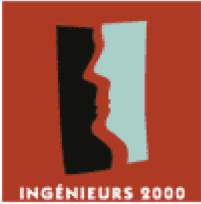


UNIVERSITÉ PARIS-EST

Design Pattern – TD n°4

LE PATTERN SINGLETON : Y A-T-IL UN CONTRÔLEUR DANS LA TOUR ?

Téléchargez le code ici. Il s'agit d'un exercice simple : des avions sont créés par le programme principal. L'aéroport possède une tour de contrôle, qui gère l'unique piste. Quand un avion souhaite décoller ou atterrir, il attend que la tour de contrôle le lui permette. Deux avions ***ne doivent pas*** *réserver la piste en même temps, sous peine d'un accident.*

La classe LoadAndShow ne vous intéresse pas, pas besoin de la regarder.

1. Si vous démarrez le *main*, quel résultat obtenez-vous ? Que fait la fonction *start* quand elle est appelée sur un objet *Plane* ? Pouvez-vous expliquer ce qui s'est mal passé ? Proposez des idées pour y remédier sans toucher aux classes *Airport*, *Plane* et *Main.*

2. Implémentez la solution retenue (en en ayant discuté avec l'intervenant de TP). Est-ce que votre programme fonction mieux ?

3. Si c'est le cas, bravo ! Téléchargez ces classes (écrasez l'ancien *Main* avec celui-ci) et testez votre code.

4. Dans la classe *ControlTower*, utilisez la fonction *Thread.sleep(1)* avant tout appel à la fonction *new*. Votre code fonctionne toujours ? Si oui, faîtes un *Thread.sleep* plus long. Si tel est le cas, Pourquoi votre code ne fonctionne plus au bout d'un moment ? Avez-vous des idées pour rendre votre code fonctionnel ?

5. Après en avoir discuté avec l'intervenant de TP, implémentez votre solution. Tout fonctionne-t-il ?

6. Modifiez la classe *Airport* afin que ses fonctions ne soient plus statiques (*Airport* devra aussi devenir un singleton). Modifiez en conséquent *ControlTower*. Votre programme fonctionne toujours ?

LE PATTERN DÉCORATEUR : RENÉ LA TAUPE

Votre bon travail à la tour de contrôle de Brétigny-sur-Orge vous a valu une promotion : vous venez d'être embauché au club Jamba, dans le département des « applications » mobiles (aussi appelé département René la taupe).

1. Récupérez l'archive zip ici, et regardez un peu le code (la classe *MyImage* ne vous intéresse pas). Dans le main, activez la partie pour placer un chapeau sur René.

2. Récupérez les lunettes de soleil ici. Sachant qu'elles doivent être positionnées aux coordonnées (255,76), écrivez le code permettant d'afficher des lunettes sur René.

On souhaite maintenant produire différents logos de René, des fois avec un chapeau, ou avec des lunettes. Ces logos seront produits à la volée quand un SMS est envoyé au bon numéro. Ce que l'on veut, c'est un mécanisme efficace pour afficher différents logos de René facilement (les SMS ne sont pas de votre ressort).

3. Tout d'abord, récupérez d'autres éléments ici : le bâton de sucrerie se place à la position (441,202), la chanson se place à la coordonnée (10,10), le smiley à la coordonnée (260,210).

Écrivez un code permettant de générer facilement un René avec lunettes, un René avec chapeau, … Évidemment, il faudra créer une interface commune entre tous les accessoires que René peut porter. Il faudrait que les autres équipes du projet puissent générer ces nouveaux logos en une seule ligne de code.

L'équipe marketing a eu une super idée... Pour rentabiliser au maximum René la taupe, faisons des logos combinant ces éléments ! René pourra ainsi avoir des lunettes et un chapeau puis chanter, ou une sucrerie et un smiley, …

4. C'est parti ! Essayez de modifier votre code afin de combiner facilement ces éléments. L'idée ici est de proposer une génération très simple d'un nouveau logo, en une ligne de code, aux autre équipes. Essayez de voir si vous ne pouvez pas regrouper tout le monde (accessoires et rené) sous une même bannière (une interface que l'on pourrait appeler *SuperLogo*).

Ensuite, regroupez les accessoires sous une autre bannière (du genre, une classe abstraite *Accessoires*). Enfin, dîtes qu'*Accessoires* devra contenir un élément de type *SuperLogo*.

Voyez si vous ne pouvez pas déclencher, entre les SuperLogos, des réactions en chaine pour superposer facilement tout le monde.

5. Implémentez aussi une fonction prix qui permet de calculer le prix de tout un ensemble de logos (toujours à partir de réactions en chaine). Chaque accessoire a un prix, que l'on ajoute au prix de René.

6. Avec la grève, tout le monde veut une moustache comme José Bové. Trouvez-en une sur Internet, et ajoutez la à vos accessoires. Était-ce difficile ?

7. En Creuse, on vient de découvrir la mode « Crazy Frog ». Le club Jamba souhaite s'ouvrir à ce nouveau segment marketing, ne proposant non pas des logos René la Taupe, mais des logos Crazy Frog (image à récupérer ici). Pouvez-vous facilement ajouter une nouvelle image de base « Crazy Frog » à votre code ?

LE PATTERN STATE

Le Pattern State vous permet de créer une interface souple de programmation pour des machines d'état et autre. Après votre succès auprès du public creusois avec Crazy Frog, vous venez d'être embauché par la mairie de Guéret pour mettre en place une machine à la pointe de la technologie : un distributeur au-to-ma-tique de billets !

1. On souhaite créer un distributeur de billets. Ce dernier peut être vu comme une machine d'états avec trois états : pas de carte insérée, en attente d'opération, et en attente de retirer des espèces. On pourra faire quatre actions sur la machine : insérer une carte, entrer un code, retirer des espèces, retirer la carte. Évidemment, selon l'état de la machine, les actions auront différentes conséquences.

Voici un squelette du code de la machine :

```
public class Distributeur
{
    final static int etat_attente_carte = 0;
    final static int etat_attente_code = 1;
    final static int etat_attente_operation = 2;

    private int etat;
    private Carte c;

    public Distributeur()
    {
        etat=etat_pas_de_carte;
    }

    public inserer_carte(Carte client)
    {
        if(etat==etat_pas_de_carte)
        {
            System.out.println("Insertion carte Client");
            c=client;
        }
        else if(etat==etat_entrer_code)
        {
            ...
        }
        ...
    }


}
```

La classe *Carte* est une classe avec un constructeur qui prend un code secret, et une méthode qui répond si le code entré est bon.

Si le *Client* entre trois fois un mauvais code, sa carte doit être avalée. Codez votre *Distributeur*.

La machine semble bien marcher, mais certains client se plaignent. En effet, certaines fois, la machine ne donne pas d'argent... Rapidement, les ingénieurs de la machine comprennent que

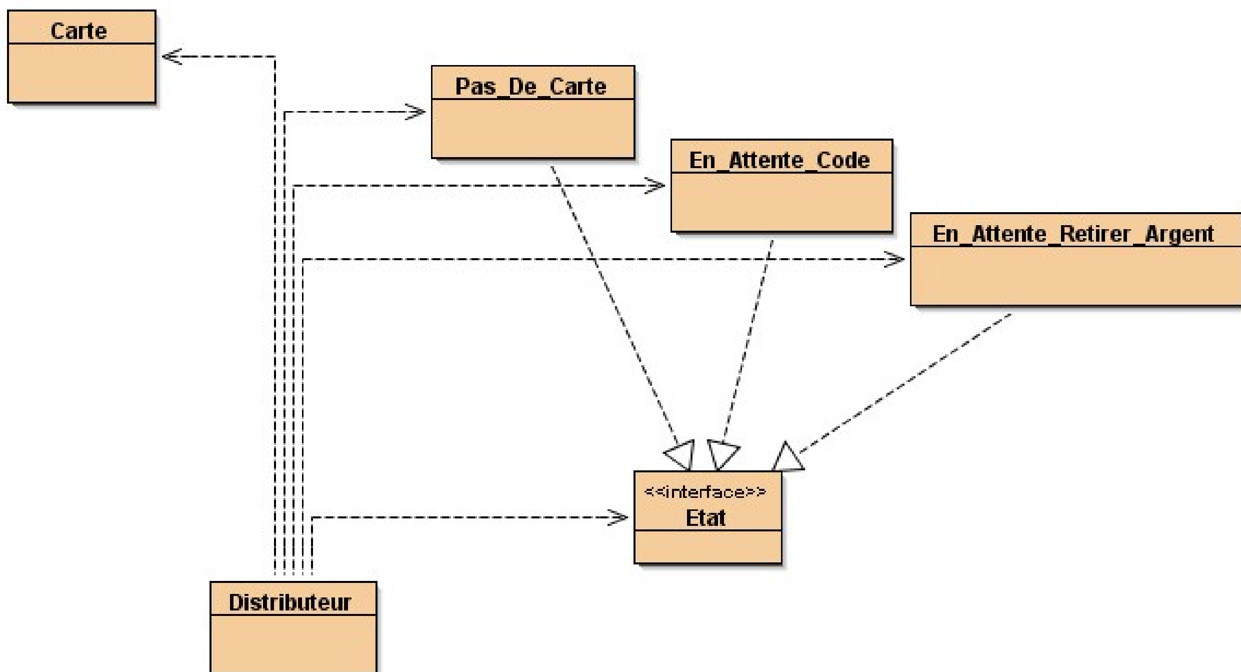
lorsque la machine n'a plus de billets, et ne peut pas donner d'argent. On doit donc rajouter un état à la machine, qui sera l'état vide. Cependant, vous voyez bien que rajouter un état change beaucoup de code et n'est donc pas très pratique.

Nous allons donc, avant de rajouter l'état vide, modifier les classes et organiser mieux le projet.

2. Créez donc une interface (ou pourquoi pas une classe abstraite...) *Etat* état qui contiendra les quatre méthodes de la machine (insérer la carte, entrer code, retirer argent, reprendre carte). Créer trois classes *Etat*, une par état possible de la machine et écrivez-y du code.

Le *Distributeur* devra posséder trois variables de classes (une par chaque type d'état possible), plus une variable de classe de type *Etat* qui sera son état courant. A chaque appel d'une méthode (insérer carte, ...), il devra appeler la méthode sur son état courant. Ce sont les *Etat* qui décideront de la marche à suivre. Cependant, le distributeur doit posséder une méthode permettant de modifier son état courant.

Pour vous aider, voici le diagramme de classe que vous devriez obtenir.



Testez avec un *Client* que tout fonctionne.

Vous venez de mettre en place le Pattern *State*. L'objet n'est pas ses états, mais possède ses états.

3. Ajoutez un état *MachineVide* pour empêcher des retirer des l'argent quand la machine est vide. Vous allez donc ajoutez au *Distributeur* une variable de classe permettant de connaître son stock d'argent, et vous vérifierez, lorsque vous donnerez de l'argent à un *Client*, qu'il y a assez d'espèces dans le *Distributeur*. Était-ce compliqué ?

4. Ajoutez aussi une méthode *lire_solde_compte*, qui permettra au client d'accéder à son solde lorsque le *Distributeur* est en mode *En_Attente_Pour_Retirer_Argent*. Était-ce compliqué ?