Les « Design Patterns »

Bruno Mermet
Université du Havre
2007-2008

Introduction

- Origine

  Design Patterns, Elements of Reusable Object-Oriented Software, E. Gamma, R. Helm, R. Johnson, J. Vlissides, Addison-Wesley, 1994.

- Définition

  Un design pattern est un "motif de conception". Il est là pour répondre à un problème récurrent. Il ne s'agit pas d'une solution toute faite mais d'une architecture générale

- Caractéristiques essentielles
  - Nom
  - Problème
  - Solution
  - Exemples

Classification

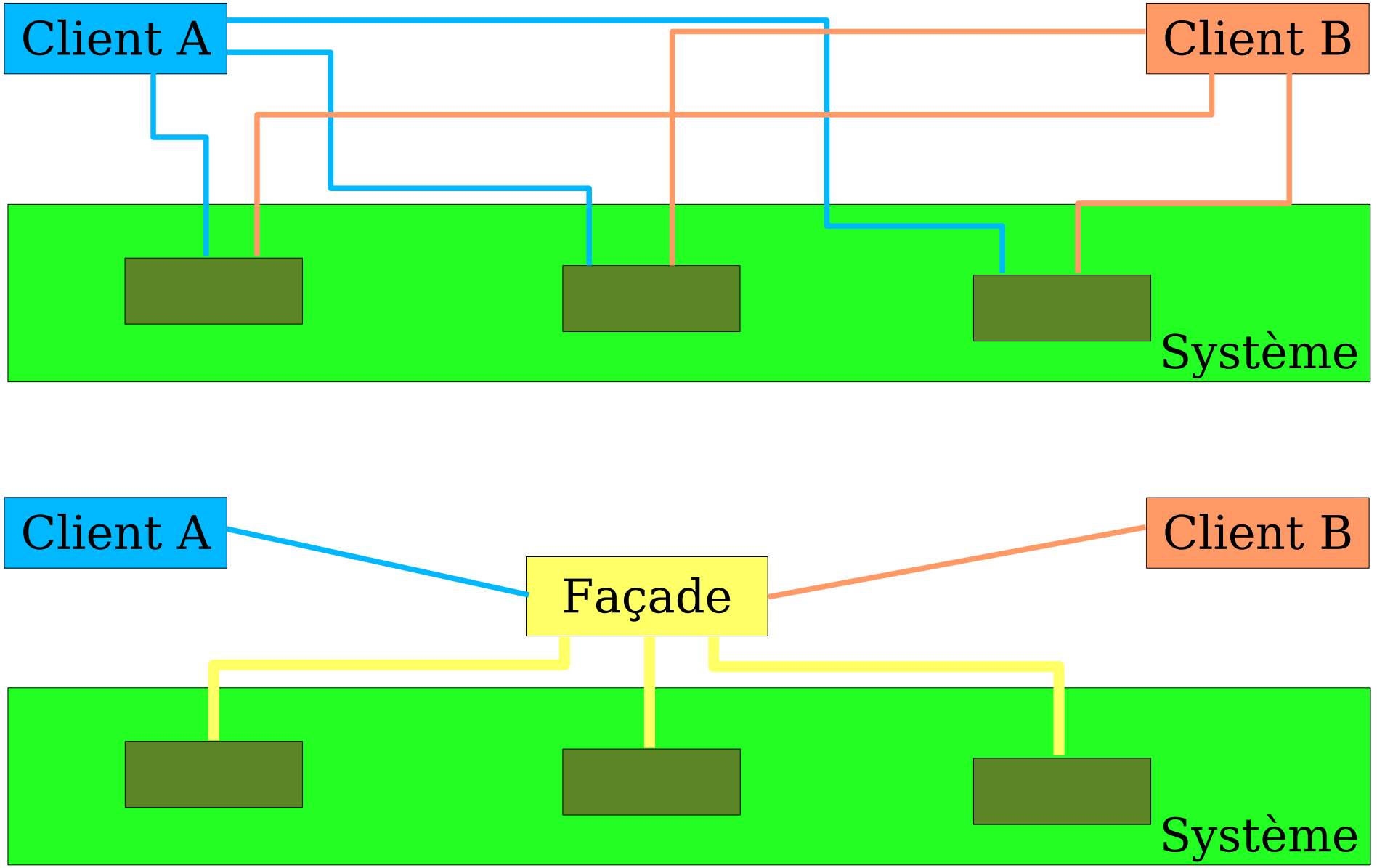
(d'après A. Beugnard, ENST Bretagne)

	Créateurs	Structuraux	Comportementaux
Classe	Fabrication	Adaptateur (class)	Interpréteur Patron de méthode
Objet	Fabrique abstraite Constructeur Prototype Singleton	Adaptateur (objet) Pont Composite Decorateur Facade Poids mouche Proxy	Chaîne de responsabilité Commande Iterateur Mediateur Memento Observateur Etat Stratégie Visiteur

Pattern Façade (1)

- Objectif
  - simplifier l'utilisation d'un système existant
  - avoir sa propre interface
- Problème
  - besoin limité à un sous-ensemble d'un système complexe
  - manipuler un système uniquement d'une façon spécifique
- Solution
  - façade fournit une nouvelle interface au système existant

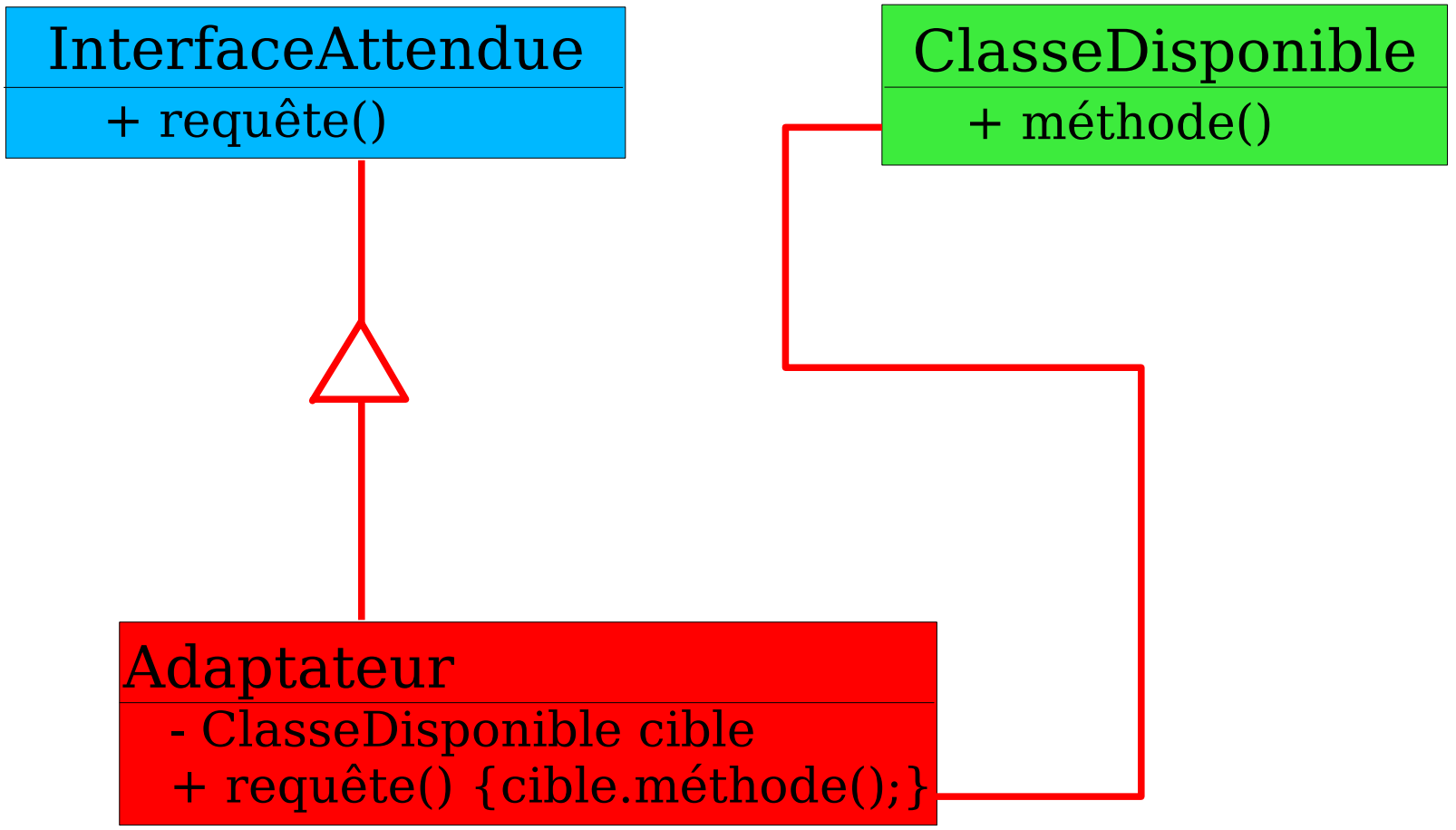
Pattern Façade (2)



Pattern Adaptateur (1)

- Objectif
  - Faire correspondre à une interface donnée un objet existant que l'on ne contrôle pas
- Problème
  - Le système a les bonnes données et les bonnes méthodes mais une mauvaise interface
- Solution
  - Adaptateur = encapsulateur avec l'interface voulue

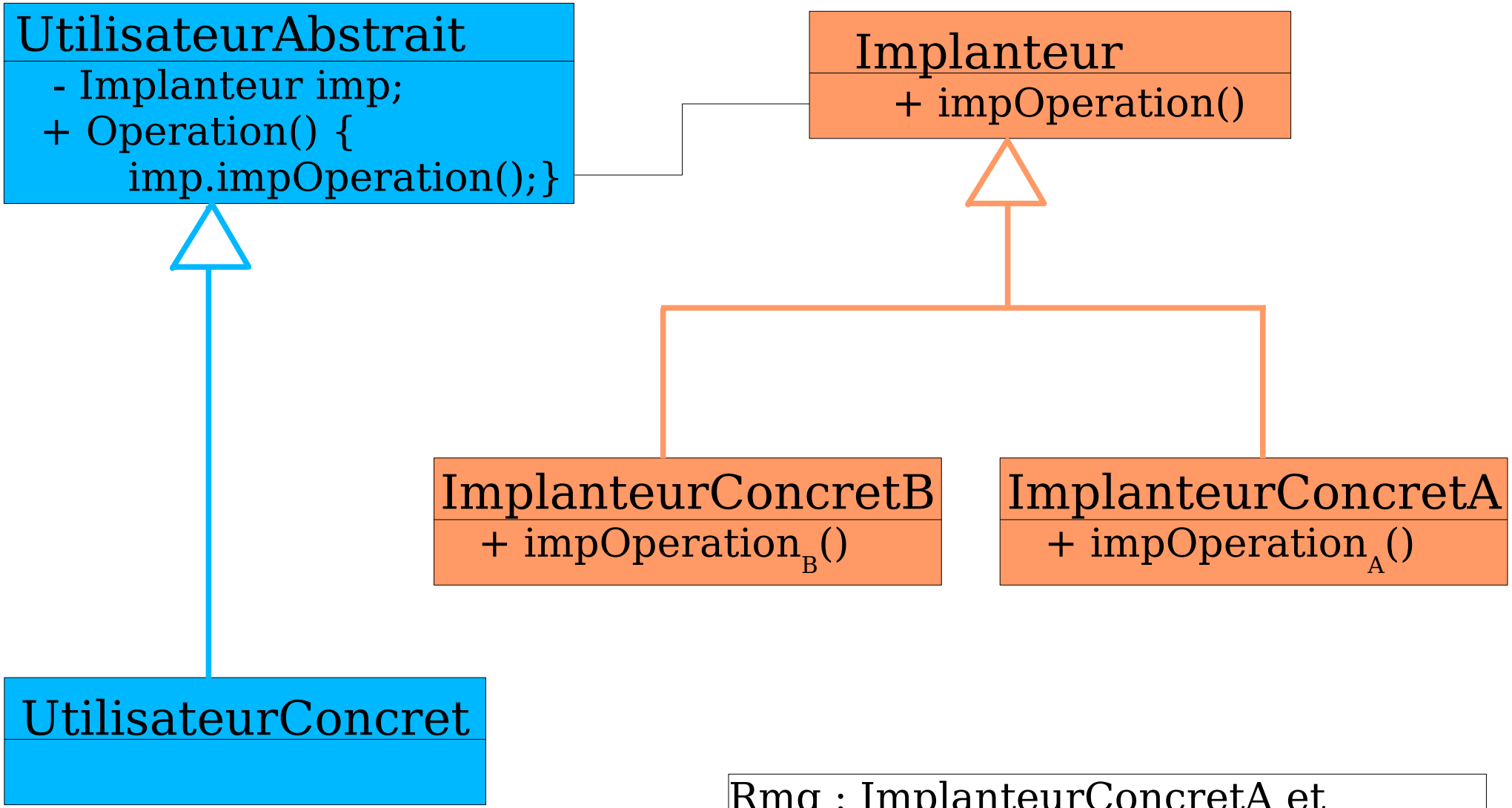
Pattern Adaptateur (2)



Pattern Pont (1)

- Objectif
  - Découpler un jeu d'implantations du jeu d'objets qui l'utilise
- Problème
  - Les dérivations d'une classe abstraite doivent utiliser plusieurs implantations sans faire exploser le nombre de classes
- Solution
  - Définir une interface pour toutes les implantations et la faire utiliser par les dérivations de la classe abstraite

Pattern Pont (2)

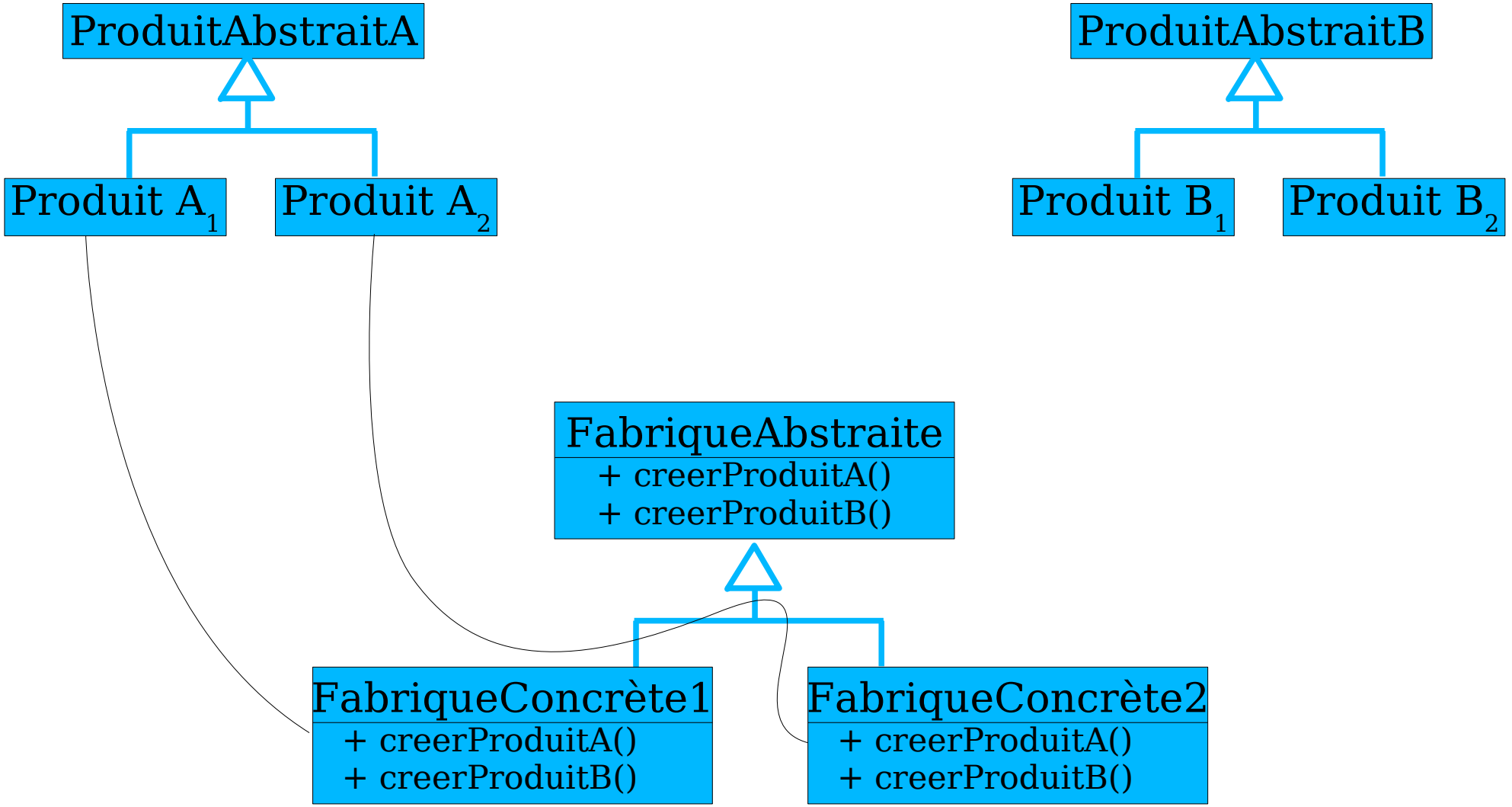


Rmq : ImplanteurConcretA et ImplanteurConcretB sont souvent des adaptateurs de classes existantes

Pattern Fabrique Abstraite (1)

- Objectif
  - Utiliser des familles ou jeux d'objets pour des clients/cas donnés
- Problème
  - Des familles d'objets associés doivent être instanciés
- Solution
  - Coordonner la création de familles d'objets
  - Extraire les règles d'instanciation de l'objet client

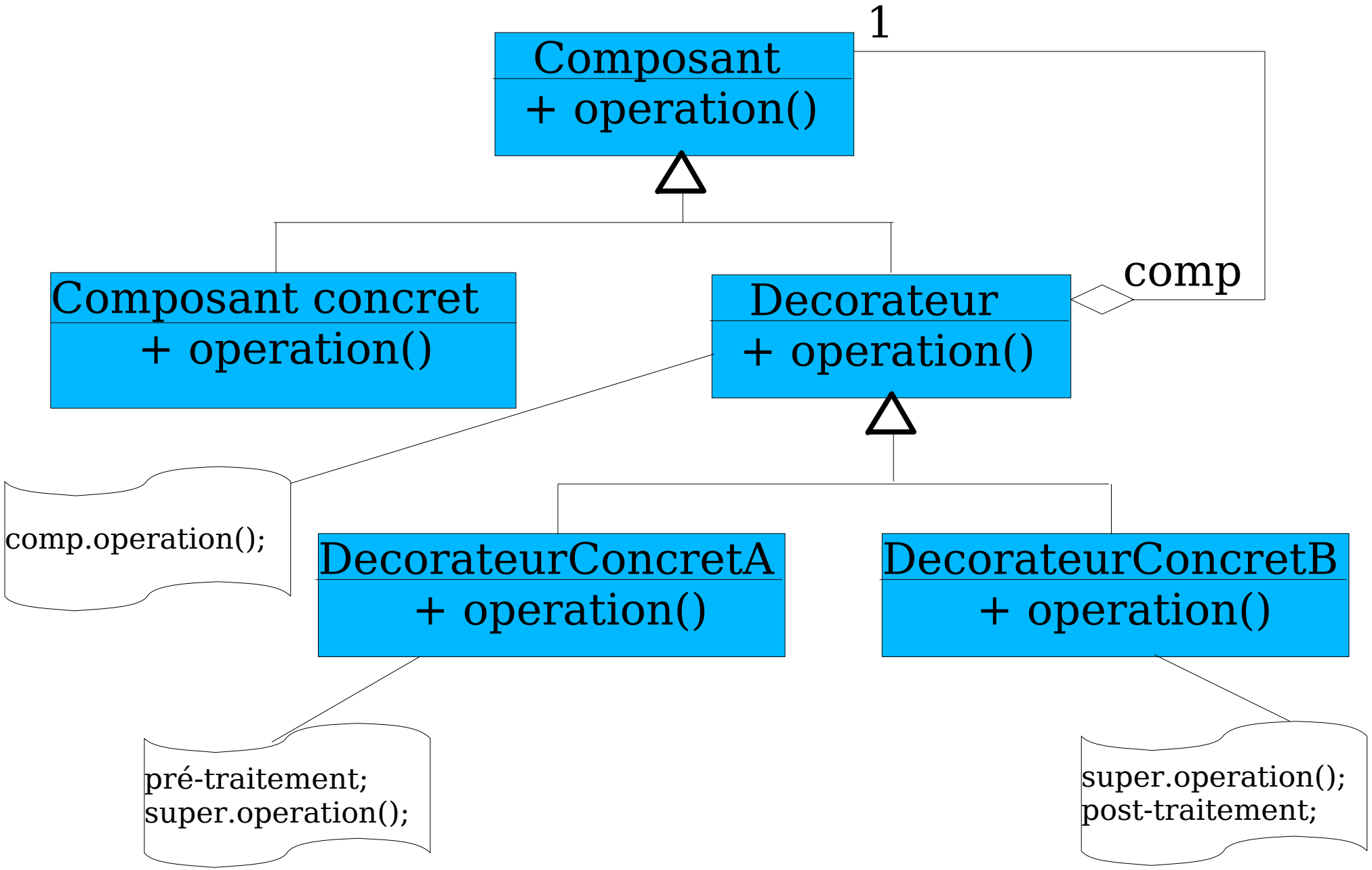
Pattern Fabrique Abstraite (2)



Pattern Décorateur (1)

- Objectif
  - Associer dynamiquement des responsabilités supplémentaires à un objet
- Problème
  - Objet à utiliser possède les fonctions de base recherchées
  - Nécessité de pouvoir ajouter dynamiquement des fonctionnalités supplémentaires s'appliquant avant ou après
- Solution
  - Permettre l'extension de la fonctionnalité d'un objet sans recourir à des sous-classes

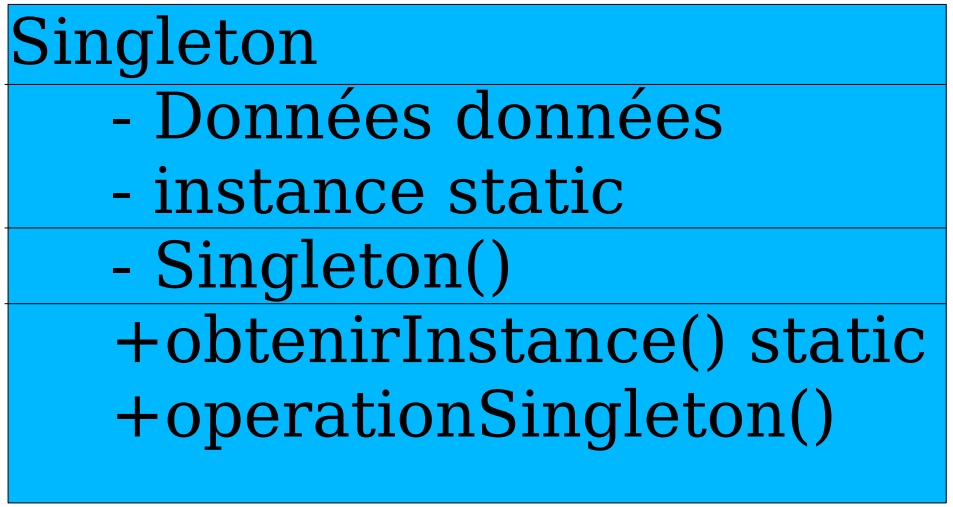
Pattern Décorateur (2)



Pattern Singleton (1)

- Objectif
  - Garantir qu'une classe n'est pas instanciée plus d'une fois
- Problème
  - Deux objets doivent se référer à la même chose
  - Être sûr qu'il n'en existe qu'une instance
- Solution
  - S'assurer qu'il n'existe qu'une instance

Pattern Singleton (2)



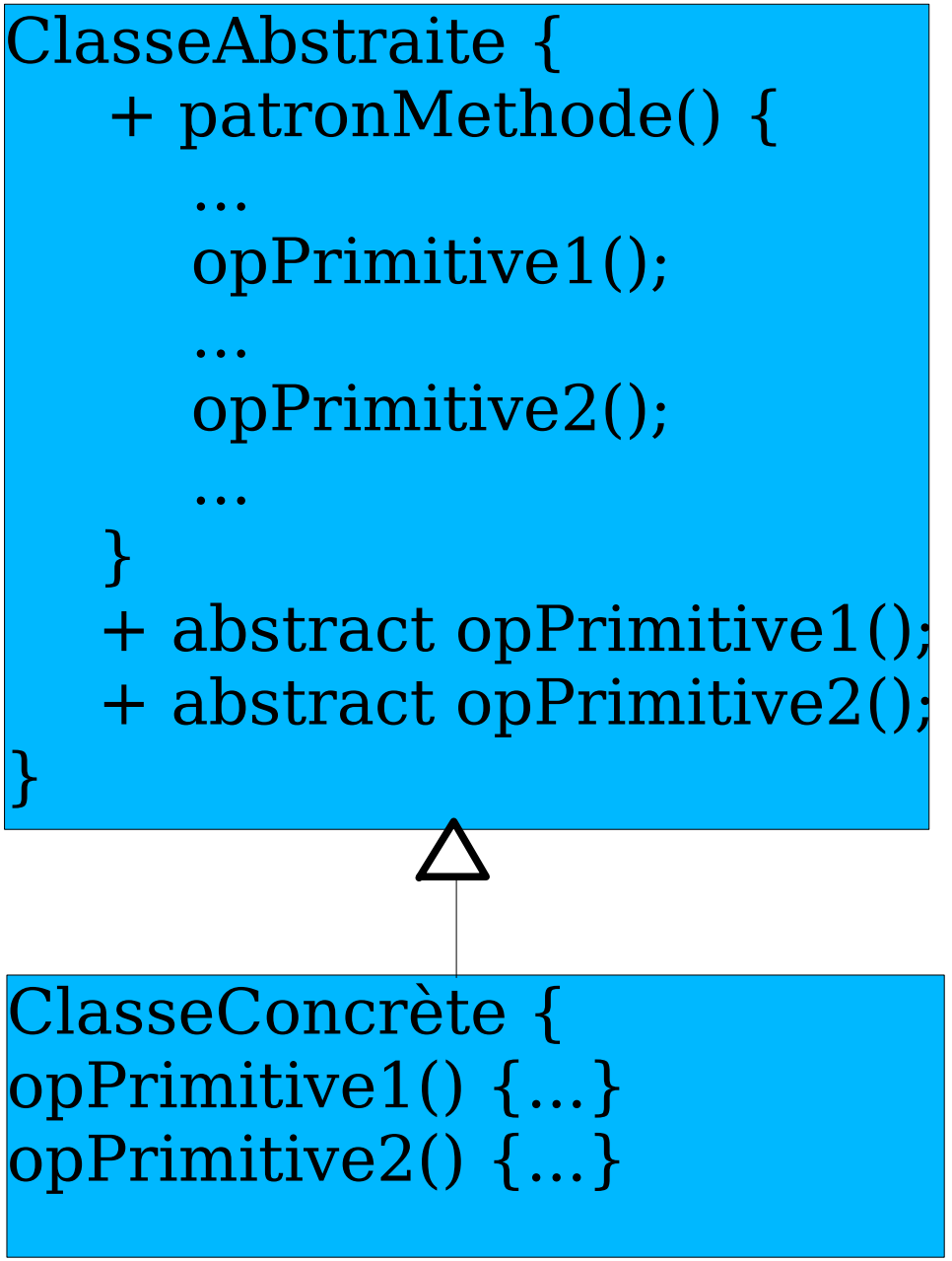
En Java :

```
public class Singleton {
  private Données données;
  private static Singleton instance = null;
  private Singleton() {
     ...
  }
  public static Singleton getInstance() {
    if (instance == null) {
      {instance = new Singleton();}
    return instance;
  }
}
```

Pattern Patron de méthode (1)

- Objectif
  - Définir une structure d'algorithme
  - Déléguer la définition de certaines étapes aux sous-classes
- Problème
  - Suivre une procédure cohérente à ce niveau de détail donné
  - Implantation différée à un niveau plus détaillé
- Solution
  - Permettre la définition de sous-étapes variant tout en concervant un processus de base cohérent

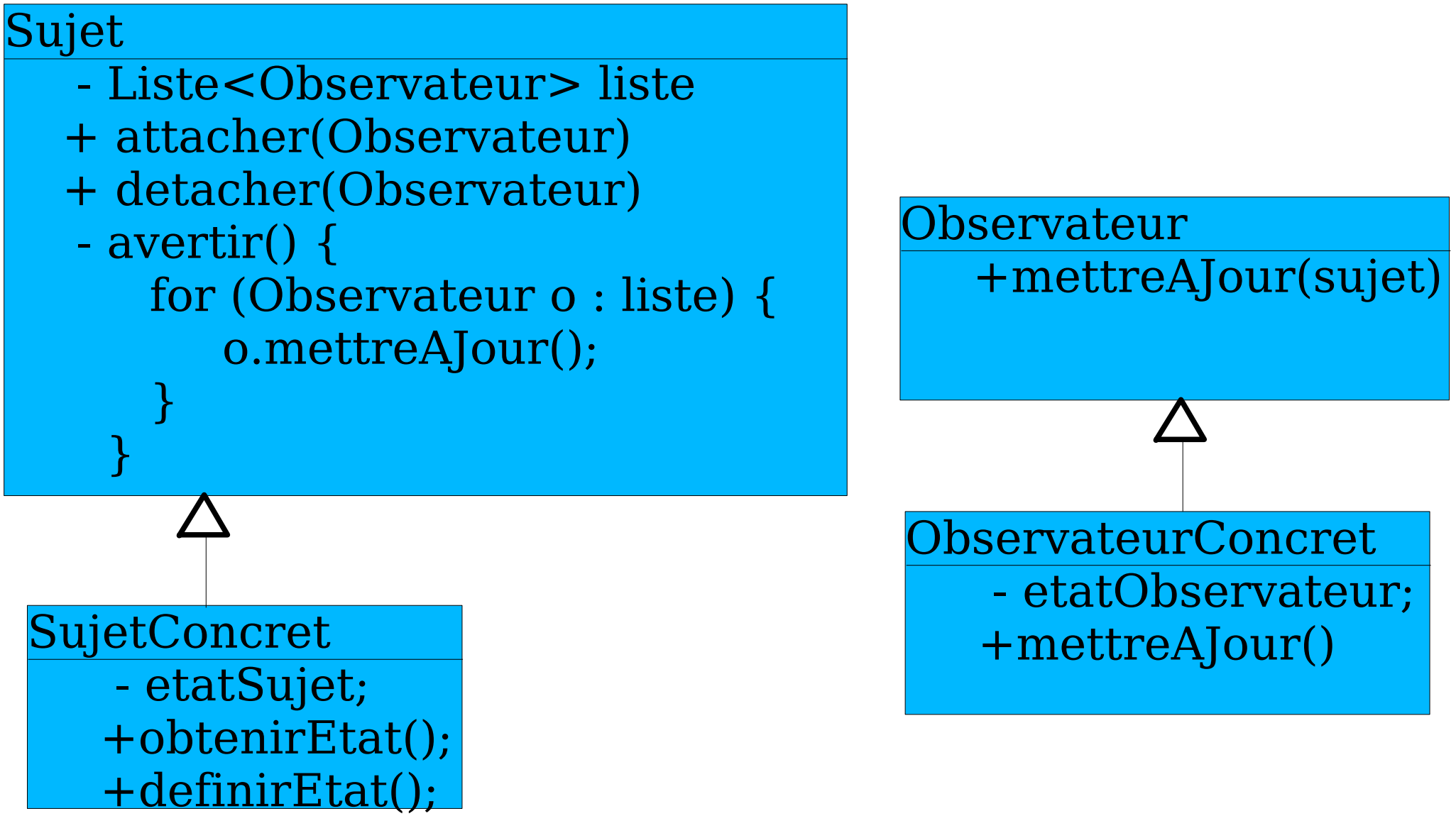
Pattern Patron de méthode (2)



Pattern Observateur (1)

- Objectif
  - Etablir une dépendance 1-N entre des objets pour que ceux qui dépendent d'un objet modifié (ou créé) soient avertis du changement d'état et mis à jour automatiquement
- Problème
  - Avertir une liste variable d'objets qu'un événement a lieu
- Solution
  - Les objets observateurs délèguent la responsabilité de contrôle d'un événement à un objet central, le *sujet*.

Pattern Observateur (2)



Pattern Stratégie (1)

- Objectif
  - Utiliser différents algorithmes ou règles métier, conceptuellement identiques, suivant le contexte
- Problème
  - Sélection d'un algorithme dépend du client ou des données
- Solution
  - Séparer la sélection de l'algorithme de son implantation

Pattern Stratégie (2)