



Le génie pour l'industrie

LOG121

Conception orientée objet

Patron Décorateur
Application des patrons

Enseignante: Souad Hadjres

- Patron Décorateur
- Comment reconnaître les patrons?
- Application des patrons

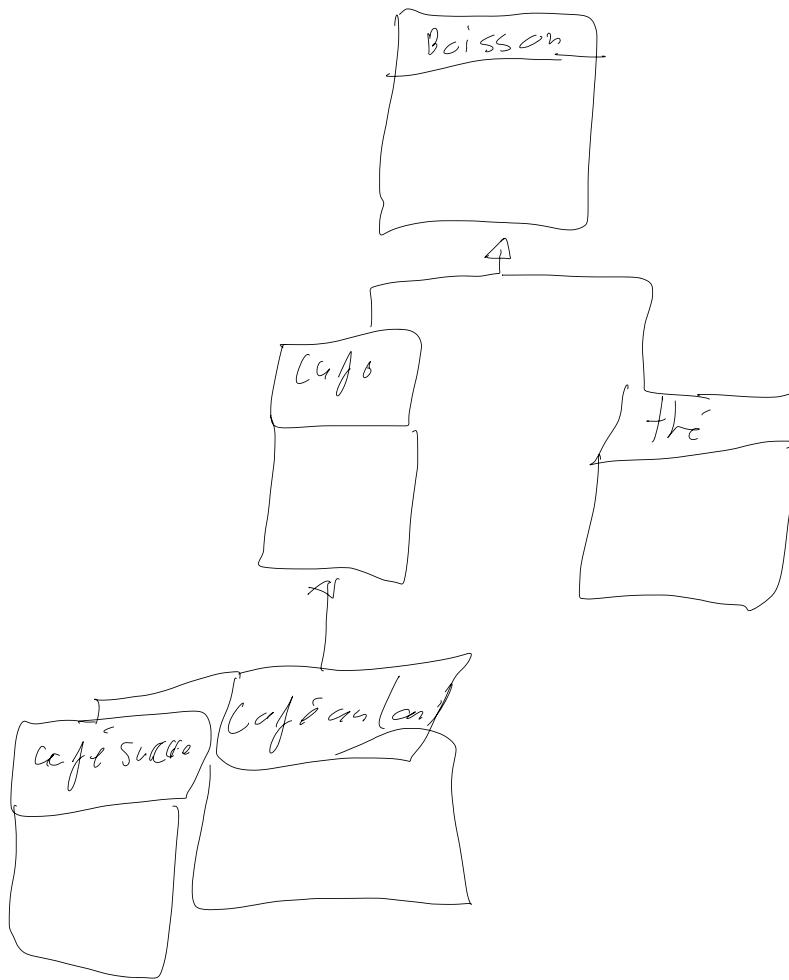
Exemple de problème de conception

3

□ Welcome to Starbuzz Coffee

- ▣ Une compagnie qui a grossi et qui a de plus en plus de succursales
- ▣ Elle a besoin de s'adapter aux commandes variées de ses clients
- ▣ Elle doit mettre à jour son système de commandes des boissons

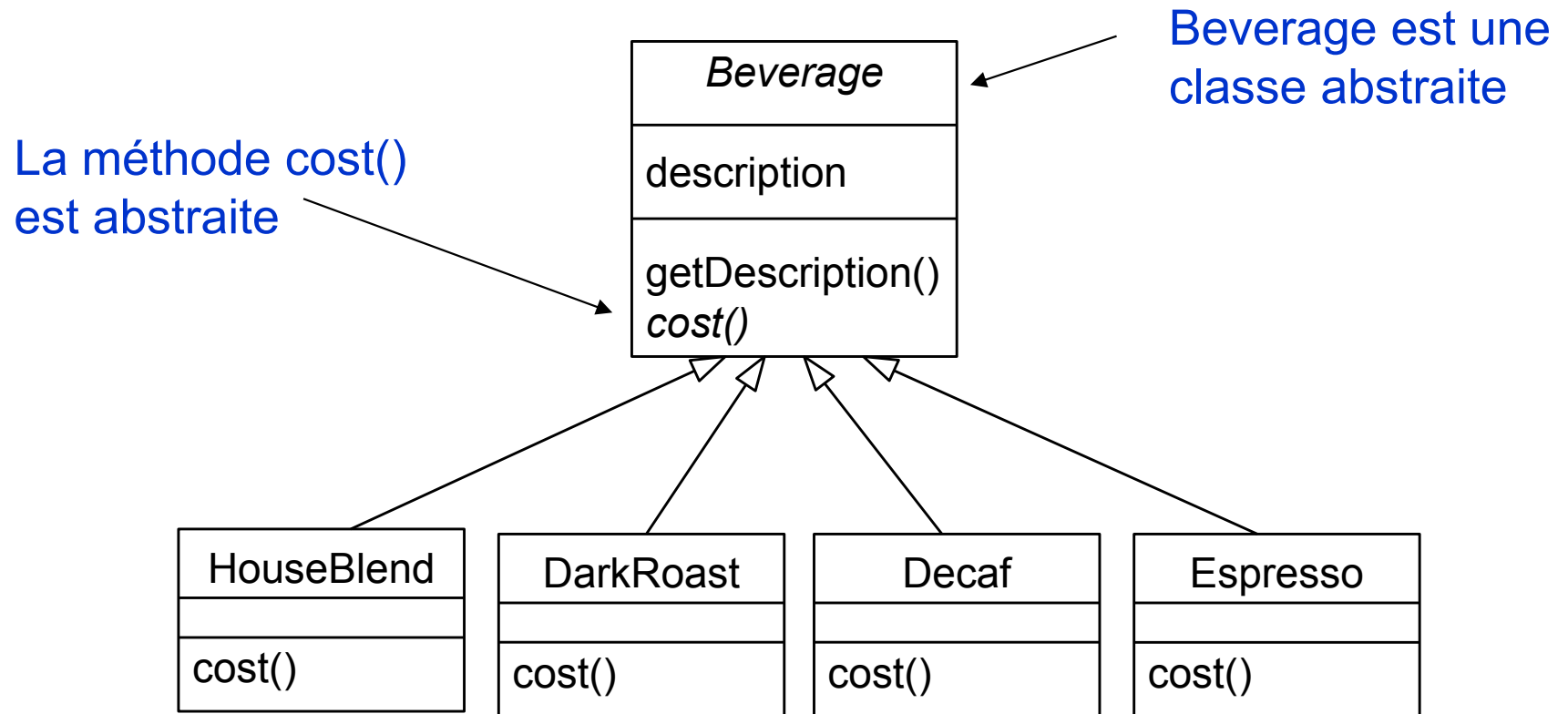




Exemple de problème de conception

4

□ La conception initiale



Chaque sous classe implémente la méthode `cost()` qui calcule le prix de la boisson

Exemple de problème de conception

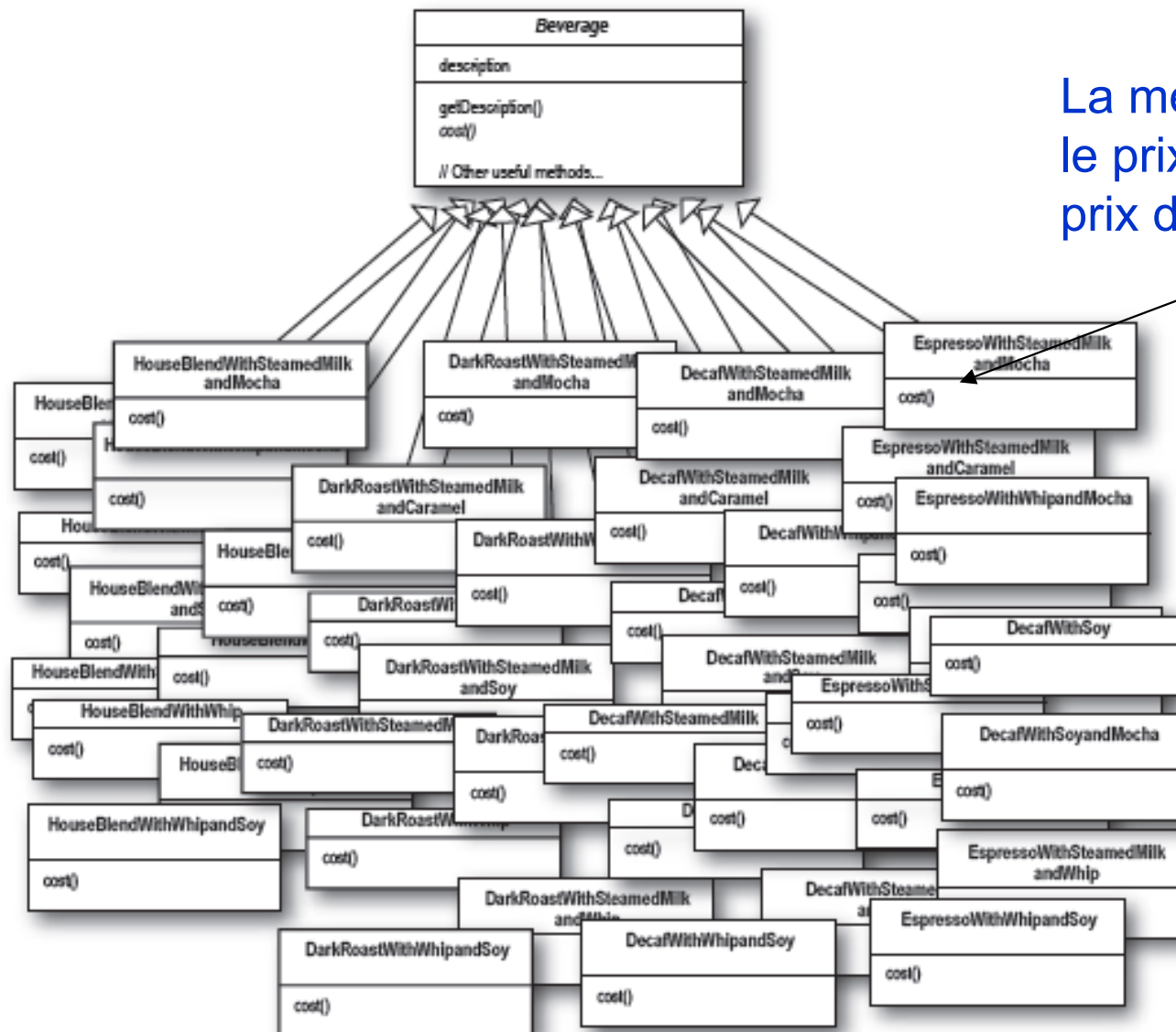
5

- Les clients peuvent ajouter plusieurs condiments à leur café
 - ▣ Mocha (chocolat)
 - ▣ Soya
 - ▣ Lait mousseux
 - ▣ Crème fouettée
- Starbuzz charge un prix supplémentaire pour chaque condiment
 - ▣ Elle veut que ce soit fait par le système

Exemple de problème de conception

6

□ Une première tentative



La méthode `cost()` calcule le prix du café avec le prix des condiments

Exemple de problème de conception

7

- Les problèmes avec cette conception
 - ▣ Explosion du nombre de classes
 - ▣ Le choix des condiments se fait de façon statique
 - ▣ Une maintenance difficile
- Comment améliorer cette conception?

Solution au problème

8



- Utiliser la composition au lieu de l'héritage
 - Créer un objet instance de la classe de boisson
 - HouseBlend, Decaf
 - Enrober l'objet boisson par un autre objet qui ajoute un condiment
 - On appelle l'objet qui enrobe: le décorateur
 - L'objet qui enrobe peut être encore enrobé pour ajouter d'autres condiments
- Utiliser la délégation pour la méthode cost() qui calcule le prix

Solution au problème



9

- Si par exemple, nous enrobions une instance de la boisson DarkRoast par le condiment Moka et on enrobe le tout par le condiment Milk

2. BeverageWithMilk appelle la méthode cost() de BeverageWithMocha

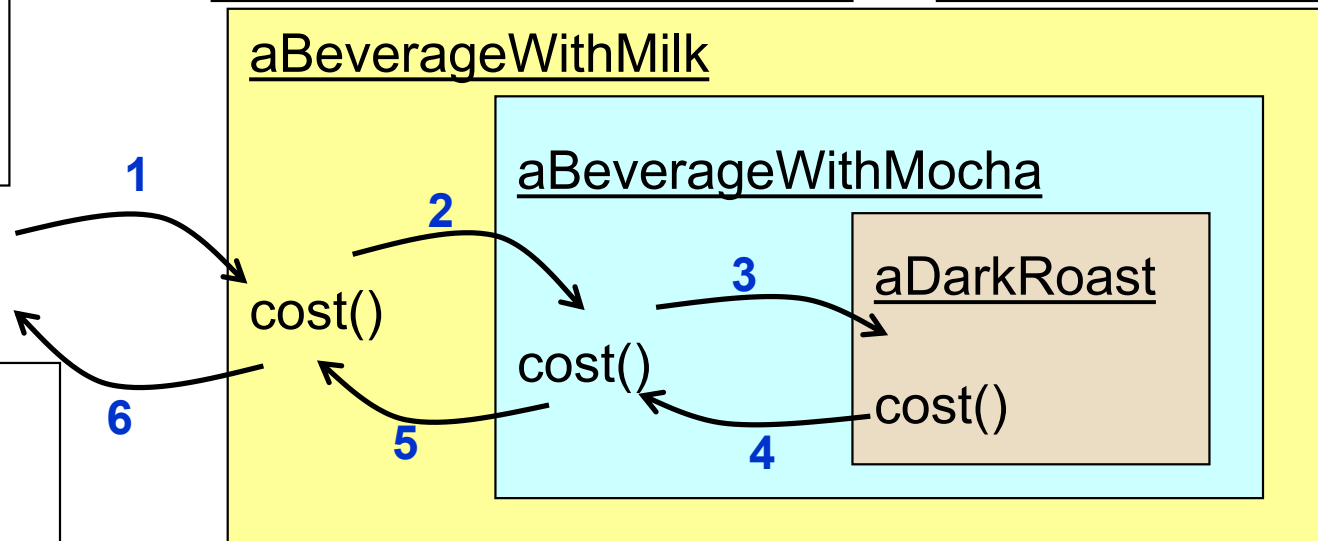
3. BeverageWithMocha appelle la méthode cost() de DarkRoast

1. On appelle la méthode cost() du décorateur externe

6. BeverageWithMilk ajoute son prix au résultat retourné par BeverageWithMocha et retourne le prix final

5. BeverageWithMocha ajoute son prix au résultat retourné par DarkRoast et retourne le nouveau prix

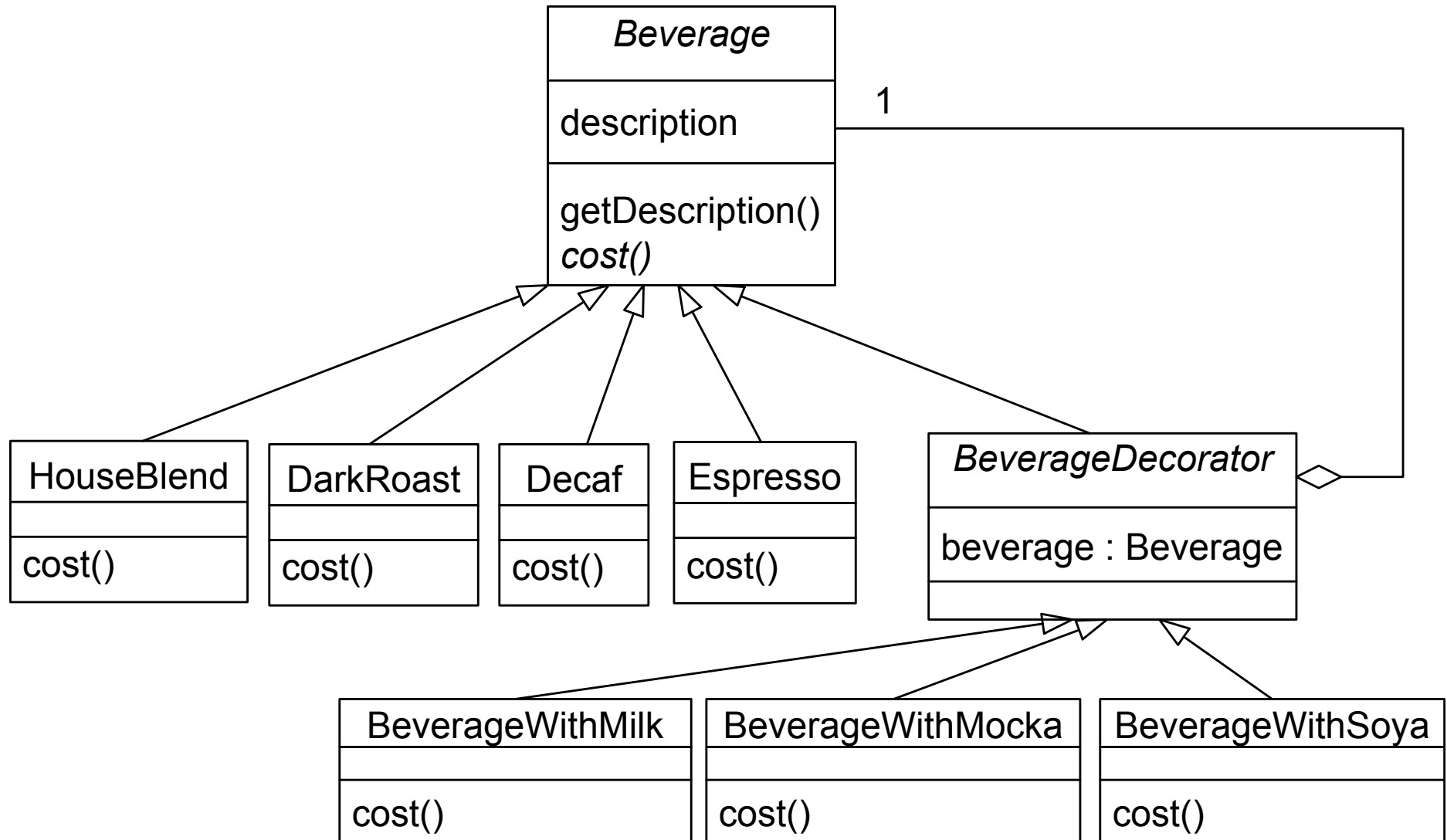
4. DarkRoast retourne son prix



Le patron Décorateur



10



Le patron Décorateur



11

- Les décorateurs de boissons ont la même interface que les boissons
 - Ils peuvent être utilisés à la place des boissons par les classes clientes
- On ajoute un comportement par composition et non pas par héritage
- La composition nous donne la flexibilité de mixer les condiments et les boissons comme on veut à *l'exécution*
- On peut facilement ajouter de nouveaux condiments par l'ajout de nouveaux décorateurs

```
public abstract class Beverage {
    String description;

    public String getDescription() {
        return description;
    }
    public abstract double cost();
}
```

```
public class Espresso extends Beverage{
    public Espresso() {
        description ="Espresso Coffee";
    }
    public double cost() {
        return 1.99;
    }
}
```

```
public class HouseBlend extends Beverage{
    public HouseBlend() {
        description ="House Blend Coffee";
    }
    public double cost() {
        return 0.89;
    }
}
```

```
public abstract class CondimentDecorator extends Beverage{
    public abstract String getDescription();
}
```

```
public class Moka extends CondimentDecorator{
    Beverage beverage;
    public Moka(Beverage beverage) {
        this.beverage = beverage;
    }
    public String getDescription() {
        return beverage.description + ", Moka";
    }
    public double cost() {
        return .20 + beverage.cost();
    }
}
```

```
public class Whip extends CondimentDecorator{
    Beverage beverage;
    public Whip(Beverage beverage) {
        this.beverage = beverage;
    }
    public String getDescription() {
        return beverage.description + ", Whip";
    }
    public double cost() {
        return .10 + beverage.cost();
    }
}
```

```
public class StarBuzzCoffee {  
  
    public static void main(String[] args) {  
        Beverage beverage = new Espresso();  
        System.out.println(beverage.getDescription() + " $ " + beverage.cost());  
  
        Beverage boisson = new HouseBlend();  
        System.out.println(boisson.getDescription() + " $ " + boisson.cost());  
  
        boisson = new Moka(boisson);  
        System.out.println(boisson.getDescription() + " $ " + boisson.cost());  
  
        boisson = new Moka(boisson);  
        System.out.println(boisson.getDescription() + " $ " + boisson.cost());  
  
        boisson = new Whip(boisson);  
        System.out.println(boisson.getDescription() + " $ " + boisson.cost());  
    }  
}
```

□ Contexte

- On veut ajouter des décorations (des comportements) à un objet
- Un objet décoré doit être utilisé de la même façon que l'objet non décoré
- La classe des objets à décorer ne veut pas prendre la responsabilité de faire la décoration
- Il peut y avoir un nombre illimité de possibilités de combiner les décorations

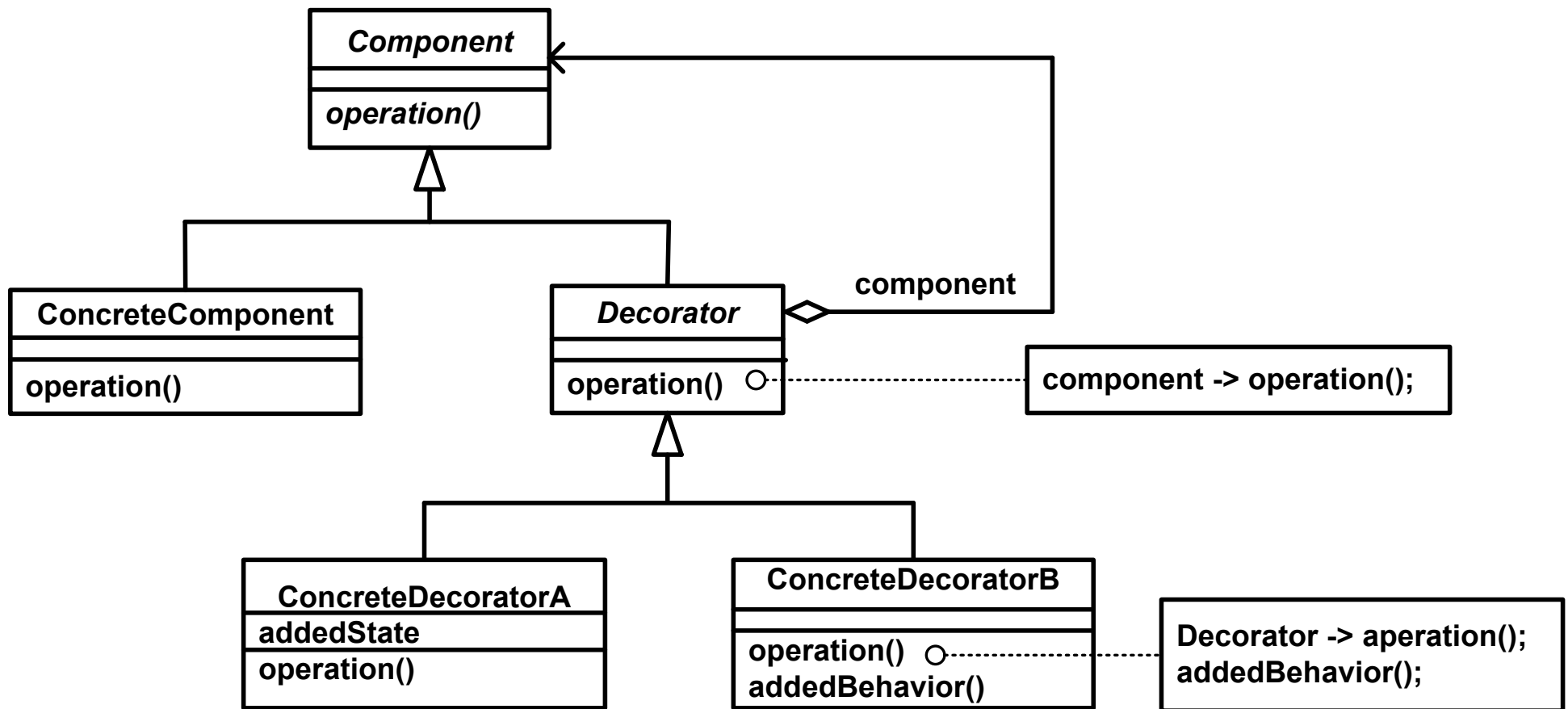
□ Solution

- ▣ Définir une interface commune «Component» aux classes des composants concrets et aux classes décoratrices
- ▣ Un objet décorateur gère l'objet composant auquel il ajoute une décoration
- ▣ Dans son implémentation d'une méthode de l'interface «Component», une classe décoratrice applique la méthode à l'objet pour lequel elle fait la décoration et elle combine le résultat avec l'effet de la décoration

Le patron Décorateur

16

□ La structure générique du patron dans GoF

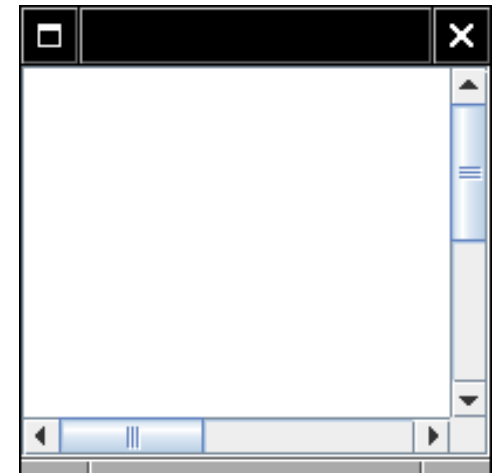


Le patron Décorateur

17

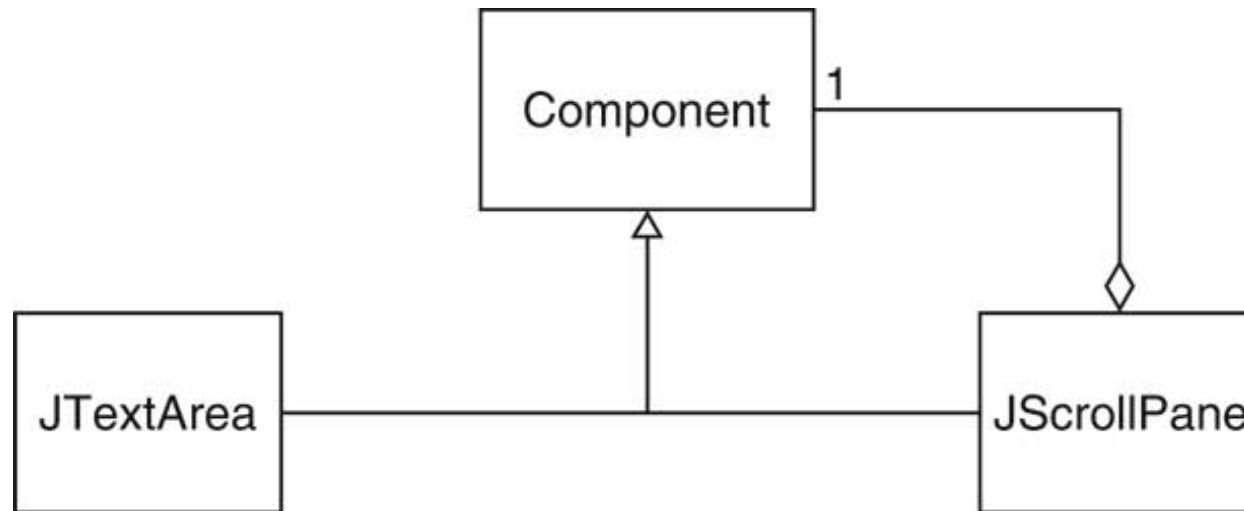
- Un autre exemple: Barres de défilement (Scroll Bars)
 - Des barres de défilement peuvent être attachées aux composants
 - Les barres de défilement entourent le composant

```
TextArea textArea = new TextArea(5, 30);  
ScrollPane scrollPane = new JScrollPane(textArea);
```
 - JScrollPane est aussi un composant



Le patron Décorateur

18



Nom dans le patron de conception	Vrai nom (barres de défilement)
Component	Component
ConcreteComponent	JTextArea
Decorator	JScrollPane
method()	une méthode de Component (exemple: paint)

□ Un autre exemple: Flux de données (Streams)

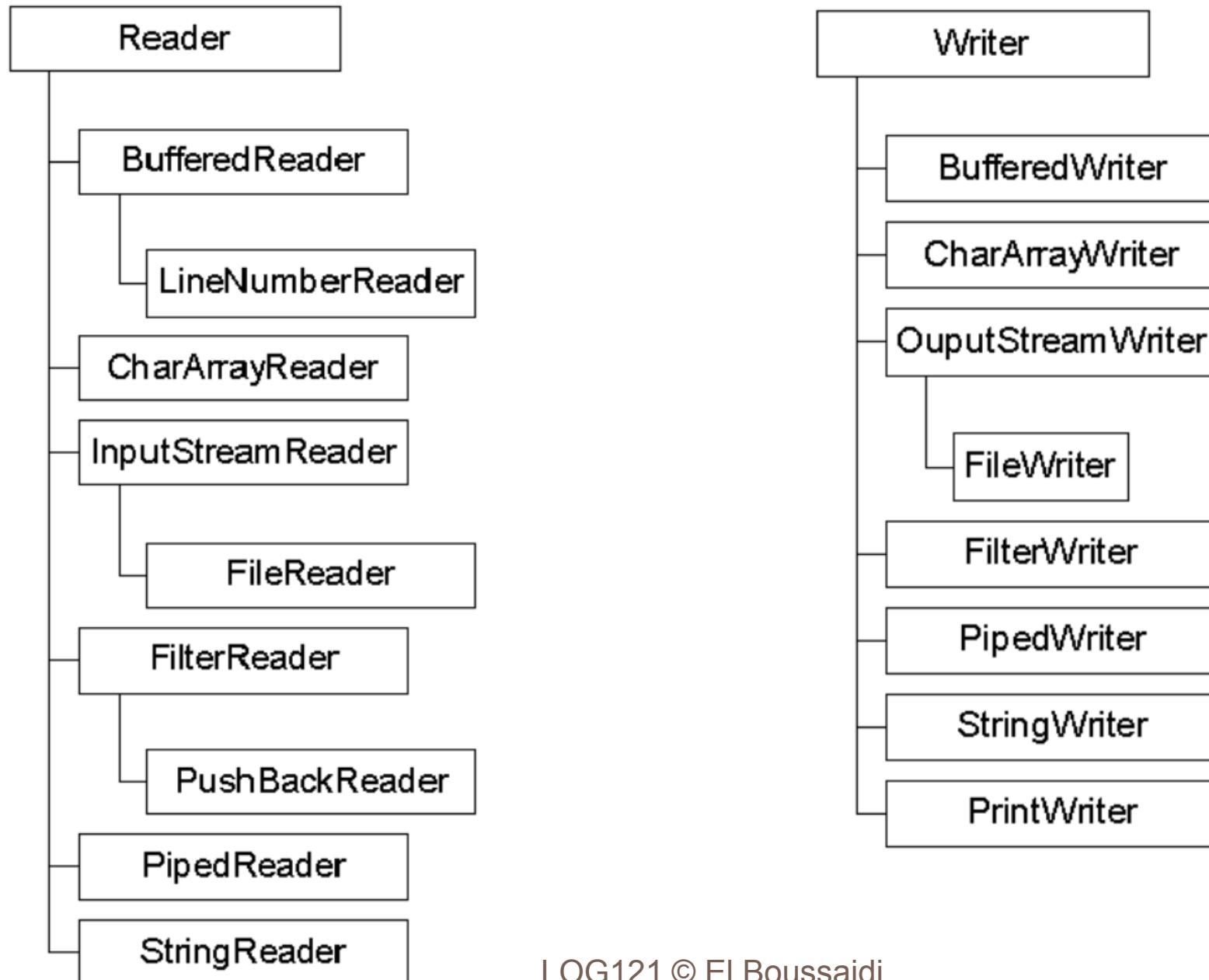
```
InputStreamReader reader = new InputStreamReader(System.in);
```

```
BufferedReader console = new BufferedReader(reader);
```

- `BufferedReader` enveloppe un `Reader` pour lui ajouter l'effet de tamponnage (c.à.d. un buffer)
 - Il lit le texte et met les mots dans le buffer
- `BufferedReader` est aussi un `Reader`
- Plusieurs autres décorateurs dans la bibliothèque des flux de données (streams) p. ex. `BufferedWriter`

Le patron Décorateur

20

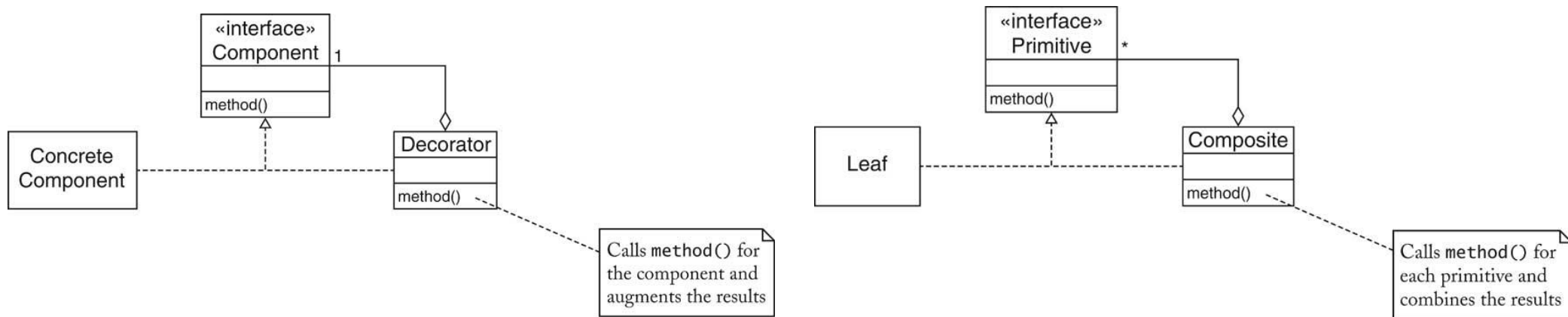


- Patron Décorateur
- Comment reconnaître les patrons?
- Application des patrons

Comment reconnaître les patrons

22

- Les patrons Décorateur et Composite ont des structures qui se ressemblent mais des intentions différentes
 - ▣ Le décorateur permet d'ajouter un comportement à un composant
 - ▣ Le composite permet de regrouper un ensemble de composants et traiter cet ensemble comme un simple composant



Comment reconnaître les patrons

23

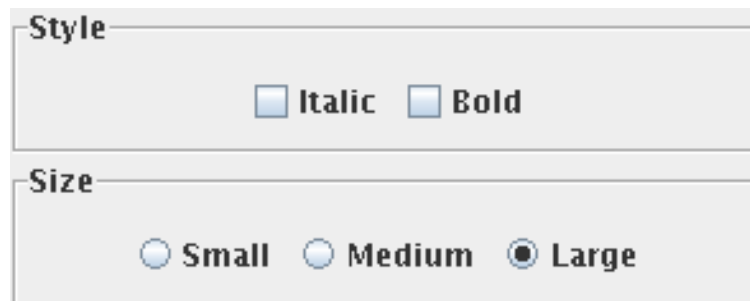
- Analyser l'intention du patron
 - ▣ Composite, Décorateur, Stratégie, Observateur et Itérateur ont des objectifs différents
- Se rappeler des utilisations fréquentes et des exemples d'application
 - ▣ Stratégie pour les gestionnaires de disposition (Layout managers)
 - ▣ Décorateur pour les barres de défilement
 - ▣ Composite pour les composants de l'interface graphique
 - ▣ Observateur pour les listeners associés aux composants de l'interface graphique

Comment reconnaître les patrons

24

- Utiliser les descriptions dans les parties **contexte** et **solution** d'un patron comme un test révélateur ("Litmus test")
 - ▣ Par exemple: On peut ajouter une bordure à un composant Swing

```
Border b = new EtchedBorder();  
component.setBorder(b);
```
 - ▣ Est-ce un exemple du patron Décorateur?



Style

☐ Italic ☐ Bold

Size

☐ Small ☐ Medium ☒ Large

Comment reconnaître les patrons

25

- Faisons le test sur les éléments dans la partie **contexte** du patron
 - ▣ On veut ajouter une décoration à des objets
 - OUI
 - ▣ L'objet décoré peut être utilisé de la même façon que l'objet non décoré
 - OUI
 - ▣ La classe composant ne veut pas prendre la responsabilité de faire la décoration
 - **NON**--la classe composant possède une méthode `setBorder`

- Patron Décorateur
- Comment reconnaître les patrons?
- Application des patrons

Appliquer les patrons

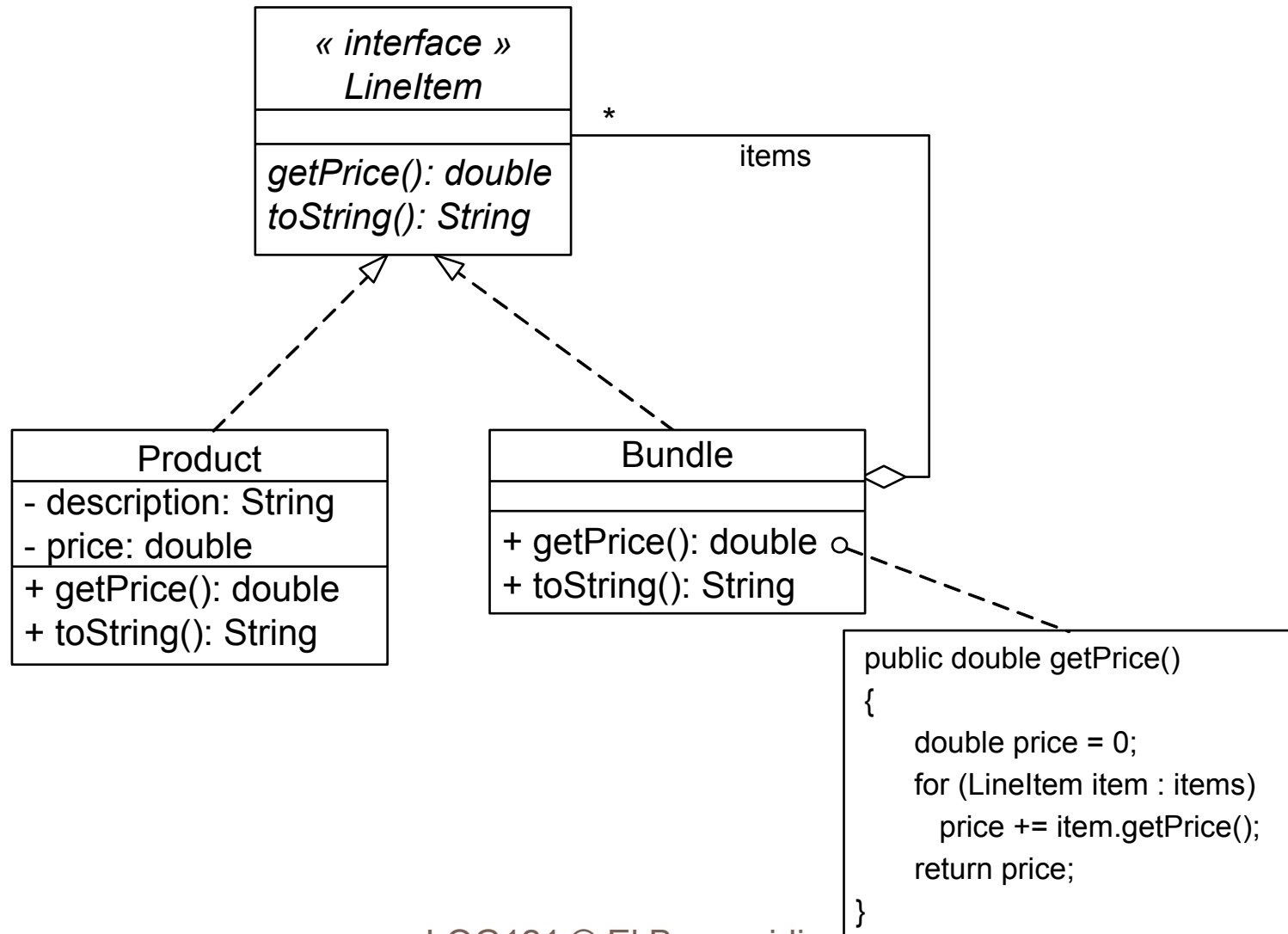
27

- Exemple tiré du manuel du cours (section 5.8):
 - ▣ Considérons une application qui génère les factures d'un magasin
 - ▣ Une facture (Invoice) contient plusieurs lignes d'articles (Line items).
 - ▣ Un « Lineltem » a une description et un prix.
 - ▣ Le Lineltem « Product » est l'item le plus simple.
 - ▣ Le Lineltem « Bundle » est composé d'un ensemble d'items reliés. Par exemple: un système de chaîne stéréo avec syntoniseur, amplificateur, CD et enceintes.
 - ▣ Un « Bundle » est aussi un Lineltem que l'on peut ajouter à une facture.

Appliquer les patrons

28

□ Exemple (suite): on applique le patron Composite

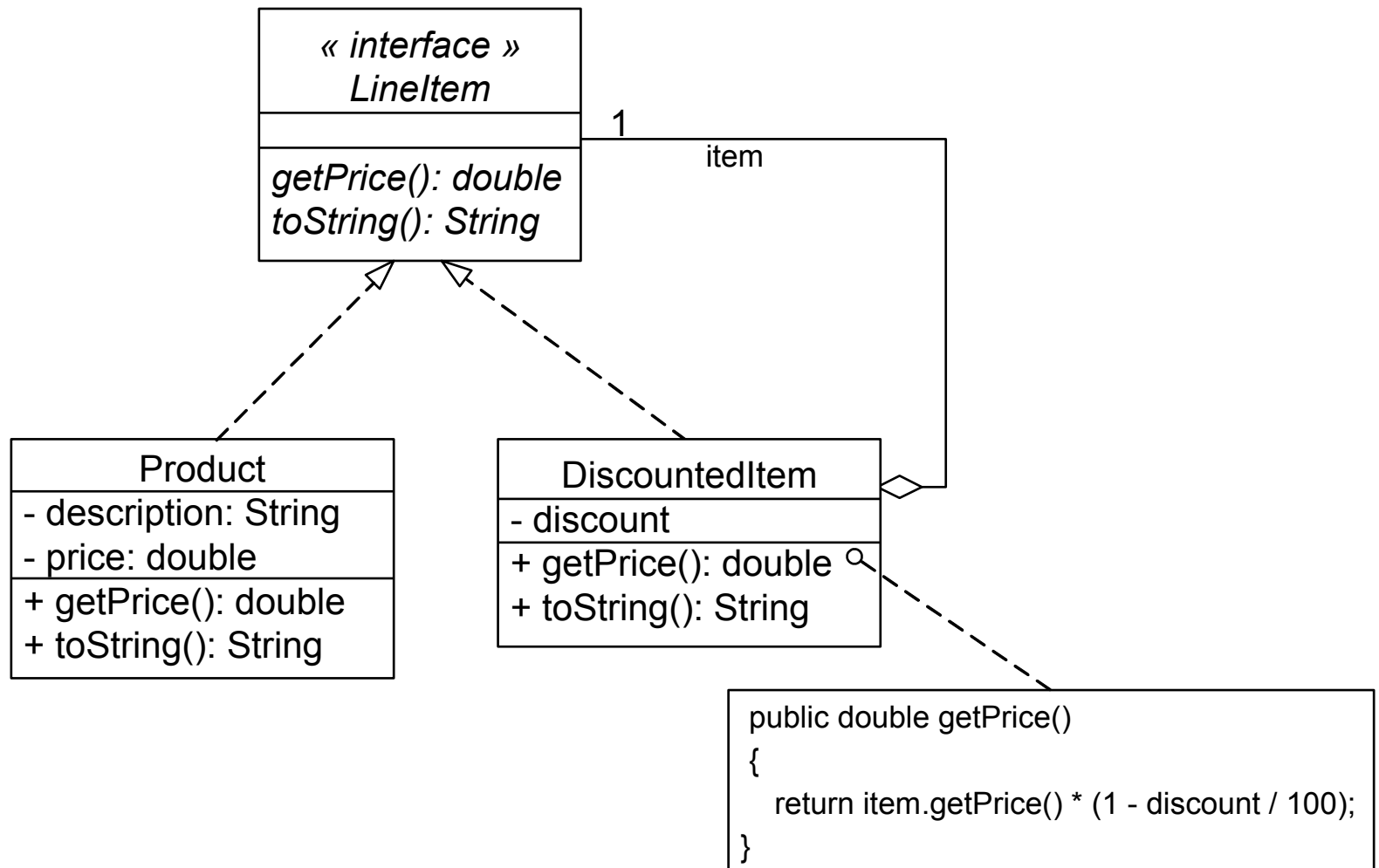


- Exemple (suite):
 - ▣ Le magasin peut offrir des articles à prix réduit (DiscountedItem).
 - ▣ Un « DiscountedItem » est un item auquel on a appliqué une réduction du prix.

Appliquer les patrons

31

□ Exemple (suite): on applique le patron Décorateur



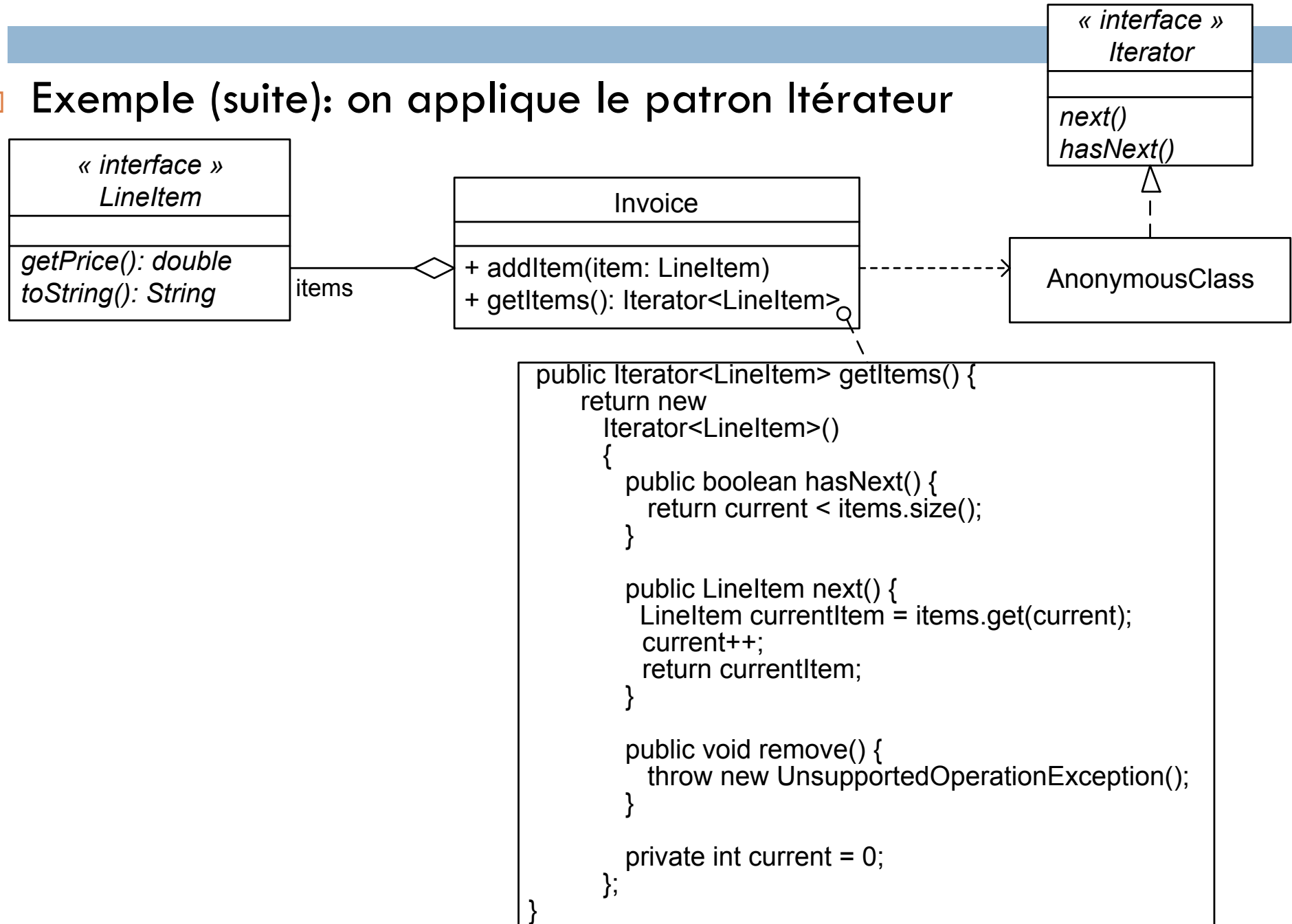
□ Exemple (suite):

- ▣ La classe « Invoice » contient une collection d'items.
- ▣ Les clients de la classe « Invoice » ont besoin de parcourir ses items.
- ▣ Cependant nous ne voulons pas exposer la structure de la classe « Invoice ».
- ▣ Au besoin, nous pourrions vouloir changer l'implémentation de cette collection.

Appliquer les patrons

34

Exemple (suite): on applique le patron Itérateur



□ Exemple (suite):

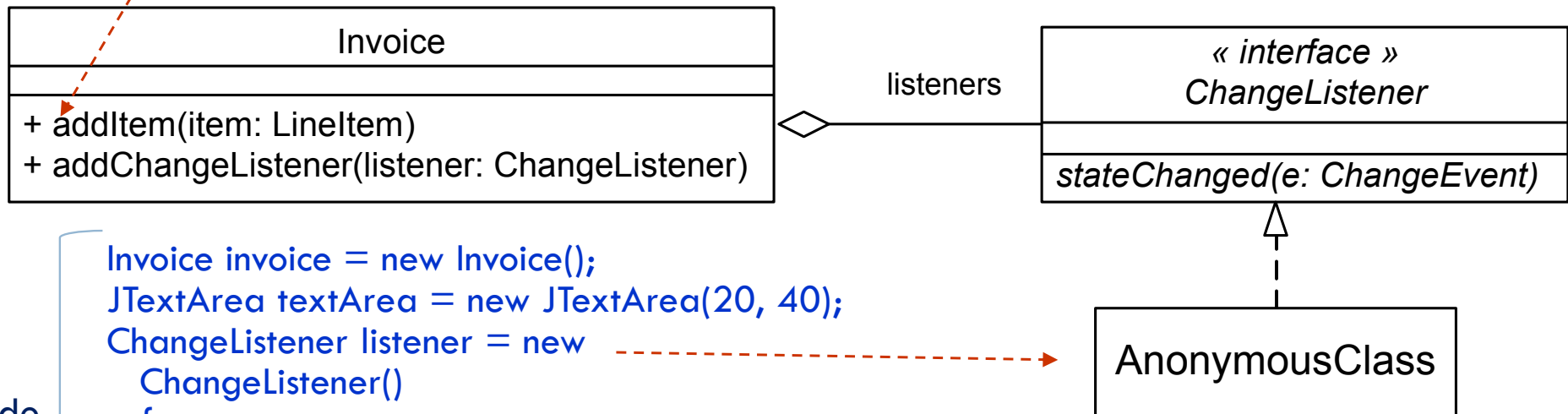
- ▣ L'application a une interface graphique qui permet d'afficher la facture dans une zone de texte.
- ▣ L'interface offre aussi des commandes pour ajouter des articles à la facture. Chaque fois qu'une facture est modifiée, la zone de texte doit être mise à jour.
- ▣ On aimerait une conception qui découple les entrées de l'affichage.

Appliquer les patrons

37

□ Exemple (suite): on applique le patron Observateur

```
public void addItem(LinItem item) {  
    items.add(item);  
    // Notify all observers of the change to the invoice  
    ChangeEvent event = new ChangeEvent(this);  
    for (ChangeListener listener : listeners)  
        listener.stateChanged(event);  
}
```



Extrait
d'une
classe de
test

```
Invoice invoice = new Invoice();  
JTextArea textArea = new JTextArea(20, 40);  
ChangeListener listener = new  
ChangeListener()  
{  
    public void stateChanged(ChangeEvent event)  
    {  
        textArea.setText(...);  
    }  
};  
invoice.addChangeListener(listener);
```

□ Exemple (suite):

- ▣ Notre programme supporte un format simple pour afficher le texte d'une facture dans une zone de texte. Cela peut-être insuffisant pour d'autres applications.
- ▣ Si on veut afficher une facture sur une page Web, le format doit contenir des balises HTML.
- ▣ On veut donc pouvoir supporter plusieurs algorithmes de formatage.

Appliquer les patrons

40

□ Exemple (suite): on applique le patron Stratégie

