Composite Pattern

1 Les Composites - Le simulateur de personnes

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

1.1 Partie UML

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.

- Définir 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.
- Définir la classe `Simulateur` qui contient une liste de `Personne` ainsi que les méthodes `ajouter(Personne)`, `deplacer(Personne)` et `afficher()`.

Question 1 • Créer 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.
  Ajouter un ensemble `selection` et les méthodes `onClick(Personne)` à la classe `Simulateur`.
  La méthode de déplacement perdra son argument, puisqu'elle déplacera les personnes sélectionnées.
  Ajouter également à la classe `Personne` les méthodes `couleurSelection()` et `couleurNormal()` qui visualiseront la sélection ou non.

Question 2 • Donnez le code Java de la méthode `onClick(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.

— Ajouter 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 • Compléter le diagramme en conséquence.

Question 5 • Donner 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 • Réusiner le diagramme existant en utilisant le pattern Composite.

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

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

1.2 Partie Java

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

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. Convertissez le en fichier Java et utilisez tout ou partie de ces fragments pour vous aider.

Question 9 • Implantez le simulateur conçu en TD. L'interface graphique n'est pas nécessaire au pattern.