Exercices de cours de C# .NET

OBJECTIFS

Ce document propose des exercices simples de cours pour comprendre les notions vues en cours. Ces exercices sont triés par ordre chronologique d'apparition dans le cours.

TABLE DES MODIFICATIONS

Auteur	Modifications	Version	Date
Marc Chevaldonné	Première version	1.0	06 août 2012
Marc Chevaldonné	Ajout des exercices du cours 3	1.1	27 septembre 2012
Marc Chevaldonné	Ajout des exercices du cours 4	1.2	4 octobre 2012
Marc Chevaldonné	Ajout des exercices du cours 5	1.3	11 octobre 2012
Marc Chevaldonné	Ajout des exercices des cours 6 et 7	1.4	01 novembre 2012

Cours 1

EXERCICE 01_01

tableaux

Soit un tableau T ayant N éléments entiers quelconques où N est une constante entière. Écrire un programme en C# permettant de :

- faire la saisie du tableau,
- calculer la somme de ses éléments.

EXERCICE 01_02

tableaux à deux dimensions

Soit un tableau T à 5 lignes et 6 colonnes. Chaque élément est un entier. Écrire un programme en C# permettant de :

- faire la saisie du tableau,
- calculer la somme de ses éléments.

EXERCICE 01_03

tableaux à deux dimensions

Créer dans un tableau P à 2 dimensions les 10 premières lignes du triangle de Pascal. Chaque élément du triangle de Pascal est obtenu par la formule :

$$P[L, C] = P[L-1, C-1] + P[L-1, C] \qquad \text{avec } L, C > 1$$

La première colonne est initialisée à 1, le reste du tableau à 0

Résultat :

1	0	0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0
1	2	1	0	0	0	0	0	0	0
1	3	3	1	0	0	0	0	0	0
1	4	6	4	1	0	0	0	0	0
1	5	10	10	5	1	0	0	0	0
1	6	15	20	15	6	1	0	0	0
1	7	21	35	35	21	7	1	0	0
1	8	28	56	70	56	28	8	1	0
1	9	36	84	126	126	84	36	9	1

On affichera uniquement le triangle de Pascal à la fin du programme.

EXERCICE 01_04

valeurs et références

- Faire une méthode statique (et l'utiliser) prenant deux paramètres par valeur et retournant un réel égal à la somme des deux.
- Faire une méthode statique (et l'utiliser) ne rendant rien et prenant deux paramètres par valeur et un troisième paramètre par référence qui devra être égal à la fin de la méthode à la somme des deux premiers. Faites cet exercice une fois avec `ref` et une autre fois avec `out`.
- Faire une méthode statique (et l'utiliser) prenant un tableau d'entiers en paramètre et rendant la somme des éléments du tableau.
- Faire une méthode statique (et l'utiliser) prenant un nombre indéterminé d'entiers en paramètres et rendant la somme des ces entiers.

EXERCICE 01_05

passage de paramètres par référence

On considère une suite (un) définie par :

$u_n = u_{n-1} + u_{n-2} \quad (n \geq 3)$

$u_1 = 1$

$u_2 = 2$

appelée suite de Fibonacci.

- Écrire le programme permettant de déterminer la valeur et le rang du premier terme de cette suite supérieur ou égal à 10000, à l'aide d'une boucle `while`.
- Recommencer en utilisant une méthode et des passages de paramètres par référence, pour remplacer les instructions de chaque itération de la boucle `while`, et mettre à jour toutes les variables nécessaires.

EXERCICE 01_06

foreach

Calculez le maximum de trois nombres entiers entrés au clavier à l'aide d'un `foreach`. Pour cela, utilisez un tableau pour stocker les nombres entrés au clavier.

EXERCICE 01_07

foreach

Écrivez le programme qui calcule la somme des N premiers nombres entiers positifs à l'aide d'un `foreach`. Utilisez un tableau.

EXERCICE 01_08

for vs foreach

Soit un tableau d'entiers. À l'aide d'une boucle `for`, incrémentez chaque entier du tableau de 1. Essayez de réaliser la même opération avec une boucle `foreach`. Que constatez-vous ?

EXERCICE 01_09

classe et méthodes
surcharge de méthodes

Écrivez une classe `Voiture`. Les voitures de cette classe possèdent un régulateur de vitesse. Ajoutez à cette classe `Voiture` :

- un attribut `mVitesse` (entier compris entre 0 et 130 inclus),
- un *getter* rendant la vitesse de la voiture,
- un *setter* privé permettant de fixer une vitesse donnée et vérifiant que la vitesse rentrée est valide,
- une méthode permettant d'incrémenter la vitesse de 2 km/h
- et une méthode permettant de décrémenter la vitesse de 2 km/h.

Surchargez ensuite la méthode d'incrémentation en lui passant le nombre de vitesses à incrémenter.

Avant de coder cette classe, réalisez un diagramme de classes UML au brouillon. Réalisez ensuite un projet de test pour tester les méthodes.

EXERCICE 01_10

propriétés

Modifiez la classe `Voiture` de l'exercice précédent en utilisant une propriété à la place du *getter* public et du *setter* privé pour lire et écrire la vitesse.

EXERCICE 01_11

propriétés automatiques et calculées

Écrivez une classe `Vecteur`, contenant trois réels `X`, `Y`, `Z`. N'écrivez pas de membres et utilisez uniquement des propriétés automatiques. Ajoutez une propriété calculée `Norme` en lecture seule.

EXERCICE 01_12

propriétés automatiques et calculées

Écrivez une classe `Pavé`, contenant trois réels `Largeur`, `Longueur`, `Hauteur`. N'écrivez pas de membres et utilisez uniquement des propriétés automatiques. Ajoutez une propriété calculée `Volume` en lecture seule.

EXERCICE 01_13

indexeurs

Soit la classe `ClassementCurling` suivante. Cette classe contient deux tableaux de même taille contenant (tableau 1) les noms des pays dans l'ordre du classement de curling et (tableau 2) le points de ces pays.

```
public class ClassementCurling
{
   string[] mPays = new string[] {"Canada",
              "Scotland/Great Britain", "Norway", "Sweden",
              "Switzerland", "Germany", "USA", "France",
       "China", "Denmark", "Czech Republic",
              "New Zealand", "Italy", "Korea", "Finland",
              "Australia", "Russia", "Ireland", "Latvia"} ;

   int[] mPoints = new int[] {1064, 784, 774, 619, 550, 451,
              434, 426, 402, 384, 192, 179, 141, 128, 122,
       120, 115, 107, 105} ;
}
```

a) Ajoutez à cette classe un indexeur permettant de rendre le `i`ème pays dans le classement. Par exemple si `cc` est une instance de `ClassementCurling`, `cc[8]` vaut `China`. Cet indexeur vérifiera que l'index passé entre crochets est bien dans les limites du tableau, sinon il rendra `null`. Testez votre indexeur dans un projet de test.

b) Ajoutez à cette classe un indexeur permettant de rendre le nombre de points du pays dont le nom est passé entre crochets. Par exemple si `cc` est une instance de `ClassementCurling`, `cc[«Russia»]` rend 115.

Cet indexeur vérifiera que la chaîne de caractères passée entre crochets est bien dans le tableau, sinon il rendra `-1`. Testez votre indexeur dans un projet de test.

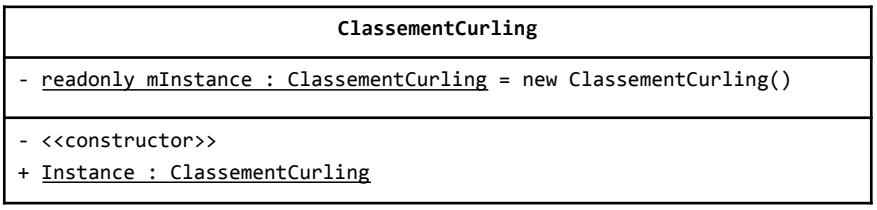
EXERCICE 01_14

static

Soit la classe `ClassementCurling` de l'exercice précédent. On veut s'assurer qu'une seule instance de cette classe peut exister. Modifiez la classe `ClassementCurling` en implémentant le *design pattern* singleton.

Note : Il n'est pas nécessaire d'avoir fait l'exercice 13 pour réaliser cet exercice.

Rappel sur le singleton avec un diagramme de classes UML (vous pourrez par exemple utiliser une propriété pour `Instance`*, et l'initialiseur pour construire le membre statique* `mInstance`*) :*



`Instance` *rend l'instance unique de* `ClassementCurling`*.*

EXERCICE 01_15

ToString

Reprenez la classe `Vecteur` de l'exercice 11 et réécrivez la méthode `ToString` pour qu'elle affiche :

`{ X, Y, Z }` où `X`, `Y`, `Z` sont remplacés par les valeurs numériques

EXERCICE 01_16

ToString

Reprenez la classe `Pavé` de l'exercice 12 et réécrivez la méthode `ToString` pour qu'elle affiche :

`L = L, l = l, h = h`, où `L`, `l`, `h` seront remplacés par les

longueur,

largeur, hauteur.

EXERCICE 01_17

struct

Reprenez la classe `Vecteur` de l'exercice 15 et faites les modifications qui s'imposent (aidez-vous du compilateur) pour en faire une structure.

EXERCICE 01_18

struct

Reprenez la classe `Pavé` de l'exercice 16 et faites les modifications qui s'imposent (aidez-vous du compilateur) pour en faire une structure.

EXERCICE 01_19

value vs reference

Reprenez la structure `Pavé` de l'exercice 18. Modifiez les propriétés de ses dimensions pour permettre leur modification par l'utilisateur de la structure (*ie* modifiez l'accessibilité des setters).

Écrivez un programme de test qui réalise les opérations suivantes :

- construction de `pavé1` de type `Pavé`, avec `L=30`, `l=20` et `h=10`,
- construction de `pavé2` de type `Pavé`, avec `L=25`, `l=15` et `h=5`,
- affichage du `ToString` de `pavé1` et `pavé2`,
- `pavé2 = pavé1`,
- affichage du `ToString` de `pavé1` et `pavé2`,
- modification des dimensions de `pavé2`,
- affichage du `ToString` de `pavé1` et `pavé2`.

Transformez la structure `Pavé` en classe (changez uniquement le mot clé `struct` en `class`) et observez les différences à l'exécution. Expliquez-les.

EXERCICE 01_20

write-once immutability

Soit la classe `Cible` suivante :

```
public class Cible
{
   public string Description
   {
      get;
      set;
   }

   public int Hauteur
   {
      get;
      private set;
   }

   public int Largeur
   {
      get;
      private set;
   }

   public void ChangeDimensions(int hauteur, int largeur)
   {
      Hauteur = hauteur;
      Largeur = largeur;
   }

   public Cible (string desc, int hauteur, int largeur)
   {
      Description = desc;
      Hauteur = hauteur;
      Largeur = largeur;
   }

   public override string ToString ()
   {
      return string.Format ("[Cible: Description={0},
                            Hauteur={1}, Largeur={2}]",
                            Description, Hauteur, Largeur);
   }
}
```

Soit la méthode `Main` de la classe `Program` suivante, qui utilise la classe `Cible` :

```
public static void Main (string[] args)
{
   Cible c = new Cible("Mickey", 100, 50);
   Console.WriteLine(c);
   c.Description = "King Kong";
   c.ChangeDimensions(200, 90);
   Console.WriteLine(c);
}
```

Faites de la classe `Cible` une structure immuable (*write-once immutable*) en faisant les modifications/suppressions qui s'imposent.

EXERCICE 01_21

write-once

immutability

Quelques questions tordues pour bien comprendre...

Soit la classe `Point` et la structure `Cercle` suivantes :

```
public class Point
{
   public float X
   {
      get { return mX; }
   }
   private float mX;

   public float Y
   {
      get { return mY; }
   }
   private float mY;

   public Point(float x, float y)
   {
      mX = x;
      mY = y;
   }

   public void Translate(float tx, float ty)
   {
      mX += tx;
      mY += ty;
   }
}

public struct Cercle
{
   public float Rayon
   {
      get { return mRayon; }
   }
   private readonly float mRayon;

   public Point Centre
   {
      get { return mCentre; }
   }
   private readonly Point mCentre;

   public Cercle(float rayon, Point centre)
   {
      mRayon = rayon;
      mCentre = centre;
   }
}
```

La classe `Point` est-elle immuable ? La structure `Cercle` est-elle immuable ? Justifiez. Changez la classe `Point` en une structure (en changeant simplement `class` en `struct`). `Point` est-elle immuable ? `Cercle` est-elle immuable ? Justifiez.

EXERCICE 01_22

write-once immutability

La structure `Point` suivante n'est pas immuable, à cause de sa méthode `Translate`. Pourquoi ?

```
public struct Point
{
   public float X
   {
      get
      {
         return mX;
      }
   }
   private float mX;

   public float Y
   {
      get
      {
         return mY;
      }
   }
   private float mY;

   public Point(float x, float y)
   {
      mX = x;
      mY = y;
   }

   public void Translate(float tx, float ty)
   {
      mX += tx;
      mY += ty;
   }
}
```

Rendez cette structure immuable (*write-once*) tout en conservant la méthode `Translate`. Que faut-il changer dans cette méthode pour que la structure soit immuable ?

EXERCICE 01_23

NullableType

Reprenez la structure `Cible` issue de l'exercice 20. Soit la classe `TirÀLaCarabine` présentée plus bas. Cette classe représente un jeu de 5 cibles à abattre avec une carabine, en pleine fête foraine. L'indexeur permet de modifier les cibles lorsqu'elles sont touchées. Malheureusement, `Cible` étant une structure, et donc un type valeur, la

valeur par défaut est une `Cible` sans `Description`, avec une `Hauteur` et une `Largeur` à `0`. Nous aurions préféré une `Cible` `null`. Ceci n'est possible qu'avec les types référence (classes). On peut toutefois le rendre possible avec un type valeur en utilisant un `NullableType`. Modifiez la classe `TirÀLaCarabine` pour qu'elle contienne 5 éléments de type «`Cible` ou `null`» et testez-la. Pour cela, utilisez la classe `Program` ci-dessous et modifiez le `ToString` de `TirÀLaCarabine` pour qu'il affiche `null` lorsqu'il n'y a pas de cible, ou la cible s'il y en a une.

```
public class TirÀLaCarabine
{
   const int NOMBRE_CIBLES = 5;

   Cible[] mCibles = new Cible[NOMBRE_CIBLES];

   public int NombreCibles
   {
      get { return mCibles.Length; }
   }

   public Cible this[int index]
   {
      get
      {
         if(index<0 || index >= mCibles.Length)
         {
            throw new IndexOutOfRangeException();
         }
         return mCibles[index];
      }
   }

   public TirÀLaCarabine(params Cible[] cibles)
   {
      int nbreMaxDeCibles = Math.Min(cibles.Length,
                                     mCibles.Length);
      for (int index = 0; index < nbreMaxDeCibles; index++)
      {
         mCibles[index] = cibles[index];
      }
   }

   public override string ToString ()
   {
      System.Text.StringBuilder sb =
                           new System.Text.StringBuilder();
      sb.AppendLine("Cibles :");
      foreach (Cible c in mCibles)
      {
         sb.AppendLine(c.ToString());
      }
      sb.AppendLine("**********************************");
      return sb.ToString();
   }
}
```

```
class Program
{
   static void Main(string[] args)
   {
      TirÀLaCarabine tir = new TirÀLaCarabine(
                  new Cible("Mickey", 90, 50),
                  new Cible("Gargamel", 150, 20),
                  new Cible("Schtroumpf à lunettes", 10, 5));
      Console.WriteLine(tir);
   }
}
```

EXERCICE 01_24

string.Format

Reprenez la classe `Vecteur` de l'exercice 15 et réécrivez la méthode `ToString` en utilisant un `string.Format` pour qu'elle affiche : `{ X, Y, Z }` où *X*, *Y*, *Z* sont remplacés par les valeurs numériques.

EXERCICE 01_25

StringBuilder

Reprenez la classe `Pavé` de l'exercice 16 et réécrivez la méthode `ToString` à l'aide d'un `StringBuilder` pour qu'elle affiche : `L = L, l = l, h = h`, où *L*, *l*, *h* seront remplacés par les longueur, largeur, hauteur.

EXERCICE 01_26

Format

Parse

Soit la classe `Personne` suivante :

```
public class Personne
{
	public Personne (string nom, bool porteDesLunettes,
	uint nbDentsPerdues, DateTime dateDeNaissance,
	TimeSpan tempsPerdu)
	{
		Nom = nom;
		PorteDesLunettes = porteDesLunettes;
		NbDentsPerdues = nbDentsPerdues;
		DateDeNaissance = dateDeNaissance;
		TempsPerdu = tempsPerdu;
	}

	public string Nom
	{  get; private set; }

	public bool PorteDesLunettes
	{  get; private set; }

	public uint NbDentsPerdues
	{  get; private set; }

	public DateTime DateDeNaissance
	{  get; private set; }

	public TimeSpan TempsPerdu
	{  get; private set; }

}
```

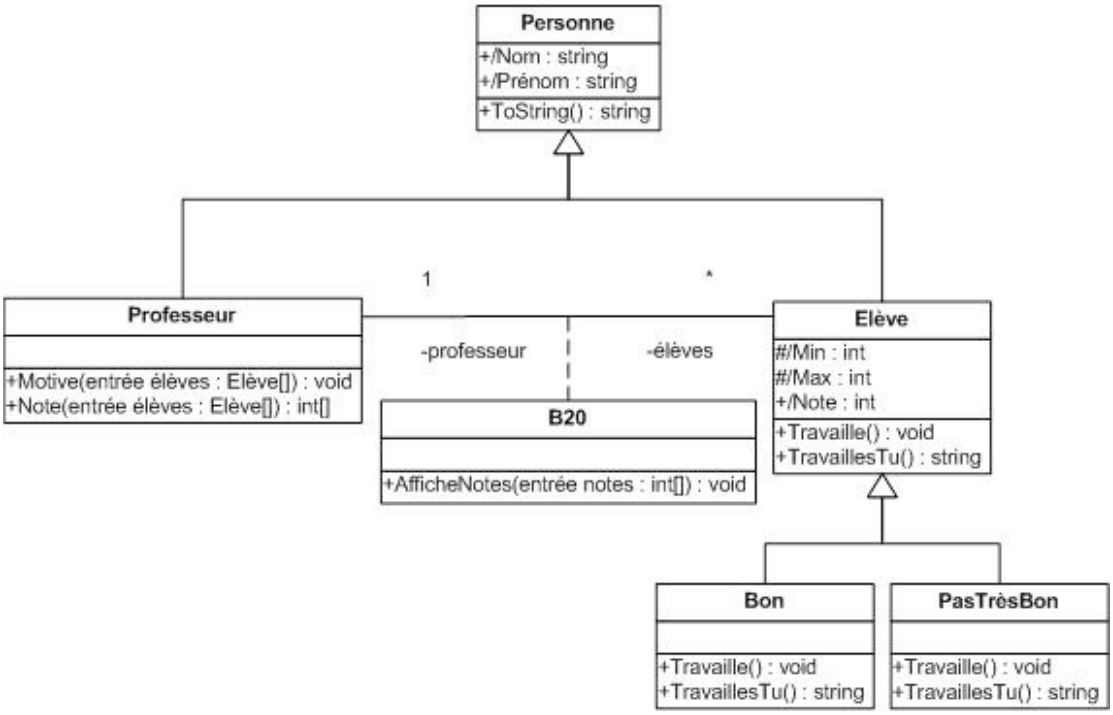
1) Écrivez une application Console permettant de construire une instance de `Personne` en posant les questions suivantes :

Question	Réponses autorisées	Exemple
`Comment vous appelez-vous ?`	n'importe quelle chaîne de caractères	`Bob l'Éponge`
`Portez-vous des lunettes ?`	true, false, True, False	`false`
`Combien de dents avez-vous perdues ?`	n'importe quel entier positif ou nul	`15`
`Quelle est votre date de naissance ?`	dates sous la forme : jj/mm/yyyy	`01/05/1999`
`Combien de temps avez-vous perdu ? (heures:minutes:secondes)`	temps sous la forme : hh:mm:ss	`16:13:54`

Vous devrez naturellement utiliser `TryParse` ou `Parse` sur les types `bool`, `uint`, `DateTime` et `TimeSpan` pour cela.

2) Ajoutez une méthode `ToString` à la classe `Personne` en utilisant un `StringBuilder` et les Format standards ou customs de `DateTime` et `TimeSpan` pour obtenir un résultat comme celui-ci (en reprenant l'exemple du tableau) :

```
Bob l'Éponge est né(e) le : samedi 1 mai 1999
Bob l'Éponge ne porte pas de lunettes
```




```
Bob l'Éponge a perdu 15 dents
Bob l'Éponge a perdu 16 heure(s) et 13 minute(s)
```

EXERCICE 01_27

Math

Random

Écrivez un programme qui affiche à l'écran un nombre entier x choisi aléatoirement entre `-100` et le nombre entier le plus grand autorisé pour un `int` et son image f(x) réelle ou f est la fonction continue par morceaux suivante :

$f(x) = 2x + 5e^x$ si $x < -1$

$f(x) = 3x^2 - 4\sqrt{(|5x-1|)}$ si $x >= -1$

Cours 2

EXERCICE 02_01

héritage
réécriture d'une méthode (new)
typage statique

On souhaite réaliser un programme qui simule une relation professeur-élèves dans la salle B20, comme proposé dans le diagramme de classes partiel ci-dessous.

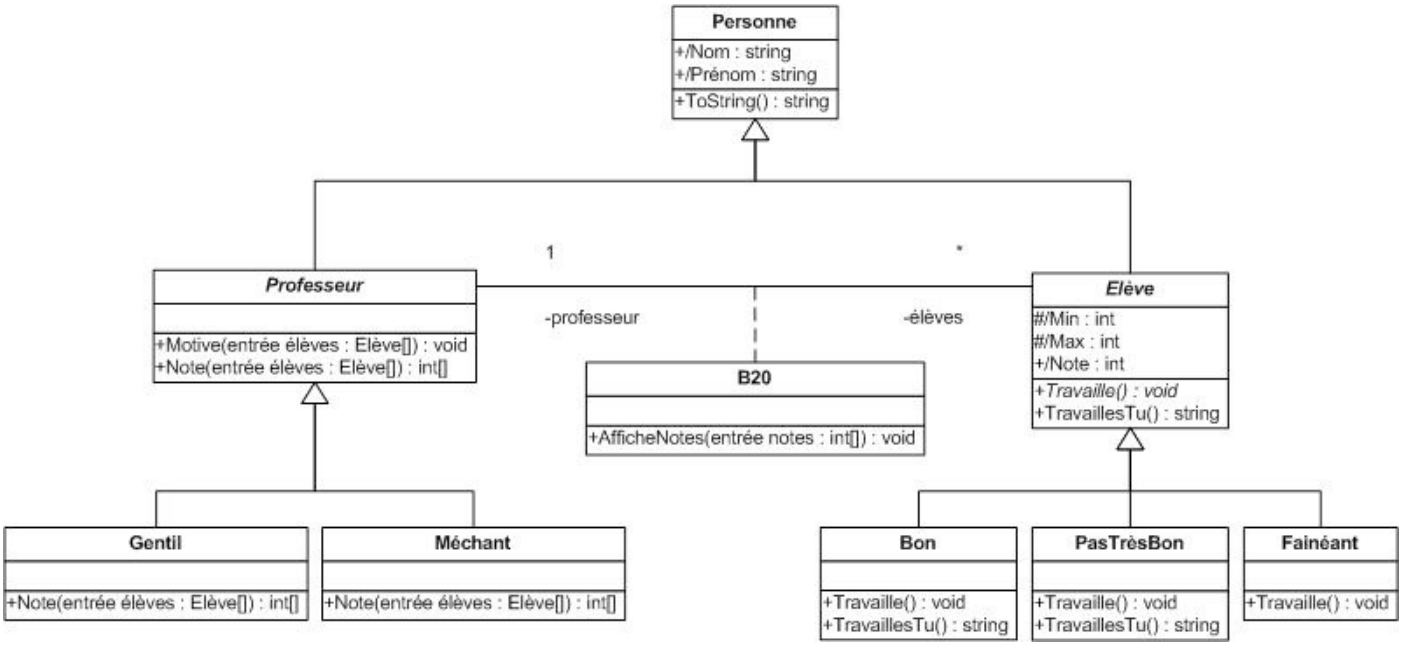
Une `Personne` possède un `Nom` et un `Prénom`.
Les classes `Professeur` et `Elève` sont des spécialisations de la classe `Personne`.
Un `Elève` possède deux propriétés `Min` et `Max` représentant respectivement la note minimale et la note maximale qu'il obtiendra lors de la notation.
La propriété calculée `Note` de `Elève` choisit un nombre entier aléatoire entre `Min` et `Max`.
Un `Elève` peut travailler, ce qui augmente ses propriétés `Min` et `Max`.
Un `Elève` peut répondre à la question «travailles-tu ?» en renvoyant un message sous la forme d'une chaîne de caractères.

Un `Professeur` peut motiver des `Elèves`. L'appel de la méthode `Motive(Elève[])` appelle la méthode `Travaille()` des `Elèves` passés en paramètres.
Un `Professeur` peut noter des `Elèves`. L'appel de la méthode `Note(Elève[])` récupère les notes des `Elèves` passés en paramètres (via leur propriété calculée `Note`).
Les différents élèves sont généralisés par la classe `Elève` et celle-ci est spécialisée dans les sous-classes `Bon` et `PasTrèsBon` qui réécrivent les méthodes Travaille et TravaillesTu ; leurs différences sont présentées dans le tableau suivant :

```
Notes du Professeur Prénom Nom :
Prénom Nom : Note/20 ; travailles-tu ? réponse
Prénom Nom : Note/20 ; travailles-tu ? réponse
Prénom Nom : Note/20 ; travailles-tu ? réponse
```

	Elève	Bon	PasTrèsBon
Min initial	0	8	0
Max initial	0	12	4
Travaille	augmente Min de 1 augmente Max de 2	Travaille deux fois plus qu'un Elève	augmente Min de 1 si Min+Max est pair augmente Max de 1 si Min+Max est impair
TravaillesTu	«*Un peu*» si la note est inférieure à 5 «*Beaucoup*» si la note est inférieure à 10 «*Passionnément*» si la note est inférieure à 15 «*À la folie*» pour les autres notes	«*Oui*»	«*Oui, mais j'ai du mal*»

Pour toutes les classes `Elève`, le retour de la méthode `TravaillesTu` est suivi de «*`[Min, Max]`*» où *`Min`* et *`Max`* sont remplacés respectivement par les valeurs des propriétés `Min` et `Max` (pour permettre de vérifier les résultats dans l'affichage Console).
La classe `B20`, qui remplace la classe `Program` et possède donc la méthode statique `Main`, créera une instance de `Professeur` et un tableau de 6 `Elèves` en utilisant les données du tableau suivant :



Type	Prénom	Nom
Professeur	Roger	Rabbit
Elève	Schtroumpf	Normal

Type	Prénom	Nom
Elève	Schtroumpf	Normal 2
Bon	Schtroumpf	Bon

Type	Prénom	Nom
Bon	Schtroumpf	Bonn 2
PasTrèsBon	Schtroumpf	PasTrèsBon
PasTrèsBon	Schtroumpf	PasTrèsBon 2

La classe `B20` possède également une méthode `AfficheNotes` qui permet l'affichage des notes attribuées dans une interface de type Console. Cette méthode affichera :

- Roger Rabbit note ses élèves,
- B20 affiche les notes,
- Roger Rabbit motive 5 fois ses élèves,
- Roger Rabbit note ses élèves,
- B20 affiche les notes,
- Roger Rabbit motive 5 fois ses élèves,
- Roger Rabbit note ses élèves,
- B20 affiche les notes,
- Roger Rabbit motive 5 fois ses élèves,
- Roger Rabbit note ses élèves,
- B20 affiche les notes.

La méthode `Main` réalisera ensuite les opérations suivantes :

Lancez plusieurs fois le programme et observez les résultats. Remarquez notamment que le tableau d'`Elèves` est de type `Elève[]`, alors que les six instances ont été construites avec des types différents : `Elève`, `Bon`, `PasTrèsBon`. Toutefois, les méthodes `Travaille` et `TravaillesTu` appelées sont toujours celles de `Elève`. Il s'agit bien d'un typage statique.

Personne
+/Nom : string
+/Prenom : string
+ToString() : string
Professeur
+Motive(entrée élèves : Elève[]) : void
+Note(entrée élèves : Elève[]) : int[]
+DemandeUnCafé(entrée serveur : ICafé) : string
Gentil
+Note(entrée élèves : Elève[]) : int[]
+PrépareLeCafé() : string
Méchant
+Note(entrée élèves : Elève[]) : int[]
ICafé
B20
+AfficheNotes(entrée notes : int[]) : void
-professeur
1
-élèves
*
Elève
#/Min : int
#/Max : int
+/Note : int
+Travaille() : void
+TravaillesTu() : string
Bon
+Travaille() : void
+TravaillesTu() : string
PasTrèsBon
+Travaille() : void
+TravaillesTu() : string
Fainéant
+Travaille() : void
Tricheur
+Travaille() : void
+Triche() : void
Fourbe
+Travaille() : void
+TravaillesTu() : string
+Triche() : void
+PrépareLeCafé() : string
Fayot
+PrépareLeCafé() : string
ICafé
«interface»
ITricheur
+Triche() : void
«interface»
ICafé
+PrépareLeCafé() : string

EXERCICE 02_02

héritage
méthodes virtuelles
polymorphisme
typage dynamique

Reprenez l'exercice précédent et rendez les méthodes `Travaille` et `TravaillesTu` dans la classe `Elève`, virtuelles. Effectuez les modifications qui s'imposent et relancez le programme. Que constatez-vous ?

Remarquez que le tableau d'`Elèves` est toujours de type `Elève[]` et les instances sont toujours construites avec des types différents : `Elève`, `Bon`, `PasTrèsBon`. Mais désormais, les méthodes `Travaille` et `TravaillesTu` dépendent du type utilisé à la construction (type dynamique) et pas celui déclaré (type statique, ici `Elève`). `Professeur` appelle la méthode `Travaille` d'un `Elève`, sans connaître son type, et pourtant, il appelle la méthode `Travaille` de ce type concret qui lui est inconnu. Vous venez de mettre en pratique le polymorphisme et le typage dynamique.

EXERCICE 02_03

héritage
méthodes virtuelles
méthodes abstraites
classes abstraites

Reprenez l'exercice précédent et rendez la méthode `Travaille` dans la classe `Elève`, abstraite. Effectuez les modifications qui s'imposent.

Ajoutez une nouvelle classe `Fainéant`, sous-classe de `Elève`, avec les caractéristiques suivantes :

	Fainéant
Min initial	0
Max initial	0
Travaille	a 1 chance sur 3 de : • augmenter Min de 1 • et augmenter Max de 2
TravaillesTu	même réponse qu'Elève

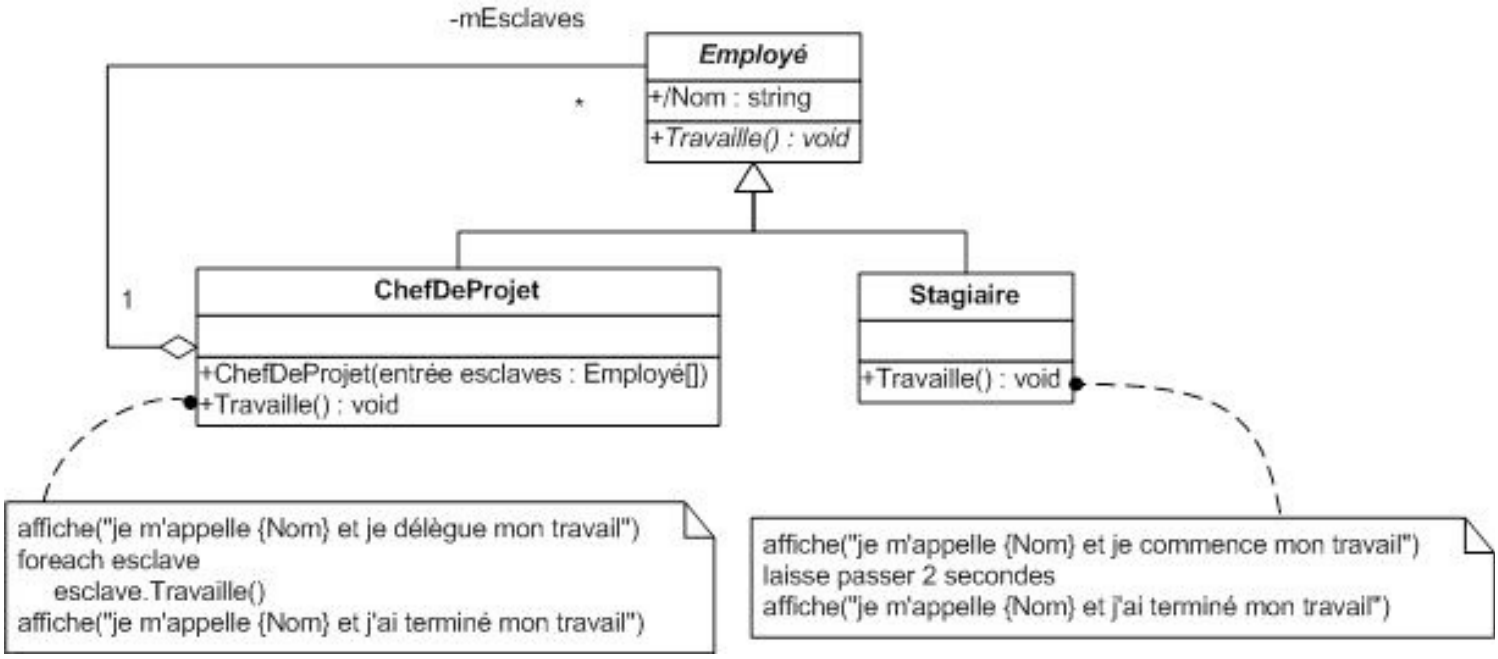
Modifiez le tableau d'`Elèves` (toujours de type `Elève[]`) pour avoir une instance de chaque type concret.

Rendez maintenant la méthode `Note` dans la classe `Professeur`, virtuelle et la classe `Professeur` abstraite. Ajoutez deux nouvelles classes, sous-classes de `Professeur` : `Gentil` et `Méchant`.

Type	Note
`Gentil`	Note les élèves comme `Professeur` puis rajoute un point à chaque `Elève` ayant une note strictement inférieure à 20.
`Méchant`	Note les élèves comme `Professeur` puis enlève un point à chaque `Elève` ayant une note strictement supérieure à 0.

Le diagramme de classes ci-dessous résume toutes les modifications.
Modifiez ensuite la méthode `B20.Main` pour que :

- un `Professeur Gentil` «*`Roger Rabbit`*» note les élèves,
- `B20` affiche les notes,



- *`Roger Rabbit`* motive 5 fois les élèves,
- *`Roger Rabbit`* note les élèves,
- `B20` affiche les notes,
- *`Roger Rabbit`* motive 5 fois les élèves,
- *`Roger Rabbit`* note les élèves,
- `B20` affiche les notes,
- *`Roger Rabbit`* tombe malade et est remplacé par un `Professeur Méchant` «*`Harvey Dent`*»,
- *`Harvey Dent`* motive 5 fois les élèves,
- *`Harvey Dent`* note les élèves,
- `B20` affiche les notes.

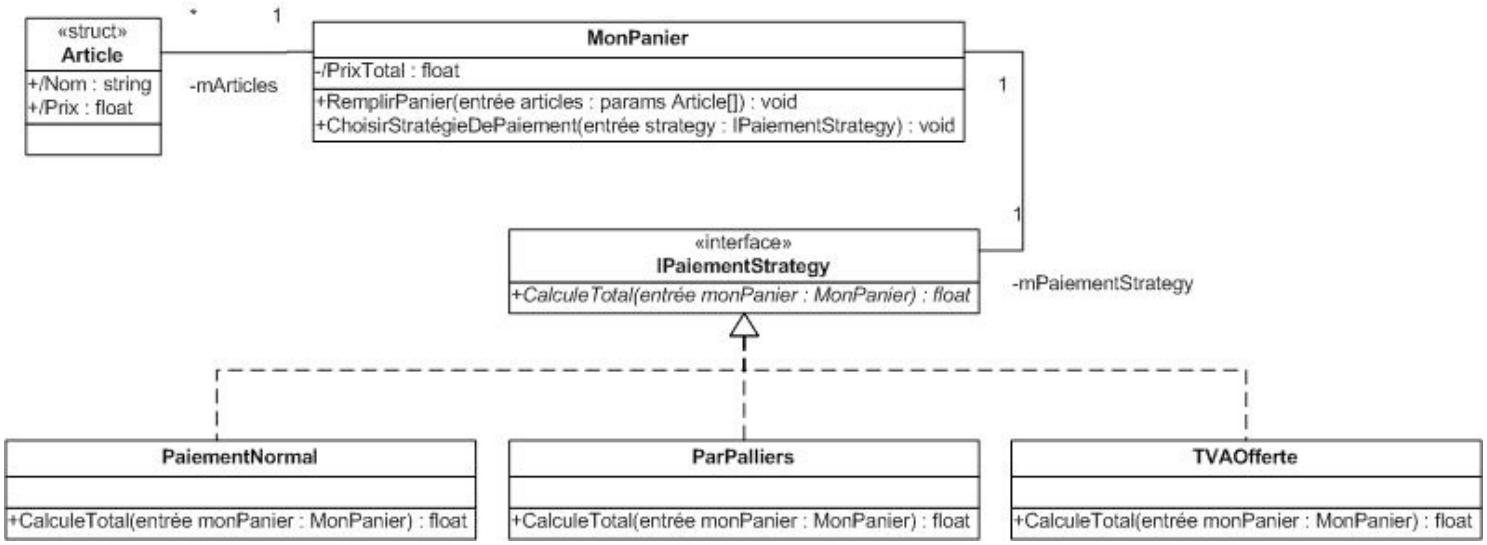
Constatez que si une classe possède une méthode abstraite, alors la classe doit être déclarée comme abstraite et ne peut pas être instanciée. Constatez que si une classe est déclarée comme abstraite, elle ne peut pas être instanciée mais ne possède pas nécessairement de méthode abstraite.

Dans une classe abstraite, on pourra donc utiliser des méthodes virtuelles pour faire des implémentations par défaut (cf. `TravaillesTu` dans `Elève` : `Fainéant` n'a pas besoin de la réécrire) et des méthodes abstraites pour forcer leur réécriture (cf. `Travaille` dans `Elève` et dans toutes ses sous-classes).

EXERCICE 02_04

interfaces *implémentation d'une interface*

implémentation d'une interface



Reprenez l'exercice précédent et faites les modifications résumées dans le diagramme de classes ci-dessous et les explications suivantes.

1. Ajoutez une interface `ITricheur` possédant une méthode `Triche`.

Créez une nouvelle sous-classe `Tricheur` de `Elève`, implémentant l'interface `ITricheur` et réalisant les modifications suivantes :

	Tricheur
Travaille	vide
TravaillesTu	même réponse qu'Elève

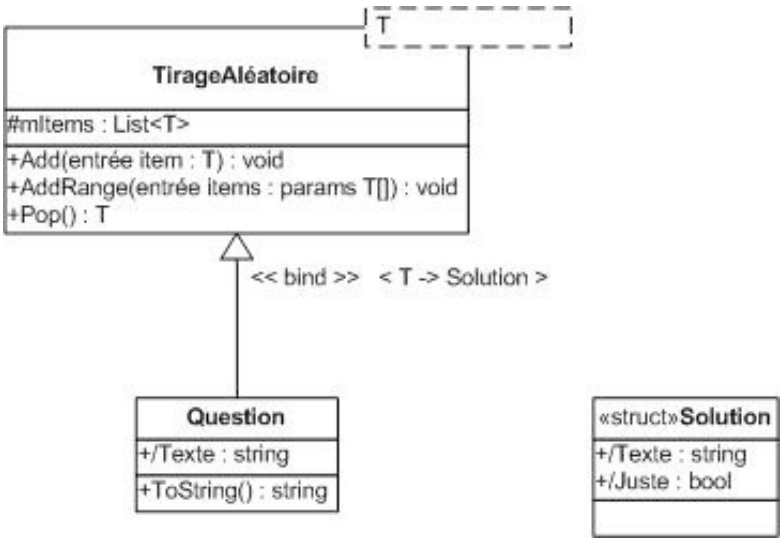
	Tricheur
Triche	passe Min à 15 passe Max à 15

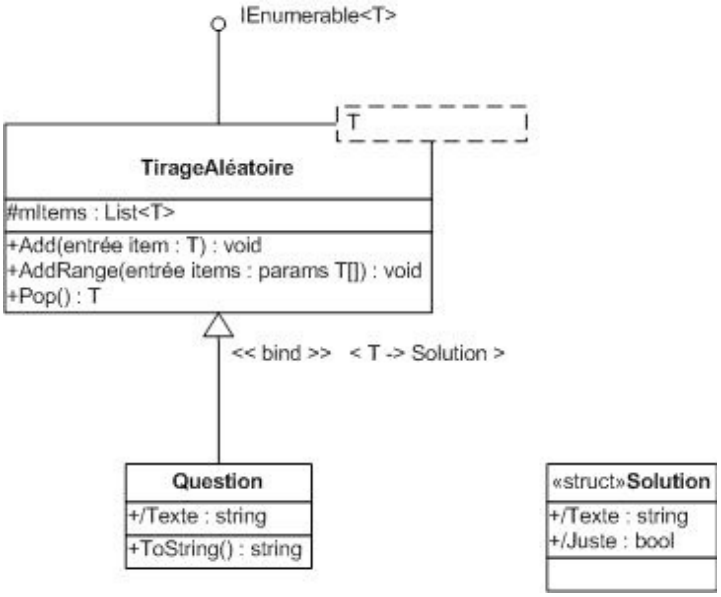
Relancez le programme et vérifiez que `Tricheur` s'en sort bien…

2. Ajoutez une interface `ICafé` possédant une méthode `PrépareLeCafé` (le retour de cette méthode sera un message de gentillesse accompagnant le café).

Modifiez la classe `Professeur` en lui ajoutant une méthode `DemandeUnCafé` prenant un paramètre de type `ICafé`. Cette méthode (non virtuelle) rendra une chaîne de caractères sur deux lignes contenant sur la première ligne «*Prénom Nom* `demande un café`» et sur la deuxième ligne, le retour de la méthode `PrépareLeCafé` appelée sur le paramètre de type `ICafé` passé en paramètre.

Créez une nouvelle sous-classe `Fayot` de `Bon` implémentant l'interface `ICafé` et réalisant les modifications suivantes :




IEnumerable<T>
T
TirageAléatoire
#mlItems : List<T>
+Add(entrée item : T) : void
+AddRange(entrée items : params T[]) : void
+Pop() : T
<< bind >> < T -> Solution >
Question
+/Texte : string
+ToString() : string
«struct»Solution
+/Texte : string
+/Juste : bool

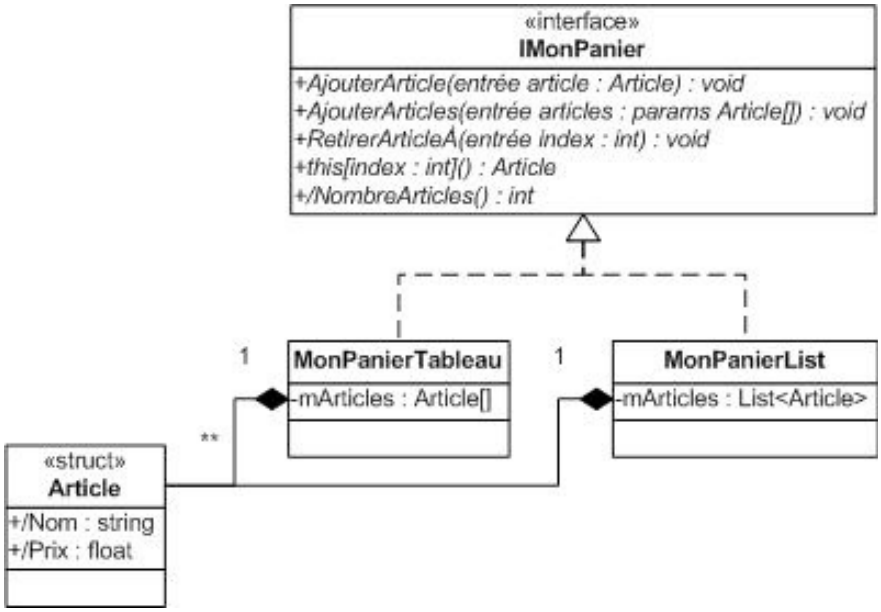
	Fayot
Travaille	comme Bon
TravaillesTu	«Bien sûr !»
PrépareLeCafé	«Voici un bon café ! Et n'oubliez pas que c'est de la part de *Prénom Nom*»

Modifiez la classe `Gentil` pour qu'elle implémente `ICafé`. Le retour de `PrépareLeCafé` sera : «`Voici un café de la part de ton collègue` *`Prénom Nom`*».

Modifiez `B20.Main` pour que `Harvey Dent` demande un café à `Schtroumpf Fayot`, puis demande un café à `Roger Rabbit`.

implémentation multiple d'interfaces

3. Créez une nouvelle sous-classe `Fourbe` de `Elève`, implémentant



`ITricheur` et `ICafé`, en utilisant les modifications suivantes :

	Fourbe
Travaille	augmente `Min` de 1 et `Max` de 1
TravaillesTu	«Avec plaisir !»
PrépareLeCafé	Augmente `Max` de 2 et rend : «Voici un bon café ! Et n'oubliez pas que c'est de la part de *Prénom Nom*»
Triche	double `Min` et double `Max`

Modifiez `B20.Main` pour que `Schtroumpf Fourbe` triche et que `Roger Rabbit` lui demande un café.

EXERCICE 02_05

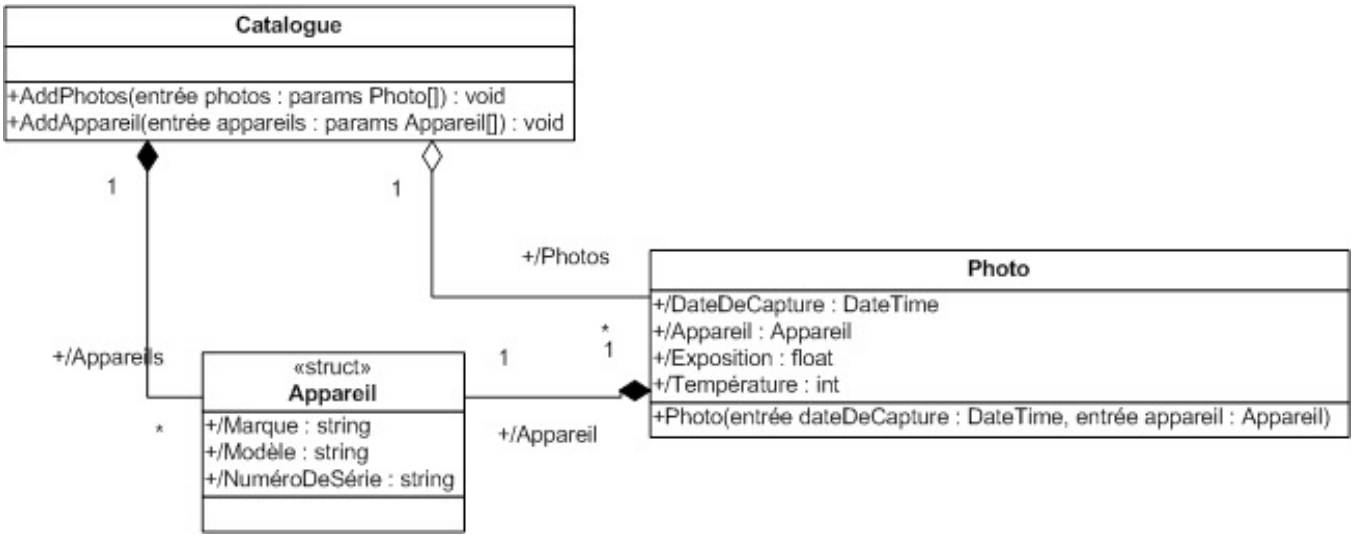
héritage
polymorphisme
design pattern
Composite

Utilisez le *design pattern Composite* pour implémenter une hiérarchie basique au sein d'une entreprise comme proposée dans le diagramme de classes partiel ci-dessous : un `Employé` peut être `ChefDeProjet` ou `Stagiaire`.

Un `Employé` possède un `Nom` et une méthode abstraite `Travaille`.

Un `Stagiaire`, lorsqu'il travaille, réalise les trois opérations suivantes :

- affiche dans une interface de type Console : «`Je m'appelle` *`Nom`* `et je commence à travailler`»
- attend 2 secondes (`System.Threading.Thread.Sleep(2000);`)



- affiche dans une interface de type Console : «`Je m'appelle Nom et j'ai fini de travailler`».

Un `ChefDeProjet` possède des esclaves de type `Employé` (qui lui sont attribués à la construction de l'objet).

Un `ChefDeProjet`, lorsqu'il travaille, réalise les trois opérations suivantes :

- affiche dans une interface de type Console : «`Je m'appelle Nom et je délègue le travail à mes esclaves`»
- fait travailler ses esclaves
- affiche dans une interface de type Console : «`Je m'appelle Nom et j'ai (enfin, mes esclaves...) terminé mon travail`».

Pour tester vos classes, vous pouvez :

- créer dans la classe `Program`, un tableau de 8 `Employés` :
  - 4 `Stagiaires` : *`Kyle`*, *`Kévin`*, *`Ken`* et *`Karl`*,
  - 4 `ChefDeProjets` : *`Jim`* (chef de *`Kyle`* et *`Kévin`*), *`John`* (chef de *`Kévin`* et *`Jim`*), *`James`* (chef de *`Kyle`*, *`Karl`* et *`John`*), *`Johnson`* (chef de projet de *`James`*, *`Ken`* et *`John`*).
- demander au programme de choisir aléatoirement un des 8 employés et faites-le travailler.

Amélioration : pour plus de clarté dans l'affichage, modifiez la méthode `Travaille()` en `Travaille(string tabulation)`. Cette tabulation sera utilisée devant chaque affichage Console : à chaque fois qu'un `ChefDeProjet` délègue à ses esclaves, il augmente la tabulation de 2 espaces, ce qui permet un affichage en «escalier», comme par exemple si on demande à *John* de travailler :

```
Je m'appelle John et je délègue le travail à mes esclaves.
  Je m'appelle Kévin et je commence à travailler.
  Je m'appelle Kévin et j'ai fini de travailler.
  Je m'appelle Jim et je délègue le travail à mes esclaves.
    Je m'appelle Kyle et je commence à travailler.
    Je m'appelle Kyle et j'ai fini de travailler.
    Je m'appelle Kévin et je commence à travailler.
    Je m'appelle Kévin et j'ai fini de travailler.
  Je m'appelle Jim et j'ai (enfin, mes esclaves...) terminé mon travail.
Je m'appelle John et j'ai (enfin, mes esclaves...) terminé mon travail.
```

EXERCICE 02_06

héritage
polymorphisme
design pattern
Strategy

Utilisez le *design pattern Strategy* pour implémenter une panier d'achats très simplifié comme proposée dans le diagramme de classes partiel ci-dessous.

Un `Article` possède un `Nom` et un `Prix`.

Un `MonPanier` peut être rempli à l'aide de la méthode `RemplirPanier` (puisqu'à ce moment du cours, nous n'avons pas encore vu les collections, vous pouvez utiliser un tableau d'`Article`s et en conséquence, `RemplirPanier` écrasera le contenu de ce tableau à chaque appel, pour le remplacer par les `Article`s passés en paramètres).

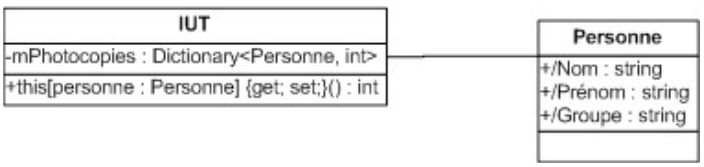
Un `MonPanier` possède une stratégie de paiement (de type `IPaiementStrategy`) qui peut être modifiée à l'aide de la méthode `ChoisirStratégieDePaiement`.

La propriété calculée `PrixTotal` calcul le coût de `MonPanier` en utilisant la stratégie de paiement choisie.
Implémentez trois stratégies de paiement :
un `PaiementNormal` qui ne fait que calculer la somme des prix des `Articles` de `MonPanier`,
un `ParPalliers` qui applique une réduction sur le total de :
10% si `MonPanier` possède plus de 2 articles,
20% si `MonPanier` possède plus de 3 articles,
25% si `MonPanier` possède plus de 4 articles,
30% si `MonPanier` possède plus de 5 articles,
un `TVAOfferte` qui rembourse la TVA.
Testez vos classes en créant différents `MonPaniers` et en utilisant différentes stratégies de paiement.

EXERCICE 02_07

Generics
List<T>

Créez une classe générique `TirageAléatoire<T>`, où `T` sera un type valeur, comme dans le diagramme ci-dessous.
Cette classe contiendra un membre de type collection de `T` (vous pourrez utiliser une `List` générique), une méthode permettant d'ajouter un élément, une autre méthode permettant d'ajouter plusieurs éléments, et enfin une méthode `Pop` qui permettra de rendre et retirer un élément de la liste, choisi aléatoirement.



Testez votre classe à l'aide des deux exemples suivants.

1. Réalisez le tirage du loto, à l'aide de `TirageAléatoire<uint>` que vous remplirez avec tous les entiers entre `1` et `50`. Tirez ensuite 10 numéros plus un numéro complémentaire et affichez-les à l'écran.
2. Écrivez une sous-classe `Question` de `TirageAléatoire`, où `T` sera de type `Solution`. `Solution` est une structure possédant un `Texte` et un booléen indiquant si cette `Solution` à la `Question` est `Juste`. `Question` possède un énoncé de la question (`Texte`) et, via `TirageAléatoire`, une collection de `Solutions`. Ajoutez un

`ToString` à `Question` qui pose la question

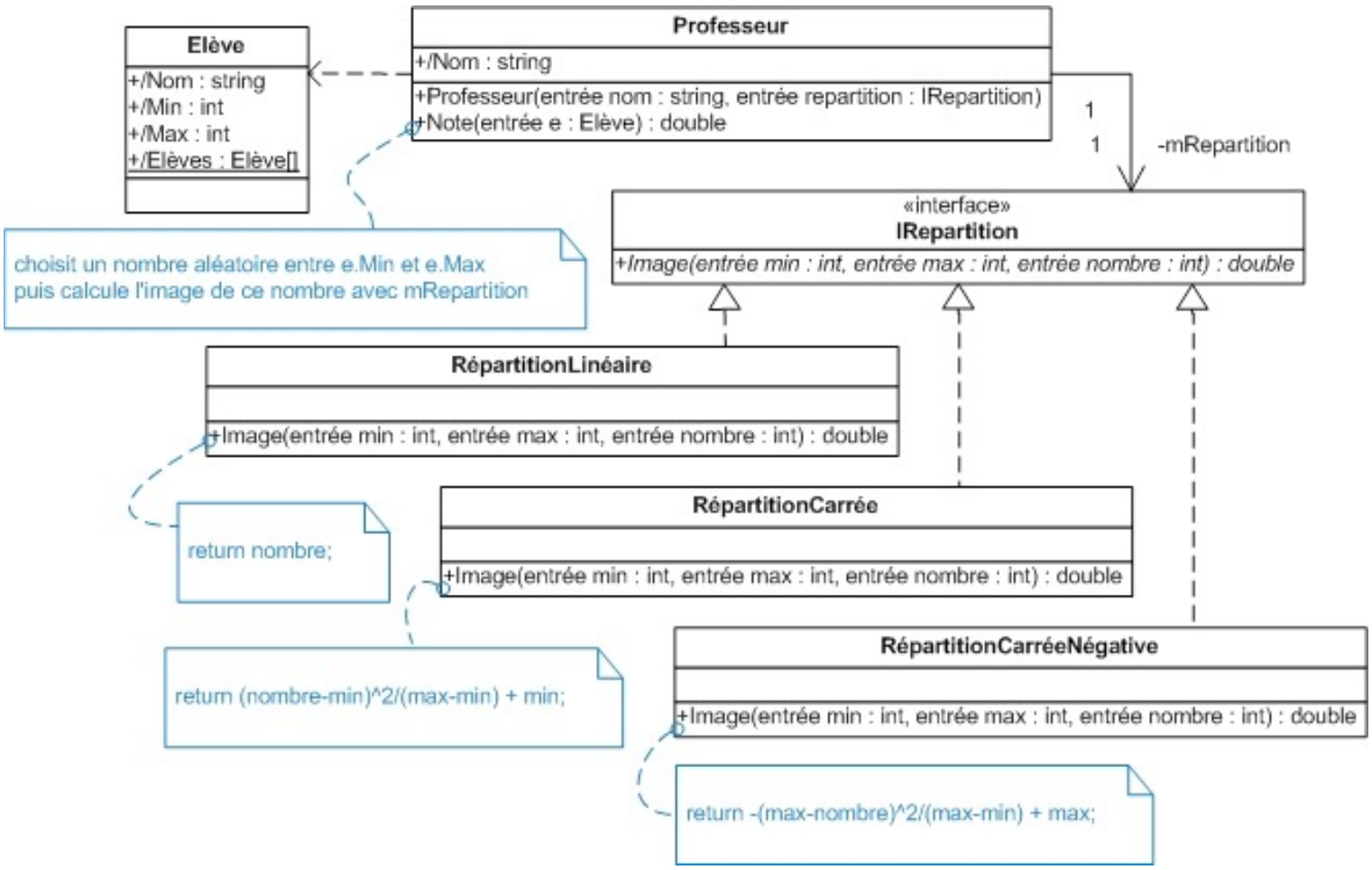
(`Texte` de la `Question` + `Texte` de chaque `Solution`). Répondez au hasard à l'aide des méthodes de `TirageAléatoire`. Vérifiez si la `Solution` choisie est la bonne.

EXERCICE 02_08

Generics

IEnumerable<T>

Reprenez l'exemple précédent. On souhaite modifier la classe `TirageAléatoire` pour permettre de faire l'énumération des éléments



List<T>

`T` de cette collection de manière aléatoire (sans retirer les éléments pendant l'énumération).

Modifiez `TirageAléatoire` pour qu'elle implémente `IEnumerable<T>`.

Testez vos modifications à travers la classe `Question` pour que la `Question` posée propose les `Solutions` de manière différente à chaque énumération. Notez que dès lors que vous avez implémenté `IEnumerable<T>`, vous pouvez utiliser `foreach` sur `Question` et `TirageAléatoire<T>` en général.

EXERCICE 02_09

List<T>

Le but de cet exercice est de comparer le tableau à la classe `List` générique. Pour cela, nous utiliserons le diagramme de classes partiel ci-dessous.

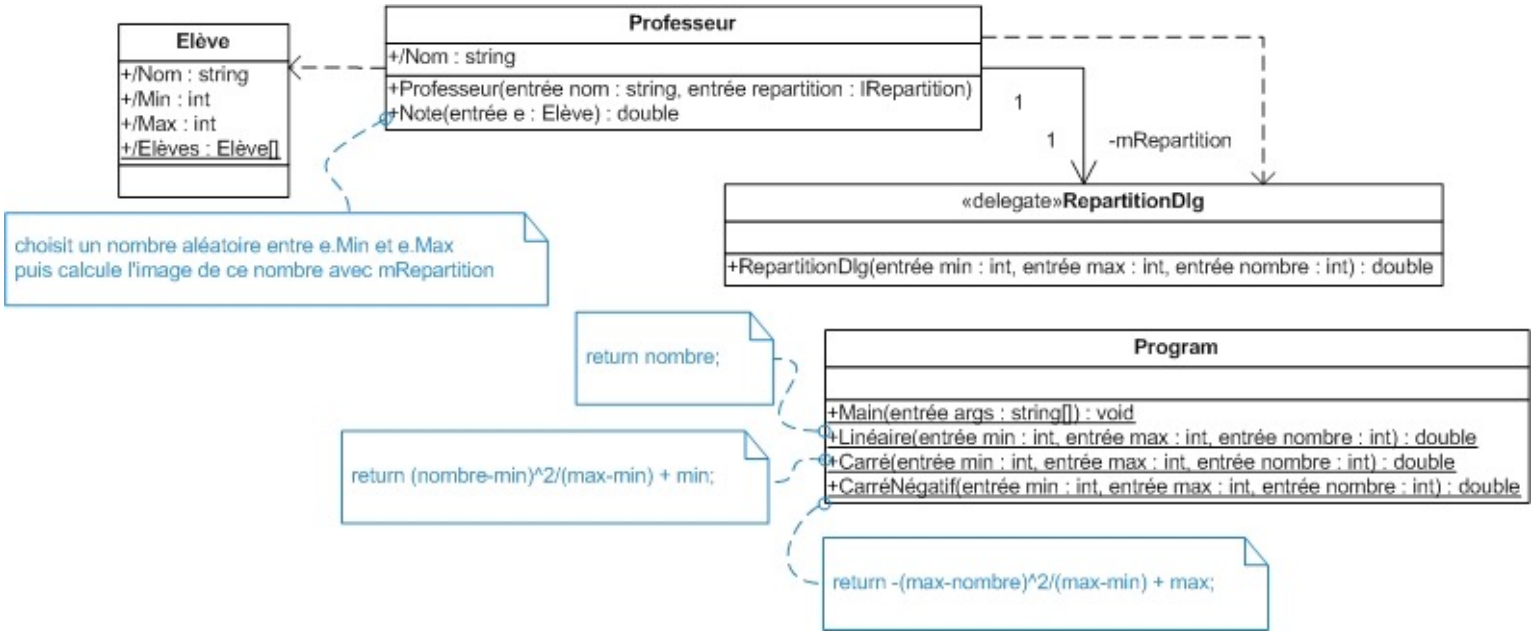
Créez une interface `IMonPanier` possédant les méthodes suivantes :

- `AjouterArticle` qui permet d'ajouter un `Article`,
- `AjouterArticles` qui permet d'ajouter plusieurs `Articles`,
- `RetirerArticleÀ` qui permet de retirer un `Article` à l'indice indiqué,
- un indexeur permettant d'atteindre un `Article` du panier,
- une propriété calculée qui permet d'obtenir le nombre d'`Articles`.

Implémentez cette interface de deux manières différentes :

1. en utilisant un tableau d'`Articles`. Vous serez obligé dans les trois premières méthodes de redéfinir un nouveau tableau avec la nouvelle taille à chaque modification des `Articles` du panier,
2. en utilisant une `List` générique.

Comparez les deux méthodes.



Cours 3

EXERCICE 03_01

Collections génériques

Comme préambule aux exercices suivants, réalisez dans un assemblage de type bibliothèque de classes, le diagramme de classes incomplet suivant. Vous pourrez utiliser la classe `List<T>` pour les collections, sans l'encapsuler pour le moment.

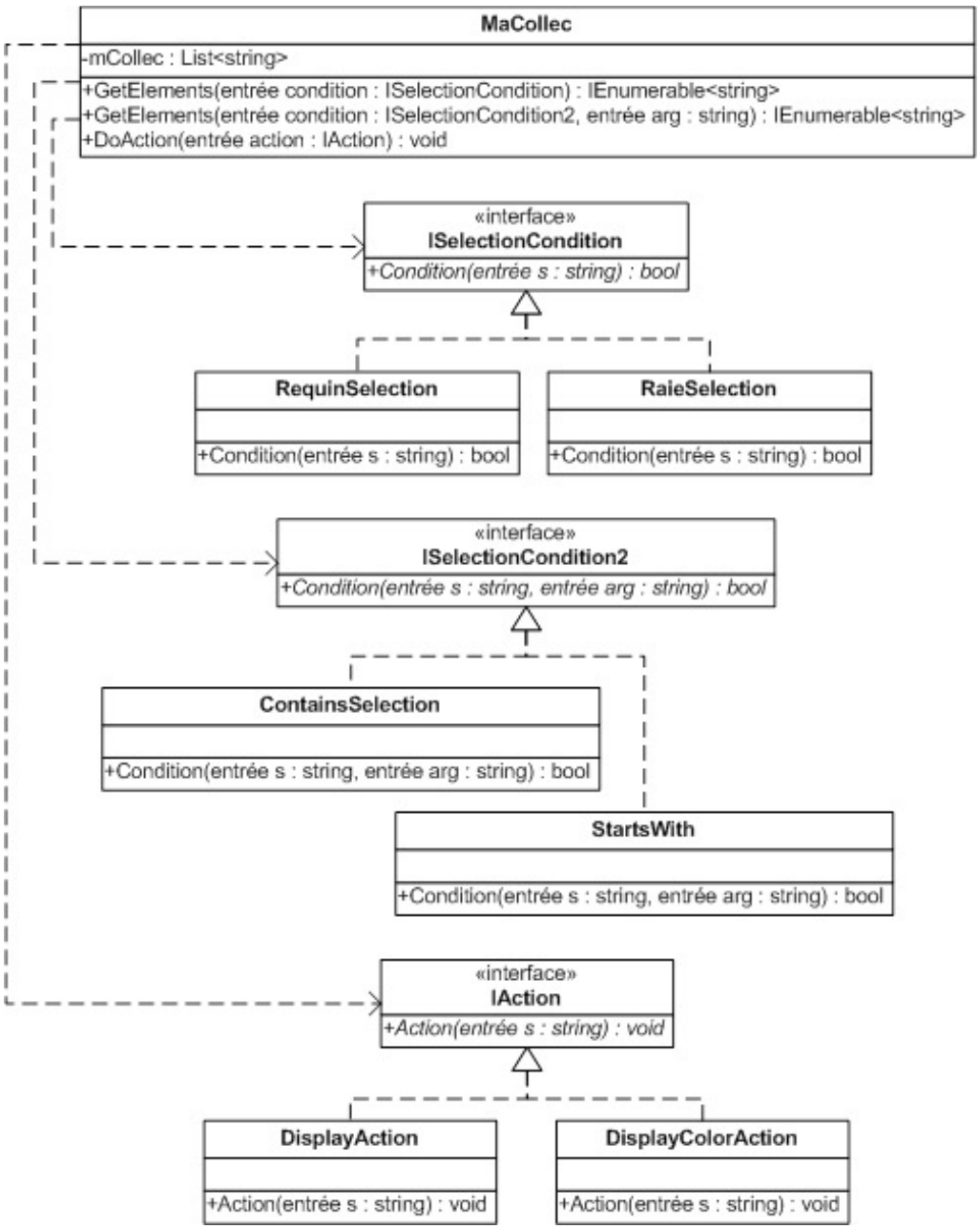
Les propriétés `Exposition` et `Température` de `Photo` ont des setters publiques. Toutefois, le constructeur de `Photo` leur donne une valeur aléatoire entre 0 et 1 pour l'exposition et entre 0 et 32000 pour la température.

Écrivez ensuite une application Console utilisant `Catalogue` et lui ajoutant des appareils et des photos, comme par exemple :

```
new Appareil("EOS 1100D", "Canon", "SN1234"),
new Appareil("EOS 650D", "Canon", "SN2345"),
new Appareil("EOS 650D", "Canon", "SN3456"),
new Appareil("EOS 60D", "Canon", "SN4567"),
new Appareil("EOS 7D", "Canon", "SN5678"),
new Appareil("EOS 7D", "Canon", "SN6789"),
new Appareil("EOS 5D Mark II", "Canon", "SN7890"),
new Appareil("EOS 5D Mark III", "Canon", "SN8901"),
```

```
new              new Photo(catalogue.Appareils[1], new DateTime(2012, 03, 01)),
Appareil("EOS
1D X",
"Canon",
"SN9012")


new
Photo(catalogu
e.Appareils[2]
, new
DateTime(2012,
01, 01)),
new
Photo(catalogu
e.Appareils[2]
, new
DateTime(2012,
02, 01)),
```



```
new Photo(catalogue.Appareils[1], new DateTime(2012, 04, 01)),
new Photo(catalogue.Appareils[2], new DateTime(2012, 05, 01)),
new Photo(catalogue.Appareils[5], new DateTime(2012, 06, 01)),
new Photo(catalogue.Appareils[5], new DateTime(2012, 07, 01)),
new Photo(catalogue.Appareils[5], new DateTime(2012, 08, 01)),
new Photo(catalogue.Appareils[2], new DateTime(2012, 09, 01)),
new Photo(catalogue.Appareils[5], new DateTime(2012, 09, 10)),
new Photo(catalogue.Appareils[1], new DateTime(2012, 09, 20))
```

Toujours dans l'application Console, affichez le contenu des deux collections à l'aide de `foreach`.

Constatez que vous pouvez utiliser deux méthodes pour ajouter des photos ou des appareils :

- soit en utilisant les méthodes `AddAppareils` et `AddPhotos`,
- soit en utilisant la méthode `Add` (ou `AddRange`) de `List<T>` sur les propriétés `Appareils` et `PhotosOriginales` de catalogue.

Ceci est dû au fait que les collections d'appareils et de photos ne sont pas encapsulées. Constatez également que vous pouvez supprimer, nettoyer les listes.

EXERCICE 03_02

encapsulation des collections

Modifiez l'assemblage précédent pour que les collections d'appareils et de photos de la classe `Catalogue` soient parfaitement encapsulées, i.e. qu'on ne puisse pas enlever d'appareils ou de photos à catalogue, qu'on ne puisse pas nettoyer les collections et qu'on ne puisse pas modifier un appareil ou une photo. Catalogue doit néanmoins toujours permettre d'ajouter un appareil ou une photo, mais seulement à travers des méthodes `Add`. Modifiez ces méthodes (corps, signature...) si nécessaire pour garantir l'encapsulation des collections. Ne modifiez pas l'accessibilité publique des setters d'`Exposition` et de `Température` dans la classe `Photo`.

Modifiez l'application Console pour qu'elles permettent d'ajouter des photos, des appareils et d'afficher l'ensemble des deux collections.

EXERCICE 03_03

Protocoles d'égalité sur les valeurs et sur les références IEquatable<T>

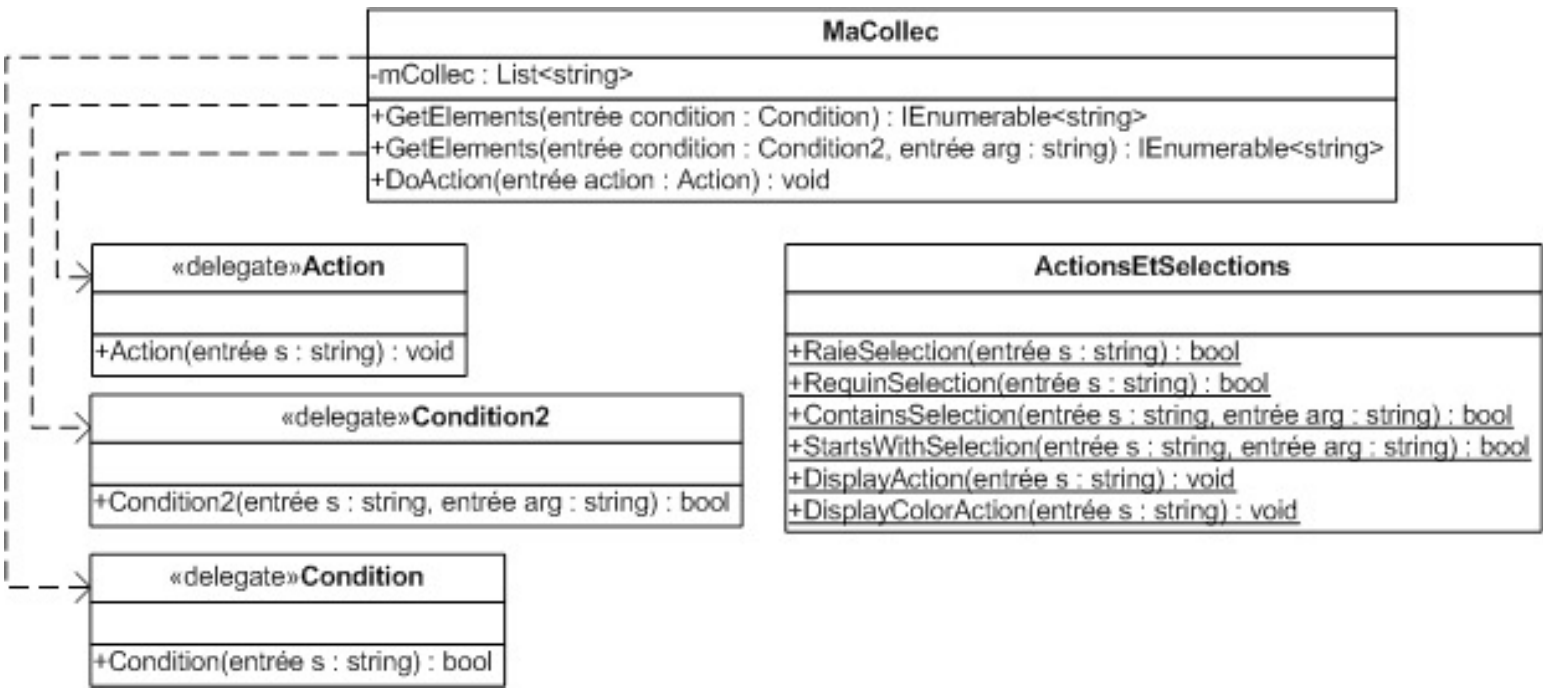
À partir du résultat de l'exercice 03_02, implémentez les protocoles d'égalité sur `Appareil` et sur `Photo`. Deux `Appareils` seront égaux si et seulement si ils sont de la même `Marque` et ont le même numéro de série. Deux `Photos` seront égales si et seulement si elles ont été prises avec le même `Appareil` et à la même date de capture.

Modifiez ensuite les méthodes `AddPhoto` et `AddAppareils` de la classe `Catalogue` pour qu'elles n'ajoutent les `Photos` et les `Appareils`, que s'ils n'existent pas déjà dans les listes. Bien évidemment, profitez des protocoles d'égalité pour cela. Vous pourrez par exemple utiliser la méthode `Contains` sur les `List` qui se base sur le protocole d'égalité, ou la méthode `Distinct` (issue de LINQ, que nous verrons plus en détails en 5ème semaine), qui se base également sur le protocole d'égalité pour supprimer les doublons.

EXERCICE 03_04

égalités IEqualityComparer <T>

Ajoutez aux résultats précédents, un nouveau `EqualityComparer` pour la structure `Appareil`. Celui-ci considérera deux `Appareils` égaux si et



seulement si ils ont la même `Marque` et le même `Modèle`. Utilisez-le dans

une application Console.

Ne remplacez pas le protocole d'égalité précédent : ajoutez une nouvelle classe implémentant `IEquatableComparer<T>`.

EXERCICE 03_05

comparaisons
IComparable<T>

À partir du résultat précédent, implémentez le protocole de comparaison sur la classe `Photo` pour permettre de trier les `Photos` par ordre chronologique de la `DateDeCapture`.

Testez ce protocole de comparaison en triant dans une application Console les photos d'une instance de `Catalogue`.

EXERCICE 03_06

comparaisons
IComparer

On veut simuler le cas où le client souhaiterait rajouter une nouvelle manière de comparer des photos, sans avoir accès au code source de `Photo`. Écrivez une application Console, utilisant la bibliothèque issue de l'exercice précédent, et créez une classe implémentant `IComparer` permettant de comparer deux `Photos` par ordre croissant d'`Exposition` (puis par ordre croissant de `Température` si les `Expositions` sont égales).

Testez-la sur la collection de `Photos` d'une instance de `Catalogue`.

EXERCICE 03_07

Dictionary<TKey, TValue>

Reprenez la classe `ClassementCurling` de l'exercice 01_14. Remplacez les deux listes de même taille par un `Dictionary<string, int>`. Mettez à jour `ClassementCurling` :

- qu'en est-il de l'indexeur `public string this[int index]` ?
- utilisez les avantages du dictionnaire pour réécrire l'indexeur `public int this[string pays]`

Testez la nouvelle classe dans une application Console.

EXERCICE 03_08

Dictionary<TKey, TValue>

Créez une classe `IUT` contenant un dictionnaire permettant de compter le nombre de photocopies réalisées par les instances de la classe `Personne`. L'indexeur de `IUT` permettra d'accéder en lecture et écriture au nombre de photocopies d'une `Personne`.

Testez vos classes dans une application Console. Vous pourrez notamment tester les lignes suivantes :

```
IUT iut = new IUT();
iut[new Personne(«Dwight», «Schrute», «GI2»)] = 17;
iut[new Personne(«Dwight», «Schrute», «GI2»)] += 18;
Console.WriteLine(iut[new Personne(«Dwight», «Schrute», «GI2»)]);
```

Le bon résultat est 35. Que faut-il rajouter à la classe `Personne` pour pouvoir l'utiliser correctement en clé du dictionnaire et obtenir ce résultat ?

Cours 4

EXERCICE 04_01

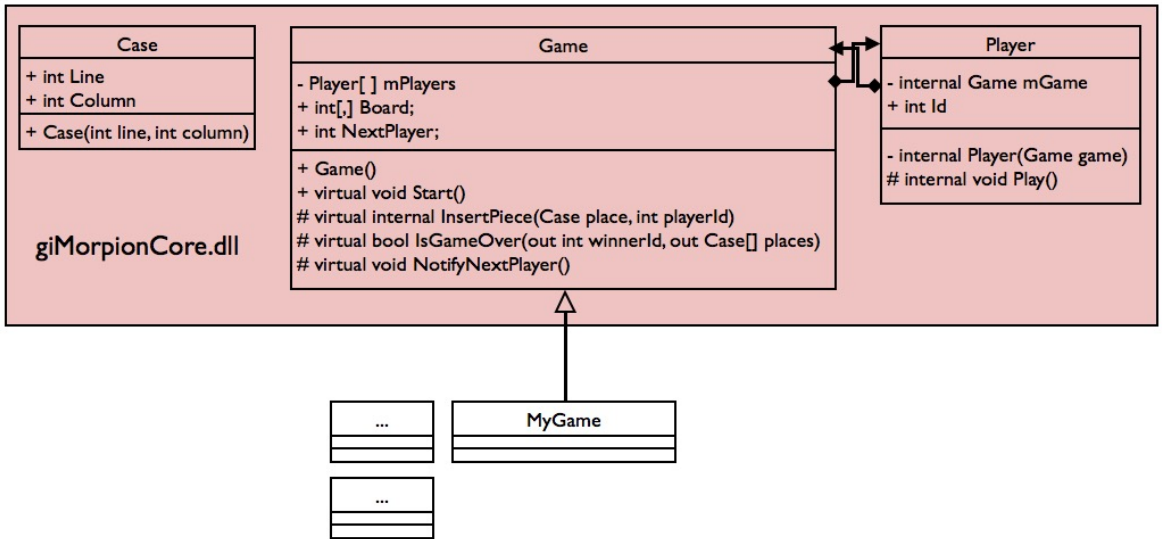
Interfaces
Strategy

Comme préambule aux exercices suivants, réalisez un programme en respectant le diagramme de classes suivant.

Un `Elève` possède un `Nom`, une note `Min`, une note `Max`. On a également un tableau d'`Elèves` prédéfinis avec au moins 3 `Elèves` dont les notes `Min` et `Max` sont différentes.

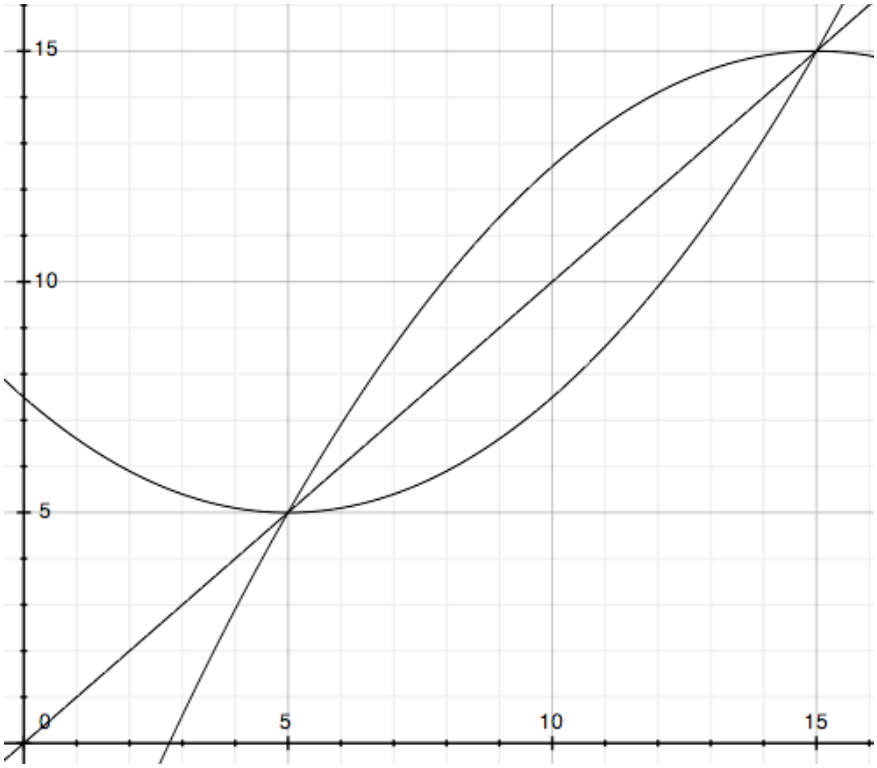
Une classe implémentant `IRépartition` devra comporter une méthode `Image`. Cette méthode prendra en paramètres un entier `min`, un entier `max` et un entier `nombre` et devra rendre l'image de ce nombre selon une équation associée à cette classe. Cette équation pourra (ou non) se baser sur les paramètres `min` et `max`. Ainsi :

- la classe `RépartitionLinéaire` rend l'image de x sur la droite y=x, i.e. rend le nombre directement, sans modifications,
- la classe `RépartitionCarrée` rend l'image de x sur la parabole $y=\frac{(x\text{-min})^2}{\text{max-min}}+\text{min}$, i.e. un nombre revu à la baisse,



- la classe `RépartitionCarréeNégative` rend l'image de x sur la parabole $y=-\frac{(\text{max-}x)^2}{\text{max-min}}+\text{max}$, i.e. un nombre revu à la hausse.

Ci-dessous, les trois méthodes avec `min=5` et `max =15`.



Un `Professeur` possède un `Nom` et une méthode de `IRépartition`. La méthode `Note` choisit un nombre aléatoirement entre `Min` et `Max` inclus de l'`Elève` passé en paramètre, puis utilise la classe concrète de `IRépartition` pour modifier cette note. La note obtenue est minorée à 0 et majorée à 20 puis rendue.

Créez trois `Professeurs` (un pour chaque méthode de `Répartition`) et notez les `Elèves` du tableau statique d'`Elèves` avec chacun d'entre eux et affichez les résultats.

EXERCICE 04_02

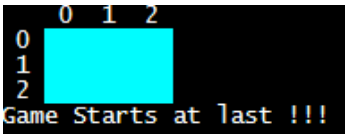
delegate
Strategy

Réécrire le programme précédent en utilisant un délégué à la place de l'interface comme le propose le diagramme de classes suivant.

EXERCICE 04_03

delegate
Strategy
Func<,,,>

.NET contient déjà un certain nombre de types délégués dans l'espace de nom System. Parmi eux, on peut distinguer notamment la famille des Func :




```
delegate TResult Func<TResult>();
delegate TResult Func<T, TResult>(T arg);
delegate TResult Func<T1, T2, TResult>(T1 arg1, T2 arg2);
delegate TResult Func<T1, T2, T3, TResult>(T1 arg1,
                                           T2 arg2, T3 arg3);
delegate TResult Func<T1, T2, T3, T4, TResult>(T1 arg1,
                                T2 arg2, T3 arg3, T4 arg4);
```

Ces types délégués décrivent des méthodes qui rendent un `TResult` et prennent en paramètre différents arguments de type `T1`, `T2`, `T3`, `T4`...

Modifiez l'exercice précédent en utilisant un type délégué `Func` plutôt que de déclarer un délégué `RepartitionDlg`.

Réécrire le programme précédent en utilisant un délégué à la place de l'interface comme le propose le diagramme de classes suivant.

EXERCICE 04_04

delegate
Strategy

Reprenez l'exercice 02_06 avec le pattern Strategy sur `MonPanier`, en utilisant un delegate à la place de l'interface et des classes concrètes.

EXERCICE 04_05

delegate
Strategy
Func

Reprenez l'exercice précédent en utilisant un delegate prédéfini `Func`.

EXERCICE 04_06

interface
collections

Le but de cet exercice est de réaliser des filtrages et des actions sur les éléments d'une collection, sans utiliser LINQ ni les délégués pour le moment. Pour cela, aidez-vous du diagramme de classes ci-dessus et réalisez les opérations suivantes :

- utilisez la classe `MaCollec` partielle qui vous est fournie et qui encapsule une collection de chaîne de caractères,
- conditions simples :
  - créez une interface `ISelectionConditio n` qui contiendra une méthode `Condition` rendant `true` ou `false` en fonction de la chaîne de caractères passée en paramètres,
  - écrivez deux classes concrètes implémentant cette interface, par exemple : `RaieSelection` recherche si la chaîne de caractères contient la chaîne «raie» (quelle que soit la casse) ; `RequinSelection` recherche si la chaîne de caractères contient la chaîne «requin»(quelle que soit la casse)
  - ajoutez une méthode à `MaCollec` prenant en paramètre une condition (de type `ISelectionCondition`) et rendant une collection de string vérifiant cette condition
  - testez-la avec les deux classes concrètes
- conditions plus élaborées :
  - créez une interface `ISelectionCondition2` qui contiendra une méthode `Condition` rendant `true` ou `false` en fonction de la chaîne de caractères passée en paramètres ainsi que d'une chaîne de caractères supplémentaires passées en argument,
  - écrivez deux classes concrètes implémentant cette interface, par exemple : `StartsWith` recherche si la chaîne de caractères commence par la deuxième et `ContainsWith` recherche si la chaîne de caractères contient la deuxième (n'utilisez pas LINQ pour cet exercice),
  - ajoutez une méthode à `MaCollec` prenant en paramètre une condition (de type `ISelectionCondition2`), un argument supplémentaire, et rendant une collection de string vérifiant cette condition
  - testez-la avec les deux classes concrètes

- actions :
  - créez une interface `IAction` qui contiendra une méthode `Action` ne rendant rien et réalisant quelque chose en fonction de la chaîne de caractères passée en paramètres,
  - écrivez deux classes concrètes implémentant cette interface, par exemple : `DisplayAction` qui affiche la chaîne de caractères passée et `DisplayColorAction` qui affiche en jaune la chaîne de caractères si elle fait moins de 5 caractères et en rouge si elle fait plus de 10 caractères
  - ajoutez une méthode à `MaCollec` prenant en paramètre une action (de type `IAction`) et ne rendant rien, qui effectuera l'action sur chaque élément
  - testez-la avec les deux classes concrètes

EXERCICE 04_07

delegate

Reprenez l'exercice 04_06, en utilisant trois delegates à la place des interfaces et des classes concrètes. Vous pourrez suivre par exemple le diagramme de classes suivants, qui définit trois types délégués internes à `MaCollec`, puis une classe statique qui contient 6 méthodes statiques (2 pour chaque type délégué). Testez le tout dans une application Console.

EXERCICE 04_08

delegate
Predicate
Func
Action

.NET contient déjà un certain nombre de types délégués dans l'espace de nom System. Parmi eux, on peut distinguer notamment la famille des `Func` (comme nous l'avons déjà vu dans l'énoncé de l'exercice 04_03) ou bien la famille des `Action` et celle des `Predicate` :

```
delegate void Action<T>(T obj);
delegate bool Predicate<T>(T obj);
delegate TResult Func<T1, T2, TResult>(T1 arg1, T2 arg2);
```

Le délégué `Action` prend un objet générique en paramètre et en fait quelque chose (sans rien rendre).

Le délégué `Predicate` prend un objet générique en paramètre et rend un `true` ou `false`.

Le délégué `Func` prend 1 ou plusieurs paramètres de différents types et rend un quelque chose d'un autre type. Modifiez l'exercice précédent en utilisant les types délégués prédéfinis `Func`, `Predicate` et `Action`, plutôt que d'utiliser les déclarations internes des types délégués `Condition`, `Condition2` et `Action` dans `MaCollec`.

Ex04_09 : examen du 2 novembre 2010

Sujet - Jeu du morpion

OBJECTIFS

L'objectif de cet examen est de réaliser une application Console qui permet de regarder deux joueurs automatiques jouer au Morpion. Pour vous aider, deux bibliothèques vous sont fournies :

• la première `giMorpionCore.dll` contient des classes et des méthodes gérant une grande partie du jeu (démarrage du jeu, insertion de pièces, indication au prochain joueur qu'il doit jouer, test de la fin du jeu...),

• la deuxième `giMorpionTools.dll` contient une classe et une méthode statiques permettant d'afficher la grille du Morpion dans une fenêtre Console.

Mais alors que reste-t-il alors à faire ?

Ces bibliothèques ne communiquent pas, et il n'y a pas d'application. Dans la classe `Game` :

• la méthode `NotifyNextPlayer` appelle la méthode `Play` du prochain joueur (qui joue ainsi son coup),

• la méthode `Start` appelle la méthode `NotifyNextPlayer` sur le premier joueur,

• la méthode `InsertPiece` se contente d'insérer ou non la pièce et d'indiquer qui sera le prochain joueur, mais la liaison avec le reste du jeu n'est pas réalisée,

• la méthode `IsGameOver` teste si le jeu est terminé mais n'est jamais appelée.

Dans la classe `Player`, l'appel de la méthode `Play` appelle la méthode `InsertPiece` de `Game` (notez que cette méthode `InsertPiece` est virtuelle !!)

Vous avez la charge de :

• réutiliser cette bibliothèque, sans avoir accès au code source,

• écrire la liaison entre toutes ces méthodes pour que le jeu puisse se dérouler,

• créer des événements pour pouvoir renseigner une application extérieure sur le déroulement du jeu,

• créer une application Console permettant de visualiser le jeu et son déroulement en s'abonnant à ces événements.

QUESTION 1 : BIBLIOTHÈQUE ET ÉVÉNEMENTS

Vous devez réaliser une bibliothèque de classes `giMorpionExam.dll` contenant vos nouvelles classes pour le déroulement du jeu du Morpion, c'est-à-dire une classe `MyGame` dérivant de la classe `Game`, dans laquelle vous devez :

- ajouter des événements (et donc peut-être des classes :-) permettant de renseigner le reste du monde que :
  - le jeu a commencé,
  - le jeu est terminé (en renseignant qui a gagné, et quelle est la ligne gagnante),
  - un joueur vient d'être renseigné qu'il doit jouer (et qui est ce joueur),
  - une pièce a été insérée (où et par qui).

  Pour cela, vous pourrez choisir de réécrire des méthodes virtuelles, ou d'écrire de nouvelles méthodes. Dans tous les cas, vous devrez profiter de l'héritage de `Game` et éviter de réécrire des choses existantes, et bien choisir à quel moment lancer l'événement.
- faire en sorte que le jeu se déroule automatiquement, c'est-à-dire en respectant l'ordre suivant :
  - un joueur est renseigné que c'est son tour de jouer,
  - il joue son coup,
  - une pièce est insérée
  - si cette insertion n'est pas bonne (case occupée ou en dehors des limites), le même joueur est renseigné qu'il doit jouer,
  - si l'insertion est bonne, la fin du jeu est testée, puis le joueur suivant est renseigné qu'il doit jouer si le jeu n'est pas terminé.

  Vous pourrez choisir par exemple de réécrire la méthode `InsertPiece`, et dans cette méthode, d'appeler les autres méthodes pour garantir l'ordre de ces appels. Il peut être d'autant plus judicieux de réécrire `InsertPiece` que `Player.Play` appelle cette méthode virtuelle.

Attention, beaucoup de choses sont déjà réalisées dans la bibliothèque `giMorpionCore.dll`. Étudiez la documentation `html` fournie pour en déduire le code à écrire et éviter de réécrire ce qui existe déjà. Apportez également une attention particulière au lancement des événements.

Un rappel utile : si vous réécrivez une méthode virtuelle `MethodA` dans une classe fille, vous pouvez appeler la méthode virtuelle `MethodA` correspondante de la classe mère à l'aide du mot clé `base` : `base.MethodA(...)`.

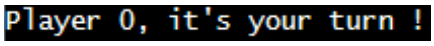
Conseil : respectez au maximum le pattern standard des événements.

QUESTION 2 : APPLICATION CONSOLE

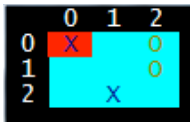
Vous devez réaliser une application Console qui s'abonnera aux 4 événements créés dans la question précédente, et permettra d'afficher le jeu de la manière suivante :

- quand le jeu débute : la grille vide et un message de début

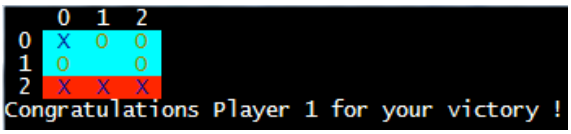
- quand un joueur est renseigné qu'il doit jouer : une phrase lui demandant de jouer



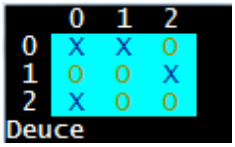
- quand un joueur a inséré une pièce : la grille avec la pièce insérée mise en évidence



- quand la partie est terminée avec un vainqueur : la grille avec la ligne gagnante mise en évidence, et un message au vainqueur,



- quand la partie est terminée sans vainqueur : la grille et un message pour dire qu'il n'y a pas de vainqueur



Conseil important : utilisez la bibliothèque `giMorpionTools.dll` pour l'affichage.

Ex04_10 : examen du 2 novembre 2011

Sujet - Tennis 2

OBJECTIFS

La fin de la saison de Tennis approche. Dwight Schrute souhaite suivre en direct pendant son travail les derniers matchs à l'aide d'un «Match Tracker» qui lui permet de connaître le score de chaque match en cours ou terminé, sans avoir à rafraîchir son application.

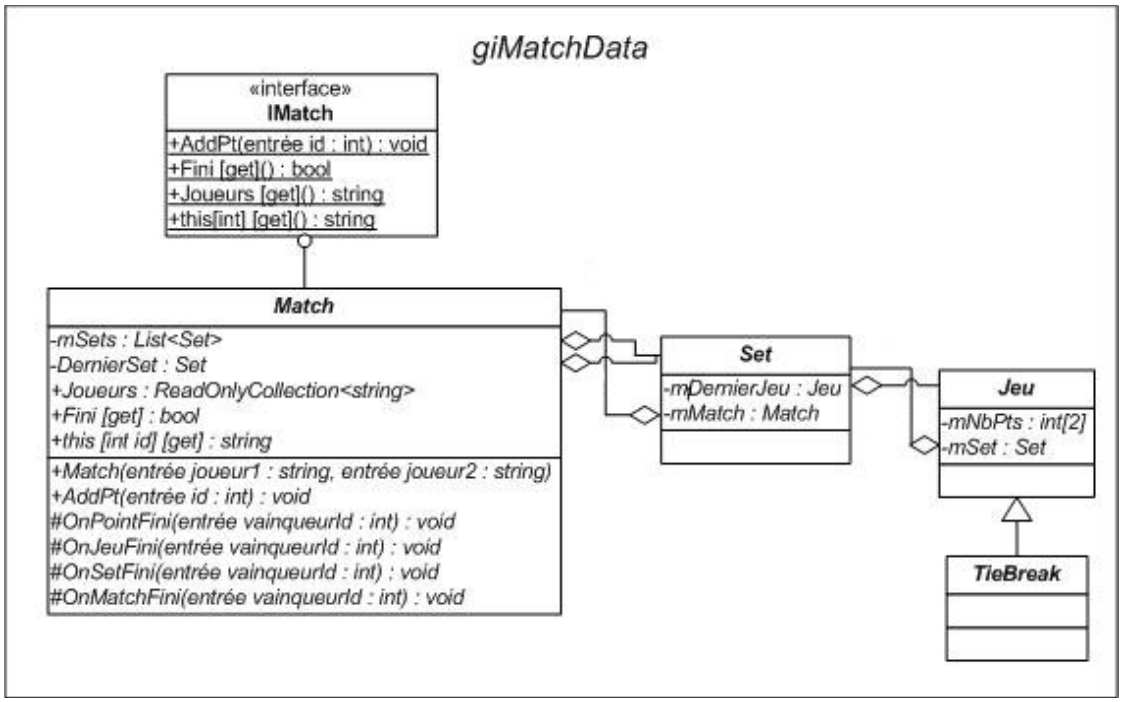
Il a déjà codé un certain nombre de classes. Malheureusement, il a beaucoup de travail à effectuer cette semaine et il n'a pas le temps de terminer son programme. C'est la raison pour laquelle il vous demande de terminer son application, afin de pouvoir suivre les résultats pendant sa semaine de travail chargée. Mais, les Schrute sont méfiants de père en fils, et, s'il accepte de vous livrer ses dlls, il refuse de vous donner son code source. Il est persuadé, que la qualité de son code vous permettrait de vous faire beaucoup d'argent sur son dos.

Votre mission, si vous l'acceptez (si vous ne l'acceptez pas, c'est o d'office...), est de terminer le Match Tracker de Dwight Schrute. Pour cela, vous pourrez suivre les conseils qui vous sont donnés dans cet énoncé.

On ne considérera ici que les matchs en deux sets gagnants. En pièce jointe à cet énoncé, vous trouverez la page wikipedia sur le Tennis avec notamment, un paragraphe Règles résumant les règles du jeu.

Dwight Schrute a implémenté trois bibliothèques :

• `giMatchData` : contient les classes permettant de stocker les résultats (ou le score actuel) d'un match, i.e. `Match`, `Set`, `Jeu` (et `TieBreak`). Le schéma suivant présente une vision simplifiée de cette bibliothèque. La méthode `AddPt` permet d'ajouter un point à une instance de `Match` en donnant l'identifiant du joueur (0 ou 1) ayant gagné le point. Les méthodes `OnPointFini`, `OnJeuFini`, `OnSetFini` et `OnMatchFini` sont protégées et virtuelles.



• `giTennisCaster` : contient une classe `TennisCaster` qui permet de simuler le flux d'informations sur chaque nouveau point dans un match. On pourra noter la présence d'un événement `PointGagné` qui est lancé à chaque fois qu'un point dans un match vient de se terminer. La méthode `Cast` permet de lancer l'écoute du flux d'informations. La propriété statique `COEFF_TEMPS` permet de modifier la vitesse de simulation du flux pour vous permettre de débugger plus rapidement (par exemple au début de l'application).

• `giMatchDisplayTool` : contient une classe statique avec une seule méthode statique permettant d'afficher un `Match` dans une fenêtre Console.

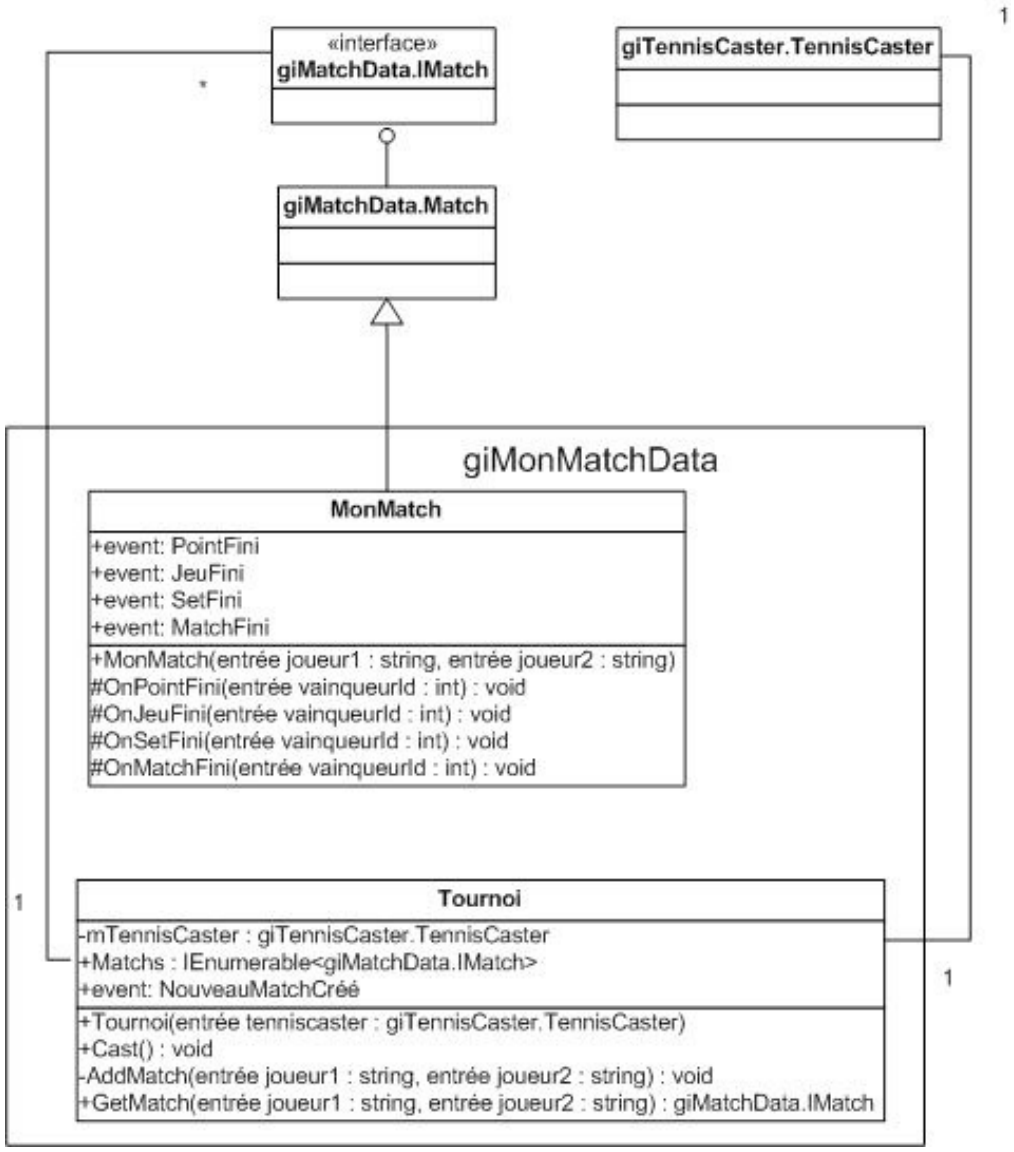
La documentation Doxygen vous est fournie !

VOUS devrez créer deux assemblages :

• `giMonMatchData` : de type bibliothèque de classes, qui devra contenir :

  • une classe `Tournoi` qui permettra de gérer les différents `Match` observés par le flux,

  • une classe `MonMatch` qui dérivera de `giMatchData.Match` et permettra l'utilisation d'événements.



• `MatchTracker` : de type Application Console, qui permettra l'affichage du déroulement des matchs en cours.

Conseil : faites des schémas au brouillon (diagramme de classes, diagramme de séquence...).

QUESTION 1 : TOURNOI ET GESTION DES MATCHS

Dans votre bibliothèque de classes `giMonMatchData`, créez une classe `Tournoi` qui permettra la gestion des `Matchs` (liste de matchs) à travers notamment :

- `GetMatch` qui permettra de récupérer le seul et unique match de la liste de matchs de ce `Tournoi` dans lequel s'affrontent les deux joueurs passés en paramètres. Si ce match n'existe pas encore dans la liste, il sera ajouté via la méthode `AddMatch`,
- `AddMatch` qui permettra d'ajouter un nouveau match dans lequel s'affrontent les deux joueurs passés en paramètres, à la liste des matchs de ce `Tournoi`.

Ces deux méthodes font que, pour récupérer un match du `Tournoi`, vous utiliserez toujours `GetMatch`, que ce match existe ou non dans le Tournoi.

- le constructeur qui permettra de s'abonner à l'événement `PointGagné` du `TennisCaster` afin d'ajouter un point au match porté par l'événement à chaque fois qu'un point est terminé.
- la méthode `Cast`, appellera la méthode `Cast` du membre de type `TennisCaster` associé à ce `Tournoi`, afin de lancer l'écoute du flux.

QUESTION 2 : ÉVÉNEMENTS SUR LE DÉROULEMENT D'UN MATCH

Ajoutez une classe `MonMatch` à votre bibliothèque de classes `giMonMatchData`, qui dérivera de `giMatchData.Match` en lui ajoutant 4 événements correspondant à la fin d'un point, la fin d'un jeu, la fin d'un set, la fin d'un match. Vous êtes responsable de la création des événements, et de leur lancement, mais pas encore de l'abonnement (question suivante). Modifiez `Tournoi` pour qu'il contienne une liste de matchs lançant ces événements.

QUESTION 3 : APPLICATION FINALE

Créez une application `MatchTracker` qui permettra d'afficher les résultats des matchs dans une fenêtre Console. Pour cela, vous devrez bien sûr vous abonner aux 4 événements de toutes les instances de `MonMatch` alors que celles-ci ne sont pas encore créées ! Pour cela, la solution proposée est la suivante :

- ajoutez à `Tournoi` un événement `NouveauMatchCréé`, qui sera lancé à chaque fois qu'un nouveau match est ajouté à la liste de matchs du `Tournoi` avec comme argument, le match créé ;
- abonnez l'application à cet événement ;
- à la réception de cet événement, l'application finale s'abonne aux 4 événements du match passé en argument.

Cours 5

EXERCICE 05_01

Méthodes anonymes

Modifiez l'exercice 04_03 en utilisant des méthodes anonymes pour remplacer les méthodes `Linéaire`, `Carré` et `CarréNégatif`.

EXERCICE 05_02

Méthodes anonymes

Modifiez l'exercice 04_08 en utilisant des méthodes anonymes et supprimer la classe statique `ActionsEtSélections`.

EXERCICE 05_03

expressions lambda

Reprenez l'exercice 05_01 en utilisant des expressions lambda.

EXERCICE 05_04

expressions lambda

Reprenez l'exercice 05_02 en utilisant des expressions lambda.

EXERCICE 05_05

méthodes d'extension

Écrivez les méthodes d'extension suivantes pour des entiers :

- `Abs` : une méthode qui transforme un entier en un sa valeur absolue,
- `Borne` : une méthode qui prend une borne inférieure et une borne supérieure en paramètres, et rend l'entier s'il est compris entre les deux, la borne inférieure si l'entier est inférieur à celle-ci, la borne supérieure si l'entier est supérieur à celle-ci,
- `ToLetter` : une méthode qui rend l'entier en lettres («zéro», «un», «deux»...) si l'entier est compris entre 0 et 9 (inclus) ou «je sais pas» s'il n'est pas compris entre ces bornes.

Testez vos méthodes dans une application Console. Vous pourrez notamment tester l'enchaînement de méthodes d'extension en réalisant par exemple :

```
Console.WriteLine((-13).Abs().Borne(0, 9).ToLetter());
```

EXERCICE 05_06

LINQ

Dans une solution, ajoutez le projet giRandoCore fourni avec l'énoncé. Créez une application Console dont giRandoCore sera une référence.

À l'aide d'instructions LINQ :

1) affichez tous les sommets de plus de 1000 mètres d'altitude
2) affichez le nom, le dénivelé et la distance des de toutes les randonnées dont le dénivelé est supérieur à 300m et la distance supérieure à 7km
3) affichez le nom et la distance de toutes les randonnées de la plus courte à la plus longue
4) affichez les pays et le nombre de randonnées des pays ayant le plus de randonnées
5) affichez le nom et la vitesse moyenne des randonnées par ordre de vitesse moyenne croissante
6) modifiez la classe `giRandoCore.Guide` en lui ajoutant une propriété permettant de ne rendre que les randonnées à pied
7) groupez les randonnées par pays, puis pour chaque groupe, affichez le nom du pays et ses randonnées (nom, dénivelé et distance)
8) transformez les randonnées en un dictionnaire dont la clé sera le pays contenant les randonnées et la valeur sera la moyenne des vitesses moyennes, puis affichez toutes les paires clé-valeur.

Cours 6

EXERCICE 06_01 *(PLUS AU PROGRAMME)*

fichier XML

Connectez vous à internet et analysez les pages du site Marmiton.org et en particulier les recettes suivantes :

http://www.marmiton.org/recettes/recette.cfm?num_recette=18588
http://www.marmiton.org/recettes/recette.cfm?num_recette=15025
http://www.marmiton.org/recettes/recette.cfm?num_recette=12023

L'objectif de cet exercice est d'écrire des documents XML correspondant à ces recettes.

Rappel : un document XML ne contient pas d'information de mise en forme mais de la sémantique. Il vous faudra donc étudier attentivement l'information contenu dans les recettes avant de vous lancer dans l'écriture des documents XML.

EXERCICE 06_02 *(PLUS AU PROGRAMME)*

fichier XML
DTD

Étudiez la DTD suivante http://marc.chevaldonne.free.fr/exercices/Ex06_02/recette.dtd et écrivez des documents XML pour les recettes précédentes en l'utilisant (en externe). Les documents devront bien entendu être valides. Pour cela, utilisez le validator gratuit à l'adresse suivante : http://www.w3schools.com/xml/xml_validator.asp, section "Validate your XML against a DTD" (uniquement avec Internet Explorer...).

EXERCICE 06_03 *(PLUS AU PROGRAMME)*

fichier XML
DTD

La DTD précédente présente quelques inconvénients. Écrivez une DTD interne inspirée de celle-ci afin de permettre[1] :

- pour un ingrédient, d'écrire son nom puis la quantité ou la quantité puis le nom
- pour un ingrédient, de ne pas être obligé d'écrire la quantité

[1] modifiez vos documents XML en conséquence pour vérifier que votre DTD est correcte

- pour une recette, de ne pas obligatoirement indiquer le temps de cuisson
- pour une recette, que les quantités soient indiquées pour 6 personnes par défaut
- pour une recette, d'indiquer l'auteur
- pour une recette, de choisir un type parmi : entrée, plat de résistance, dessert
- de rajouter des commentaires (soit de l'auteur, soit d'un lecteur)
- si ce n'est déjà fait, d'utiliser un élément "ustensile" dans la description des étapes
- de rajouter une URL sur une image du plat ainsi que des informations sur la taille de l'image
- ce que vous voulez

EXERCICE 06_04 *(PLUS AU PROGRAMME)*

fichier XML
DTD
entités

Créez une entité permettant d'inclure la recette de la pâte brisée (http://www.marmiton.org/recettes/recette.cfm?num_recette=11757) dans l'élément `<ingrédient>pâte brisée</ingrédient>` de la recette de la tarte aux pommes. Modifiez la DTD en conséquence, afin qu'elle puisse contenir soit :

- nom puis quantité
- quantité puis nom
- une autre recette

Créez une entité externe analysée pour cela.

EXERCICE 06_05

namespace

```
<?xml version="1.0" encoding="utf-8"?>
<vol:volcans xmlns:vol="http://www.volcans.com"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://www.volcans.com
                          volcans.xsd">
  <vol:volcan vol:activité="actif" vol:pays="France">
    <vol:nom>Piton de la Fournaise</vol:nom>
    <vol:dernière_éruption>2008</vol:dernière_éruption>
    <vol:altitude>2361</vol:altitude>
  </vol:volcan>
</vol:volcans>
```

Dans quel espace de noms est l'élément `volcan` ?

Dans quel espace de noms est l'attribut `activité` ?

EXERCICE 06_06

namespace

```
<?xml version="1.0" encoding="utf-8"?>
<vol:volcans xmlns:vol="http://www.volcans.com"
             xmlns:pr="http://www.volcans.com" >
  <vol:volcan vol:activité="actif" pays="France">
    <vol:nom>Piton de la Fournaise</vol:nom>
    <dernière_éruption>2008</dernière_éruption>
    <pr:altitude>2361</pr:altitude>
  </vol:volcan>
</vol:volcans>
```

Dans quel espace de noms est l'élément `volcan` ?

Dans quel espace de noms est l'élément `dernière_éruption` ?

Dans quel espace de noms est l'élément `altitude` ?

Dans quel espace de noms est l'attribut `activité` ?

Dans quel espace de noms est l'attribut `pays` ?

EXERCICE 06_07

namespace

```
<?xml version="1.0" encoding="utf-8"?>
<vol:volcans xmlns:vol="http://www.volcans.com"
             xmlns:pr="http://www.machin.com" >
  <vol:volcan vol:activité="actif" pays="France">
    <vol:nom>Piton de la Fournaise</vol:nom>
    <dernière_éruption>2008</dernière_éruption>
    <pr:altitude>2361</pr:altitude>
  </vol:volcan>
</vol:volcans>
```

Dans quel espace de noms est l'élément `volcan` ?

Dans quel espace de noms est l'élément `dernière_éruption` ?

Dans quel espace de noms est l'élément `altitude` ?

EXERCICE 06_08

namespace

```
<?xml version="1.0" encoding="utf-8"?>
<vol:volcans xmlns:vol="http://www.volcans.com"
             xmlns:pr="http://www.machin.com"
             xmlns="http://www.machin.com" >
  <vol:volcan vol:activité="actif" pays="France">
    <vol:nom>Piton de la Fournaise</vol:nom>
    <dernière_éruption>2008</dernière_éruption>
    <pr:altitude>2361</pr:altitude>
  </vol:volcan>
```

```
</vol:volcans>
```

Dans quel espace de noms est l'élément `volcan` ?

Dans quel espace de noms est l'élément `dernière_éruption` ?

Dans quel espace de noms est l'élément `altitude` ?

Dans quel espace de noms est l'attribut `activité` ?

Dans quel espace de noms est l'attribut `pays` ?

EXERCICE 06_09

namespace

```
<?xml version="1.0" encoding="utf-8"?>
<vol:volcans xmlns:vol="http://www.volcans.com"
             xmlns:pr="http://www.machin.com"
             xmlns="http://www.volcans.com" >
  <vol:volcan vol:activité="actif" pays="France">
    <vol:nom>Piton de la Fournaise</vol:nom>
    <dernière_éruption>2008</dernière_éruption>
    <pr:altitude>2361</pr:altitude>
  </vol:volcan>
</vol:volcans>
```

Dans quel espace de noms est l'élément `volcan` ?

Dans quel espace de noms est l'élément `nom` ?

Dans quel espace de noms est l'élément `dernière_éruption` ?

Dans quel espace de noms est l'élément `altitude` ?

EXERCICE 06_10

namespace

```
<?xml version="1.0" encoding="utf-8"?>
<vol:volcans xmlns:vol="http://www.volcans.com"
             xmlns:pr="http://www.machin.com"
             xmlns="http://www.volcans.com" >
  <vol:volcan vol:activité="actif" pays="France">
    <vol:nom xmlns:vol="http://www.machin.com">Piton</
                                                  vol:nom>
    <vol:dernière_éruption>2008</vol:dernière_éruption>
    <pr:altitude>2361</pr:altitude>
  </vol:volcan>
</vol:volcans>
```

Dans quel espace de noms est l'élément `volcan` ?

Dans quel espace de noms est l'élément `nom` ?

Dans quel espace de noms est l'élément `dernière_éruption` ?

Dans quel espace de noms est l'élément `altitude` ?

EXERCICE 06_11

namespace

```xml
<?xml version="1.0" encoding="utf-8"?>
<volcans xmlns:vol="http://www.volcans.com">
  <volcan vol:activité="actif" pays="France">
    <dernière_éruption rien="rien">2008</dernière_éruption>
    <nom xmlns="http://www.machin.com">Piton</vol:nom>
    <altitude>2361</altitude>
  </volcan>
</volcans>
```

Dans quel espace de noms est l'élément `volcan` ?

Dans quel espace de noms est l'élément `dernière_éruption` ?

Dans quel espace de noms est l'élément `nom` ?

Dans quel espace de noms est l'élément `altitude` ?

EXERCICE 06_12

namespace

```xml
<?xml version="1.0" encoding="utf-8"?>
<volcans xmlns="http://www.volcans.com">
  <volcan vol:activité="actif" pays="France">
    <dernière_éruption rien="rien">2008</dernière_éruption>
    <nom xmlns="http://www.machin.com">Piton</vol:nom>
    <altitude>2361</altitude>
  </volcan>
</volcans>
```

Dans quel espace de noms est l'élément `volcan` ?

Dans quel espace de noms est l'élément `dernière_éruption` ?

Dans quel espace de noms est l'élément `nom` ?

Dans quel espace de noms est l'élément `altitude` ?

Cours 7

EXERCICE 07_01

schéma XML

Vous trouverez un schéma XML de bdthèque à l'URL suivant : http://marc.chevaldonne.free.fr/exercices/Ex07_01/bd.xsd.

Étudiez ce schéma et écrivez des documents XML valides pour des BD que vous trouverez sur le site suivant : http://www.bedetheque.com/[2]. Les documents devront bien entendu être valides. Faites bien attention d'utiliser les espaces de noms et le schéma.

EXERCICE 07_02

schéma XML

Rendez vous sur le site web http://www.lfp.fr/ligue1/ pour récupérer les informations vous permettant d'écrire un schéma XML contenant les informations suivantes[3] :

- les clubs de la ligue 1 et/ou 2
- pour chaque club, un ou plusieurs entraîneurs
- pour chaque club, des joueurs
- pour chaque club, la ligue (1 ou 2 seulement) dans laquelle il évolue
- chaque individu devra posséder des informations sur son nom, son prénom, sa nationalité et sa date de naissance
- les joueurs posséderont en plus une information sur leur poste : gardien, défenseur, attaquant ou milieu
- les résultats par journée
- chaque journée contiendra les informations de chaque match
- chaque match contiendra les informations du club jouant à domicile, du club jouant à l'extérieur, du nombre de buts marqués par les deux clubs et des buteurs
- le numéro de la journée (entre 1 et 38)

[2] si vous n'avez pas d'idées de BD, choisissez par exemple : http://www.bedetheque.com/serie-3-BD-De-cape-et-de-crocs.html et http://www.bedetheque.com/serie-57-BD-Garulfo.html

[3] modifiez régulièrement un document XML valide pour vérifier que votre schéma est correct

- ce que vous voulez

Ex07_03 : examen du 3 novembre 2010

Sujet - Randonnées (encore...)

OBJECTIFS

L'objectif de cet examen est de réaliser un schéma XML permettant de stocker des randonnées et leurs données. Une instance de ce schéma devra permettre de stocker des communes, des lieux d'intérêt et des randonnées.
À la fin de cet énoncé, les données de 3 randonnées sont détaillées et devront être utilisées pour remplir l'instance du schéma XML.

Partie 1 : le guide

Si un aveugle guide un aveugle, tous les deux tomberont dans un trou
Saint-Luc

Cette partie a pour but de créer le schéma et son instance.

QUESTION 1 : CRÉATION DU SCHÉMA

Créez un schéma XML `Rando.xsd`. Utilisez un espace de noms. Ce schéma décrira des instances qui devront contenir un élément racine `Guide`, qui contiendra lui-même un et un seul élément `Randonnées`, un et un seul élément `Communes`, un et un seul élément `Lieux`.

QUESTION 2 : CRÉATION D'UNE INSTANCE DE CE SCHÉMA

Écrivez une instance `Rando.xml` de ce schéma que vous ferez évoluer au fur et à mesure des questions pour vérifier que votre schéma correspond à ce qui est demandé.

Partie 2 : les communes

Il y a cent ans commun commune
Comme un espoir mis en chantier
Ils se levèrent pour la Commune
En écoutant chanter Potier
Jean Ferrat

Cette partie a pour but d'écrire la partie du schéma XML permettant de stocker les communes et de remplir l'instance.
La commune contiendra 3 attributs :

- un nom,
- un code postal,
- un code INSEE (unique).

QUESTION 1 : CRÉATION DU TYPE SIMPLE DÉCRIVANT LES CODES DES COMMUNES

Le code postal et le code INSEE sont des entiers à cinq chiffres. Écrivez un type simple personnalisé `codeCommuneType` permettant de décrire ce genre de valeurs.

QUESTION 2 : CRÉATION DU TYPE COMPLEXE COMMUNE

Écrivez un type complexe `CommuneType` contenant trois attributs :

- le nom (obligatoire),
- le code postal (de type `codeCommuneType`),
- le code INSEE (de type `codeCommuneType` et obligatoire).

QUESTION 3 : UTILISATION DU TYPE COMPLEXE

Faîtes en sorte que l'élément `Guide/Communes` puissent contenir 0 ou une infinité de sous-éléments `Commune` de type `CommuneType`.

QUESTION 4 : AMÉLIORATION DE L'INSTANCE

Remplissez votre instance en utilisant les communes utilisées dans les données en annexe.

Partie 3 : les lieux

L'avenir est un lieu commode pour y mettre des songes
Anatole France

Cette partie a pour but d'écrire la partie du schéma XML permettant de stocker les lieux d'intérêts proche des randonnées et de remplir l'instance.
Un lieu contiendra les attributs suivants :

- un nom,
- une altitude,
- un type (sommet ou monument),
- des coordonnées GPS (latitude, longitude).

QUESTION 1 : CRÉATION DU TYPE SIMPLE DÉCRIVANT LES DIFFÉRENTS TYPES DE LIEUX

Écrivez un type simple personnalisé `TypeLieuType` permettant de choisir entre deux valeurs possibles : sommet et monument.

QUESTION 2 : CRÉATION DU TYPE SIMPLE DÉCRIVANT L'ALTITUDE

Écrivez un type simple personnalisé altitudeType permettant de décrire une altitude comme une chaîne de caractères faite d'un ou plusieurs chiffres suivis d'un `m`.

QUESTION 3 : LATITUDE ET LONGITUDE DU LIEU

Une longitude peut être décrite par un nombre réel avec 4 chiffres après la virgule variant entre -180 et 180. Créez un type simple `LongitudeType` permettant de la décrire.
Une latitude est décrite comme la longitude, sauf qu'elle varie entre 0 et 90. Créez un type simple `LatitudeType` dérivant de `LongitudeType` par restriction (*ok, ça ne sert à rien dans ce cas, mais c'est pour l'exercice...*).
Créez un groupe d'attributs `GPSType` contenant deux attributs latitude et longitude utilisant les types précédents.

QUESTION 4 : CRÉATION DU TYPE COMPLEXE POUR LES LIEUX

Écrivez un type complexe `LieuType` qui contiendra les attributs :

- nom (obligatoire),
- altitude (utilisant le type `altitudeType` et optionnel),
- type (utilisant le type `TypeLieuType` et obligatoire),
- une latitude et une longitude.

QUESTION 5 : UTILISATION DU TYPE COMPLEXE

Faîtes en sorte que l'élément `Guide/Lieux` puissent contenir 0 ou une infinité de sous-éléments `Lieu` de type `LieuType`.

QUESTION 6 : AMÉLIORATION DE L'INSTANCE

Remplissez votre instance en utilisant les lieux utilisés dans les données en annexe.

Partie 4 : les randonnées

Voyager, c'est être infidèle. Soyez-le sans remords. Oubliez vos amis avec des inconnus.
Paul Morand

Cette partie a pour but d'écrire la partie du schéma XML permettant de stocker les randonnées et de remplir l'instance.
La randonnée contiendra :

- un nom,
- une durée,
- une distance,
- un dénivelée,
- un point culminant et un point le plus bas,
- une description,
- des lieux d'intérêt proches,
- des communes traversées par la randonnée.

QUESTION 1 : CRÉATION DU TYPE COMPLEXE

Créez un type complexe `RandoType` qui contiendra les informations suivantes :

- un nom,
- une durée,
- une distance (nombre réel suivi de `km`),
- un dénivelée,
- un point culminant et un point le plus bas.

Choisissez les types les plus adaptés pour chacune de ces informations.

Modifiez `RandoType` pour qu'il puisse contenir dans n'importe quel ordre, un et un seul élément `description`, un et un seul élément `lieux_proches`, un et un seul élément `communes_traversées`.

QUESTION 2 : UN ÉLÉMENT MIXTE

Modifiez le sous-élément `description` de `RandoType` pour qu'il soit mixte et puisse contenir éventuellement et dans n'importe quel ordre et n'importe quelle quantité des sous-éléments `lieu` et `commune`.

QUESTION 3 : LIEUX PROCHES ET COMMUNES TRAVERSÉES

Modifiez le sous-élément `lieux_proches` de `RandoType` pour qu'il puisse contenir des sous-élément `lieu` (possédant un attribut `nom` et un attribut `type` de type `TypeLieuType`).
Modifiez le sous-élément `communes_traversées` de `RandoType` pour qu'il puisse contenir des sous-éléments `commune` (possédant un attribut `code_INSEE` du type `codeCommuneType`).
Pour le moment, aucun lien n'est fait avec les lieux et communes précédents ; ceci sera fait dans la partie suivante.

QUESTION 4 : UTILISATION DU TYPE COMPLEXE

Faîtes en sorte que l'élément `Guide/Randonnées` puissent contenir 0 ou une infinité de sous-éléments `Randonnée` de type `RandoType`.

QUESTION 5 : AMÉLIORATION DE L'INSTANCE

Remplissez votre instance en utilisant les randonnées utilisées dans les données en annexe.

Partie 5 : key/keyref

Perdre les clefs de son appartement n'est pas dramatique... si en plus vous avez perdu l'adresse.
Pierre Perret

Cette partie a pour but de rendre unique les communes et les lieux des instances XML et de permettre de les référencer.

QUESTION 1 : UNICITÉ DES COMMUNES ET DES LIEUX

Rendez les éléments `Guide/Communes/Commune` et `Guide/Lieux/Lieu` uniques à l'aide de clés (`key`).

Les communes seront rendus uniques par leur `code_INSEE`.

Les lieux seront rendus uniques par la combinaison `nom` + `type`.

QUESTION 2 : RÉFÉRENCES AUX COMMUNES ET AUX LIEUX

À l'aide de `keyref`, faites en sorte que chaque élément `Guide/Randonnées/Randonnee/communes_traversées/commune` fasse toujours référence à un lieu `Guide/Communes/Commune`.

À l'aide de `keyref`, faites en sorte que chaque élément `Guide/Randonnées/Randonnee/lieux_proches/lieu` fasse toujours référence à un lieu `Guide/Lieux/Lieu`.

CONSEIL : VÉRIFICATION DANS L'INSTANCE

Vérifiez dans votre instance qu'il ne peut pas y avoir de doublons et que les références fonctionnent correctement.

Annexe : Données

RANDONNÉES

Nom	Randol
Communes	Saint-Saturnin (CP : 63450 / INSEE : 63396) Cournols (CP : 63450 / INSEE : 63123)
Description	Sans aller jusqu'au très joli village de Randol, qui mérite le détour, cette balade parcourt le plateau volcanique séparant les vallées de la Veyre et de la Monne, dans un paysage où se mêlent le sauvage et le cultivé.
Durée	2h15
Distance	7,5 km
Dénivelée	214 m
Point culminant	701 m
Point le plus bas	510 m
Lieux proches	Le Puy de Peyronère (716 m / sommet / lat.: 45,6491 / long.: 3,1014) Église romane de Saint-Saturnin (517 m / monument / lat.: 45,6603 / long.: 3,0931)

Nom	Le Puy de Saint-Sandoux
Communes	Saint-Sandoux (CP : 63450 / INSEE : 63395) Ludesse (CP : 63320 / INSEE : 63199)
Description	Au coeur d'un pays de transition entre Allier et Sancy, l'anonyme Puy de Saint-Sandoux est un étonnant belvédère au pied duquel pommiers et vignes connaissent un renouveau.
Durée	2h45
Distance	9 km
Dénivelée	377 m
Point culminant	798 m
Point le plus bas	600 m
Lieux proches	Le Puy de Saint-Sandoux (848 m / sommet / lat.: 45,6286 / long.: 3,1083) Le Puy de Peyronère (716 m / sommet / lat.: 45,6491 / long.: 3,1014) Château de Travers (613 m / monument / lat.: 45,6467 / long.: 3,1058)

Nom	Le Puy de Peyronère
Communes	Saint-Amant-Tallende (CP : 63450 / INSEE : 63315) Saint-Sandoux (CP : 63450 / INSEE : 63395) Saint-Saturnin (CP : 63450 / INSEE : 63396)
Description	Dans un décor lumineux de vergers, ce chemin offre des panoramas sur la vallée de la Monne et la Comté, et découvre le riche patrimoine de Saint-Saturnin, Saint-Sandoux et Saint-Amant-Tallende.
Durée	3h15
Distance	10 km
Point culminant	620 m
Point le plus bas	430 m
Lieux proches	Le Puy de Peyronère (716 m / sommet / lat.: 45,6491 / long.: 3,1014) Église romane de Saint-Saturnin (517 m / monument / lat.: 45,6603 / long.: 3,0931) Château de Travers (613 m / monument / lat.: 45,6467 / long.: 3,1058)

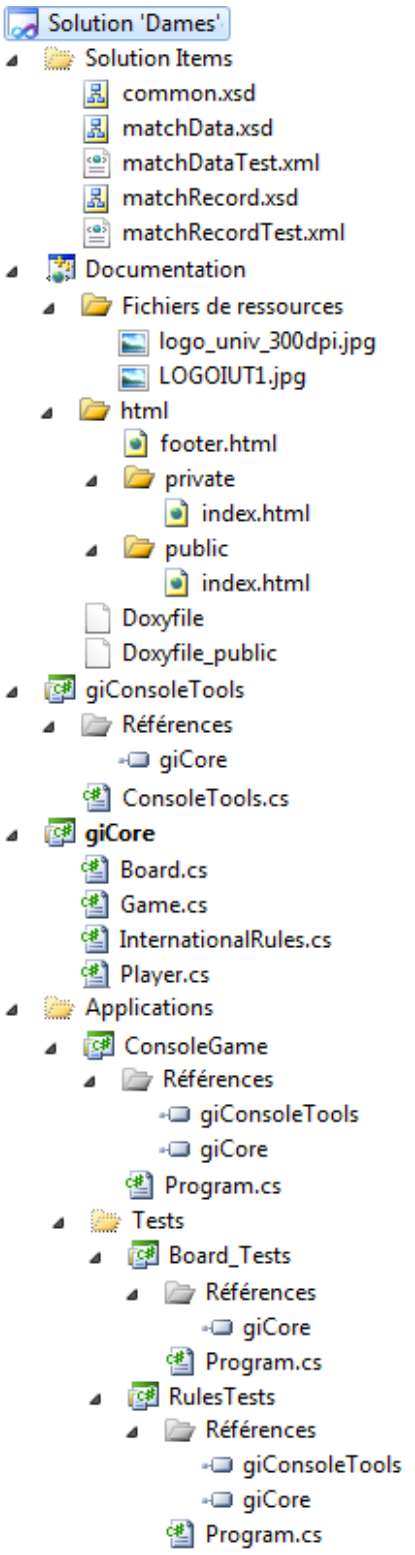
Ex07_04 : examen du 10 novembre 2011

Sujet - un schéma pour solution

OBJECTIFS

L'objectif de cet examen est de réaliser un schéma XML permettant d'organiser des solutions de Visual Studio. Une instance de ce schéma devra permettre de représenter une solution, ses projets, ses fichiers, etc...

Dans cet énoncé, les données (simplifiées) de la solution `Dames.sln` sont données ci-contre pour être utilisées pour remplir l'instance du schéma XML.



QUESTION 1 : CRÉATION DU SCHÉMA ET D'UNE INSTANCE

Créez un schéma XML, permettant de décrire une solution VisualStudio. Utilisez un espace de noms. Utilisez les informations suivantes :

- une seule solution par instance
- une solution possède :
    - obligatoirement un nom,
    - puis pèle-mêle :
        - des «dossiers de solutions» («`Solution Items`», «`Applications`» et «`Tests`»)
        - et des projets

- un «dossier de solution» possède :
    - obligatoirement un nom
    - puis pèle-mêle :
        - des «dossiers de solution»
        - des projets
        - des fichiers
- un fichier possède :
    - obligatoirement un nom qui :
        - peut terminer par :
            - `.cs`
            - `.xml`
            - `.xsd`
            - `.html`
            - `.jpg`
        - ou commencer par `Doxyfile`
        - et posséder au maximum 25 caractères
- un projet possède :
    - obligatoirement un nom
    - obligatoirement un type parmi :
        - `csproj`
        - `vcproj`
    - obligatoirement un nom de fichier de sortie devant se terminer par :
        - `.exe`
        - ou `.dll`
    - des références
    - puis pèle-mêle :
        - des fichiers
        - des dossiers («`Fichiers de ressources`», «`private`», «`public`», «`html`»)
- un dossier possède :
    - obligatoirement un nom
    - puis pèle-mêle :
        - des dossiers,
        - des fichiers
- une référence possède :
    - obligatoirement un nom devant se terminer par `.dll`

Choisissez entre attributs et éléments et respectez toutes les contraintes décrites dans la liste ci-dessus.

Créez une instance valide utilisant les données présentées dans l'image page 3. Vous pourrez également utiliser les informations dans le tableau suivant :

Projet	Fichier de sortie	Type	
`ConsoleGame`	`ConsoleGame.exe`	`csproj`	Application
`Documentation`	`Documentation.exe`	`vcproj`	Application
`giConsoleTools`	`giConsoleTools.dll`	`csproj`	Bibliothèque de classes
`giCore`	`giCore.dll`	`csproj`	Bibliothèque de classes
`Board_Tests`	`Board_Tests.exe`	`csproj`	Application
`Rules_Tests`	`Rules_Tests.exe`	`csproj`	Application

QUESTION 2 : PROJETS UNIQUES ET RÉFÉRENCES

• Modifiez votre schéma pour garantir que les projets sont uniques. Ils seront rendus uniques par la valeur de leur fichier de sortie qui devra être unique dans chaque instance.
• Modifiez le schéma pour que les références des projets fassent toujours référence à un fichier de sortie d'un projet.

Aide : le motif xpath suivant : `.//element` *permet d'atteindre n'importe quel élément* `element` *dans les descendants de l'élément courant.*

QUESTION 3 : DEUX TYPES DE PROJETS

Grâce à des dérivations, créez dans votre schéma deux types de projet :
• un pour les bibliothèques de classes : un projet dont le fichier de sortie doit se terminer par `.dll`
• un pour les applications : un projet dont le fichier de sortie doit se terminer par `.exe`
Faîtes en sorte qu'on ne puisse plus utiliser un projet sans utiliser un de ses types dérivés, bibliothèque de classes ou application.