TD 7 - Patrons de conception

Exercice 1. Patron « Singleton »

Les bibliothèques Java proposent une classe `java::util::Random` qui permet de produire des nombres pseudo-aléatoires. Les principales méthodes de la classe Random sont les suivantes :

— `Random()` : constructeur par défaut

— `Random(long seed)` : constructeur avec spécification de graine

— `float nextFloat()` : retourne le prochain nombre de la séquence pseudo-aléatoire, avec une distribution uniforme entre 0.0 et 1.0.

Les nombres générés ne sont jamais réellement aléatoires mais sont produits par une fonction complexe produisant des nombres difficilement prévisibles. La graine (seed) joue ici un rôle particulier : elle permet de donner une valeur initialisant la fonction en question. Plusieurs instances de `java::util::Random` donneront les mêmes suites de nombres aléatoires pourvu que la graine soit la même. Spécifier une graine permet ainsi de reproduire le comportement d'un système même lorsqu'il utilise le hasard. Pour plus de commodité dans le reproduction du comportement d'un système, on cherche souvent à garantir de n'avoir qu'une seule instance générant des nombres aléatoires, cette instance pouvant être initialisée avec une graine.

Question. Utilisez le patron de conception singleton pour proposer le code d'une classe garantissant l'unicité d'une telle instance.

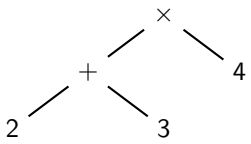
Exercice 2. Figures géométriques

Une figure simple peut être un point, une ligne ou un cercle. Une figure peut être composée d'autres figures, simples ou elles-mêmes composées d'autres figures. Toutes les figures peuvent être dessinées ou translatées.

Question. Utilisez le pattern composite pour construire un diagramme de classe rendant compte de cette hiérarchie d'objets.

Exercice 3. Expressions arithmétiques

Une expression arithmétique peut se représenter de manière arborescente. Par exemple, l'expression $(2+3)\times 4$ peut se représenter comme le résultat de l'opération $\times$ appliquée à 4 et au résultat d'une seconde opération $+$ appliquée à 2 et à 3. 4 et $+(2,3)$ sont dits « opérandes » de l'expression qui a $\times$ comme opérateur. Selon ce principe, une autre notation pour cette expression arithmétique est $\times(+(2,3),4)$, aussi appelée notation « polonaise » et notamment utilisée sur les calculatrice HP. Cette notation peut se représenter de avec un arbre comme suit. Il est à noter qu' il devient ainsi possible de se passer tout à fait de parenthèses : il n'y a aucune ambiguïté lorsqu'on utilise la notation $\times + 234$.



Il y a deux types d'expressions : les expressions binaires comme $+(2,3)$ et les expressions unaires qui utilisent des opérations ne prenant qu'un seul argument, comme par exemple l'opérateur de changement de signe $-$ dans l'expression -1. Ces deux types d'expression sont caractérisées par un opérateur et disposent d'une opération `calculerValeur`(). Le « terme » est un concept plus général qu'une expression : il peut être soit une valeur constante (1,2, 3 ou 4) soit une expression comme $+(2,3)$. Dans tous les cas, un terme doit disposer d'une opération `calculerValeur`(). Un autre type de terme peut être une variable qui, en plus d'une valeur comme pour les constantes, dispose d'un nom.

Question. Utilisez le pattern composite pour produire un diagramme de classes adéquat pour la représentation des expressions arithmétiques.

Exercice 4. Patron « Adapter »

Un éditeur de jeux développe un jeu permettant aux enfants de connaître les animaux. Les enfants peuvent, en particulier, apprendre la forme et le cri des animaux parmi lesquels le chat et la vache. Le chat est modélisé par la classe `LeChat` possédant au moins les deux méthodes `formeChat`() et `criChat`() et la vache est modélisée par la classe `LaVache` possédant les deux méthodes `criVache`() et `formeVache`(). Comme le montrent les noms des méthodes, la première spécification de ce jeu est propre aux animaux modélisés. L'éditeur souhaite améliorer ce jeu en créant une interface commune à tous les animaux qui lui permette d'en ajouter de nouveaux, sans modifier l'interface avec le code client, et d'utiliser le polymorphisme dans la gestion des animaux (manipuler des troupeaux hétéroclites...).

Question 1. Proposez une modélisation des classes pour cette nouvelle version du jeu en faisant apparaître le client. Les classes seront `Chat`, `Vache`... et non plus `LeChat`, `LaVache`...

Question 2. On souhaite réutiliser tout le code développé dans la version précédente (avec `LeChat`, `LaVache`...) dans le nouveau logiciel utilisant les nouvelles classes (`Chat`, `Vache`...).

Proposez une modélisation permettant d'incorporer les anciennes méthodes pour éviter de les récrire. Vous pourrez utiliser le patron de conception « adapter ».