SEQUENCE 2 – ASSOCIATION

SEANCES 1 ET 2 : ENTREPOT DE MEUBLES

OBJECTIFS

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

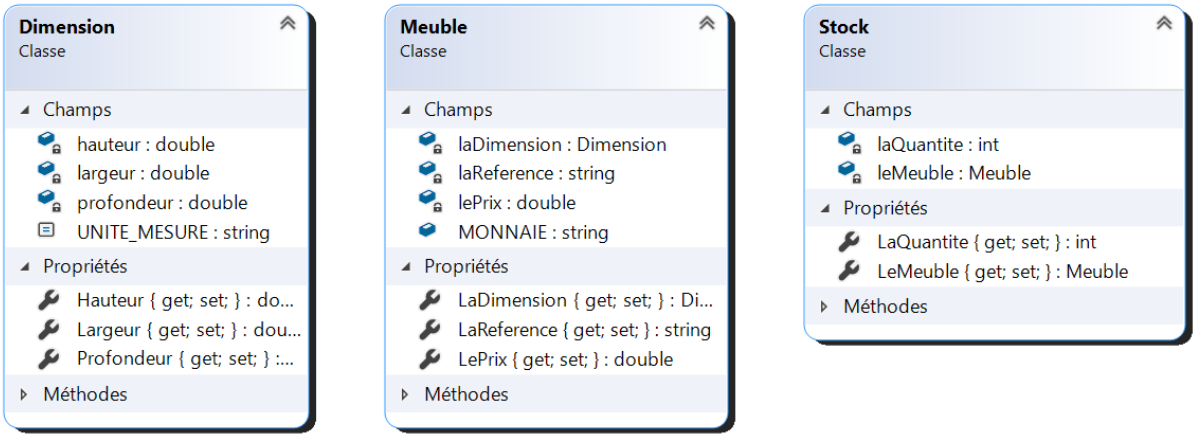
CONSIGNES

Au sein de votre répertoire « **R2_01_Approfondissement_C# »** , créez un sous répertoire « **Sequence_2_Association** » Créez un projet « **Partie1_GestionStocks »** dans une solution « **Seance_EntrepotMeubles** ».

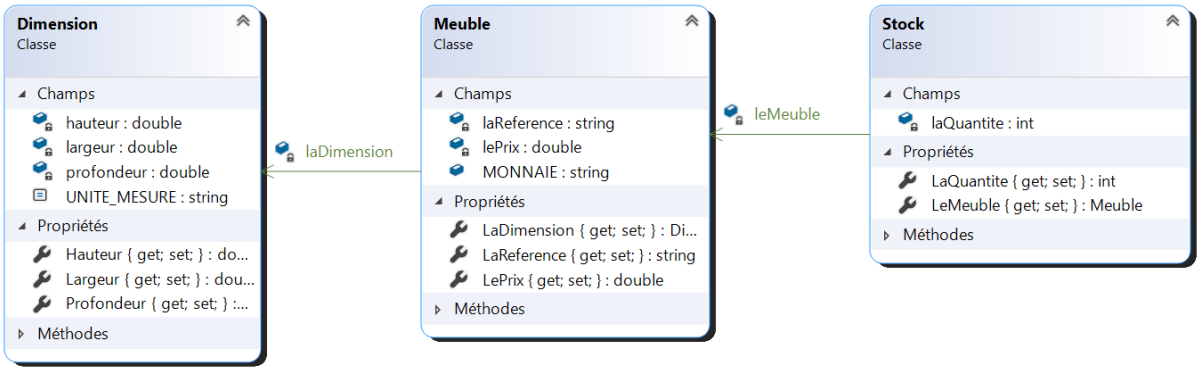
PARTIE 1 : GESTION DE STOCKS DANS LE MAIN

Ici, un stock correspond au stock d'un meuble en particulier, un meuble a une dimension une référence et un prix. Sous l'ide, on peut afficher le diagramme de 2 manières différentes :

- Avec tous les champs (mais pas de liens entre les classes)



- Avec les liens entre les classes : les champs faisant le lien ne sont alors plus représentés au sein de la classe.



Attention : ici, UNITE_MESURE et MONNAIE sont des constantes : non modifiables et surtout du coup, static, ce qui signifie 1 seule valeur possible pour toutes les instances.

```
public const String UNITE_MESURE = "cm";
public const String MONNAIE = "euro(s)";
```

1. Définissez les 3 classes ci-dessus : commencez par la classe qui ne dépend d'aucune autre ! Vous **générerez** en quelques minutes avec les assistants : propriétés (contrôle sur la hauteur, largeur, profondeur, lePrix, laQuantite >=0) , constructeur, méthodes substituées : Equals, GetHashCode. Et définissez les ToString de manière à avoir ce genre d'affichage pour un stock :

```
Reference : RAZA
Prix : 59,99 euro(s)
Dimension : 200x120x55cm
Quantite : 100
```

2. Instanciez un stock de 100 meubles « RAZA » d'un prix de 59.99 € ayant une largeur de 200 cm , une hauteur de 120 cm et une profondeur de 55cm. Puis testez l'affichage. Pour faire cela, vous devez créer une dimension avant de créer un meuble, et créer un meuble avant le stock !

3. Surchargez le constructeur afin de pouvoir créer un meuble sans avoir à créer de dimension au préalable. Il s'agit ici d'une relation d'agrégation (voir de composition) entre les 2 classes : la dimension sera créée lors de la construction du meuble en interne. La dimension est une partie du meuble.

   Ex : `Meuble mRaza = new Meuble(200, 120, 55, "RAZA", 59.99)`

4. Surchargez le constructeur Stock afin de pouvoir créer un stock sans avoir à créer un meuble au préalable. Utilisez ce nouveau constructeur pour créer un nouveau stock pour un autre meuble :

```
Ex : Stock stockTikka = new Stock(120, 100, 50, 45, "TIKKA", 45.99));
// 120 meubles TIKKA de dimension 100 *50 *45 au prix de 45.99 euros
```

5. Dans la classe Stock, définissez :
   a. void AjouteStock (int nb) : ajoute le nb indiqué à la quantité déjà en stock
   b. bool RetireStock (int nb) : retire le nb indiqué à la quantité déjà en stock si cela est possible et renvoie true, renvoie false si la quantité en stock n'est pas suffisante pour être retirée.
6. Au sein de la classe Stock, codez les méthodes et vérifiez que vous obtenez bien le jeu de test donné ci-dessous :
   a. CalculeMontant: elle retourne le montant d'un stock.
   b. CalculeVolume : elle retourne le volume total en m3. Ajoutez une unité de mesure du volume en m3 dans la classe Dimension. Attention, à bien répartir les calculs au sein des différentes classes.

Données en entrée	**Résultats attendus**	
Stock	CalculePrix	CalculeVolume

• uneQuantite : 120
• unMeuble : 100 x50x45
  • uneReference : TIKKA
  • unPrix : 45.99
	5 518.80 €	100 X 50 X 45 x 120 / 1 000 000 = 27 m3

7. **Assurez-vous que toutes vos classes et propriétés soient publiques et qu'elles contiennent un constructeur par défaut.**

8. Au sein du programme, on veut manipuler une liste de stocks. Cette liste est chargée à partir d'un fichier .json grâce à une méthode générique dont la signature est la suivante :
```
private static List<T> Charge<T>(String pathName)
```
Son code ainsi que celui du menu est donné après :

```
0.Quitter
1.Voir tous les stocks
2.Voir les stocks épuisés
3.Info stocks : montant et volume
4.Voir/modifier un stock
5.Créer un nouveau stock
----------------------
```

Remarques : dans le menu, pour :

- l'option 2 : pensez à afficher un message s'il n'y a pas de stocks épuisés
- l'option 3 : affichez le montant total et le volume total des stocks ainsi que le détail du calcul comme ci-dessous :

```
----------------------
INFO STOCKS
----------------------
MONTANT
----------------------
RAZA 100 x  59,99 =  5 999,00
TIKKA 120 x  45,99 =  5 518,80
TIKKI 10 x  125,90 =  1 259,00
RATUM 0 x  76,00 =  0,00
----------------------
Montant total : 12 776,80 euro(s)
----------------------
VOLUME
----------------------
RAZA 100 x  1,32 =  132,00
TIKKA 120 x  0,23 =  27,00
TIKKI 10 x  0,18 =  1,80
RATUM 0 x  0,08 =  0,00
----------------------
Volume total : 160,80 m3
```

- l'option 4 : Demandez à l'utilisateur la référence du meuble dont il faut modifier le stock. Si elle n'existe pas dans la liste, l'avertir sinon demandez lui s'il faut faire un ajout ou un retrait puis utilisez vos méthodes AjouteStock et RetireStock

```
// CHARGEMENT DES STOCKS
   String pathName = "stocks.json" ;
   List<Stock> lesStocks = null ;
   bool encore = true;
   try
     {
      lesStocks = Program.Charge<Stock>(pathName);
     }
    catch (Exception e)
     {
               Console.WriteLine("Probleme lors du chargement des stocks ");
               Console.WriteLine(e);
               Environment.Exit(2);
      }
  while (encore)
    {
      Console.Clear();
      Console.WriteLine("----------------------");
      Console.WriteLine("0.Quitter");
      Console.WriteLine("1.Voir tous les stocks");
      Console.WriteLine("2.Voir les stocks épuisés ");
      Console.WriteLine("3.Info stocks : montant et volume ");
      Console.WriteLine("4.Voir/modifier un stock ");
      Console.WriteLine("----------------------");
      int  rep = SaisieInt(0,4) ;
       switch (rep)
           {
             case 0: encore = false; break;
             case 1:
                       {
                           Console.WriteLine("----------------------");
                           Console.WriteLine("TOUS LES STOCKS");
                           Console.WriteLine("----------------------");
                  break;
                       }
            case 2: break;
            case 3: break;
            case 4: break;
          }
        // pour faire une pause avant le clear
        Console.ReadLine();
       }

     // SAUVEGARDE DES STOCKS : A FAIRE
     }
  public static void AfficheListe (List<Stock> liste)
      {
          foreach (Stock unStock in liste)
              Console.WriteLine(unStock);
      }

  private static List<T> Charge<T>(String pathName)
       {
           List<T> lesInfos = null;
           try
           {
            String contenuFichier = File.ReadAllText(pathName);
            lesInfos = JsonConvert.DeserializeObject<List<T>>(contenuFichier);
           }
           catch (Exception e) { throw e; } //on relance l'exception
           //à celui qui appelle la méthode : ici le main pour
           // qu'il décide de l'action à faire
           return lesInfos;
       }
```

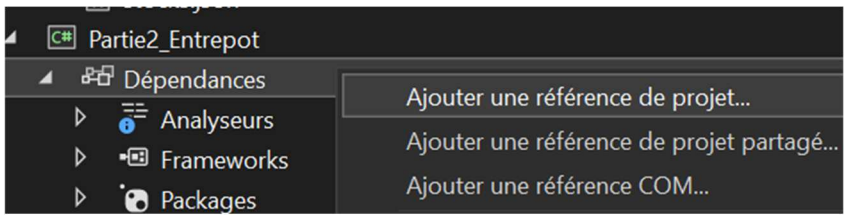
```
public static int SaisieInt(int min, int max)
   {
    int nb =0; bool ok;
    String choix = Console.ReadLine();
    do
       {
         ok = true;
          if (!(int.TryParse(choix, out nb) && nb >= min && nb <= max))
                {
                    Console.WriteLine($"Erreur- Choix entre {min} et {max} :");
                    choix = Console.ReadLine();
                    ok = false;
                }
         } while (!ok);
       return nb; }
```

9. Programmez la sauvegarde : inspirez-vous de la méthode générique de chargement :

```
private static void Sauvegarde<T>(List<T> lesInfos, String pathName)
```

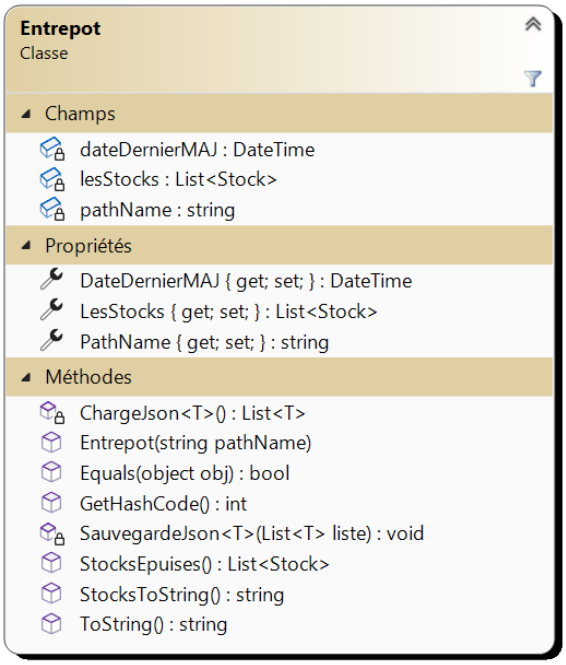
PARTIE 2 : GESTION DE STOCKS DANS ENTREPOT

1. Ajoutez un nouveau projet « **Partie2_Entrepot** **» à votre solution.**
2. Ajoutez à ce projet une référence au projet précédent afin de pouvoir utiliser les classes Stock, Meuble et Dimension. Attention : ces classes doivent être publiques pour être utilisée dans un autre projet.



3. Reprenez le code (copier/coller) de votre ancien Program.cs ajoutez-y le using suivant : `using Partie1_GestionStocks;`

4. Maintenant, allégez le code du programme principal : il n'y aura plus de traitement sur la liste de Stocks, car celle-ci sera définie et gérée au sein de la classe Entrepot. Le programme principal ne doit plus que contenir (ou presque) les instructions d'affichage et de saisie ! Ce n'est plus que l'intermédiaire entre utilisateur final et la classe Entrepot. On pourra ainsi aisément envisager si le besoin s'en fait sentir la possibilité de gérer une liste d'entrepôts …



La classe Entrepot est à compléter : il manque des méthodes

- Le constructeur Entrepot : initialise la liste des stocks à partir d'un chargement de données liées au fichier indiqué par le pathName et met pour date de dernière mise à jour la date du moment (DateTime.Now)
- La méthode ToString : retourne un résumé
- La méthode TousLesStocksToString : retourne une chaîne constituée de tous les stocks : utile pour l'option 1 du menu
- La méthode StocksEpuises : retourne une liste des stocks epuisés.
- A compléter …

5. On doit souvent récupérer le prix et le volume de l'entrepôt, faire appel aux méthodes de calcul peut être couteux (parcours systématique de la liste). Quelle solution pourriez-vous proposer ?