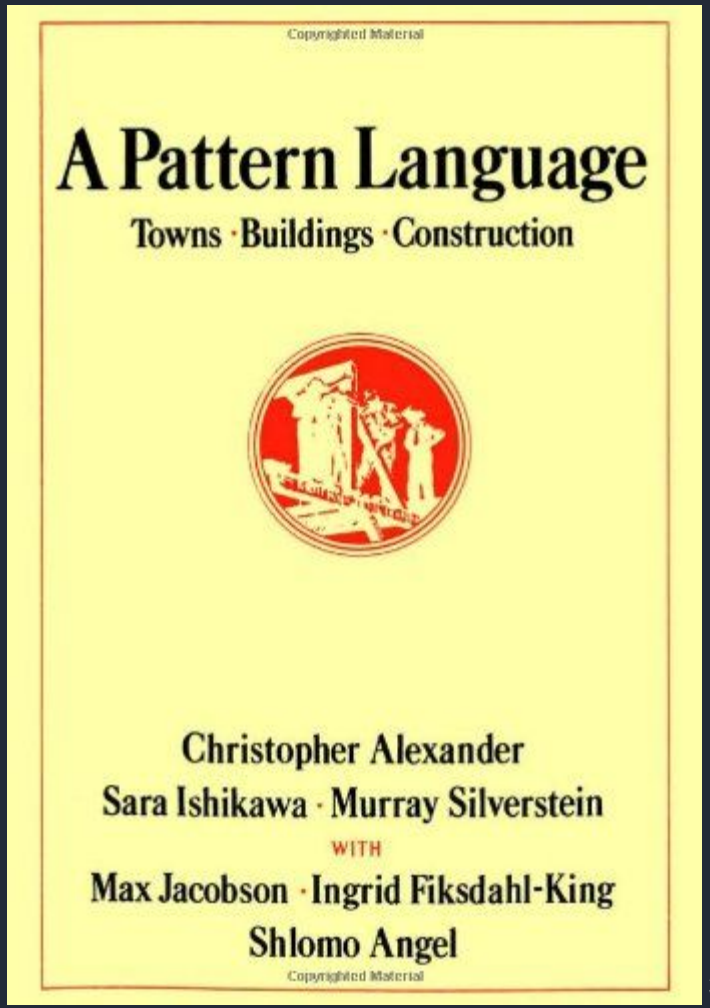
Kevin Ayrault, Maxime Nguyen, Maxime Constans, Aymeric Bachelet

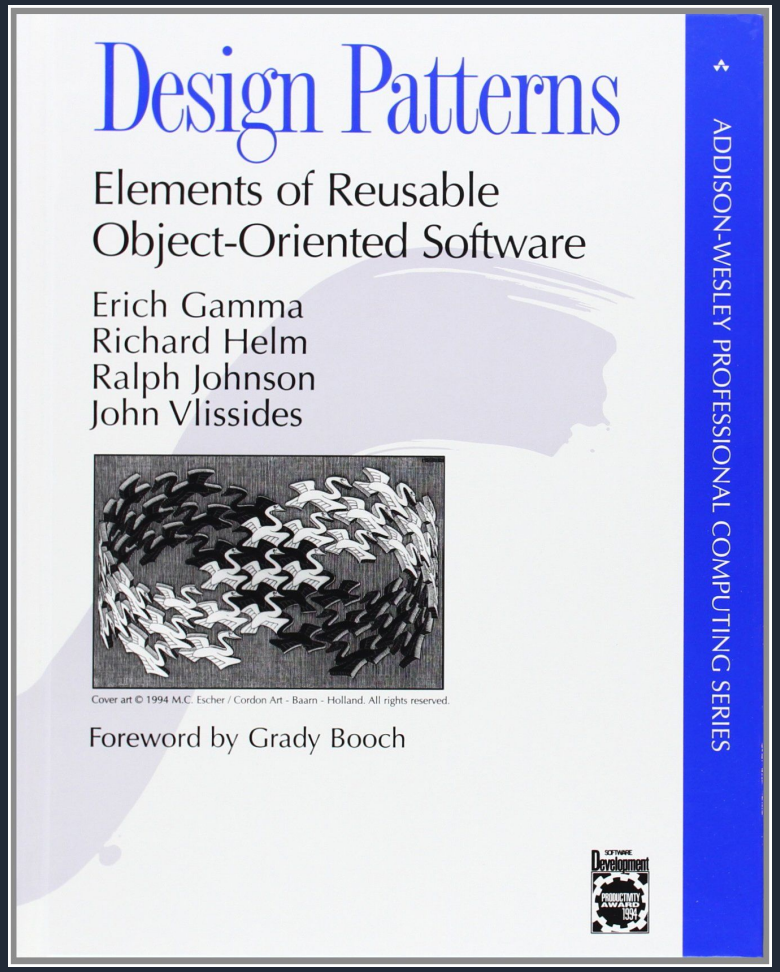
Sommaire

- Introduction sur les patterns
- Modélisation du pattern
- Description du pattern
- Lien avec les autres patterns

Origine des designs patterns

Publié en 1977





Origine des designs patterns

Publié en 1994

Introduction sur les patterns

- Confère des propriétés caractéristiques
- Solution «abstraite» pour des problèmes communs
- Template qui peut être utilisée dans différentes situations.

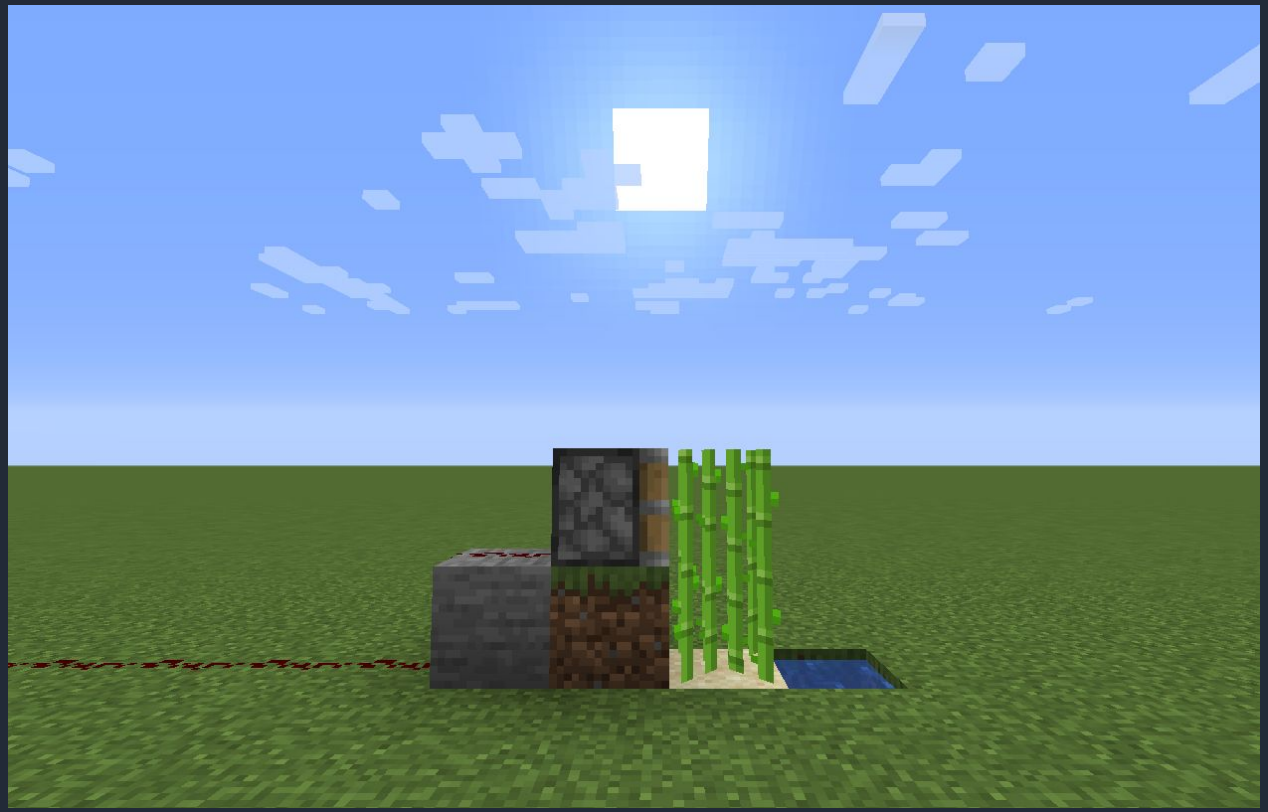
➔ solution orienté objets qui montre des relations et des interactions entre des classes ou objets sans les spécifier concrètement.

Le besoin

Ecrire des fonctions à tout bout de champ

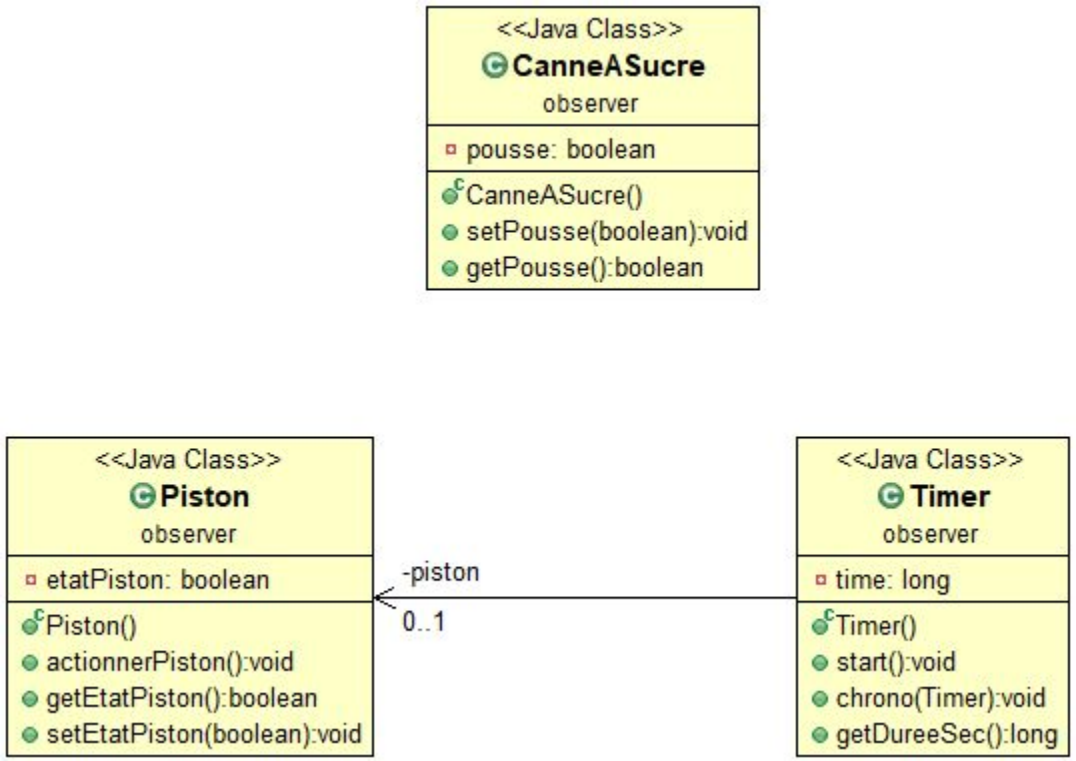
Utiliser des patterns déjà tout fait pour avoir un code SOLID

Modélisation des besoins



Modélisation des besoins

Modélisation des besoins



Modélisation des besoins

```java
public class CanneASucre {

    private boolean pousse;


    public CanneASucre() {
        this.pousse = false;
    }

    public void setPousse(boolean pousse) {
        this.pousse = pousse;
    }

    public boolean getPousse() {
        return pousse;
    }

}
```

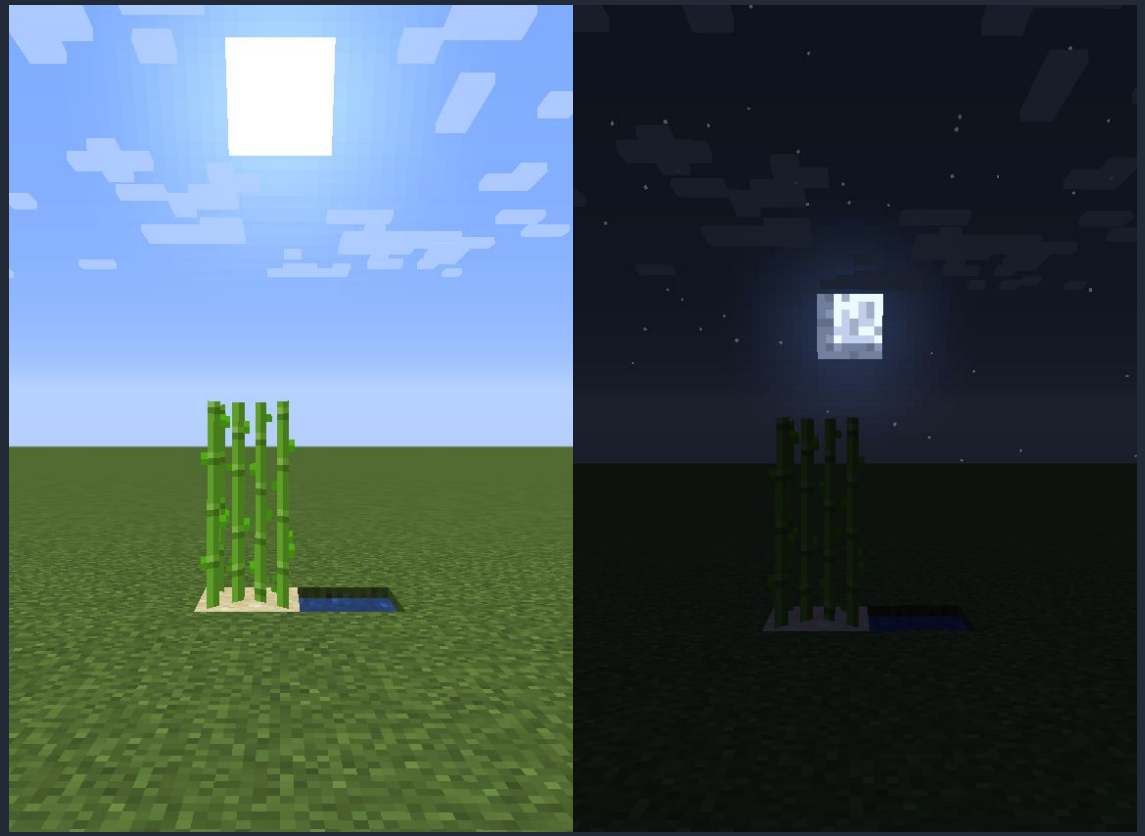
```java
public class Piston {

    private boolean etatPiston;

    public Piston() {
        this.etatPiston = false;
    }

    public void actionnerPiston() {
        this.setEtatPiston(true);
        this.setEtatPiston(false);
    }

    public boolean getEtatPiston() {
        return etatPiston;
    }

    public void setEtatPiston(boolean etatPiston) {
        this.etatPiston = etatPiston;
    }


}
```

Modélisation des besoins

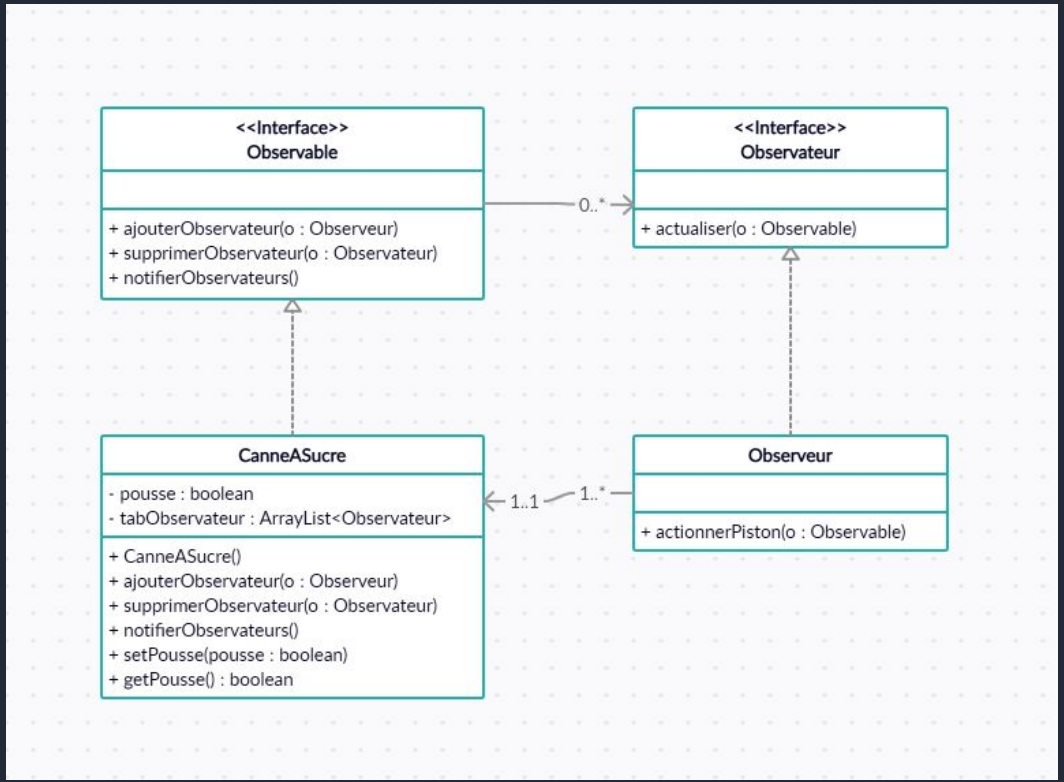
```java
public class Timer {
    private long time = 0;
    private Piston piston;

    public void start() {
        time=System.currentTimeMillis();
        }

    public void chrono(Timer timer) {
        if (timer.getDureeSec() >= 1800) {
            piston.actionnerPiston();
        }
    }

    public long getDureeSec() {
        return time/1000;
        }
}
```

```java
public class Main {

    public static void main(String[] args) {
        CanneASucre c = new CanneASucre();
        Timer t = new Timer();

        t.start();
        c.setPousse(true);
        while(t.getDureeSec() < 1800) {
            t.chrono(t);
        }
    }

}
```

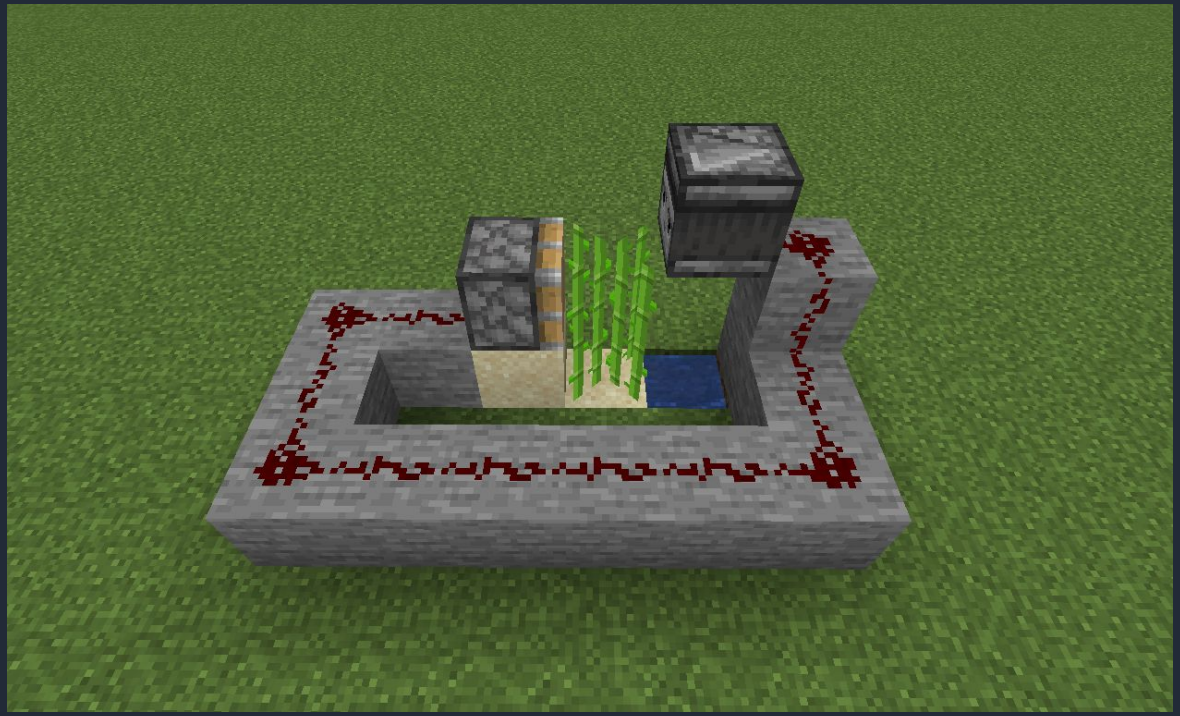
Les problèmes



Résolution des problèmes



Résolution des problèmes



Résolution des problèmes

```java
package observer;

//Interface implémentée par tous les observateurs.
public interface Observateur
{
      // Méthode appelée automatiquement lorsque l'état du bloc change.
      public void actualiser(Observable o);
}
```

```java
package observer;

//Interface implémentée par toutes les classes souhaitant avoir des observateurs à leur écoute.
public interface Observable
{
     // Méthode permettant d'ajouter (abonner) un observateur.
     public void ajouterObservateur(Observateur o);
     // Méthode permettant de supprimer (résilier) un observateur.
     public void supprimerObservateur(Observateur o);
     // Méthode qui permet d'avertir tous les observateurs lors d'un changement d'état.
     public void notifierObservateurs();
}
```

Résolution des problèmes

```java
package observer;

public class Observer implements Observateur {

    @Override
    public void actualiser(Observable o) {
        if(o instanceof CanneASucre)
        {
            //Ici une méthode qui permet d'activer le piston pour remettre la canne a sucre à son état initial
            System.out.println("Le piston s'active, la canne à sucre retourne à sa taille initiale.");
        }
    }

}
```

```java
package observer;

import java.util.ArrayList;

public class CanneASucre implements Observable {

     private boolean pousse;// true si la canne a sucre est prête
     private ArrayList tabObservateur;// Tableau d'observateurs.

    public CanneASucre()
    {
            pousse=false;
            tabObservateur=new ArrayList();
    }

    @Override
    public void ajouterObservateur(Observateur o) {
        tabObservateur.add(o);
    }

    @Override
    public void supprimerObservateur(Observateur o) {
        tabObservateur.remove(o);
    }

    @Override
    public void notifierObservateurs() {
         for(int i=0;i<tabObservateur.size();i++)
         {
                Observateur o = (Observateur) tabObservateur.get(i);
                o.actualiser(this);
         }
    }

    public void setPousse(boolean pousse)
    {
            this.pousse=pousse;
            notifierObservateurs();
    }

    public boolean getPousse()
    {
            return pousse;
    }
}
```

Résolution des problèmes

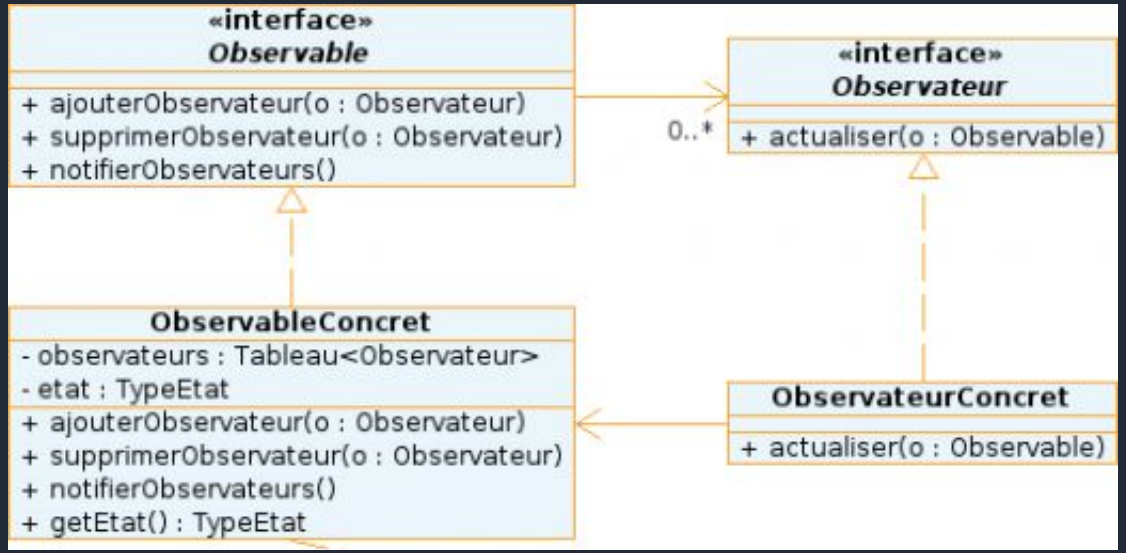
```java
package observer;

public class Main {
     public static void main(String[] args)
     {
             // Création de l'objet CanneASucre observable.
             CanneASucre c = new CanneASucre();
             // Création de l'observeur
             Observer ob = new Observer();
             // On ajoute Observer comme observeur de CanneASucre.
             c.ajouterObservateur(ob);
             // On simule la pousse de la canne à sucre.
             c.setPousse(true);

     }
}
```

Problems @ Javadoc Declaration Console

<terminated> Main [Java Application] C:\Users\kqasp\.p2\pool\plugins\org.eclipse.justj.ope

```
Le piston s'active, la canne à sucre retourne à sa taille initiale.
```

Diagramme de classe



Le pattern Observer

- Pattern Comportementale (Behavior)
  - Résolution des problèmes liés aux interactions entre les objets

- Informe un objet (Observateur) tout changement d’un autre objet (Observable) s’ils sont liés

La solution qu’il apporte

- Interface Observable :
  - Informer de son changement d’état à un Observateur liée

- Interface Observateur :
  - Récupérer et traiter le changement d’état de l’Observable

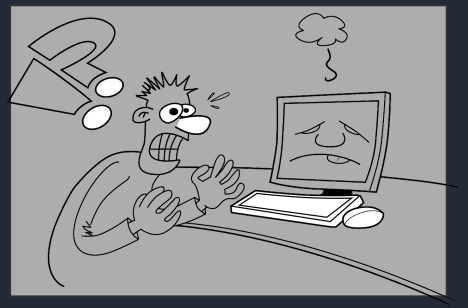
Lien avec le principe SOLID

SRP → 

OCP → 

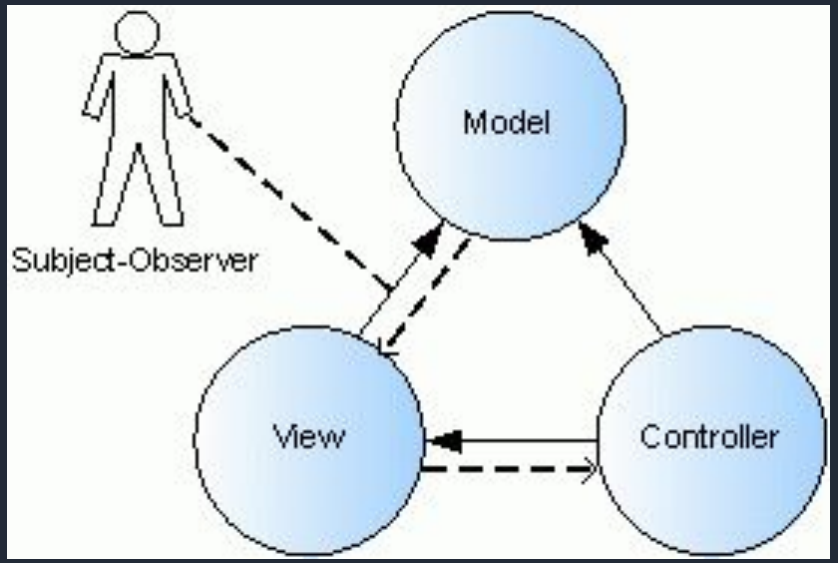
DIP → 

Limite du pattern Observer



Lien avec un autre pattern

MVC (Model-View-Controller)



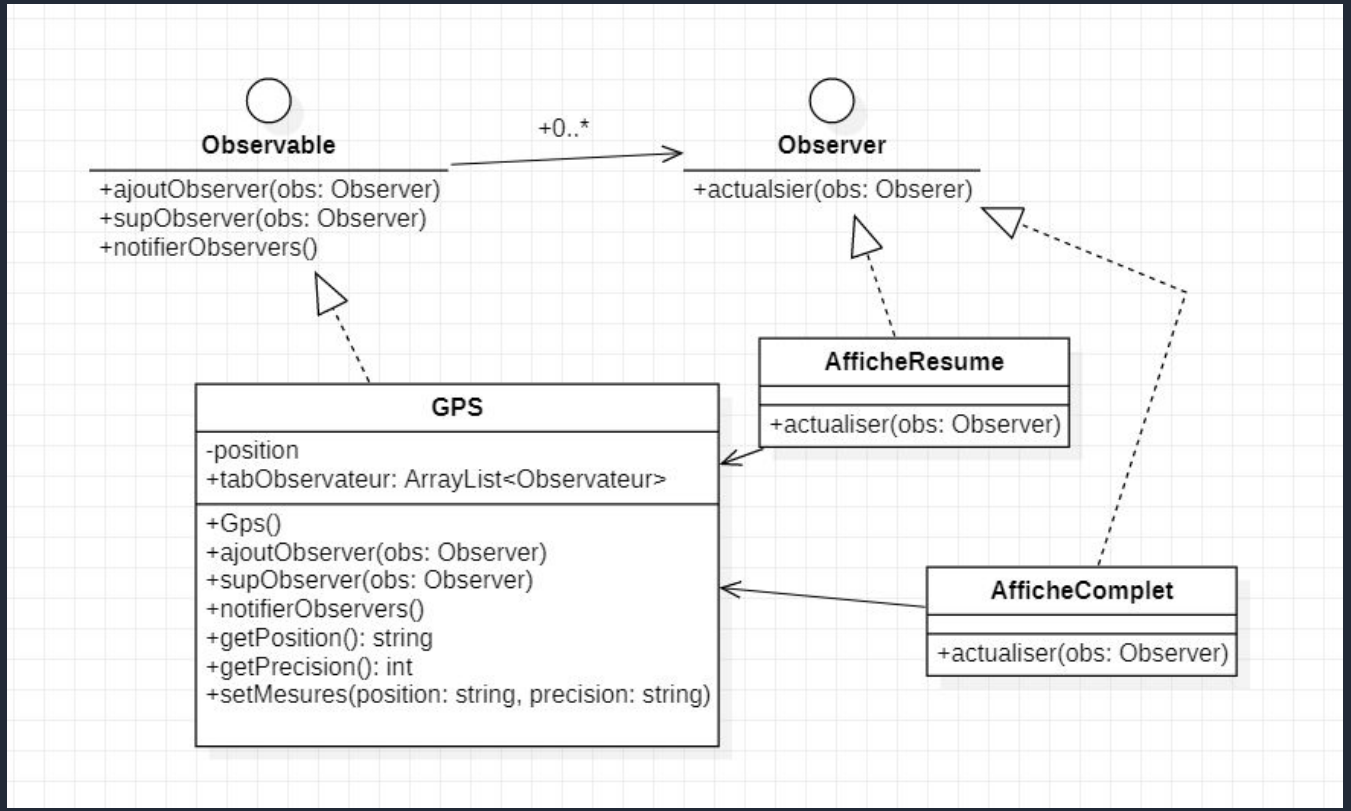
Autres exemple...

Position

Actualisation

Notifier
Observer

Autres exemple...



QCM

Sur quel Design pattern était la présentation ?

- [x] Observer
- [ ] Aube Serveur
- [ ] Oops Cerf Vert
- [ ] Hein ? J’ai entendu un QCM, on parle de quoi ?

QCM

A quel type de patron de conception appartient l’Observateur ?

- ❑ Création
- ❑ Structure
- ☒ Comportement

QCM

Lesquels de ces principes SOLID sont respectés ?

- ❑ SCP
- ☒ OCP
- ❑ LSP
- ❑ ISP
- ☒ DIP
- ❑ Je choisis les bonnes réponses

QCM

Le pattern Observer fait-il partie du GoF ?

- [x] Oui
- [ ] Non
- [ ] Pain au chocolatine

QCM

Parmis les choix suivants, quel est le pattern en relation avec l'Observer ?

- ❑ Composite
- ❑ Decorator
- ❑ Idiotisme
- ☒ MVC (Model-View-Controller)
- ❑ Aucun

QCM

Ce pattern fait-il partie du GoF ? (Adapter selon la réponse précédente)

- ❑ Oui
- ☒ Non
- ❑ Patate bien que oui, patate bien que non

QCM

Quel est le rôle du pattern Observer ?

- [x] Définir une dépendance parmis tant d'autre entre objet
- [ ] Gagner 1 Millions de dollars
- [x] Notifier et mettre à jour automatiquement
- [ ] Observer ce que fait une classe

Références utilisés

- https://www.freecodecamp.org/news/the-basic-design-patterns-all-developers-need-to-know/
- https://fr.wikipedia.org/wiki/Observateur_(patron_de_conception)
- https://design-patterns.fr/observateur
- https://sourcemaking.com/design_patterns/observer
- http://goprod.bouhours.net/

Merci de votre attention !