Les modèles de comportements

- Ce rôle désigne tous les patterns dont le rôle est de spécifier :
  - la façon dont des classes ou des objets interagissent,
  - la répartition des responsabilités de différents objets.

Les modèles comportementaux

- Dans cette famille, nous étudierons les patterns suivants :

- **Chaîne de responsabilités**
- **Commande**
- **Observateur**
- **Modèle Vue Contrôleur**
- **Stratégie**
- **Etat**
- **Visiteur**
- **Médiateur**
- **Itérateur**
- **Interpréteur**

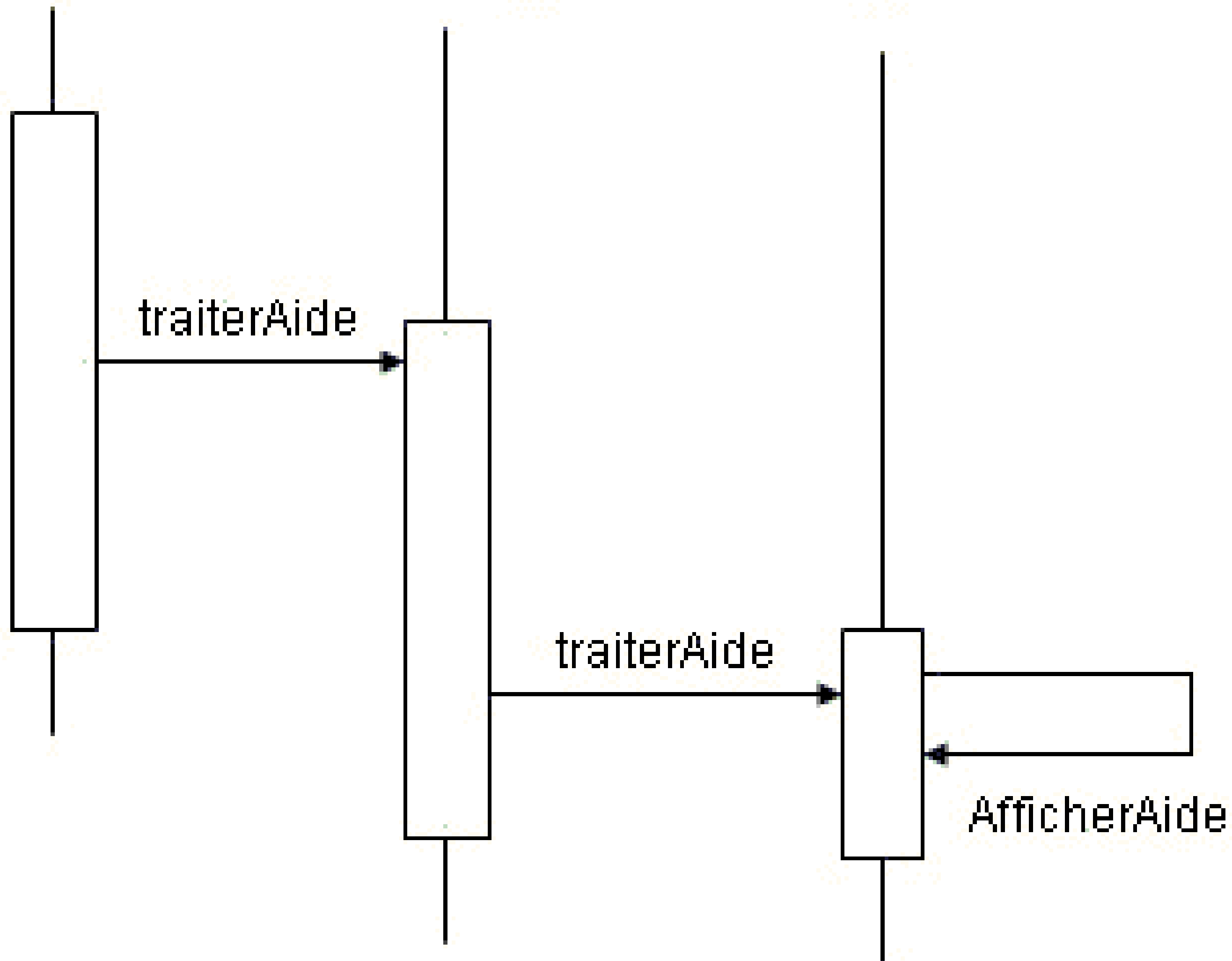
Le pattern Chaîne de responsabilités

- **Motivation**
  - Dans une IHM, on veut gérer une aide contextuelle.
  - Quand l'utilisateur clique dans un composant avec le bouton droit, il désire une aide : de deux choses l'une
    - Une aide est prévue sur le composant, et on l'affiche
    - On demande au composant immédiatement englobant de traiter la requête

Motivation (Ch. de resp.)

- **Question ?**
  - Faire le diagramme de séquences relatif au scénario :
    « l'utilisateur a cliqué sur un widget bouton marqué "imprimé" pour demander une aide »
  - On suppose que ce bouton est dans une boite de dialogue elle même dans la fenêtre d'application et que l'aide est concentrée dans cette fenêtre.

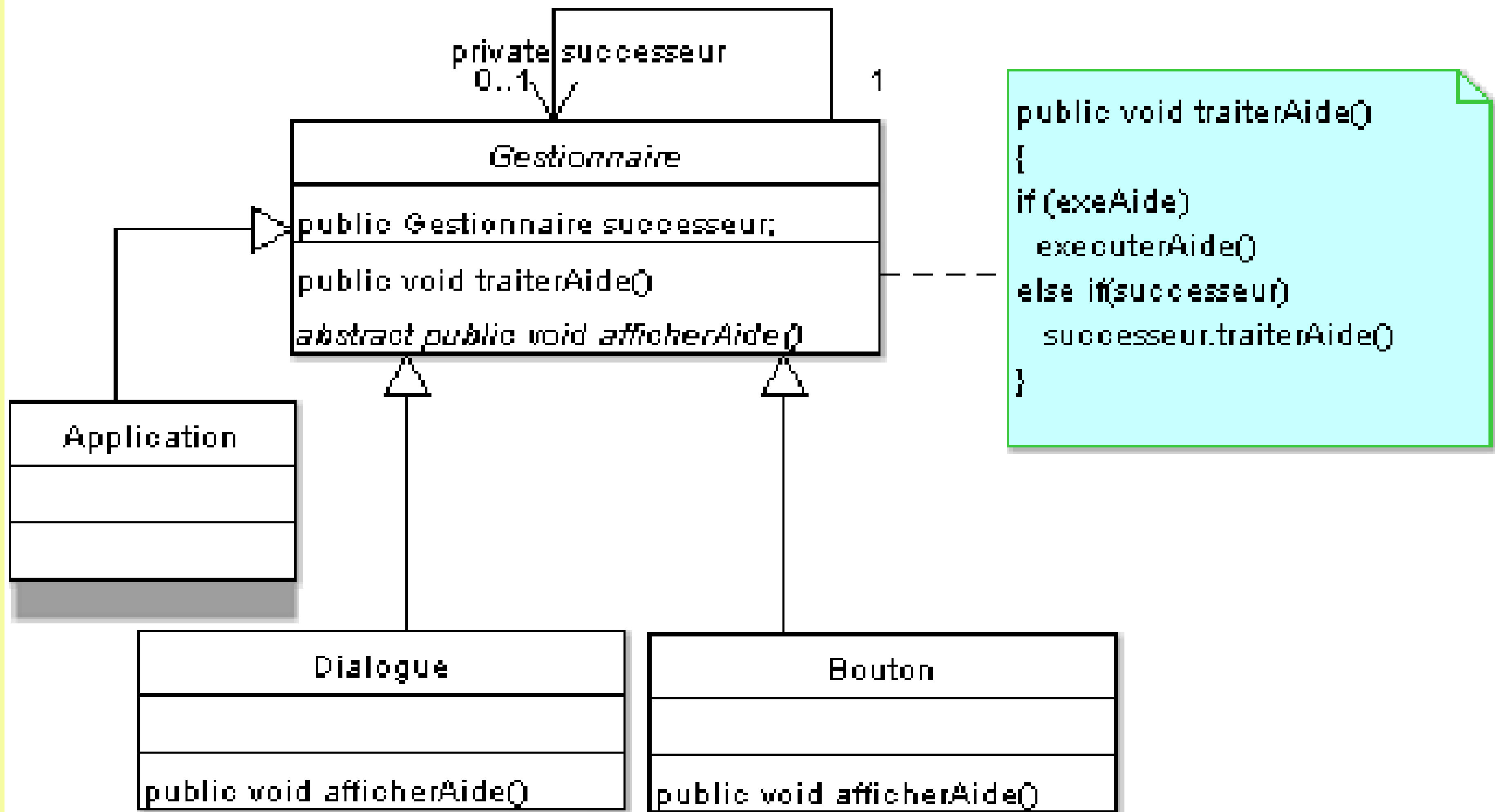
Motivation (Ch. de resp.)



Motivation (Ch. de resp.)

- Question ?
  - Faire le diagramme de classes correspondant à l'énoncé

Motivation (Ch. de resp.)

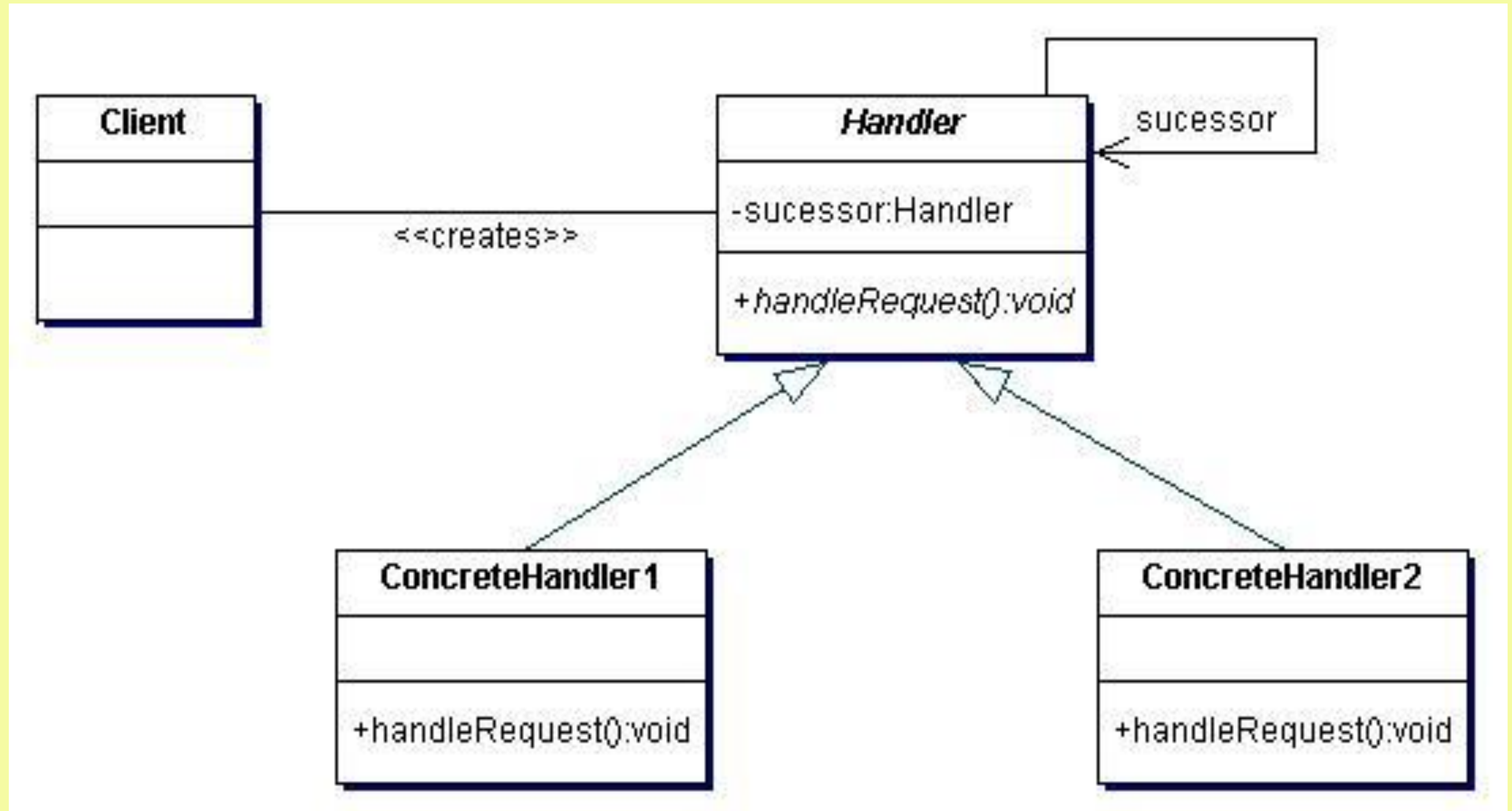


Le pattern Chaîne de responsabilités

- **Intention**
  - Ce pattern permet d’éviter le couplage entre l’émetteur d’une requête et les récepteurs.
  - L’émétteur ne doit pas connaître le premier récepteur et plus largement toute la chaîne de récepteurs qui traite la requête.

Le pattern Chaîne de responsabilités

- **Structure**



Le pattern Chaîne de responsabilités

- **Constituants**
  - **Une interface nommée Handler qui définit les prototypages des requêtes du client**
  - **Des classes gestionnaires concrètes (composants graphiques dans notre exemple) qui implémentent Handler**
  - **Une classe Client qui propose la requête à un objet gestionnaire du début de la chaîne**

Le pattern Chaîne de responsabilités

- **Implémentation**

```
public void handleRequest() {
   // Je fais ce que je sais faire
   ….
   // Si j'ai un successeur, il traite aussi
   if (successor != null)
      successor.handleRequest();
}
```

Le pattern Iterateur

- **Motivation**
  - On peut vouloir parcourir un conteneur de plusieurs façons différentes
    - Parcourir un arbre en profondeur ou parcourir un arbre en largeur
    - Parcourir une liste du premier élément au dernier élément ou parcourir du dernier élément au premier

Le pattern Iterateur

- **Motivation**
  - Au même instant, on peut avoir en cours, plusieurs parcours d'un même conteneur
  - On peut enfin vouloir parcourir un conteneur sans risque de modification intempestive de la structure interne du conteneur

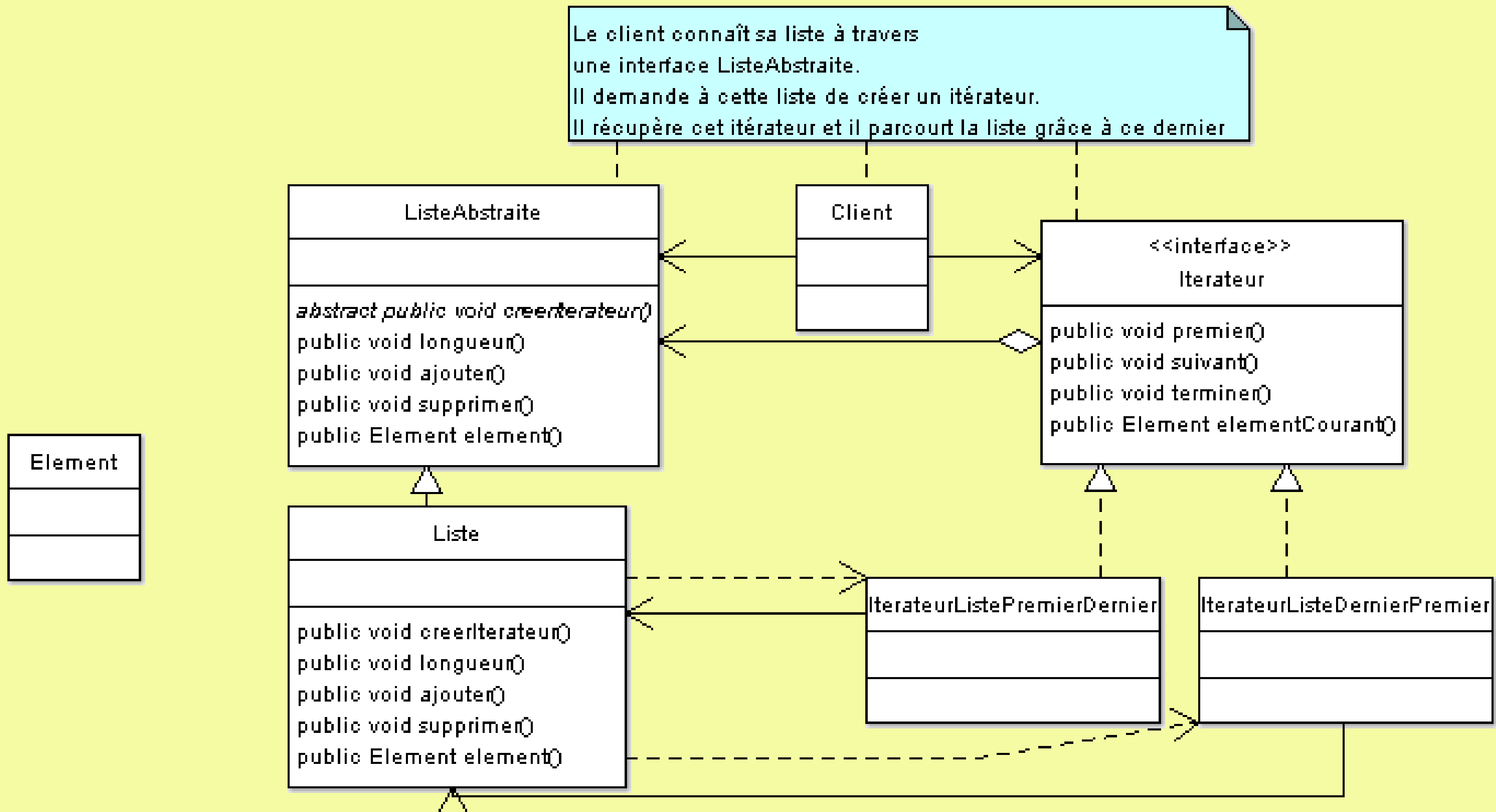
Le pattern Iterateur

- **Intention**

  Fournir un moyen d'accès séquentiel à un agrégat d'objets sans mettre à découvert la représentation interne de ce dernier

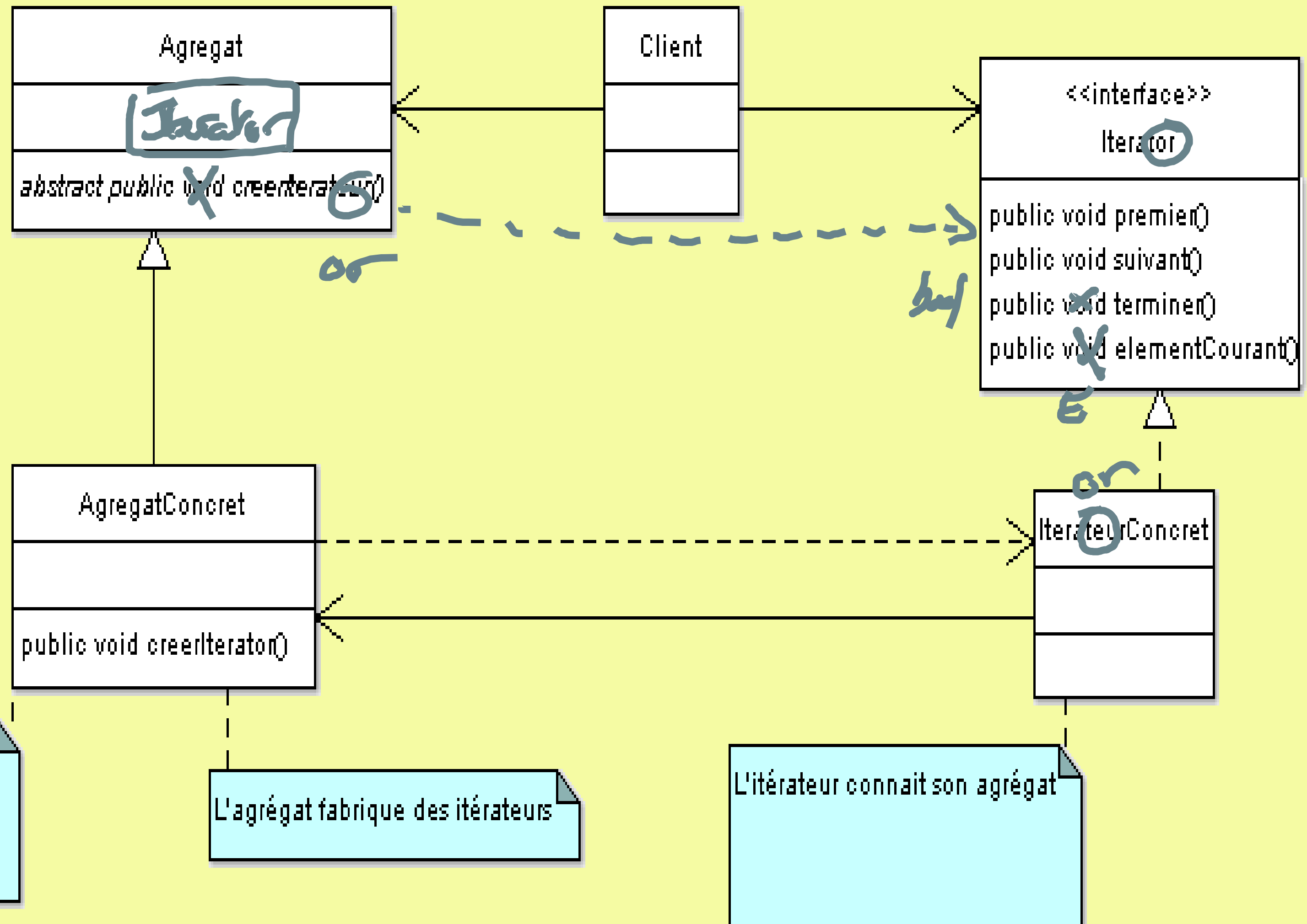
Le pattern Iterateur

La structure du Pattern et ses composants avec une liste



Le pattern Iterateur

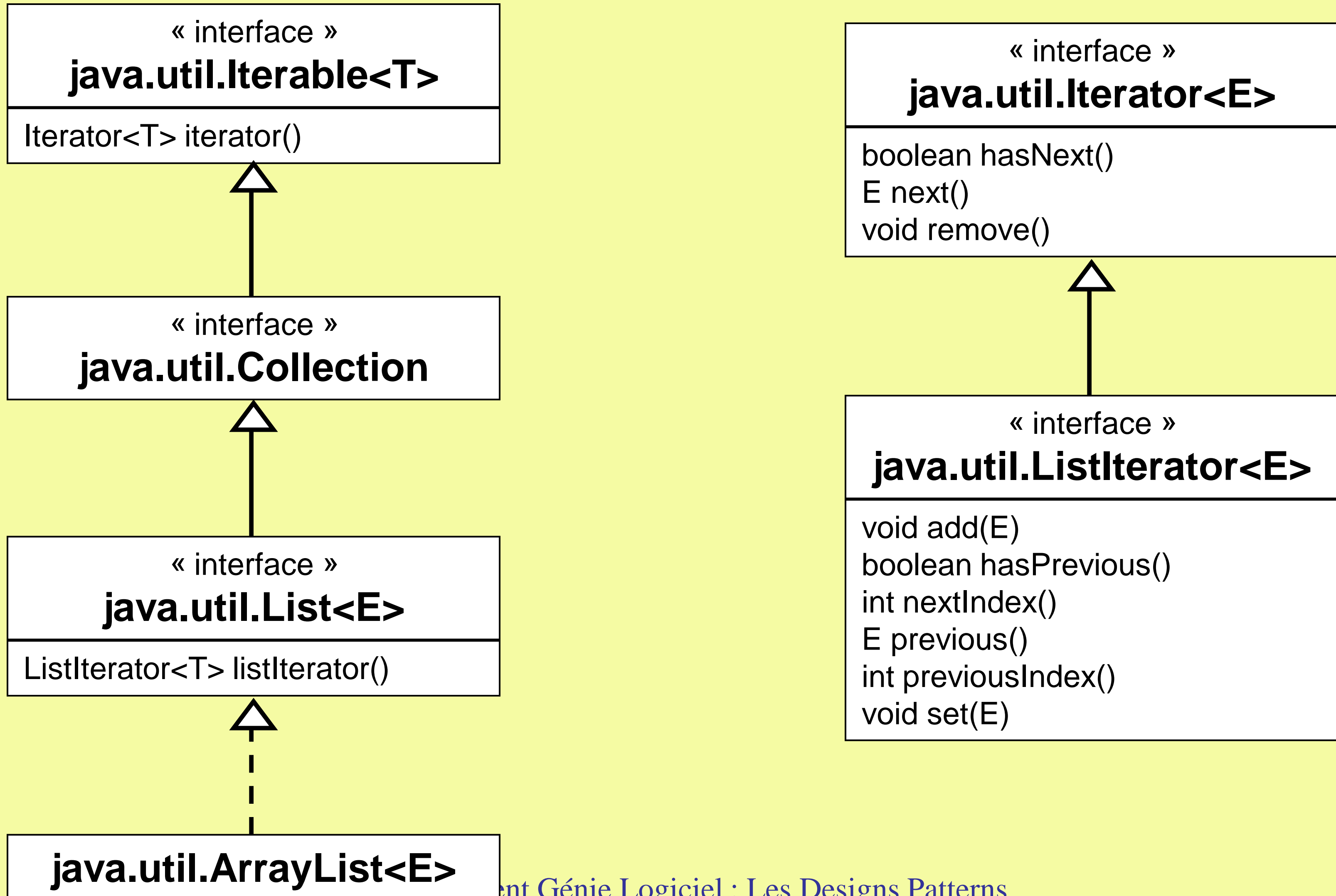
La structure générique du Pattern et ses composants



Le pattern Iterateur

- **Composant**
  - La classe IterateurNul est un itérateur dégénéré qui est toujours terminé
  - Les itérateurs en Java

Le pattern Iterateur



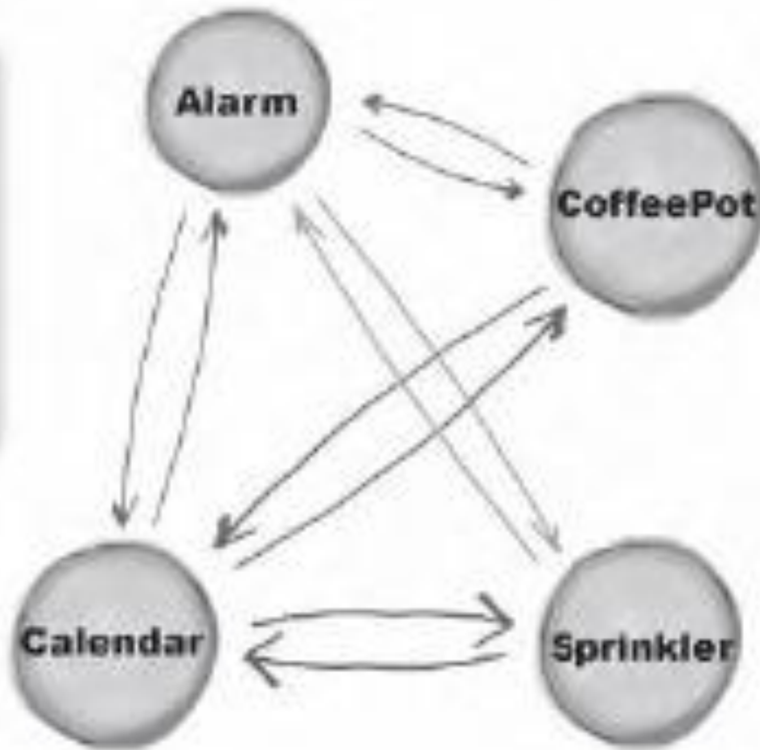
Le pattern Iterateur

```
ArrayList<String> al = new ArrayList<String>();
al.add("C");al.add("A");al.add("E");
al.add("B");al.add("D");al.add("F");

    System.out.print("Contenu initial de al : ");
    Iterator<String> itr = al.iterator();
    while (itr.hasNext()) {
      String element = itr.next();
      System.out.print(element + " ");
    }
    System.out.println();

    ListIterator<String> litr = al.listIterator();
    while (litr.hasNext()) {
      String element = litr.next();
      litr.set(element + "+");
    }

    System.out.print("Liste inversée modifiée : ");
    while (litr.hasPrevious()) {
      String element = litr.previous();
      System.out.print(element + " ");
    }
```

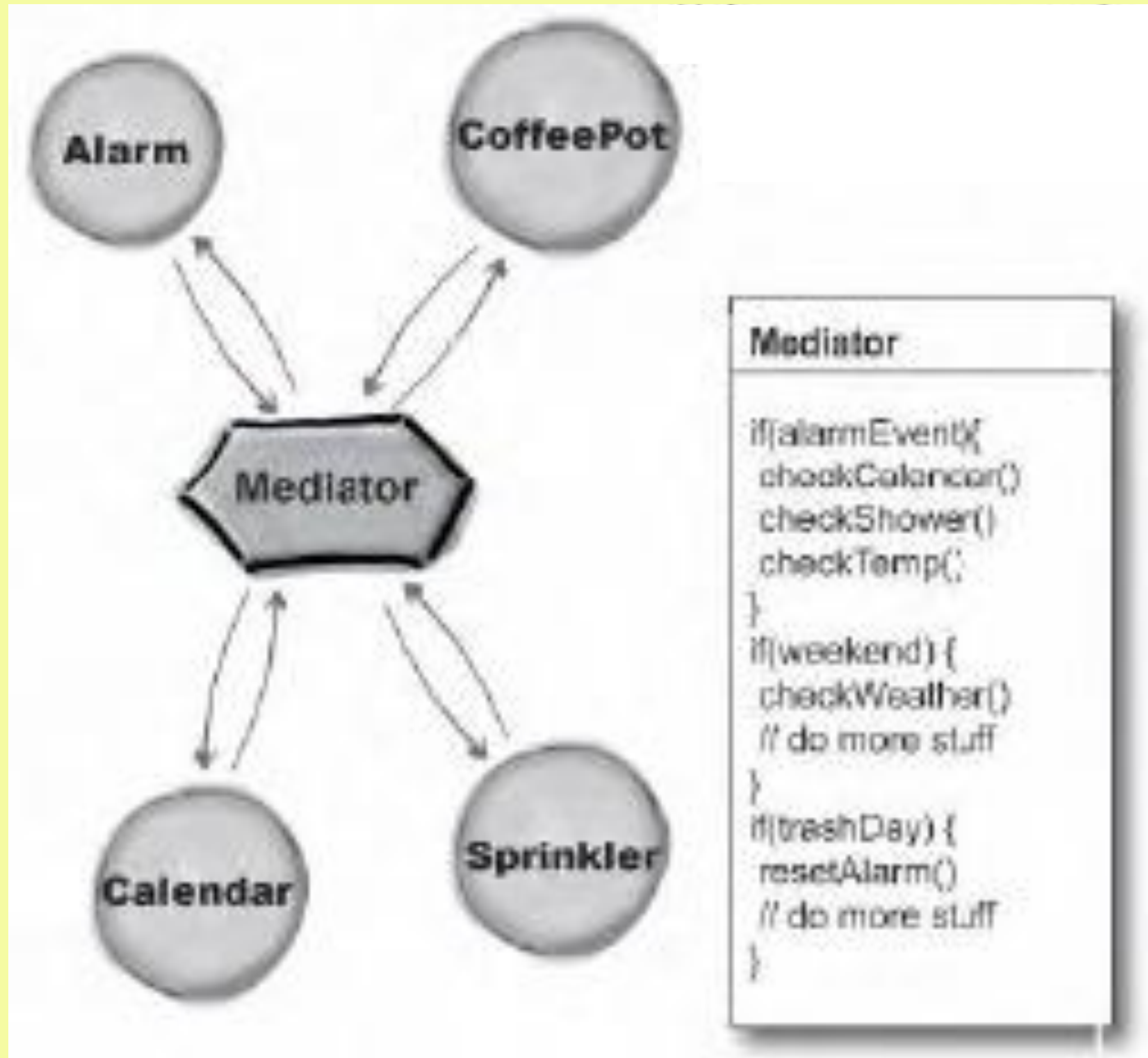
Le pattern Médiateur

- Motivation
  - Un système domotique du futur
    - L’arrêt du réveil déclenche la machine à café
    - Cela doit fonctionner tous les jours sauf le week-end
    - L’arrosage automatique doit s’arrêter 15 min avant la programmation d’une douche
    - Mettre le réveil plus tôt le jour des poubelles
    - …

Le pattern Médiateur



Le pattern Médiateur



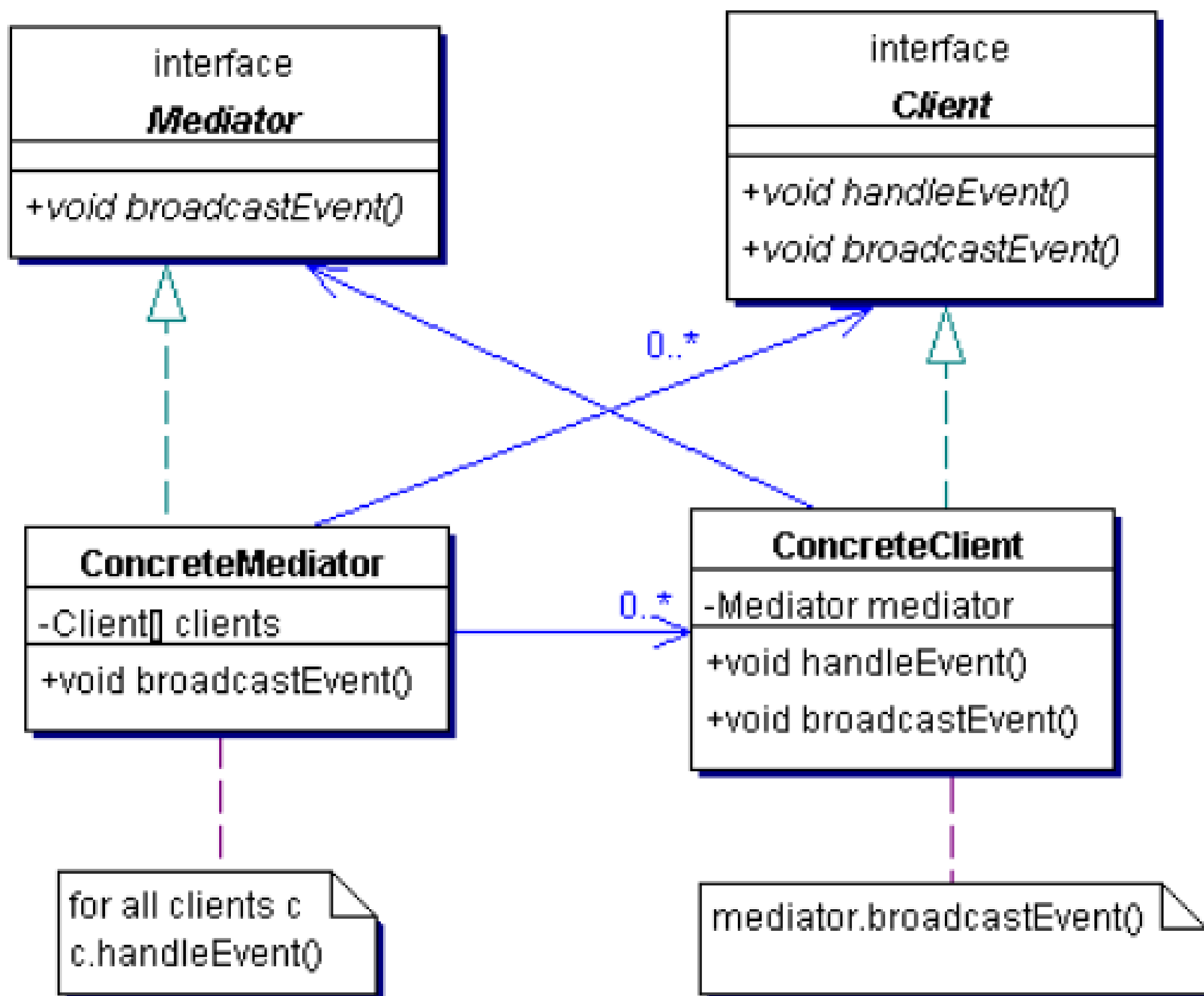
Le pattern Médiateur

- Intention
  - Encapsuler dans un objet m les modalités d'interaction d'un grand ensemble d'objets E
  - Cet objet m est le médiateur
  - Il permet donc un couplage faible en dispensant les objets de E de se faire explicitement référence

Le pattern Médiateur

- Constituants
  - une interface Médiateur pour déclarer les opérations qui permettront aux objets dit collègues de communiquer entre eux
  - une classe MédiateursConcrets qui implémente l'interface Médiateur. Un objet de cette classe a des références sur les différents collègues à gérer
  - les classes Collègues qui vérifient les propriétés suivantes :
    - Un objet Collègue connait son médiateur
    - Un objet Collègue communique avec un autre objet Collègue par envoi de message à son médiateur

Le pattern Médiateur



Le pattern Interpréteur

- Motivation
  - On désire pour un langage donné
    - définir une représentation de sa grammaire
    - définir un interpréteur utilisant cette représentation pour interpréter les phrases du langage

Le pattern Interpréteur

- Motivation
  - Soit le langage

```
- expression ::= litteral | alternation |
  sequence | repetition | '(' expression ')'
- alternative ::= expression '|' expression
- sequence ::= expression '&' expression
- repetition ::= expression '*'
- literal ::= 'a' | … | 'z' {'a' | … | 'z'}*
```

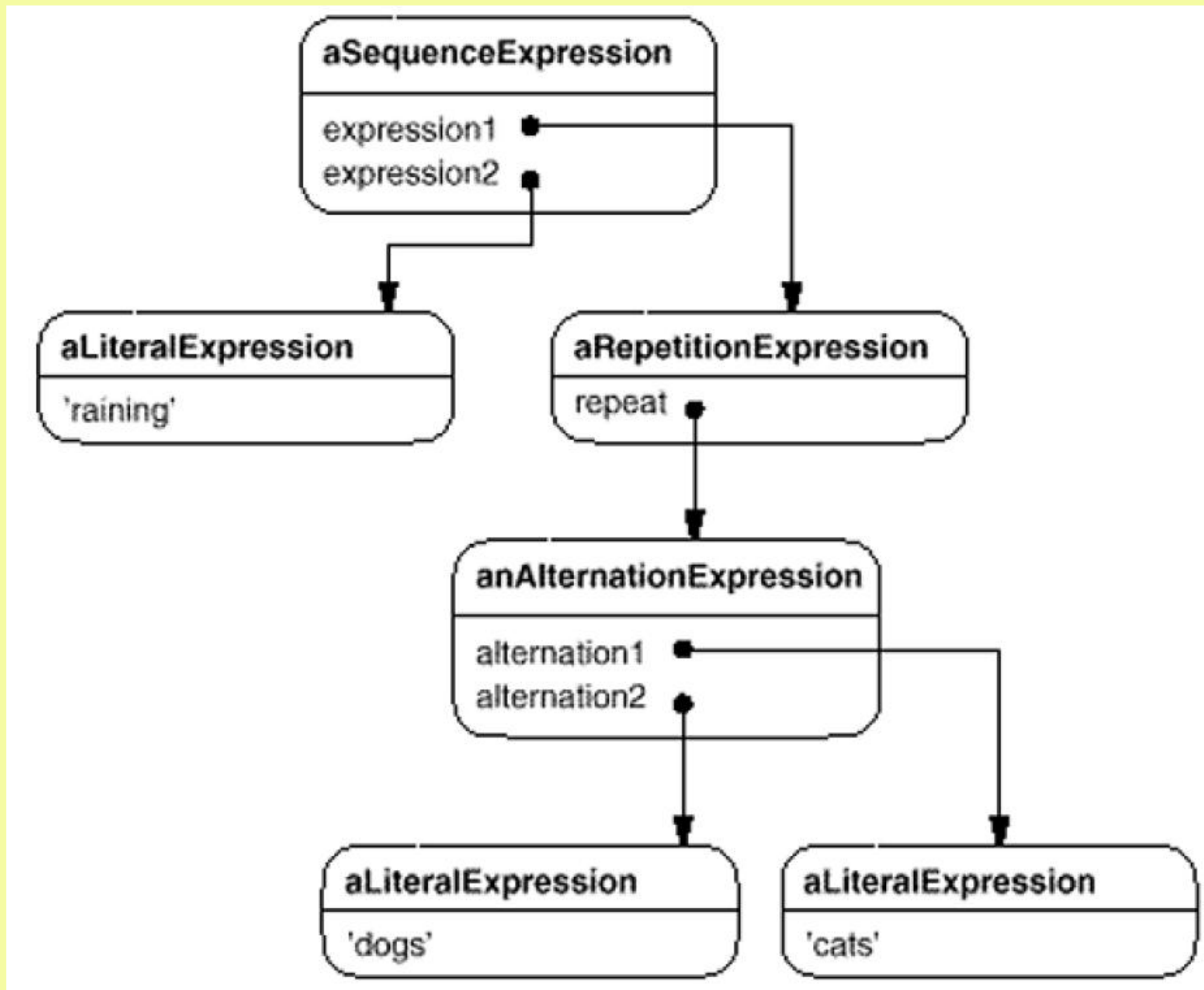
Le pattern Interpréteur

- Motivation
  - Le point de départ de l'analyse est l'expression non terminale (le point de départ constitue une classe particulière)
  - Chaque règle définit une classe
  - Les symboles de la partie de droite d'une règle qui définissent une forme sont des variables d'instance de ces classes

Le pattern Interpréteur

- Question ?
  - Exprimer l’expression régulière **`raining & (dogs | cats)`** sous forme d’un arbre syntaxique dont chaque nœud est une instance d’une des classes définies ci-dessus

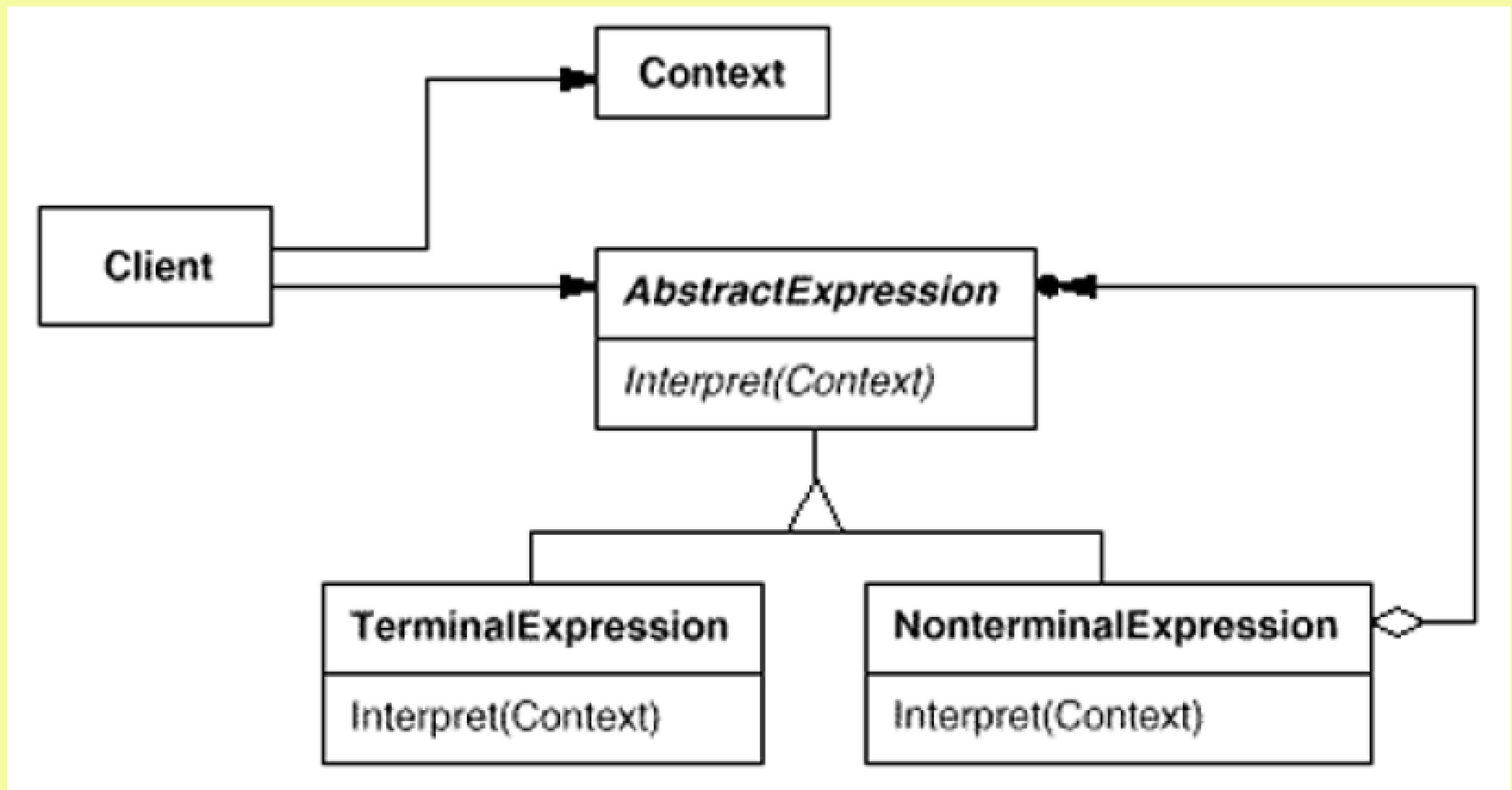
Le pattern Interpréteur



Le pattern Interpréteur

- Intention
  - Définir un interpréteur pour un langage

Le pattern Interpréteur



Le pattern Interpréteur

- Constituants
  - AbstractExpression
    - déclare une opération abstraite interpret() communes à tous les nœuds dans l'arbre de la syntaxe abstraite
  - TerminalExpression
    - implémente une opération interpret() associée à un symbole terminal dans la grammaire
    - une instance est requise pour chaque symbole terminal dans une phrase

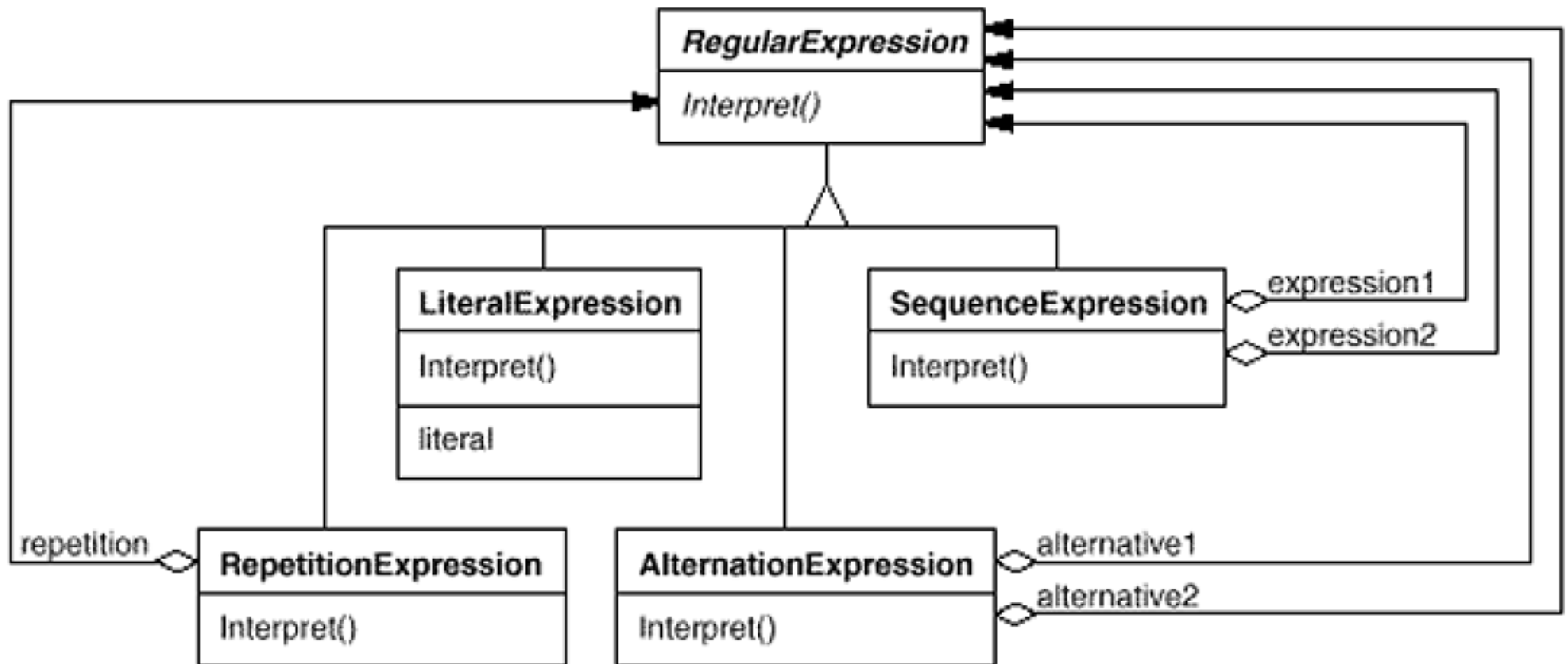
Le pattern Interpréteur

- Constituants
  - NonterminalExpression
    - une telle classe est requise pour chaque règle `R::=` $R_1R_2...R_n$ dans la grammaire
    - maintient des variables d'instance de type AbstractExpression pour chaque symbole de R_1 à R_n
    - implémente une opération interpret() les symboles non terminaux dans la grammaire. Typiquement, interpret() s'appelle récursivement sur les variables représentant $R_1,...,R_n$

Le pattern Interpréteur

- Constituants
  - Context
    - contient l'information globale à l'interpréteur
  - Client
    - construit (ou on lui donne) un arbre syntaxique abstrait représentant une phrase particulière du langage définie par la grammaire. L'arbre syntaxique abstrait est assemblé à partir des instances des classes NonterminalExpression et TerminalExpression
    - invoque l'opération interpret()

Le pattern Interpréteur



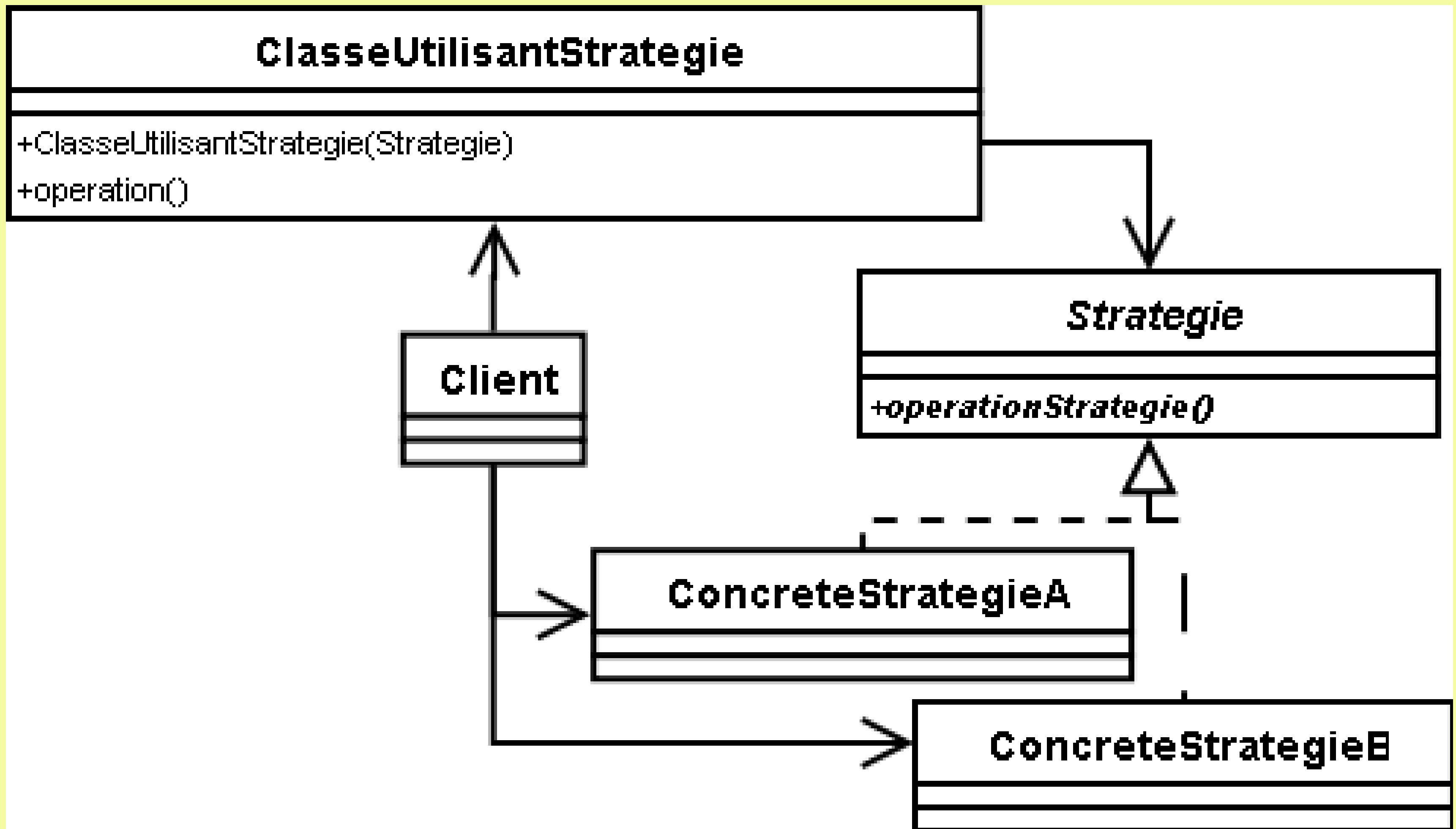
Le pattern Stratégie (Strategy or Policy)

- Motivation
  - Définir une famille d’algorithmes interchangeables
  - Permettre de les changer indépendamment de la partie cliente

Le pattern Stratégie

- Exemple
  - Une liste triée. A chaque insertion, la liste place le nouvel élément à l'emplacement correspondant au tri.
  - Le tri peut être alphabétique, inverse, les majuscules en premiers…
  - La partie de l'algorithme qui varie (le tri) est la stratégie utilisée au moyen de son *interface*.

Le pattern Stratégie



Le pattern Stratégie

- Constituants
  - Strategie (*Tri*)
    - définit l'interface commune des algorithmes.
  - ConcreteStrategieA et ConcreteStrategieB (*TriAlphabetique, TriMajuscule,…*)
    - implémentent les méthodes d'algorithme.
  - ClasseUtilisantStrategie (*ListeTriee*)
    - utilise un objet Strategie.
  - Client (*Application*)
    - configure un objet ClasseUtilisantStrategie avec un objet Strategie et appelle la méthode de ClasseUtilisantStrategie qui utilise la stratégie.