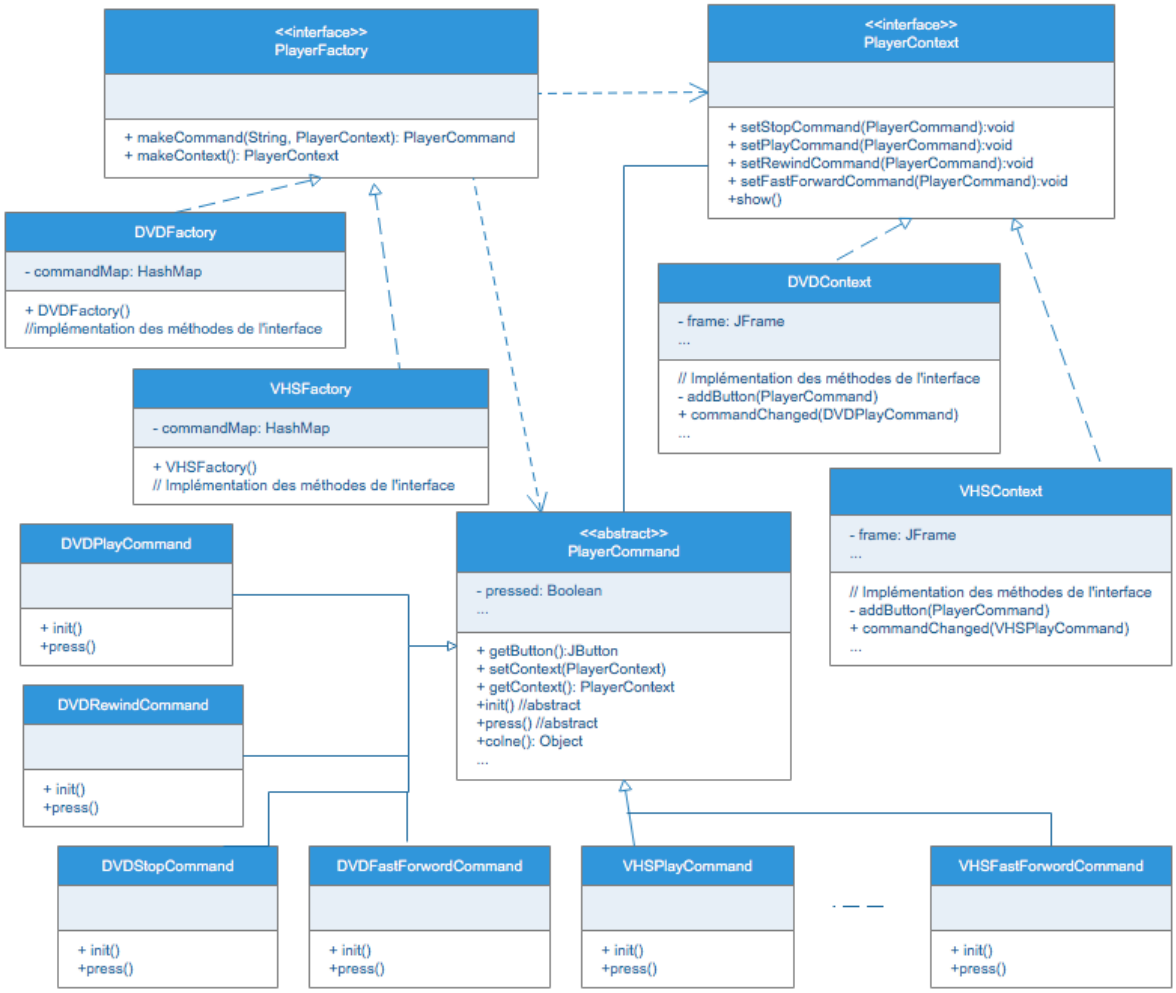
UEF 7.3. Patrons De Conception

Corrigé du Contrôle Intermédiaire

2H - Documents Interdits

Exercice 1 (6 pts):

Soit le diagramme de classes, donné ci-après.



1. Quels principes avancés de POO sont-ils appliqués dans ce diagramme ? Expliquez.

Rep : (0,5*2= 1pt) +0,5 bonus si tout est cité.

DIP, LSP, Programmer pour des interfaces : (citer l'un d'entre eux suffit)

- **LSP:** Les sous-types doivent sont des substituts pour leur ancêtres. On le voit dans les implémentation de playerCommand (par exemple la méthode init), playerFactory (l'appel des méthodes et Context.show(),
- **DIP** : Les modules de haut niveau ne doivent pas dépendre des modules de bas niveau. Les deux doivent dépendre d'abstraction (on le voit à travers les liens entre interfaces/classes abstraites).
- **Programmer pour des interfaces:** les paramètres, attributs sont déclarés comme instances de la classe abstraite/interface plutôt que comme instances de classes particulières (on le voit partout)

UEF 7.3. Patrons De Conception

- **SRP, forte cohesion (citer l'un d'entre eux suffit)**: les factory s'occupe de la création, le contexte de l'affichage, et les PlayerCommand des traitements relatifs aux commandes.
- **Aussi OCP, Faible Couplage : découle de ce qui précèdent** (facultatif)

2. a. Quels patrons GRASP et/ou GoF pourriez vous identifiez dans ce diagramme ?
   b. Quelle est l'utilité de chacun.
   c. Pour les patrons GoF donnez la correspondance entre les éléments de chaque patron et celle du diagramme.

Rep : 3pts

GRASP : 1,5= (0.5*3)

Nom du patron	**Utilité**
Expert, forte cohésion	Voir les principes
Creator, Expert	Player factory
Polymorphisme	classe abstraite+redéfinition) exp. AbstractPlayerCommand et ses filles

GoF : (0.25*3= 0,75)*2= 1,5pts

Nom du patron	**Utilité**	**Correspondance**
Abstract Factory	Player factory pour la création des commandes avec le context associé	Product 1 et 2 sont le PlayerContext et PlayerCommand ; Abstract factory est PlayerFactory et les concrete sont DVDFactory et VHS Factory
Prototype	Il y a plusieurs types de commandes. Pour faciliter la création, on les crée une fois et on les met dans la Map	PlayerCommand est le prototype. La méthode est clone. Les concrètes prototype sont les sous classes de PlayerCommand

3. Complétez le code suivant : **(2pts)**

```
public class DVDFactory implements  PlayerFactory {  0,25
 private Map<String,PlayerCommand> commandMap = new HashMap<String,PlayerCommand>();
    public DVDFactory() {
        commandMap.put("play", new DVDPlayCommand());
        commandMap.put("stop", new DVDStopCommand());
        commandMap.put("fastForward", new DVDFastForwardCommand());
        commandMap.put("rewind", new DVDRewindCommand());
    }
    public PlayerCommand makeCommand(String command, PlayerContext context) {
        PlayerCommand playerCommand = commandMap.get(command);
        PlayerCommand newPlayerCommand = playerCommand.clone();  0,25
```

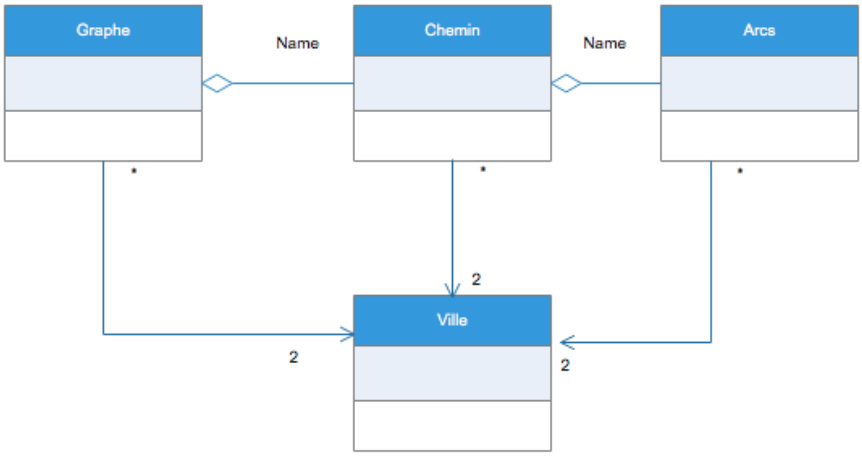
UEF 7.3. Patrons De Conception

```
        newPlayerCommand.setContext(context);
        newPlayerCommand.init();
        return newPlayerCommand;
    }
    public PlayerContext makeContext() {
        return new DVDContext();0,25
    }
    }
public class Test {
    public static void main(String[] args) {
        PlayerFactory factory = new DVDFactory();
        PlayerContext context = factory.makeContext();0,25
        PlayerCommand stopCommand = factory.makeCommand("stop",context); 0,25
        PlayerCommand playCommand = factory.makeCommand("play",context); 0,25
        PlayerCommand rewindCommand = factory.makeCommand("rewind",context); 0,25
        PlayerCommand fastCommand = factory.makeCommand("fastForward",context); 0,25
        context.setStopCommand(stopCommand);
        context.setPlayCommand(playCommand);
        context.setRewindCommand(rewindCommand);
        context.setFastForwardCommand(fastCommand);
        context.show();
     }
}
```

Exercice 2 (7 pts):

On voudrait construire un système de calcul du coût de réalisation d'autoroute entre deux villes et qui passe par des villes intermédiaires. Pour modéliser toutes les possibilités d'itinéraires, nous proposons la classe Graphe définie par une ville de départ, une ville d'arrivée, et l'ensemble des chemins possibles. Chaque Chemin est défini par sa ville de départ, sa ville d'arrivée et son coût calculé par la somme des couts des arcs qu'il contient. Sur chaque arc reliant deux villes nous avons les coûts de réalisation de ce tronçon d'autoroute.

1. Donnez le diagramme de classe correspondant (uniquement les classes). **1,5pts= 0,25*4 classes +0,25*2 relations.**



2. A quelle classe doit-on assigner la responsabilité de calculer le cout optimal de réalisation de l'autoroute ? Quel patron GRASP avez vous appliqué ? **0,25*4=1pt**
   Réponse :
   Les responsables sont Arc (getCout), Chemin (somme des couts), Graphe (appel de la méthode cout optimal qui parcours les chemins possibles et choisi le chemin optimal.

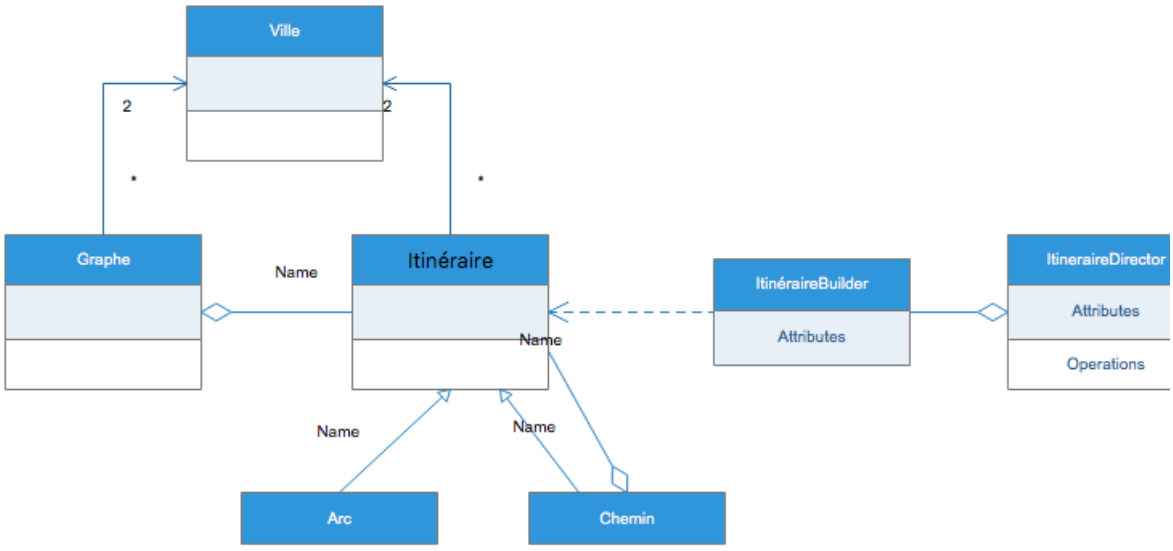
UEF 7.3. Patrons De Conception

On aura donc des experts partiels (patron Expert)

3. Selon le schéma actuel, à qui doit-on assigner la responsabilité de créer les arcs ? Expliquez quel patron GRASP avez vous appliqué ? **0,25 patron+ 0,25 explication=0,5pt**
   En appliquant le patron Creator le responsable sera Chemin car il y a une agrégation (chemin contient des arcs).

4. Nous souhaitons enregistrer ce graphe dans une base de donnée. Comment y procéder avec les patrons GRASP pour garder une cohésion élevée et un couplage faible? **0,25 patron+ 0,25 explication =0,5pt**
   Nous créons une pure fabrication qui permet de gérer les transactions avec la BDD.

5. Nous souhaitons améliorer la modélisation en utilisation les patrons GoF. En justifiant vos réponses :
   a. Proposez un patron de structure pour représenter les itinéraires. **0,5 =0,25*2 (patron+explication)**
   R : Composite : un itinéraire est soit un chemin ou un arc, et on peut voir un chemin comme une composition de sous chemins (d'itinéraire).
   b. Proposez un patron de création pour les créer. **0,5 (patron+explication)**
   R : Builder, c'est le patron le mieux adapté pour la création d'un composite.
   c. Donnez le diagramme de classe correspondant. **1,25=0,75 (composite)+0,5 (builder)**
   R :

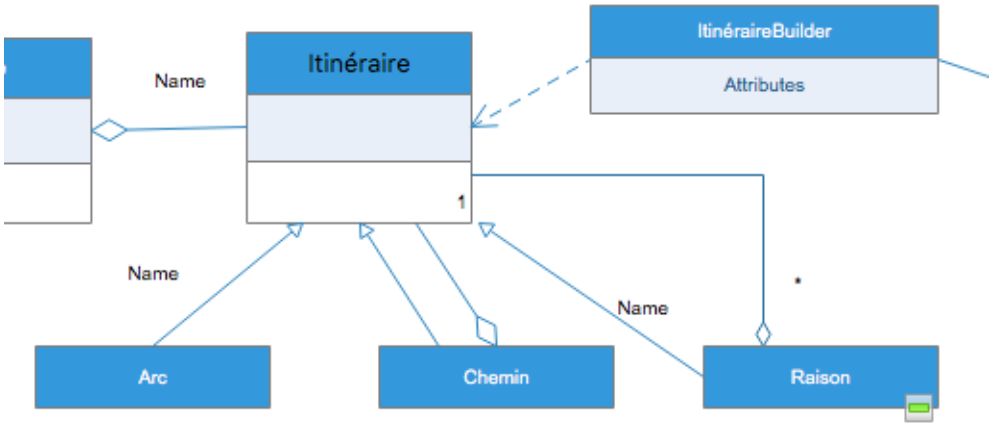


6. Lors de la réalisation de l'autoroute, nous avons remarqué que des couts supplémentaire peuvent être ajoutés sur un ou plusieurs tronçon pour plusieurs raisons possibles : des dédommagements des destructions de maisons et autres édifices, la création d'aires de repos,….etc. Comment peut-on répondre à ce changement de spécifications en utilisant les patrons du GoF, tout en respectant OCP (la fonction coût ne doit pas changer)? Complétez le diagramme de classe précédent (question 5) avec cette solution

R : 0,25 (patron)+0,5(expliquer le comment)=**0,75pts** Decorator : les raisons sont les décorations, redéfinissent la fonction cout en ajoutant le cout supplémentaire au cout de réalisation.

Schéma: **0,5**

UEF 7.3. Patrons De Conception



UEF 7.3. Patrons De Conception

Nom :……
Prénom :……..……..

Questions (7 points) :

1. Qu'est-ce qui rend le couplage "faible"? **1pt**

- **minimiser les dépendances (nombre réduit de lien, éviter beaucoup d'héritage, etc.),**
- **usage des interfaces/classes abstraires, programmer pour des interfaces pas des classes concrètes**
- **respecter SRP, forte cohésion**
- **respecter variation protégée.**

2. Quels patrons GRASP pourriez vous identifiez dans les patrons GoF suivant : ***Singleton, Abstract Factory, Bridge, Decorator*** **.** Expliquez. **2 pt = (0,25+0,25)*4 (en gras les plus importants)**

- **Singleton:**
  - **Creator** (c'est un patron de création qui s'occupe de la création d'une seule instance),
  - **Expert** (la classe Singleton est l'expert en info pour connaître le nombre d'instance créée)
- **Abstract Factory :**
  - **Creator** : C'est un patron de création
  - **Variation protégée** : la création de la famille de produits est encapsulé dans des classes appropriées pour protéger le reste du code des modifs
- **Bridge :**
  - **Variation protégée :**
  - polymorphisme
- **Decorator :**
  - **Polymorphisme**
  - Variation protégée

3. Analysez le code suivant. Implémente-t-il le patron Factory Method ? Expliquez.

```
public class XMLReaderFactory {
    public static XMLReader createXMLReader(){…}
}
public interface XMLReader {
    public void setContentHandler(ContentHandler handler):
    public void parse(InputStream is);
}
```

Réponse : 1= 0,25*4pt
Non, c'est une **fabrique statique** pas une factory method, **il y a une seule méthode de création, on ne délègue pas au classe fille la création du produit.**

4. Répondez par vrai ou faux en justifiant vos réponses (une réponse non justifiée ne sera pas considérée) : **6*(0,25+0,25)= 3pts**

A. L'application du patron « Expert » peut conduire à une cohésion faible.
Vrai, cas du contrôleur système par exemple
B. Le patron Flyweight est identique au patron Prototype.
Faux, Flyweight est un patron de structure, on l'utilise quand l'identité des objets n'est pas importante ce qui nous permet de partager la même instance ; Prototype est un patron de

UEF 7.3. Patrons De Conception

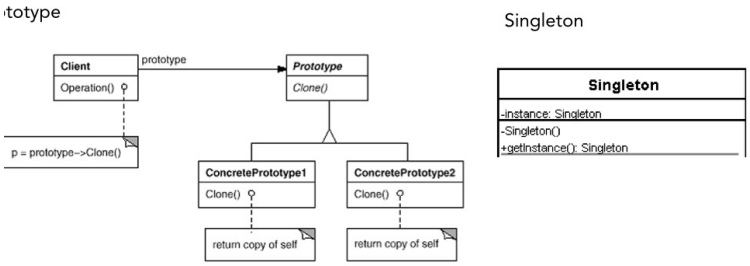
création utilisé quand le coût de création est supérieur au coût de la duplication ou quand on a des instances qui diffèrent juste par l'état permettant de minimiser le nombre de sous classes.

C. Le patron Builder peut être utilisé pour créer des objets structurés en « Composite »
Vrai, Builder permet de créer les objets compliqué, un composite est un.

D. Le patron « variation protégée » est fortement lié au patron Factory Method.
Vrai, dans FactoryMethod on encapsule la création en appliquant le principe « variation protégée »

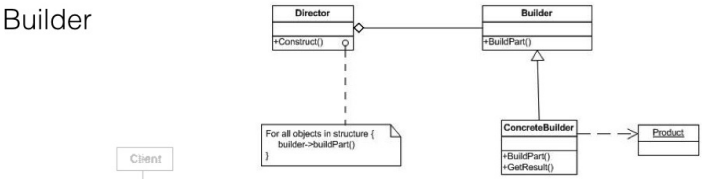
E. Le patron Indirection ne respecte pas le principe « faible couplage ».
Faux, au contraire l'intention d'Indirection est d'éviter le couplage entre deux classes..

F. Dans le patron Controller, la classe contrôleur est toujours une pure fabrication.
Faux, ça peut être une classe du domaine

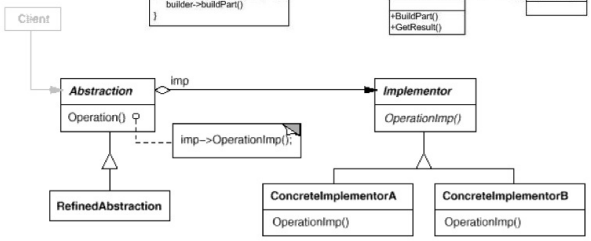


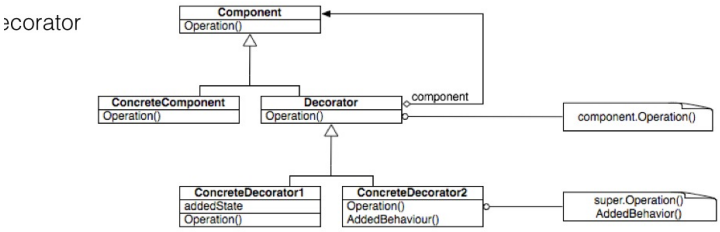
Singleton

Builder

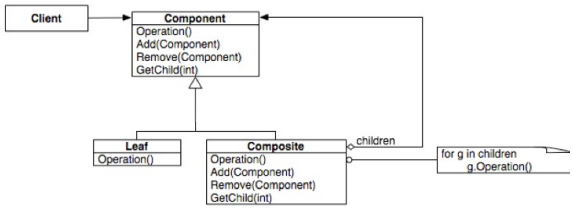


Bridge

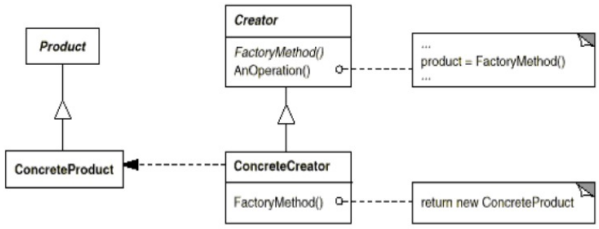




Composite



Factory Method



Abstract Factory