Amphi 4 – Design Patterns (patrons de conception)

Design patterns : patrons / motifs récurrents d'architecture de programmes orientés objet.

Patterns des motifs qu'on utilise pour la programmation objet dans la conception/objet.

Les design patterns sont des collections / ensemble de types de classes.

Il y a des designs patterns standards, c'est un vocabulaire commun pour les programmeurs.
Comment résoudre les problèmes dans les cas différents ?
Comment on l'utilise, etc…

En général, c'est une bonne chose de savoir les utiliser.

1 – Pattern Factory

Le design pattern Fabrique (Factory Method) définit une interface pour la création d'un objet en déléguant à ses sous-classes le choix des classes à instancier.

Méthodes statiques qui créent des objets

- Comme des constructeurs mais n'utilisent pas la classe spécifique

Constructeur(C) ---> objet de type C

Méthode static créer() {

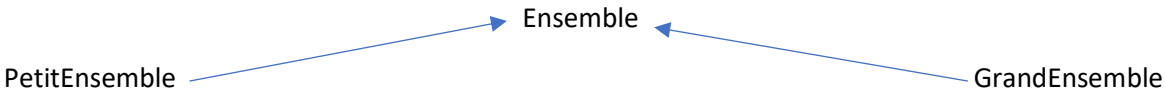
//Pas besoin de connaitre le nom, l'ID de l'objet envoyé

new (c) ;

}

C implements T ;

La représentation interne d'un ensemble, ses différentes représentations ont des plus et des moins, il y a des types plus performants pour des types petit et d'autres pour des types grands.



Liste d'entiers Table de hachage

Si on ne connait même pas, on doit être sûr que l'interface ensemble soit envoyé.

- Découpe le code
- Objets factory parfois utiles pour encore plus découpler, en particulier s'il y a plusieurs implémentations/versions.

Un objet ou une classe qui crée des objets d'un certain type, on peut appeler méthode creerEnsemble.

2 – Pattern Singleton

C'est un ensemble avec un seul élément.

On veut exactement une instance d'une certaine classe d'où le « single » de « singleton »

Raisons possibles : performance, responsabilité unique centralisée, constructeurs doivent être privés.

Exemple : un endroit où il y a toutes les impressions, on ne peut pas mettre le constructeur en public

L'objet est trop grand ou en est contrôleur qui surveille les usines, on veut un seul contrôleur unique qui s'occupe. Etre sûr qu'il y est au plus une instance, il faut que les instances soient privées.

On met les constructeurs en privée, pour garantir qu'il y a une seule instance, on met une référence, on va retourner l'instance à chaque fois que quelqu'un veut accéder à cette instance.

Quel est l'intérêt d'un singleton à une classe statique ? Il y a des librairies qui doivent des objets par droit de connexion réseau, on peut beaucoup faire avec les singletons.

On implémente le singleton en écrivant une classe contenant une méthode qui crée une instance uniquement s'il n'en existe pas encore. Sinon elle renvoie une référence vers l'objet qui existe déjà. Dans beaucoup de langages de type objet, il faudra veiller à ce que le constructeur de la classe soit privé, afin de s'assurer que la classe ne puisse être instanciée autrement que par la méthode de création contrôlée.

Le but du singleton : Si on est dans un cas où on veut une seule instance.

Centraliser la responsabilité de quelque chose à un endroit unique.

3 – Pattern State

Si la représentation interne de l'état de l'objet peut changer dynamiquement

- Changer à une représentation plus performante (exemple : vector vs table de hachage)
- Sur type pour les différentes représentations internes, caché à l'extérieur

C'est un design pattern où nous pouvons changer la représentation interne de l'objet.

On ne peut pas changer la classe de « this »

La classe « Ensemble » a comme champ privé :

Si on reçoit une demande d'insertion, on peut voir si l'ensemble d'état est petit ou grand.

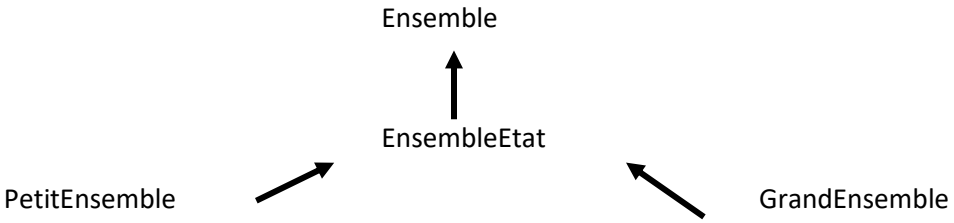
Ça va implémenter les opérations dans les sous-classes de « EnsembleEtat ».

Si on crée un nouvel ensemble, on va initialiser un ensemble vide, si on dépasse le seuil de min,

Le champ EnsembleEtat va convertir petitEnsemble en grandEnsemble.

Ils vont hériter d'EnsembleEtat.

La technique du patron de conception (design pattern en anglais), ou encore modèle de conception, comportemental état utilisé en génie logiciel est utilisé entre autres lorsqu'il est souhaité pouvoir changer le comportement d'un objet sans pour autant en changer l'instance. Ce type de comportement généralise les automates à états qui sont souvent utilisés comme intelligences artificielles simples.

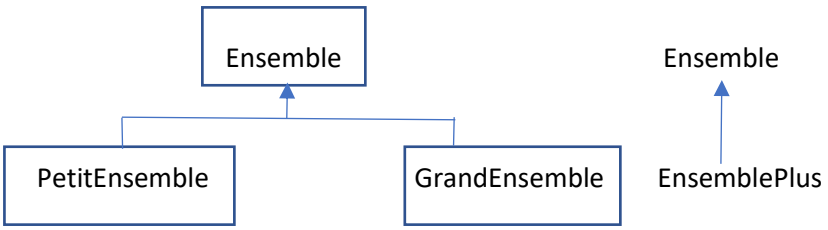


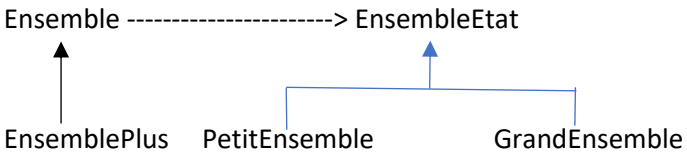
4 – Pattern Bridge

Si on a hiérarchies qui se contredisent, on va les séparer et introduire un bridge.

Le pont est un patron de conception de la famille structuration, qui permet de découpler l'interface d'une classe et son implémentation.

La partie concrète (implémentation réelle) peut alors varier, indépendamment de celle abstraite (définition virtuelle), tant qu'elle respecte le contrat de réécriture associé qui les lie (obligation de se conformer aux signatures des fonctions/méthodes, et de leurs fournir un corps physique d'implémentation).





((((0+1)+2)+3)+4)+5 = Somme

5 – Strategy and command

Le type d'une fonction doit renvoyer quelque chose.

(new somme (3,5)).LaFonction() threads : Runable run() ;

Le patron de conception stratégie est utile pour des situations où il est nécessaire de permuter dynamiquement les algorithmes utilisés dans une application. Le patron stratégie est prévu pour fournir le moyen de définir une famille d'algorithmes, encapsuler chacun d'eux en tant qu'objet, et les rendre interchangeables. Ce patron laisse les algorithmes changer indépendamment des clients qui les emploient.

6 – Observer

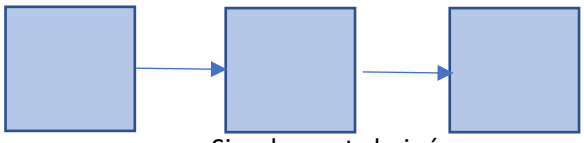
Le patron de conception observateur est utilisé en programmation pour envoyer un signal à des modules qui jouent le rôle d'observateurs. En cas de notification, les observateurs effectuent alors l'action adéquate en fonction des informations qui parviennent depuis les modules qu'ils observent (les « observables »).

Collections : il y en a plusieurs en JAVA.

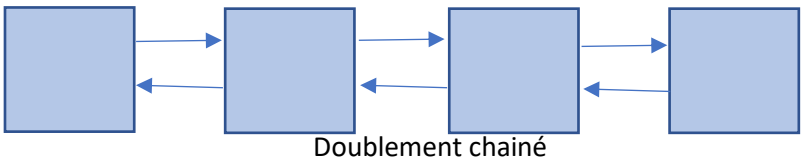
Tableaux : tab[3]

Vecteur : objet qui contient un tab à l'intérieur on peut modifier le tab comme le design pattern.

LinkedList <E> :



Simplement chainé



Doublement chainé