# TP3 Dot Net

## Exercice 1 :

Créez une classe « Person », une classe « Student » et une autre classe « Teacher », les deux dernières héritent de la classe « Person ».

La classe « Student » aura une méthode publique « GoToClasses », qui affichera à l'écran «I'm going to class. ».

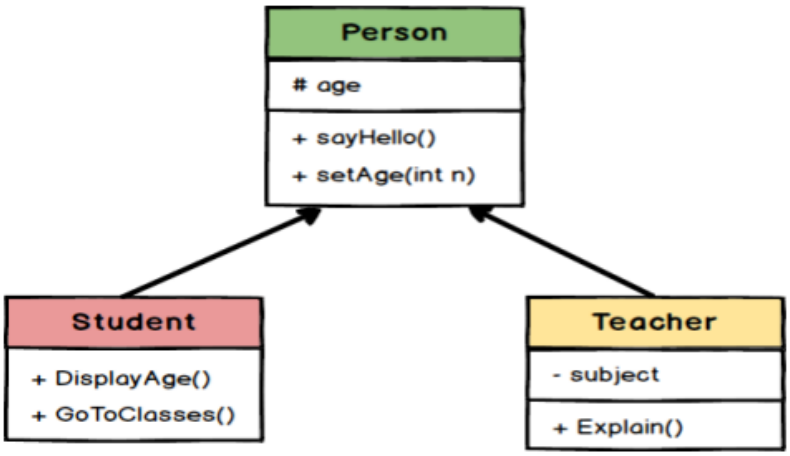
La classe « Teacher » aura une méthode publique « Explain », qui affichera à l'écran «Explanation begins ». En plus, il aura un attribut privé « subject » de type string.

La classe « Person » doit avoir une méthode « SetAge(int n) » qui indiquera la valeur de leur âge (par exemple, 15 years old).

La classe « Student » aura une méthode publique « DisplayAge » qui écrira sur l'écran « My age is: XX years old ».

Tester ces classes en :

- Créant un objet Person et faites-lui dire « Hello ».
- Créant un objet Student, définir son âge à 15 ans, faites-lui dire « Hello », « I'm going to class. » et afficher son âge.
- Créant un objet Teacher, 40 ans, demandez-lui de dire « Hello » puis commence l'explication.




## Exercice 2 :

Soit les classes suivantes :

- Une classe **Personne** qui comporte trois champs privés, nom, prénom et date de naissance. Cette classe comporte un constructeur pour permettre d'initialiser les données. Elle comporte également une méthode polymorphe Afficher pour afficher les données de chaque personne.

- Une classe **Employé** qui dérive de la classe Personne, avec en plus un champ *Salaire* accompagné de sa propriété, un constructeur et la redéfinition de la méthode Afficher.
- Une classe **Chef** qui dérive de la classe Employé, avec en plus un champ *Service* accompagné de sa propriété, un constructeur et la redéfinition de la méthode Afficher.
- Une classe **Directeur** qui dérive de la classe Chef, avec en plus un champ *Société* accompagné de sa propriété, un constructeur et la redéfinition de la méthode Afficher.

**Travail à faire :**

1. Ecrire les classe Personne, Employé, Chef et Directeur.
2. Créez un programme de test qui comporte tableau de huit personnes avec cinq employés, deux chefs et un directeur (8 références de la classe Personne dans lesquelles ranger 5 instances de la classe Employé, 2 de la classe Chef et 1 de la classe Directeur).
   Affichez l'ensemble des éléments du tableau à l'aide de for.
   Affichez l'ensemble des éléments du tableau à l'aide de foreach.

**Exercice 3 :**

Soit la classe abstraite Employé caractérisée par attributs suivants :

- Matricule
- Nom
- Prénom
- Date de naissance

La classe Employé doit disposer des méthodes suivantes :

- un constructeur d'initialisation
- des propriétés pour les différents attributs
- la méthode Tostring
- une méthode abstraite GetSalaire.

Un ouvrier est un employé qui se caractérise par sa date d'entrée à la société.

Tous les ouvriers ont une valeur commune appelée SMIG=2500 DH. L'ouvrier a un salaire mensuel qui est : Salaire = SMIG + (Ancienneté en année)*100.

De plus, le salaire ne doit pas dépasser SMIG*2.

Un cadre est un employé qui se caractérise par un indice.

Le cadre a un salaire qui dépend de son indice :

1 : salaire mensuel 13000 DH

2 : salaire mensuel 15000 DH

3 : salaire mensuel 17000 DH

4 : salaire mensuel 20000 DH

Un patron est un employé qui se caractérise par un chiffre d'affaires et un pourcentage.

Le chiffre d'affaires est commun entre les patrons.

Le patron a un salaire annuel qui est égal à x% du chiffre d'affaires : Salaire = CA*pourcentage/100

Travail à faire :

- Créer la classe abstraite Employé.
- Créer la classe Ouvrier, la classe Cadre et la classe Patron qui héritent de la classe Employé, et prévoir les Constructeurs et la méthode ToString de chacune des 3 classes.
- Implémenter la méthode GetSalaire() qui permet de calculer le salaire pour chacune des classes.

**Exercice 4 :**

Définir une classe Vecteurs2D caractérisée par l'abscisse X et l'ordonné Y, ainsi qu'un attribut qui renseigne sur le nombre de vecteurs créés lors de l'exécution du programme.

- Définir les constructeurs (par défaut, d'initialisation et de recopie), et les accesseurs aux différents attributs.
- Définir les méthodes ToString et Equals, la première retourne une chaine de caractères représentant l'abscisse et l'ordonné du vecteur comme suit : X = 1.5 - Y=2, la deuxième retourne True lorsque l'abscisse et l'ordonné des deux vecteurs sont égaux.
- Définir une méthode Norme qui retourne la norme d'un vecteur, cette méthode peut être redéfinie dans les classes dérivées.
- Définir une classe Vecteurs3D dérivée de la classe Vecteur2D qui contient, en plus des propriétés de la classe de base, la propriété Z symbolisant la troisième dimension.
- Définir les constructeurs (par défaut, d'initialisation et de recopie) de cette classe, ainsi que les accesseurs des propriétés.
- Redéfinir les méthodes ToString et Equals pour intégrer la troisième propriété.
- Redéfinir la méthode Norme pour qu'elle retourne la norme d'un vecteur dans l'espace.
- Définir un programme de test permettant de créer deux objets de type Vecteurs2D et deux autres de type Vecteurs3D. Afficher les informations de tous les objets, et tester les méthodes Equals et Norme. Afficher le nombre d'objet créés.

**Exercice 5 :**

- Un compte bancaire possède à tout moment une donnée : son solde. Ce solde peut être positif (compte créditeur) ou négatif (compte débiteur).
- Chaque compte est caractérisé par un code incrémenté automatiquement.
- le code et le solde d'un compte sont accessibles en lecture seulement.

- A sa création, un compte bancaire a un solde nul et un code incrémenté.
- Il est aussi possible de créer un compte en précisant son solde initial.
- Utiliser son compte consiste à pouvoir y faire des dépôts et des retraits. Pour ces deux opérations, il faut connaître le montant de l'opération.
- L'utilisateur peut aussi consulter le solde de son compte par la méthode ToString().
- Un compte Epargne est un compte bancaire qui possède en plus un champ « Taux Intérêt=6 » et une méthode calculIntérêt() qui permet de mettre à jour le solde en tenant compte des intérêts.
- Un ComptePayant est un compte bancaire pour lequel chaque opération de retrait et de versement est payante et vaut 5 DH.

Questions :

1. Définir la classe Compte.
2. Définir la classe CompteEpargne.
3. Définir la classe ComptePayant.
4. Créer un programme permettant de tester ces classes avec les actions suivantes:

    - Créer une instance de la classe Compte , une autre de la classe CompteEpargne et une instance de la classe ComptePayant
    - Faire appel à la méthode deposer() de chaque instance pour déposer une somme quelconque dans ces comptes.
    - Faire appel à la méthode retirer() de chaque instance pour retirer une somme quelconque de ces comptes.
    - Faire appel à la méthode calculInterêt() du compte Epargne.
    - Afficher le solde des 3 comptes.

**Exercice 6 :**

Ecrivez une classe Bâtiment avec l'attribut adresse. La classe Bâtiment doit disposer des constructeurs suivants:

- Batiment(),
- Batiment (adresse).

La classe Bâtiment doit contenir des accesseurs et mutateurs (ou propriétés) pour les différents attributs. La classe Bâtiment doit contenir une méthode *ToString ()* donnant une représentation du Bâtiment.

Ecrivez une classe Maison héritant de Bâtiment avec l'attribut NbPieces (Le nombre de pièces de la maison). La classe Maison doit disposer des constructeurs suivants:

- Maison(),
- Maison(adresse, nbPieces).

La classe Maison doit contenir des accesseurs et mutateurs (ou des propriétés) pour les différents attributs. La classe Maison doit contenir une méthode *ToString ()* donnant une représentation de la Maison.

Ecrivez un programme afin de tester ces deux classes.

**Exercice 7 :**

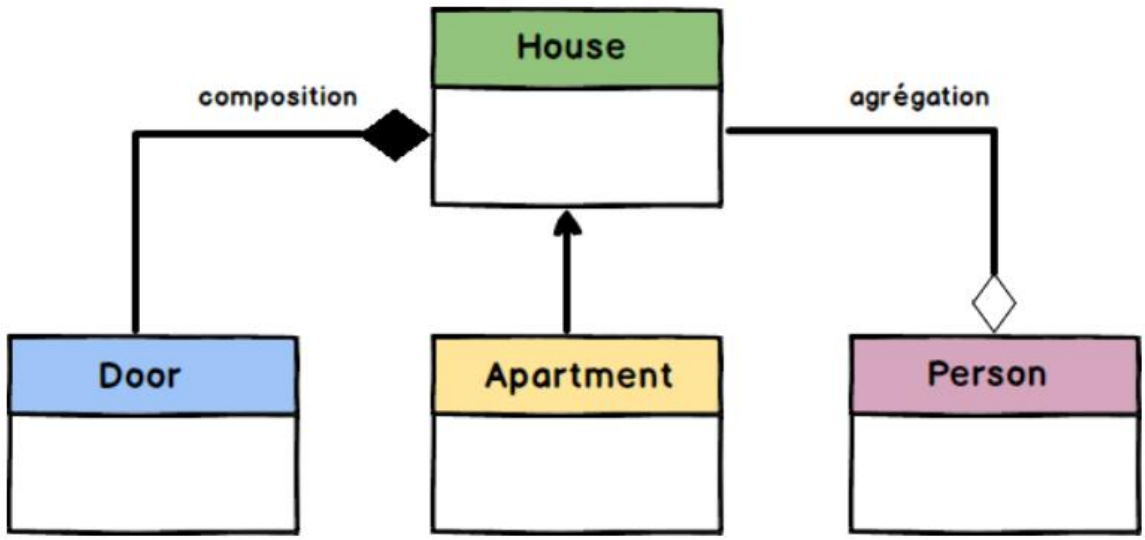
Créez une classe « House », avec un attribut « surface », un constructeur qui définit sa valeur et une méthode « Display » pour afficher « Je suis une maison, ma surface est de XXX m2 » (XXX: la valeur de surface). Incluez aussi des getters et des setters pour la surface.

La classe « House » contiendra une porte (Door). Chaque porte aura un attribut « color » (de type String), et une méthode « Display » qui affichera « Je suis une porte, ma couleur est bleu » (ou quelle que soit la couleur). Inclure un getter et un setter. Créez également la méthode « GetDoor » dans la classe « House ».

La classe « Apartment » est une sous-classe de la classe « House », avec une surface prédéfinie de 50m2.

Créez également une classe Person, avec un nom (de type String). Chaque personne aura une maison. La méthode « Display » pour une personne affichera son nom, les données de sa maison et les données de la porte de cette maison.

Écrivez un Main pour créer un Appartement, une personne pour y vivre et pour afficher les données de la personne.



**Exercice 8 :**

On souhaite créer une application POO avec C# permettant de calculer le cout de transport des marchandises transportées dans des cargaisons. Une marchandise est définie par son numéro, son poids et son volume. Une cargaison peut transporter plusieurs marchandises. Il existe deux types de cargaisons : Routière et aérienne. Chaque cargaison est définie par la distance du parcours. Les opérations de cette classe sont :

- Ajouter une marchandise.
- Afficher toutes les marchandises.
- Consulter une marchandise sachant son numéro.
- Consulter le volume total des marchandises.
- Consulter le poids total des marchandises.
- Calculer le cout de la cargaison qui dépend du type de cargaison.

Une cargaison aérienne est une cargaison dont le coût est calculé selon la formule suivante :

- Cout= 10*distance*poids total des marchandises, si le volume total < 80000
- Cout=12*distance*poids total des marchandises, si le volume total > =80000

Une cargaison routière est une cargaison dont le cout est calculé selon la formule suivante :

- Cout = 4*distance*poids total, si le volume total > 380000
- Cout = 4*distance*poids total, si le volume total > 380000