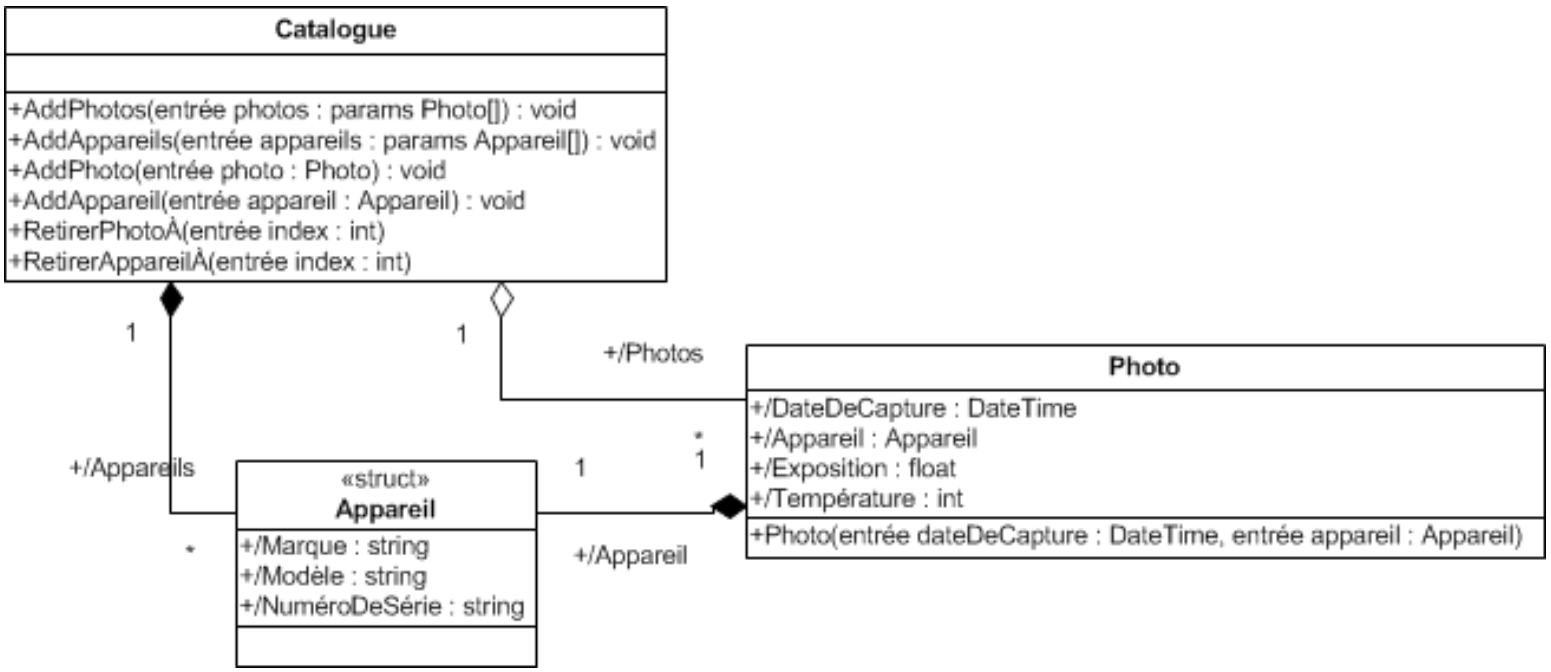
TP 3

Les objectifs de ce troisième TP sont :

- l'utilisation des collections (List, Dictionary)
- l'utilisation des protocoles d'égalité,
- l'encapsulation des collections.

EXERCICE 1 : LIST

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 Appareil("EOS 1D X", "Canon", "SN9012")
```

```
new Photo(catalogue.Appareils[2], new DateTime(2012, 01, 01)),
new Photo(catalogue.Appareils[2], new DateTime(2012, 02, 01)),
new Photo(catalogue.Appareils[1], new DateTime(2012, 03, 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.

Protocoles d'égalité

À partir du résultat précédent, implémentez les protocoles d'égalité sur `Appareil` et sur `Photo`. Deux `Appareil`s seront égaux si et seulement si ils sont de la même `Marque` et ont le même numéro de série. Deux `Photo`s 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 `AddPhotos` et `AddAppareils` de la classe `Catalogue` pour qu'elles n'ajoutent les `Photo`s et les `Appareil`s, 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.

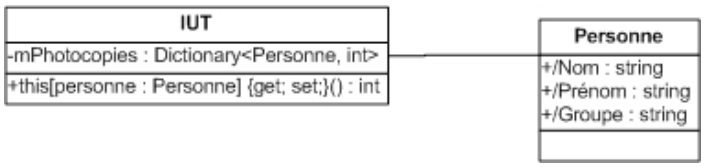
Encapsulation des collections

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 02 : DICTIONARY

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 ?

Compléments pour le cours 3

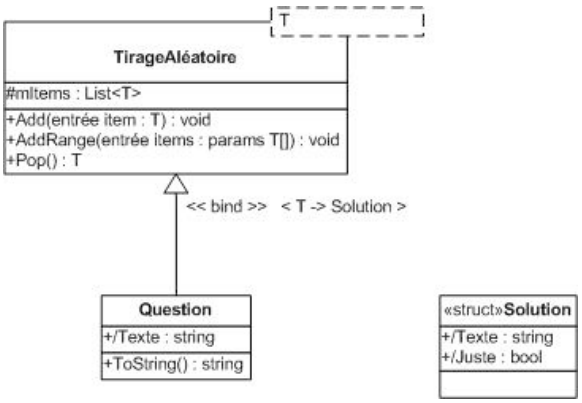
En complément du TP3, voici d'autres exercices sur des thèmes abordés lors des 3 premiers cours, que je vous conseille de faire afin de vous entrainer davantage.

EXERCICE 03_01

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 03_02

Generics
IEnumerable<T>
List<T>

Reprenez l'exemple précédent. On souhaite modifier la classe `TirageAléatoire` pour permettre de faire l'énumération des éléments `T` de cette collection de manière aléatoire (sans retirer les éléments pendant l'énumération).

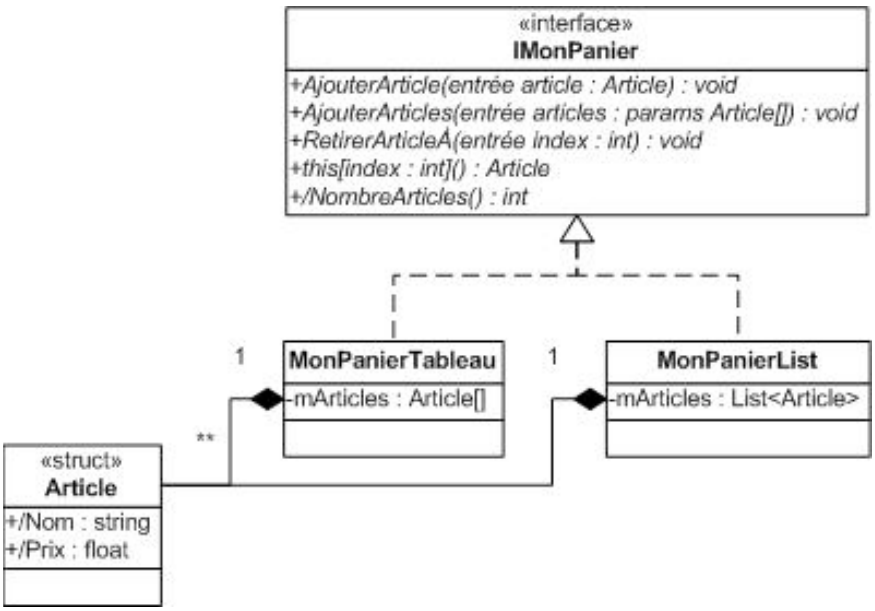
Modifiez `TirageAléatoire` pour qu'elle implémente `IEnumerable<T>`.

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 03_03

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'`Article`s.

Implémentez cette interface de deux manières différentes :

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

Comparez les deux méthodes.

EXERCICE 03_04

égalités
IEqualityComparer<T>

Ajoutez aux résultats de l'exercice 1 du TP3, un nouveau `EqualityComparer` pour la structure `Appareil`. Celui-ci considérera deux `Appareil`s 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 `Photo`s 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

À partir du résultat précédent, 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 `Photo`s par ordre croissant d'`Exposition` (puis par ordre croissant de `Température` si les `Expositions` sont égales).

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

EXERCICE 03_07

Dictionary<TKey, TValue>

Reprenez la classe `ClassementCurling` de l'exercice 02_11. 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.