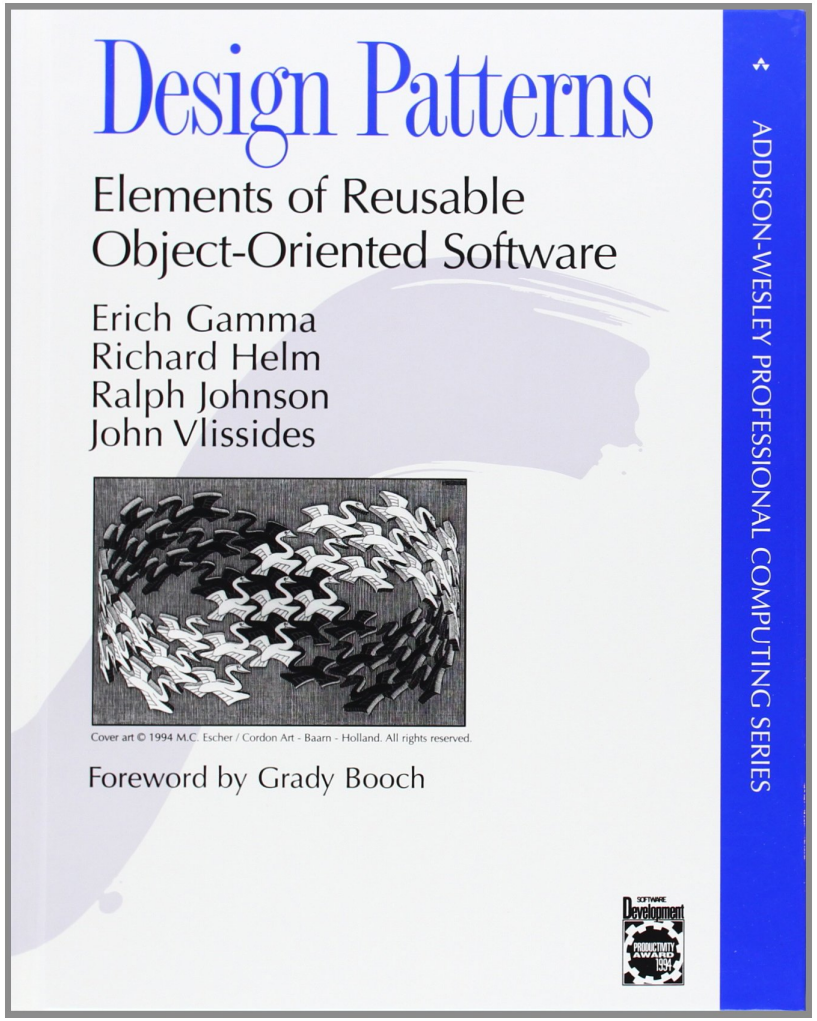
DESIGN PATTERN ADAPTATEUR

MAUREEN CAGNON, ETIENNE COUZON,
RAPHAEL BAULANT, TOM SOUDET

SOMMAIRE

PRÉSENTATION DESIGN PATTERNS

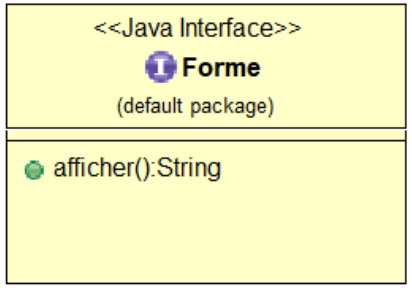
- Solution pour résoudre des problèmes en développement logiciel
- Pour un code propre, optimisé et maintenable

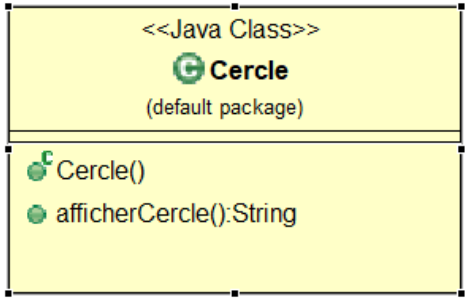


PRÉSENTATION DESIGN PATTERNS

Création	Structuration	Comportement
Abstract Factory	Adapter	Command
Factory	Bridge	Observer
Builder	Composite	State
Object Pool	Decorator	Strategy
Prototype	Facade	Null Object
Singleton	Proxy	Visitor

IMAGINONS UNE SITUATION






```java
public interface Forme {
    String afficher();
}
```

```java
public class Cercle {
    public String afficherCercle() {
        return "Cercle";
    }
}
```

Problème : Classe Cercle inutilisable en raison de son incompatibilité avec l'interface Forme

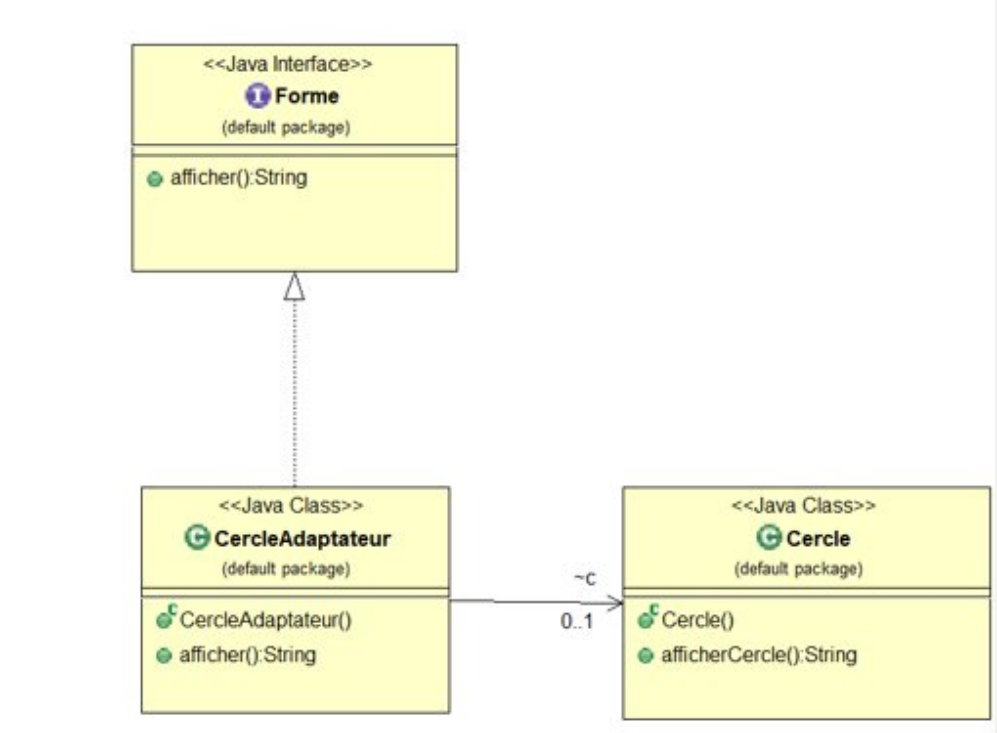
UNE FAUSSE BONNE IDÉE ?

```java
public class Cercle {
    public String afficherCercle() {
        return "Cercle";
    }
}
```

```java
public class Cercle implements Forme {
    public String afficher() {
        return "Cercle";
    }
}
```

OCP : interdiction de modifier du code existant et fonctionnel

UNE SOLUTION POSSIBLE

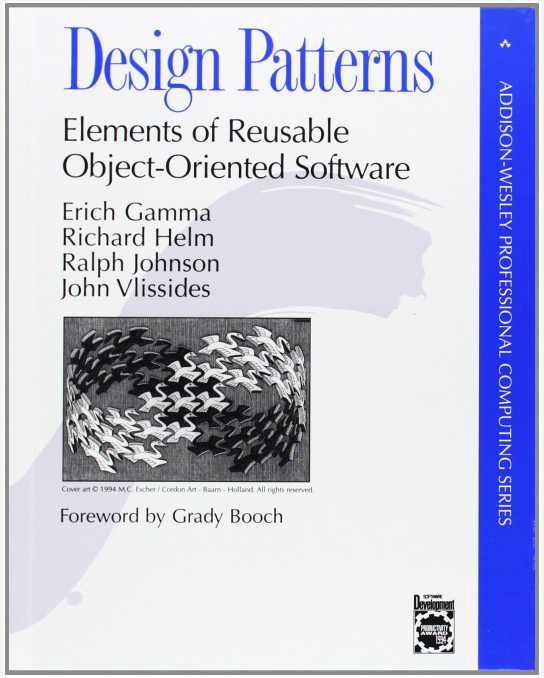


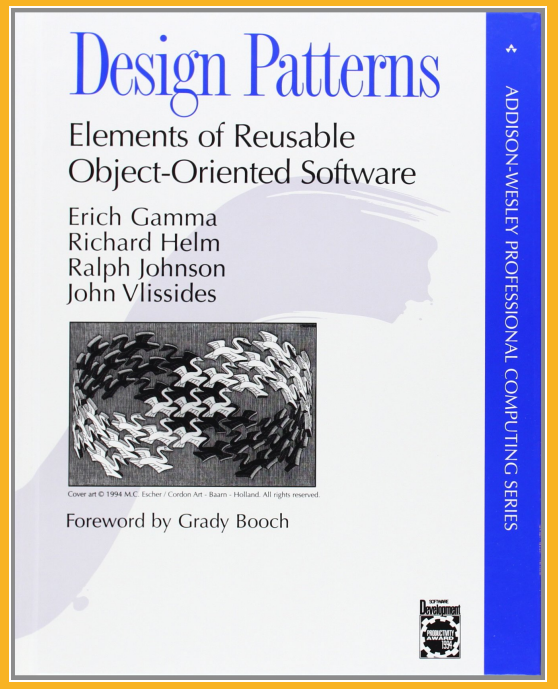

```java
public class CercleAdaptateur implements Forme {
    Cercle c = new Cercle();

    @Override
    public String afficher() {
        return c.afficherCercle();
    }
}
```

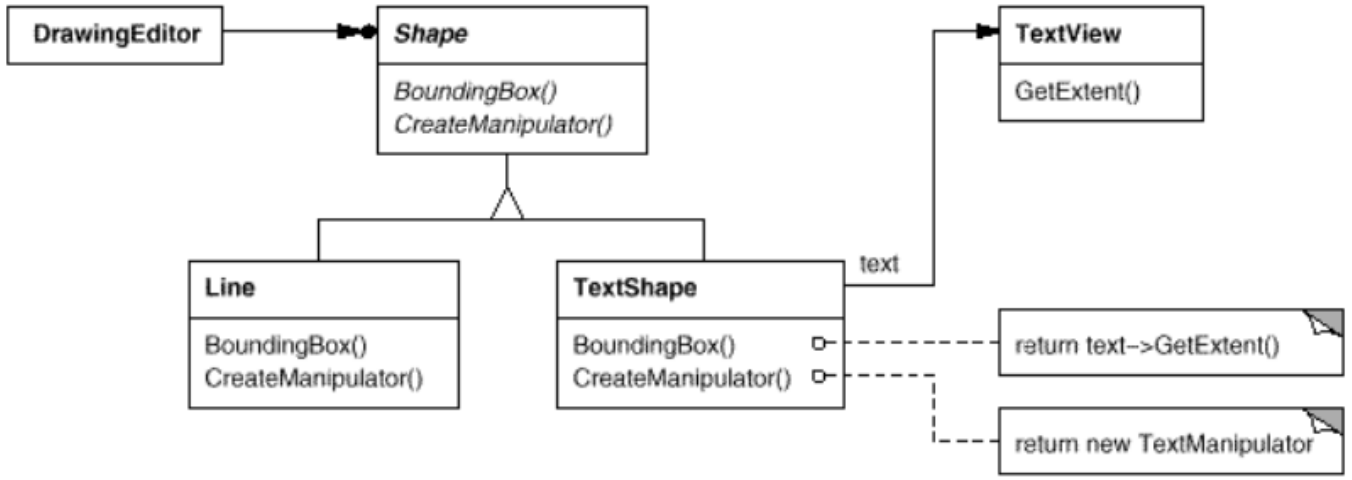
DESIGN PATTERN ADAPTER

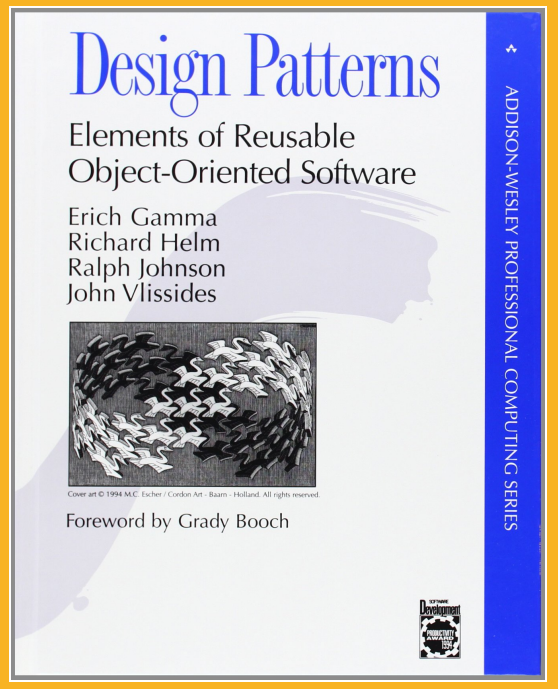
- Design Pattern Adapter ⇨ Structure
- Permet à deux classes incompatibles de communiquer
- Exemple réel : adaptateur de prise électrique

DANS LA LITTÉRATURE

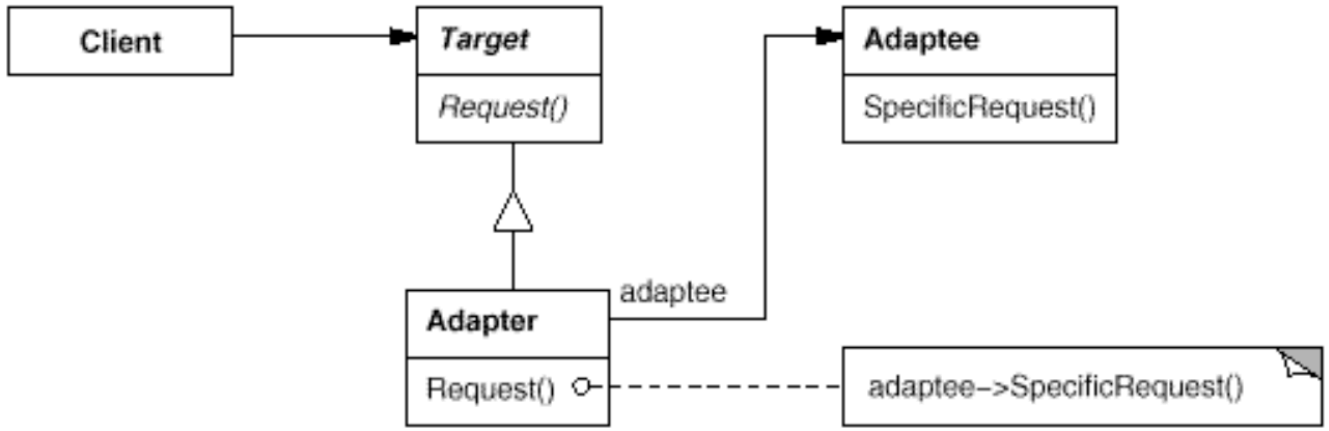


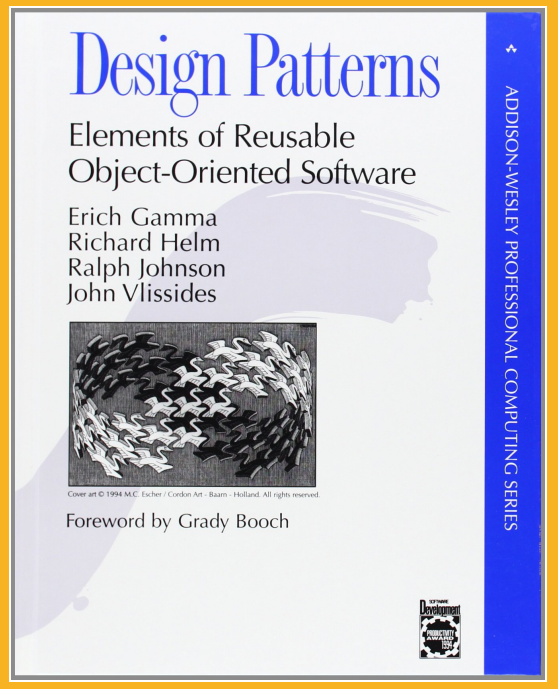

Design Patterns
Elements of Reusable
Object-Oriented Software
Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides
Foreword by Grady Booch
ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES


DrawingEditor
Shape
BoundingBox()
CreateManipulator()
TextView
GetExtent()
Line
BoundingBox()
CreateManipulator()
TextShape
BoundingBox()
CreateManipulator()
text
return text->GetExtent()
return new TextManipulator

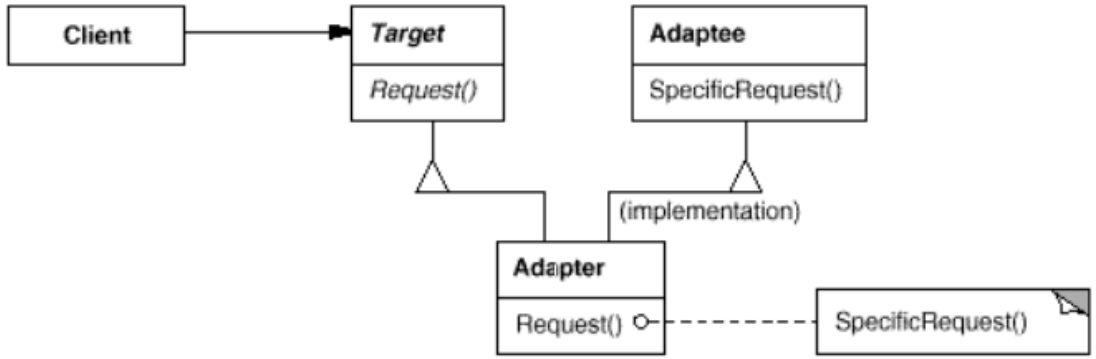

Design Patterns
Elements of Reusable
Object-Oriented Software
Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides
Foreword by Grady Booch
ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES

SOLUTION 1 :


Client
Target
Request()
Adaptee
SpecificRequest()
adaptee
Adapter
Request()
adaptee->SpecificRequest()


Design Patterns
Elements of Reusable
Object-Oriented Software
Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides
Foreword by Grady Booch
ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES

SOLUTION 2 :


Client
Target
Request()
Adaptee
SpecificRequest()
(implementation)
Adapter
Request()
SpecificRequest()

L'ADAPTATEUR ET LES PRINCIPES SOLID

SRP : Responsabilité unique – Un objet doit avoir qu'une seule responsabilité

OCP : Ouvert/Fermé – Les classes sont ouvertes aux extensions mais fermées aux modifications

LSP : Substitution de Liskov - Tout type de base doit pouvoir être remplacé par l'un de ses sous-types

ISP : Ségrégation des Interfaces - Un client ne doit jamais être forcé de dépendre d'une interface qu'il n'utilise pas

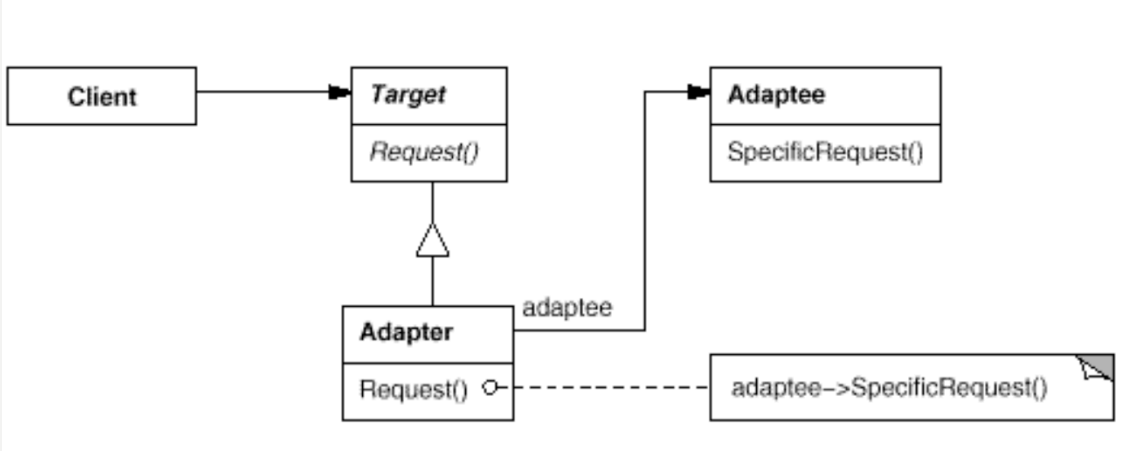
DIP : Inversion des Dépendances - Dépendre des abstractions et non des implémentations.

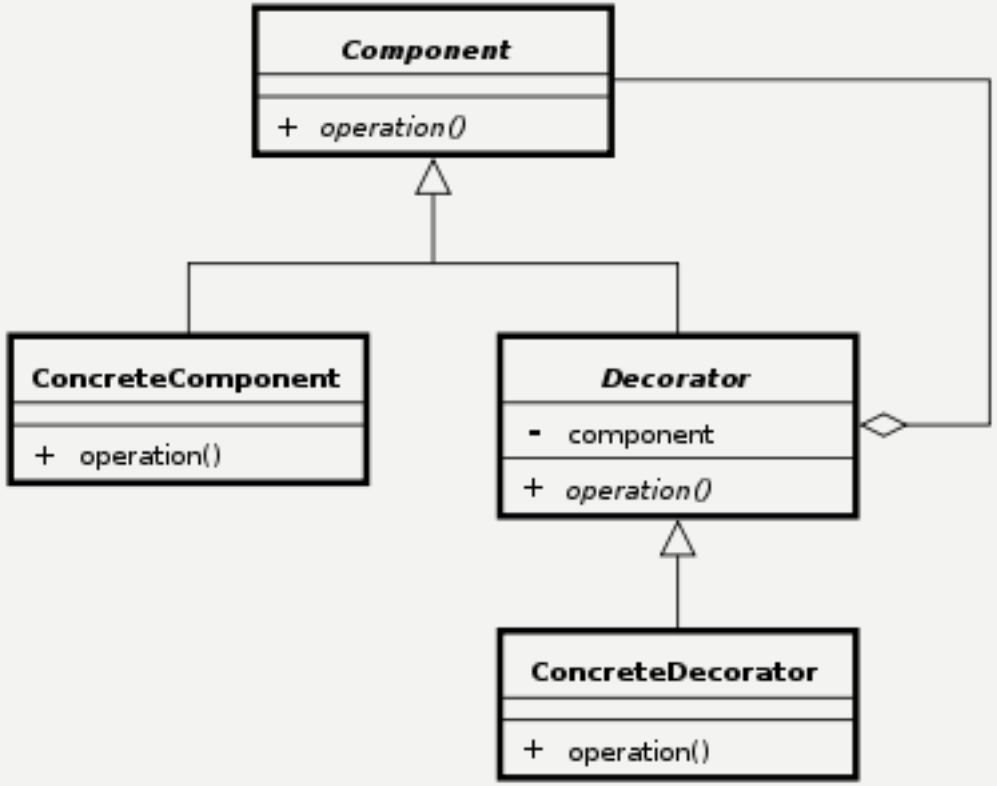
LIMITES DU PATTERN ADAPTER

Plus il y a d'adaptateurs,
plus il y a de classes et d'interfaces

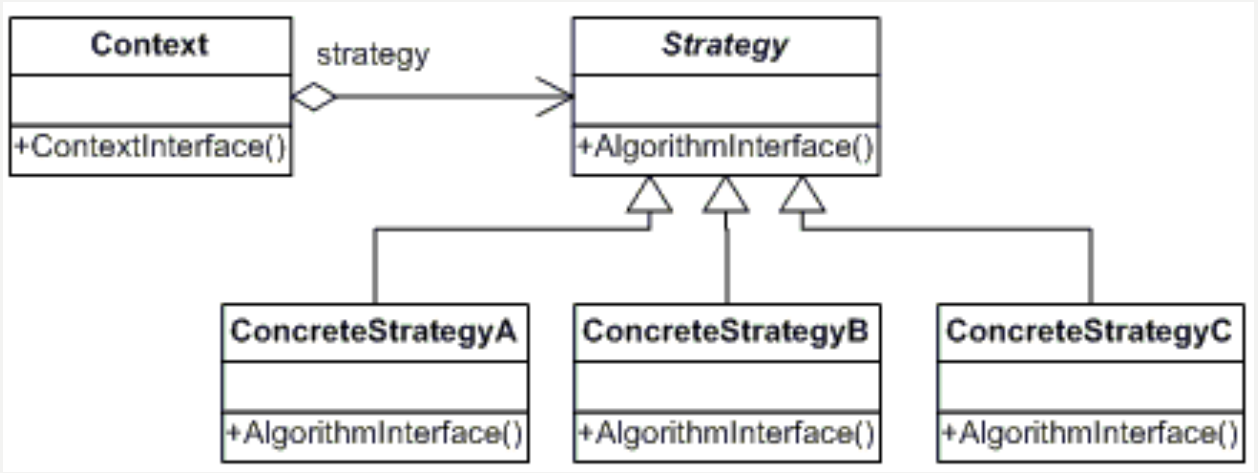
Complexité du code
augmente

MISE EN RELATION AVEC D'AUTRES PATTERNS : DECORATOR





MISE EN RELATION AVEC D'AUTRES PATTERNS : STATE/STRATEGY



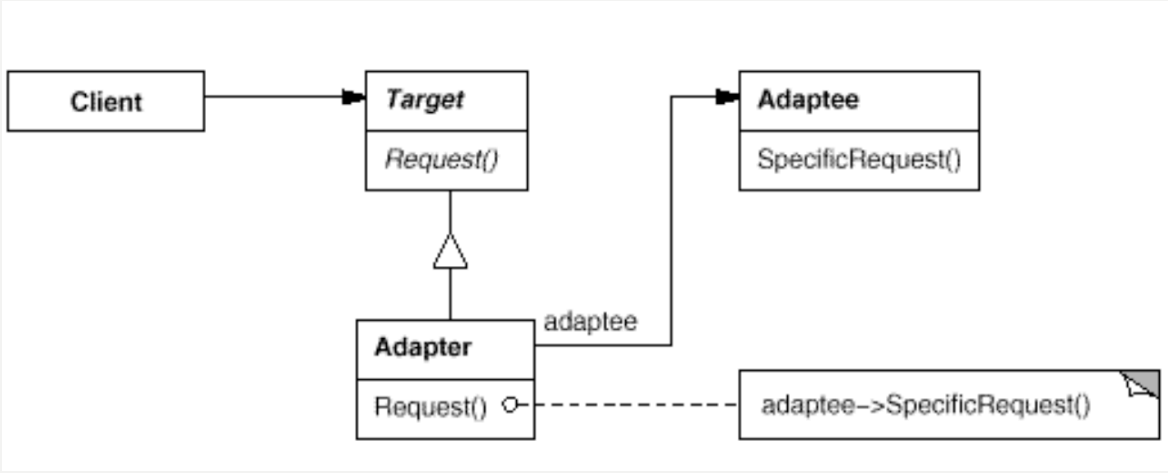
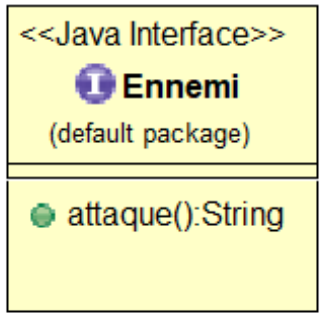
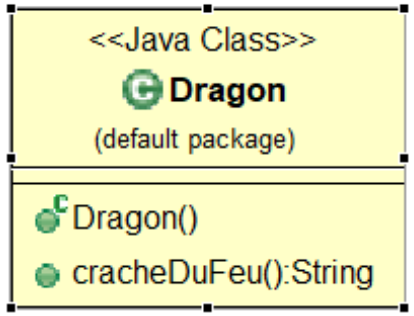


ILLUSTRATION DANS UN JEU

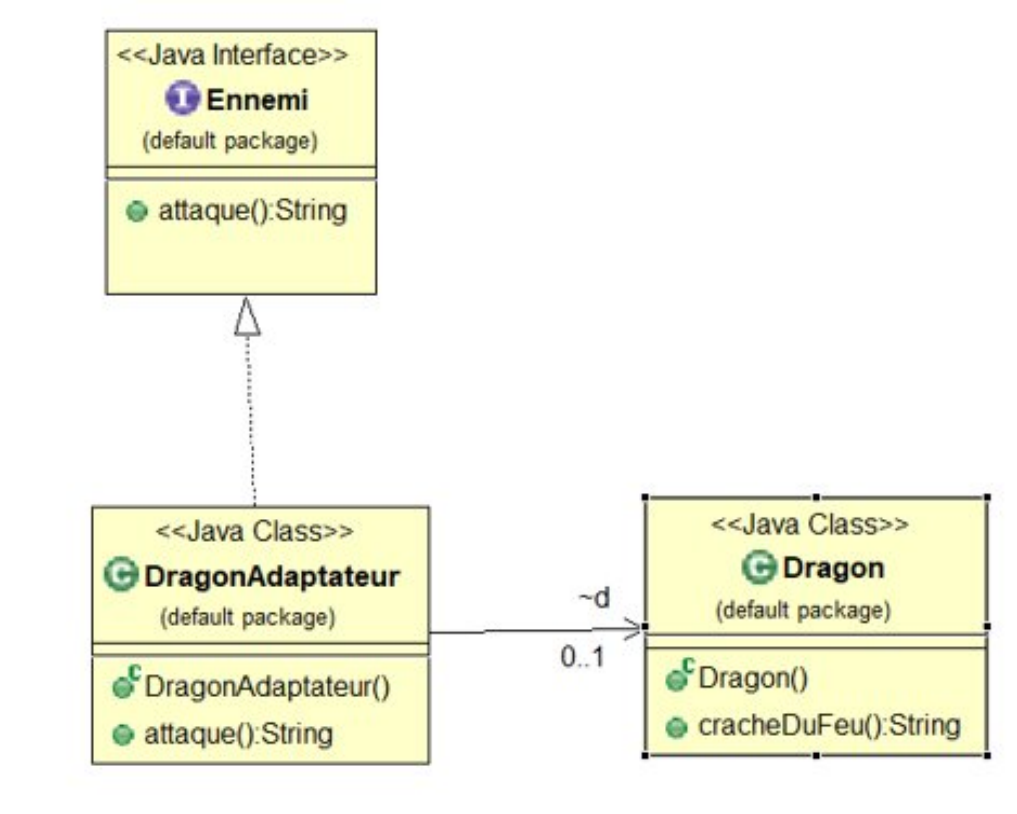





```java
public interface Ennemi {
    String attaque();
}
```

```java
public class Dragon {
    public String cracheDuFeu() {
        return "Crache du feu";
    }
}
```

ILLUSTRATION DANS UN JEU




```java
public class DragonAdaptateur implements Ennemi {
    Dragon d = new Dragon();

    @Override
    public String attaque() {
        return d.cracheDuFeu();
    }
}
```

QCM

Question 1 :

À quel type de design pattern, le pattern Adapter appartient-il ?

- Pattern de création
- ✓ Pattern de structure
- Pattern de comportement

Question 2 :

Pour mettre un adaptateur dans un code, il faut ajouter combien de classe ?

- ✔ 1
- 2
- 3

Question 3 :

Quels principes SOLID le pattern Adapter respecte-t-il ?

- [x] Principe SRP
- [x] Principe OCP
- [ ] Principe LSP
- [ ] Principe ISP
- [ ] Principe DIP

Question 4 :

Quelle est la limite du design pattern Adapter ?

- Plus d'adaptateurs entraîne moins de classes et d'interfaces et diminue la complexité du code
- Moins d'adaptateurs entraîne plus de classes et d'interfaces et diminue la complexité du code
- ✓ Plus d'adaptateurs entraîne plus de classes et d'interfaces et augmente la complexité du code
- Moins d'adaptateurs entraîne plus de classes et d'interfaces et augmente la complexité du code

Question 5 :

À quels patterns est lié le pattern Adapter ?

- [x] **Decorator**
- [ ] **Observer**
- [x] **State/Strategy**
- [ ] **Builder**
- [ ] **Facade**

Question 6 :

Le design pattern Adapter permet de faire collaborer des classes dont l'interface est incompatible

- **Vrai**
- **Faux**

RÉFÉRENCES UTILISÉES

- https://refactoring.guru/fr/design-patterns/adapter
- https://blog.cellenza.com/developpement-specifique/le-design-pattern-adapter/
- https://anceret-matthieu.fr/2018/08/les-design-patterns-et-les-principes-solid-en-développement-logiciel-1/4/
- https://algocool.fr/adapter/

MERCI DE VOTRE ATTENTION