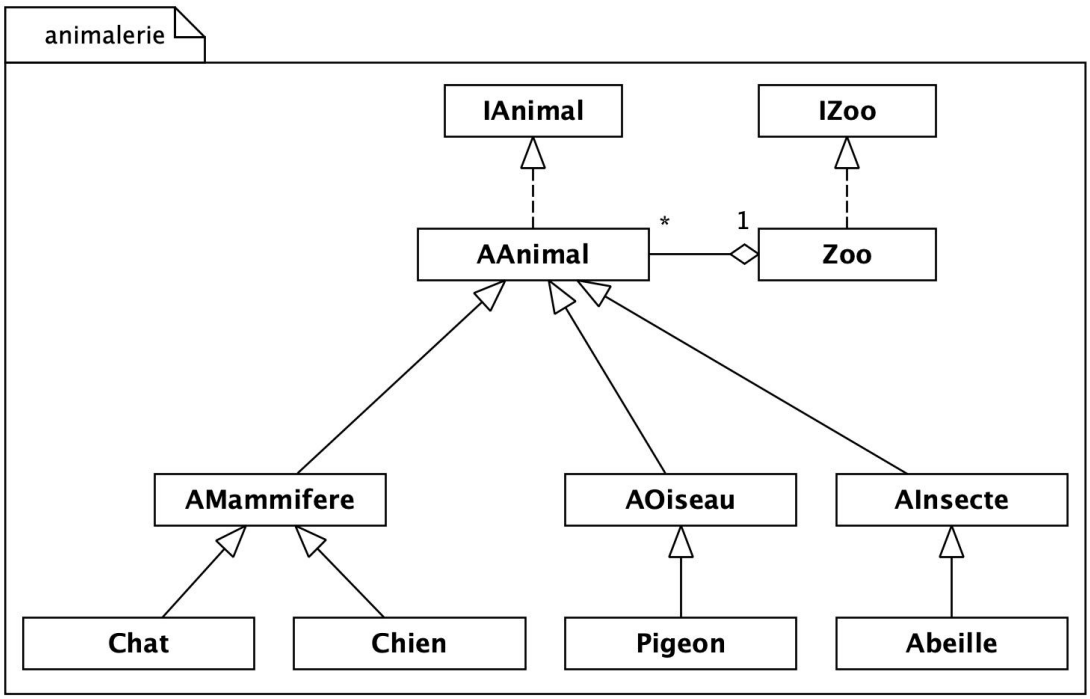
TP n°6

Classes et méthodes abstraites

1 Animalerie le retour

Dans cette partie, nous allons modifier le package `animalerie`. Il est d'ailleurs conseillé de faire une copie du répertoire `animalerie` dans le répertoire du TP5, afin d'en conserver une trace. Voici le diagramme de classes simplifié de la nouvelle version du package :



1. Selon vous, quelles classes ne doivent pas être instanciées ?
2. Quelle(s) solution(s) connaissez-vous pour empêcher l'instanciation (directe) d'une classe ?

Nous souhaitons que la classe `Animal` soit une classe abstraite (nous l'appelons maintenant `AAnimal` ; attention à bien modifier toutes les classes qui l'utilisent). Un animal est caractérisé par un nom, un nombre de pattes, mais aussi un nombre d'ailes. N'oublions pas d'ajouter le nouveau *getter*. Nous rappelons que la classe `AAnimal` possède une méthode `toString` et une méthode `afficher` (qui affiche la chaîne retournée par la méthode `toString`). L'affichage d'un animal est le suivant : `Bidule, animal à 4 patte(s) et 0 aile(s).`

3. La méthode crier étant inutile dans la classe `AAnimal`, est-il possible de la déclarer sans mettre de code ?
4. Écrivez le contenu de la classe `AAnimal`.

Un mammifère (classe abstraite `AMammifere`) possède 4 pattes mais pas d'ailes ; il s'affiche comme suit : `Bob, mammifere`.

5. À votre avis, la classe `AMammifere` doit-elle posséder une méthode `crier` ? une méthode `toString` ? une méthode `afficher` ?
6. Écrivez la classe `AMammifere`.
7. Nous souhaitons maintenant écrire la classe abstraite `AOiseau`. Un oiseau possède obligatoirement 2 ailes et 2 pattes. Il s'écrit comme suit : `Bob, oiseau`. Écrivez cette classe (n'hésitez pas à faire un copier/coller de la classe `AMammifere` pour gagner du temps).
8. Maintenant, nous allons écrire la classe `AInsecte`. Un insecte possède obligatoirement 6 pattes et peut posséder ou non des ailes. Il s'affiche comme suit : `Bob, insecte a 4 aile(s)`. Écrivez cette classe.
9. Écrivez maintenant les autres classes :
   — Un chien possède 4 pattes et doit s'afficher comme suit :
   "*Chien Medor, mammifere*".
   — Un chat possède 4 pattes et doit s'afficher comme suit :
   "*Chat Azrael, mammifere*".
   — Un pigeon possède 2 pattes et 2 ailes et doit s'afficher comme suit :
   "*Pigeon Hector, oiseau*".
   — Une abeille possède 6 pattes et 4 ailes et doit s'afficher comme suit :
   "*Abeille Maya, insecte a 4 aile(s)*".
   — Vous savez également comment ces différents animaux *crient*...
10. Dans le répertoire du TP 6, écrivez la classe de test.

2 Un peu de cruauté

Dans cet exercice, nous supposons qu'il est possible d'arracher une patte ou une aile à un animal. Si le pauvre animal possède le dit membre à arracher, il crie [1].

Dans la classe `AAnimal` :

1. Écrivez la méthode `arrachePatte` : si l'animal possède au moins une patte, il en perd une et crie. Dans le cas contraire, il ne se passe rien.
2. Écrivez la méthode `arracheAile` : si l'animal possède au moins une aile, il en perd une et crie. Dans le cas contraire, il ne se passe rien.
3. Réécrivez les méthodes `toString` de `AMammifere`, `Aoiseau` et `AInsecte` pour permettre de suivre le nombre de pattes et/ou d'ailes. Faites de même, si nécessaire, pour les `Chien`, `Chat`, `Pigeon`, ...

Modifiez votre classe de test pour vérifier le fonctionnement des méthodes ajoutées.

Pour rappel, un `Zoo` possède comme attribut un tableau d'`AAnimal`, correspondant aux cages. Au moment de la construction, soit on spécifie le nombre de cages, soit le nombre de cages est choisi par défaut (valeur aléatoire entre les deux constantes 10 et 20). Il possède les méthodes suivantes :

— `getAnimal` : retourne l'animal situé à la position spécifiée (cette méthode prend en paramètre le numéro de la cage)
— `setAnimal` : ajoute un animal à la position spécifiée (sauf si la cage est occupée)

1. Il est interdit de reproduire ce type de cruauté en dehors de ce TP d'Info0201.

— `toString` et `afficher` pour permettre l'affichage d'un zoo en indiquant le nombre de cages et le contenu de chacune d'elles.

Modifiez la classe `Zoo` comme suit :

4. Dans un premier temps, modifiez l'interface en remplaçant `Animal` par `AAnimal`.
5. Écrivez la méthode `desastre` : cette méthode choisit une cage au hasard et arrache au hasard une patte ou une aile si un animal est présent.
6. Écrivez la méthode `getNombrePattes` qui retourne la somme des nombres de pattes de tous les animaux du zoo.
7. Idem pour les nombres d'ailes (méthode `getNombreAiles`).
8. Si ce n'est déjà fait, modifiez la classe `TestZoo`.

3 Ajouts (facultatif)

Dans cet exercice, nous allons modifier les différentes classes du package `animalerie`.

Tous les animaux possèdent un âge (initialisé à 0 lors de la naissance, *i.e.* lors de la construction). L'âge doit être affiché lorsque l'animal s'affiche et doit pouvoir être récupéré via un *getter*. L'espérance de vie, propre à chaque espèce, est précisée par une constante dans la classe donnée. Une méthode `getEsperanceVie` permet de récupérer l'espérance de vie de l'animal.

Une méthode `vieillir` permet de faire vieillir un animal. Elle retourne `true` si l'animal est encore en vie ou `false` si l'animal meurt de vieillesse. Chaque année au-delà de l'espérance de vie de l'espèce, un animal a une chance sur 2 de mourir.

1. Écrivez la méthode `getEsperanceVie`.
2. Réalisez l'ensemble des modifications.
3. Ajoutez une méthode `vieillir` dans la classe `Zoo` qui permet de faire vieillir tous les animaux du zoo. Quand un animal meurt de vieillesse, il est automatiquement retiré de sa cage.
4. Modifiez les classes de test pour mettre en évidence les modifications.