Programmation Orientée Objet

TD 1 – Conception Objet, associations inter-classes

Exercice 1 : Résolution d'équation - Conception UML d'une classe

On souhaite créer un moteur simplifié de résolution d'équations de degré 1. La classe `EqDegOne` représente une équation très simple de type ($ax + b = 0$) où a et b sont les coefficients réels. La résolution se fait par une méthode particulière `solve`. L'équation possède de plus un état indiquant si elle est résolue ou non (avant l'appel de la méthode `solve`, par exemple, elle n'est pas résolue).

a : Dans un premier temps, on considère que les paramètres a et b de l'équation ne peuvent être modifiés ; la classe doit donc permettre d'initialiser les coefficients. Donner une modélisation UML d'un objet de type `EqDegOne`. Pour chaque entité de la modélisation, on précisera le type, la visibilité. On s'intéressera plus particulièrement :

— aux attributs de la classe.
— à chaque méthode, dont on précisera en plus le prototype complet (paramètres et type de retour), excepté pour la méthode `solve`.
— au(x) constructeur(s) dont on précisera également le(s) prototypes.

b : La méthode `solve` calcule systématiquement la solution puis la renvoie.

— Définir les paramètres de la méthode `solve` puis écrire, en pseudo code, son corps.
— Écrire un main instanciant une équation et affichant sa solution.

c : Il est inutile (et pourrait être lourd) de résoudre plusieurs fois la même équation.

— Modifier la modélisation précédente et la méthode `solve` afin d'éviter cela. On y inclura en particulier une nouvelle méthode retournant la solution de l'équation (qu'elle soit résolue ou non au moment de l'appel).
— Écrire, en pseudo-code, la méthode précédente.

d : On souhaite à présent rendre possible la modification des paramètres a et b de l'équation. Quitte à les rendre modifiables, nous pourrions les rendre publiques ! Ne pas le faire [1]. Pourquoi ? Suivre les bonnes pratiques et modifier la modélisation précédente en conséquence. Écrire le pseudo-code d'une des deux méthodes ajoutées.

Exercice 2 : Coffre-fort - Relations entre classes

On souhaite créer un coffre-fort dans lequel seront stockés un certain nombre d'objets précieux.

a : Dans un premier temps, nos coffre-forts seront limités au stockage de pierre précieuses. Sans préciser les attributs et méthodes de chaque classe, définir complètement avec UML la relation entre `Gemstone` et `Safe` (nature et cardinalités).

1. Jamais.

b : Définir complètement avec UML la classe `Gemstone`. Un objet de cette classe symbolisera une pierre précieuse. Chaque pierre possède une valeur *value*, un poids *weight* et un *volume*. Sa valeur n'est déterminée que lors d'une expertise.

c : Définir complètement avec UML la classe `Safe` qui contiendra des pierres précieuses. par un code. On suppose que la capacité globale du coffre est limitée (pour simplifier, en nombre de pierres). Un certain nombre de fonctionnalités sont nécessaires à une bonne utilisation :

— Ouverture et fermeture du coffre ;
— Vérification que le coffre n'est pas plein ;
— Ajout d'une pierre ;
— Retrait d'une pierre ;
— Valeur, poids et volume du coffre