

**POO2**

# **Design Patterns**

**Pascal Hurni  
CPNV**

# Introduction

## Design Patterns (Patrons de conception)

- ◆ **Général**

C'est une solution générique à un problème commun.

- ◆ **Orienté-Objet**

Un patron nomme, décrit et propose une solution à un problème de conception. Il définit dans quelles situations l'utiliser.

- ◆ **A l'extrême**

Un patron fait ressortir une certaine faiblesse (ou plusieurs) d'un langage.

# Gang of Four (GoF)

Les quatres auteurs qui ont formalisés dans leur ouvrage en 1995 les DP les plus utilisés:

- ◆ Erich Gamma
- ◆ Richard Helm
- ◆ Ralph Johnson
- ◆ John Vlissides

# Catégories

- ◆ **Créationnels**

S'occupe de l'instantiation des objets, donc de la manière dont les objets sont créés.

- ◆ **Structuraux**

S'occupe de la façon dont les objets sont combinés entre eux, donc de l'architecture du logiciel. Ceci est défini avant l'exécution.

- ◆ **Comportementaux**

S'occupe de la façon dont les objets communiquent entre eux, donc de la manière d'appeler des traitements.

# Créationnels – Singleton

- ◆ Permet de garantir qu'au cours de l'exécution du programme **UNE SEULE** instance de la classe existe.
- ◆ Comment?
  - Rendre « private » les constructeurs
  - Fournir une méthode de classe qui s'occupe d'instancier LA référence ou de la retourner si elle est déjà instanciée

# Créationnels – Factory

- ◆ Evite au code ayant besoin d'un objet de connaître la classe exacte à instancier.

Avant

```
SomeLogger logger;  
logger = new SomeLogger();  
logger.log("Been here");
```

Après

```
Logger logger;  
logger = LoggerFactory.create();  
logger.log("Been here");
```

# Créationnels – Factory

Code inhérent au DP:

```
interface Logger {  
    public void log(String message);  
}
```

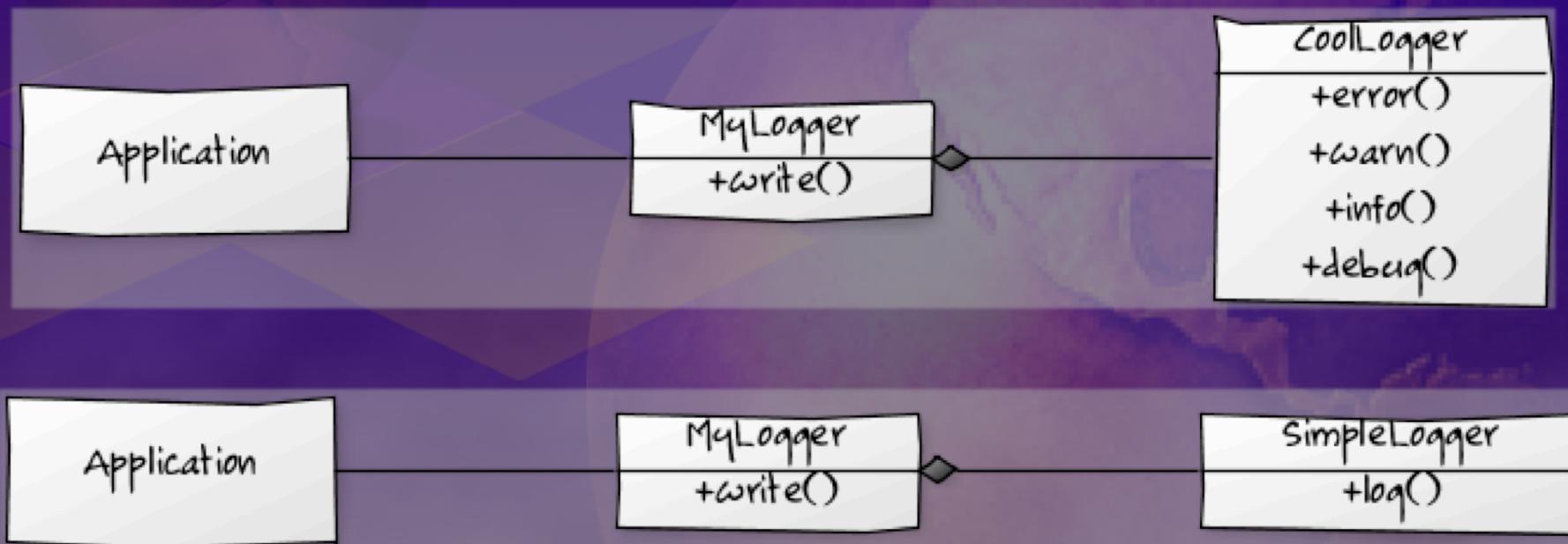
```
class LoggerFactory {  
    public static Logger create() {  
        return new MySuperLogger();  
    }  
}
```

Implémentation:

```
class MySuperLogger implements Logger {  
    ...  
  
    public void log(String message) {  
        ...  
    }  
}
```

# Structuraux – Wrapper (Adapter)

- ◆ Permet de connecter un composant à l'application quelque soit son interface.



# Structuraux – Decorator

- ◆ Permet d'ajouter **des responsabilités supplémentaires à une instance**. Ceci sans utiliser l'héritage. L'ajout des responsabilités se fait à la construction de l'instance.

```
class ConsoleLogger {  
    public void log(String msg) {  
        System.out.println(msg);  
    }  
}
```

```
class FileLogger {  
    ...  
    public void log(String msg) {  
        file.println(msg);  
    }  
}
```

```
class PidLogger extends ?WHICH?Logger {  
    public void log(String msg) {  
        super(Magic.getPid() + " " + msg);  
    }  
}
```

# Structuraux – Decorator

Code inhérent au DP:

```
interface Logger {  
    public void log(String message);  
}
```

Code original adapté:

```
class ConsoleLogger implements Logger {  
    public void log(String msg) {  
        System.out.println(msg);  
    }  
}
```

```
class FileLogger implements Logger {  
    ...  
    public void log(String msg) {  
        file.println(msg);  
    }  
}
```

# Structuraux – Decorator

## Nouvelle responsabilité:

```
class PidLogger implements Logger {  
    protected Logger decorator;  
  
    public PidLogger(Logger decorator) {  
        this.decorator = decorator;  
    }  
  
    public void log(String message) {  
        decorator.log(Magic.getPid() + " " + message);  
    }  
}
```

## Code utilisateur:

```
Logger l1 = new PidLogger(new ConsoleLogger());  
l1.log("<- Process Id out to console")  
  
Logger l2 = new PidLogger(new FileLogger("app.log"));  
l2.log("<- Process Id out to file")
```

# Comportementaux – Chain of responsibility

- ◆ Permet à un nombre variable d'objets de répondre à une requête sans qu'ils se connaissent.
- ◆ L'initiateur décide de la manière d'appeler les maillons et de récolter les résultats qui sont tous de même nature.

```
class SpamBlocker {  
    public boolean shouldBlock(EMail email) {  
        boolean block = false;  
        block |= wordsHandler.shouldBlock(email);  
        block |= attachmentHandler.shouldBlock(email);  
        block |= recipientsHandler.shouldBlock(email);  
        return block;  
    }  
}
```

# Comportementaux – Chain of responsibility

Code inhérent au DP:

```
interface SpamHandler {  
    public boolean shouldBlock(EMail email);  
}
```

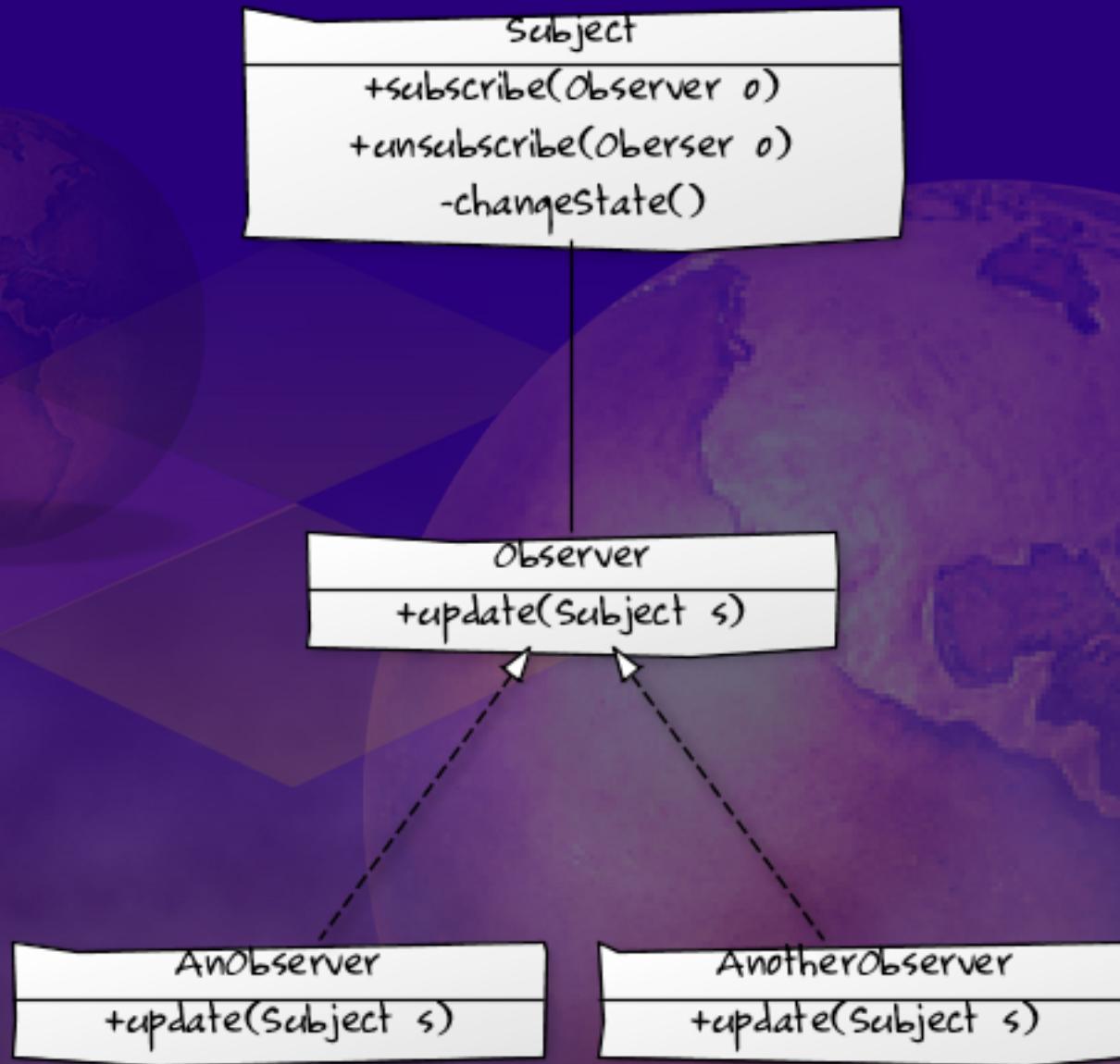
Code original adapté:

```
class SpamBlocker {  
    public addHandler(SpamHandler handler) {  
        handlers.add(handler);  
    }  
  
    public boolean shouldBlock(EMail email) {  
        boolean block = false;  
        for (SpamHandler handler: handlers) {  
            block |= handler.shouldBlock(email);  
        }  
        return block;  
    }  
}
```

# Comportementaux – Observer

- ◆ Permet à un objet de communiquer son changement d'état à des abonnés.  
Aussi connu sous le terme **publish-subscribe** ou **document-view**.
- ◆ Caractéristiques
  - Les observateurs peuvent s'inscrire et se désinscrire au cours de la vie du sujet
  - Le sujet ne fait qu'annoncer, il ne traite donc pas de retour de la part des observateurs

# Comportementaux – Observer



# Comportementaux – Visitor

- ◆ Permet de découpler les structures de données des algorithmes appliqués dessus.
- ◆ L'objectif est de pouvoir utiliser de manière interchangeable des algorithmes sans qu'ils connaissent les détails d'implémentation des structures de données et vice-versa.

```
class Words extends ArrayList<String> {  
    public String countWords(String word) {  
        int count = 0;  
        for (int i=0; i<length(); i++) {  
            if (at(i).equals(word)) {  
                count++;  
            }  
        }  
    }  
}
```

```
Words words;  
int count = words.countWords("api");
```

# Comportementaux– Visitor

Code inhérent au DP:

```
interface WordsVisitor {  
    public void visit(String item);  
}  
interface WordsBrowsable {  
    public void browse(WordsVisitor visitor);  
}
```

Code original adapté:

```
class Words extends ArrayList<String> implements WordsBrowsable {  
    public void browse(WordsVisitor visitor) {  
        for (int i=0; i<length(); i++) {  
            visitor.visit(at(i));  
        }  
    }  
}
```

```
Words words;  
...  
WordCounter counter = new WordCounter("api");  
words/browse(counter);  
int count = counter.getTotalCount();
```