Université Claude Bernard Lyon 1 4A , 2019–2020
Polytech Lyon - Département Informatique

ISI-3

TD3. Design Patterns

Laëtitia Matignon

---

1 Design Pattern Observateur/Observé

Soit le code suivant utilisant les classes représentées dans le diagramme de la figure 1 :

```
SujetConcret s = new SujetConcret("PopCorn", 1.29f);
NameObserver nameObs = new NameObserver();
PriceObserver priceObs = new PriceObserver();
s.addObserver(nameObs);
s.addObserver(priceObs);
s.setName("Frosties");
s.setPrice(4.57f);
s.setPrice(9.22f);
s.setName("Smacks");
```

On souhaite obtenir l'affichage suivant lors de l'exécution du code précédent :

```
Name changed to Frosties
Price changed to 4.57
Price changed to 9.22
Name changed to Smacks
```

Question 1 *Proposer une implémentation pour :*
- *les méthodes* `setName` *et* `setPrice` *de la classe* `SujetConcret`
- *les méthodes* `update` *pour chaque* `Observer`,

de sorte à obtenir l'affichage souhaité.

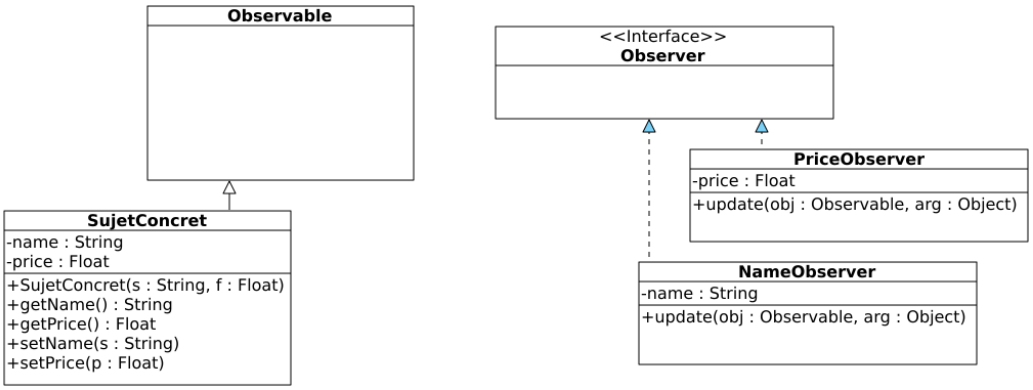

FIGURE 1 – Design Pattern Observateur Observé

2 Design Pattern

Le code ci-dessous sert à tester l'énoncé suivant : *Le directeur s'occupe des achats. S'ils sont inférieur à 100 euro, il peut donner son accord. S'ils sont supérieurs à 100 euro mais inférieur à 100000 euro, il faut l'accord du vice-président. Au delà, il faut l'accord du président.*

```
President eric = new President();
Vicepresident antoine = new Vicepresident();
Directeur samuel = new Directeur();
samuel.setSuivant(antoine);
antoine.setSuivant(eric);
Achat p = new Achat(50, "Formation"); samuel.acheter(p);
p = new Achat(24000, "Voiture"); samuel.acheter(p);
p = new Achat(150000, "Maison"); samuel.acheter(p);}
```

Question 2 *Proposer un design pattern et son implémentation pour faire fonctionner le programme précédent.*

3 Design Pattern Fabrique

Soit le diagramme de classes de la figure 2. On peut, à partir de ces classes, créer différents types de labyrinthes.

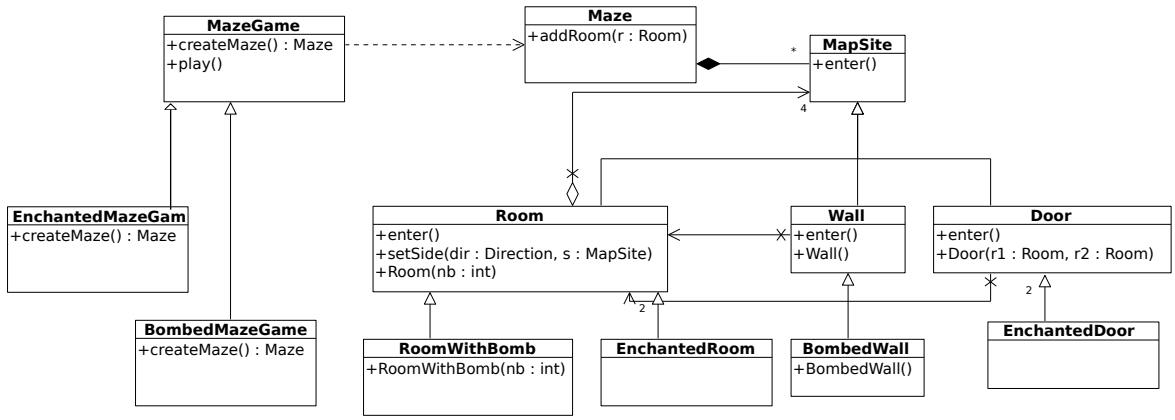

FIGURE 2

Pour créer un labyrinthe classique, on utilise la méthode `createMaze()` de la classe `MazeGame` qui se charge d'instancier un labyrinthe avec des éléments classiques de type `Room`, `Door`, `Wall`. Par exemple on a le code suivant :

```
class MazeGame {
void Play() {...}
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;}}
```

Pour créer un labyrinthe avec des bombes, on utilise la méthode `createMaze()` de la classe `BombedMazeGame` qui se charge d'instancier un labyrinthe avec des éléments de type `RoomWithABomb`, `Door`, `BombedWall`. Par exemple on a le code suivant :

```
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());
r2.setSide(North, new BombedWall());r2.setSide(East, new BombedWall());
r2.setSide(South, new BombedWall());r2.setSide(West, theDoor);
return aMaze;}}
```

De même pour un labyrinthe enchanté. **Dans tous les cas, l'organisation du labyrinthe est la même (nombre et position des éléments).**

Question 3 *Proposer une application du design pattern* ***Fabrique abstraite*** *permettant de n'avoir qu'une seule méthode* `createMaze`*. Cette méthode créera les éléments du labyrinthe sans connaitre leur type précis.*