3 TP 3

Exercice 19. Créer une classe qui permet de demander à l'utilisateur d'entrer son année de naissance puis calcule à partir de l'année courante l'âge de la personne en question et l'affiche. Votre fonction `Main` ne fait aucune des tâches citées ci-dessus. Dans la classe effectuant tout le travail, vous séparerez chacune des actions exigées dans une méthode **non statique** différente (il y en a 3 en tout).

Utilisez `DateTime.Today.Year` pour connaître l'année actuelle.

Exercice 20.

1. Créez une classe `Produit` possédant les attributs suivants : `string nom; double prix;` dont la signification est évidente (un prix négatif correspond à un prix inconnu).
2. Rédéfinissez la méthode `ToString` (méthode virtuelle héritée de la classe `object`) afin qu'elle renvoie une description du produit (sous forme d'une chaîne de caractères).
3. Ajouter une méthode `Saisie` qui demande à l'utilisateur de saisir les deux caractéristiques du produit.
4. Pour représenter un catalogue de produits, créez une classe `Catalogue` qui aura comme attribut un tableau d'objets de la classe `Produit` et aussi une variable contenant l'année correspondant au catalogue.
5. Rédéfinissez la méthode `ToString` pour la classe `Catalogue` de sorte qu'elle renvoie le contenu du catalogue (on fera appel à la méthode `ToString` de la classe `Produit`).
6. Ajouter une méthode `Saisie` à la classe `Catalogue` qui demande à l'utilisateur de saisir les caractéristiques du catalogue. Cette méthode devra faire appel à la méthode `Saisie` de la classe `Produit`.

Exercice 21.

1. Créez une classe `ProduitAlimentaire` qui hérite de la classe `Produit` et qui possède un attribut supplémentaire : la date de péremption représenté par une variable de type `DateTime` (c'est une structure qui permet de représenter des dates et qui possède la méthode `TryParse` pour convertir une chaîne de caractère).
2. Redéfinir les méthodes `ToString` et `Saisie` pour cette sous-classe (en prenant soin de modifier la classe `Produit` afin de déclarer `Saisie` comme une méthode virtuelle). On fera appel à la méthode de la classe mère avec le mot-clef `base`.
3. Modifier la méthode `Saisie` de la classe `Catalogue` de sorte que l'utilisateur puisse fabriquer un catalogue contenant à la fois des instances de la classe `Produit` et de la classe `ProduitAlimentaire`.

Exercice 22. Créez un programme qui donne la décomposition d'un nombre entier n en produit de nombres premiers.

Pour cela fabriquez une classe possédant deux attributs : une variable de type `long` correspondant à l'entier n et un tableau de type `long[]` correspondant aux facteurs premiers. Le calcul de la décomposition en facteurs premiers se fera dans le constructeur : utiliser que le

plus petit entier k plus grand que 2 qui divise n est un facteur premier de n et donc, une fois que k est déterminé on est ramené à décomposer n/k (on peut donc concevoir une procédure récursive).

Exercice 23. A partir de n vecteurs $a_1, \ldots, a_n$ de $\mathbb{R}^d$ linéairement indépendants, formant une base d'un sous-espace vectoriel F de $\mathbb{R}^d$, l'algorithme de Gram-Schmidt permet de construire une base orthonormale $(b_1, \ldots, b_n)$ de F. Les b_i sont définies récursivement de la manière suivante :

$$b_1 = a_1/\sqrt{\langle a_1, a_1\rangle}$$
$$u_k = a_k - \sum_{\ell=1}^{k-1} \langle a_k, b_\ell\rangle b_\ell, \quad \text{et} \quad b_k = u_k/\sqrt{\langle u_k, u_k\rangle}, \quad 2 \le k \le n$$

où $\langle , \rangle$ désigne le produit scalaire euclidien de $\mathbb{R}^d$.

1. Dans le but de représenter un vecteur de $\mathbb{R}^d$, écrire une classe `Vecteur` possédant un attribut `private double[] elts` qui servira à stocker les composantes du vecteur.
2. Ecrire un constructeur `public Vecteur(int dim)` prenant comme argument la dimension d.
3. Ecrire une propriété `public Length` permettant d'obtenir d.
4. Définir un indexeur (`public`) permettant de manipuler les composantes du vecteur. On choisira comme indice un entier entre 1 et d.
5. Ecrire une méthode statique qui renvoie le produit scalaire de deux vecteurs (représentés par deux objets de la classe `Vecteur`).
6. Ecrire deux méthodes statiques, une qui renvoie la somme de deux vecteurs, l'autre le produit d'un vecteur et d'un réel (type `double`).
7. Ecrire une méthode statique
   `public static void GramSchmidt(Vecteur[] a)`
   qui met en oeuvre l'algorithme de Gram-Schmidt. L'argument est un tableau de `Vecteur` représentant $a_1, \ldots, a_n$. Après l'appel de la méthode, ce tableau de `Vecteur` doit correspondre à $b_1, \ldots, b_n$. (Utiliser la méthode `Math.Sqrt` de la classe statique `Math` pour calculer une racine carrée)
8. Pour $n = d$ et
   $$a_i = \Big(\frac{1}{i+j-1}\Big)_{1 \le j \le d}$$
   faire afficher dans la méthode `Main` les produits scalaires $\langle b_i, b_{i+1}\rangle$ pour $d = 5$ puis $d = 10$.