TD de programmation orientée objet en Java
Héritage

Exercice 1 : Animaux

Commencez par écrire une classe qui s'appelle `Animal`. Celle-ci doit comporter uniquement :

- une méthode qui s'appelle `cri` qui ne prend aucun paramètre en entrée et qui retourne une chaîne de caractères vide.
- une redéfinition de la méthode `toString` qui retourne simplement la chaîne de caractères "Je suis un animal".

Maintenant, créez une classe qui s'appelle `Chat` qui hérite de la classe `Animal` et qui :

- redéfinit la méthode `cri` afin de retourner la chaîne de caractères "Miaaaw !".
- redéfinit la méthode `toString` afin de retourner la chaîne de caractères "Je suis un chat !".

Enfin, créez une troisième classe qui s'appelle `Chien` qui hérite, elle aussi, de la classe `Animal` et qui contient :

- une redéfinition de la méthode `toString` qui retourne la chaîne de caractères "Je suis un chien !".
- une méthode qui s'appelle `aboyer` qui ne prend aucun paramètre en entrée et qui retourne la chaîne de caractères "Haw haw !".

Testez vos trois classes grâce au bout de code suivant. Que remarquez vous ?

```
public class TestAnimaux {
	public static void main(String[] args){

		// Création d'un animal, un chat et un chien.
		Animal animal = new Animal();
		Chat chat = new Chat();
		Chien chien = new Chien();

		// Affichage des trois animaux (toString)
		System.out.println(animal);
		System.out.println(chat);
		System.out.println(chien);

		// Affichage des cris des animaux
		System.out.println(animal.cri());
		System.out.println(chat.cri());
		System.out.println(chien.cri());
		System.out.println(chien.aboyer());

		// On considère le chat et le chien comme étant des animaux
		Animal achat = chat;
		Animal achien = chien;

		System.out.println(achat);
		System.out.println(achien);
	}
}
```

Essayez d'ajouter l'instruction `System.out.println(achien.aboyer());` à la fin du code. Est-ce que ça fonctionne ? Pourquoi ?

Exercice 2 : Formes 2D

Créez une classe qui s'appelle `Forme2D` qui représente, de façon générique, des formes géométriques à deux dimensions. Cette classe doit contenir :

- un attribut dont la visibilité est `private` qui s'appelle `couleur` et qui représente la couleur de la forme en question (vous pouvez utiliser le type Java `Color`).
- un constructeur par défaut (qui ne prend aucun paramètre) qui crée une forme 2D dont la couleur est noire.
- un constructeur qui prend en paramètre une couleur et qui crée une forme 2D dont la couleur correspond à celle-ci.
- un accesseur qui permet de récupérer la valeur de la couleur de la forme 2D.
- une méthode `aire` (qui est sensée retourner l'aire de la forme 2D) qui retourne 0. (N.B. Cette méthode sera redéfinie dans les classes filles de `Forme2D` pour calculer l'aire correctement)
- une méthode `perimetre` (qui est sensée retourner le périmètre de la forme 2D) qui retourne 0. (N.B. Cette méthode sera redéfinie dans les classes filles de `Forme2D` pour calculer le périmètre correctement)

Maintenant, créez une classe `Rectangle` qui hérite de la classe `Forme2D`. La classe `Rectangle` doit contenir :

- deux attributs supplémentaires `longueur` et `largeur`.
- un constructeur qui prend en paramètres une `couleur`, une `longueur` et une `largeur` et qui instancie le rectangle correspondant.
- les accesseurs adéquats pour les deux attributs `longueur` et `largeur`.
- une redéfinition de la méthode `aire` qui retourne l'aire du rectangle.
- une redéfinition de la méthode `perimetre` qui retourne le périmètre du rectangle.

Créez également une classe `Disque` qui hérite elle aussi de la classe `Forme2D` et qui contient :

- un attribut qui s'appelle `rayon`.
- un constructeur qui prend une `couleur` et un `rayon` en paramètre et qui crée le disque correspondant.
- un accesseur pour l'attribut `rayon`.
- une méthode `diametre` qui renvoie le diamètre du disque.
- les redéfinitions des méthodes `aire` et `perimetre` qui calculent l'aire et le périmètre d'un disque correctement.

Au choix dans l'une des deux classes `Rectangle` et `Disque`, essayez d'écrire un modificateur `setCouleur` qui prend en paramètre une couleur et qui l'attribue à la forme concernée (autrement dit, essayez de modifier l'attribut `couleur` que `Rectangle` et `Disque` héritent de leur classe mère `Forme2D`). Est-ce que ça marche ? Pourquoi ? Modifiez la visibilité de l'attribut `couleur` dans la classe `Forme2D` pour la passer en `protected`. Est-ce que ça marche maintenant ?

Amusez-vous (mais pas trop :-P) à créer des rectangles et des disques. Considérez les tantôt comme appartenant à leurs classes spécifiques, tantôt comme étant des formes 2D et vérifiez quelles sont les méthodes que vous pouvez invoquer ou non.

Exercice 3 : Pokémon

Les Pokémon sont certes de très mignonnes créatures, mais ils sont également un bon exemple pour illustrer l'héritage. Je vous propose donc de commencer par créer une classe `Pokemon` qui contient (entre autres) :

- un attribut `nom` qui représente le nom du Pokémon.
- un attribut `hp` (pour Health Points) qui représente les points de vie du Pokémon.
- un attribut qui s'appelle `atk` qui représente la force de base de l'attaque du Pokémon.
- un constructeur pour instancier des Pokémon adéquatement.
- des getters (accesseurs) qui permettent de consulter les attributs du Pokémon.
- une méthode `isDead()` qui retourne un `boolean` pour indiquer si un Pokémon est mort (`hp == 0`) ou non.
- une méthode `attaquer(Pokemon p)` qui permet au Pokémon appelant d'attaquer le Pokémon passé en paramètre. L'attaque déduit `atk` points de la vie `hp` du Pokémon attaqué `p`.
- une redéfinition de la méthode `toString` qui affiche les informations du Pokémon.

En plus des Pokémon normaux (décrits à travers la classe `Pokemon`) on ressence trois types de Pokémon. Les Pokémon de type Feu, les Pokémon de type Eau et les Pokémon de type Plante (en réalité il existe 17 types en tout mais on ne va pas s'amuser à tous les coder) :

- les Pokémon de type Feu sont super efficaces contre les Pokémon de type Plante et leur infligent deux fois plus de dégâts (`2*atk`). Par contre, ils sont très peu efficaces contre les Pokémon de type Eau ou de type Feu et ne leur infligent que la moitié des dégâts (`0.5*atk`). Ils infligent des dégâts normaux aux Pokémon de type Normal.
- les Pokémon de type Eau sont super efficaces contre les Pokémon de type Feu et leur infligent deux fois plus de dégâts (`2*atk`). Par contre, ils sont très peu efficaces contre les Pokémon de type Eau ou de type Plante et ne leur infligent que la moitié des dégâts (`0.5*atk`). Ils infligent des dégâts normaux aux Pokémon de type Normal.
- enfin, les Pokémon de type Plante sont super efficaces contre les Pokémon de type Eau et leur infligent deux fois plus de dégâts (`2*atk`). Par contre, ils sont très peu efficaces contre les Pokémon de type Plante ou de type Feu et ne leur infligent que la moitié des dégâts (`0.5*atk`). Ils infligent des dégâts normaux aux Pokémon de type Normal.

Créez trois classes `PokemonFeu`, `PokemonEau` et `PokemonPlante` qui héritent de la classe `Pokemon` et qui représentent les trois types de Pokémon susmentionnés. Ensuite, amusez-vous à faire des combats de Pokémon.

Remarque : Connaitre le nom de la classe d'un objet

Afin de connaitre la classe à laquelle un objet `o` appartient, vous pouvez utiliser la méthode `getClass()` (appel : `o.getClasse()`). Pour connaitre le nom de cette classe, vous pouvez utiliser la méthode `getName()` (`o.getClass().getName()`) qui retourne le nom complet de la classe (y compris le nom du package auquel elle appartient) ou la méthode `getSimpleName()` (`o.getClass().getSimpleName()`) qui retourne seulement le nom de la classe sans le nom du package.