uneVoiture;
    this.LesClients = new List<Client>();
}
```

- Et des méthodes d'ajout, suppression, modification de la liste. Ex:

```
public bool AddClient(Client unClient)
{
    if (unClient == null)
        throw new ArgumentNullException("Attention, pas d'ajout de client null !");
    if (this.LesClients.Contains(unClient))
        return false;
    this.LesClients.Add(unClient);
    return true;
}
```

Association 1-N en code C#

- Avec une surcharge possible :

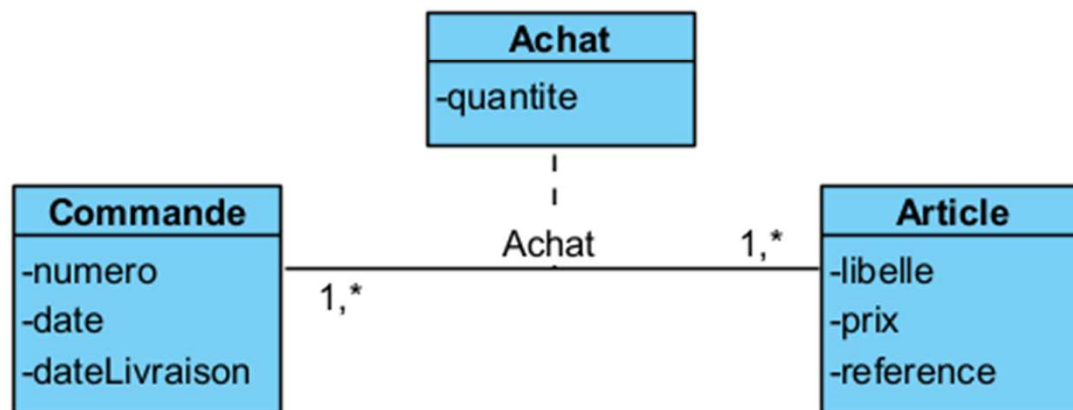
```
public bool AddClient(string nom, string adresse, string codePostal, string ville)
{
    Client unClient = new Client (nom,adresse,codePostal,ville);
    if (this.LesClients.Contains(unClient))
        return false;
    this.LesClients.Add(unClient);
    return true;
}
```

- Pour simplifier la construction dans le main : ici pas de gestion de liste, ni de création de client (déporté dans la classe)

```
Commercial commercial = new Commercial("2771244", "Gruson", "Nathalie", voiture);
commercial.AddClient("Auchan", "zone grand Epagny", "74330", "Epagny");
commercial.AddClient("Carrefour", "rue du Périmètre", "74000", "Annecy");
```

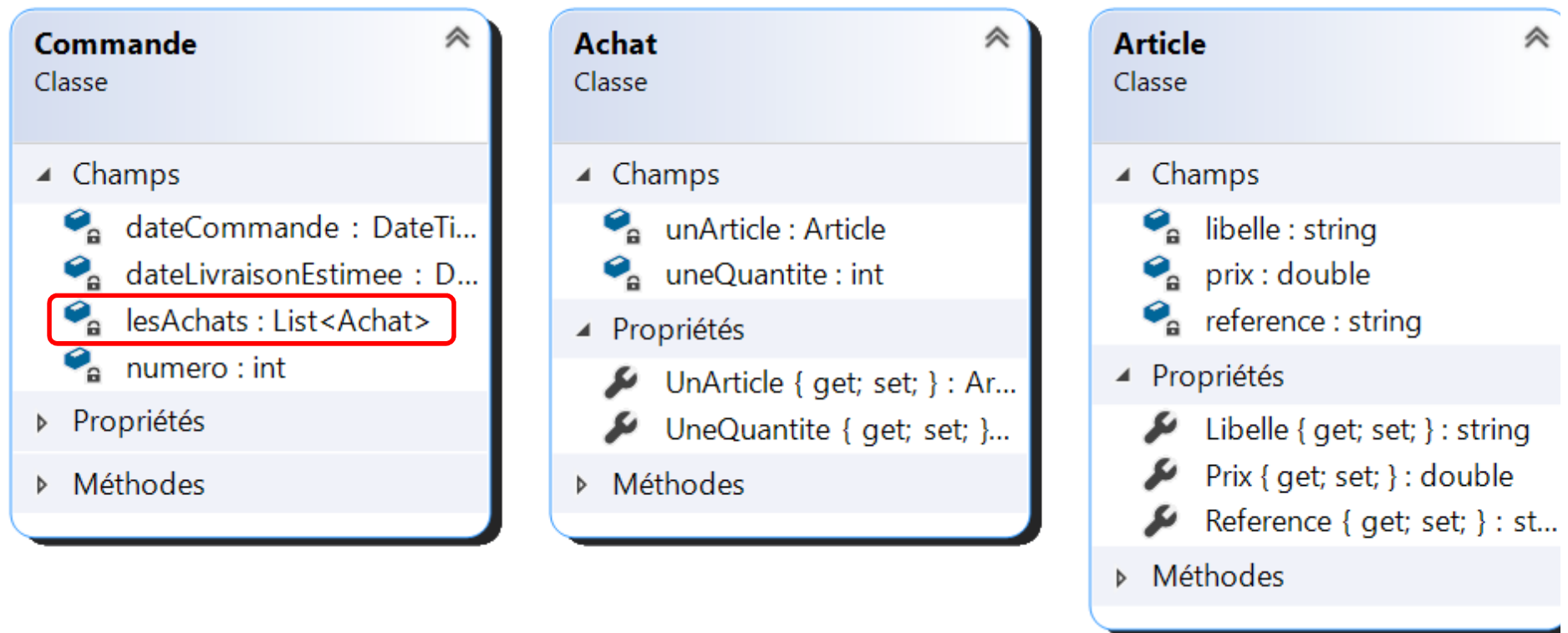
Association avec classe portée

Exemple : Une commande : peut regrouper l'achat d'un ou plusieurs articles avec une certaine quantité. La quantité dépend de la commande et de l'article, on l'indique alors en classe portée.



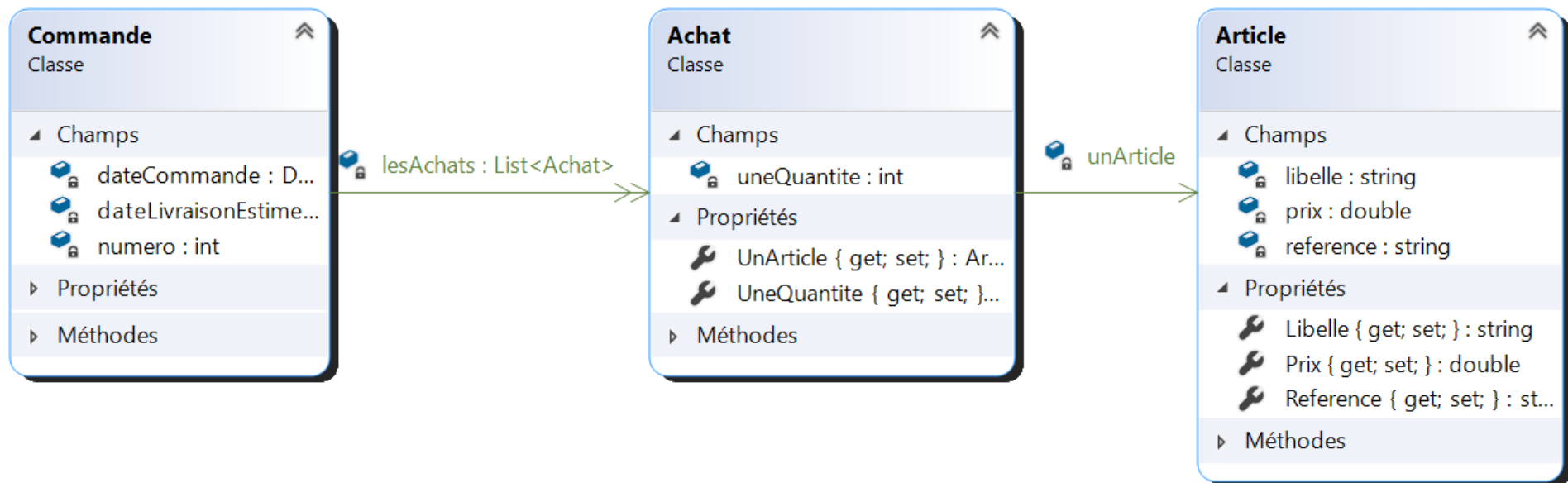
Association 1-N

Le diagramme sous visual sera plus proche du code final. La classe Achat sera une classe intermédiaire entre Commande et Article



Association avec classe portée

Le diagramme sous visual sera plus proche du code final



Association 1-N

Ici, pour instancier une commande, il faudra déjà instancier une liste d'achats

- Constructeur

```
public Commande(int numero, DateTime dateCommande, DateTime
dateLivraisonEstimee, Client leClient, List <Achat> lesAchats)
{
    this.Numero = numero;
    this.DateCommande = dateCommande;
    this.DateLivraisonEstimee = dateLivraisonEstimee;
    this.LeClient = leClient;
    this.LesAchats = lesAchats;
}
```

```
Client unClient = new Client("Gruson", "Nathalie", "", "nathalie@gmail.com", "59 impasse
....", "74330", "Epagny");
```

```
List <Achat> desAchats = new List <Achat> ( );
desAchats.Add( new Achat(1, new Article("ER456", "Enceinte bluetooth", 99.99));
```

```
Commande cmd = new Commande(1, DateTime.Today, DateTime.Today.AddDays(7),
unClient, desAchats);
```

Association 1-N

On permet alors la création d'une commande sans avoir à créer une liste d'achat

- Surcharge allégée

```
public Commande(int numero, DateTime dateCommande, DateTime
dateLivraisonEstimee, Client leClient )
{
    this.Numero = numero;
    this.DateCommande = dateCommande;
    this.DateLivraisonEstimee = dateLivraisonEstimee;
    this.LeClient = leClient;
    this.LesAchats = new List<Achat> ( );
}
```

Et on initialise la liste

```
Client unClient = new Client("Gruson", "Nathalie", "", "nathalie@gmail.com", "59 impasse
....", "74330", "Epagny");
```

```
Commande cmd = new Commande(1, DateTime.Today, DateTime.Today.AddDays(7),
unClient);
```

Association 1-N

- Et on ajoute une méthode d'ajout à la liste:

```
public class Commande()  
{  
    public void AjouteUnAchat(int quantite, String reference, string libelle, double prix)  
    {  
        this.LesAchats.Add(new Achat(quantite, new Article ( reference, libelle, prix)));  
    }  
}
```

```
Client unClient = new Client("Gruson", "Nathalie", "", "nathalie@gmail.com", "59 impasse  
....", "74330", "Epagny");
```

```
Commande cmd = new Commande(1, DateTime.Today, DateTime.Today.AddDays(7),  
unClient);
```

```
cmd.AjouteUnAchat(2, "erzr12", "ecouteur bluetooth", 15.99);
```