Départ. Informatique
BUT s3 – R3-04. Qualité de Conception

V. Deslandres

TD4 - Design Patterns (2)

EXERCICE 1 – Reconnaître un modèle dans du code

Voici un code Java comportant un patron de conception. Le reconnaissez-vous ?

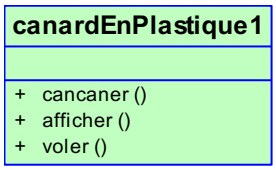
```
class CPU {
    public void freeze() { ... }
    public void jump(long position) { ... }
    public void execute() { ... }
}

class HardDrive {
    public byte[] read(long lba, int size) { ... }
}

class Memory {
    public void load(long position, byte[] data) { ... }
}

class MyClass {
    private CPU processor;
    private Memory ram;
    private HardDrive hd;

    public MyClass() {
        this.processor = new CPU();
        this.ram = new Memory();
        this.hd = new HardDrive();
    }

    public void start() {
        processor.freeze();
        ram.load(BOOT_ADDRESS,
hd.read(BOOT_SECTOR, SECTOR_SIZE));
        processor.jump(BOOT_ADDRESS);
        processor.execute();
    }
}

class ClientClass {
    public static void main(String[] args) {
        MyClass computer = new MyClass();
        computer.start();
    }
}
```

EXERCICE 2 – SuperCanard

Vous êtes développeur dans une société de jeux, où le gros succès repose sur des jeux de pédagogie et de simulation, basés sur des canards (avec notamment un héros SuperCanard). On a donc des applications où on doit représenter et déplacer des canards de toute sorte. Les concepteurs du jeu ont créé une super classe Canard, dont héritent différentes sous-classes de canards (schéma suivant), avec une hiérarchie qui permet d'envisager la réutilisation pour les applications de la société :

2.1- Tous les canards nagent et une grande majorité cancanent. Par contre, chaque type de canard possède ses propres caractéristiques (couleur dominante, secondaire, couleur des pattes, année de naissance, etc.) et certains canards ont des comportements qui leur sont propres (marcher, couver, réagir face à un prédateur, se reproduire, etc.). **Qu'est-ce que ça signifie pour les méthodes ?**

2.2 - Une amélioration du jeu est prévue qui prévoit de **faire voler** les canards en plus de nager. Vous envisagez dans un 1er temps d'ajouter une méthode *voler()* à la super classe Canard. Mais vous aviez oublié qu'une des sous-classes était :



Voir des canards en plastique traverser l'écran ne satisfait pas votre chef de projet… De plus il s'avère qu'un canard en plastique *couine* : il ne cancane pas…

Quelles solutions de conception mettre en place ?
*Indice : **isoler ce qui varie** de **ce qui ne varie pas** d'un canard à l'autre.*

Avec l'AGL, imaginez **une conception** qui permette d'autre part de :

- Affecter **dynamiquement** un comportement à une instance ;
- Décrire le comportement **par un ensemble d'attributs**, par ex. *pour le vol* : nombre de battements d'ailes par minute, altitude max, vitesse max, etc.; pour le *cri* : plage de décibels par ex.) ;
- **Ajouter un nouveau type de comportement**, par exemple de vol : *vol à propulsion* pour une classe *Canardator…*
- Pouvoir **réutiliser** le cancannement pour une autre application demandée par le CPT (Chasse Pêche et Traditions), où ils ont une classe Appeau (= instrument qui imite le cri du canard utilisé par les chasseurs).

2.3 - Une fois le modèle défini avec l'AGL, **générer le code java**. Pour cela :

- Compléter avec les éléments de conception nécessaire (identifiants, types),
- Ajouter les attributs qui vous semblent compléter l'application (attributs et méthodes propres aux sous-classes),
- Puis générer le code Java de l'application SimulateurCanards : `Langage – Générer le code Java.`
- Modifier le code sous un IDE et tester l'application : ça doit tourner !

2.4 - **Quel Design Pattern avons-nous utilisé ici ?**

EXERCICE 3 – Festival de musiques

Vous êtes chargé de proposer le S.I. d'une association qui organise des festivals (musique, théâtre, etc.). Si chaque festival possède ses particularités (lieu, dates, façon de communiquer, etc.), la préparation des festivals regroupe des tâches sont communes : gérer les participants, faire la programmation, définir le budget, rechercher des financements, communiquer, réserver le lieu, monter les scènes, réserver le SAMU, gérer les bénévoles, faire le bilan financier, etc.

On voudrait donc pouvoir créer des festivals *locaux (concrets)* à partir d'un festival *générique (abstrait)*, pour lequel on définit toutes les tâches communes à réaliser par ce festival (c'est le 'contrat', certaines méthodes étant abstraites, par ex. : réserver le lieu). Ce contrat sera modifié (ou implémenté) au niveau des festivals concrets.

On imagine ainsi créer une classe *AssociationFestival*, avec une méthode générale *créerFestival (String type)* qui va renvoyer un *Festival*, qui est valable pour tous les festivals. Les types de festival gérés à ce jour sont : **inDoor** et **outDoor**. Chaque type de festival **inDoor** va être capable de créer un

'vrai' festival selon le type donné en paramètre : Jazz à Vienne sera par ex. défini comme un festival **outDoor** de musique *Jazz*.
Pour un festival de musique on enregistre le genre, et on réserve un lieu. Pour un festival **inDoor** de musique classique, on devra en plus vérifier l'acoustique de la salle. Pour un festival **inDoor** pour les jeunes, on doit obtenir une autorisation spéciale et définir la plage des âges concernés.

Le festival est caractérisé par son nom, une ville et un lieu, une taille (on distingue les *gros* festivals des *moyens*, et aussi les *petits* : utile pour les filtres), une date de début et de fin, un budget souhaité, un budget alloué, un montant des dépenses, et qui regroupe des participants (les groupes), des mécènes et des sponsors, et est en relation avec d'autres associations partenaires.

1- Créer la classe abstraite *Festival* avec tous les attributs que vous pensez utile et les relations avec les classes nécessaires. Ajoutez le contrat tel que mentionné dans l'énoncé (que vous pouvez compléter : réserverLieu(), gérerParticipants(), définirBilletterie(), etc.).

2- Créer les sous-classes des festivals 'concrets' : OutDoorMusique, InDoorMusique, InDoorClassique, InDoorJeunes et InDoorThéatre, avec leurs spécificités mentionnées dans l'énoncé.

3- Créer maintenant une classe abstraite AssociationFestival, définie par son président, co-président, trésorier et secrétaire, et qui propose 2 méthodes :
   - La méthode *préparerFestival(String type)* qui consiste, à partir d'une instance de Festival créée localement, à appliquer toutes les 'méthodes du contrat' ;
   - La méthode abstraite *créerFestival(String type)* retournant une instance de notre classe Festival.
   - Bien entendu *préparerFestival()* fait appel à *créerFestival()*.

4- Quel Design Pattern avons-nous utilisé ici ?

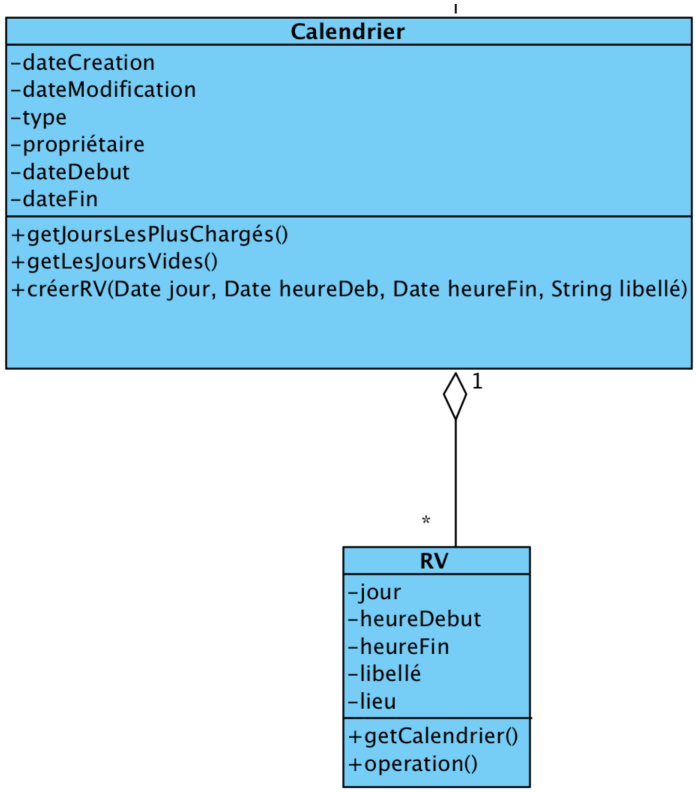
EXERCICE 4 – Jeu d'aventures

Imaginez que vous ayez à concevoir un jeu d'aventure, avec des personnages : roi, reine, chevalier, troll, etc. Chaque personnage peut faire usage d'armes variées : arc et flèches, poignard, hache, épée, mais une seule à la fois. Par contre le personnage peut en changer au cours de son aventure.

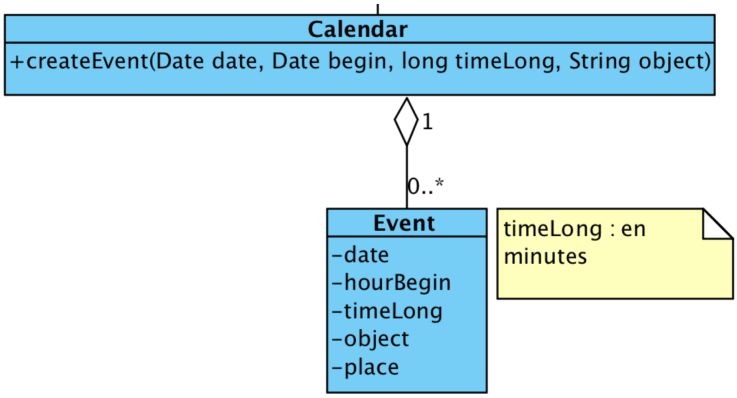
Quel Design Pattern serait utile ici ? Proposez une modélisation sous l'AGL (sans l'implémenter).

EXERCICE 5 – Calendrier

Au sein de votre équipe, pour la majorité des applications, on utilise un Calendrier avec les classes suivantes :



Voilà qu'un nouveau (gros) client vous demande d'interfacer votre logiciel phare avec son propre calendrier, et qui possède une structure similaire mais légèrement différente :



Date begin est l'heure de début de l'événement.
Que faut-il faire pour intégrer ce calendrier à votre application de manière à satisfaire le Client ? Donnez le DCL UML.

EXERCICE 6 – Recettes de cuisine

6.1 - Préparation

Modéliser une classe "Met" qui possède les méthodes :

- *toString()* qui renvoie le nom de ce met et les informations suivantes
- *estSucre(), estSale(), estSucreSale(),*
- le *nombreDeCalories()*
- *estDietetique()* : retourne *true* si le nombre de kcalories est inférieur à 200

Créer les classes Sucre, Sel, Poire, Pomme, Framboise, Veau, Bœuf, Chocolat, Carotte, HaricotVert, qui héritent de Met.

6.2 - Plat composé

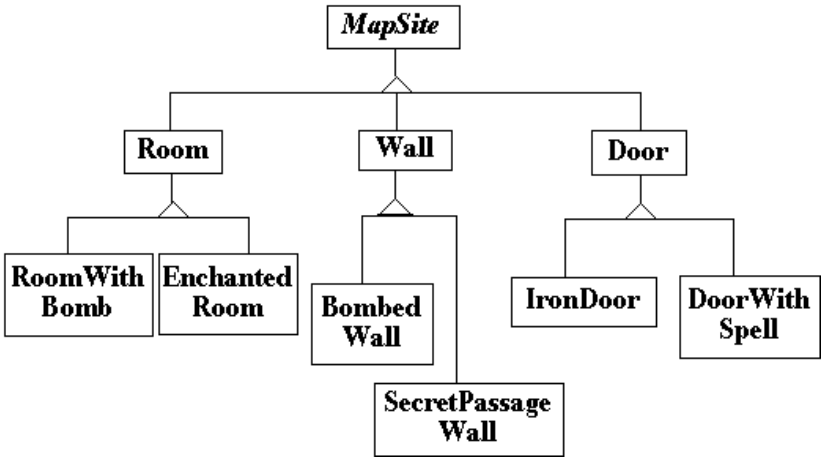
On souhaite représenter un plat composé (un bœuf carotte par exemple, ou une poire belle hélène) comme un met particulier.

Dans ce cas, la méthode *toString()* renvoie la liste des ingrédients du met composé. Un met est alors sucré si au moins un des éléments est sucré, salé si un au moins est salé et sucré-salé quand au moins un élément est sucré et un autre salé.

Comment procéder ?
Modéliser une solution en UML puis générer le code Java.
Tester votre implémentation avec quelques objets.

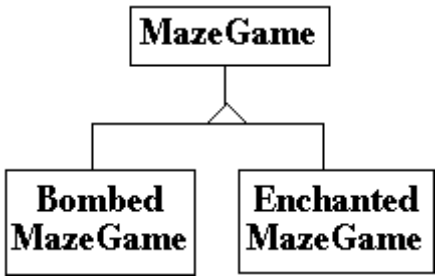
EXERCICE 7 – Labyrinthe

Soit le jeu de labyrinthe suivant :



Implémentation proposée :

```
class MazeGame {

     public Maze createMaze(){
          Maze aMaze = new Maze();

          Room r1 = new Room( 1 );
          Room r2 = new Room( 2 );
          Door theDoor = new Door( r1, r2);

          aMaze.addRoom( r1 );
          aMaze.addRoom( r2 );

          r1.setSide( North, new Wall() );
          r1.setSide( East, theDoor );
          r1.setSide( South, new Wall() );
          r1.setSide( West, new Wall() );

          r2.setSide( North, new Wall() );
          r2.setSide( East, new Wall() );
          r2.setSide( South, new Wall() );
          r2.setSide( West, theDoor );

          return aMaze;
     }
}
```



7.1 - On a 2 sortes de murs, de pièces… comment faire des labyrinthes spécifiques ? On pourrait créer des sous-classes et redéfinir **createMaze()** avec les bons éléments, par exemple :

```
class BombedMazeGame extends MazeGame {

     public Maze createMaze(){
          Maze aMaze = new Maze();

          Room r1 = new RoomWithABomb( 1 );
          Room r2 = new RoomWithABomb( 2 );
          Door theDoor = new Door( r1, r2);

          aMaze.addRoom( r1 );
          aMaze.addRoom( r2 );

          r1.setSide( North, new BombedWall() );
          r1.setSide( East, theDoor );
          r1.setSide( South, new BombedWall() );
          r1.setSide( West, new BombedWall() );
```

Idem pour EnchantedMaze game. Mais ça fait beaucoup de copier/coller.

7.2 – L'idée serait donc de créer une classe abstraite **MazeGame :**

```
abstract class MazeGame {

     public Maze makeMaze() { return new Maze(); }
     abstract public Door makeDoor(Room r1, Room r2) ;
```

```
    abstract public Room makeRoom(int n );
    abstract public Wall makeWall();
```

Avec la méthode `createMaze()` légèrement différente :

```
    public Maze createMaze(){
        Maze aMaze = makeMaze();

        Room r1 = makeRoom( 1 );
        Room r2 = makeRoom( 2 );
        Door theDoor = makeDoor( r1, r2);

        aMaze.addRoom( r1 );
        aMaze.addRoom( r2 );

        r1.setSide( North, makeWall() );
        r1.setSide( East, theDoor );
        r1.setSide( South, makeWall() );
        r1.setSide( West, makeWall() );

        r2.setSide( North, makeWall() );
        r2.setSide( East, makeWall() );
        r2.setSide( South, makeWall() );
        r2.setSide( West, theDoor );

        return aMaze;
        }
    }
}
```

Implémentez, dans les sous-classes **`BombedMazeGame`** `et` **`EnchantedMazeGame`**, les méthodes abstraites avec les éléments qui vont bien.

Quel **design pattern** avons-nous donc utilisé ici ?