POO

Accessibilité d'un champ

- Jusqu'à présent, on a fait des champs privés avec un accès en lecture/écriture via des propriétés publiques, mais un champ peut être :

Modificateur	Accessibilité
private	Uniquement accessible au sein de la classe
public	Accesible partout
internal	Uniquement accessible au sein du projet
protected	Accesible dans la classe et les classes filles

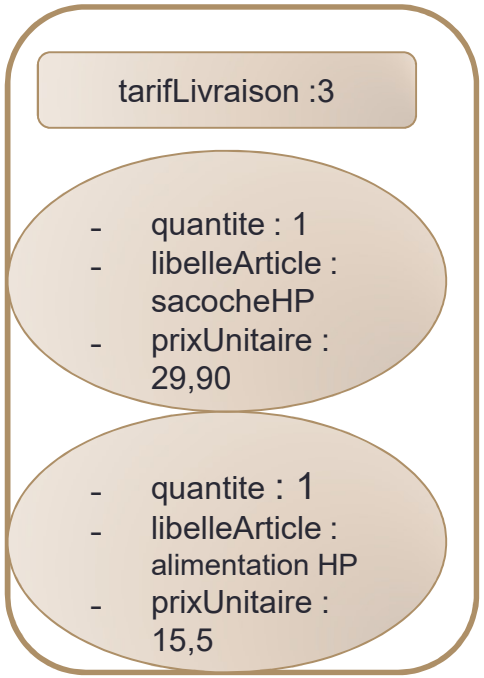
Champs statiques

- Un champ statique : pour stocker une donnée commune à tous les objets de la classe ! Stockée une seule fois en mémoire, dissociée des objets !

```
class Commande
{
   private static double tarifLivraison = 3;

   public static double TarifLivraison
       {
           get { return Commande.tarifLivraison; }
           set { Commande.tarifLivraison= value; }
       }
```

```
Commande c1 = new Commande(1,"alimentation HP", 15.5);
Commande c2 = new Commande(1,"sacoche HP", 29.90);
Console.WriteLine(Commande.TarifLivraison);
Commande.TarifLivraison = 5;
```



On y accède depuis la classe

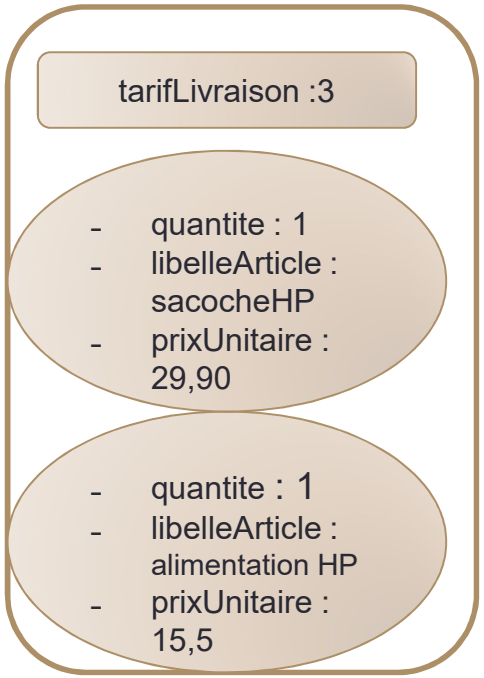
Champs statiques

- On peut jouer sur son accessibilité. Ici, seule la classe pourra modifier sa valeur

```
class Commande
{
   private static double tarifLivraison = 3;

   public static double TarifLivraison
       {
           get { return Commande.tarifLivraison; }
           private  set { Commande.tarifLivraison= value; }
       }
```

```
Commande c1 = new Commande(1,"alimentation HP", 15.5);
Commande c2 = new Commande(1,"sacoche HP", 29.90);
Console.WriteLine(Commande.TarifLivraison);
```



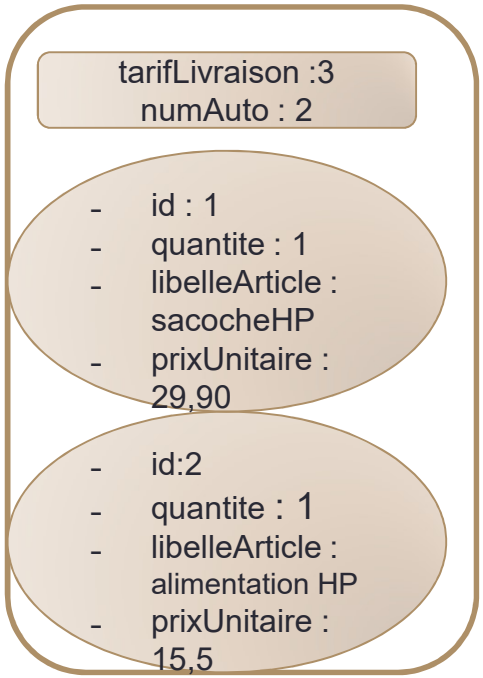
On y accède depuis la classe

Champs statiques

- Très utile aussi pour faire une donnée auto incrémentée : un identifiant.

```
class Commande
  {
    private static int numAuto = 0;

      public static int NumAuto
      {
        get {
            numAuto++;
            return Commande.numAuto;
          }
        set { Commande.numAuto = value; }
      }
public Commande(String pLibelle, double pPrixUnit, int pQuantite)
      {
        Id = NumAuto;
        LibelleArticle = pLibelle;
        PrixUnitaire = pPrixUnit;
        Quantite = pQuantite;
      }
```



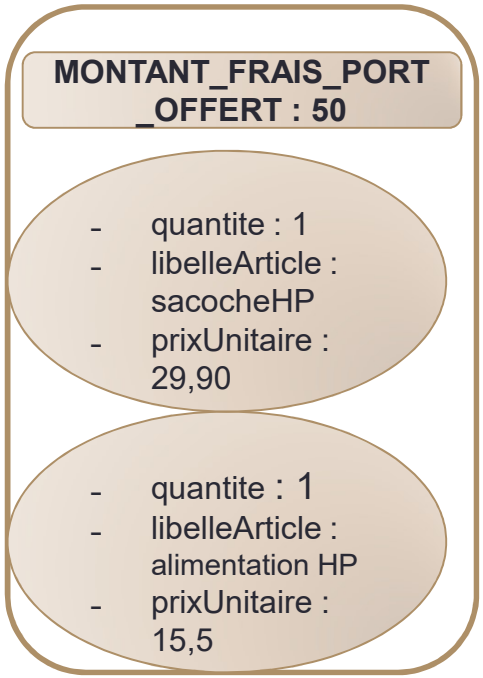
Constantes (champs implicitement statiques)

- Une constante est un champ statique qui ne peut pas être modifié : elle est souvent publique.

```
class Commande
{
  public const double MONTANT_FRAIS_PORT_OFFERT = 50;
```

```
Commande c1 = new Commande(1,"alimentation HP", 15.5);
Commande c2 = new Commande(1,"sacoche HP", 29.90);
Console.WriteLine(Commande. MONTANT_FRAIS_PORT_OFFERT);
```

On y accède depuis la classe



Champs statiques / champs d'instance

- Par défaut, les champs sont d'instances.

Champ d'instance	Champ statique
Donnée propre à un objet	Donnée commune à tous les objets
Appartient à l'objet	Appartient à la classe
Initialisée au sein du constructeur	**Initialisée dès sa déclaration**

Enum

- Pour définir une énumération

```
public enum SexeOfIndividu { F ='F', M ='M'};

private SexeOfIndividu sexe;
public SexeOfIndividu Sexe
        {
            get { return sexe; }
            set { sexe = value;
            }
        }
```

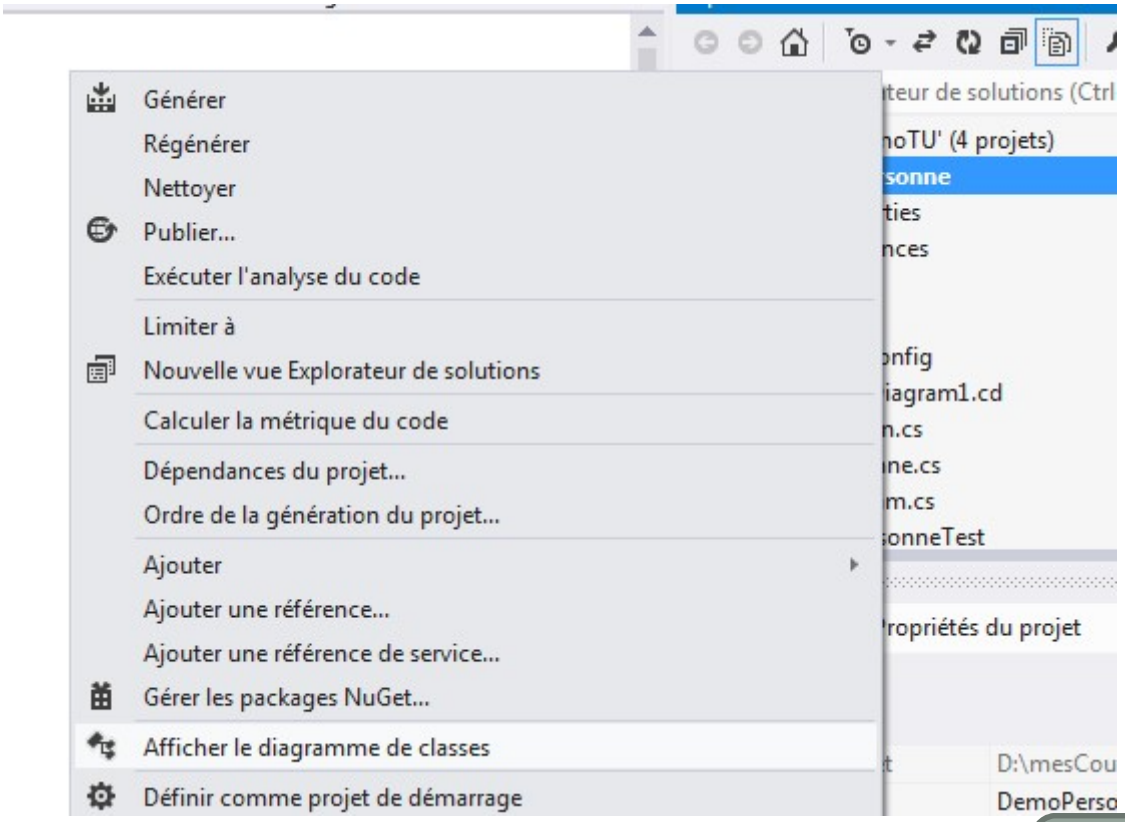
- Et éviter cela :

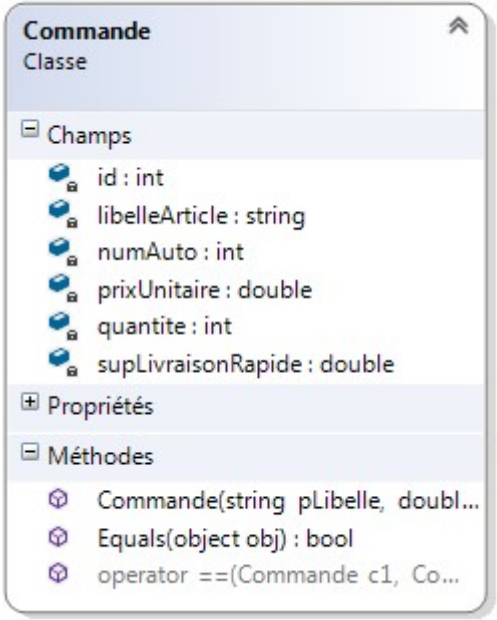
```
public const char SEXE_F ='F';
public const char SEXE_M ='M';

private char sexe;
public char Sexe
       {
           get { return sexe; }
           set { if ( value != SEXE_F && value != SEXE_M )
                        throw new ArgumentException (« valeur inattendue »);
                 sexe = value ;
                }
```

Générer un diagramme UML : **rétro ingénierie**

- Visual studio peut générer le diagramme de classe à partir de votre code.

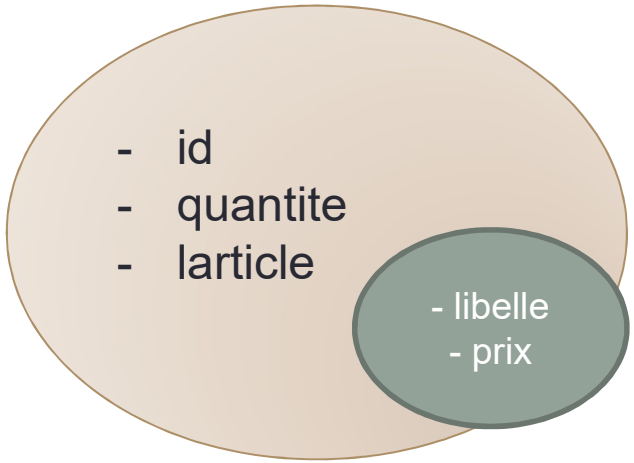


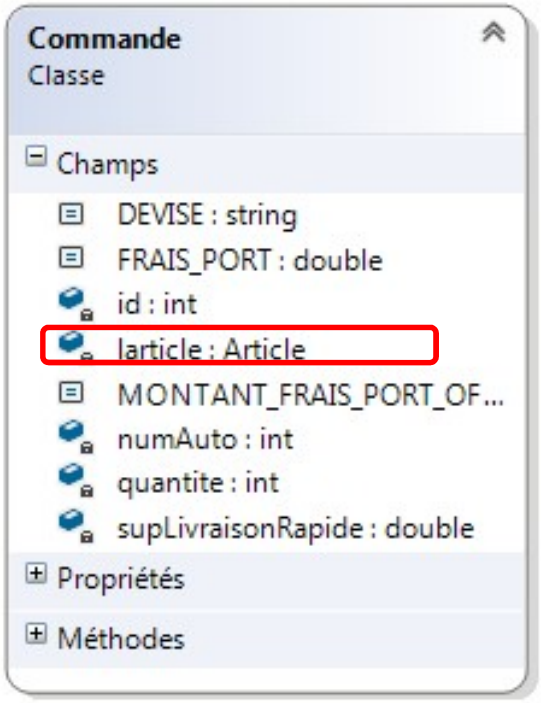


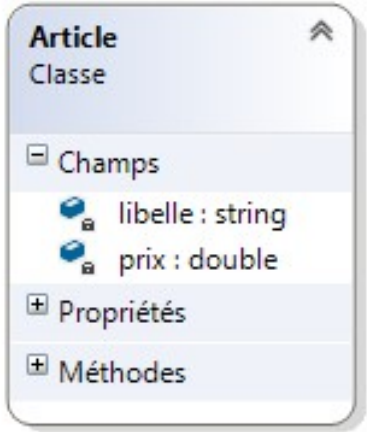
Attention : toute modification entraîne une modification du code

Association simple

- Un objet composé d'un autre objet !

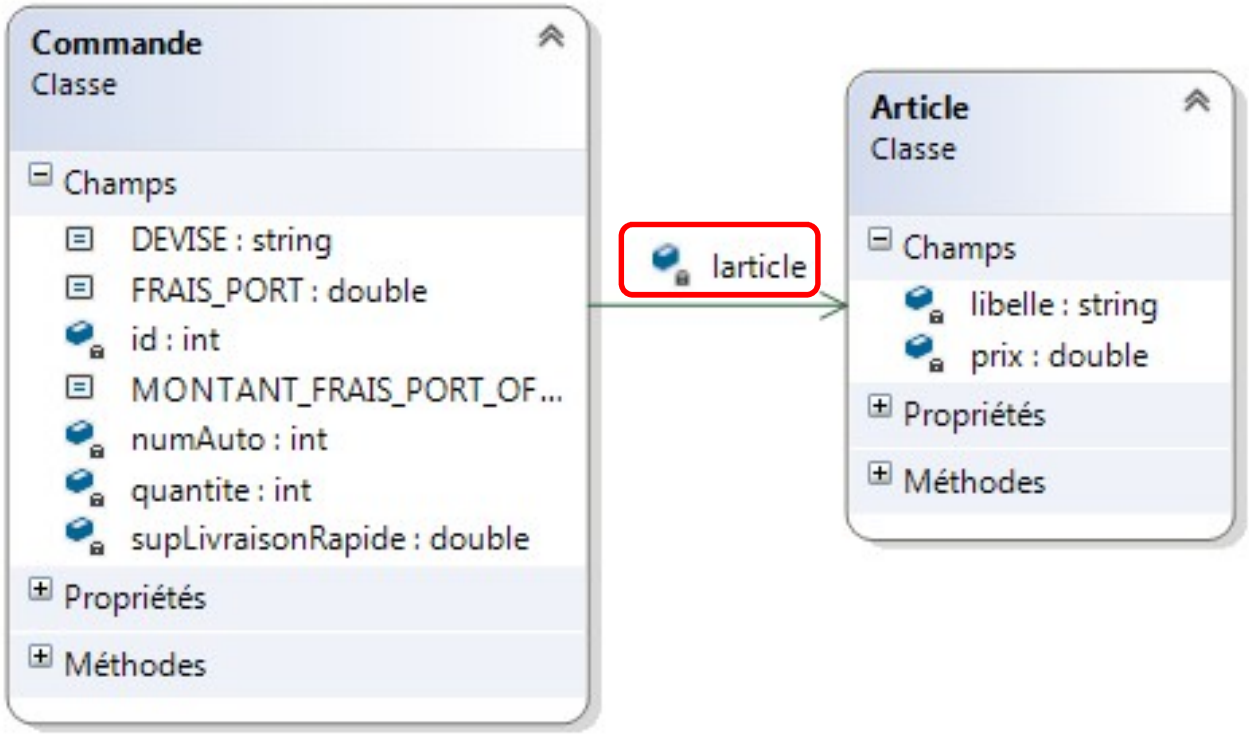






Association simple

- Le champ peut être représenté sous forme d'association.



Association simple

- Généralement, on fera alors une surcharge du constructeur:
  - Une version avec l'objet passé en paramètre

```
public Commande(Article a , int pQuantite)
      {
         Id = NumAuto;
         Quantite = pQuantite;
         Larticle = a;
      }
```

```
Article a = new Article ( "Alimentation HP", 15.5);
Commande c = new Commande(a, 1);
```

Pour instancier une commande, il faudra déjà instancier l'article

  - Une version avec les données nécessaires à la création de l'objet en interne

```
public Commande(String pLibelle, double pPrixUnit, int pQuantite)
      {
         Id = NumAuto;
         Quantite = pQuantite;
         Larticle = new Article(pLibelle, pPrixUnit);
      }
```

```
Commande c = new Commande("Alimentation HP", 15.5, 1);
```

Association simple

- Rappel pour factoriser le code

```
public Commande(Article a , int pQuantite)
      {
         Id = NumAuto;
         Quantite = pQuantite;
         Larticle = a;
      }
public Commande(String pLibelle, double pPrixUnit, int pQuantite)
      {
         Id = NumAuto
         Quantite = pQ
         Larticle = new
      }
```

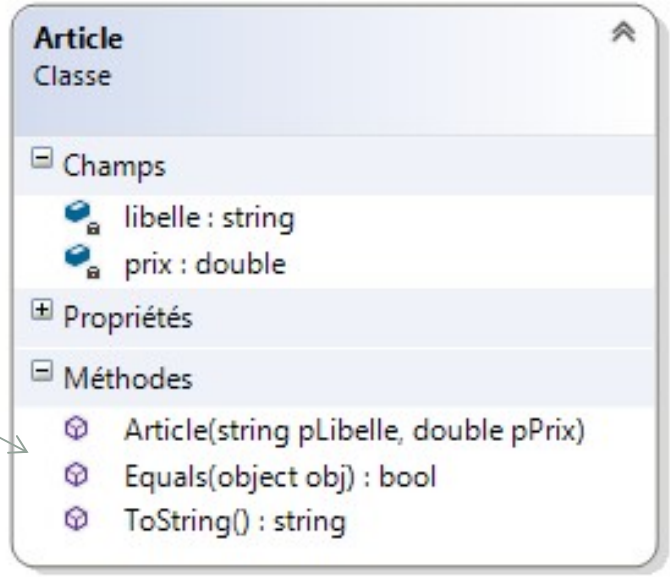
```
public Commande(Article a , int pQuantite)
      {
         Id = NumAuto;
         Quantite = pQuantite;
         Larticle = a;
      }
public Commande(String pLibelle, double pPrixUnit, int pQuantite) :
this (new Article(pLibelle, pPrixUnit) , pQuantite );
      {  }
```

Association simple

- On pourra déclencher au niveau de la classe Commande, les méthodes de la classe Article par l'intermédiaire de l'objet larticle

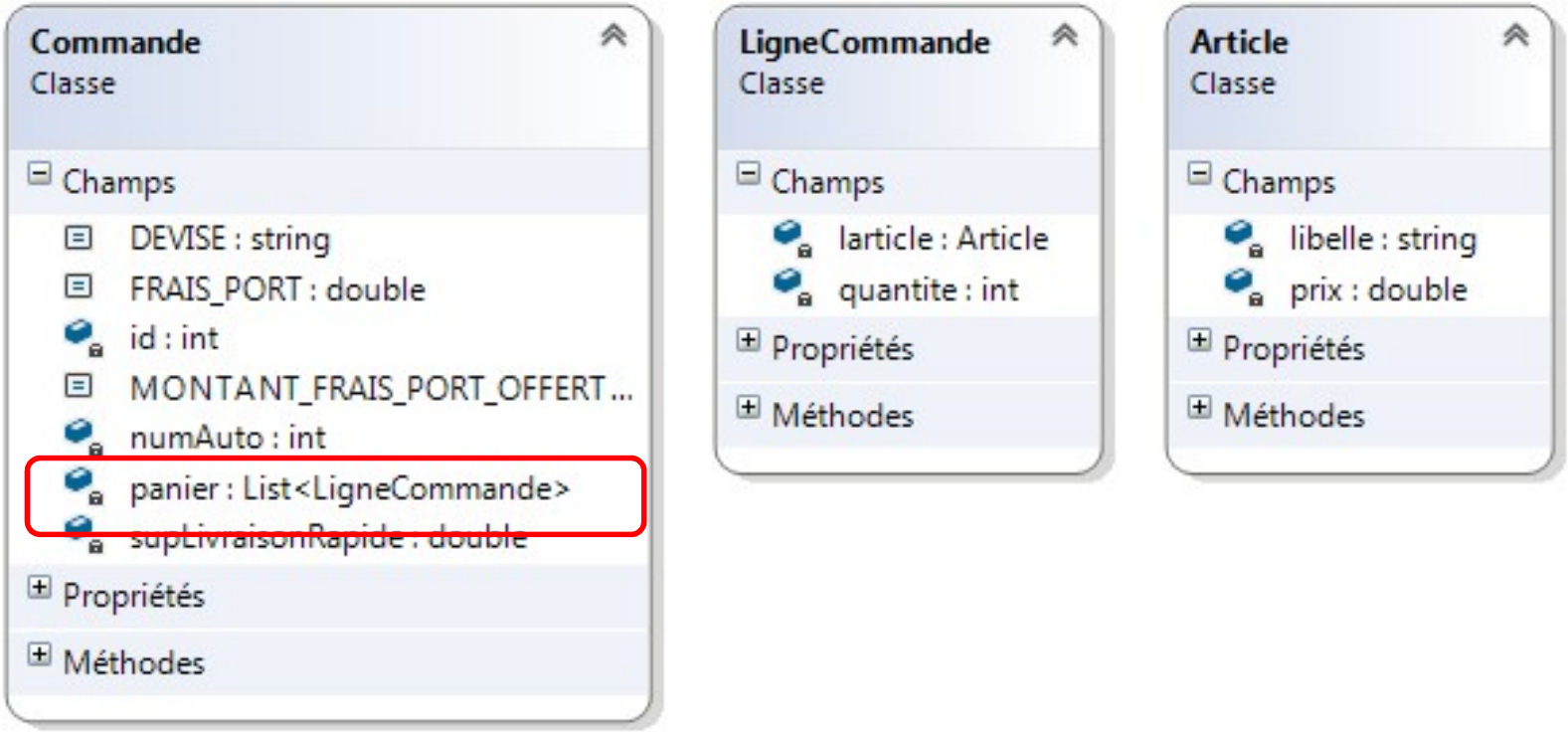
```
public override bool Equals(object obj)
      {
        if  (obj == null )
            return false;
        if ( obj.GetType() != GetType() )
           return false ;
        Commande c = obj as Commande;
        return Larticle.Equals(c.Larticle)
                    && Quantite == c.Quantite;

      }
```



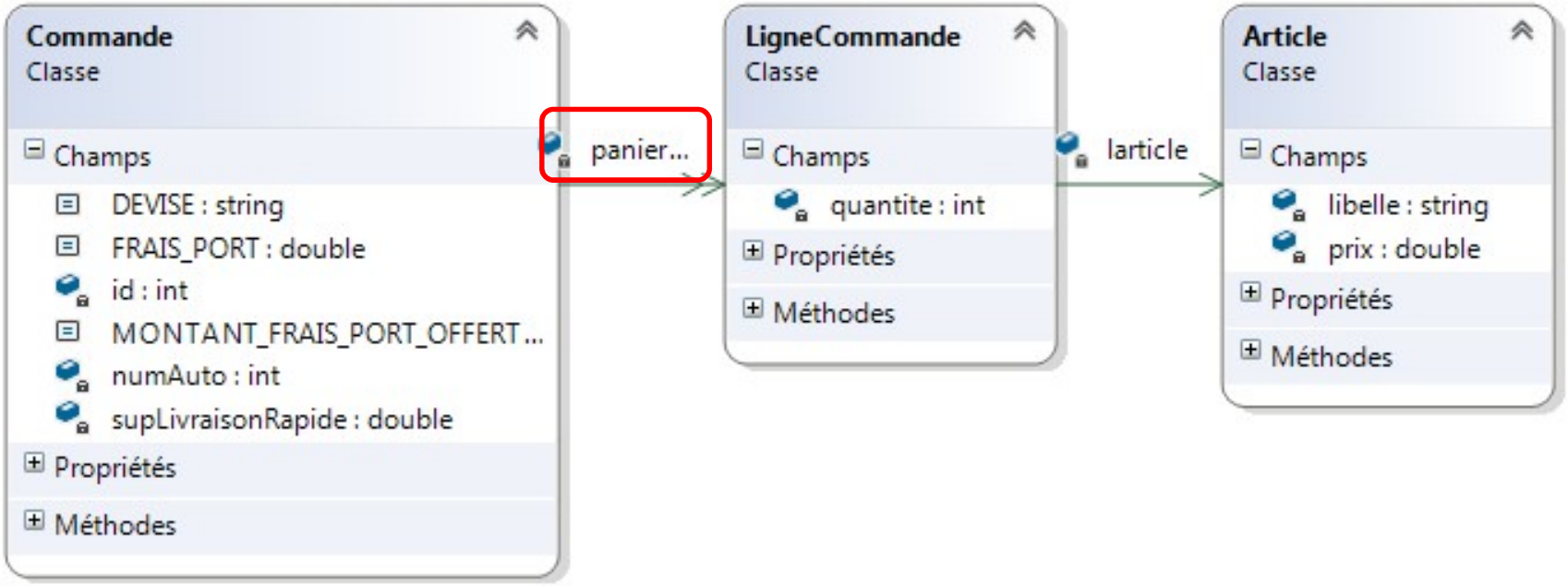
Association multiple

- Les variables peuvent être une collection d'objets.



Association multiple

- La collection peut être représentée sous forme d'une association



Association multiple

- Généralement, on fera alors une surcharge du constructeur:
  - Une version avec la collection passée en paramètre

```
public Commande(List<LigneCommande> l)
      {
          Id = NumAuto;
          panier = l;
      }
```

```
List<LigneCommande> l = new List<LigneCommande>();
l.Add(new LigneCommande(1, new Article("Alimentation HP", 15.5)));
l.Add(new LigneCommande(2, new Article("Sacoche HP", 29.9)));
Commande c = new Commande(l);
```

Pour instancier une commande, il faudra déjà instancier une liste de lignes de commande

Association multiple

- Généralement, on fera alors une surcharge du constructeur:
  - Une version « allégée »

```
public Commande()
      {
          Id = NumAuto;
          panier = new List<LigneCommande>();
      }
public void AjouteUneLigne( int pQuantite, String pLibelle, double pPrix)
      {
          panier.Add(new LigneCommande(pQuantite, new Article(pLibelle, pPrix)));
      }
```

```
Commande c = new Commande();
 c.AjouteUneLigne(1, "Alimentation HP", 15.5);
 c.AjouteUneLigne(2, "Sacoche HP", 29.9);
```

+ facile à utiliser !

Association multiple

- Il ne faudra pas oublier que cette variable est une collection de données : penser aux boucles. Exemple avec ToString :

```
public override string ToString()
    {
        String txt = "\nCommande n°: " + id;
        foreach ( LigneCommande l in panier)
        { txt = txt + l.ToString(); }
        return txt;
    }
```