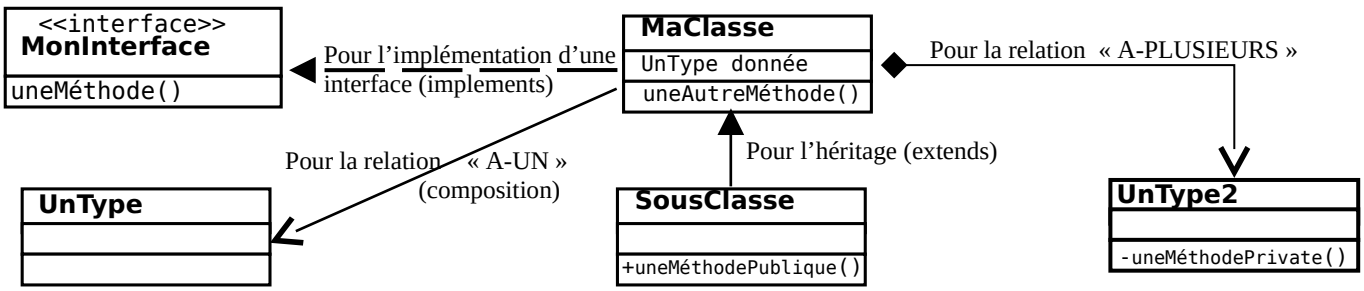
Notations Petit rappel sur les notations à la « UML » à utiliser pour votre rédaction :



Exercice 1 - Principes

Pour chaque question, vous devez justifier votre réponse !

a) Quel(s) inconvénient(s) peuvent survenir dans l'utilisation (en particulier à long terme) du pattern Etat (s'il est codé comme vu en cours) ? Quelles solutions peuvent être utilisées pour palier à ces inconvénients ?
b) Pourquoi le pattern Décorateur est-il classé dans les patterns structurels et pas dans les patterns comportementaux ? Argumentez (le pour et le contre) pour l'une et pour l'autre des classifications.
c) Les Collections Java ne supportent pas le multi-threads par défaut. On doit appeler une méthode statique `Collection synchronizedCollection(Collection)` pour obtenir une collection synchronised. Quel(s) pattern(s) est(sont) utilisé(s) ici ?
d) Supposons que l'on veuille brancher un chargeur avec une prise française (`priseFR`) sur une prise murale américaine (`priseUS`) avec un adaptateur, quelles sont les deux solutions pour représenter ce branchement ? Donnez 2 diagrammes de classes synthétiques différents pour le montrer.

Exercice 2 - Pierre-Feuille-Ciseaux

Vous voulez implanter le jeu Chifoumi (ou Papier-ciseaux-caillou, …) avec deux joueurs ; pour que le jeu se termine, les joueurs doivent, à un moment, jouer des coups différents : aléatoires, toujours le même coup, des coups alternés ...
a) Proposez un diagramme UML complet (avec les méthodes, mais sans les getters/setters !) pour représenter votre jeu ; si vous utilisez un ou plusieurs design patterns, justifiez leurs rôles.
b) Donnez le code Java complet (toutes les classes/interfaces mais sans les getters/setters ni toString) de votre solution ainsi qu'un main qui fait une partie à deux joueurs.
c) En cas d'égalité sur un coup (même coup des deux joueurs), le joueur 1 voudrait pouvoir jouer le coup suivant pour gagner en supposant que l'autre joueur va jouer le même coup (eg Feuille-Feuille, le joueur 1 joue ensuite Ciseaux en supposant que joueur 2 va rejouer Feuille). Modifiez (en justifiant) votre diagramme UML et votre code pour cela.

Exercice 3 - ****Chat

Vous avez mis en ligne votre super serveur de chat « Harmony » pour concurrencer « Discord ». Votre serveur fonctionne parfaitement bien mais les personnes connectées n'arrêtent pas d'utiliser des gros mots dans les salons ! Vous avez trouvé un dictionnaire des « gros mots de l'Orléanais » que vous voulez interdire, et vous voulez censurer ces mots lorsqu'ils sont envoyés pour les remplacer par des * dans les messages affichés sur le chat.
a) Proposez un diagramme de classes pour modéliser votre application de chat et votre censure des gros mots.
b) Fichtre, vous trouvez maintenant vraiment gênant certains émoticones ! Vous ajoutez un nouveau filtre pour les remplacer par un « <3 », diantre ! Quelles sont les modifications à faire sur votre application ?
c) Certains malotrus continuent, malgré votre censure, à écrire de nombreux gros mots ! La répression, voilà la seule réponse ! Vous mettez donc en place des punitions : au premier gros mot ou émoticone inapproprié, un blâme qui bloque l'utilisateur pendant 1 minute. Au deuxième, blocage de 10 minutes. Au troisième, il est définitivement bloqué. Pour permettre aux utilisateurs de se racheter, chaque « <3 » envoyé permettra d'annuler un précédent gros mot. Proposez une modélisation pour ce mécanisme.

Exercice 4 – Au pas Toy Story !

Après votre stage à l'armée, vous êtes excédé par l'anarchie qui règne dans les épisodes de Toy Story ! Pour une gestion de ressources humaines (grh) saine et rigoureuse des prochains épisodes, vous décidez de mettre enfin en place une vraie hiérarchie chez les jouets : un jouet peut avoir un supérieur et aussi avoir des subordonnés. Si un jouet a des subordonnés, c'est un jouet du genre « manager » (eg Buzz l'éclair, le Shérif Woody), sinon c'est un jouet « normal » (Mr Patate, Zigzag, ...). Pour déterminer les cachets à payer à chaque jouet lors du prochain épisode de Toy Story, chaque jouet a un mode de rémunération : « fixe » ou « variable ». La rémunération du type « fixe » est calculée en fonction du niveau hiérarchique du jouet ; la rémunération du type « variable » est calculée en fonction de la « performance » individuelle du jouet (eg. nombre de cascades réalisées).

a) Quels design pattern(s) pourraient a priori être utile(s) dans cette modélisation ? Pour chaque pattern proposé, argumentez pour ou contre son utilisation dans ce cas précis.
b) Proposez une modélisation par un diagramme de classe, en précisant le(s) pattern(s) que vous avez finalement retenu(s) de votre réponse a)
c) Donnez le main qui montre enfin que Buzz l'éclair est le chef de woody !!!

Exercice 5 - Chemins

Vous récupérez le code d'un programme qui avait été mise en place par votre prédécesseur. Ce programme permet, étant donné un chemin complet vers un fichier Windows, d'extraire juste le nom du fichier.
Si vous testez ce code en lui passant en paramètre « C:\\Windows\\hello.dll », il renvoie « hello.dll ».
Le DSI ayant enfin décidé de mettre en place de nouveaux serveurs sous Linux, on vous demande d'ajouter la possibilité de trouver, étant donné un chemin vers un fichier Linux (de type «/user/share/hello.rc »), le nom du fichier en question.
Mais il est déjà envisagé (de nombreuses réunions sont prévues) de vous demandez d'ajouter d'autres fonctionnalités au code : récupérer le nom du répertoire source qui contient le fichier, compter le nombre de sous-répertoire dans le chemin, …

a) Proposez un diagramme de classe
b) Donnez un main qui affiche le répertoire et le fichier du chemin windows C:\\Windows\\hello.dll puis d'un chemin Linux /user/share/hello.rc
c) On veut maintenant ajouter l'extraction du lecteur/volume (C :,D :,…) d'un chemin. Comment peut-on modifier votre solution précédente pour le faire ?