TP4 LOG2410

16/11/2019

Patron Composite

- **Réponses aux questions :**

1 - a) L'intention du patron Composite :

Selon les notes du cours L'intention du patron Composite est « Traiter les objets individuels et les objets multiples, composés récursivement, de façon uniforme. »

En d'autres termes un patron composite permet de gérer des objets ayant des fonctions similaires de la même manière en ignorant les différences entre les classes. Ce patron augmente l'uniformité et la flexibilité du code.

b) la structure des classes réelles qui participent au patron ainsi que leurs rôles :

- **Objet (**Component : l'interface uniforme) **: AbsTeamCompenent**
  - déclare l'interface pour la composition d'objets
  - met en œuvre le comportement par défaut
- **Objet Simple** (feuilles : pour les objets qui n'ont pas des enfants) **: TeamMembre, TeamManager,**
  - représente les objets manipulés, ayant une interface commune
- **Objet Composite :Team**
  - définit un comportement pour les composants ayant des enfants
  - stocke les composants enfants : définit un container (`TeamComponentContainer m_members`)
  - met en œuvre la gestion des composants enfants

- Voir DiagrammeDeClasses_Composite.pdf

Patron Decorator

- **Réponses aux questions :**

1 - a) L'intention du patron Decorator :
Selon le cours « Attacher dynamiquement des responsabilités additionnelles à un objet. Le patron Décorateur fournit une alternative flexible à la dérivation pour étendre la fonctionnalité d'une classe » Autrement dit, c'est un patron qui permet d'ajouter de nouvelles fonctionnalités à un objet existant sans modifier sa structure et ce, en créant une classe de décorateur qui enveloppe la classe d'origine et fournit des fonctionnalités supplémentaires en préservant la signature des méthodes.

- **Component** : *TeamMemberRole* dérive, aussi, de la classe de base abstraite *AbsTeamComponent*, selon le patron Decorator.

- **Decorator**: *TeamMemberRole* permet d'ajouter du texte sur la photo d'un membre d'une équipe et ainsi d'afficher chaque membre avec son rôle

- **ConcreteComponent**: toutes les classes de primitives soient *TeamMember* et *TeamManager*

1 – b) La structure des classes réelles qui participent au patron ainsi que leurs rôles :

- Voir DiagrammeDeClasses_Decorator.pdf

Architecture 3 niveaux et modèle MVC

Modèle – Vue – Contrôleur (MVC)

Le modèle MVC est un patron de conception très utile en Programmation événementielle et surtout dans la conception d'interfaces graphiques. Il permet de séparer les tâches en plusieurs classes pour diminuer la complexité du programme et permettre la réutilisabilité du code. Il consiste à séparer les données (modèle), de la présentation (vue), et des traitements (contrôleur).

La Vue est l'interface avec laquelle l'utilisateur interagit, elle permet de signaler au modèle qu'elle veut être avertie lors de modifications des données et d'afficher les données de façon graphique dépendamment des actions de l'utilisateur qui seront traitées par le contrôleur. Dans notre cas c'est la classe TeamViewer et la classe TeamComponentView qui s'occupent de la vue.

Le Modèle sert à maintenir l'état des données et fournir des méthodes d'accès et de modification (getters/setters) pour celles-ci. Elle permet aussi de signaler aux vues que l'état de

ses attributs a été modifié et de gérer les évènements qui ne proviennent pas de l'interface graphique.
Le Contrôleur sert à gérer les évènements et interpréter les actions de l'utilisateur en un changement sur le Modèle ou sur sa Vue. Si c'est un changement esthétique il sera appliqué sur la vue et si c'est une modification de données ça se fait dans le modèle.
Le Contrôleur ne fait aucun traitement. Il traduit juste les actions en modifications.

Dans ce travail, on utilise QT pour l'implémentation de des patrons de conception. D'après ce que nous avons appris dans le cours de cours de programmation orientée objet INF1010, la bibliothèque Qt n'utilise pas vraiment le l'architecture à trois niveaux MVC, mais il utilise plutôt une version simplifiée : l'architecture Modèle-Vue.
En fait, dans ce modèle, le Contrôleur est intégré à la Vue. On diminue ainsi un peu la complexité de MVC, tout en gardant les données séparées de l'affichage.

En effet, les classes qui appartient au (Modèle - Contrôleur fusionnés) sont :
Team, TeamMembre, TeamMembreRole, TeamManager, SingleImage, AbsTeamCompenent

- Voir DiagrammeArchitectural.pdf