IUT Paris Descartes – Département Informatique – 2e année

Conception et Programmation à Objets Avancées

Mikal Ziane, Mourad Ouziri, Karim Foughali

TP 1

Thèmes : impact des changements, spécification vs implémentation d'un type, polymorphisme

Exercice 1 (suite du TP1)

Considérez le code de l'exercice 1 fourni sur le serveur (disque commun). Le programme concerne un refuge pour animaux avec un nombre de places limités pour chaque espèces (chiens et chats actuellement).

Nous voulions apporter un certain nombre de changements à ce programme mais nous avons vu au TP1 que certains de ces changements seraient couteux. Aujourd'hui nous corrigeons donc les défauts constatés pour que ces changements puissent être plus faciles.

Limiter l'impact de A : modifier la capacité d'accueil pour une des espèces

On pourrait utilisez une constante symbolique pour définir la taille mais il vaut mieux adopter une structure de données dont la taille maximale n'est pas fixe comme une ArrayList, mais faire la question B avant du coup.

La difficulté est ensuite que chaque capacité dépend de l'espèce animale mais sans doute aussi du refuge donc il n'est pas simple de décider dans quelle classe placer cette information.

Limiter l'impact de B : remplacer les tableaux d'animaux par des ArrayLists

Utilisez, là où c'est possible, des boucles foreach. Pour ensuite pourvoir plus facilement changer entre les types de listes (ArrayList ou LinkedList) déclarer les listes avec l'interface List.

Limiter l'impact de C : remplacer le nom des animaux par un numéro de tatouage

Factorisez les attributs dans une classe abstraite Animal (ou dans une classe AnimalDeleg par délégation) et déclarer les private.

Limiter l'impact de D : ajouter une espèce d'animal

Introduisez une interface IAnimal et utilisez-là autant que possible dans le code client (c'est-à-dire sauf pour les créations d'instances).

Introduisez une méthode ajouter(IAnimal) à Refuge ce qui permet d'externaliser les créations d'instances (dans le main par exemple).

Limiter l'impact de E : changer d'interface utilisateur

Si le temps le permet : écrire un classe RefugeConsole qui appelle Refuge et gère l'IHM alors que Refuge ne s'occupera plus du tout des entrées-sorties.

Exercice 2

Créez un projet *matrices* et importez les sources du répertoire `énoncé` sur le serveur commun.

Le projet est un embryon d'un programme de calcul matriciel. Rappelons qu'une matrice HxL est un objet mathématique similaire à un tableau à deux dimensions : sa hauteur H et sa largeur L. Si M est une matrice 2x3 alors ses postes sont

M11 M12 M13

M21 M22 M23

Un moyen simple d'implémenter une matrice est donc tout simplement à l'aide d'un tableau à deux dimensions (voir la classe **`MatricePleine`**). Toutefois, si de nombreux postes de la matrice sont nuls alors il est possible de gagner de la place en ne stockant que les coordonnées et la valeur des postes non nuls. (Voir la classe **`MatriceCreuse`**). Par exemple si seul le poste d'indices (1,3) est non nul alors une matrice creuse associera simplement à la clé (1,3) la valeur de ce poste (mettons 5) ce qui est plus compacte que de stocker le tableau complet

0 0 5

0 0 0

La classe **`Calcul`** doit permettre de manipuler des matrices (une seule dans cette version) mais sans avoir à se préoccuper de son implémentation, c'est-à-dire **sans dépendre de la classe utilisée pour représenter les matrices** (`MatriceCreuse`, `MatricePleine` voire d'autres classes à l'avenir).

Quelles dépendances sont nuisibles dans la classe Calcul ?

Faites le diagramme d'une architecture qui corrige ce problème et mettez-la en œuvre dans le code.