Programmation orientée objet

TD 3 : Héritage

1 Héritage simple

1.1 Un point

On veut créer une classe `Point` représentant un point en deux dimensions qui pourra calculer sa distance à un autre point. Proposer une modélisation UML de cette classe.

1.2 Un polygone

En utilisant cette classe `Point`, proposer une modélisation UML d'une classe `Polygon` qui doit être capable de :

- créer un polygone à partir d'une liste de sommets donnée
- calculer son aire
- calculer son périmètre.

Quel lien y a-t-il entre la classe `Point` et la classe `Polygon` ?

1.3 Un polygone particulier

On souhaite créer une version plus spécifique d'un polygone : un `Triangle`. Proposer une modélisation UML de cette classe. Quel est son lien avec la classe `Polygon`.

2 Héritage un peu plus compliqué

2.1 Des personnages

Créer une classe `Personnage` qui représente un personnage qui possède un score d'attaque et des points de vie et qui peut :

- donner et modifier sa vie ainsi que son score d'attaque

- taper un autre personnage
- se faire taper par un autre personnage

2.2 Des personnages spéciaux

On souhaite donner naissance à des personnages plus spécifiques : une guerrière et un magicien.
La guerrière dispose d'un score de bloquage qui représente son pourcentage de chances de ne pas perdre de vie quand un autre personnage l'attaque. Proposer une modélisation UML de la classe `Guerrière`.
Le magicien peut faire tout ce que peut faire un personnage normal mais il dispose en plus d'un score d'attaque magique qui détermine les dégâts qu'il fait en lançant un sort. Modéliser la classe `Magicien`.

3 Design pattern composite

3.1 Expression arithmétique

On suppose que l'on a deux opérations : + et *. En représentant chaque opération par un noeud, proposer une structure en arbre pour représenter l'expression : $2 * (3 * x + y + z) + t$.

3.2 Composite

Le design pattern composite, qui permet d'utiliser un ensemble d'objets comme on utiliserait un objet simple comprend plusieurs modèles de classes :

- la *feuille* : l'objet de base,
- les *composites* qui représentent les composants qui ont des enfants et qui permettent la manipulation de ces enfants,
- le *composant* qui est l'abstraction de tous les composants (incluant les composites et les feuilles).

Que sont concrètement les *feuilles*, *composites*, *composant* dans le cas de l'expression arithmétique ?

On souhaite pouvoir faire des opérations de dés et faire par exemple `3 * d6 + d6` en écrivant `d6.mult(3).add(d6)`.

Proposer une classe `MultDice` représentant la multiplication par un facteur multiplicateur d'un dé. On voudra pouvoir obtenir la valeur de cet objet et le faire rouler.

De la même manière proposez une classe `SumDice` représentant la somme de deux dés.

Compléter la modélisation par une classe `DiceComponent` représentant les expressions de dés et l'utiliser pour que `SumDice` et `MultDice` soient des opérations d'expressions de dés.