TD2

Rendu : une archive avec le code source en Java et un document au format PDF.

1 Cours : révision et complément

Questions 1. Dans le cadre de quel patron un rappeur a-t-il été cité dans le cours ?

Question 2. Quel était le parallèle fait pour illuster ce patron ?

Question 3. Quel patron apparaît deux fois dans la classification GoF ? Pourquoi ?

Question 4. Le patron modèle-vue-contrôleur (MVC) permet d'organiser les interfaces graphiques. Se renseigner à son sujet. Quels sont les trois patrons qu'il met en œuvre ?

Question 5. Quel rôle joue chaque module du patron MVC ?

2 Système de surveillance

Nous voulons concevoir un système de sécurité qui possède des détecteurs de fumée et des détecteurs de présence. Ces familles d'objets peuvent être conçues par différentes entreprises. Si de la fumée ou du mouvement est détecté, un SMS est envoyé au propriétaire. De plus, si de la fumée est détectée l'installation fixe d'extinction automatique à eau se déclenche et si du mouvement est détecté l'alarme s'allume. Ces installations peuvent être arrêter manuellement. Par ailleurs, il y a une journalisation de la détection de fumée ou de mouvement.

2.1 Le patron Singleton

Question 1. Ecrire une classe `Journal` qui implémente le patron Singleton. Elle a une méthode `Journal getInstance()` retournant l'instance. Elle possède également une méthode `void ecrire(String str)` qui écrit dans un fichier `domotique.log` la date et la chaîne de caractère passée en paramètre.
Exemple de contenu du fichier `domotique.log` (le format de la date peut être diférent) :

> Tue Apr 05 08:41:20 CEST 2022
> Ma chaîne de caractère
> Tue Apr 05 08:43:40 CEST 2022
> Ma chaîne de caractère 2

2.2 Trois classes quelconques

Question 2. Ecrire une classe `AlarmeMouvement` ayant des méthodes `void demarrer()` et `void arreter()` affichant respectivement "Alarme démarée..." et "L'alarme a été arrêtée.".

Question 3. De manière similaire, écrire une classe `ExtincteurAutomatique` ayant des méthodes `void ouvrir()` et `void fermer()` affichant respectivement "L'extincteur automatique est ouvert." et "L'extincteur automatique est fermé.".

Question 4. Ecrire une classe `Sms` avec une méthode `void envoyerMessage(String str)` qui renvoie "SMS :" suivi de la chaîne de caractère.

2.3 Le patron Commande

Question 5. Créer une interface `Commande` avec la méthode `void executer()`.

Question 6. Créer les quatre commandes qui implémentent l'interface `Commande` :

- `CommandeAllumerAlarme`,
- `CommandeEteindreAlarme`,
- `CommandeAllumerExtincteur` et
- `CommandeEteindreExtincteur`.

Les deux premières classes prennent une instance de la classe `AlarmeMouvement` comme attribut, les deux suivantes prennent une instance de la classe `ExtincteurAutomatique` comme attribut. Lors de l'appel de la méthode `void executer()`, les quatre classes affichent respectivement

- "Démarrage de l'alarme.",
- "Arrêt de l'alarme.",
- "Allumage de l'extincteur automatique." et
- "Arrêt de l'extincteur automatique.".

La méthode `void executer()` affiche le message et réalise l'action correspondante.

Question 7. Créer une classe représentée par le diagramme suivant. Elle représente une télécommande avec deux boutons marche/arrêt (les attributs sont initialisés par le constructeur). La méthode `void setCommande(...)` permet d'affecter des commandes à un bouton.

TelesurveillanceMaison
- commandesMarche : Commande[] - commandesArret : Commande[]
+ setCommande(int i, Commande comMarche, Commande comArret) + appuyerMarche(int i) + appuyerArret(int i)

2.4 Le patron Fabrique abstraite

Question 8. Implémenter une Fabrique abstraite pour créer les détecteurs. Nous avons deux fabriques `FabriqueA` et `FabriqueB` qui implémentent l'interface `FabriqueDetecteur`, leur imposant d'avoir les méthodes

- `DetecteurFumee creerDetecteurFumee(Commande cmd)` et
- `DetecteurMouvement creerDetecteurMouvement(Commande cmd)`.

Chacune crée ses détecteurs : `ADetecteurFumee`, `ADetecteurMouvement`, `BDetecteurFumee` et `BDetecteurMouvement`.

Question 9. `ADetecteurMouvement` et `BDetecteurMouvement` héritent de la classe abstraite `DetecteurMouvement` dont le diagramme de classe est le suivant.

DetecteurMouvement
commande : Commande
alerter()

La méthode `alerter()` crée un SMS et envoye le message "Mouvement detecté !", exécute la commande qui lui a été attribuée et écrit "Mouvement" dans le journal.

Question 10. Ecrire les classes `ADetecteurMouvement` et `BDetecteurMouvement`. Elles redéfinissent la méthode `alerter()` : elles indiquent quelle fabrique crée leurs objets, puis appellent la fonction de leur classe mère.

Question 11. Reprendre la question 9. avec le détecteur de fumée.

DetecteurFumee
commande : Commande
eteindreFeu()

Cette fois les messages sont "Fumée detectée !"et "Fumée".

Question 12. Reprendre la question 10. pour le cas du détecteur de fumée.

2.5 Le patron Observateur

Question 13. Créer une interface `Observateur` imposant d'implémenter `detecter(boolean feu, boolean individu)`. S'il y a un intru, l'alarme devra sonner. S'il y a du feu, l'extincteur devra être déclenché.

Question 14. Les classes abstraites `DetecteurMouvement` et `DetecteurFumee` implémentent maintenant l'interface `Observateur`. Effectuer les modifications appropriées. Noter que seul l'un des deux états reçus en paramètre est utile pour chaque classe.

Question 15. Ecrire une classe `Maison` représentant le sujet observé. La modification de ses attributs notifie tous ses observateurs et transmet la valeur de ses états.

Maison
- observateurs : List<Observateur> - feu : boolean - individu : boolean
+ enregister(Observateur observateur) + setFeu(boolean b) + setIndividu(boolean b) + notifier()

Question 16. Télécharger et exécuter le main. Le message du détecteur de fumée apparaît deux fois sur la sortie standard et dans le fichier `domotique.log`. Pourquoi ?

2.6 Vue d'ensemble

Question 17. Réaliser un diagramme de classes simplifié avec l'ensemble de classes / interfaces. Peuvent n'apparaître que les attributs et méthodes suffisants à la compréhension du fonctionnnement du système de surveillance.