

## SEQUENCE 3 – HERITAGE

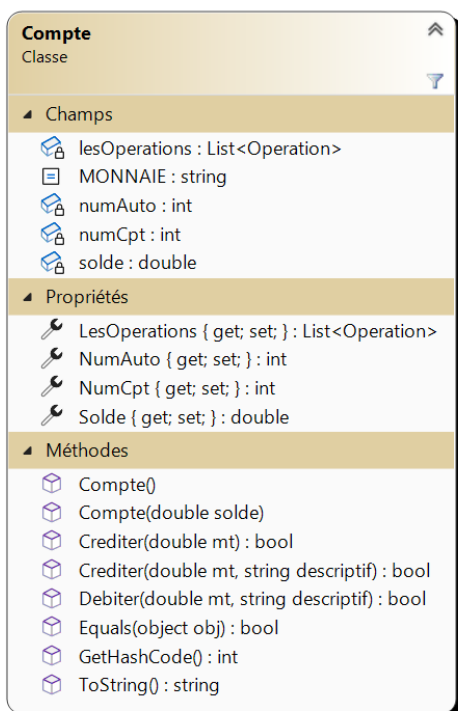
### SEANCES 1 ET 2 : COMPTES

#### OBJECTIFS

- Découvrir et manipuler la notion d'héritage
- Comprendre le polymorphisme et savoir le gérer

#### PARTIE 1 : UN COMPTE

1. Au sein de votre répertoire « R2\_01\_Approfondissement\_C# », créez un répertoire « **Sequence\_3\_Heritage** ». Créez un nouveau **projet** « **Partie1-UnCompte** » dans une **solution** « **GestionComptes** »

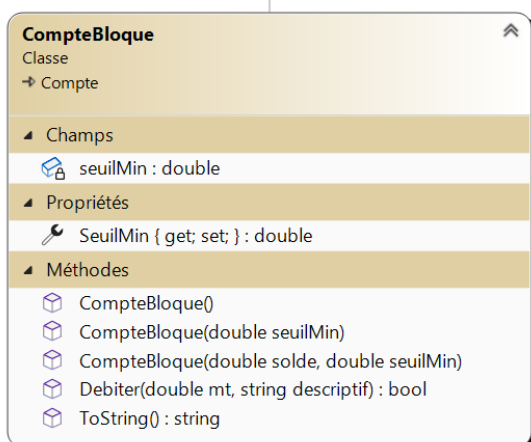


2. Reprenez vos classes de la séance précédente: Compte (et Operation si vous avez fini la gestion de la liste d'opérations au sein de la classe Compte) et Program pour les mettre dans votre nouveau projet. Changez les namespace.

3. Qualifiez les méthodes Crediter et Debiter de **virtual** pour permettre la substitution

4. Définissez la classe CompteBloque : elle **hérite de Compte**, elle a un champ supplémentaire : seuilMin. il représente le solde minimum autorisé, au-dessous duquel tout débit est interdit.

```
public class CompteBloque : Compte
{
    private double seuilMin;
```



5. Encapsulez seuilMin: il doit être  $\leq 0$  (à vérifier au sein de la propriété)

6. Définissez les 3 constructeurs : conseil : commencez toujours par celui qui a le plus de paramètres !

- **public CompteBloque**( double solde, double seuilMin)

**Rappel : Construire un CompteBloqué c'est avant tout construire un Compte à l'aide du mot clef **base**.**

```
public CompteBloque(double solde ,double seuilMin) : base(solde)
{
    this.SeuilMin = seuilMin;
}
```

- `public CompteBloque(double seuilMin) :` vous pouvez :
  - soit réutiliser le constructeur précédemment créé avec `this` :

```
public CompteBloque(double seuilMin) : this(0, seuilMin)
{ }
```

- soit réutiliser un constructeur de la classe `Compte` avec `base` :






```
public CompteBloque(double seuilMin) : base()
{ this.SeuilMin = seuilMin; }
```

Rem : si vous omettez de mettre : `base ( )`, c'est fait par défaut !

- `public CompteBloque( ) :` crée un compte bloqué par défaut à 0 avec un solde à 0 : à vous de jouer ...

7. Générez les substitutions de :

Choisir les membres à substituer

<input type="checkbox"/>	 Equals(object)
<input type="checkbox"/>	 GetHashCode()
<input checked="" type="checkbox"/>	 ToString()
<input type="checkbox"/>	 Crediter(double, string)
<input checked="" type="checkbox"/>	 Debiter(double, string)

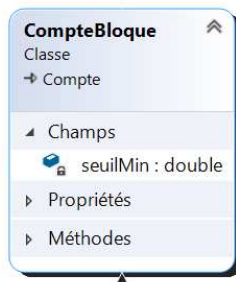
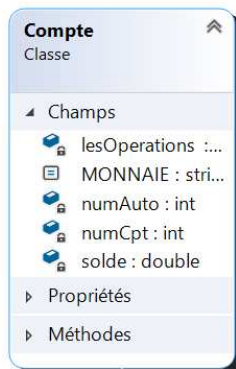
Rem : Les autres méthodes (`Equals`, `GetHashCode` et `Créditer` ) pourront être utilisées telles quelles sont définies dans la classe `Compte`, le fonctionnement est le même pour un compte bloqué.

8. **Substituez partiellement** la méthode habituelle `ToString`. Rappel : cela signifie qu'elle utilise tout de même ce que fait la classe mère : `base.ToString()` )

```
public override string ToString()
{
    return base.ToString() + "\nSeuil min : " + this.SeuilMin;
}
```

9. **Substituez partiellement** la méthode `Debite` : renvoie `true` et débite le solde du montant passé en paramètre si le débit est autorisé `false` sinon.

10. Dans la classe : Program. Remplacez votre objet Compte par un objet de classe CompteBloque. Modifiez l'appel à débiter pour prendre en compte que le retour puisse être false ! Puis testez avec des valeurs de votre choix à l'aide du menu, les différentes méthodes héritées : Crediter et celles substituées : Debiter, ToString.



11. Définissez la classe dérivée

CompteEpargne, elle hérite de CompteBloque, elle a deux champs privés supplémentaires :

a. **txEpargne** : compris entre 0 et 1 sinon ArgumentOutOfRangeException

b. **seuilMax** : représente le solde maximum à ne pas dépasser : seuil max > 0 sinon ArgumentException

12. Définissez les constructeurs en veillant toujours à ne pas faire de copier/coller. Attention : on ne doit pas pouvoir créer un CompteEpargne avec un solde supérieur au seuil Max. Un compte épargne n'autorise aucun découvert : le seuil minimum est initialisé à 0 par défaut.

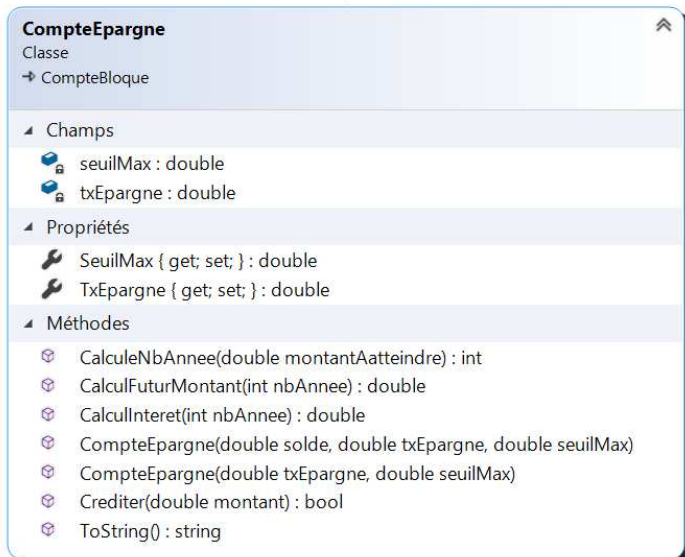
13. Substituez ToString et Crediter en veillant toujours à ne pas faire de copier/coller.

14. Définissez les méthodes suivantes :

- CalculFuturMontant (int nbAnnee) : double – retourne le total obtenu : gains + solde de départ en fonction du nombre d'année et du taux d'épargne.

$$V_f = V_i(1 + \rho)^a$$

Remarque : avec Vf pour la valeur finale, Vi pour valeur initiale, a pour nombre d'années et p pour taux d'intérêt . Le taux est en pourcentage, on écrit 0.02 pour 2%.



- CalculInteret(int nbAnnee) : double - retourne uniquement les gains obtenus.
- CalculNbAnnee(double montantAAtteindre) : int – retourne le nombre d'année à attendre pour atteindre le montant passé en paramètre.

$$a = \frac{\ln \frac{V_f}{V_i}}{\ln (1 + \rho)}$$

15. Dans la classe : Program. Remplacez votre objet CompteBloque par un objet de classe CompteE-pargne et appliquez-lui les différentes méthodes. Pour cela, améliorez le menu.
16. Améliorez votre classe : Program pour permettre à l'utilisateur de choisir entre la création d'un Compte, d'un Compte Epargne ou d'un Compte bloqué.

## PARTIE 2 : DES COMPTES

1. Faites un nouveau projet « Partie2-DesComptes ». Glissez vos classes ou ajoutez une dépendance vers le projet Partie1

2. Définissez la classe Client : Nom et liste de comptes.

- Recherche : vous pouvez utiliser la méthode Find de la classe List, cela évite de faire une boucle pour rechercher le compte dont le numéro est passé en paramètre
- Crediter / debiter : attention, il faut créditer et/ou débiter le bon compte et donc vérifier s'il existe avant.

3. Au sein de Program, instanciez un client. Ajoutez lui 3 comptes : un normal, un bloqué et un épargne avec des valeurs de votre choix. Faites un menu pour :

- voir la synthèse des comptes,
- créditer/ débiter un compte en particulier, vous utiliserez recherche avant de déclencher Crediter/Debiter
- ouvrir un nouveau compte.
- faire un virement d'un compte à un autre

**Client**  
Classe

- ▲ Champs
  - 🔒 nom : string
  - 🔒 sesComptes : List<Compte>
- ▲ Propriétés
  - 🔧 Nom { get; set; } : string
  - 🔧 SesComptes { get; set; } : List<Compte>
- ▲ Méthodes
  - 🔧 AjouteCompte(Compte c) : void
  - 🔧 Client(string nom)
  - 🔧 Client(string nom, List<Compte> sesComptes)
  - 🔧 Crediter(int numero, double montant) : bool
  - 🔧 Debiter(int numero, double montant) : bool
  - 🔧 Recherche(int numero) : Compte
  - 🔧 ToString() : string

## POUR LES + RAPIDES :

Ajoutez les options :

- voir les opérations d'un compte
- exporter les opérations d'un compte (un fichier excel (.csv ou mieux .xls package) )