POO Suite

I. Les classes :

1) Définition :

Soit un objet du monde réel (ou de notre domaine de travail).

Décrire le comportement d'un objet, c'est créer une **classe.**

Un **objet** s'appelle l'instance d'une **Classe.**

Une **classe** possède 3 sortes de membres (!) :

a) **Méthode** ou opération : c'est une fonction décrivant un comportement de l'objet.
La liste des instructions d'une méthode s'appelle le **corps** de la méthode.

b) **Propriété ou Champs** ou attribut : représente l'état de l'objet.

Suivant sa visibilité (ou portée), champs et méthode peut être :

- privé : accessible seulement depuis la classe où il est défini.
- Protégé : accessible aussi par toutes les classes dérivées.
- Public : accessible par toutes les classes.

c) Propriété par accesseurs : semblable a un champ public mais accessible via deux méthodes appelés accesseurs (affecter et récupéré).

2) Héritage :

quand on veut concevoir une classe A possédant le même comportement qu'une autre classe B avec en plus un nouveau comportement, on utilise le mécanisme d'héritage.

Une classe **est dérivée** ou **hérite** d'une classe de **base** si elle a :
• au moins le comportement de cette classe de **base** • mais en plus un comportement ajouté.

On dit que la classe dérivée étend le comportement de la classe de base. (: en C#)

En C#, les classes « sealed »(scellée) ne peuvent être dérivée.

NB : comme en Algèbre où la factorisation est le processus inverse du développement, la **généralisation** est le processus inverse de l'**héritage**. Quand deux

classes possèdent des comportements communs, on peut vouloir créer une classe de base rassemblant ces comportements communs.

3) Héritage multiple : le fait pour une classe d'hériter de plusieurs classes de base.
   En C# : pas d'héritage multiple pour une classe mais possibilité d'hériter d'une classe et de plusieurs interfaces.

4) **Agrégation** : quand les champs d'une classe A sont des objets d'une classe B, on dit qu'il y a agrégation.

   On ne peut agréger que des classes qui sont visibles. (public ou définis dans la portée).

NB : il s'agit d'une relation entre classe et comme avec les bases de données relationnelles, il y a une multiplicité.

5) Conversion de type (cast) : lorsqu'elle est possible permet de transformer un objet de type TA en un objet de type TB.
   (placer le 5) après le 7))

```
TA objet1;
TB objet2 = (TB) objet1;
```

6) **Surcharge** de méthode : action qui consiste à redéfinir une méthode en changeant les paramètres et le corps.

7) Polymorphisme : quand une classe n'a pas le même comportement que sa classe de base.

   En effet, on peut changer le comportement d'une classe de base :

   - Soit en ajoutant des méthodes.
   - Soit en changeant le corps des méthodes de la classe de base.

Ceci n'est possible en C# qu'avec les méthodes virtuelles.
Quand on veut surcharger une méthode « virtual »,

- si on utilise le mot clef « override », la méthode originale est toujours accessible par le mot clef « base ».
- si on utilise le mot clef « new », c'est que l'on veut masquer la méthode de la classe de base. La méthode originale n'est plus accessible.

Si on veut qu'une méthode ne puisse pas être modifiée par une classe dérivée, on utilise «sealed» (scellée).
//15/01/10

(cf ExempleClasse)

8) Classe abstraite : c'est une classe qui contient au moins une méthode sans corps (pas de code).
   Les méthodes sans corps s'appellent des méthodes abstraites.
   On ne peut instancier une classe abstraite.
   Une méthode abstraite se comporte comme une méthode virtuelle.
   (mettre un exemple du framework)

9) Constructeur, destructeur :
   a) Création d'un objet càd de l'instance d'une classe :
      initialisation des champs de la classe de façon à placer l'objet dans un état bien défini. La méthode appelée à la création de l'objet s'appelle le **constructeur**.
   b) La vie d'un objet :

	OS	Avec C#
Naissance	Réserver l'espace mémoire Mettre l'objet dans l'état initial.	new Constructeur TA()
Vie	Appel des méthodes, opération sur les champs publics, affectation, Test d'égalité,	
Mort	Libérer les ressource Détruire les champs utilisés libérer l'espace mémoire	Finalize qui appelle Dispose puis le **destructeur** ~TA()

10) Interface : c'est une sorte de patron (au sens de calque) décrivant un comportement mais sans décrire les champs de la classe.

L'**Encapsulation** consiste à séparer les services (méthodes) offerts par l'objet du corps des méthodes càd du code.
Une classe A **implante** une interface I si toutes les opérations décrite dans l'interface I sont prises en charge.

Une classe peut implanter plusieurs interfaces.
// 22/01/10

II. En surface ou en profondeur :

En C#, l'accès aux objets se fait par référence :
soit la classe TA, la variable TA a; est une référence sur un objet de classe TA.
Soit une variable de même classe TA b; est une référence sur un objet de classe TA.

1) Affectation :
   a) En surface : b = a; ne recopie que la référence, il ne crée pas un nouvel objet.
      Après exécution, b et a sont deux références désignant le même objet.

   b) En profondeur : il faut créer une méthode qui clone l'objet dans l'exemple b = a.Clone() crée un nouvel objet qui est dans le même état que l'objet b.

2) Egalité :
   a) En surface : a==b est vrai si les références sont égales (donc les valeurs).
   b) En profondeur : il faut définir une méthode qui renvoie vrai si les valeurs des champs sont égales et faux sinon.
      En C#, nous avons la possibilité de remplacer l'opérateur ==.
      La méthode Equals de « objet » (base de toutes les classes) permet tout de même de comparer les références. Elle nous indique si les 2 objets ont les même propriétés.

III. Les packages ou unités de compilation ou espace de noms:

1) Définition :

En C#, les unités de compilation sont appelés « namespace ».

2) Importation ou utilisation :

Avec C#, soit l'espace de nom est définis dans le projet, soit il est importé par le biais d'une librairie (voir ajouter une référence).

L'instruction « using » permet d'utiliser les classes d'un « namespace » sans avoir à préciser son nom.

Exemple : projet ExempleNamespace

3) Visibilité des classes d'un package : seule les classes publiques sont utilisables.

IV. Les Patrons :

Par exemple Liste<T> est un patron, template ou pattern :
à la place de T on peut placer la classe d'objet de son choix.

On crée donc une classe de liste universel.