Composite and Proxy Patterns

1 Partie UML

Dans cette partie, vous allez traiter deux exemples indépendants. Vous aurez donc deux diagrammes à réaliser.

1.1 Les Composites – Le simulateur de déplacements de personnes

Au cours de cet exercice, nous allons construire un diagramme de classe incrémentalement. Chacune des questions enrichira donc celui-ci. Pour chaque classe, indiquer ses attributs (privés), ses opérations (incluant la redéfinition de la méthode `toString()`), et son ou ses constructeurs. On omettra les accesseurs/mutateurs pour simplifier le diagramme.

On souhaite simuler le comportement d'un ensemble de personnes, par exemple dans un grand magasin, ou sur un champ de bataille.

- Définissez la classe `Personne`, caractérisée par une position (`x, y` de type `float`), un nom (de type `String`) ainsi qu'une couleur (de type `String`). Vous ajouterez la méthode `deplacer()` qui déplace la personne.
- Définissez la classe `Simulateur` qui contient une liste de `Personne` ainsi que les méthodes `ajouter(Personne)`, `deplacer(Personne)` et `afficher()`.

Question 1 • Créez le diagramme de classes comprenant les classes `Personne` et `Simulateur`.

Le déplacement de chaque personne une par une étant pénible, on désire ajouter un système permettant de déplacer plusieurs personnes à la fois.

- Ajoutez un ensemble `selection` et une méthode `selectionner(Personne)` à la classe `Simulateur`. La méthode de déplacement perdra son argument, puisqu'elle déplacera les personnes sélectionnées.
- Ajoutez également à la classe `Personne` les méthodes `selectionner()` et `deselectionner()`.

Question 2 • Donnez le code Java de la méthode `selectionner(Personne)` de `Simulateur`, sachant que l'interface `Set` (ensemble) de Java propose les méthodes `boolean contains(Object)`, `void add(Object)` et `boolean remove(Object)`.

Question 3 • Donnez le code Java de la méthode `deplacer()`, sachant que l'interface `Set` étend l'interface `Iterable`, et permet donc l'usage du `foreach`.

1.1.1 Les familles

On se rend compte à l'usage que la plupart des personnes se déplacent en petit groupe : les familles. On va donc modifier le simulateur en conséquence.

— Ajoutez une classe `Famille`, caractérisée par un ensemble de `Personne` et disposant des méthodes `ajouter(Personne)` et `getMembres() : Set<Personne>`. On n'utilisera pas le pattern composite dans un premier temps.

Question 4 • Complétez le diagramme en conséquence.

Question 5 • Donnez le code de la méthode `deplacer()` de la classe **Simulateur**.

1.1.2 Les groupes généralisés

On veut maintenant pouvoir gérer des groupes de personnes et de familles. On conçoit rapidement que les familles sont des groupes "simples", composés uniquement de personnes. On replacera donc la classe `Famille` par la classe `Groupe`, plus générale.

Question 6 • Reprenez le diagramme existant en utilisant le pattern Composite.

Question 7 • Donnez le code de la méthode `deplacer()` de la classe `Simulateur`.

Question 8 • Donnez le code de la méthode `deplacer()` de la classe `Groupe`.

1.2 Le patron Proxy – La suite de Fibonacci

Vous allez maintenant travailler sur un projet différent, et donc commencer un nouveau diagramme UML.

On se propose d'implanter (bêtement) la suite de Fibonacci. Rappelons la définition mathématique de celle-ci :

$$\begin{cases} fib(0) = 1 \\ fib(1) = 1 \\ fib(n) = fib(n-1) + fib(n-2) \quad \forall n > 1 \end{cases}$$

— Proposez une classe Java permettant de calculer la valeur d'un terme de la suite. Cette classe définira une méthode unique `calcul(int n) : long`.

Question 9 • Donnez le code de la fonction `calcul`.

Question 10 • Tracer l'exécution, sur papier, de `calcul(5)`. Combien d'appels sont nécessaires ? Combien de fois `calcul(2)` est-il appelé ?

1.2.1 Le Proxy-Cache

Afin d'éviter le recalcul des termes de la suite de Fibonacci, vous allez mettre en place un proxy-cache.

— Proposez une classe `ProxyFibonacci` qui serve de proxy à la classe de calcul conçue précédemment.

Question 11 • Donnez le code de la fonction `calcul`.

2 Partie Java

L'objectif de cette partie est d'implémenter et de tester les différents diagrammes que vous avez réalisés en TD. Comme toujours, n'attendez pas d'avoir tout écrit pour tester !

2.1 Le simulateur de déplacements

Vous trouverez dans votre dépôt local l'interface javaFX `GUI.fxml`, les classes `Simulateur` et `Controleur`, et un fichier texte.
`Simulateur` s'occupe de charger et d'afficher l'interface graphique, laquelle repose sur `Controleur` pour ses événements.
Le fichier `Entite.txt` contient des fragments de code concernant divers objets et effets graphiques. Utilisez tout ou partie de ces fragments pour vous aider.

Question 12 • Implémentez le simulateur conçu en TD. L'interface graphique n'est pas nécessaire au pattern.

2.2 La suite de Fibonacci

Question 13 • Implémentez la classe `Fibonacci` sans utiliser le proxy.

Question 14 • Testez la classe en calculant et en affichant successivement les termes de 0 à 45. Vous pouvez bien sûr réduire la borne supérieure si votre machine prend trop de temps à calculer *(La machine de l'enseignant prend 15 secondes de 0 à 45).*
Mesurez le temps de calcul en utilisant `System.currentTimeMillis()` qui retourne le nombre de millisecondes écoulées depuis EPOCH (01/01/1970 à 00H00).
Mesurez le temps de calcul nécessaire pour afficher les termes de 45 à 0 (en ordre inverse).

Question 15 • Implémentez enfin le proxy et renouvelez les tests et les mesures de temps. Donnez vos conclusions.