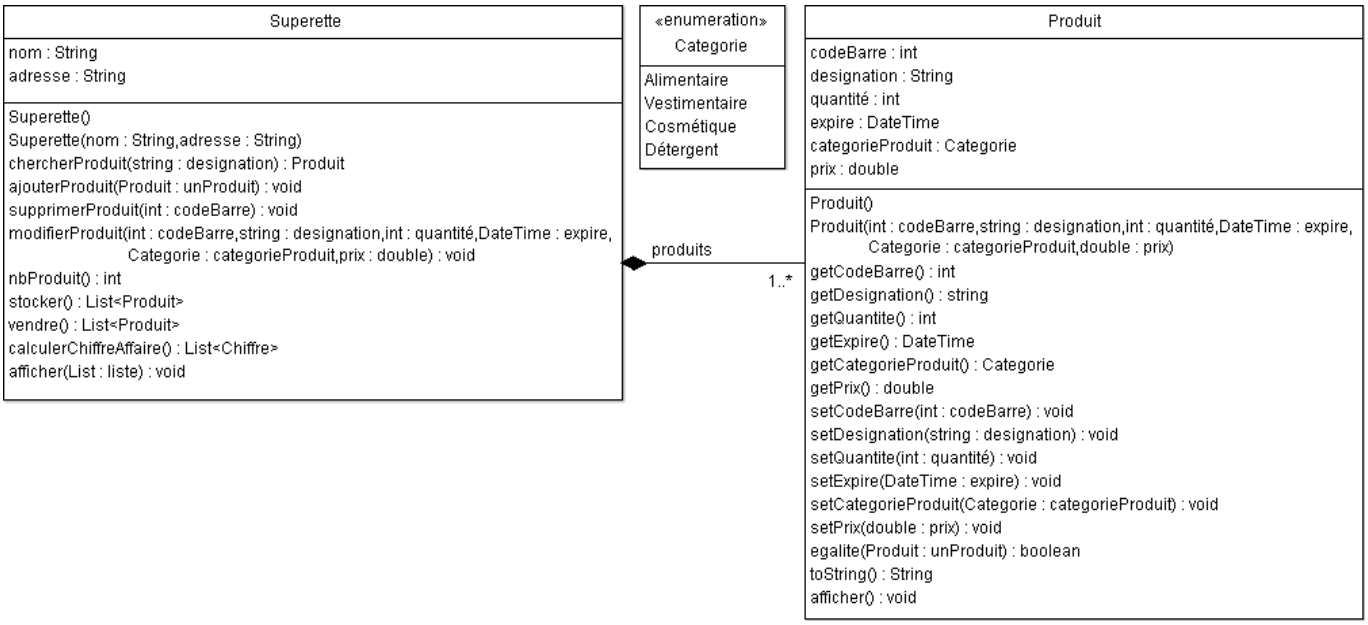
# Programmation Objets avec C# – TP1 –

**I/ Objectif du TP**

Le but de ce TP est de faire une révision des concepts de base la programmation par objets avec C# à travers l'implémentation d'une **application Console** pour sa première partie et d'une **application Windows Forms** pour sa deuxième partie.

## Partie I/

Considérant le diagramme de classes suivant relatif à une supérette vendant plusieurs catégories de produits :




**II/ Implémentation de la classe « `Produit` »**

Implémentez la classe `Produit`, qui représente un produit avec les attributs suivants :

**II.1/ Six variables d'instance** :

1. `codeBarre` : un nombre entier pour identifier un `Produit` ;
2. `designation` : une chaîne string pour désigner le nom du `Produit` ;
3. `quantité` : un nombre entier pour désigner la quantité en stock du `Produit` ;
4. `expire` : une date pour désigner la date d'expiration du `Produit` ;
5. `categorieProduit` : un type `Categorie` qui est une énumération ayant les valeurs "`Alimentaire`", "`Vestimentaire`", "`Cosmétique`", "`Détergent`" ou "`Rien`" ;
6. `prix` : nombre en double déterminant le prix de vente du `Produit`.

# Programmation Objets avec C# – TP1 –

**II.2/ Deux constructeurs** :

1. `Produit(int codeBarre , string designation, int quantité, DateTime expire, Categorie categorieProduit, double prix)` : constructeur avec valeurs initiales explicites pour un livre donnée.
2. `Produit()` : constructeur sans arguments, initialisant les attributs aux valeurs par défaut.

**II.3/ Douze accesseurs**

1. **Six getters**
   - `int getCodeBarre()`: accesseur en lecture, renvoie le code du produit
   - `string getDesignation()`: accesseur en lecture, renvoie le nom du produit
   - `int getQuantité()`: accesseur en lecture, renvoie la quantité en stock du produit
   - `Categorie getCategorieProduit()`: accesseur en lecture donnant la catégorie du produit
   - `DateTime getExpire()` : accesseur en lecture donnant la date d'expiration du produit
   - `double getPrix()` : accesseur en lecture, renvoie le prix de vente du produit

2. **Six setters**
   - `void setCodeBarre (int codeBarre)`: accesseur en écriture, change le code du produit
   - `void setDesignation (string designation)` : accesseur changeant le nom du produit
   - `void setQuantité (string quantité)` : accesseur changeant la quantité du produit
   - `void setCategorieProduit (Categorie categorieProduit)`: accesseur changeant la catégorie produit
   - `void setExpire (DateTime expire)` : accesseur changeant la date d'expiration du produit
   - `void setPrix (double prix)` : accesseur changeant le prix du produit

**II.4/ Propriétés**

Six propriétés pour accéder aux attributs de la classe `Produit` en lecture et en écriture, au lieu d'utiliser des accesseurs :

```
1. public int CodeBarre,
2. public string Designation,
3. public int Quantité,
4. public DateTime Expire,
```

# Programmation Objets avec C# – TP1 –

5. `public Categorie CategorieProduit,`
6. `public double Prix.`

### II.5/ Trois méthodes d'instance

1. `bool egalite(Produit unProduit)` : vérifie l'égalité des prix du produit courant avec le produit `unProduit` et rend `true`, si les deux prix sont égaux ou `false`, sinon ;
2. `String toString()` : retourne une chaîne de caractères représentant l'état du `Produit`. Par exemple " **1, 'Gâteau Bomo', 300, 2024, Alimentaire, 150 DA** " pour un produit dont la code est 1, le nom est 'Gâteau Bimo' , dont la quantité en stock est de 300, expire en 2024 et il est vendu à 150 dinars.
3. `void afficher()` : affiche à l'écran la chaîne caractérisant un `Produit`.

## III/ Implémentation de la classe « `Superette` »

Implémentez la classe `Superette`, qui représente une superette avec les attributs suivants :

### III.1/ Trois variables d'instance :

7. `nom` : une chaîne string pour désigner le nom d'une `Superette` ;
8. `adresse` : une chaîne string représentant l'adresse de la `Superette` ;
9. `produits` : une `List<Produit>` des produits vendus par la superette ;

### III.2/ Deux constructeurs :

1. `Superette(string nom, string adresse, List<Produit> produits)` : constructeur avec valeurs initiales explicites pour une superette donnée.
2. `Superette()` : constructeur sans arguments, initialisant les attributs aux valeurs par défaut.

### III.3/ Six accesseurs

3. **Trois getters**
   - `string getNom()`: accesseur en lecture, renvoie le nom de la superette.
   - `string getAdresse()`: accesseur en lecture, donne l'adresse de la superette
   - `List<Produit> getProduits()`: accesseur en lecture renvoyant la liste des produits disponibles au niveau de la superette

# Programmation Objets avec C# – TP1 –

4. **Trois setters**
   - `void setNom (string nom)`: accesseur en écriture pour changer le nom de la superette
   - `void setAdresse(string adresse)` : accesseur modifiant l'adresse de la superette
   - `void setProduits (List<Produit> produits)`: accesseur affectant une nouvelle liste de produits à la superette

### III.4/ Propriétés

Trois propriétés pour accéder aux attributs de la classe `Superette` en lecture et en écriture, au lieu d'utiliser des accesseurs :

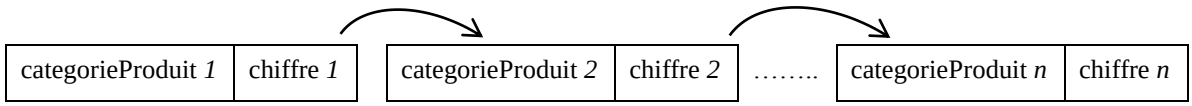
1. `public string Nom`,
2. `public string Adresse`,
3. `public List<Produit> Produits.`

### III.5/ Neuf méthodes d'instance

1. `Produit chercherProduit(string : designation)`: Recherche un produit dans la liste par son nom.
2. `void ajouterProduit(Produit : unProduit)`: Ajoute un nouveau produit à la liste.
3. `void supprimerProduit(int : codeBarre)`: Supprime un produit en le recherchant d'abord dans la liste.
4. `void modifierProduit(int : codeBarre,string : designation,int : quantité,DateTime : expire,Categorie : categorieProduit,prix : double)`: modifie les données relatives à un produit particulier.
5. `int nbProduit()`: compte le nombre de produits de la liste.
6. `List<Produit> stocker()` met dans une liste `stock` tous les produits dont le prix de vente dépasse 5000 DA ;
7. `List<Produit> vendre()`: met dans une liste `vente` tous les produits de la catégorie "Cosmétique", dont le prix de vente sera inferieur à 1000 DA et qui vont expirer en 2021 qui seront vendu en promotion.
8. `List<Chiffre>calculerChiffreAffaire()` : comptabilise les chiffres d'affaires qui seront enregistrés par la vente de chaque catégorie de produits.

   Une `List<Chiffre>` est une liste ayant la structure suivant :

# Programmation Objets avec C# – TP1 –




| **Exemple** | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | Alimentaire | 5690000 | Vestimentaire | 987000 | Cosmétique | 650000 | Détergent | 878000 |

**9.** `void afficher(List : liste)` : Affiche le contenu d'une liste.

### IV/ Construction de la classe contenant le Main

En utilisant les classes `Produit` et `Superette`, écrire la classe `Main` afin de tester toutes les méthodes implémentées. Calculer en plus le chiffre d'affaire global de la superette.

# Programmation Objets avec C# – TP1 –

## *V*/ Quelques notions en C# :

### *V.1*/ Propriété :

- Il existe une autre façon d'accéder aux attributs d'une classe à travers la notion de propriété.
- La propriété nous permet de manipuler des attributs privés comme s'ils étaient publics.
- La propriété remplace le getter et le setter d'un attribut.
- Une propriété permet de lire (`get`) ou de fixer (`set`) la valeur d'un attribut. Sa déclaration comme suit:

```
public Type Propriété {
        get { ... }
        set { ... }
}
```

où `Type` doit être le type de l'attribut géré par la propriété.

**Exemple :**

```
// Propriété
public string Nom {
    get { return nom; }
    set {
        // Vérifier si le nom est valide ?
        if (value == null || value.Trim().Length == 0) {
            throw new Exception("Le nom (" + value + ") est invalide");}
        else { nom = value;}
    } // Fin du set
} // Fin de public
```

### *V.2*/ Enumération :

1. **Définir un type Énumération**

```
public enum Saison {
        Printemps,
        Eté,
        Automne,
        Hiver
}
```

# Programmation Objets avec C# – TP1 –

### 2. Utiliser un type Énumération

```
// Propriété
public class Exemple {
    // 1. Variable attribut d'une classe
    private Saison s1;

    // 2. Variable paramètre d'une méthode
    public void Methode(Saison s) {
        // 3. Variable locale d'une méthode
        Saison s2;
        // 4. Afficher la valeur Printemps de la variable s2
        Console.Writeline(s2. Printemps) ;
        ……
    } // Fin de Méthode
}
```

## ***V.3*/ Structure :**

- Une structure est presque comme une classe. Elle permet de créer des objets, possèdent des variables ou propriétés, des méthodes.
- A la différence d'une classe qui est de type référence ne possédant pas la valeur de l'objet mais une référence vers cet objet, la structure quant à elle est un type valeur et contient donc la valeur de l'objet.

```
public struct Personne {
    public string Prenom { get; set; }
    public int Age { get; set; }

    /* Surcharge de la méthode ToString*/
    public override string ToString() {
        return Prenom + " a " + Age + " ans";
    }
}
```

```
Personne Benhamouda = new Personne() { Prenom = "Ibrahim", Age = 20 };
Console.WriteLine(Benhamouda.ToString());
```

# Programmation Objets avec C# – TP1 –

## *V.4*/ Switch :

```
// Switch
switch (paramètre) {
  case v1 actions1 ; break ;
  case v2 actions2 ; break ;
  ……
  default actions_sinon ; break ;
} // Fin de switch
```

- ***Paramètre*** peut être un entier, un caractère, une chaîne de caractères
- ***default*** peut être absent
- ***break*** fait sortir de la structure de cas.
- Chaque ***case*** doit se terminer par ***break***, ***goto*** ou return sinon erreur.

## *V.5*/ List :

- Une ***List***: Une collection qui permet d'y accéder avec un index. ***Array*** et ***ArrayList*** sont des listes.
- ***Count*** permet de connaître le nombre d'éléments dans la collection
- ***using System.Collections***; doit figurer dans les ***using***
- Utiliser ***Add*** pour ajouter un ***object*** à la fin de la List, ***ArrayList***
- Utiliser ***insert*** pour ajouter un ***object*** à une position dans la List, ***ArrayList***
- Utiliser ***remove*** pour supprimer un ***object*** de la List, ***ArrayList***
- Utiliser **[]** pour accéder aux éléments de la ***ArrayList***
- Utiliser ***Clear*** pour supprimer tous les éléments

**Exemple :**

```
// Déclarer une Liste de type Produit
List<Produit> produits;
// Initialiser une liste à Null
this.produits = new List<Produit>();

// Ajouter à la liste un élément
private Produit p=new Produit();
produits.Add(p);
```

```
// Supprimer de la liste un élément
produits.Remove(p);
// Compter le nombre d'élément de la
liste
produits.Count;
// Insérer à une position un élément
produits.Insert(2,p);
```