Le patron de conception Pont

Rappelez-vous le TP n°2, avec l'Abstract Factory. On créait des CountFolderWindows, des ParseFileNameLinux, etc... On avait d'un côté des OS (Windows, Linux, …) et de l'autre côté, des fonctionnalités (compter le nombre de dossier, récupérer le nom de fichier). Si l'on souhaitait rajouter une seule fonctionnalité, on devait rajouter un nombre de classe égal au nombre d'OS à gérer, et si l'on souhaitait rajouter un OS, on devait rajouter un nombre de classe égal au nombre de fonctionnalités à implémenter. Bref, pas super.

Exercice 1

On souhaite réaliser le projet suivant : créer une interface MaListe qui gère des ensemble d'entiers et qui contient trois fonctions : push, pop, et isEmpty. Cette interface pourra être déclinée par des classes abstraites Lifo, Fifo, Random (qui affiche un nombre choisi au hasard dans l'ensemble quand on appelle pop) et Middle (qui affiche le nombre rangé au centre de la liste quand on appelle pop). De plus, chacun de ces listes pourra êtres implémentée à l'aide d'un tableau ou d'une système de liste chaînée.

Voici le schéma de classe du projet :

Après un séjour au club Jamba, vous avez appris à en faire le moins possible. Or, vous vous rendez compte que, le jour où vous voudrez ajouter la gestion des ensembles avec des arbres binaires, il vous faudra rajouter 4 classes, et le jour ou vous souhaiterez ajouter un système de gestion d'ensemble qui affiche le sort à chaque fois le plus petit élément de la liste, il vous faudra rajouter deux classes. En résumé, si vous avez N abstractions et M implémentations, le jour où vous souhaitez rajouter une seule implémentation, vous devrez écrire N classes, et le jour où vous souhaitez rajouter une seule abstraction, vous devrez rajouter M classes.

Tentons de trouver mieux.

Dans la suite, vous devrez implémenter vos classes sans utiliser les classes List de Java (pas d'ArrayList, LinkedList, …).

Vous aurez besoin d'une interface MonImplementation, qui contiendra les méthodes que vous choisirez (du genre getsize, removeElementAt, addElementAt) et dont hériteront les classes ArrayList et LinkedList. Vous aurez aussi besoin d'une classe abtsraite MaListe, qui contiendra un MonImplementation, des fonctions abstraites pop, push et isEmpty, et dont hériteront les classes Fifo, Lifo, Random et Middle. En séparant les systèmes d'accès aux éléments (Lifo, Fifo, …) et les structures internes des listes (Array, LinkedList), on

évite l’explosion de classe lorsque l’on souhaitera, plus tard,

rajouter un élément.

Votre schéma de classe devra ressembler à ceci :