SEQUENCE 2 – ASSOCIATION

SEANCE 5 : COMPTE ET OPERATIONS

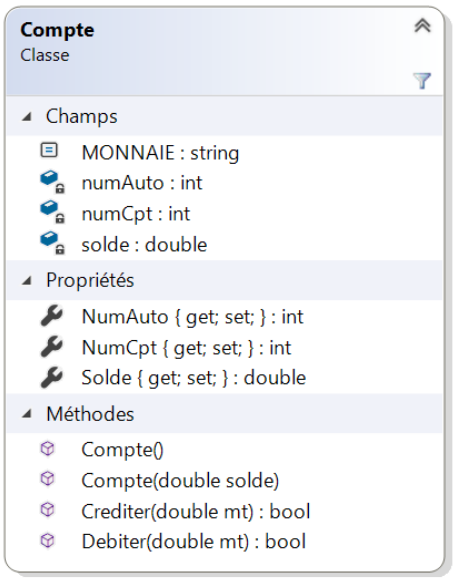
OBJECTIFS

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

CONSIGNES

1. Au sein de votre répertoire , créez un nouveau **projet** « **GestionCompte** » dans une **solution** « **GestionCompte** »

PARTIE 1 :



2. définissez la classe Compte avec :
- 2 champs privés : numCpt et solde.
- 1 champ statique privé : numAuto initialisé à 0
- 1 champ constant : MONNAIE initialisé à « € »

RAPPEL : un champ constant est avant tout statique.
Tous les champs statiques doivent être initialisés dès leur déclaration dans la zone de déclaration des champs alors que tous les champs d'instance (de l'objet) doivent être initialisés au sein du constructeur.

3. Encapsulez les 3 champs non constants. Observez la propriété de NumAuto : elle est elle aussi statique ! Faites un **accès privé sur le set de Solde, NumCpt et NumAuto** : ainsi elles seront consultables mais non modifiables en dehors de la classe.

4. Définissez deux constructeurs :
   - `public Compte(double solde)` : doit incrémenter NumAuto puis initialiser numCpt avec, et initialiser le solde à l'aide du montant passé en paramètre. Le solde ne peut pas être négatif à la création. Mais il peut l'être par la suite.
   - `public Compte()` : s'appuie sur le précèdent pour créer un compte avec un solde par défaut à 0.

5. Substituez (Redéfinissez) les méthodes habituelles :
   - ToString
   - Equals (et GetHashCode) et opérateurs: elle teste juste le numéro de compte.

6. Créez 2,3 comptes pour voir si la numérotation automatique fonctionne. Rem : pour afficher les caractères spéciaux : au début du main :
   Console.OutputEncoding = Encoding.UTF8; (avec using System.Text;)

7. Définissez les méthodes :
   - Crediter : elle ajoute le montant passé en paramètre au solde du compte **puis renvoie true. Attention : si le montant passé est négatif, elle envoie une ArgumentException**
   - Debiter : elle retire le montant passé en paramètre au solde du compte **puis renvoie true.Le solde peut être négatif : il n'y a pas de limite. Attention : si le montant passé est négatif, elle envoie une ArgumentException**

8. Reprenez le menu ci-dessous et complétez les case 1,2,3 :

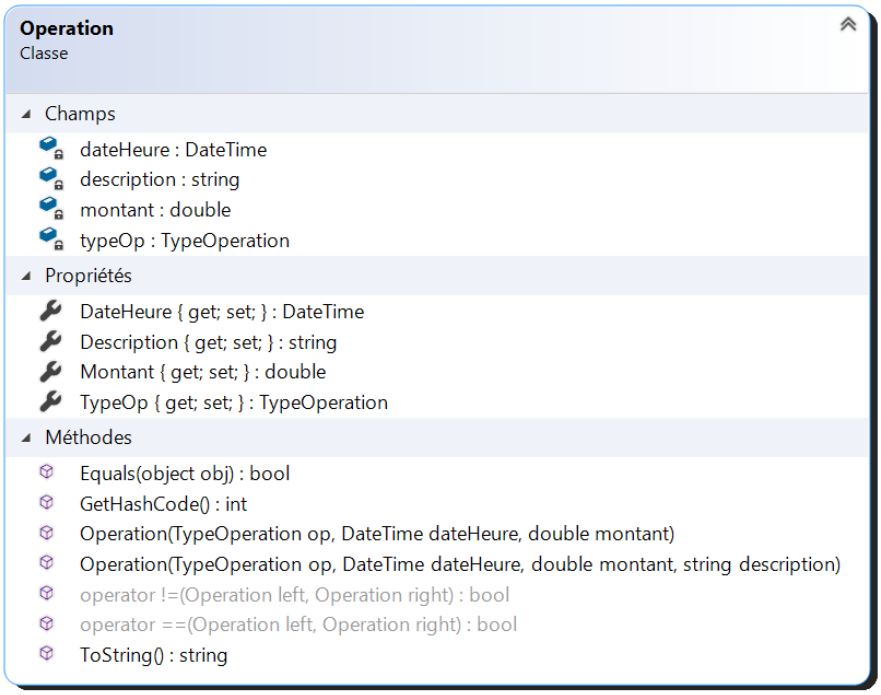
```
static void Main(string[] args)
{       Console.OutputEncoding = Encoding.UTF8;
        Compte c = new Compte();
        int choix ;
        do
        {       Console.WriteLine("0.Quitter");
                Console.WriteLine("1.Créditer");
                Console.WriteLine("2.Débiter");
                Console.WriteLine("3.Consulter le solde");
                choix = Program.SaisieInt(0, 3);
                switch (choix)
                {
                        case 0: Console.WriteLine("Au revoir."); break;
                        case 1:
                                {
                                Console.WriteLine("------------------------------");
                                Console.WriteLine("CREDITER :");
                                Console.WriteLine("------------------------------");
                                Console.WriteLine("Montant à crediter :");
                                double mt = Program.SaisieDoublePositif();
                                break;
                                }
                        case 2:
                                {
                                Console.WriteLine("------------------------------");
                                Console.WriteLine("DEBITER :");
                                Console.WriteLine("------------------------------");
                                Console.WriteLine("Montant à débiter :");
                                double mt = Program.SaisieDoublePositif();
                                break;
                                }
                        case 3:
                                {
                                Console.WriteLine("------------------------------");
                                Console.WriteLine("CONSULTER LE SOLDE :");
                                Console.WriteLine("------------------------------");
                                Console.WriteLine(c);
                                break;
                                }
                }
                Console.WriteLine("appuyez sur une touche...");
                Console.ReadLine();
                Console.Clear();
        } while (choix != 0);
   }

public static double SaisieDoublePositif()
{
        string saisie = Console.ReadLine();
        double nb;
        while (! ( double.TryParse(saisie, out nb) && nb >=0))
        {
```

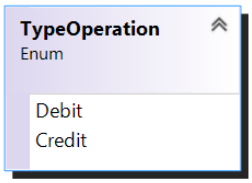
```
                Console.WriteLine("Erreur. Nb >= 0");
                saisie = Console.ReadLine();
        }
        return nb;
}
public static int SaisieInt(int min,int max)
{
        string saisie = Console.ReadLine();
        int nb;
        while (!  ( int.TryParse(saisie, out nb) && nb >= min && nb<= max ) )
        {
                Console.WriteLine($"Erreur. Nb >= {min} et nb <= {max}");
                saisie = Console.ReadLine();
        }
        return nb;
}
```

PARTIE 2 :

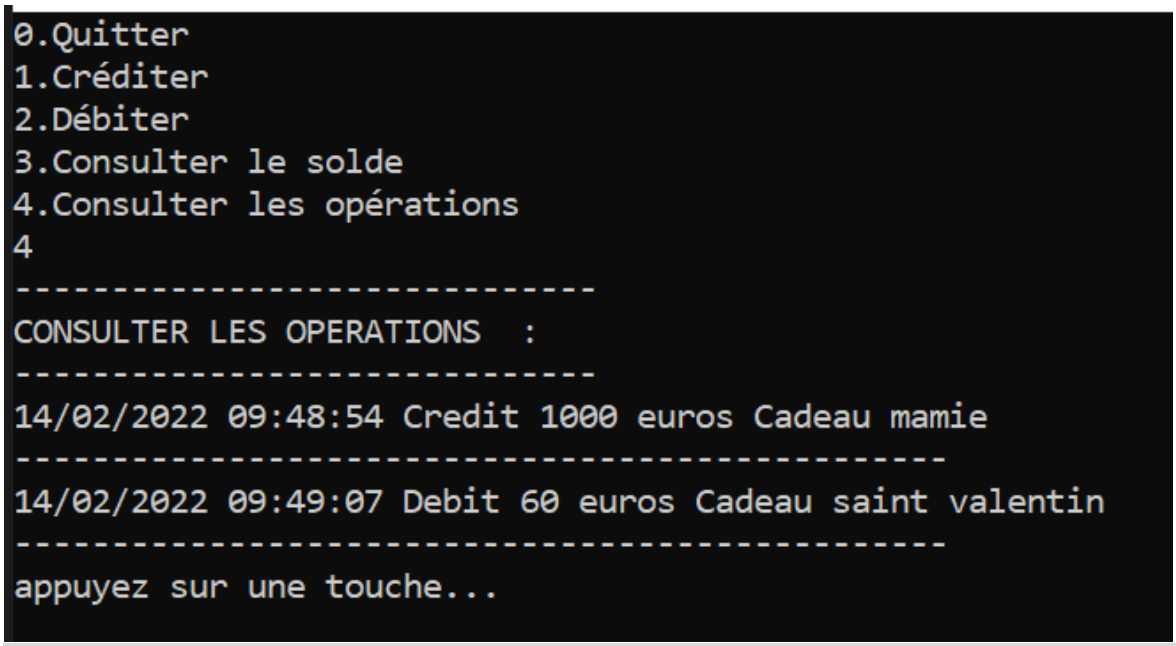
1. Ajoutez un nouveau projet « GestionCompteAvecOperations ». Faites glisser vos classes précédentes dans l'explorateur de solution (changez les namespaces)

2. On veut pouvoir ajouter une option « consulter la liste des opérations » : l'utilisateur doit pouvoir visualiser la liste des débits/crédits avec date et horaire, on pourra éventuellement avoir aussi une description (il faudra en faire la saisie). Pour cela, vous ajouterez la classe Operation et l'énumération TypeOperation





Vous ferez les modifications nécessaires dans la classe Compte afin de pouvoir stocker les opérations et rendrez l'option 4 du menu effective.



PARTIE POUR LES + RAPIDES

Vous sauvegarderez les opérations du compte dans un fichier . (json ou excel mieux encore !)