

Exercice 1

En vue de la gestion d'une librairie, on veut écrire une application orientée objet pour organiser des livres. L'application sera basée sur 2 classes :

Une classe appelée **Auteur** contient :

- 3 variables d'instance privées : nom (de type **string**), email (de type **string**), et sexe (de type **char** soit 'm' ou 'f') ;
- 1 constructeur qui initialise tous les arguments **Auteur(string nom, string email, char sexe)**;
- 2 accesseurs de lecture seule pour nom et sexe, et 1 accesseur de lecture et modification pour email;
- 1 méthode **bool Equals(Auteur auteur)** qui teste l'égalité de 2 auteurs en se basant sur leurs e-mails;
- 1 méthode **ToString()** qui retourne les informations sur chaque instance sous la forme : « nom (sexe) à email », comme indiqué dans cet exemple : « ALAOUI Mohamed (m) à alaoui-med@mail.ma ».

Une classe appelée **Livre** contient :

- 4 variables d'instance privées : titre (de type **string**), auteurs (de type **List<Auteur>** en considérant un ou plusieurs auteurs par livre), prix (de type **double**), et quantité_en_stock (de type **int**);
- 2 constructeurs : **Livre(string titre, List<Auteur> auteurs, double prix)** et **Livre(string titre, List<Auteur> auteurs, double prix, int quantité_en_stock)**;
- 2 accesseurs de lecture seule pour titre et **List<Auteur>**, et 2 accesseurs de lecture et modification pour prix et quantité_en_stock;
- 1 méthode **bool Equals(Livre livre)** qui teste l'égalité de 2 livres en se basant sur leurs titres et leurs auteurs;
- 1 méthode **ToString()** qui retourne les informations sur chaque instance sous la forme :

« titre de nomAuteur et ..., coûte prix DH avec quantité_en_stock exemplaires en stock. », comme indiqué dans cet exemple :

« Exercices en C# de ALAOUI Mohamed et ALAMI Meryem, coûte 200.00 DH avec 64 exemplaires en stock. »,

Travail à réaliser :

- a. Écrire les 2 classes **Auteur** et **Livre**.

b. Ajouter un jeu de tests.

Exercice 2

On désire réaliser une application orientée objet permettant de créer des formes géométriques à partir de points.

Une classe appelée **Point** permet de manipuler les points d'un plan, et contient :

- 2 variables d'instance privées : abscisse et ordonnée (de type **int**) qui doivent être comprises entre -999 et 999;
- 4 constructeurs : 1 constructeur par défaut sans argument, 1 constructeur à 1 argument, 1 constructeur à 2 arguments et le constructeur de copie **Point(Point p)**.
- 2 accesseurs pour abscisse et ordonnée;
- 1 méthode **ToString()** qui retourne les coordonnées d'un point;
- 1 méthode **float Norme(Point p)** qui permet de calculer la norme du vecteur constitué du point courant et du point passé en argument.

Rappel : Si A représente le point courant, B représente le point passé en argument, x représente l'abscisse et y représente l'ordonnée :

$$\|\overrightarrow{AB}\| = \sqrt{(x_B - x_A)^2 + (y_B - y_A)^2}$$

- 1 méthode **bool Equals(Point p)** qui permet de tester la coïncidence du point courant avec le point passé en argument;
- 1 méthode **void Translater(int dx, int dy)** qui permet de translater un point en ajoutant à l'abscisse la valeur dx et à l'ordonnée la valeur dy;
- 3 méthodes :
void setCoordonnées(double abscisse, double ordonnée),
void setCoordonnées(Point p),
et **void setCoordonnées(int abscisse, int ordonnée)**.
(Penser à utiliser la fonction **Math.Round(double argument)** pour arrondir les valeurs à l'entier);
- 2 méthodes : **double Distance(int abscisse, int ordonnée)** et **double Distance(Point p)**, permettant de calculer la distance entre le point courant et les coordonnées passées en arguments.

Rappel : Dans un espace de dimension n, pour 2 points ayant respectivement les coordonnées **(x₁, x₂, ..., x_n)** et **(y₁, y₂, ..., y_n)**, la distance entre les 2 points est :

$$\sqrt{\sum_{i=1}^n |x_i - y_i|^2}$$

Une classe appelée **Triangle** contient :

- 3 variables d'instance privées : A, B et C (de type **Point**) ;
- 2 constructeurs :
Triangle(int xA, int yA, int xB, int yB, int xC, int yC)
 et **Triangle**(**Point** A, **Point** B, **Point** C);
- 1 méthode **double** getPérimètre() qui retourne le périmètre d'un triangle (utiliser la méthode Distance(**Point** p) de la classe **Point**;
- 1 méthode **string** getTriangleType() qui retourne le type du triangle :
 "équilatéral" si ses 3 côtes sont égaux, "isocèle" si 2 de ses 3 côtes sont égaux, ou "scalène" dans le cas échéant.
- 1 méthode ToString().

Travail à réaliser :

a. Écrire les 2 classes **Point** et **Triangle**.

b. Modifier la classe **Cercle** (réalisée précédemment) en ajoutant :

- 1 variable d'instance privée : centre (de type **Point**) qui représente le centre d'un cercle;
- 3 constructeurs : 1 constructeur à 3 arguments, le constructeur de copie **Cercle**(**Cercle** c) et un constructeur **Cercle**(**double** rayon, **string** couleur, **int** abscisse, **int** ordonnée);
- 1 accesseur pour centre.

c. Mettre à jour la méthode ToString() de la classe **Cercle** pour retourner les informations sur une instance de la classe **Cercle** comme indiqué dans cet exemple :

« Le cercle de centre (5 ,7) a un rayon de 1 et une surface de 3,14. »

d. Écrire une classe **Cylindre** qui contient :

- 2 variables d'instance privées : base_cylindre (de type **Cercle**) qui représente la base du cylindre, et hauteur (de type **double**) avec valeur par défaut de 1.0;
- 4 constructeurs : 1 constructeur à 2 arguments, le constructeur de copie **Cylindre** (**Cylindre** c), 1 constructeur sans arguments **Cylindre**(), et 1 constructeur **Cylindre** (**double** rayon, **double** hauteur);
- 1 accesseur pour hauteur.

- 1 méthode `double` `getVolume()` qui retourne le volume d'un cylindre (Utiliser la méthode `getSurface()` de la classe `Cercle`).
- 1 méthode `ToString()`.