Génie Logiciel : Conception

Intro aux Patrons de Conception

Sébastien Mosser

INF 5153 - Cours #6 - A19

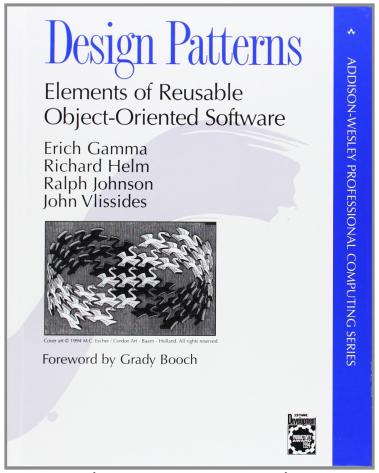
UQÀM | Département d'informatique

Crédit Images: Pixabay & Pexels

ace

Bibliographie

(le style est "discutable", mais le contenu raisonnable)



(référence, mais indigeste)

1 La danse des canards

2 Principes & Classification

3 Retours sur le Poker

La danse des canards

Jean-Michel Bruel
Professor of Software Engineering
University of Toulouse

1

5

Les canards savent ...

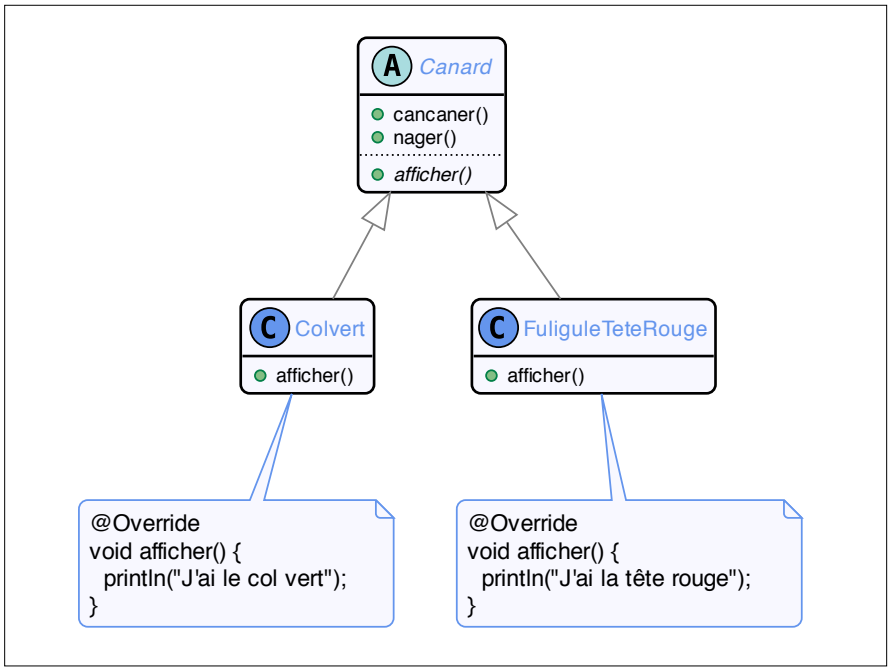
cancaner
nager

Vrai pour toutes les espèces de canard

s'afficher

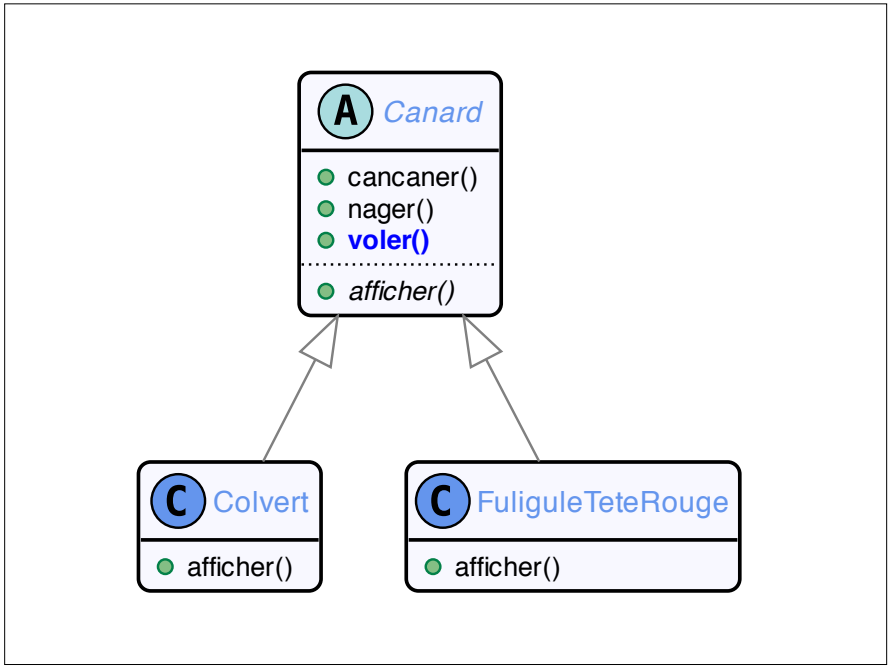
Dépend de l'espèce

6



7

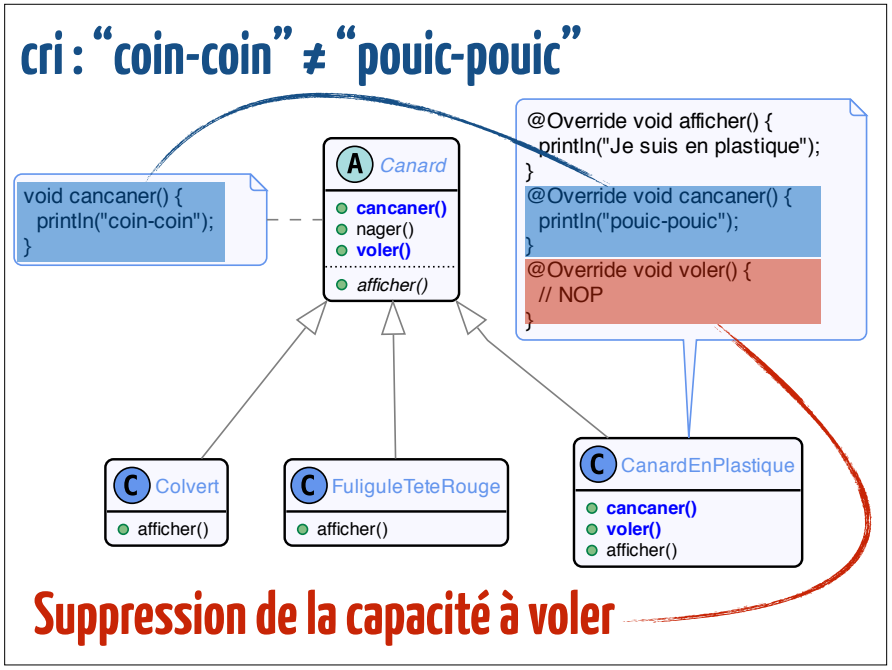
8


Canard
cancaner()
nager()
voler()
afficher()
Colvert
afficher()
FuliguleTeteRouge
afficher()

9


Et les canards en plastique ?

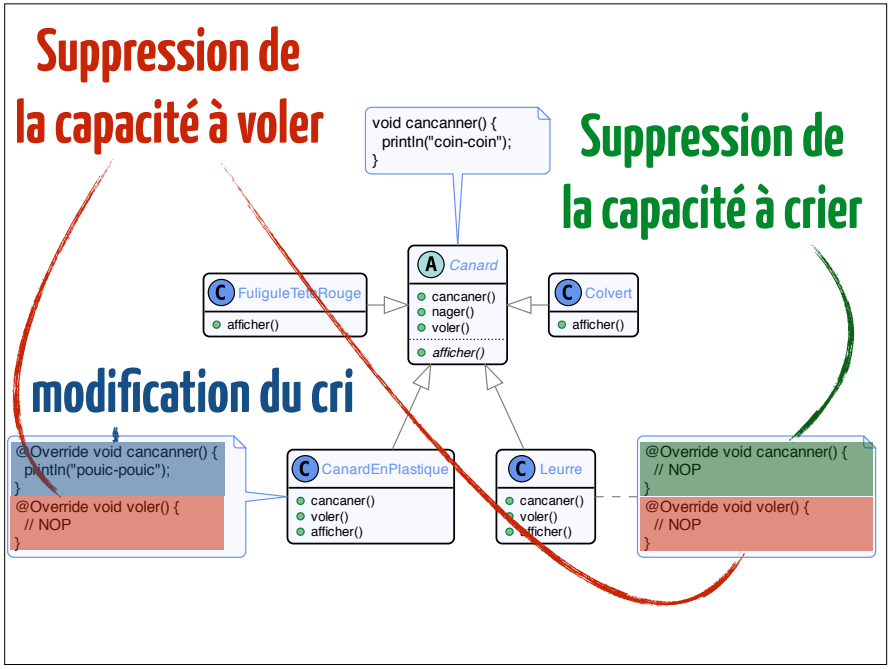
10


cri : “coin-coin” ≠ “pouic-pouic”
void cancaner() {
println("coin-coin");
}
Canard
cancaner()
nager()
voler()
afficher()
@Override void afficher() {
println("Je suis en plastique");
}
@Override void cancaner() {
println("pouic-pouic");
}
@Override void voler() {
// NOP
}
Colvert
afficher()
FuliguleTeteRouge
afficher()
CanardEnPlastique
cancaner()
voler()
afficher()
Suppression de la capacité à voler

11


Et les leurres ?

12



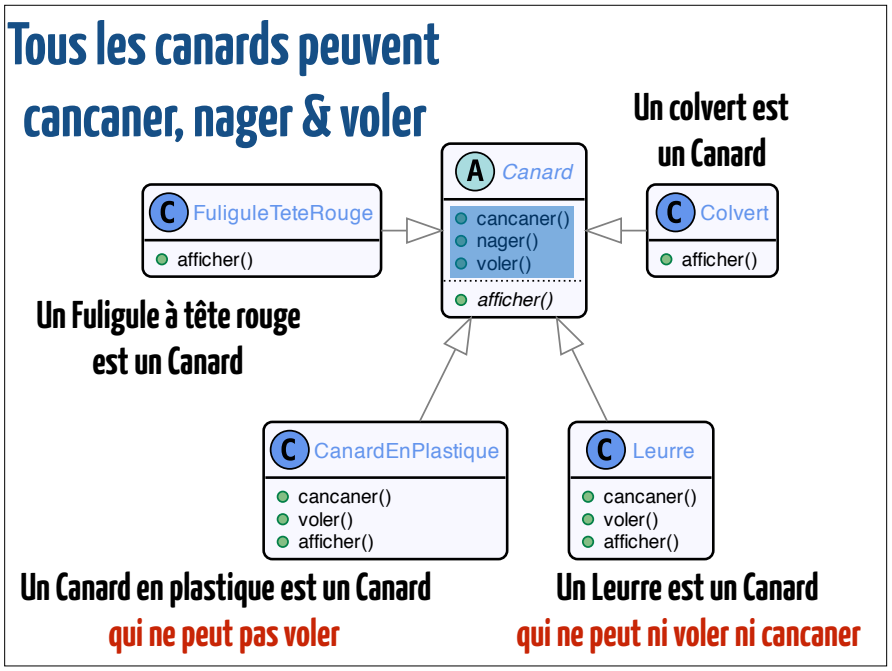
13

Est-ce raisonnable ?

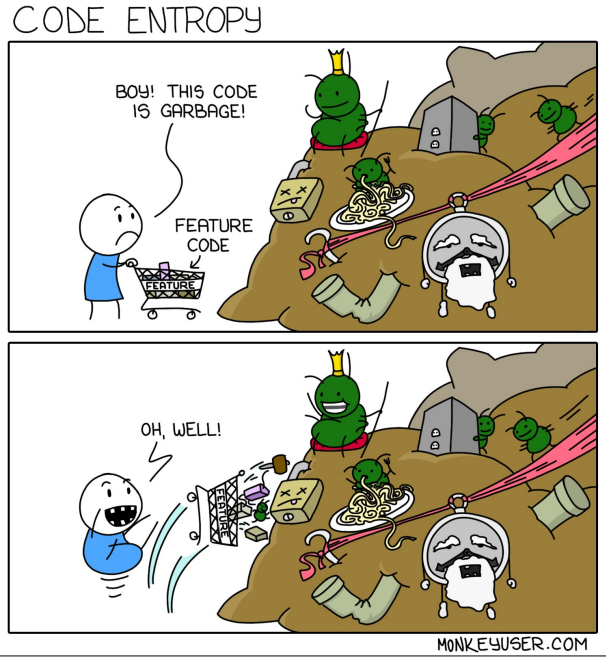
On respecte pourtant les piliers objets ...

Abstraction, Encapsulation,
Polymorhpisme & Héritage

14



15



16

Et si notre problème, c'était en fait notre solution ?

Bref, utiliser l'héritage est-il la bonne approche ?

17

Interface plutôt qu'implémentation

Design Principle

Program to an interface, not an implementation.

18

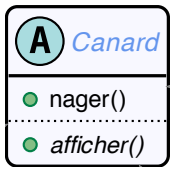
Trois Abstractions

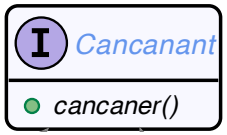
Structurelle :

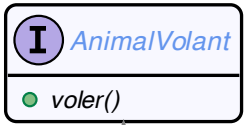
- **être un canard**

Fonctionnelle :

- **cancaner**
- **voler**



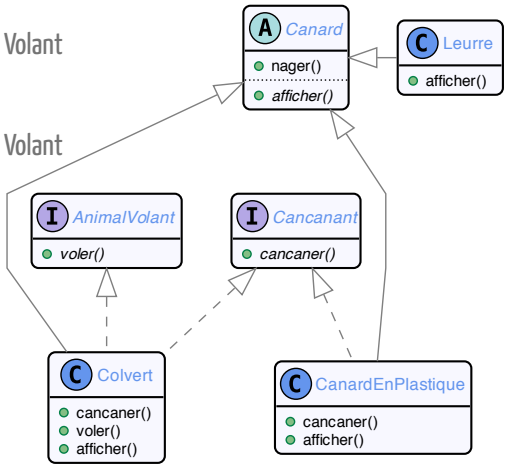




19

Création de canards à la carte !

- Colvert :
  - Canard ⊕ Cancanant ⊕ Animal Volant
- Fuligule à tête rouge :
  - Canard ⊕ Cancanant ⊕ Animal Volant
- Canard en plastique :
  - Canard ⊕ Cancanant
- Leurre :
  - Canard



20

Création de canards à la carte !

- Colvert :
  - Canard ⊕ Cancanant ⊕ Animal Volant
- Fuligule à tête rouge :
  - Canard ⊕ Cancanant ⊕ Animal Volant
- Canard en plastique :
  - Canard ⊕ Cancanant
- Leurre :
  - Canard

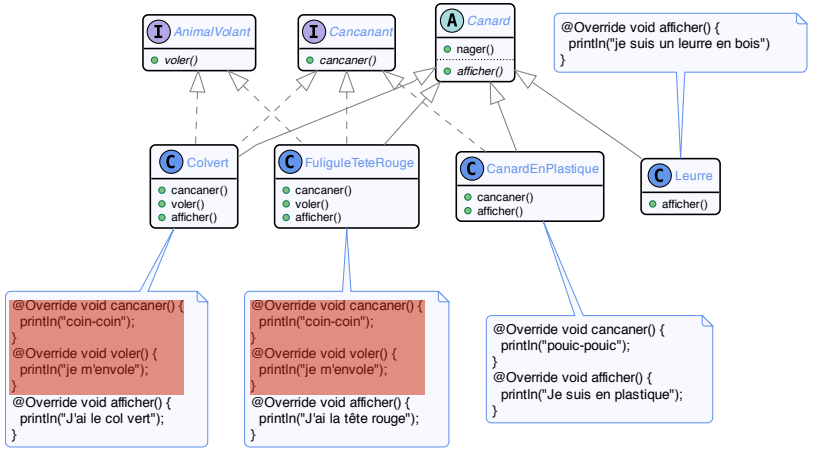
```
public class Colvert
  extends Canard
  implements AnimalVolant, Cancanant { ... }

public class CanardEnPlastique
  extends Canard
  implements Cancanant { ... }

public class Leurre
  extends Canard { ... }
```

21

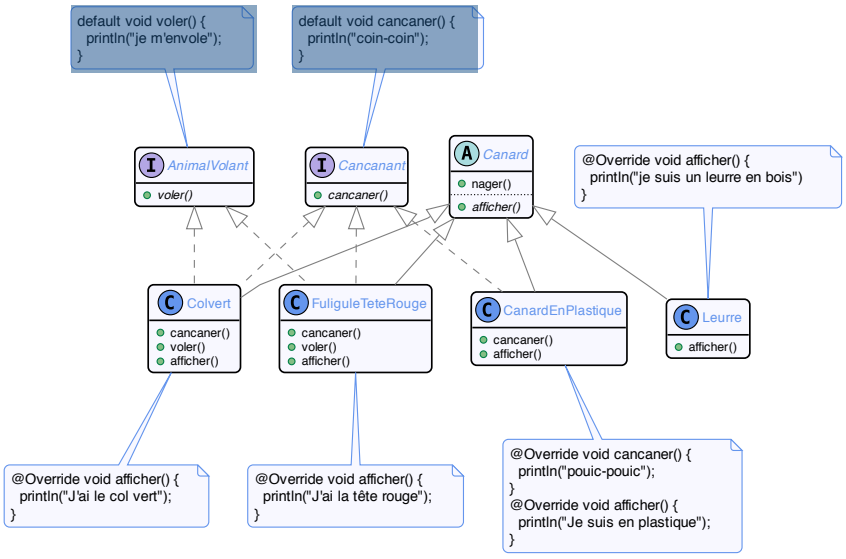
Au bout du compte ...



DRY: Don't Repeat Yourself

22

Java 8 propose des implémentation par défaut



23

On aurait pas juste déplacé le problème ?

```
Colvert colvert = new Colvert();
c.voler();

Canard unCanard = (Canard) new Colvert();
unCanard.voler() // Ne compile pas !!

if (unCanard instanceOf AnimalVolant) {
  ((AnimalVolant) unCanard).voler();
}
```

24

25

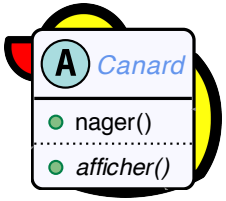
Identifier ce qui varie

Design Principle

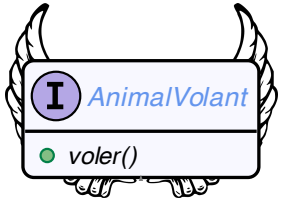
Identify the aspects of your application that vary and separate them from what stays the same.

26

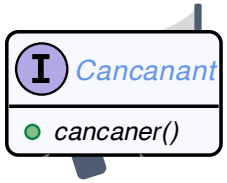
Wait, What ?



Être un Canard



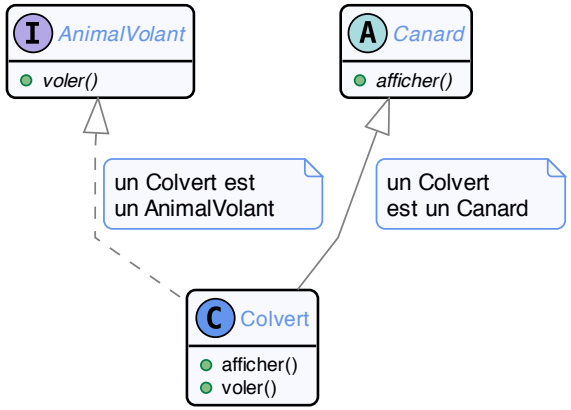
Voler



Faire du bruit

27

(Mauvaise) version avec des interfaces



Généralisation & Réalisation impliquent Sous-Typage

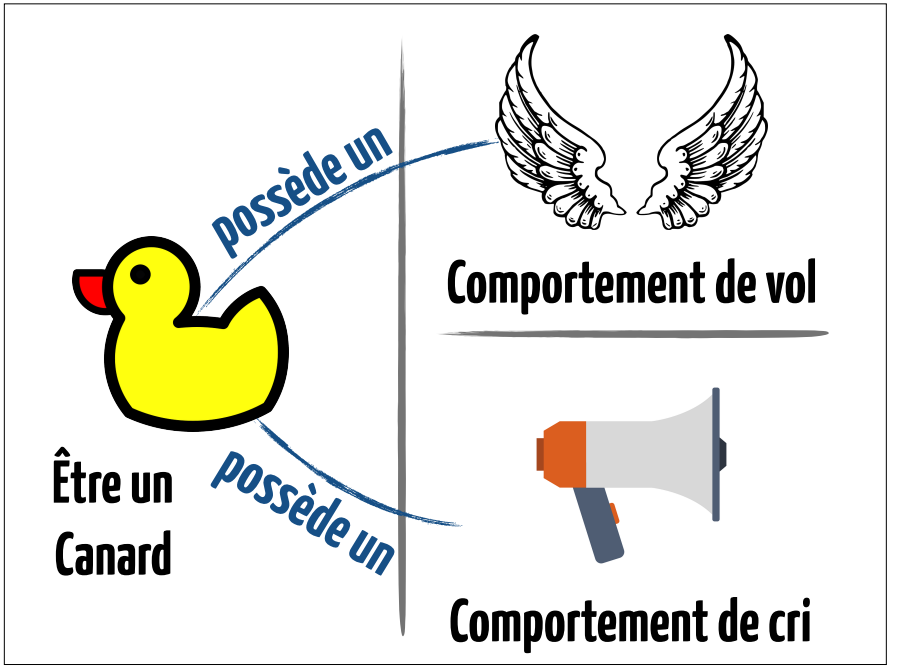
28

Composition vs Héritage

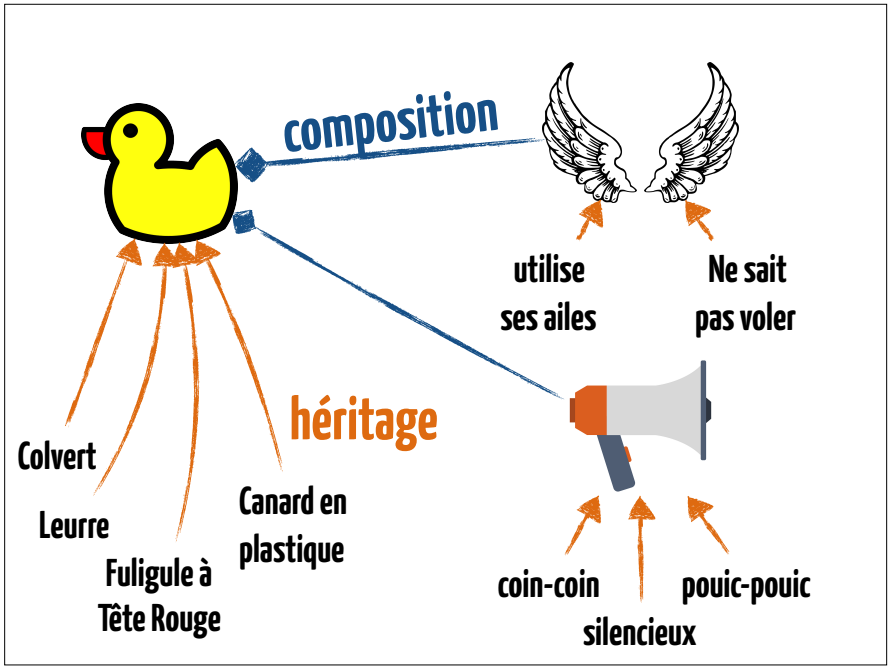
Design Principle

Favor composition over inheritance.

29

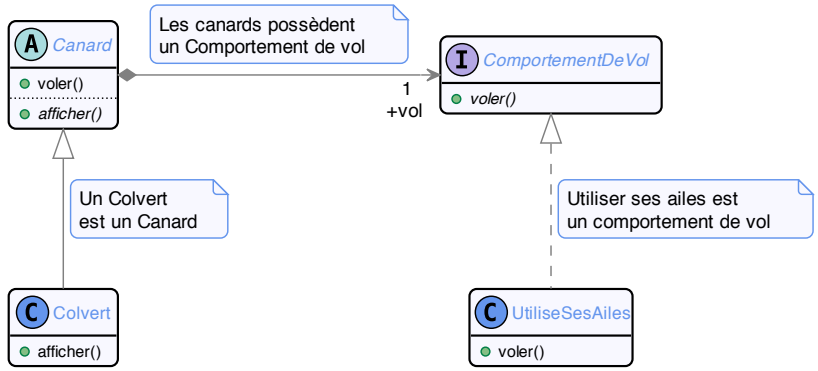


30



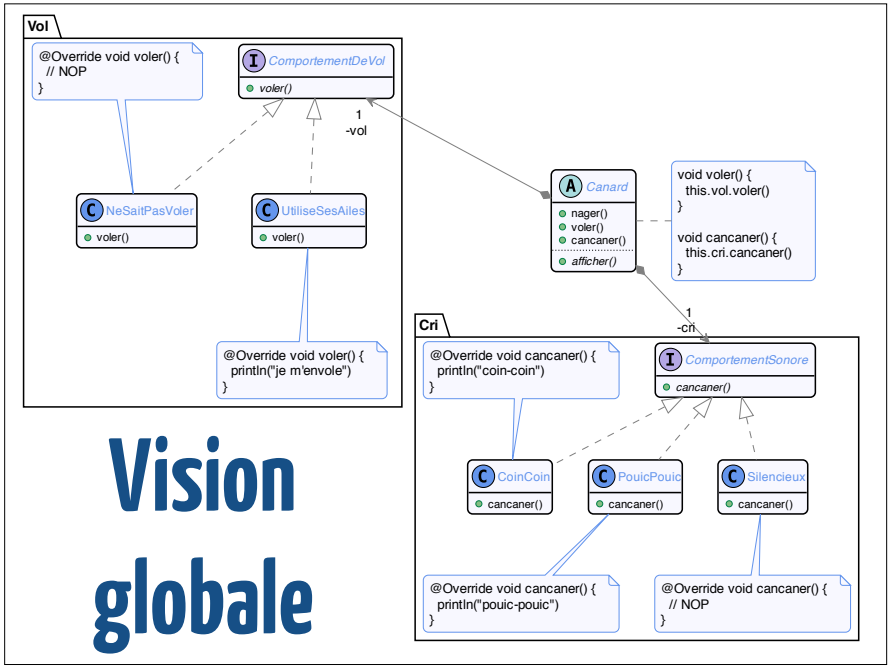
31

```
public abstract class Canard {
  private ComportementDeVol vol;
  public void voler() { this.vol.voler(); }
}
```

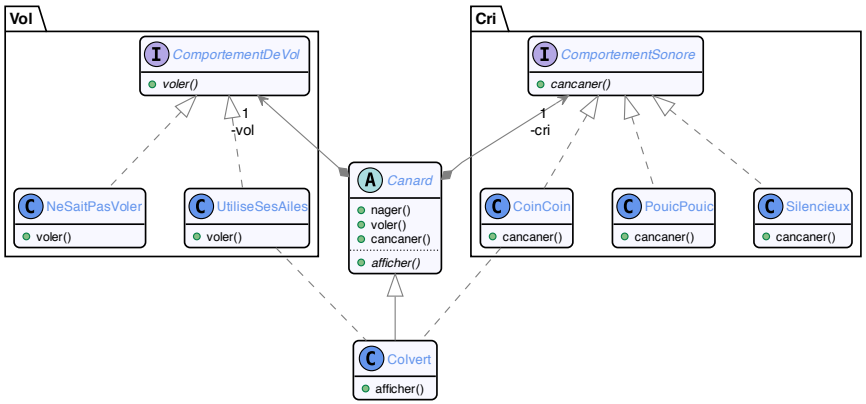



```
public class Colvert extends Canard {
  public Colvert() { this.vol = new UtiliseSesAiles(); }
}
```

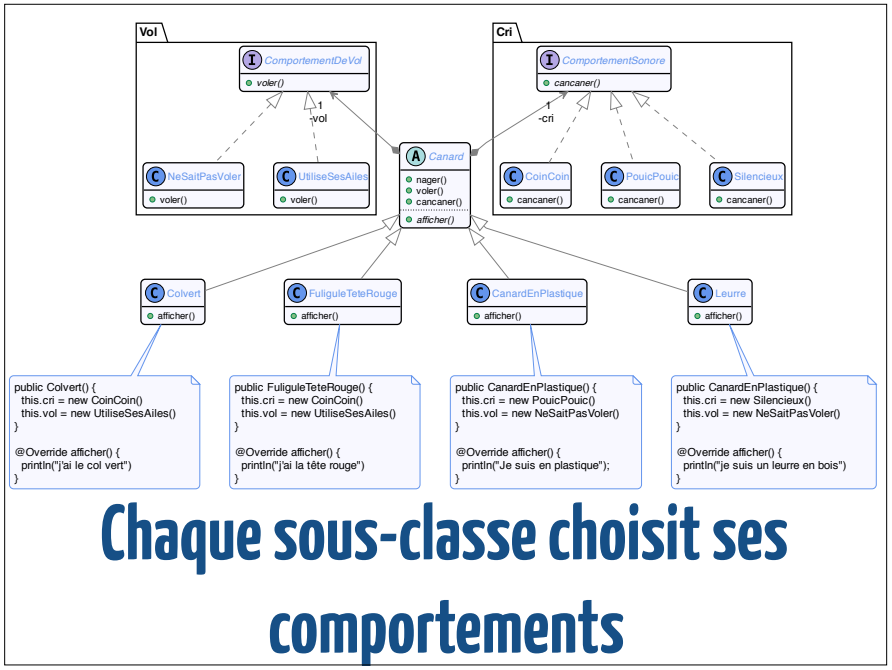
32


Vol
@Override void voler() {
// NOP
}
ComportementDeVol
voler()
1
-vol
NeSaitPasVoler
voler()
UtiliseSesAiles
voler()
@Override void voler() {
println("je m'envole")
}
Canard
nager()
voler()
cancaner()
afficher()
void voler() {
this.vol.voler()
}
void cancaner() {
this.cri.cancaner()
}
1
-cri
Cri
@Override void cancaner() {
println("coin-coin")
}
ComportementSonore
cancaner()
CoinCoin
cancaner()
PouicPouic
cancaner()
Silencieux
cancaner()
@Override void cancaner() {
println("pouic-pouic")
}
@Override void cancaner() {
// NOP
}
Vision globale

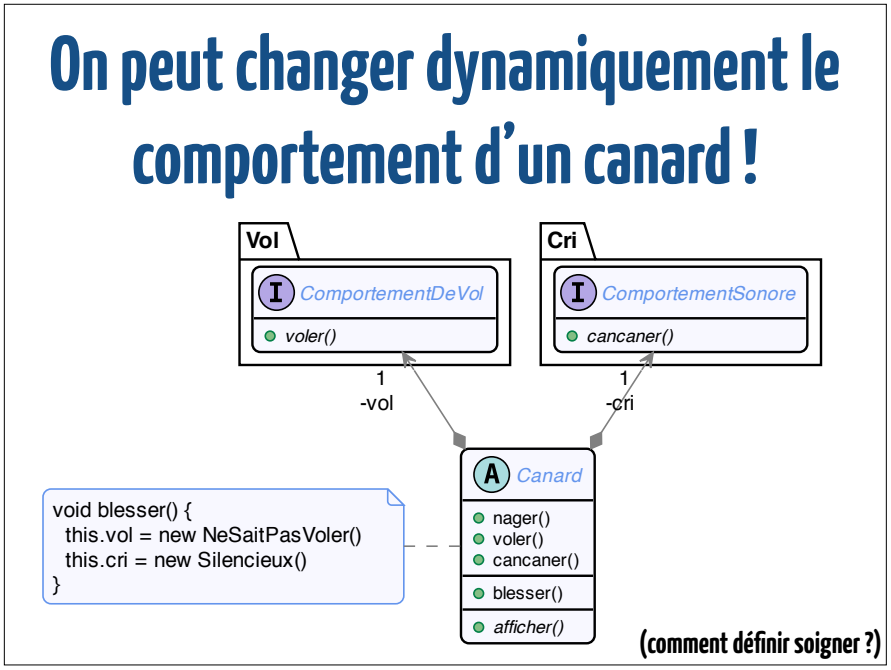
33


Assemblage "a la carte"
Vol
ComportementDeVol
voler()
1
-vol
NeSaitPasVoler
voler()
UtiliseSesAiles
voler()
Canard
nager()
voler()
cancaner()
afficher()
Cri
ComportementSonore
cancaner()
1
-cri
CoinCoin
cancaner()
PouicPouic
cancaner()
Silencieux
cancaner()
Colvert
afficher()

34

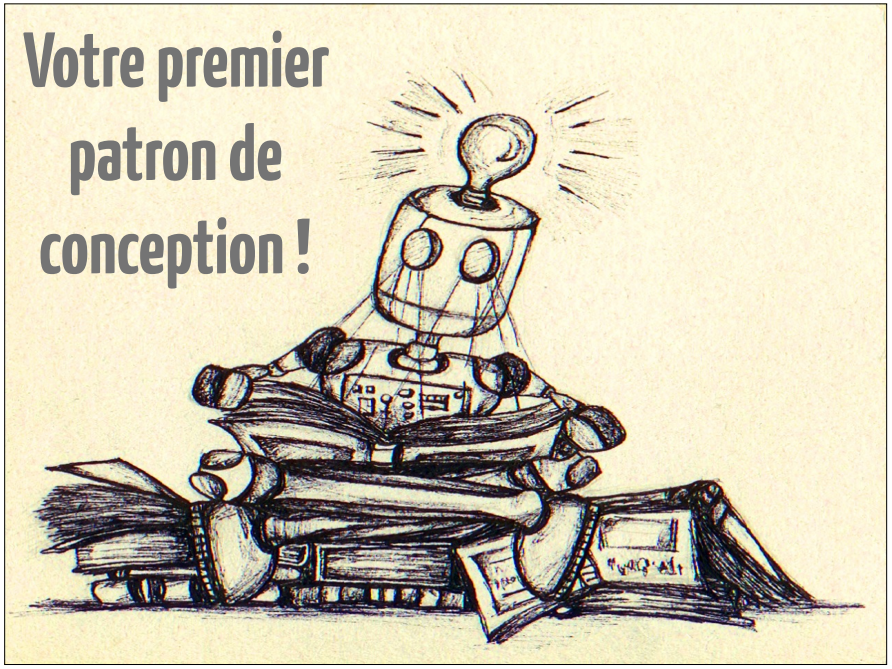

Vol
ComportementDeVol
voler()
1
-vol
NeSaitPasVoler
voler()
UtiliseSesAiles
voler()
Cri
ComportementSonore
cancaner()
1
-cri
CoinCoin
cancaner()
PouicPouic
cancaner()
Silencieux
cancaner()
Canard
nager()
voler()
cancaner()
afficher()
Colvert
afficher()
FuligineTeteRouge
afficher()
CanardEnPlastique
afficher()
Leurre
afficher()
public Colvert() {
this.cri = new CoinCoin()
this.vol = new UtiliseSesAiles()
}
@Override afficher() {
println("j'ai le col vert")
}
public FuligineTeteRouge() {
this.cri = new CoinCoin()
this.vol = new UtiliseSesAiles()
}
@Override afficher() {
println("j'ai la tête rouge")
}
public CanardEnPlastique() {
this.cri = new PouicPouic()
this.vol = new NeSaitPasVoler()
}
@Override afficher() {
println("Je suis en plastique");
}
public CanardEnPlastique() {
this.cri = new Silencieux()
this.vol = new NeSaitPasVoler()
}
@Override afficher() {
println("je suis un leurre en bois")
}
Chaque sous-classe choisit ses comportements

35


On peut changer dynamiquement le comportement d'un canard !
Vol
ComportementDeVol
voler()
1
-vol
Cri
ComportementSonore
cancaner()
1
-cri
Canard
nager()
voler()
cancaner()
blesser()
afficher()
void blesser() {
this.vol = new NeSaitPasVoler()
this.cri = new Silencieux()
}
(comment définir soigner ?)

36

Votre premier patron de conception !



37

STRATEGY — Object Behavioral

Intent

Define a family of algorithms, encapsulate each one, and make them interchangeable. Strategy lets the algorithm vary independently from clients that use it.

Also Known As

Policy

Motivation

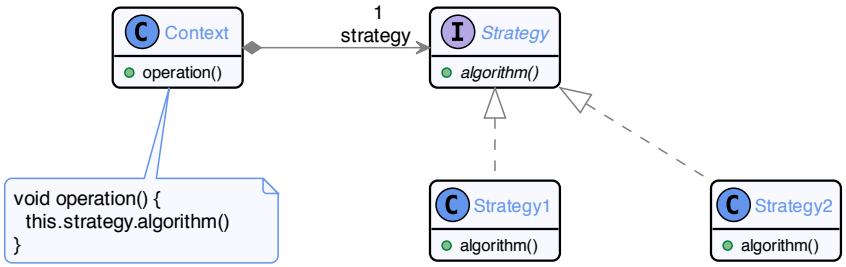
Many algorithms exist for breaking a stream of text into lines. Hard-wiring all such algorithms into the classes that require them isn't desirable for several reasons:

- Clients that need linebreaking get more complex if they include the linebreaking code. That makes clients bigger and harder to maintain, especially if they support multiple linebreaking algorithms.
- Different algorithms will be appropriate at different times. We don't want to support multiple linebreaking algorithms if we don't use them all.
- It's difficult to add new algorithms and vary existing ones when linebreaking is an integral part of a client.

We can avoid these problems by defining classes that encapsulate different linebreaking algorithms. An algorithm that's encapsulated in this way is called a **strategy**.

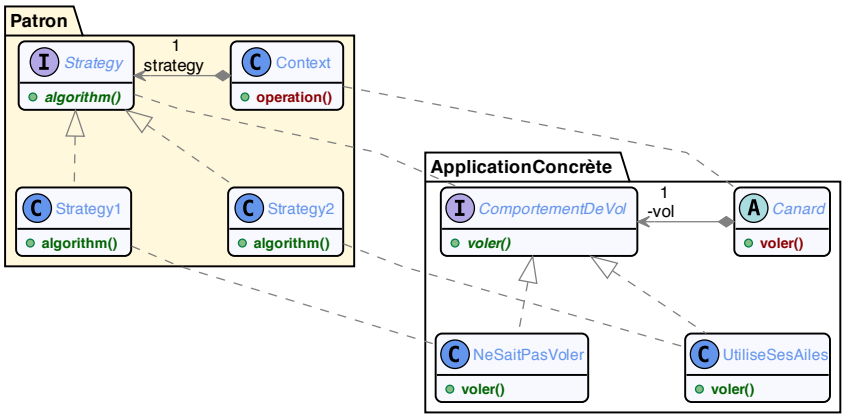
38

Le patron "Stratégie"



39

Des Canards Stratèges !



40

Principes & Classification

2

41

Principe des patrons de conceptions

- Standardisation des concepts de modélisation ;
- Capturer l'experience de conception ;
- Réutiliser des solutions élégantes & efficace pour des problèmes récurrents ;
- Améliorer la documentation ;
- Faciliter la maintenance.

Le cri des canards ? C'est une "Stratégie"

42

Capturer l'experience de conception

Stratégie {

Design Principle

Favor composition over inheritance.

Design Principle

Identify the aspects of your application that vary and separate them from what stays the same.

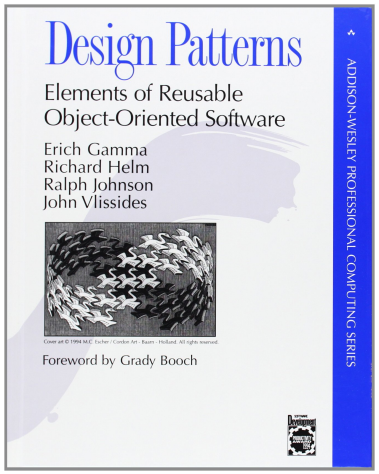
Design Principle

Program to an interface, not an implementation.

43

Historique

- Livre "fondateur" publié en 1994
- Les auteurs sont le "Gang des Quatres"
- Proposition de 23 patrons de conceptions
- Séparation en 3 familles :
  - "Créationels"
  - Structurels
  - Comportementaux



44

Définition

Un patron de conception nomme, abstrait et identifie les aspects essentiels d'une structuration récurrente, ce qui permet de créer une modélisation orientée objet réutilisable

Dimension de classification

		Objectif		
		Création	Structure	Comportement
Portée	classe			
	objet			**Stratégie**

		Objectif		
		Création	Structure	Comportement
Portée	classe	Factory	Adapter	Interpreter Template Method
	objet	Abstract Factory Builder Prototype Singleton	Adapter Bridge Composite Decorator Facade Flyweight Proxy	Chain of Responsibility Command Iterator Mediator Memento Observer State Strategy Visitor

INF-5153

		Objectif		
		Création	Structure	Comportement
Portée	classe	**Factory**	**Adapter**	~~Interpreter~~ **Template Method**
	objet	**Abstract Factory** **Builder** **Prototype** **Singleton**	**Adapter** ~~Bridge~~ **Composite** **Decorator** **Facade** ~~Flyweight~~ **Proxy**	~~Chain of Responsibility~~ **Command** ~~Iterator~~ ~~Mediator~~ ~~Memento~~ **Observer** **State** **Strategy** **Visitor**

Retours sur le TP1

3

49

50