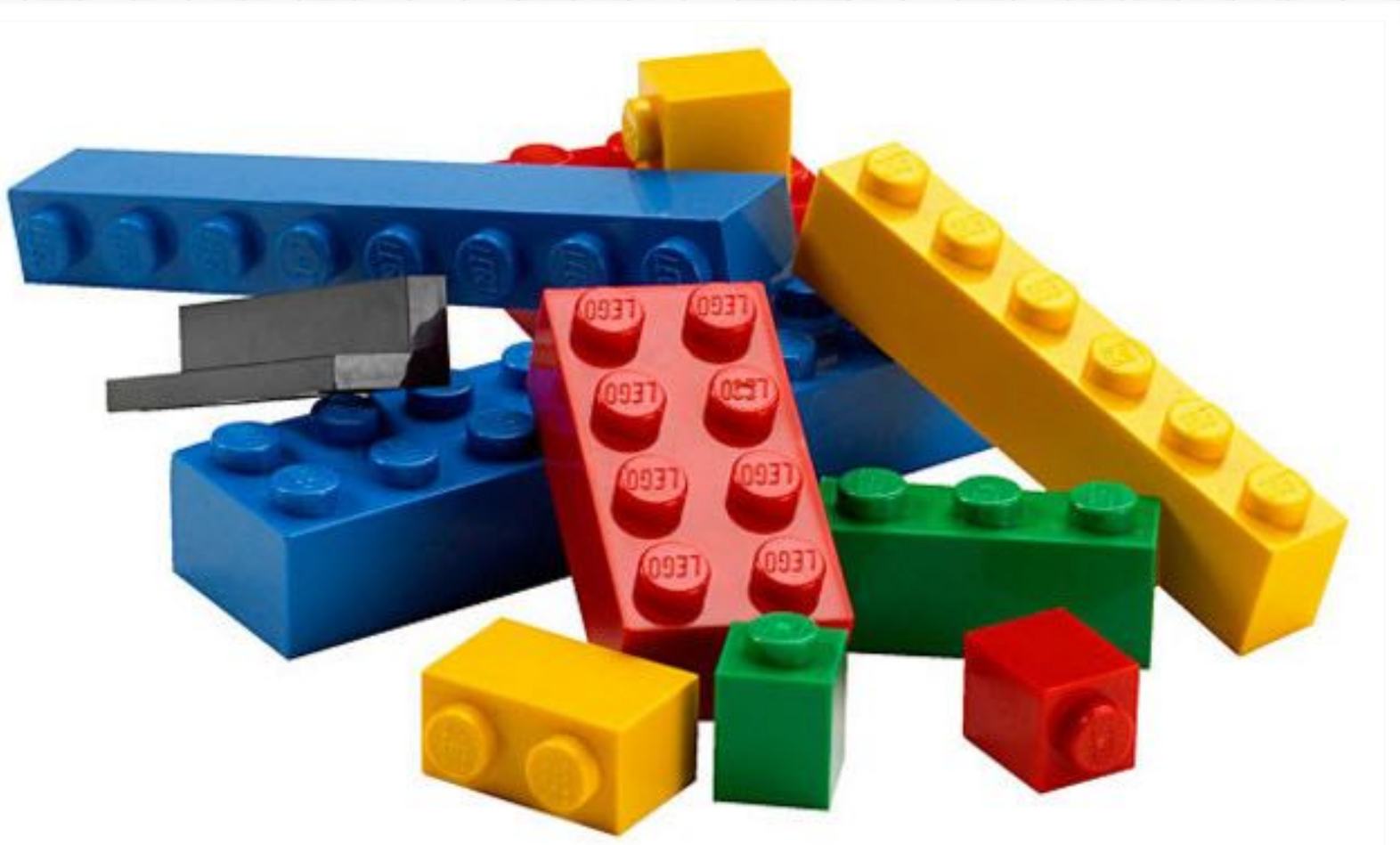
CONCEPTION AVANCÉE

COURS 2: DESIGN PATTERNS

lazaar@lirmm.fr



Pattern ou Patron

Pattern ou Patron

Un pattern décrit à la fois un **problème** qui se produit très fréquemment dans l' environnement et l'architecture de la **solution** à ce problème de telle façon que l'on puisse **utiliser** cette solution des milliers de fois sans jamais **l'adapter** deux fois de la même manière. C. Alexander

DESIGN PATTERNS

Modèles ou Patrons de Conception

DESIGN PATTERNS

Modèles ou Patrons de Conception

- **Coad [Coad92] –** Une abstraction d'un doublet, triplet ou d'un ensemble de classes qui peut être réutilisé encore et encore pour le développement d'applications
- **Appleton[Appleton97] –** Une règle tripartite exprimant une relation entre un certain contexte, un certain problème qui apparaît répétitivement dans ce contexte et une certaine configuration logicielle qui permet la résolution de ce problème

DESIGN PATTERNS

Modèles ou Patrons de Conception

- **Coad [Coad92] –** Une abstraction d'un doublet, triplet ou d'un ensemble de classes qui peut être réutilisé encore et encore pour le développement d'applications
- **Appleton[Appleton97] –** Une règle tripartite exprimant une relation entre un certain contexte, un certain problème qui apparaît répétitivement dans ce contexte et une certaine configuration logicielle qui permet la résolution de ce problème
- **Aarsten [Aarsten96] –** Un groupe d'objets coopérants liés par des relations et des règles qui expriment les liens entre un contexte, un problème de conception et sa solution

DESIGN PATTERNS

Object Oriented Design

DESIGN PATTERNS

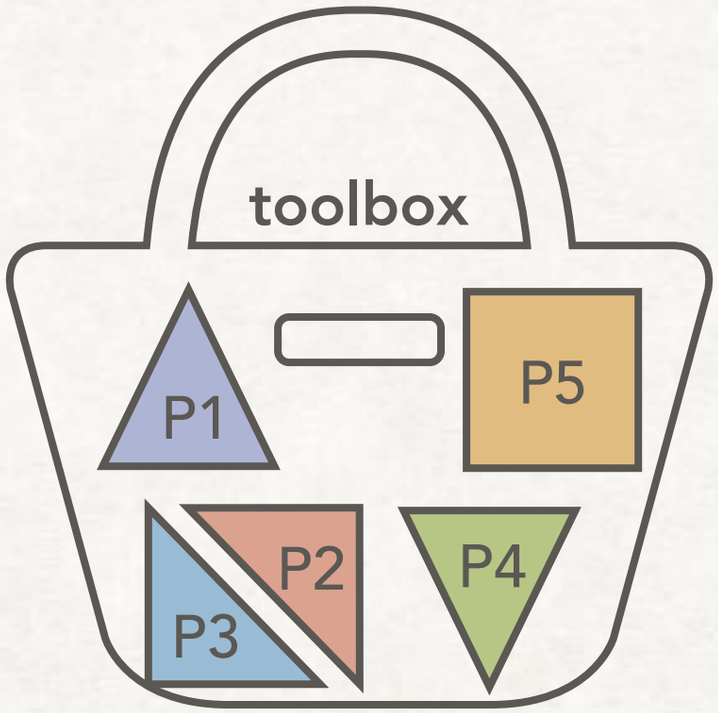
Object Oriented Design

DESIGN PATTERNS

Object Oriented Design

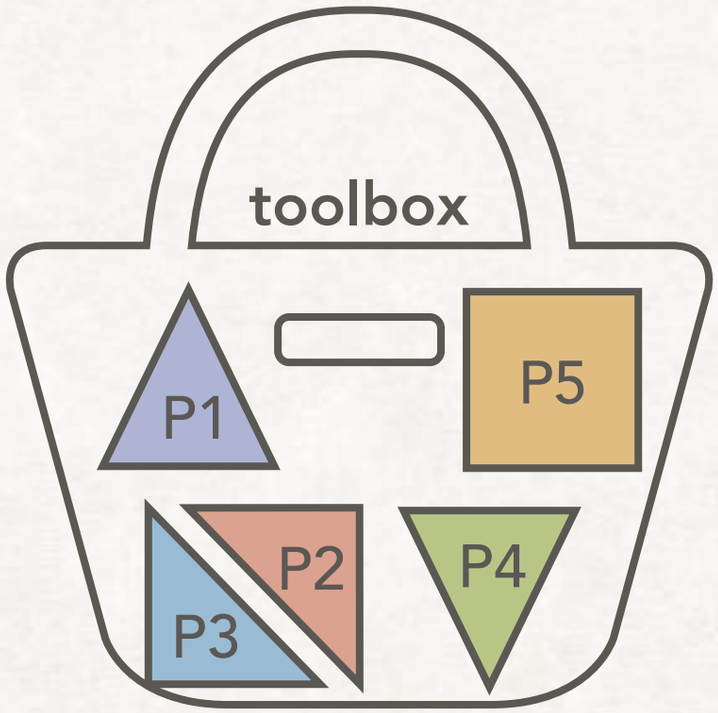
DESIGN PATTERNS

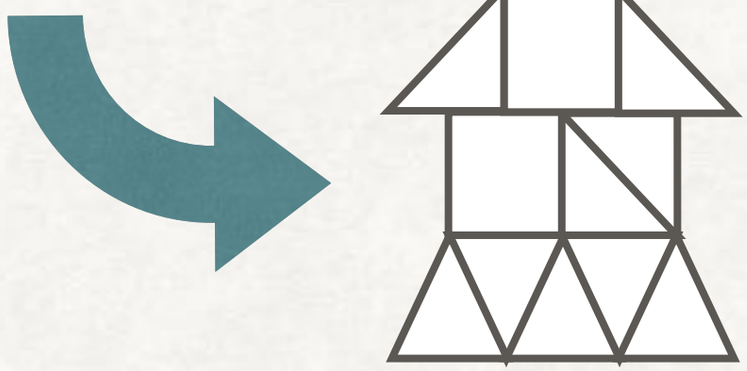
Object Oriented Design



DESIGN PATTERNS

Object Oriented Design

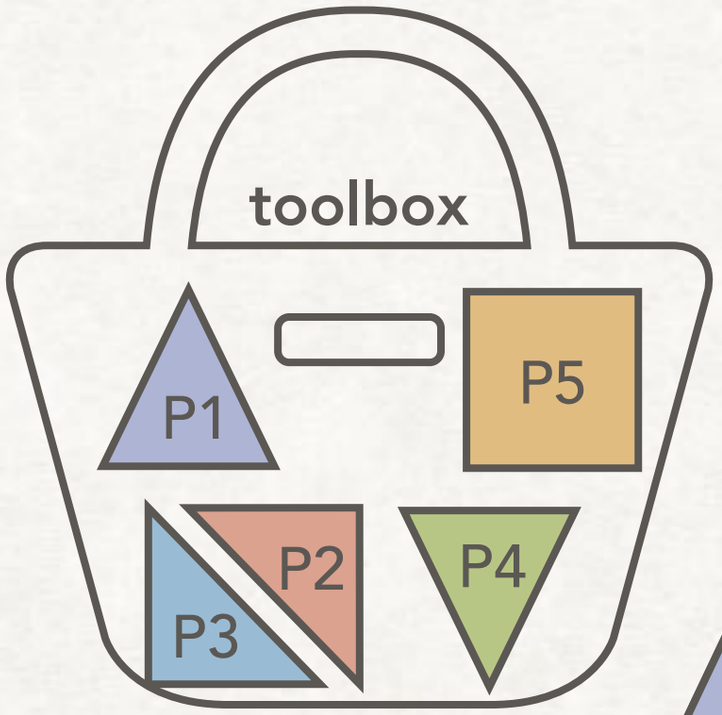


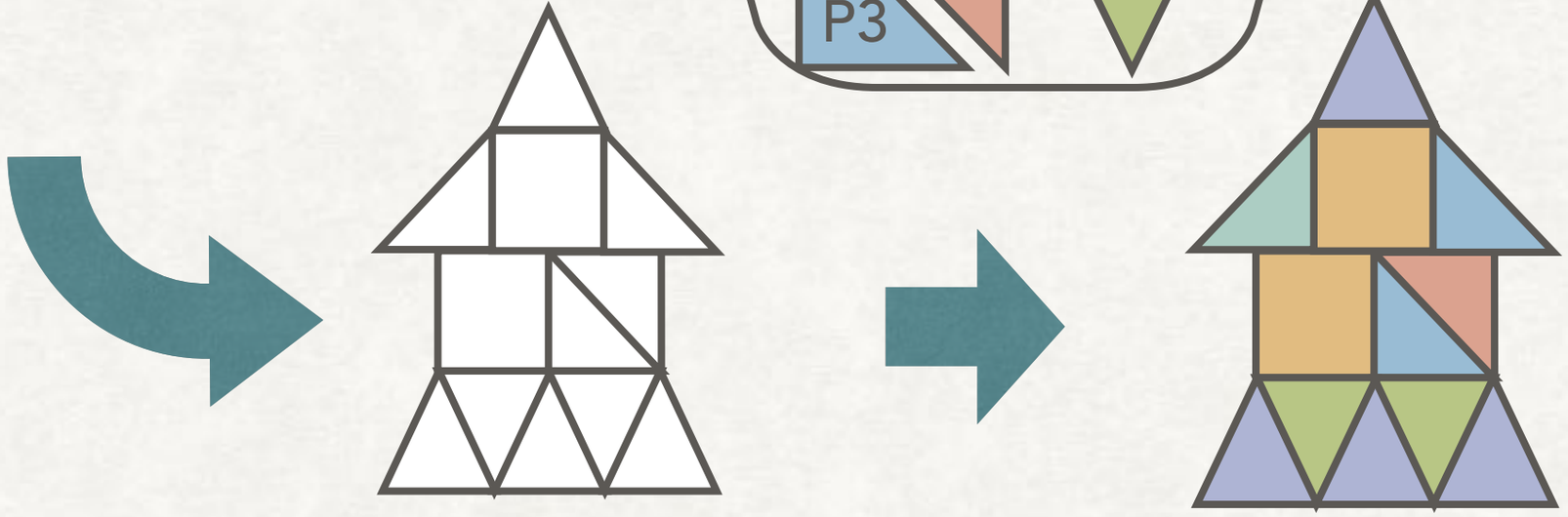


Règles de découpage

DESIGN PATTERNS

Object Oriented Design





Règles de découpage

Règles de composition

PATTERNS CATEGORIES

PATTERNS CATEGORIES

- **Patterns de création :** Ces patterns sont très courants pour déléguer à d'autres classes la construction des objets.

PATTERNS CATEGORIES

- **Patterns de création :** Ces patterns sont très courants pour déléguer à d'autres classes la construction des objets.
- **Patterns de structure :** Ces patterns tendent à concevoir des agglomérations de classes avec des macro-composants.

PATTERNS CATEGORIES

- **Patterns de création :** Ces patterns sont très courants pour déléguer à d'autres classes la construction des objets.
- **Patterns de structure :** Ces patterns tendent à concevoir des agglomérations de classes avec des macro-composants.
- **Patterns de comportement :** Ces patterns tentent de répartir les responsabilités entre chaque classe (l'usage est plutôt dynamique).

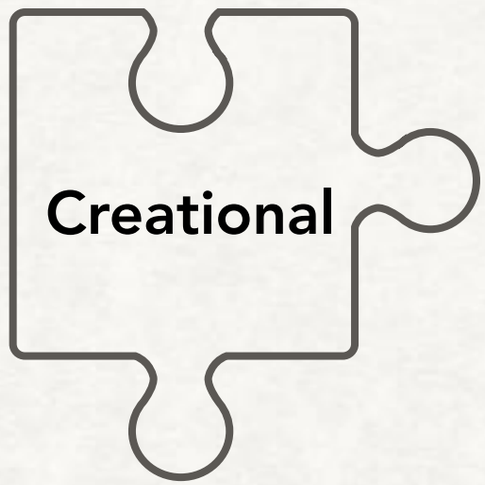
PATTERNS VS UML

Patterns

UML

PATTERNS VS UML

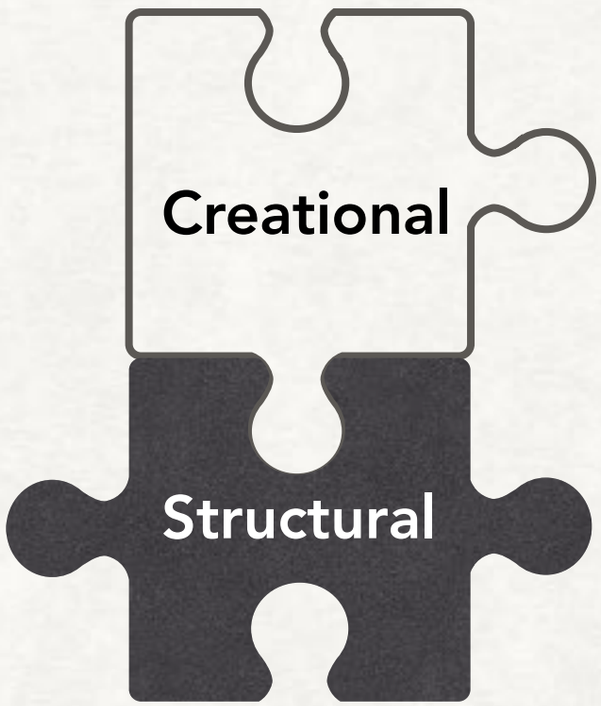
Patterns



UML

PATTERNS VS UML

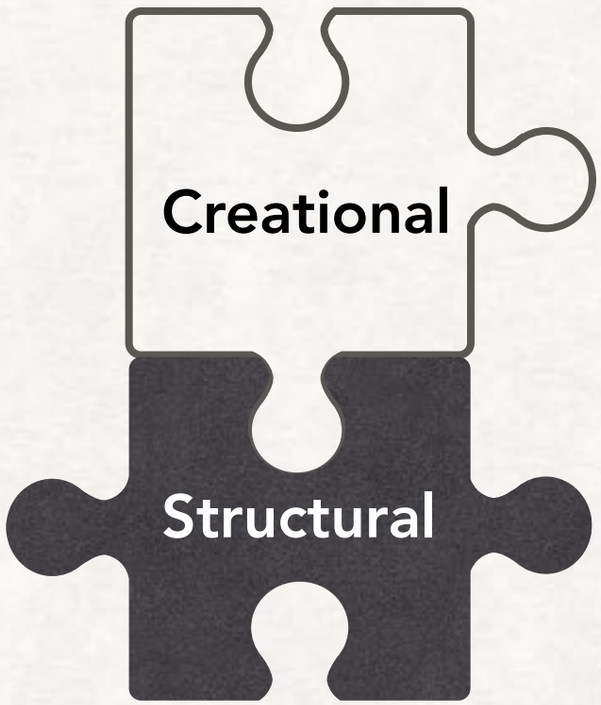
Patterns



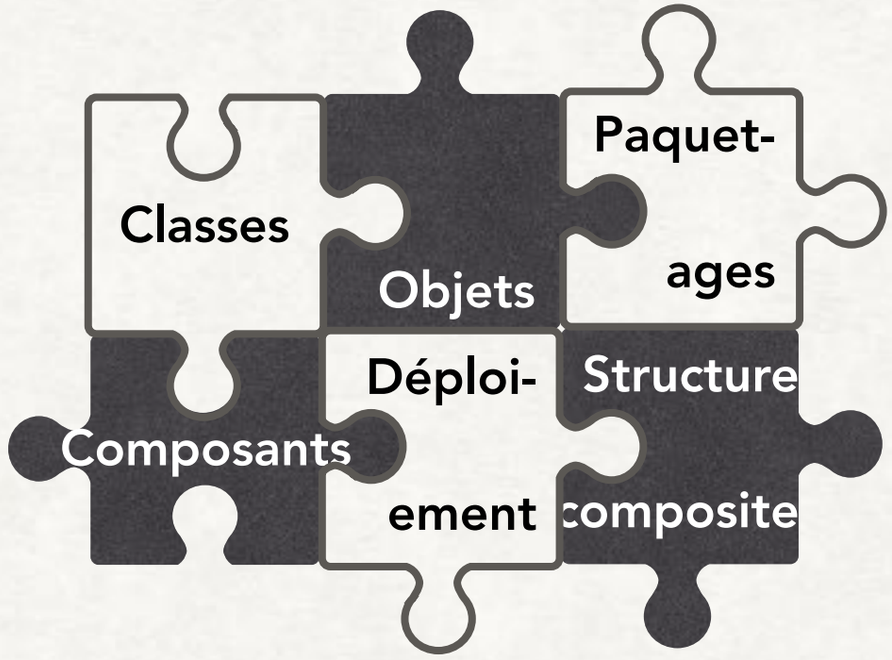
UML

PATTERNS VS UML

Patterns

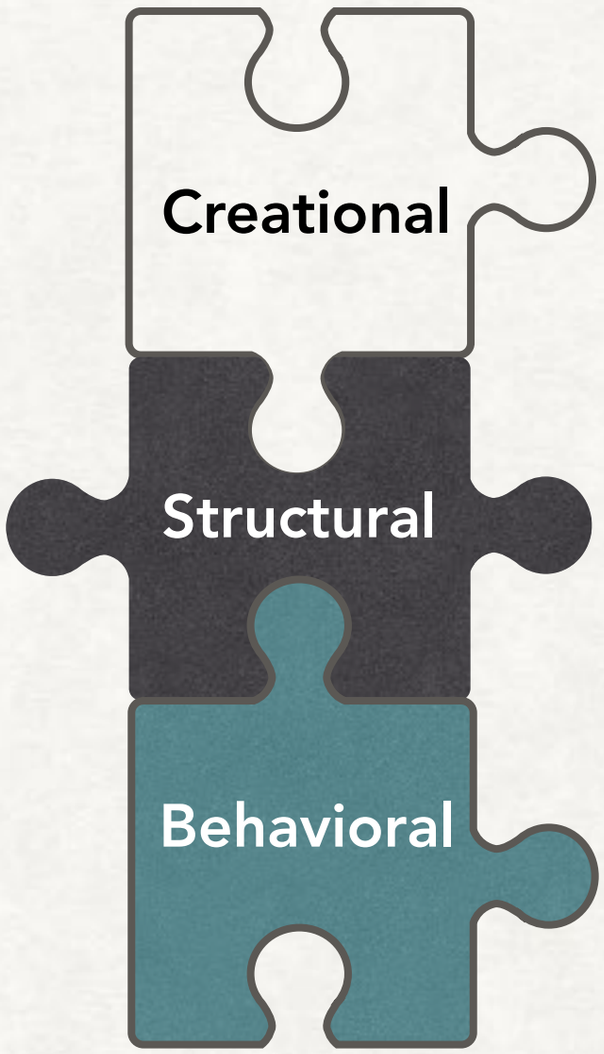


UML

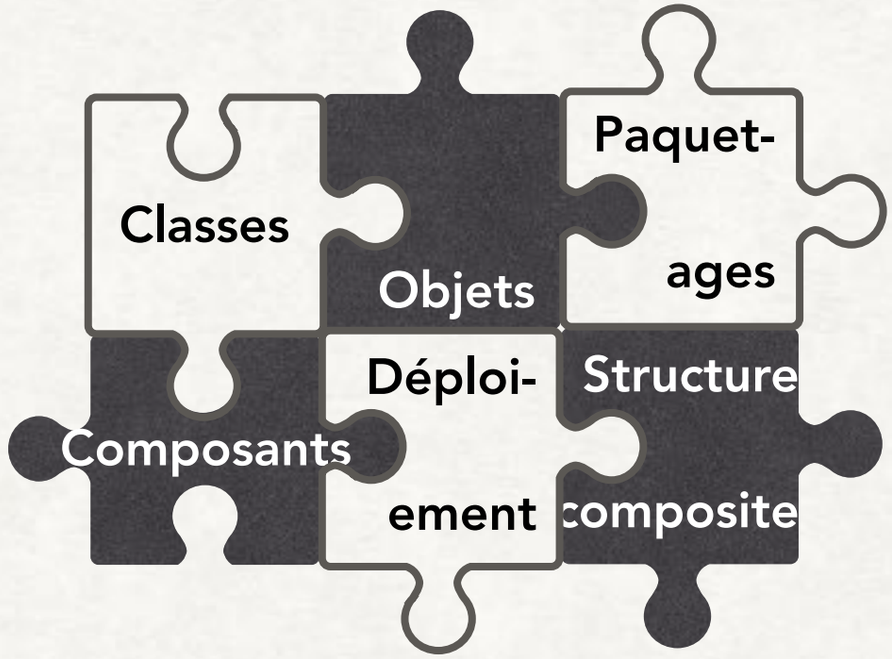


PATTERNS VS UML

Patterns

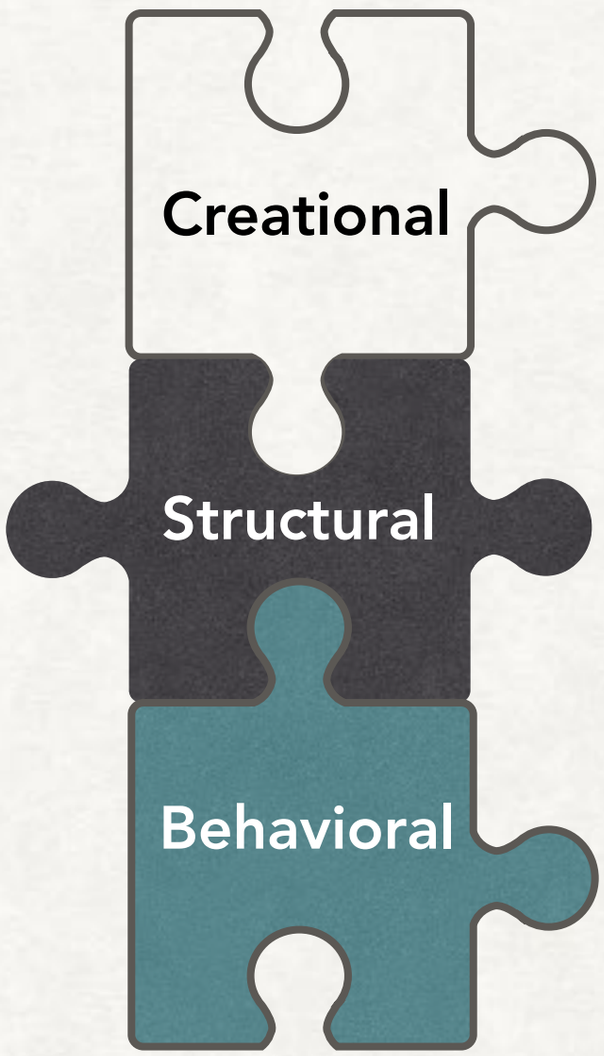


UML

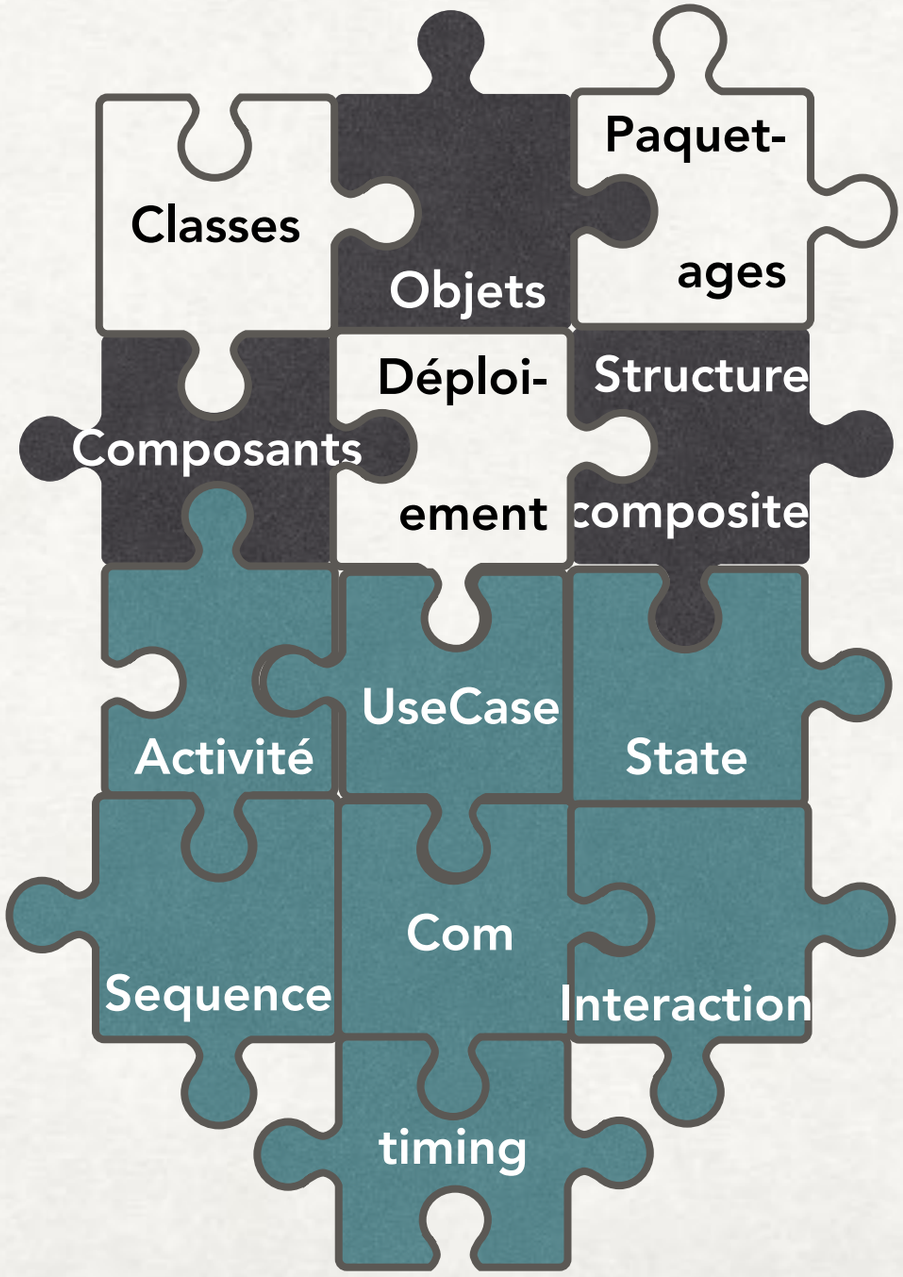


PATTERNS VS UML

Patterns



UML



CREATIONAL PATTERNS

FACTORY

CREATIONAL PATTERNS

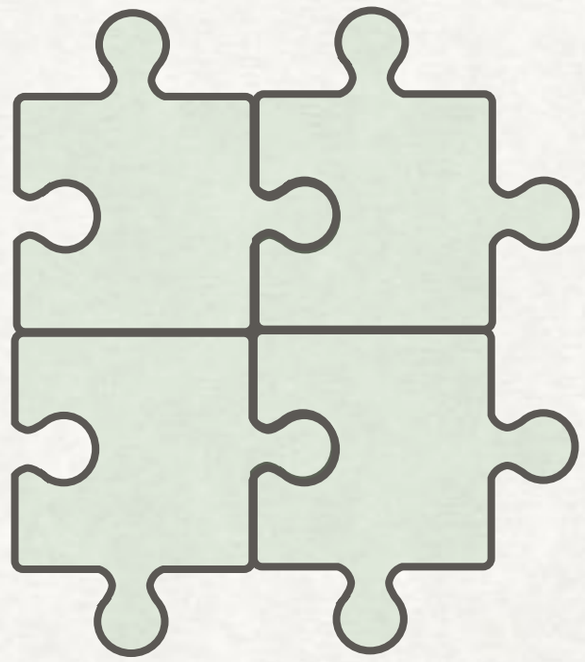
FACTORY

- Problème d'évolution des classes
- Chercher la ligne de code effectuant une instanciation d'un objet (dans une même famille par héritage)

CREATIONAL PATTERNS

FACTORY

- Problème d'évolution des classes
- Chercher la ligne de code effectuant une instanciation d'un objet (dans une même famille par héritage)

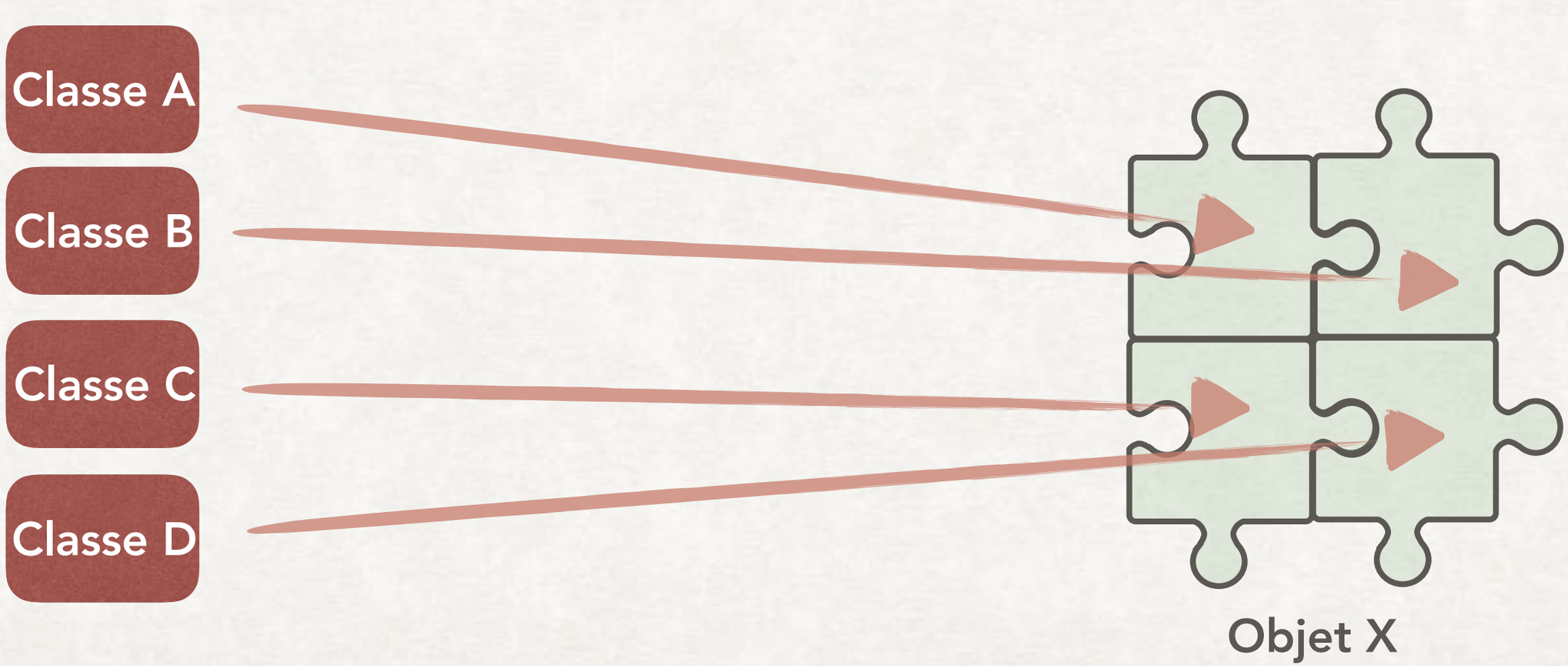


Objet X

CREATIONAL PATTERNS

FACTORY

- Problème d'évolution des classes
- Chercher la ligne de code effectuant une instanciation d'un objet (dans une même famille par héritage)



CREATIONAL PATTERNS

FACTORY

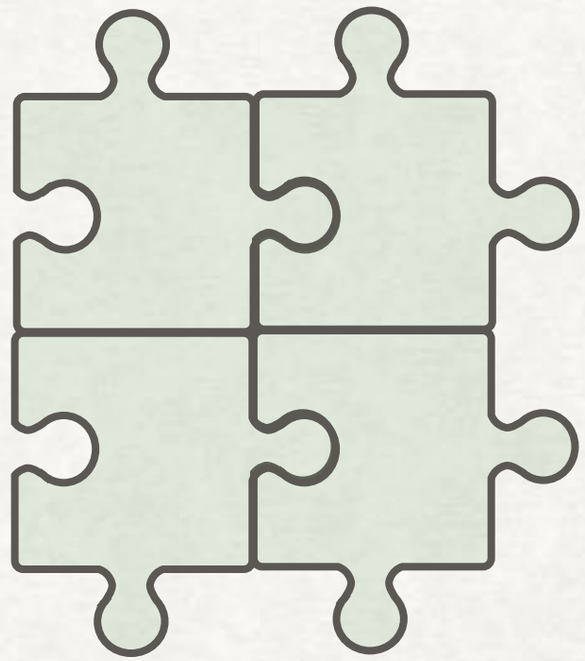
- Problème d'évolution des classes
- Chercher la ligne de code effectuant une instanciation d'un objet (dans une même famille par héritage)

Classe A

Classe B

Classe C

Classe D

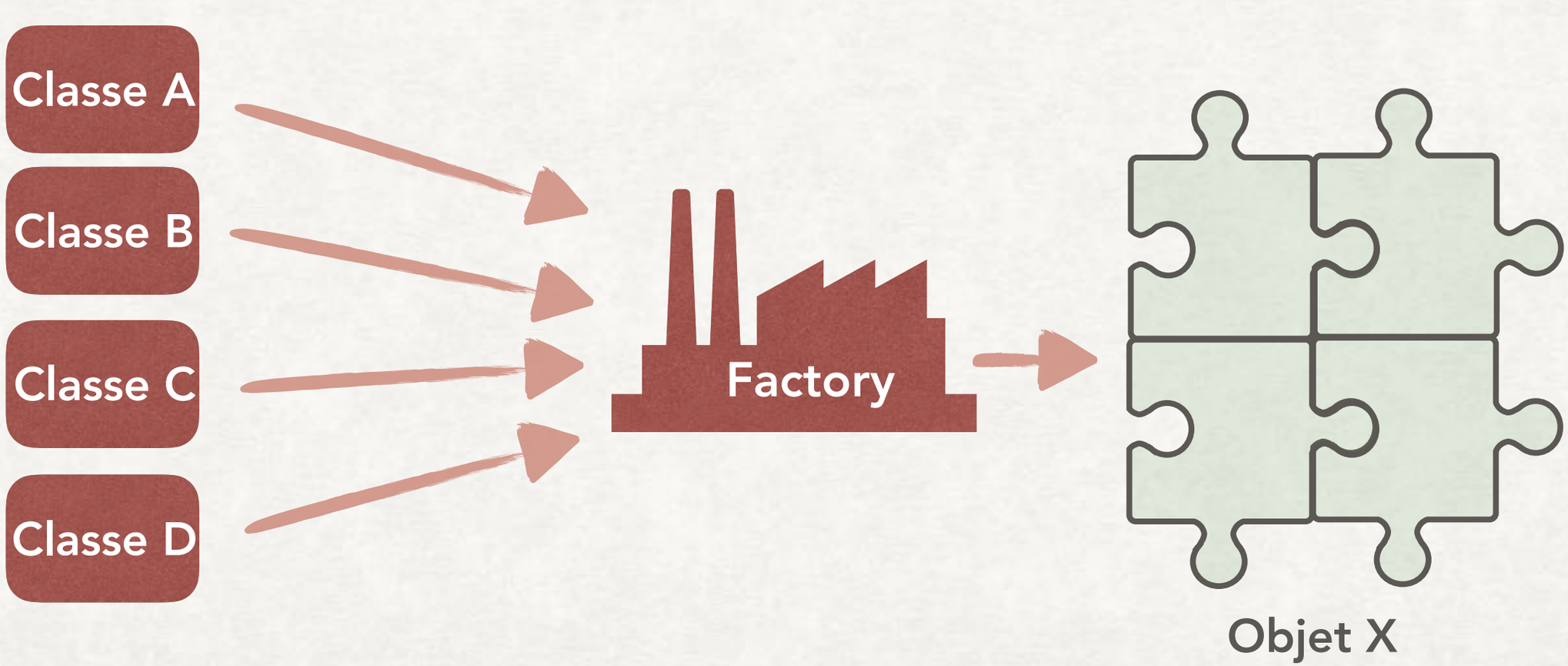


Objet X

CREATIONAL PATTERNS

FACTORY

- Problème d'évolution des classes
- Chercher la ligne de code effectuant une instanciation d'un objet (dans une même famille par héritage)



CREATIONAL PATTERNS

FACTORY

CREATIONAL PATTERNS

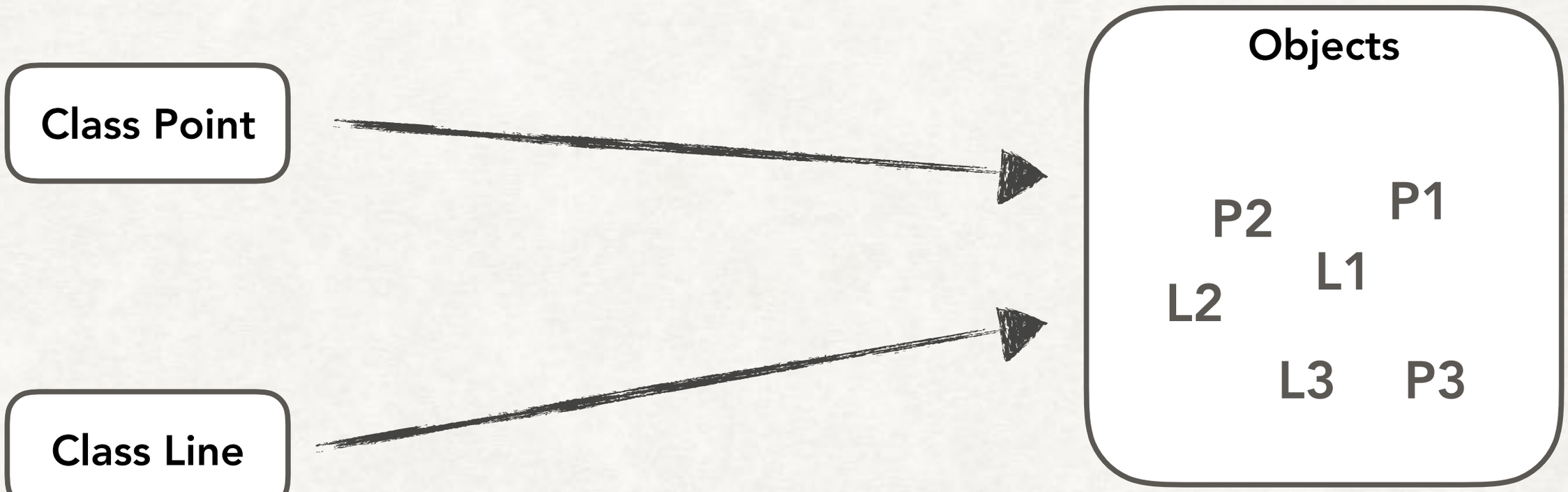
FACTORY

Class Point

Class Line

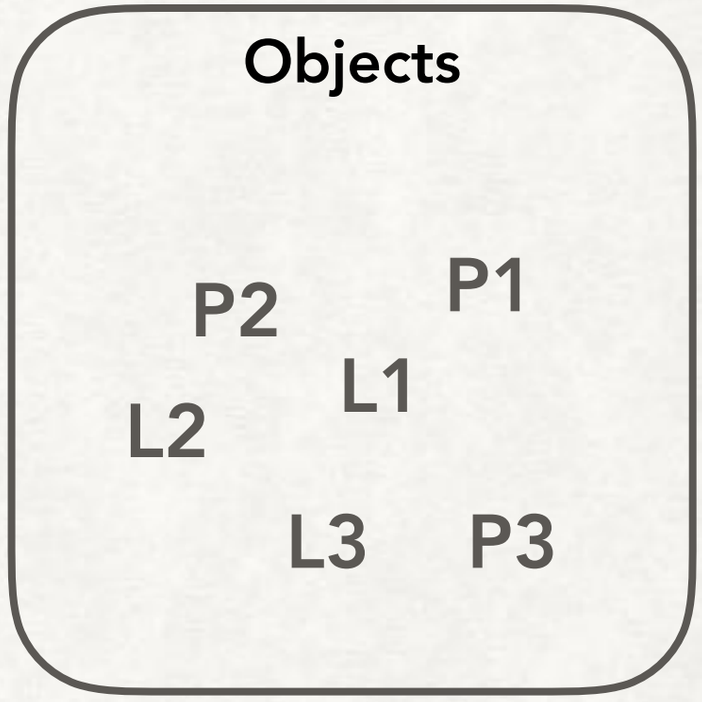
CREATIONAL PATTERNS

FACTORY



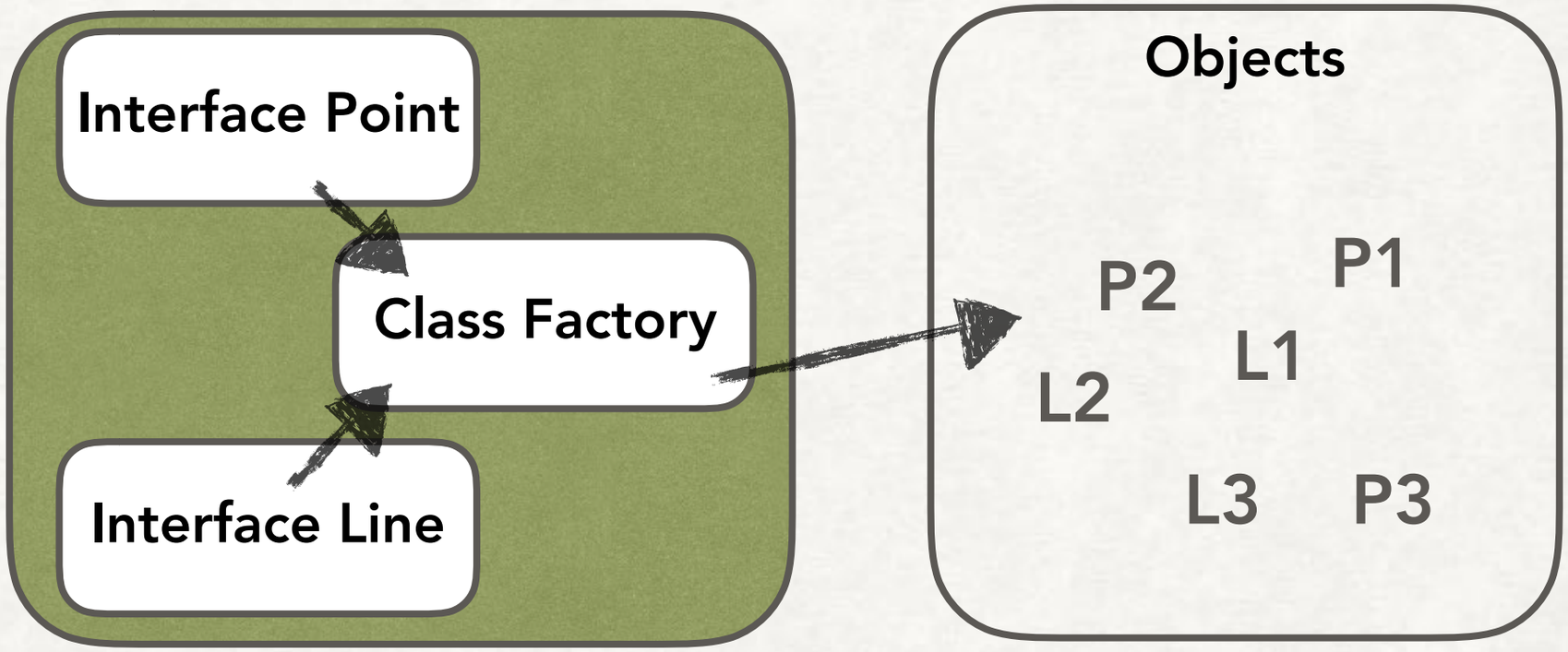
CREATIONAL PATTERNS

FACTORY



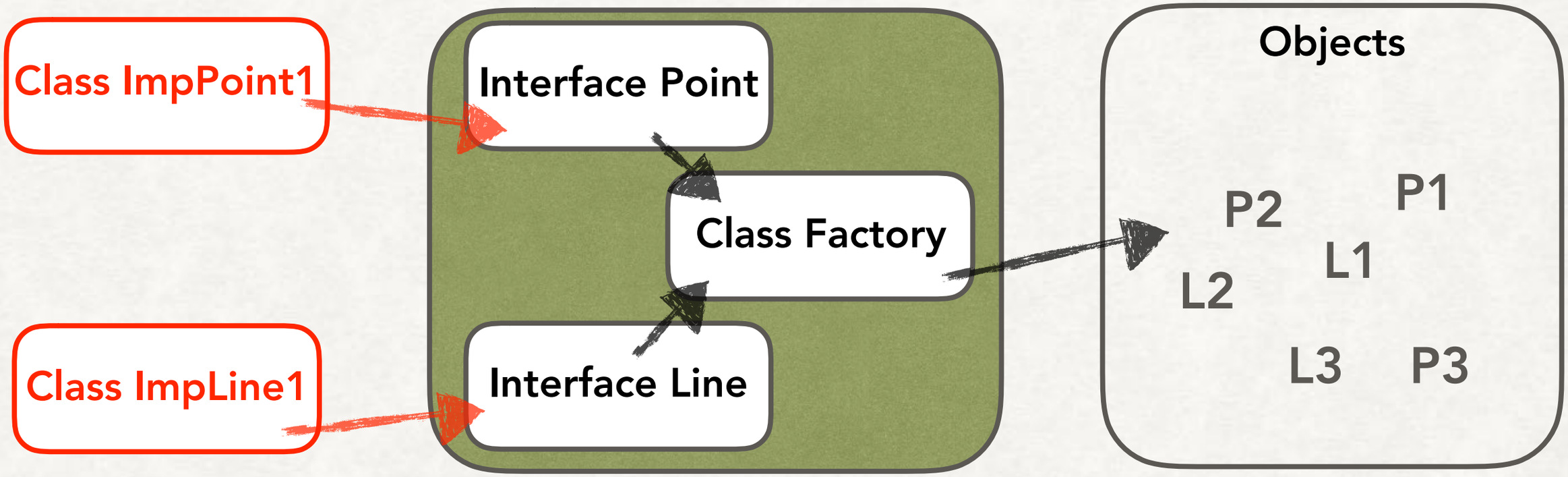
CREATIONAL PATTERNS

FACTORY



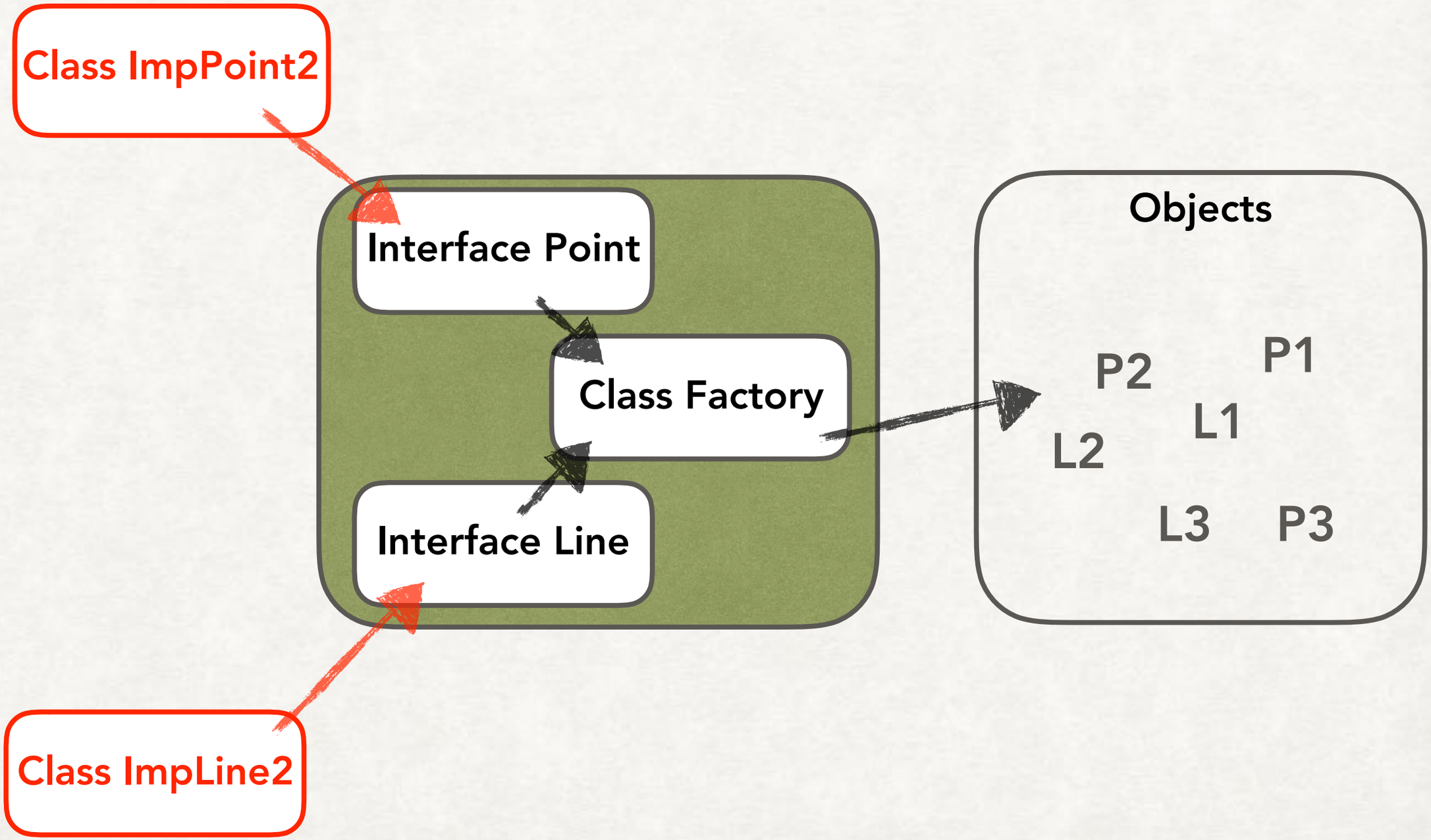
CREATIONAL PATTERNS

FACTORY



CREATIONAL PATTERNS

FACTORY

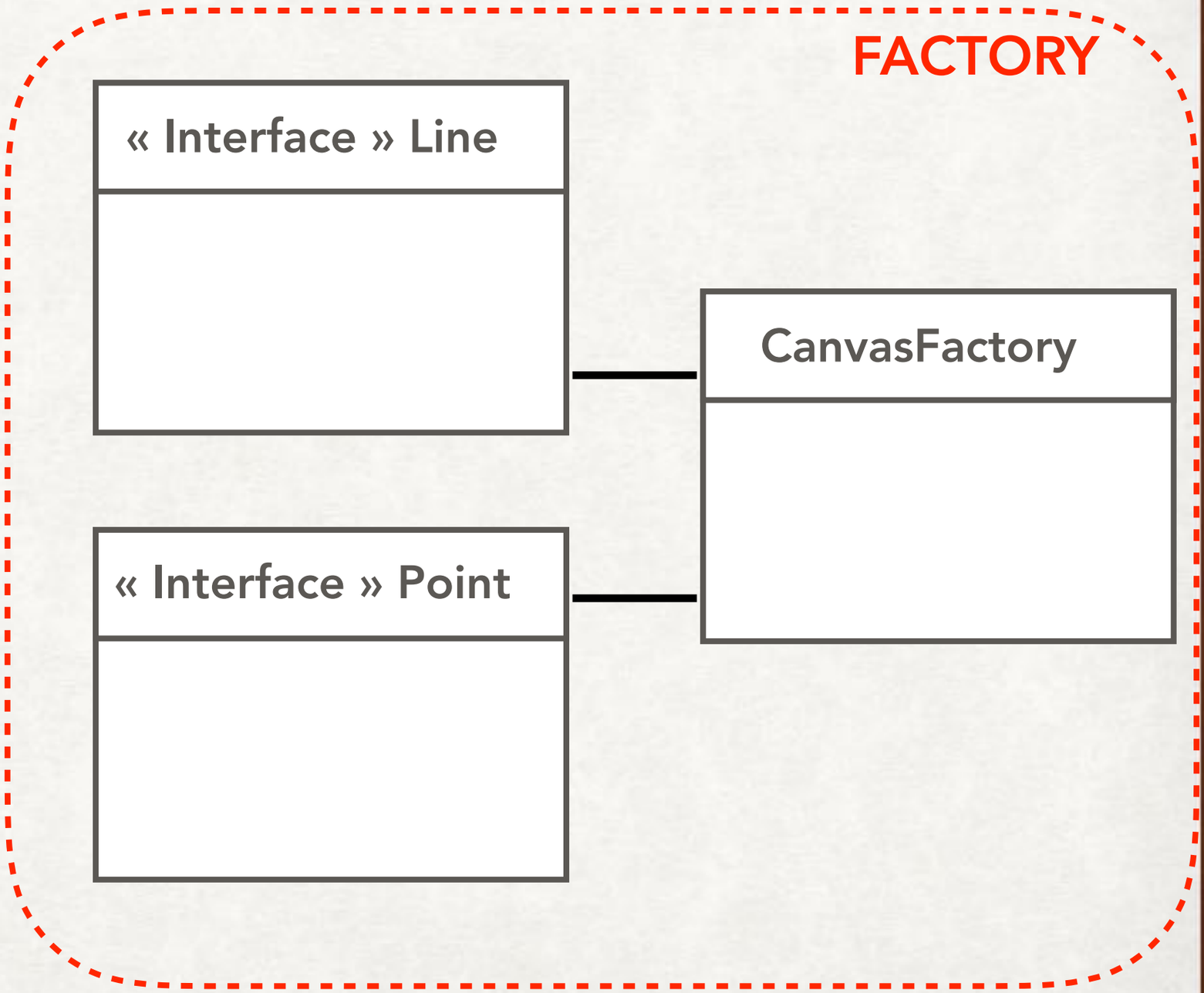


CREATIONAL PATTERNS

FACTORY

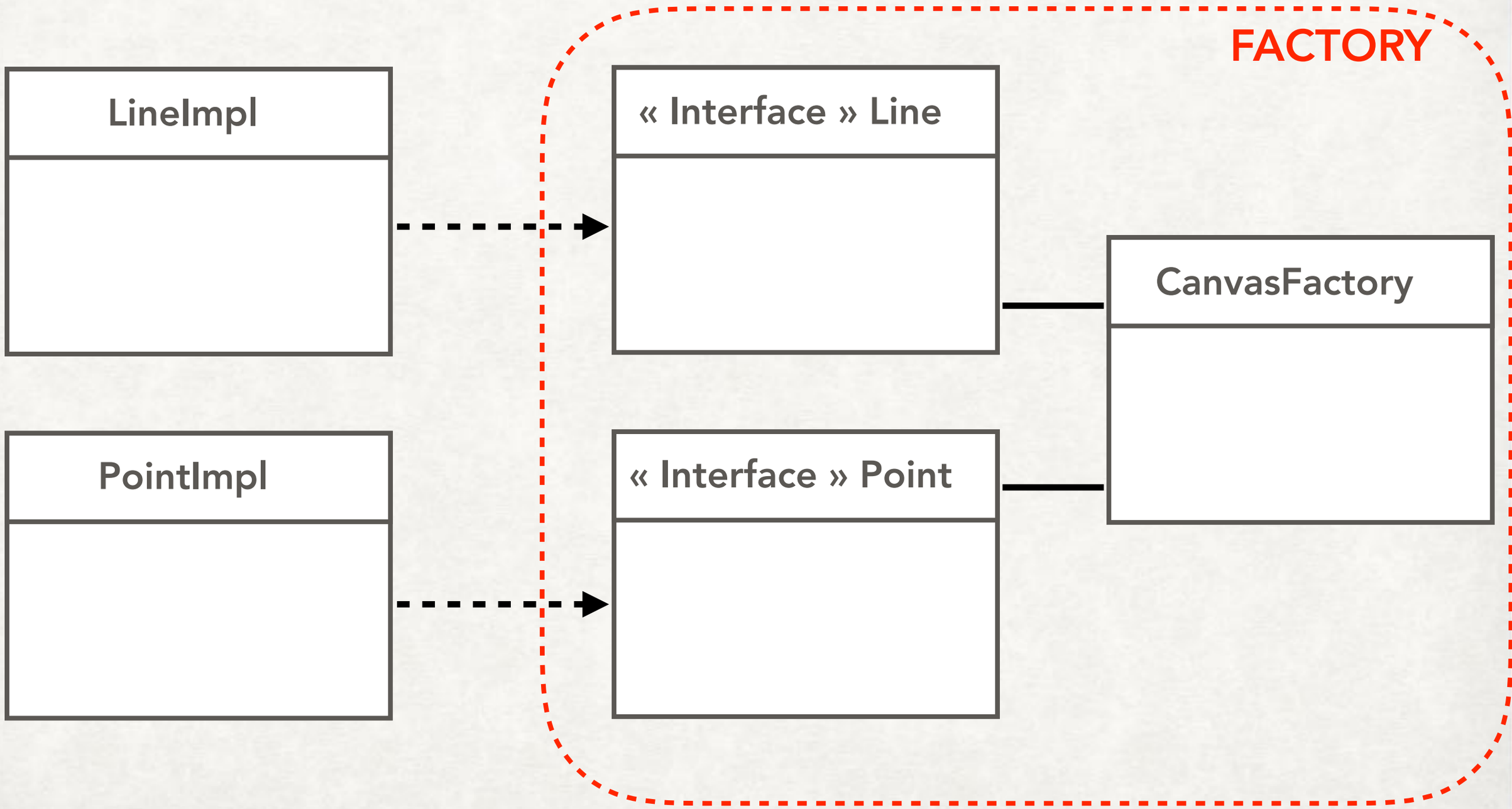
CREATIONAL PATTERNS

FACTORY



CREATIONAL PATTERNS

FACTORY



CREATIONAL PATTERNS

FACTORY

CREATIONAL PATTERNS

FACTORY

```
/** Interface de description d'un point */
public interface Point {
    /** Retourne l'abscisse du point */
    public int getX();
    /** Retourne l'ordonnée du point */
    public int getY();
}
```

CREATIONAL PATTERNS

FACTORY

```
/** Interface de description d'un point */
public interface Point {
    /** Retourne l'abscisse du point */
    public int getX();
    /** Retourne l'ordonnée du point */
    public int getY();
}
```

```
/** Interface de description d'une ligne */
public interface Line {
    /** Retourne les coordonnées du premier point */
    public int getX1();
    public int getY1();
    /** Retourne les coordonnées du deuxième point */
    public int getX2();
    public int getY2();
}
```

CREATIONAL PATTERNS

FACTORY

```
/** Interface de description d'un point */
public interface Point {
    /** Retourne l'abscisse du point */
    public int getX();
    /** Retourne l'ordonnée du point */
    public int getY();
}
```

```
/** Interface de description d'une ligne */
public interface Line {
    /** Retourne les coordonnées du premier point */
    public int getX1();
    public int getY1();
    /** Retourne les coordonnées du deuxième point */
    public int getX2();
    public int getY2();
}
```

```
/** Fabrique retournant des objets de types point ou ligne */
public class CanvasFactory {
    /** Retourne un Point aux coordonnées x,y */
    public Point getPoint( int x, int y ) {
        return new PointImpl( x, y );
    }
    /** Retourne une Ligne aux coordonnées x1,y1,x2,y2 */
    public Line getLine( int x1, int y1, int x2, int y2 ) {
        return new LineImpl( x1, y1, x2, y2 );
    }
}
```

CREATIONAL PATTERNS

SINGLETON

CREATIONAL PATTERNS

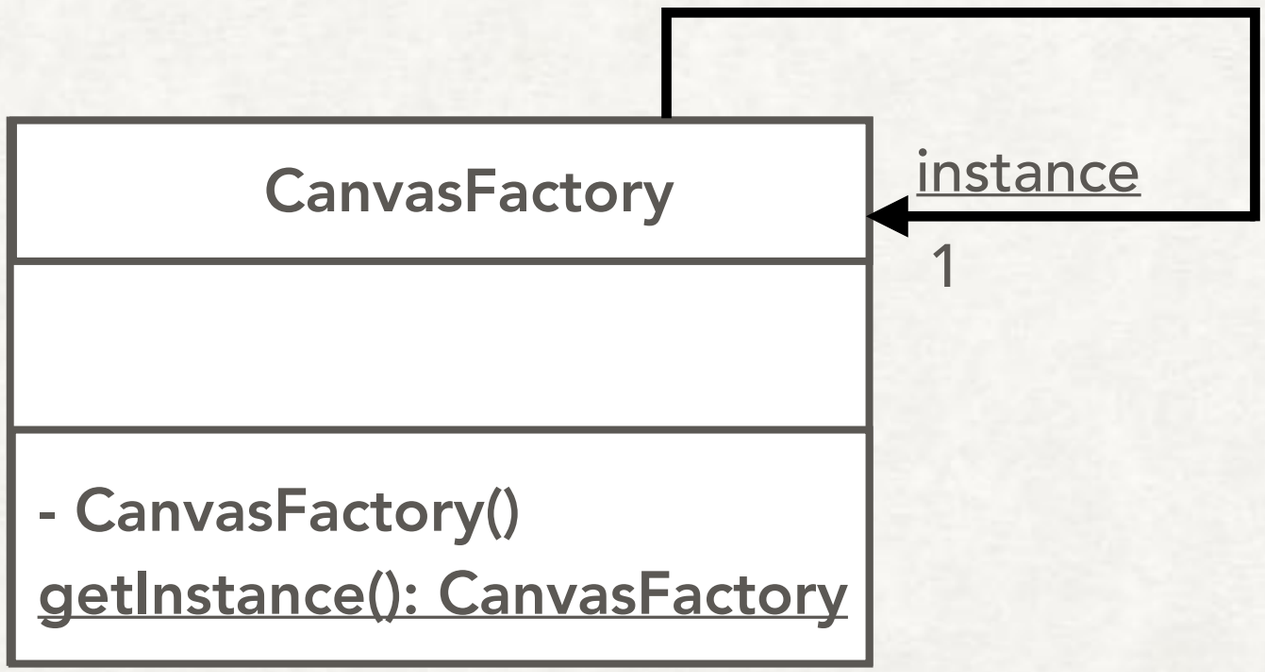
SINGLETON

- Contrôler le nombre d'instances d'un pattern et/ou une classe
- Pratique pour les classes sans état et effectuant un même traitement
- limiter le nombre d'instance en mémoire

CREATIONAL PATTERNS

SINGLETON

- Contrôler le nombre d'instances d'un pattern et/ou une classe
- Pratique pour les classes sans état et effectuant un même traitement
- limiter le nombre d'instance en mémoire



CREATIONAL PATTERNS

SINGLETON

- Contrôler le nombre d'instances d'un pattern et/ou une classe
- Pratique pour les classes sans état et effectuant un même traitement
- limiter le nombre d'instance en mémoire

CREATIONAL PATTERNS

SINGLETON

- Contrôler le nombre d'instances d'un pattern et/ou une classe
- Pratique pour les classes sans état et effectuant un même traitement
- limiter le nombre d'instance en mémoire

CREATIONAL PATTERNS

SINGLETON

- Contrôler le nombre d'instances d'un pattern et/ou une classe
- Pratique pour les classes sans état et effectuant un même traitement
- limiter le nombre d'instance en mémoire

```
public class CanvasFactory {
/** Donnée de classe contenant l'instance courante */
private static CanvasFactory instance = new CanvasFactory();
/** Constructeur privé interdisant l'instanciation en dehors de cette classe */
private CanvasFactory () {}
/** Singleton de la classe courante */
public static CanvasFactory getInstance() { return instance; }
...
}
```

CREATIONAL PATTERNS

SINGLETON

- Contrôler le nombre d'instances d'un pattern et/ou une classe
- Pratique pour les classes sans état et effectuant un même traitement
- limiter le nombre d'instance en mémoire

```
public class CanvasFactory {
/** Donnée de classe contenant l'instance courante */
private static CanvasFactory instance = new CanvasFactory();
/** Constructeur privé interdisant l'instanciation en dehors de cette classe */
private CanvasFactory () {}
/** Singleton de la classe courante */
public static CanvasFactory getInstance() { return instance; }
...
}
```

```
CanvasFactory cf = CanvasFactory.getInstance();
```

STRUCTURAL PATTERNS

ADAPTER

STRUCTURAL PATTERNS

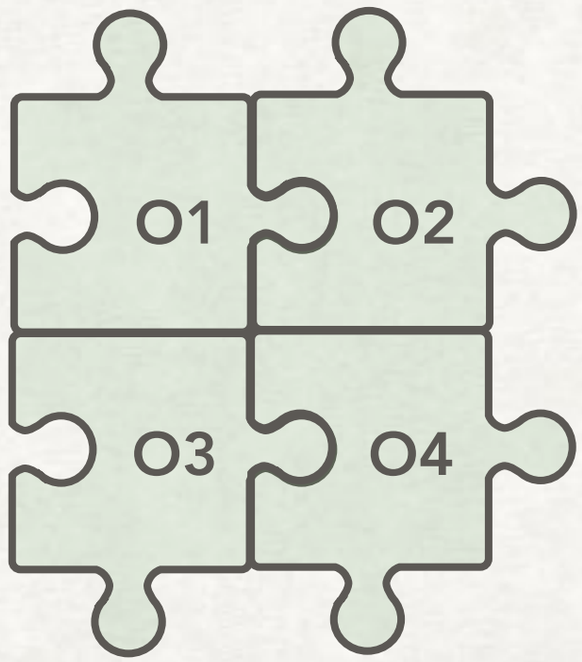
ADAPTER

- Construire de nouvelles structures à partir de patterns
- Un objet peut être constitué d'un groupe d'objets
- limiter le nombre d'instance en mémoire

STRUCTURAL PATTERNS

ADAPTER

- Construire de nouvelles structures à partir de patterns
- Un objet peut être constitué d'un groupe d'objets
- limiter le nombre d'instance en mémoire

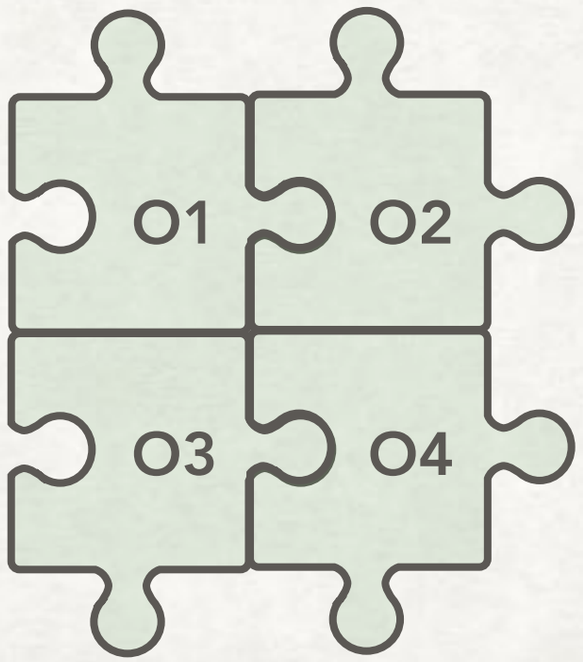


Objet X

STRUCTURAL PATTERNS

ADAPTER

- Construire de nouvelles structures à partir de patterns
- Un objet peut être constitué d'un groupe d'objets
- limiter le nombre d'instance en mémoire



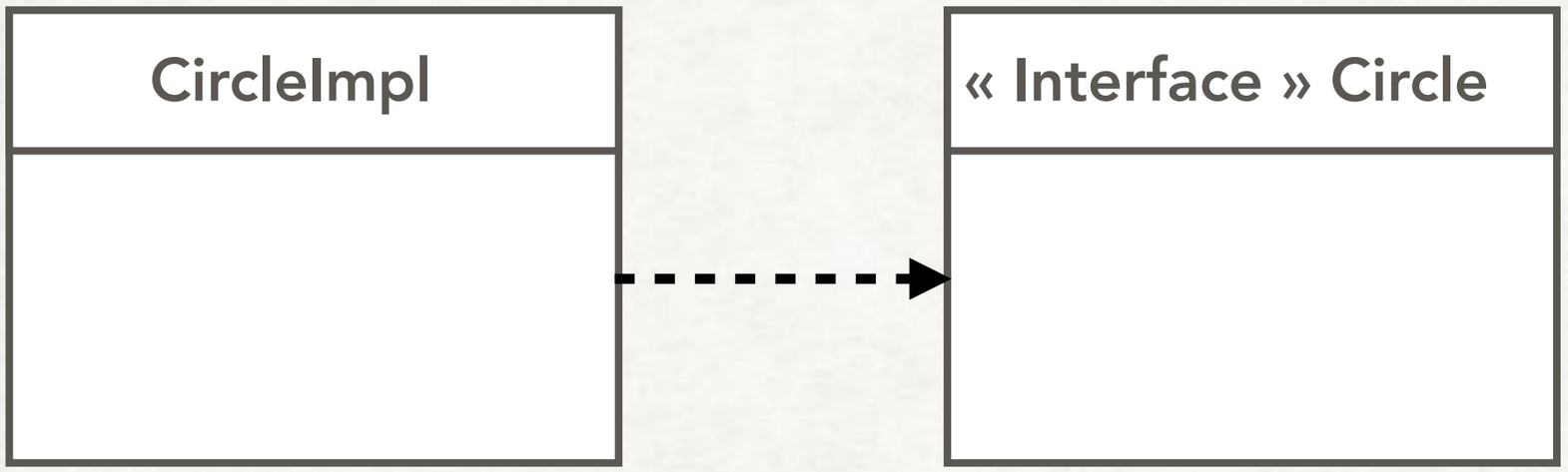
Objet X

STRUCTURAL PATTERNS

ADAPTER

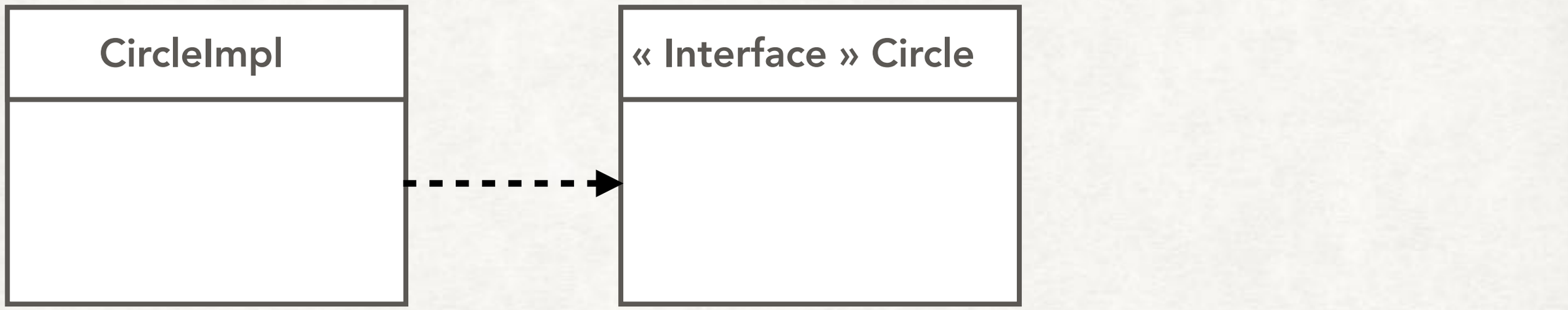
STRUCTURAL PATTERNS

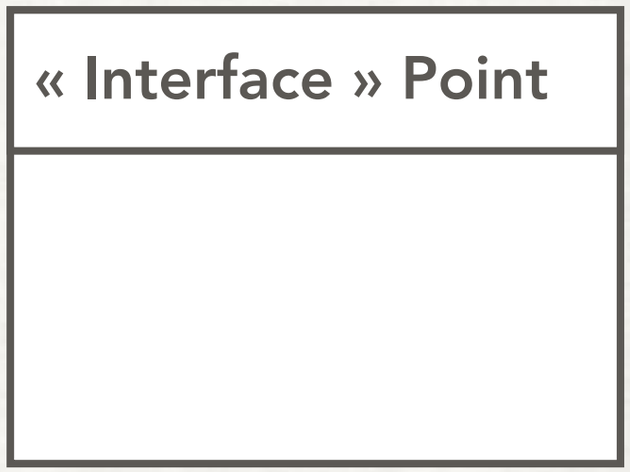
ADAPTER



STRUCTURAL PATTERNS

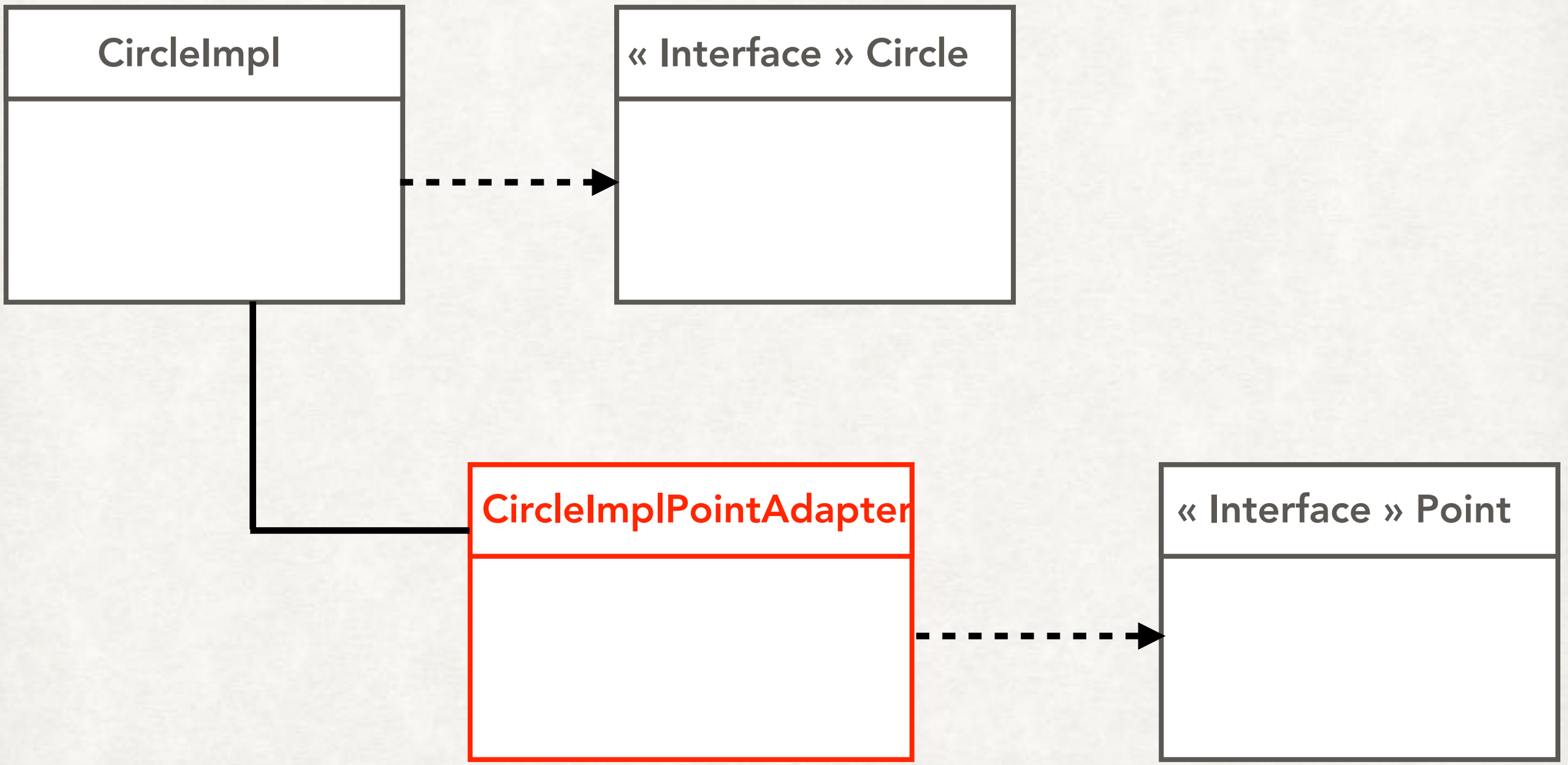
ADAPTER





STRUCTURAL PATTERNS

ADAPTER



STRUCTURAL PATTERNS

ADAPTER

STRUCTURAL PATTERNS

ADAPTER

```
/** Interface de représentation d'un cercle */
public interface Circle {
    /** Retourne l'abscisse du centre du cercle */
    public int getX();
    /** Retourne l'ordonnée du centre du cercle */
    public int getY();
    /** Retourne le rayon du cercle */
    public int getR();
}
```

STRUCTURAL PATTERNS

ADAPTER

```
/** Interface de représentation d'un cercle */
public interface Circle {
    /** Retourne l'abscisse du centre du cercle */
    public int getX();
    /** Retourne l'ordonnée du centre du cercle */
    public int getY();
    /** Retourne le rayon du cercle */
    public int getR();
}
```

```
/** Classe implémentant l'interface Circle */
public class CircleImpl implements Circle {
...
}
```

STRUCTURAL PATTERNS

ADAPTER

```
/** Interface de représentation d'un cercle */
public interface Circle {
    /** Retourne l'abscisse du centre du cercle */
    public int getX();
    /** Retourne l'ordonnée du centre du cercle */
    public int getY();
    /** Retourne le rayon du cercle */
    public int getR();
}
```

```
/** Classe implémentant l'interface Circle */
public class CircleImpl implements Circle {
...
}
```

```
/** Adapteur pour transformer le circle en un point */
public class CircleImplPointAdapter implements Point {
    private Circle c;
    public CircleImplPointAdapter( Circle c ) {
        this.c = c;
    }
    public int getX() { return c.getX(); }
    public int getY() { return c.getY(); }
}
```

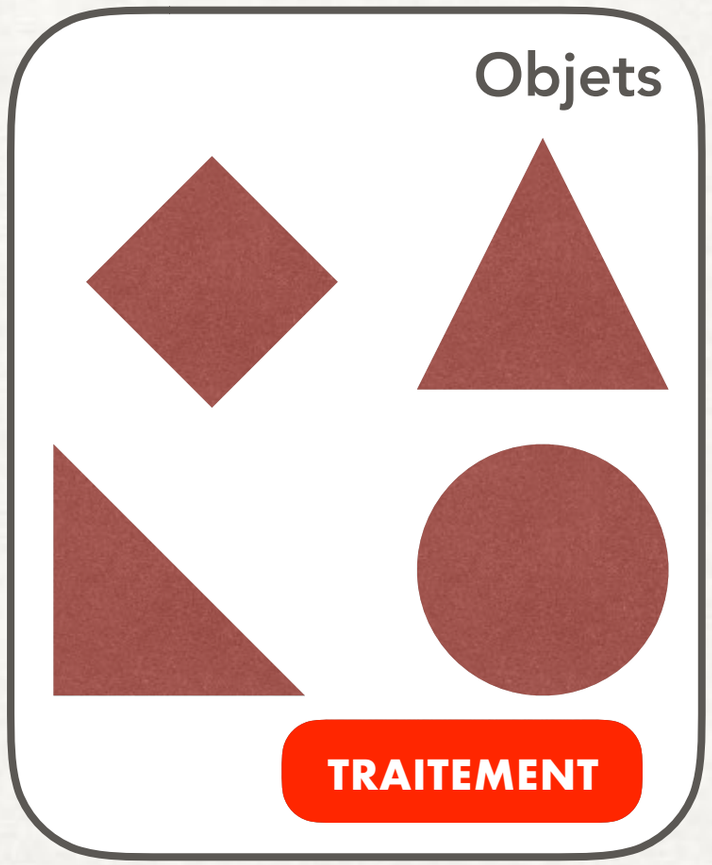
BEHAVIORAL PATTERNS

BEHAVIORAL PATTERNS

- Behavioral patterns facilitent l'organisation d'un traitement sur un ensemble d'objets

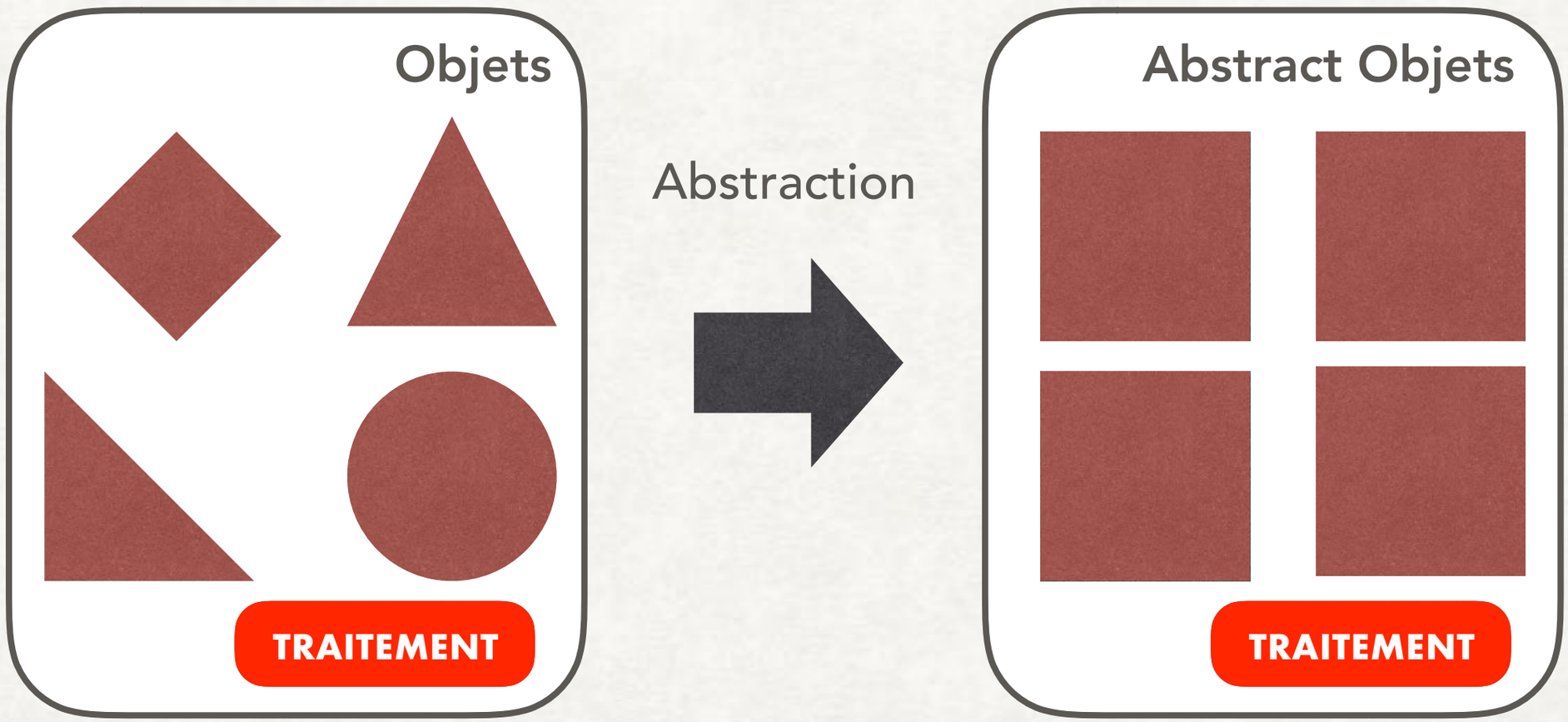
BEHAVIORAL PATTERNS

- Behavioral patterns facilitent l'organisation d'un traitement sur un ensemble d'objets



BEHAVIORAL PATTERNS

- Behavioral patterns facilitent l'organisation d'un traitement sur un ensemble d'objets

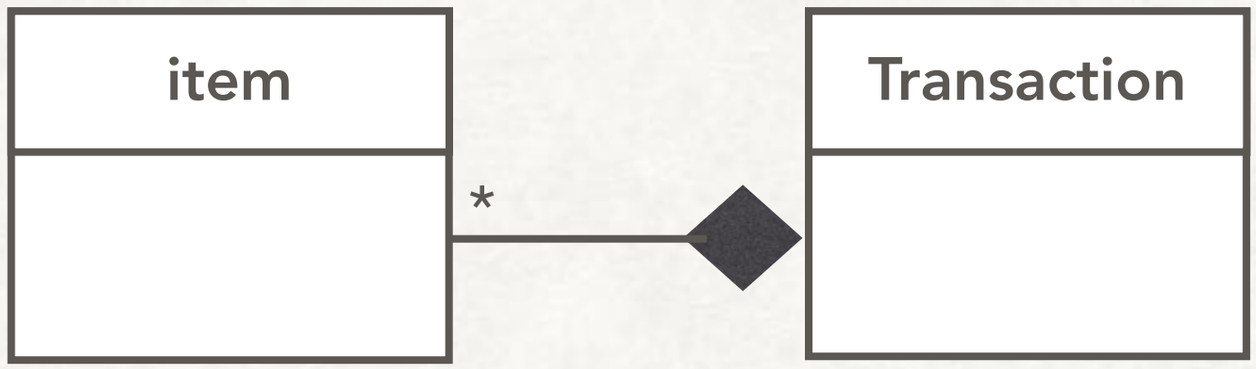


BEHAVIORAL PATTERNS

ITERATOR

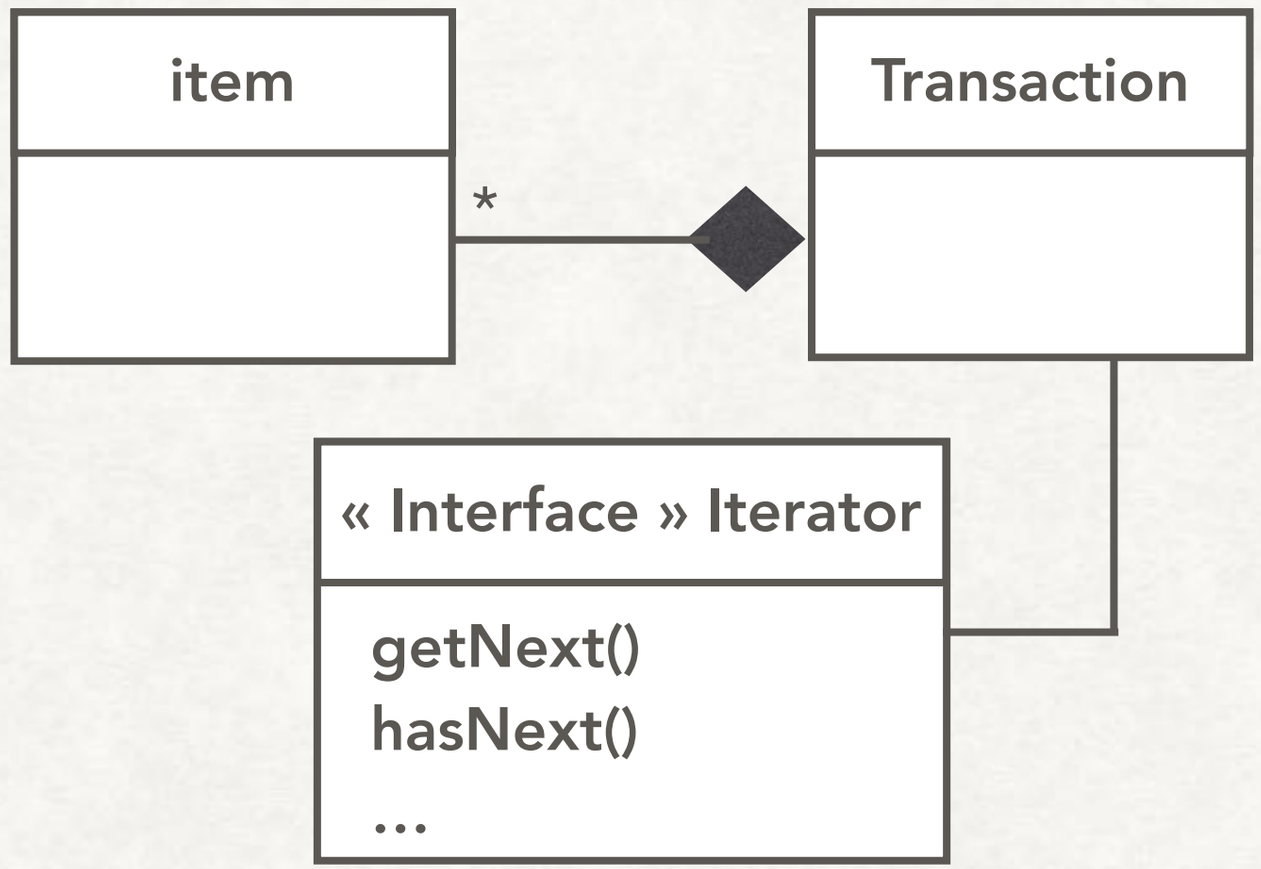
BEHAVIORAL PATTERNS

ITERATOR



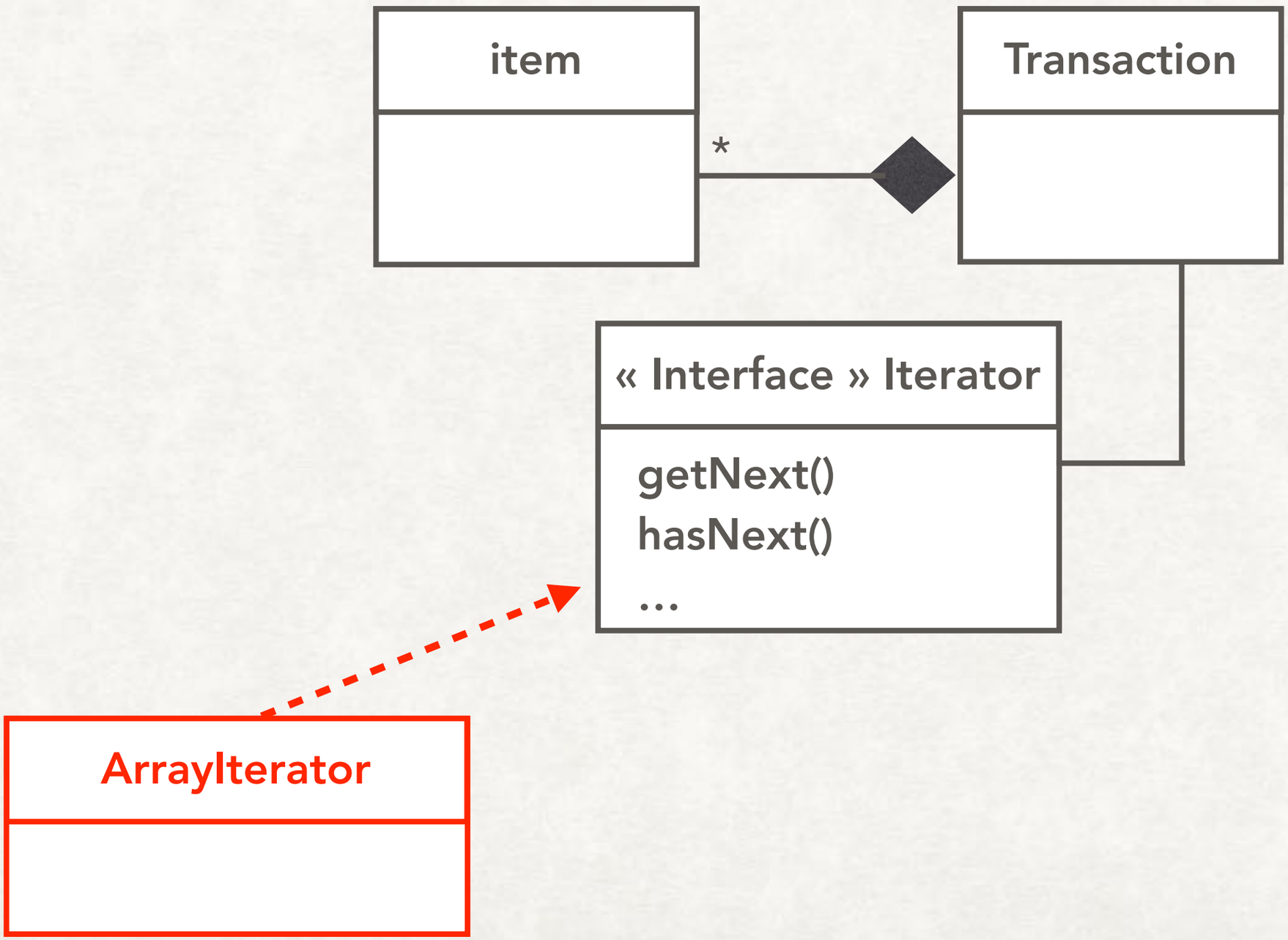
BEHAVIORAL PATTERNS

ITERATOR



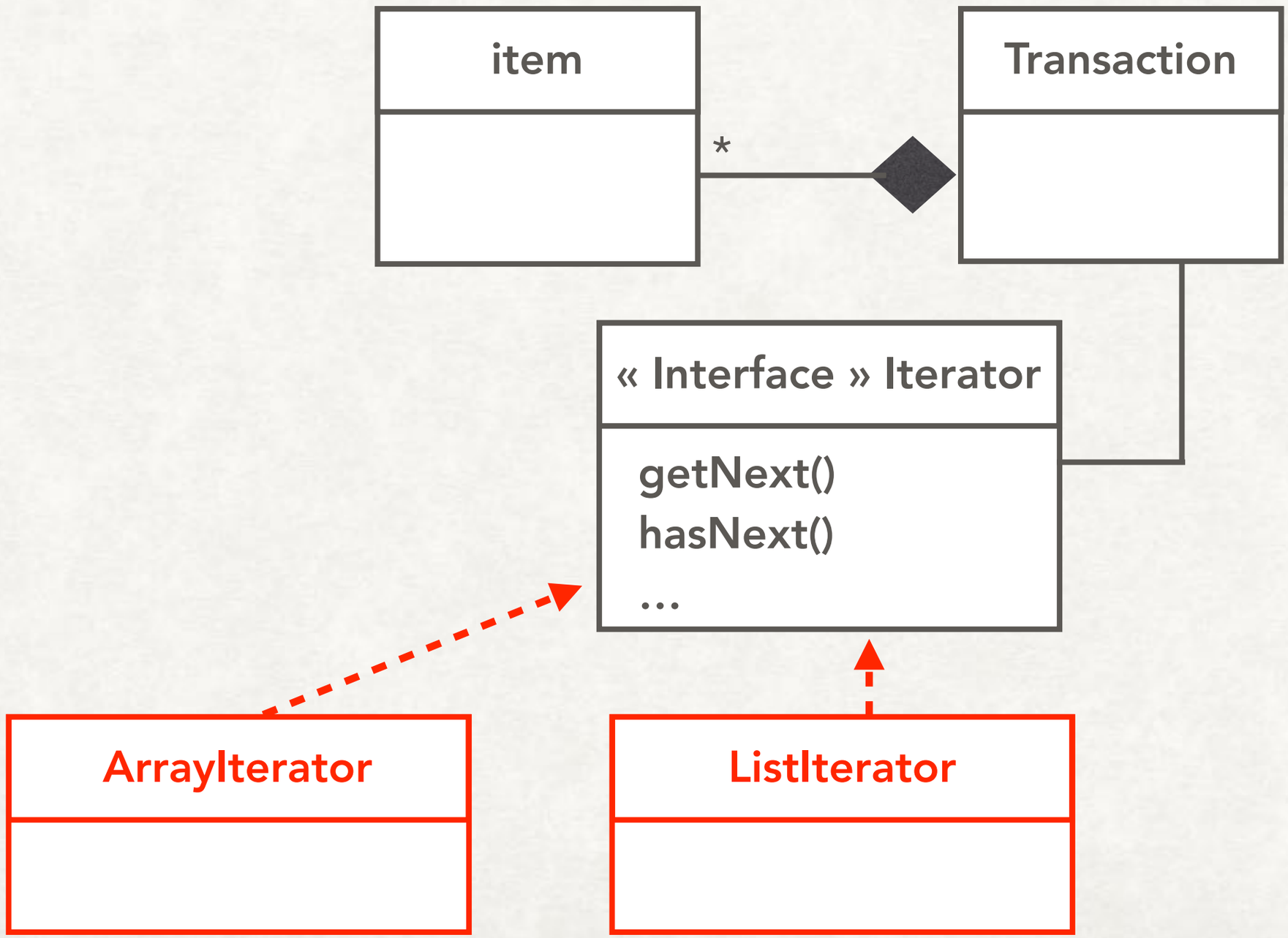
BEHAVIORAL PATTERNS

ITERATOR



BEHAVIORAL PATTERNS

ITERATOR



BEHAVIORAL PATTERNS

ITERATOR