Prog logiciel : Réutilisation et frameworks

Définitions :

Extensibilité : capacité à se voir ajouter de nouvelles fonctionnalités pour de nouveaux contextes d'utilisation.

Adaptabilité : capacité à voir ses fonctionnalités adaptées à de nouveaux contextes d'utilisation.

Entité générique : entité apte à être utilisée dans, ou adaptée à, différents contextes.

Variabilité : néologisme dénotant la façon dont un système est susceptible de fournir des fonctionnalités pouvant varier dans le respect de ses spéciations.

Paramètre : nom dénotant un élément variable d'un concept ou d'un calcul.

Procédés élémentaires – abstraction, application, composition :

Fonction : nommage (abstrait) d'une composition d'opérations, permettant sa réutilisation sans recopie.

Procédure (abstraction procédurale) : nomme (abstrait) une suite d'instructions, permettant sa réutilisation sans recopie.

Fonction ou Procédure avec Paramètres : abstrait une composition d'opérations ou une suite d'instructions des valeurs de ses paramètres. → void carre(int x){return x*x;}

Application : Application d'une fonction ou d'une procédure à des arguments. Liaison des paramètres formels aux paramètres actuels (arguments) puis exécution de la fonction dans l'environnement résultant. → carre(3) → 9.

Composition : Enchaînement de fonctions → racine(carre(9)) → 9.

Généralisation – Réutilisation en 2 fois 2 idées :

Décomposer en éléments : procédure, fonction, objet, aspect, composant, classe, trait, type, interface, descripteur, module …

Paramétrer : identifier, nommer et matérialiser (paramètre) ce qui peut varier dans un élément. → Dans une classe A : public A(B arg){ b=arg ;} → A est utilisable avec différentes sorte de B.

Configurer : Configurer les éléments en les instantiant et valuant leurs paramètres. → carre(5) ; carre(7) ; → J'ai configuré carre en valuant mon paramètre avec 5, 7.

Composer : Composer/Assembler/Connecter les éléments configurés.

Réutilisation par fonctions d'ordre supérieur :

Une fonction d'ordre supérieur (fonctionnelle) est une fonction qui va prendre en argument et/ou retourner une fonction en valeur. → void map(int(*f)(int), int t1[], int r1[], int taille) ; → Ici f est une fonction passée en paramètre.

Schémas basiques de réutilisation en PPO : (Programmation Par Objet)

La programmation par objets a introduit de nouveau schémas de réutilisation plus simples et intuitifs à mettre en œuvre via : Extension (description différentielle, héritage), Paramétrage (encapsulation, passage d'objets en argument et liaison dynamique).

Schémas basiques PPO – Rappels :

Envoi de message : autre nom donné à l'appel de méthode en programmation par objet.

Receveur courant : Au sein d'une méthode M, le receveur courant accessible via le mot clé *this* ou *self(python)*, est l'objet auquel a été envoyé le message ayant conduit à l'exécution de M. *this* ne peut pas varier durant l'exécution d'une méthode.

Liaison dynamique : L'appel de méthode (donc envoi de message) se distingue de l'appel de fonction car lors d'un appel de méthode, l'analyse statique du code à la compilation ne connaît pas le type du receveur, qui ne sera connu qu'a l'exécution.

Schémas basiques PPO – Description différentielle :

Définition d'une sous-classe par expression des différences (propriétés supplémentaires) structurelles et comportementales entre les objets décrits par la nouvelle classe, et ceux décrits par celle qu'elle étend.

Par exemple : Une sous-classe Point3D qui hérite de Point, à une propriété supplémentaire qui est l'axe Z.

L'intérêt est qu'il n'y a pas de modification du code existant, il y a partage d'informations contenues dans la super-classe par différentes sous-classe.

Schémas basiques PPO – Spécialisation (redéfinition) de méthode :

Si la Description différentielle permet l'ajout de nouvelles propriétés dans une sous-classe, la spécialisation/redéfinition utilise les propriétés déjà existantes, en particulier des méthodes.

Définition d'une méthode de nom M sur une sous-classe SC d'un classe C où une méthode de nom M est déjà définie.

Le Masquage est lorsque l'on fait une redéfinition de M dans une sous-classe, alors celle-ci va masquer la méthode M de la super-classe dans les instances de la sous-classe.

Affectation polymorphique, ou transtypage ascendant (« upcasting »)

En présence de polymorphisme d'inclusion, où un type peut être définit comme un sous-type d'un autre, on appelle affectation polymorphique l'affectation d'une valeur d'un type ST, sous-type de T, à une variable de type T.

Exemple : List l = new ArrayList();

Redéfinition versus surcharge

La surcharge M' d'une méthode M est une méthode de même nom externe que M mais qui n'est pas une redéfinition de M (Qui n'a pas la même signature par exemple).

La nécessité du transtyage descendant (« downcasting »)

Exemple : ((Point)o),getx() ;

Le downcasting ou transtypage descendant promet au compilo qu'une variable statiquement typée T contiendra toujours un objet de type ST (ST sous-type de T). Si la promesse n'est pas respectée, une exception « typecast » est levée à l'exécution.

Schémas basiques PPO – Spécialisation (redéfinition) partielle :

Spécialisation/Redéfinition qui fait appel à la méthode redéfinie (et donc masquée).

Pour faire cela, dans la méthode spécialisée, on utilise en Java le mot clé « super ». → super.scale(1.5) ;

Paramétrage par Spécialisation : classe et méthodes abstraites

Schéma de paramétrage d'une méthode à nouveaux besoins ou a un nouveau contexte sans modification et sans duplication de code. Adaptation via des sous-classes.

Méthode Abstraite : méthode déclarée mais non définie (corps vide).

Classe Abstraite : classe (non instanciable) déclarant des méthodes non définies.

Qu'est-ce qu'un Framework

C'est une application logicielle partielle, intégrant les connaissances d'un domaine, dédiée à la réalisation de nouvelles applications du domaine visée, dotée d'un cœur (code) générique, extensible et adaptable.

Ne pas confondre avec une bibliothèque ! La bibliothèque s'utilise, un framework s'étend ou se paramètre.

La bibliothèque fera partie du code d'une nouvelle application, le framework appelle le code de la nouvelle application.

Inversion de contrôle (Principe de Hollywood)

Le code du framework (pré-existant) invoque (callback) les parties de code représentant la nouvelle application en un certain nombres d'endroits prédéfinis nommés (points d'extensions ou point de paramétrages ou encore « Hot spot »).

Injection de dépendance

L'inversion de contrôle suppose qu'en un ensemble de points d'extension définis (et documentés), le contrôle va être passé à des extensions s'il y en a.

On indique à un framework à qui passer le contrôle en réalisant une injection de dépendance.

Une injection est une association d'une extension à un point d'extension. C'est un renseignement d'un paramétrage.

Frameworks type boite blanche « WBF » : Inversion de contrôle en Paramétrage par Spécialisation.

Frameworks type boite noir « BBF » : Inversion de contrôle en Paramétrage par composition.

Prog logiciel : Design patterns

Les design patterns permettent de répondre aux problèmes récurrents lors de la programmation PPO.

Singleton :

Problème : Faire en sorte qu'une classe ne puisse avoir qu'une seule instance.

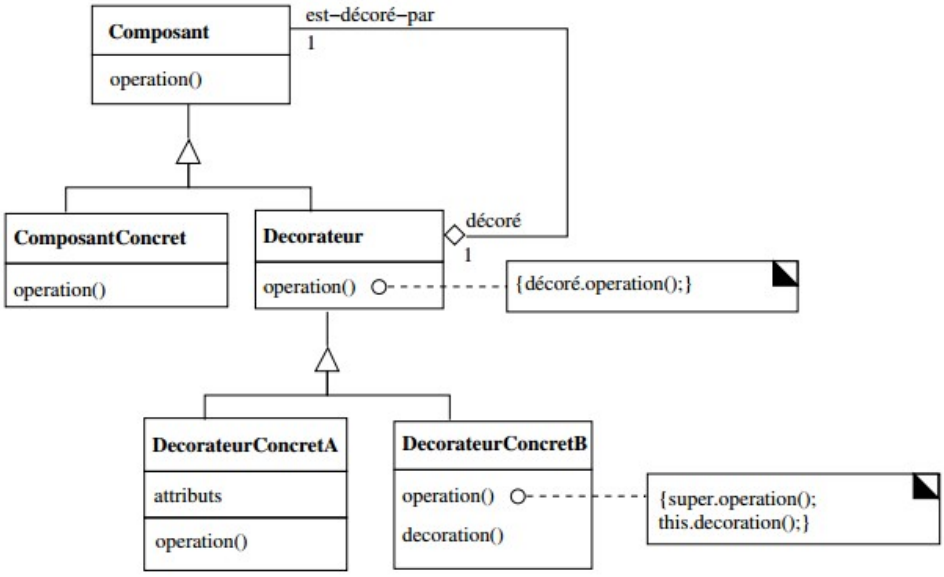
```
public class Singleton {
    private static Singleton INSTANCE = null;
    /** La présence d'un constructeur privé (ou protected) supprime
     * le constructeur public par défaut. */
    private Singleton() {}
    /** synchronized  sur la méthode de création
     * empêche toute instanciation multiple même par différents threads.
     * Retourne l'instance du singleton. */

    public synchronized static Singleton getInstance() {
    if (INSTANCE == null)
        INSTANCE = new Singleton();
    return INSTANCE;}
}
```

Décorateur (ou Wrapper) :

Problème : Ajouter ou retirer dynamiquement des fonctionnalités à un objet individuel, sans modifier sa classe.

On l'utilise beaucoup pour les objets graphiques. (Ajouter des décorations comme les bordures…)



Voir http://www.lirmm.fr/~dony/notesCours/cpatterns.pdf page 5 à 6 pour les sources.

Adapteur :

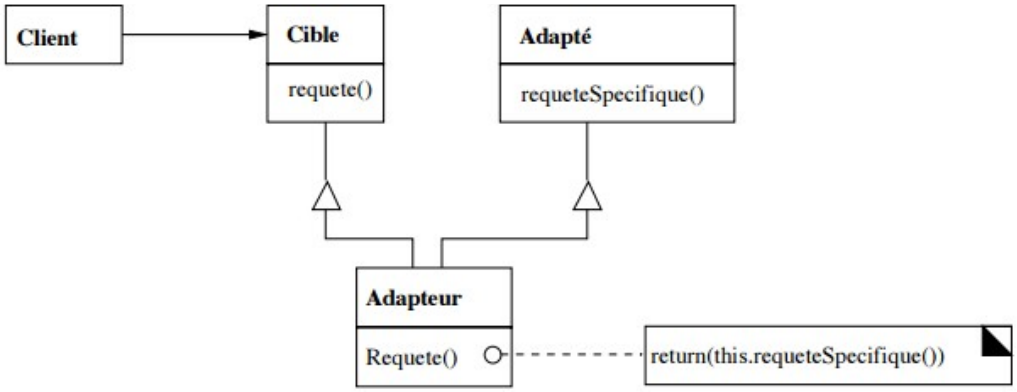
Adapter une classe dont l'interface ne correspond pas à la façon dont un client doit l'utiliser.

Cible : Objet définissant le protocole commun à tous les objets manipulés par le client (shape dans l'exemple)

Client : Objet utilisant les cibles (éditeur de dessins)

Adapté : Objet que l'on souhaite intégrer à l'application.

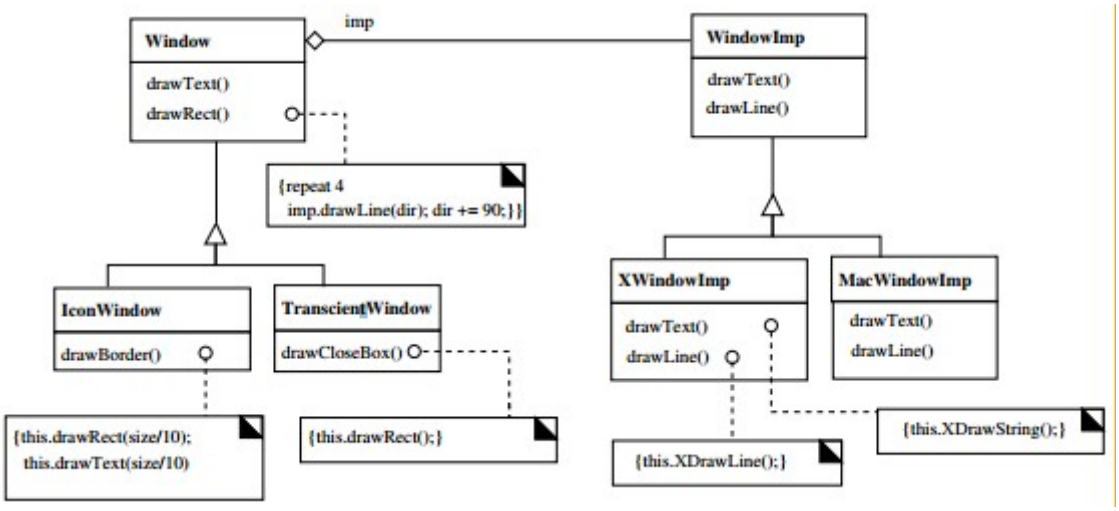
Adapteur : Objet réalisant l'intégration.



Voir http://www.lirmm.fr/~dony/notesCours/cpatterns.pdf page 10 et 11 pour les sources.

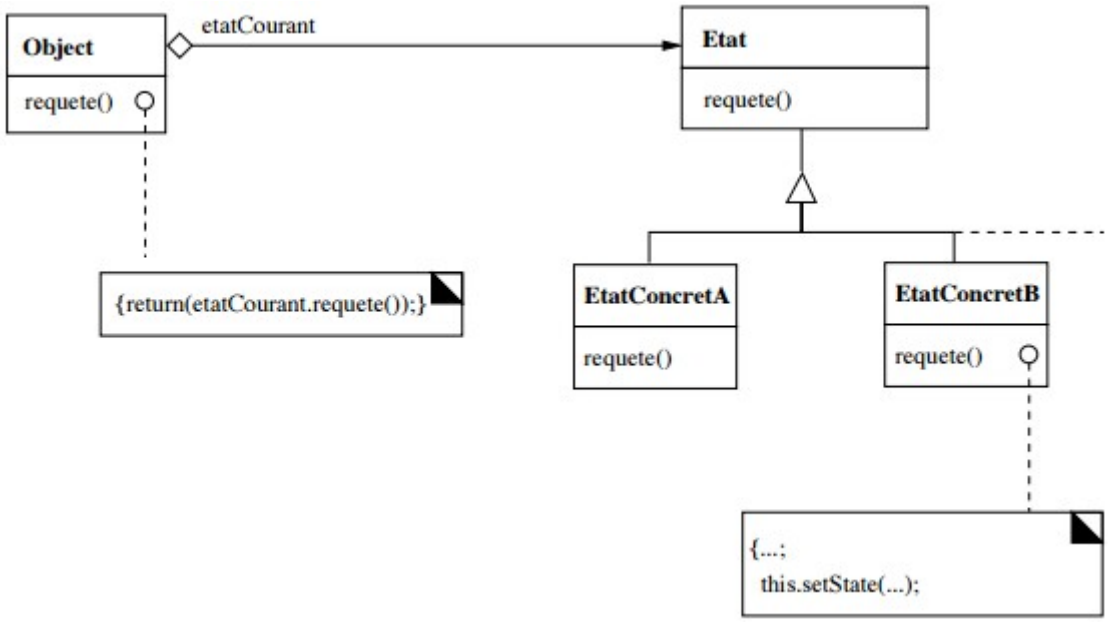
Bridge :

Problème : Découpler une hiérarchie de concept des hiérarchies réalisant ses différentes implantations. Bridge utilise l'adaptation par composition pour séparer une hiérarchie de concepts de différences hiérarchies représentant différentes implantations de ces concepts.



State :

Permet à un objet de changer de comportement quand son état interne change.

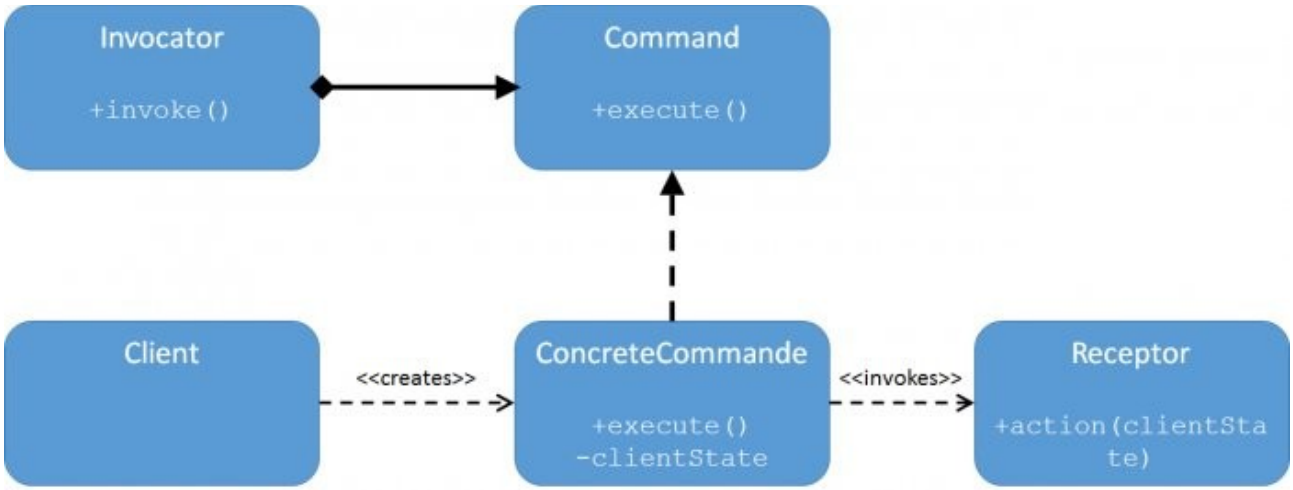


Voir http://www.lirmm.fr/~dony/notesCours/cpatterns.pdf page 13 à 15 pour les sources.

Commande (TP-5) :

Permet de garder des modules faiblement couplés : le module d'exécution a juste à connaître l'interface de Command sans se soucier de la création de nouvelles commande.

Il permet surtout d'ajouter de nouveaux types de commandes dans le module de création, sans modifier le module d'exécution.



Il est très utilisé aussi pour pouvoir faire des « Undo/Redo ».