

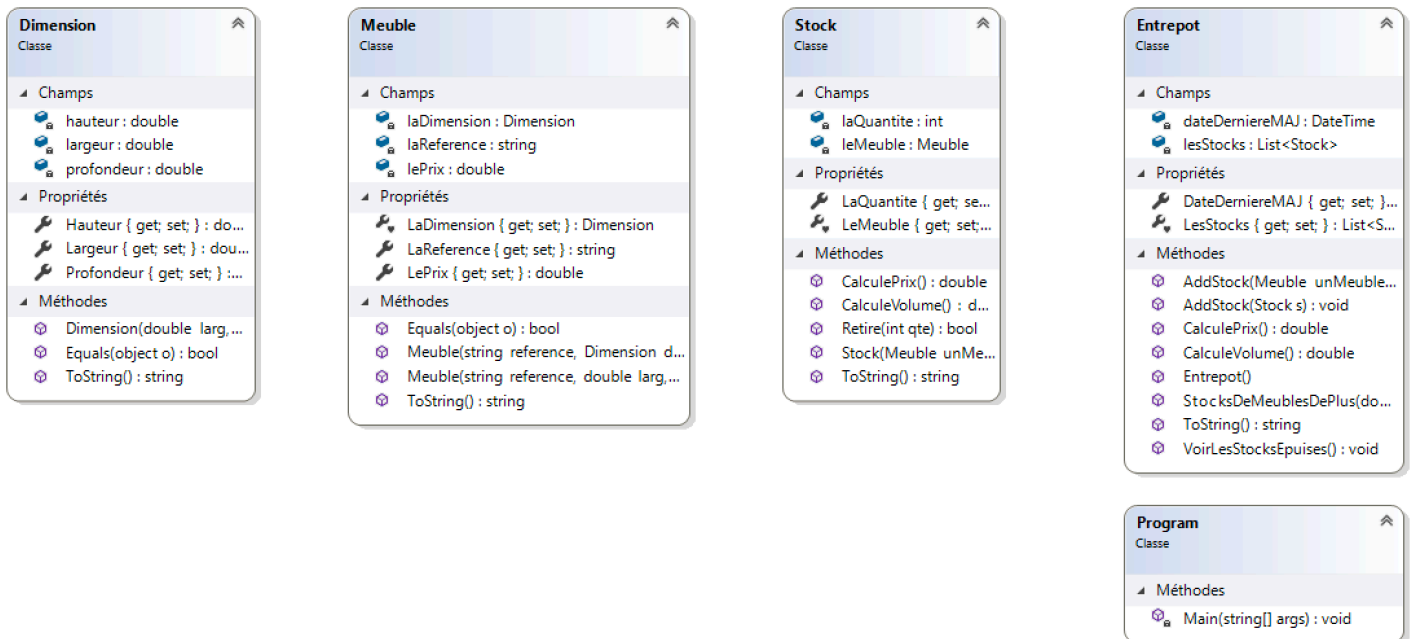
1 Rétro ingénierie

La rétro ingénierie consiste récupérer les fichiers source d'un programme et de générer le diagramme de classe équivalent. Tout modification du code source entraine alors la modification du diagramme de classe.

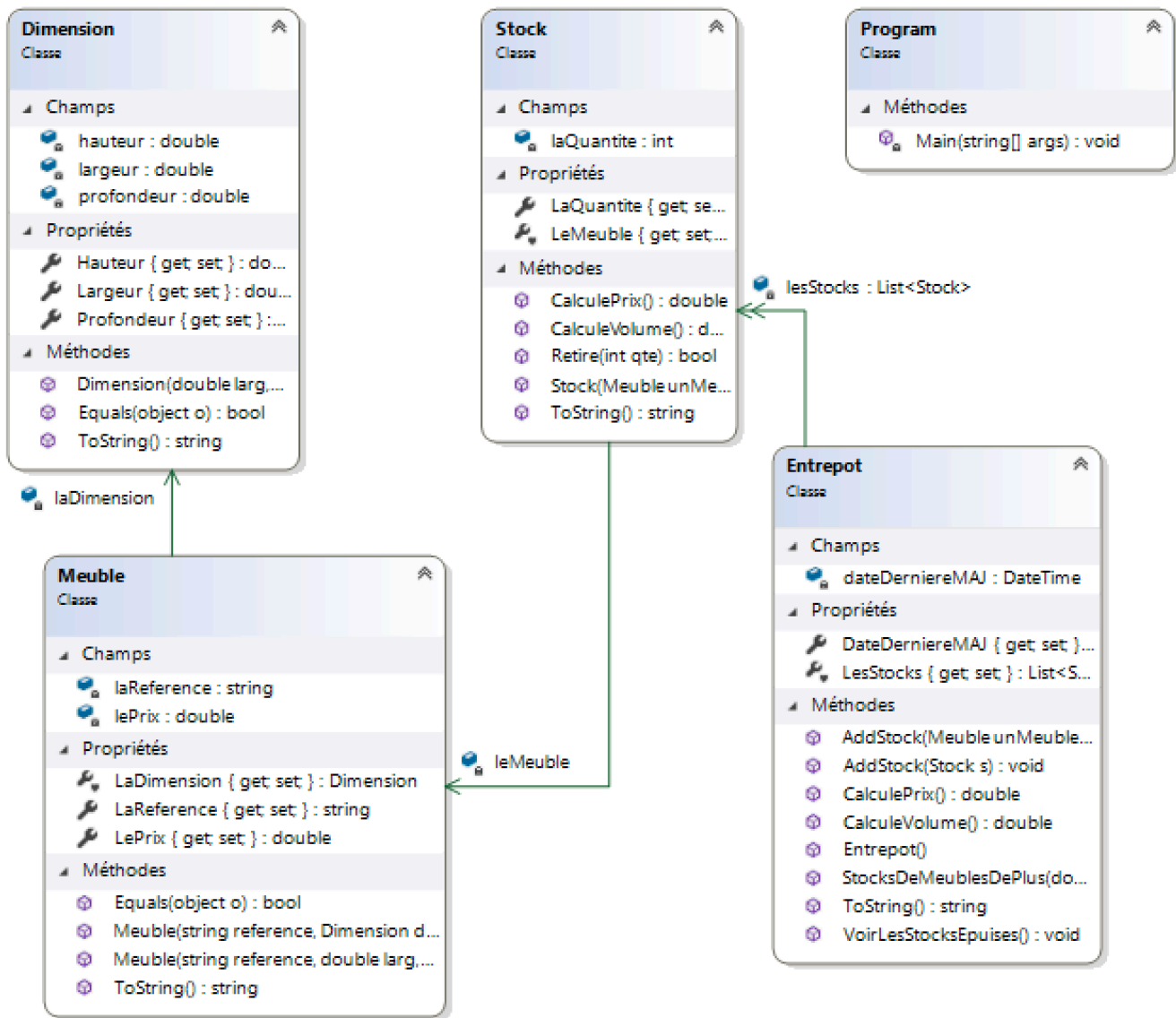
Récupérez la solution TP Retro sur le réseau. Il s'agit de la gestion d'un entrepôt de meubles.
Affichez le diagramme de classe (projet → Affichage diagramme de classe) :



Il est possible d'améliorer la lisibilité :
(Modifier format des membres)



Il est possible d'afficher les associations :
 (Cliquez sur une propriété associée : afficher en tant qu'association)



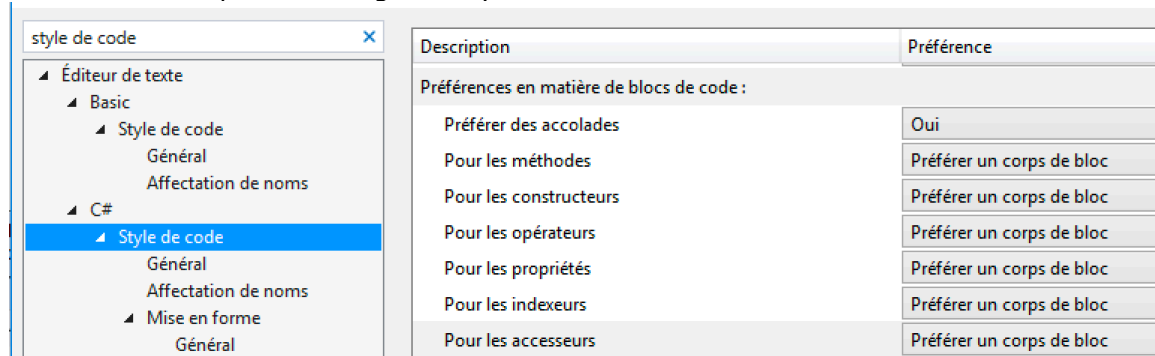
Une modification de code sera immédiatement répercutée sur le diagramme de classe

2 Génération de code

Il est possible de concevoir directement le squelette de son code à partir d'un diagramme de classe. Cela nécessite une petite préparation de Visual Studio.

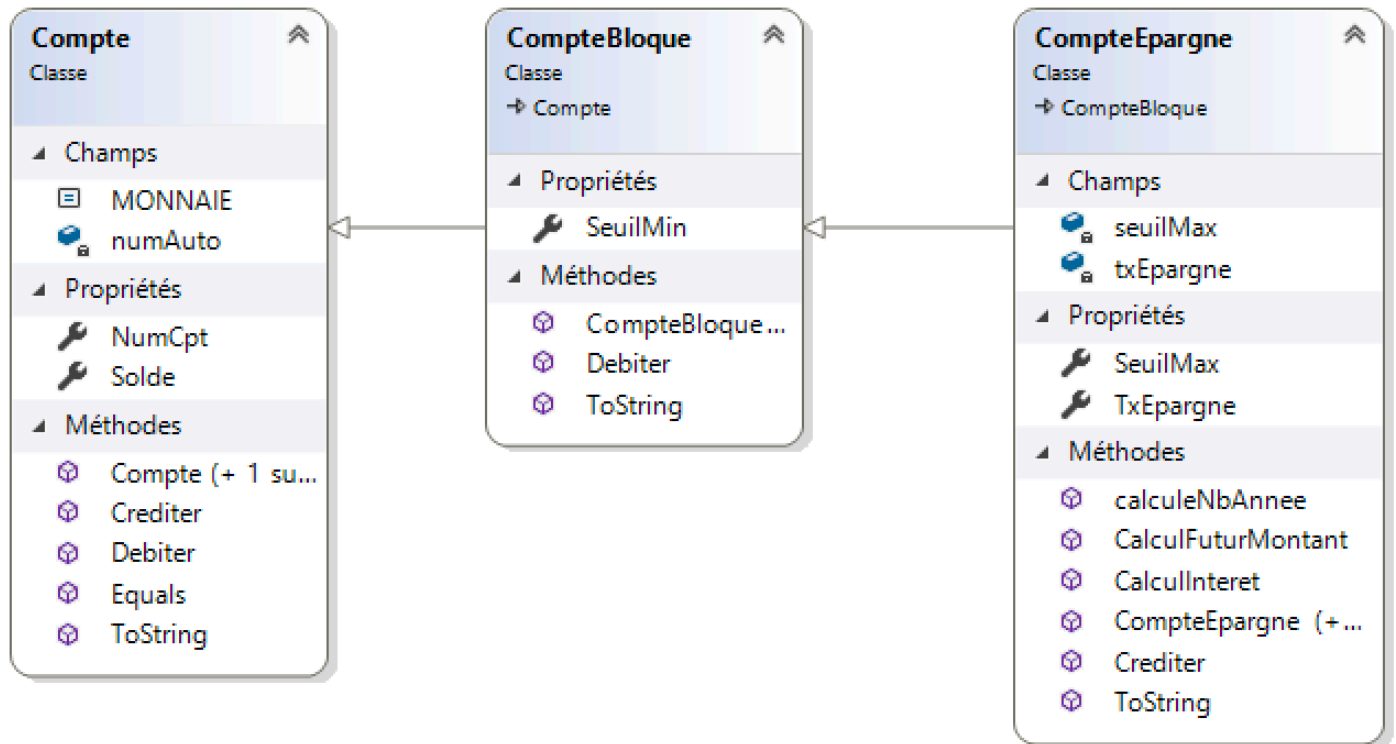
2.1 Préparation de Visual Studio

Dans outils → options configurer style de code C# :



Rq : vous pouvez aussi configurer l'utilisation du « this »

2.2 Génération de code – Exercice 1



Pour les besoins d'une application destinée au secteur bancaire, vous allez définir 3 classes : Compte, CompteBloque, CompteEpargne.

Créez un nouveau projet Console. Affichez le diagramme de classe. A l'aide de la boîte à outils créez les 3 classes en respectant le diagramme de classe ci-dessus.

Une fois l'architecture respectée complétez le code en respectant les consignes suivantes :

Dans la classe Compte codez :

- Un accès privé sur le set de NumCpt, car le NumCpt est initialisé (pour ne plus être modifié) à la construction de l'objet Compte à l'aide de la variable statique numAuto (initialisée à 0) qui sera préalablement auto-incrémentée.
- Un accès privé sur le set de Solde
- Les méthodes :
 - Crediter : elle crédite le solde du compte avec le montant passé en paramètre puis renvoie true
 - Debiter : elle débite le solde du compte avec le montant passé en paramètre puis renvoie true
 - ToString et Equals (elle teste juste le numéro de compte)

« Testez » les méthodes au sein de la classe Program en faisant un menu :

0. quitter
1. créditer
2. débiter
3. consulter le compte

Dans la classe CompteBloque le champ interne supplémentaire seuilMin représente le solde minimum autorisé, au-dessous duquel tout retrait est interdit.

- Définissez les 3 constructeurs : en 1er celui avec les 2 paramètres : il doit s'appuyer sur le constructeur de la classe mère. Faites appel au mot clef base au lieu de faire un copier/coller. Les 2 autres doivent s'appuyer sur ce 1er constructeur. Faites appel au mot clef this
- Substituez la méthode Debite : renvoie true et débite le solde du montant passé en paramètre si le débit est autorisé false sinon.

- Substituez la méthode ToString() : String afin de prendre en compte le nouvel attribut et faites appel au mot clef base ...

4 Dans la classe Program. Remplacez votre objet Compte par un objet de classe CompteBloque avec des valeurs de votre choix puis testez à l'aide du menu, les différentes méthodes héritées : Crediter et celles substituées : Debiter, ToString.

La classe dérivée CompteEpargne possède deux champs privés supplémentaires : txEpargne (compris entre 0 et 1), seuilMax qui représente le solde maximum à ne pas dépasser. Le seuil minimum est initialisé à 0 par défaut.

- Définissez les constructeurs en veillant toujours à ne pas faire de copier/coller.
- Substituez ToString et Crediter en veillant toujours à ne pas faire de copier/coller.
- Définissez les méthodes suivantes :
 - CalculInteret(int nbAnnee) : double - retourne uniquement les gains obtenus.
 - CalculFuturMontant (int nbAnnee) : double – retourne le total obtenu : gains + solde de départ en fonction du nombre d'année et du taux d'épargne.
 - CalculNbAnnee(double montantAAteindre) : int – retourne le nombre d'année à attendre pour atteindre le montant passé en paramètre.

Dans la classe Program remplacez votre objet CompteBloque par un objet de classe CompteEpargne et appliquez-lui les différentes méthodes. Pour cela, améliorez le menu.

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.

Définissez la classe Client : Nom et liste de comptes.

Au sein de Program, instanciez un client. Ajoutez lui 3 comptes avec des valeurs de votre choix. Faites un menu pour voir la synthèse des comptes, créditer/ débiter un compte en particulier, ouvrir un nouveau compte.

Définissez la classe Opération : type (+ pour crédit ou – pour retrait), montant, date. Puis modifiez le code de la classe Compte pour stocker et afficher (dans la méthode ToString) chaque opération. Ajoutez 3 champs privés totalCredits, totalDebits, debitMax avec les propriétés associées pour une lecture seule. Améliorez votre programme pour avoir un menu pour accéder à la liste des opérations sur un compte.

Bonus : Créez les tables utiles pour réaliser la persistance des données.

2.3 Génération de code – Exercice 2 pour les plus rapides

2.3.1 Classe Animal

Définissez la classe abstraite Animal :

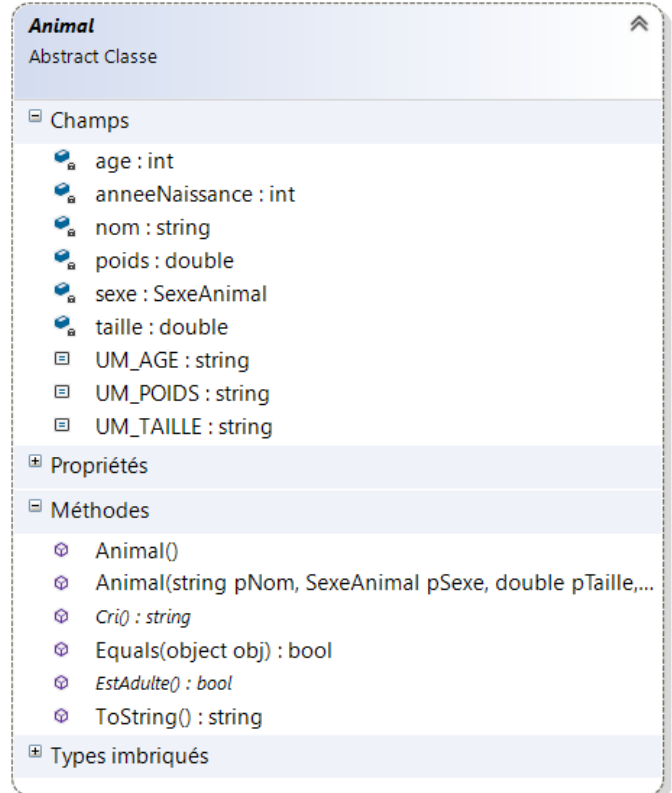
- Elle a 1 enum :

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

- Elle a 3 constantes :

```
public const String UM_TAILLE = "m";  
public const String UM_POIDS = "kg";  
public const String UM_AGE = "an(s)";
```

- L'âge n'est pas un champ privé comme les autres : il est en accès lecture uniquement (private set) : il ne doit pas être initialisé par un paramètre. Son initialisation est liée à la mise à jour du champ privé anneeNaissance (instructions à mettre dans le set de AnneeNaissance)
- L'année de naissance ne doit pas être supérieure à l'année actuelle
- Le nom ne doit pas être nul, ne doit pas être vide et doit être mis en majuscule
- Le poids et la taille ne doivent pas être négatifs.
- La méthode Equals : elle ne compare que le nom, car le nom identifie de manière unique l'animal.
- Les méthodes Cri() et EstAdulte() sont abstraites.
- La méthode ToString indique, en plus des valeurs propres à l'animal, s'il s'agit d'un adulte pour cela, elle s'appuie sur la méthode EstAdulte, même si elle est abstraite.



```
Nom : DJUMBO  
Sexe : M  
Annee de naissance : 1988  
Age : 29 an(s)  
Adulte : True  
Poids : 200 kg  
Taille : 2,54 m
```

2.3.2 Classe Elephant

1. Définissez la classe Elephant. Elle est fille de la classe Animal. Elle n'est pas abstraite.

Ajoutez l'énum suivante :

```
public enum Continent { Asie , Afrique };
```

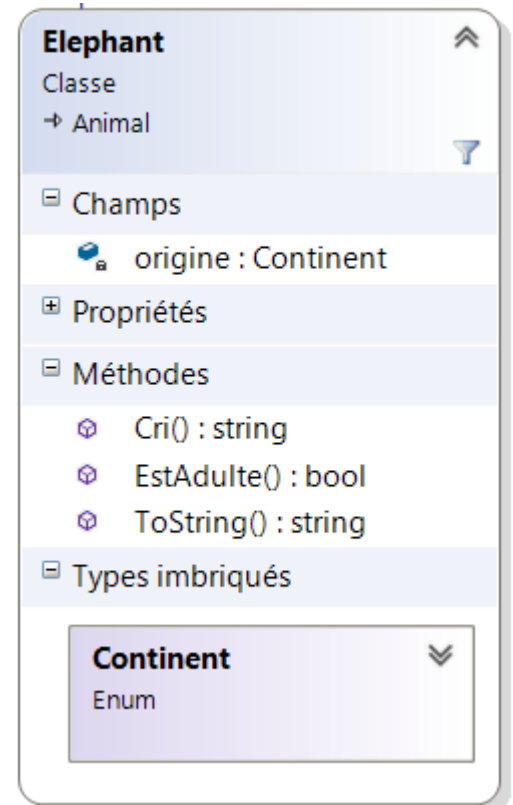
- ToString : elle doit faire appel au ToString hérité puis en plus indiquer :
 - o qu'il s'agit d'un éléphant
 - o son origine
 - o la taille des oreilles de l'éléphant et la présence ou non de défenses, sachant qu'un éléphant d'Afrique a de très grandes oreilles alors qu'un éléphant d'Asie a de petites oreilles. Les éléphantesses (sexe féminin) d'Asie n'ont pas de défense.
- EstAdulte : retourne vrai s'il a plus de 13 ans
- Cri : retourne une chaîne correspondant au barrissement de l'éléphant, celle-ci doit donner la sensation d'un barrissement plus fort s'il s'agit d'un éléphant adulte.

2. Au sein de la classe program, instanciez 5 éléphants, ajoutez les à une liste : List <Animal> :

- 1 adulte mâle et 1 adulte femelle d'Afrique,
- 1 adulte mâle et 1 adulte femelle d'Asie
- 1 non adulte mâle d'Afrique

Affichez-les et affichez leurs cris à l'aide d'une boucle.

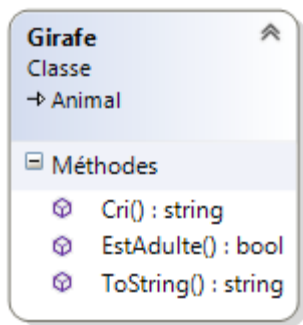
Assurez-vous d'avoir un affichage similaire à ceci :



```
Nom : DJUMBO
Sexe : M
Annee de naissance : 1988
Age : 29 an(s)
Adulte : True
Poids : 200 kg
Taille : 2,54 m
Eléphant d'Afrique avec de grandes oreilles et défenses
Son cri est : HUUUUMMMMMM

Nom : ASIII
Sexe : F
Annee de naissance : 2010
Age : 7 an(s)
Adulte : False
Poids : 150 kg
Taille : 1,7 m
Eléphant d'Asie avec de petites oreilles sans défense
Son cri est : huummmm
```

2.3.3 Classe Girafe



1. Définissez la classe Girafe :

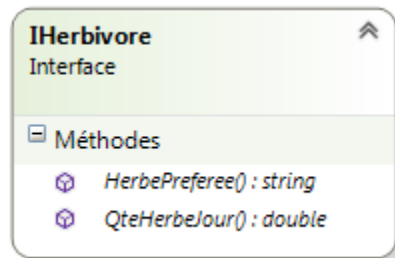
- ToString : elle doit en plus indiquer qu'il s'agit d'une girafe
- EstAdulte : retourne vrai si c'est un mâle de plus de 4 ans ou si c'est une femelle de plus de 5 ans.
- Crie : retourne une chaîne correspondant au mugissement de la girafe, celle-ci doit donner la sensation d'un mugissement plus fort s'il s'agit d'une girafe non adulte. En effet, les adultes font moins de bruit.

2. Au sein de la classe program, instanciez deux girafes : Une femelle adulte et un mâle non adulte.

Ajoutez-les à la liste puis affichez-les et affichez leurs cris.

2.3.4 Interface IHerbivore

1. Définissez l'interface : IHerbivore . Elle contient uniquement la signature des méthodes.

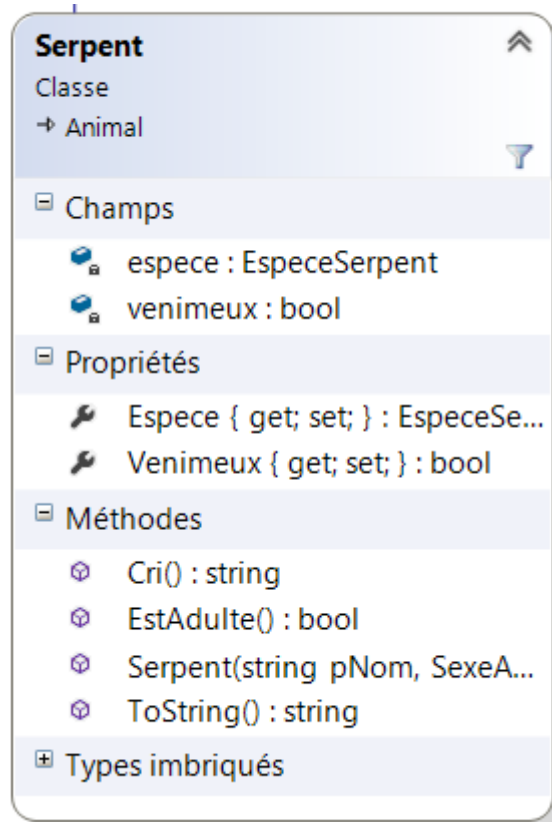


2. Implémentez l'interface : Indiquez que les classes Elephant et Girafe sont herbivores. Puis codez les méthodes :
 - Un éléphant mange 25 % de son poids (un éléphant d'une tonne mange à peu près 250 kg d'herbes par jour). Les éléphants africains se nourrissent de feuillages et d'arbustes alors que les éléphants asiatiques préfèrent les herbes. Jusqu'à 5 ans, il est alimenté par le lait maternel.
 - La girafe se nourrit essentiellement d'acacias, elle mange 5% de son poids. Jusqu'à 2 ans, elle est alimentée par le lait maternel.
3. Transformez votre liste en liste d'Iherbivore, de manière à afficher le repas journalier de chaque animal contenu dans la liste, pensez à indiquer son nom en face du repas.

2.3.5 Classe Serpent

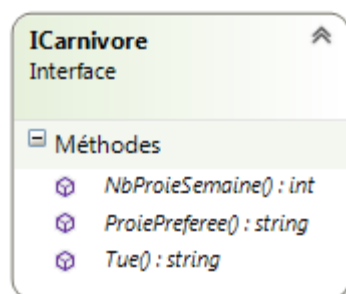
1. Définissez la classe Serpent :

- Il a pour enum :
`public enum EspeceSerpent { Crotale, A_sonette, Cornu, Echidé, Autre }`
- ToString doit en plus indiquer qu'il s'agit d'un serpent et s'il est venimeux.
- EstAdulte : retourne vrai s'il a plus de 2 ans
- Cri : retourne une chaine correspondant au bruit émis par le serpent. De nombreux serpents sifflent, mais certains produisent des sons plus insolites : comme le bruit de grelot que font les crotales ou serpents à sonnette. Le céraste cornu et les échidés ont sur les flancs des écailles denticulées, comme des morceaux de papier de verre, qu'ils frottent les uns contre les autres pour produire un crissement d'avertissement



2.3.6 interface ICARNIVORE

1. Définissez l'interface : ICarnivore . Elle contient uniquement la signature des méthodes.



2. Implémentez l'interface : Indiquez que la classe serpent est Carnivore. Puis codez les méthodes :
 - La nourriture des serpents dépend de leur âge :
 - À moins d'un an : 2 souriceaux ou équivalents par semaine
 - À partir d'un an : 2 souris ou équivalents par semaine
 - A partir de 2 ans : 1 cochon d'inde ou équivalent par semaine
 - Tue renvoie une chaîne décrivant le procédé employé par le carnivore pour tuer ses proies

2.3.7 Classe Ours : iherbivore et icarnivore

Un ours appartient à une espèce : Polaire, Brun, Noir. Vous définirez une énum.

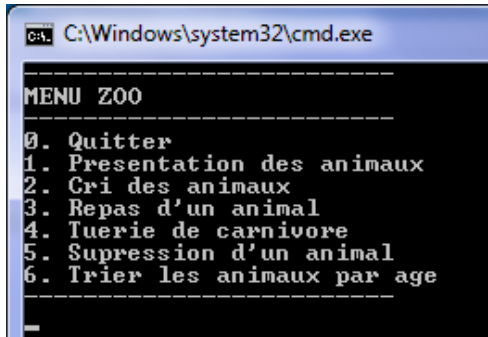
Un ours est omnivore, il est à la fois carnivore et herbivore !

- ToString pour indiquer qu'il s'agit d'un ours et pour avoir l'espèce
- EstAdulte : retourne vrai s'il a plus de 4 ans pour l'ours brun, sinon 3 ans pour les autres espèces.
- Crie : retourne une chaîne correspondant au grognement d'un ours

2.3.8 Classe ZOO

Codez de la classe Zoo : c'est une liste d'animaux.

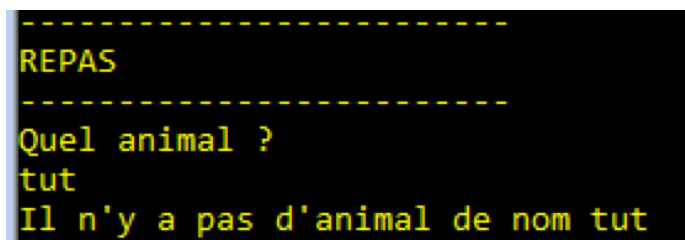
1. Définissez la méthode : AddAnimal. Attention, avant d'ajouter, elle doit vérifier que l'animal passé n'est pas nul et qu'il n'y a pas déjà un animal au sein de la liste qui porte le même nom. Elle doit lancer une exception en cas de nullité et renvoie false en cas d'animal déjà existant. Le programme (classe Program) doit alors afficher un message : soit « l'animal a bien été ajouté », soit « l'animaln'a pas pu être ajouté car il existe déjà un animal avec ce nom ». Modifiez votre classe Program en conséquence.
2. Créez le menu suivant :



```

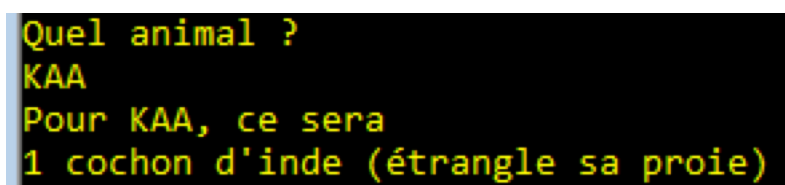
C:\Windows\system32\cmd.exe
-----
MENU ZOO
-----
0. Quitter
1. Presentation des animaux
2. Cri des animaux
3. Repas d'un animal
4. Tuerie de carnivore
5. Suppression d'un animal
6. Trier les animaux par age
-----
  
```

3. Définissez la méthode : PresenteSesAnimaux. Remarque : vous utiliserez la boucle
4. Définissez la méthode : FaitCrierSesAnimaux.
5. Définissez la méthode : NourritUnAnimal.



```

-----
REPAS
-----
Quel animal ?
tut
Il n'y a pas d'animal de nom tut
  
```



```

Quel animal ?
KAA
Pour KAA, ce sera
1 cochon d'inde (étrangle sa proie)
  
```

6. Définissez la méthode : RemoveAnimal. (voir la doc concernant Remove et RemoveAt de la classe List sur MSDN pour comprendre comment elles fonctionnent) Elle renvoie false si la suppression n'a pas eu lieu, si le nom donné ne correspond à aucun animal. Le programme principal doit afficher un message : soit « l'animal ... a bien été supprimé », soit « l'animaln'a pas pu être supprimé car il n'existe pas »
7. Définissez public List<ICarnivore> ExtraitCarnivore() - Elle retourne une liste d'animaux du Zoo exclusivement carnivores. Ainsi on pourra déclencher la méthode tue() sur chaque animal qui constitue cette liste de manière à déclencher une tuerie.

2.3.9 Tri par âge :

On désire trier la liste d'animaux par âge (du plus vieux au plus jeune). Comment faire ?

2.3.10 Ajout :

Codez l'ajout d'animaux dans le zoo :

```

-----
MENU ZOO
-----
0. Quitter
1. Presentation des animaux
2. Cri des animaux
3. Repas d'un animal
4. Supression d'un animal
5. Trier les animaux par age
6. Ajouter un elephant
7. Ajouter un serpent
8. Ajouter un ours
-----

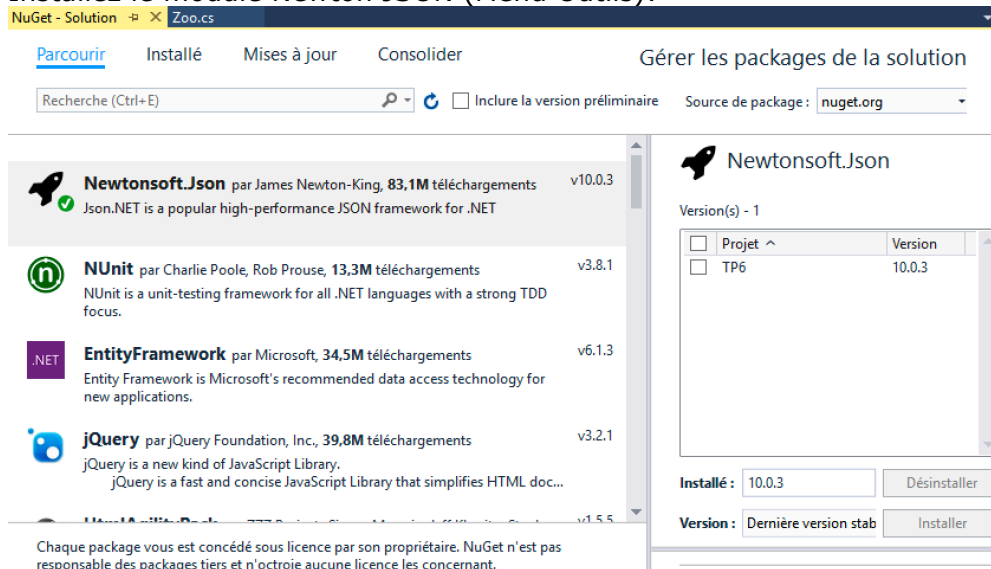
```

2.3.11 Persistence :

JSON :

Mettre en place la persistance des données du module JSON.

Installez le module Newton JSON (Menu Outils):



Créez un fichier JSON par animal (Elephant, Serpent, Ours)
Utilisez la classe File (ReadAllText et WriteAllText) et JSonConvert (sérialisation et désérialisation) pour la lecture/écriture de fichier
Attention à bien gérer les exceptions d'accès fichier.

Table Animaux :

Mettez en place la persistance à l'aide d'une table (local ou sur srv-jupiter) Animaux (contenant le type pour différencier l'animal)

Rq : on utilisera cette solution pour simplifier la modélisation car l'héritage n'est pas simple à mettre en œuvre sous SQL.

3 Utilisation de GitHub

3.1 Ouverture d'un compte GitHub

Ouvrir un compte GitHub.com

3.2 Utilisation de GitHub avec Visual Studio

Lire la documentation de Microsoft : [https://msdn.microsoft.com/fr-fr/library/hh850437\(v=vs.120\).aspx](https://msdn.microsoft.com/fr-fr/library/hh850437(v=vs.120).aspx)

Tester à l'aide d'un projet que vous avez déjà réalisé.