Etudes de patrons de conception

Exercice 1 : le singleton

Il existe des situations ou l'on ne veut garantir qu'une instance de classe ne pourra être crée. C'est par exemple le cas d'une classe d'authentification ou de logging (écriture dans un fichier journal). L'instance d'une classe dont on doit garantir l'unicité s'appelle un singleton.

1. Envisager toutes les solutions possibles pour garantir l'unicité d'une instance de classe. Comparer chaque solution proposée en termes d'avantages et d'inconvénients.

Le modèle de conception du singleton repose sur deux éléments essentiels : l'utilisation de méthodes statiques et la définition d'un constructeur privé.

2. Proposer un diagramme de classe qui mette en œuvre ce modèle dans le cas d'une classe de logging.

Exercice 2 : Observation et notification

Dans la programmation orientée-objet, il est utile de pouvoir être tenu informé d'un changement survenu dans une autre partie d'un programme. En d'autres termes, cela signifie que des objets doivent pouvoir observer d'autres objets et être tenus informés des changements qui surviennent sur les objets qu'ils observent. Les interfaces graphiques sont un très bon exemple de ce principe d'observation.

1. Proposer différentes solutions qui permettent d'observer des objets. Pour ce faire, on considèrera le cas suivant : dans un simulateur de trafic routier, les véhicules (voitures, camions, …) peuvent écouter une radio info-trafic. En cas d'accident, cette dernière notifie tous ses auditeurs de la présence d'un accident en interrompant le programme en cours.

Dans le modèle de conception observer/observable, la classe Observable permet d'indiquer qu'un objet qui en hérite peut être observé par d'autres objets. Les observateurs quant à eux sont des objets qui implémentent l'interface Observer.

Etant données cette classe et cette interface, le fonctionnement d'observation est le suivant. La classe Observable permet d'ajouter et supprimer un observateur (méthodes `addObserver` et `deleteObserver`). Elle permet aussi de notifier ses observateurs (méthode `notifyObservers`). Pour un observateur, la notification d'un changement sera faire par le biais d'un appel à sa méthode `update`.

2. Proposer un diagramme de classes qui illustre le fonctionnement de ce modèle dans le cas du simulateur routier.
3. A l'aide d'un diagramme de séquences, montrer comment sont effectuées les notifications.

Exercice 3 : Les objets composites

Le modèle de conception composite a pour objectif de permettre au client d'une classe de manipuler de manière identique un objet simple et une collection d'objet. Pour illustrer ce patron, nous prenons l'exemple d'un logiciel de dessin vectoriel qui manipule des figures. Une figure peut être ici une forme de base (carré, cercle, …) ou bien une composition de figures, qui peuvent elles mêmes être des formes de base ou des compositions de figures, etc.

1. Proposer différentes solutions de mise en œuvre du patron composite. Les comparer.

Pour implanter ce patron, il suffit de voir les objets simples et les objets composites de la même manière. Pour cela on utilise une interface dédiée qui contient les opérations qui devront pouvoir s'appliquer sur un objet simple et sur un ensemble d'objets composés.

2. Proposer un diagramme de classes qui respecte ce patron de conception. La seule opération spécifiée par l'interface sera l'affichage de la figure.