

- Téléchargez l'archive TP03.zip et décompressez-la dans votre répertoire M3105
- Ouvrez dans Netbeans le projet TP03

Exercice 1. Mister Patate

Composition, Agrégation (Cours 3, Chapitre 7)

On veut développer un jeu de type « Mister Patate » où l'on s'amuse à modéliser un **Visage** doté de plusieurs attributs :

Quoi ma gueule ?



- Possède forcément un **Nez**, caractérisé par sa longueur
- Est forcément associé à une **Ethnie**, décrite par un type (une variété de patate en l'occurrence 😊).
- Peut arborer une **Moustache**, caractérisée par sa largeur
- Peut également, surtout à l'adolescence, être porteur de **Boutons** (caractérisés par leur diamètre)
- Peut porter, ou pas, un **Chapeau**, caractérisé par son poids
- Peut être agrémenté de **Bijoux** caractérisés par leur prix.

Question 1.1. Modéliser

Complétez le diagramme de classes ci-dessous en réfléchissant bien à la nature des associations (composition ou agrégation) entre **Visage** et chaque attribut, ainsi qu'à leur cardinalité.

Nez
- longueur : int
+ Nez(longueur : int)
+ getLongueur() : int

Ethnie
- type : string
+ Ethnie(type : string)
+ getType() : string

Moustache
- largeur : int
+ Moustache(largeur : int)
+ getLargeur() : int

Bouton
- diametre : int
+ Bouton(diametre : int)
+ getDiametre() : int

Chapeau
- poids : int
+ Chapeau(poids : int)
+ getPoids : int

Bijou
- prix : int
+ Bijou(prix : int)
+ getPrix() : int

Visage
+ Visage(longueurNez : int, ethnie : const Ethnie &)
+ setMoustache(largeur : int) : void
+ addBouton(diametre : int) : void
+ setChapeau(unChapeau : Chapeau &) : void
+ addBijou(unBijou : const Bijou &) : void
+ Visage(const Visage & unVisage)
+ operator=(const Visage & unVisage) : Visage &
+ ~Visage()

Question 1.2. Coder

- On vous fournit le code des classes **Ethnie**, **Nez**, **Moustache**, **Chapeau**, **Bouton** et **Bijou** (toutes codées de manière minimaliste dans le fichier **AttributsVisage.h**). Pour chacune de ces classes, on a aussi surchargé l'opérateur << pour pouvoir écrire ces objets sur un flux.
- Vous devez compléter la spécification et l'implémentation de la classe **Visage** (fichiers **Visage.h** et **Visage.cpp**). Il vous faudra aussi surcharger l'opérateur << pour pouvoir écrire un objet **Visage** sur un flux de sortie.
- Le fichier **exercice1.cpp** contient un petit test de création et d'affichage d'un **Visage**. Vous verrez en commentaire le résultat qui est attendu.

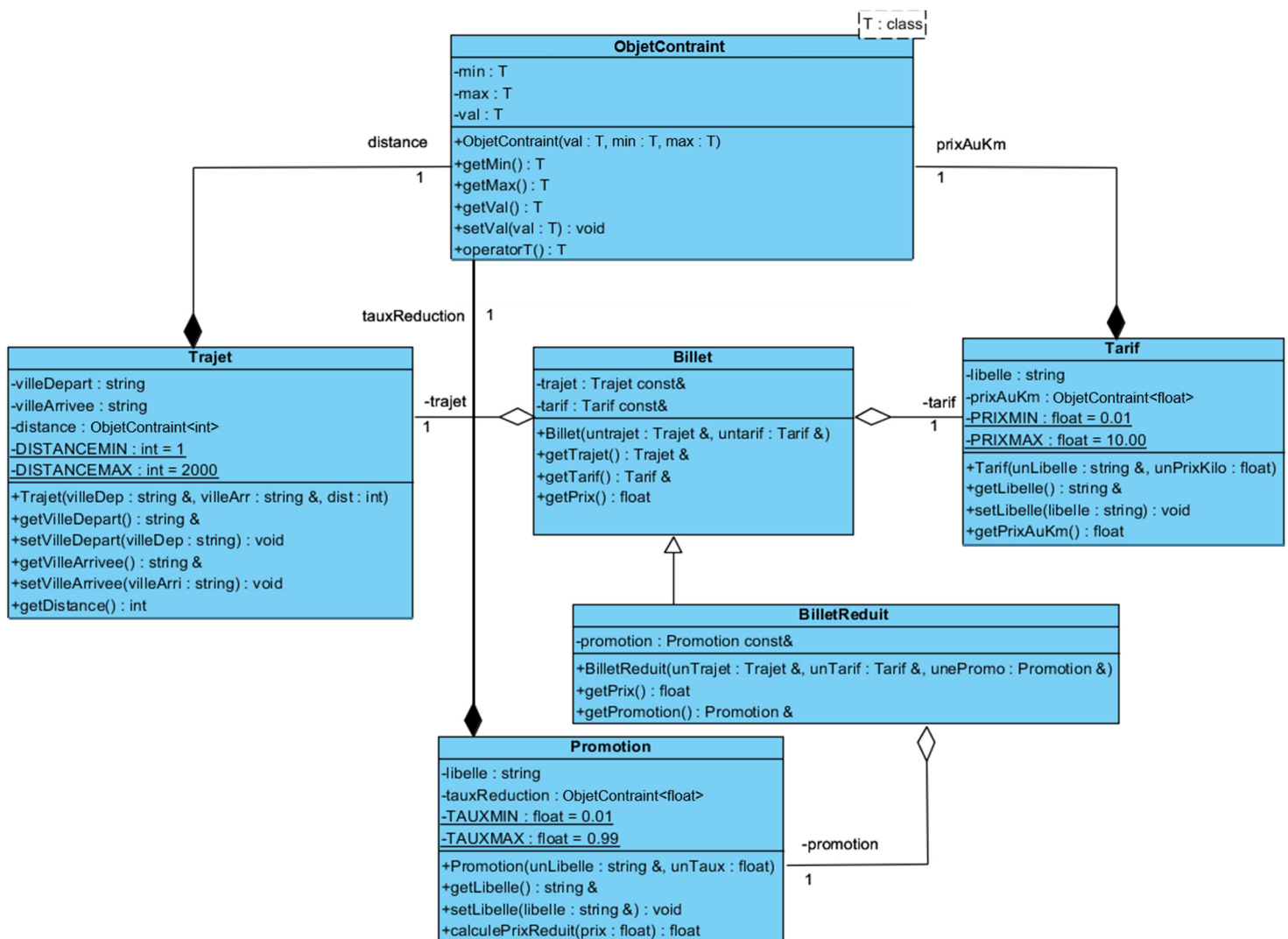
Exercice 2. SNCF, c'est possible

Héritage, Polymorphisme (Cours 3, Chapitre 8)

Dans cet exercice, nous allons modéliser l'offre de transport d'une petite compagnie ferroviaire régionale. On introduit les notions « métier » suivantes :

- Un **Trajet** comporte une ville de départ et une ville d'arrivée (ex. « Grenoble », « Lyon ») et une distance en km qui est un entier compris entre 1 et 2000
- Un **Tarif** est caractérisé par un intitulé (ex « tarif normal 2ème classe ») et un prix au km qui est un réel compris entre 0.01 et 10 €
- Un **Billet** est composé d'un trajet et d'un tarif, on doit pouvoir connaître son prix
- Une **Promotion** est caractérisée par un libellé (ex « Promotion de Noël ») et d'un taux de réduction qui est un réel compris entre 0.01 et 0.99
- Un **BilletReducit** est un **billet** auquel on applique une **Promotion** et dont on doit pouvoir calculer le prix.

On vous propose le diagramme de classe ci-dessous qui a été rapidement produit par un stagiaire probablement distrait. En l'implémentant, vous devrez corriger les éventuelles erreurs sur les types de paramètres (références, const, ...) et compléter s'il vous manque des méthodes.



Question 2.1. Coder les classes métier

- Ecrire les classes **Tarif**, **Trajet**, **Billet**, **Promotion**, **BilletReducit**
- On utilisera le *template* **ObjetContraint<T>**, développé précédemment et fourni ici, pour représenter les nombres qui doivent être contraints (pensez à déclarer des constantes de classe pour représenter les bornes)
- Implémenter pour chacune des classes l'opérateur << d'écriture sur un flux
- **Compiler et tester les classes au fur et à mesure (F9)**, dans l'ordre indiqué ci-dessus. Pour tester une classe, écrivez dans le fichier **exercice2.cpp** une procédure de test dédiée à cette classe ; appelez cette procédure dans la procédure **main** puis commentez l'appel une fois le test validé

Question 2.2. Ecrire une petite interface de création de Billets

- Dans le fichier **exercice1.cpp**, écrivez une fonction **newBillet** :
 - La fonction recevra en paramètre un **Conteneur<Trajet>**, un **Conteneur<Tarif>** et un **Conteneur<Promotion>**. On utilisera donc le *template* **Conteneur<T>** développé précédemment (fourni).
 - Cette procédure devra demander à l'utilisateur (en utilisant les méthodes **afficher** et **choisir** du *template* **Conteneur<T>**) :
 1. De choisir un **Trajet** parmi ceux disponibles dans le conteneur
 2. De choisir un **Tarif** parmi ceux disponibles dans le conteneur
 3. D'indiquer s'il souhaite une **Promotion**
 4. Si oui, de choisir une **Promotion** parmi celles disponibles dans le conteneur
 - La fonction renverra en résultat un pointeur sur un **Billet** (ou un **BilletReducit** s'il y a eu une **Promotion** choisie). Le **Billet** (ou le **BilletReducit**) devra être alloué dynamiquement et construit en fonction du **Trajet**, du **Tarif** et de l'éventuelle **Promotion** choisis.
- Dans la procédure **main** du fichier **exercice1.cpp** :
 - Déclarez un **Conteneur<Trajet>**, un **Conteneur<Tarif>** et un **Conteneur<Promotion>** et ajoutez quelques éléments dans ces conteneurs.
 - Déclarez un **Conteneur<Billet>** (vide au départ)
 - Ecrivez une petite itération pour :
 - Demander à l'utilisateur s'il veut encore créer un nouveau **Billet**
 - Si la réponse est « oui », appelez **newBillet** et ajoutez le **Billet** créé au **Conteneur<Billet>**
 - Si la réponse est « non », sortez de l'itération et affichez le contenu du **Conteneur<Billet>**