TP 4

Les objectifs de ce quatrième TP sont :

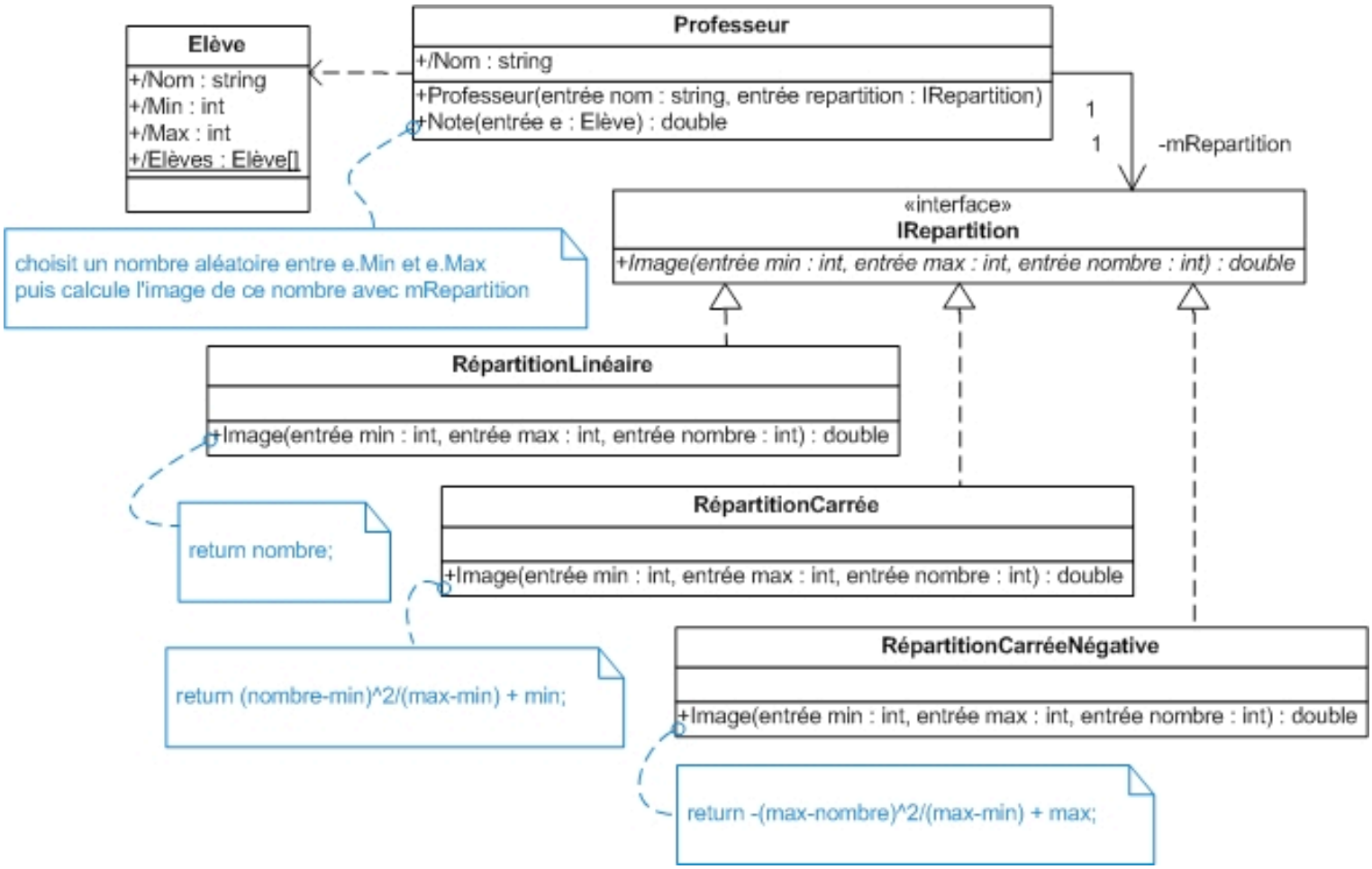
- l'utilisation de délégués
- l'utilisation d'événements
- l'utilisation du pattern standard des événements

STRATEGY ET INTERFACE

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

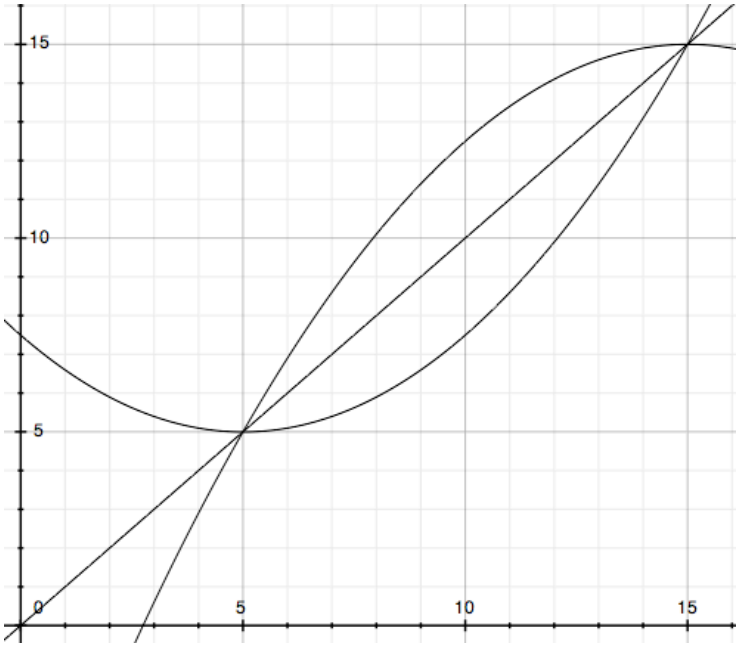
Un `Elève` possède un `Nom`, une note `Min`, une note `Max`. On a également un tableau d'`Elèves` prédéfinis avec au moins 3 `Elèves` dont les notes `Min` et `Max` sont différentes.

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`.

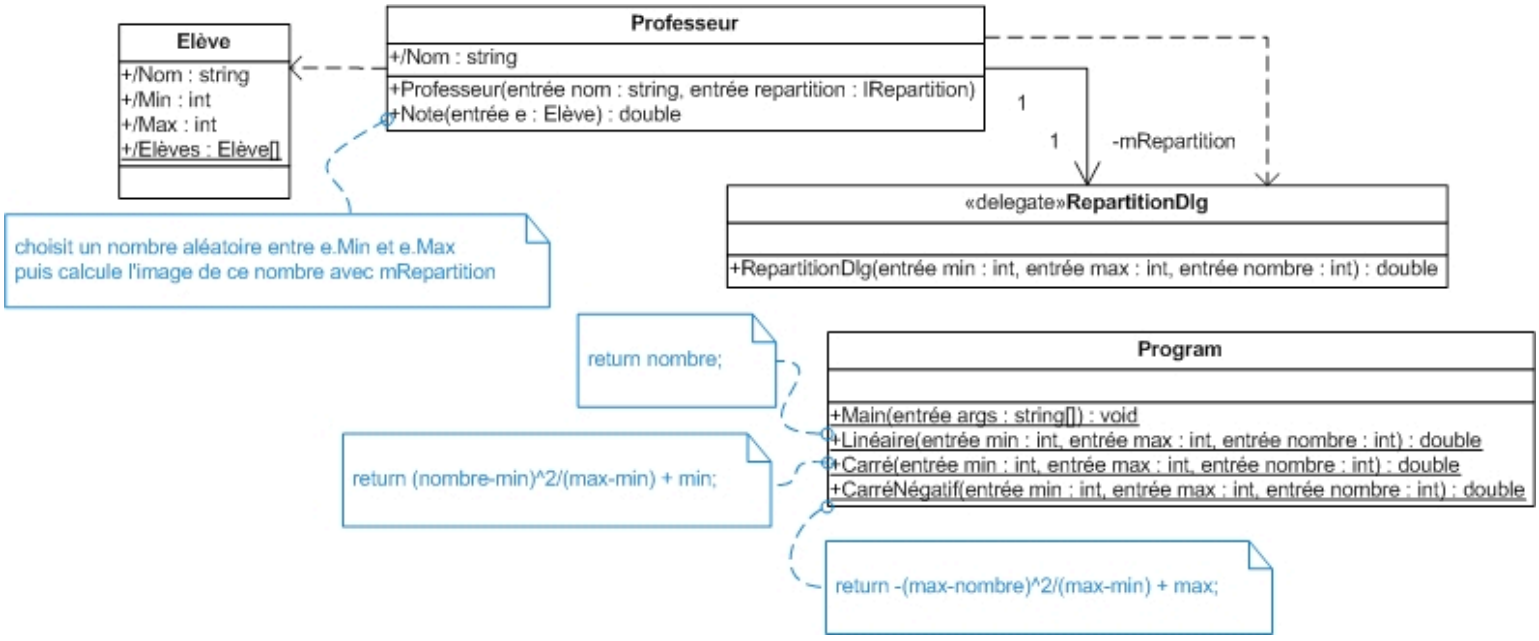


Un `Professeur` possède un `Nom` et une méthode de `IRépartition`. La méthode `Note` choisit un nombre aléatoirement entre `Min` et `Max` inclus de l'`Elève` passé en paramètre, puis utilise la classe concrète de `IRépartition` pour modifier cette note. La note obtenue est minorée à 0 et majorée à 20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

STRATEGY ET DÉLÉGUÉ

Réécrire le programme précédent en utilisant un délégué à la place de l'interface comme le propose le diagramme de classes suivant. Le diagramme propose de mettre les méthodes Linéaire, Carré et CarréNégatif en méthode statique de Program, mais vous pouvez aussi les mettre dans une autre classe ou dans un autre assemblage.



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.

Compléments pour le cours 4

En complément du TP4, voici d'autres exercices sur des thèmes abordés lors du quatrième cours, que je vous conseille de faire afin de vous entrainer davantage.

EXERCICE 04_01

delegate
Strategy

Reprenez l'exercice 02_18 avec le pattern Strategy sur `MonPanier`, en utilisant un delegate à la place de l'interface et des classes concrètes.

EXERCICE 04_02

delegate
Strategy
Func

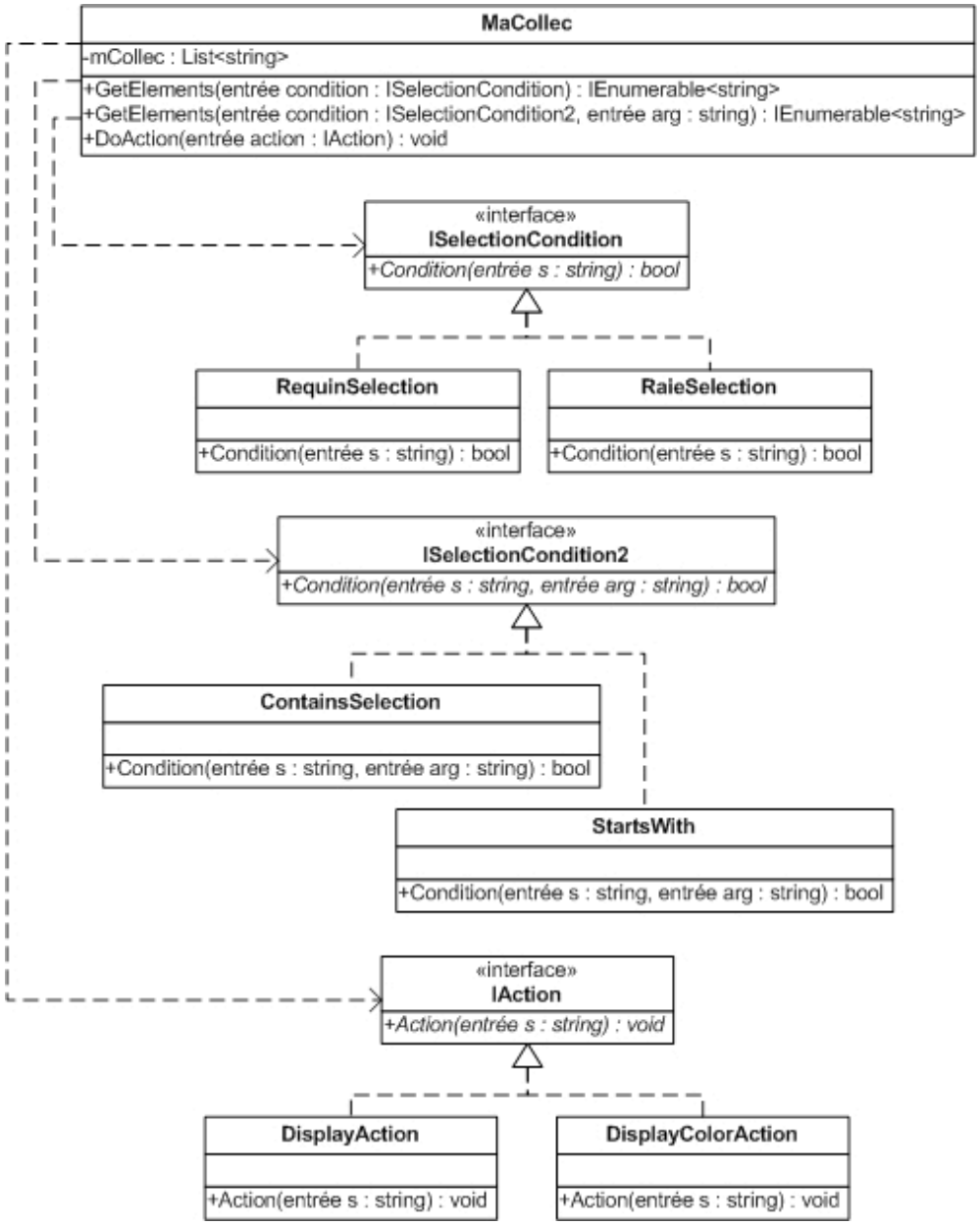
Reprenez l'exercice précédent en utilisant un delegate prédéfini `Func`.

EXERCICE 04_03

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-dessous 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 `ISelectionCondition` 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_04

delegate

Reprenez l'exercice 04_03, 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_05

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 la troisième partie du TP) 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`.