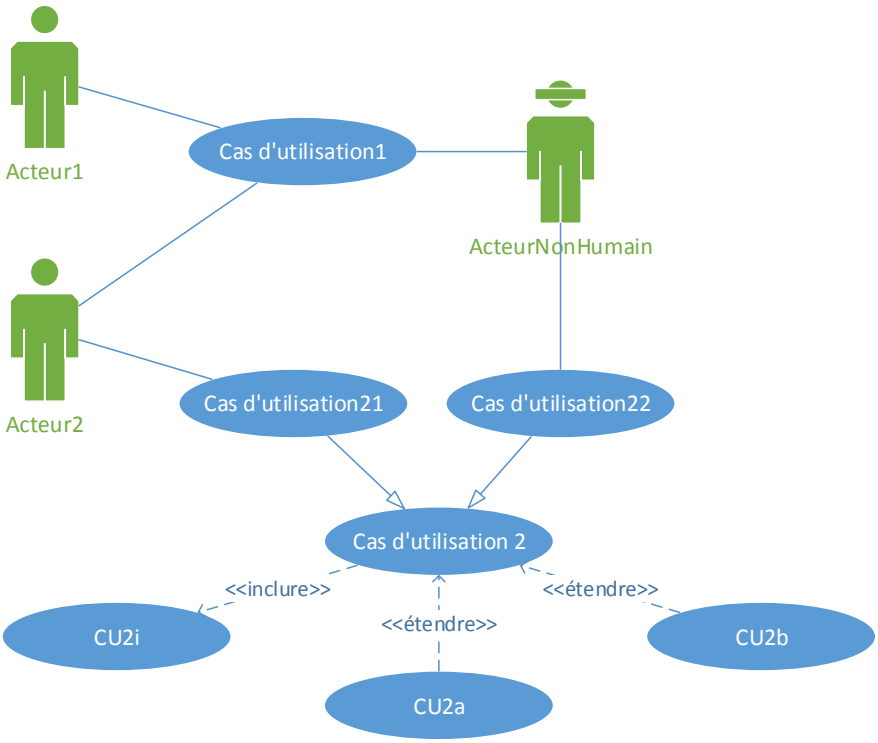
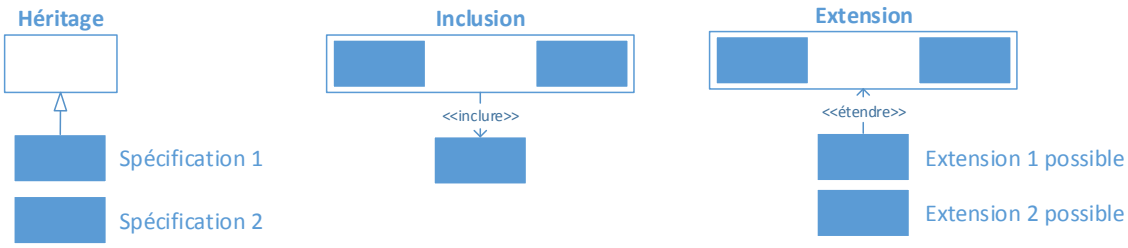
Résumé d'UML

ASI3 – Fiche

I. *Diagramme de cas d'utilisation*





II. Fiche de Cockburn (Description d'un cas d'utilisation)

- **Titre :** …
- **Résumé :** …
- **Acteurs :** Act1 (A), Act2 (B), …
- **Motivation :** A veut…
- **Pré-condition(s) :** …
- **Post-condition(s) :** …
- **Exceptions :**
  - Exception E1 : *[Titre]*
    [Actions]
  - …
- **Remarques ergonomiques (éventuellement) :** …
- **Contraintes non fonctionnelles (éventuellement) :** …
- **Scénario nominal :** *[Enchainement de cockburn / DAC / DSS]*

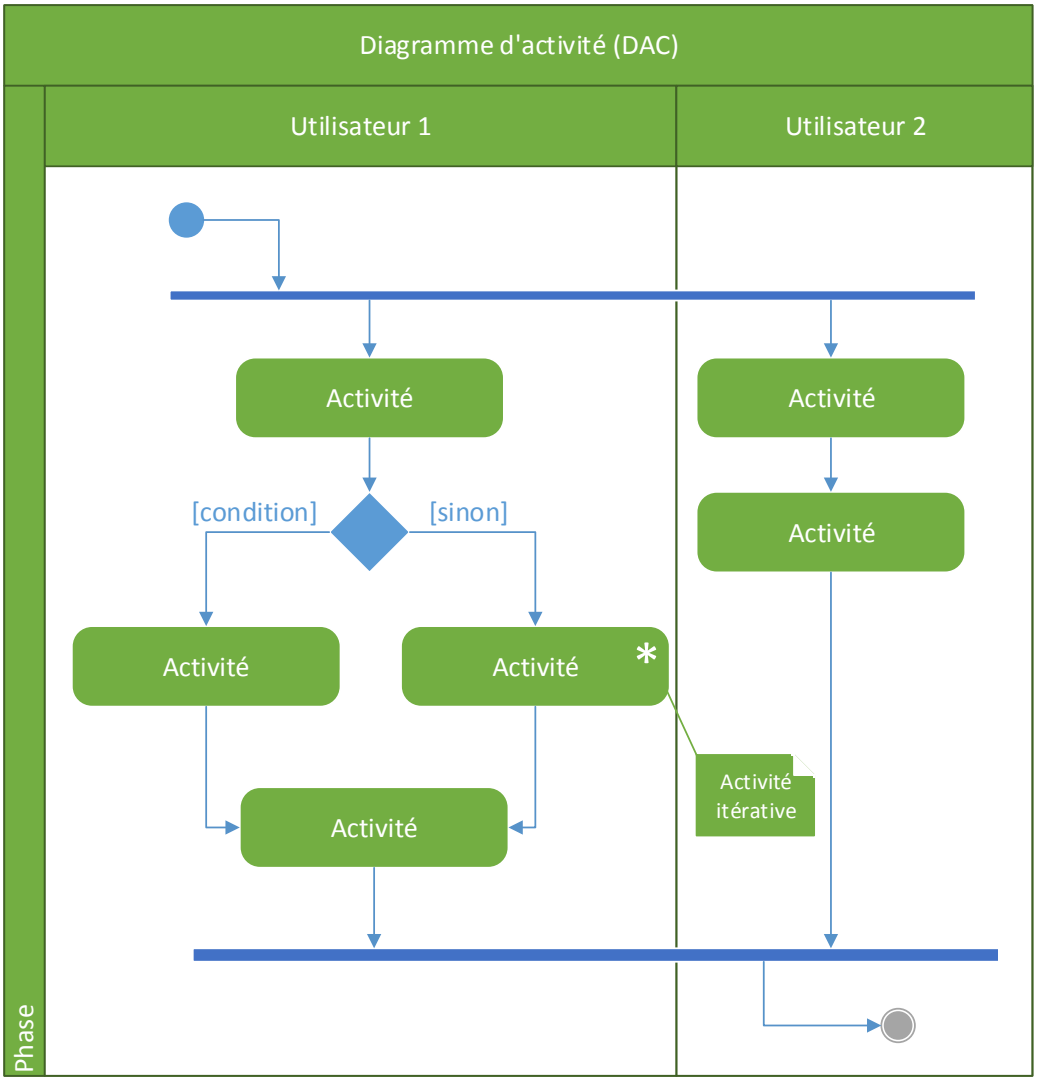
III. Enchainement de Cockburn

- Action de départ : …

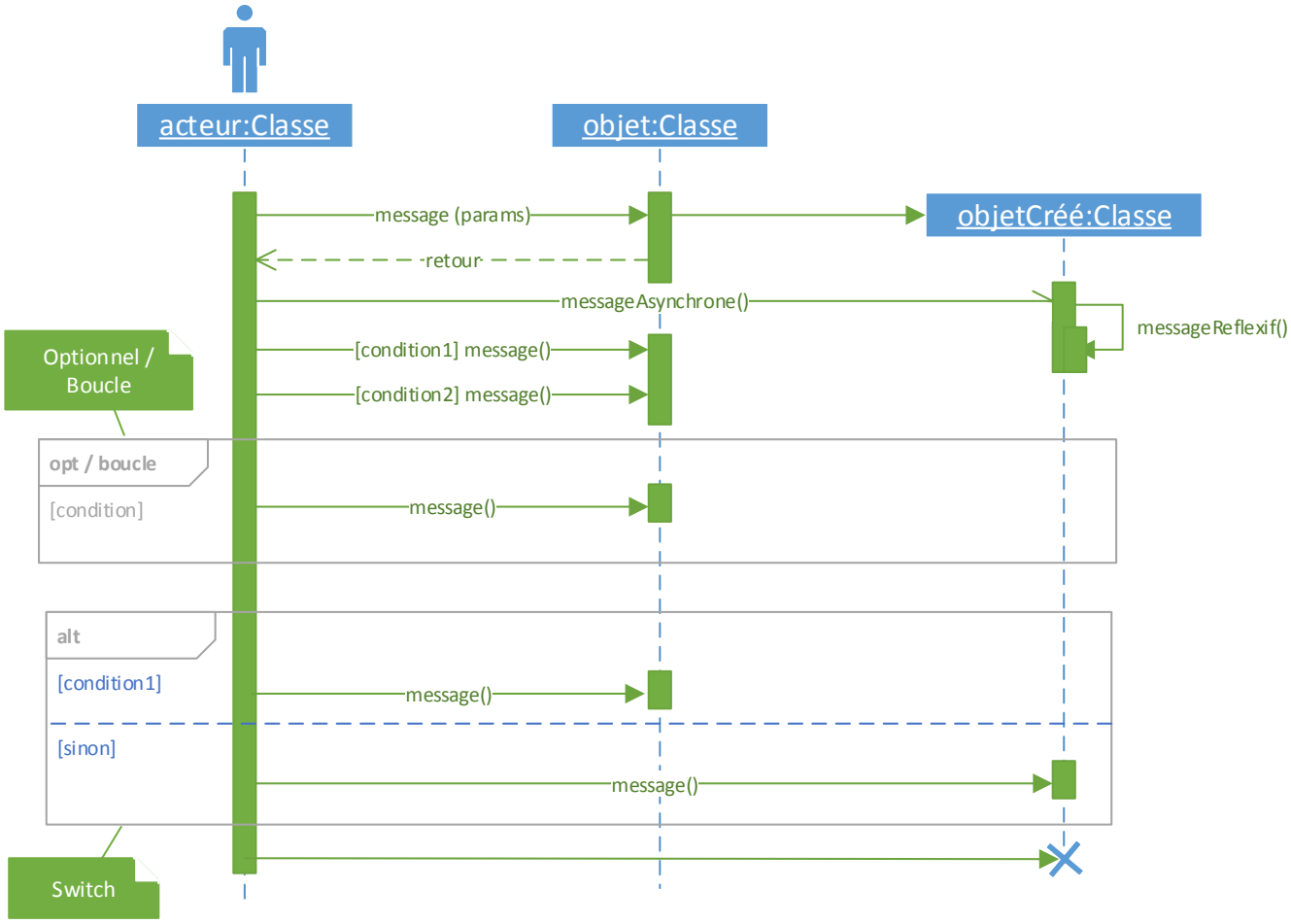
Action acteur	Action système
1. (A) fait …	
2. (B) fait …	3. Le système …
4. …	

- Action de fin : …

IV. Diagramme d'activité

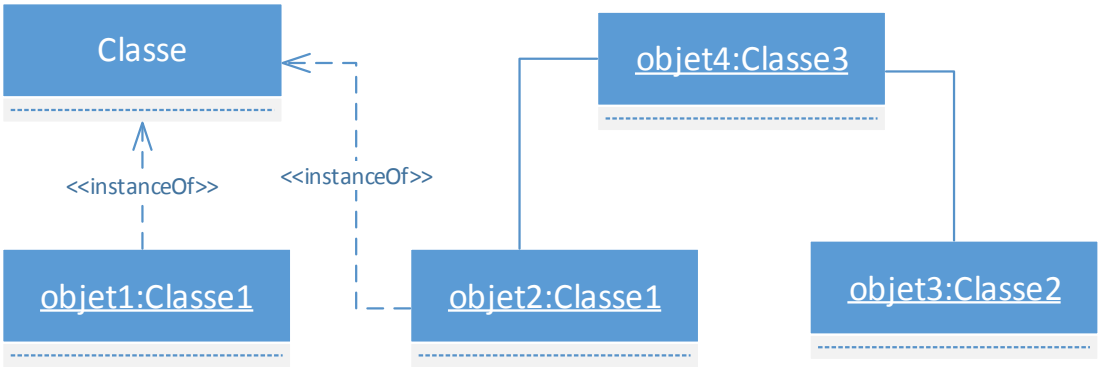


V. Diagramme de séquence



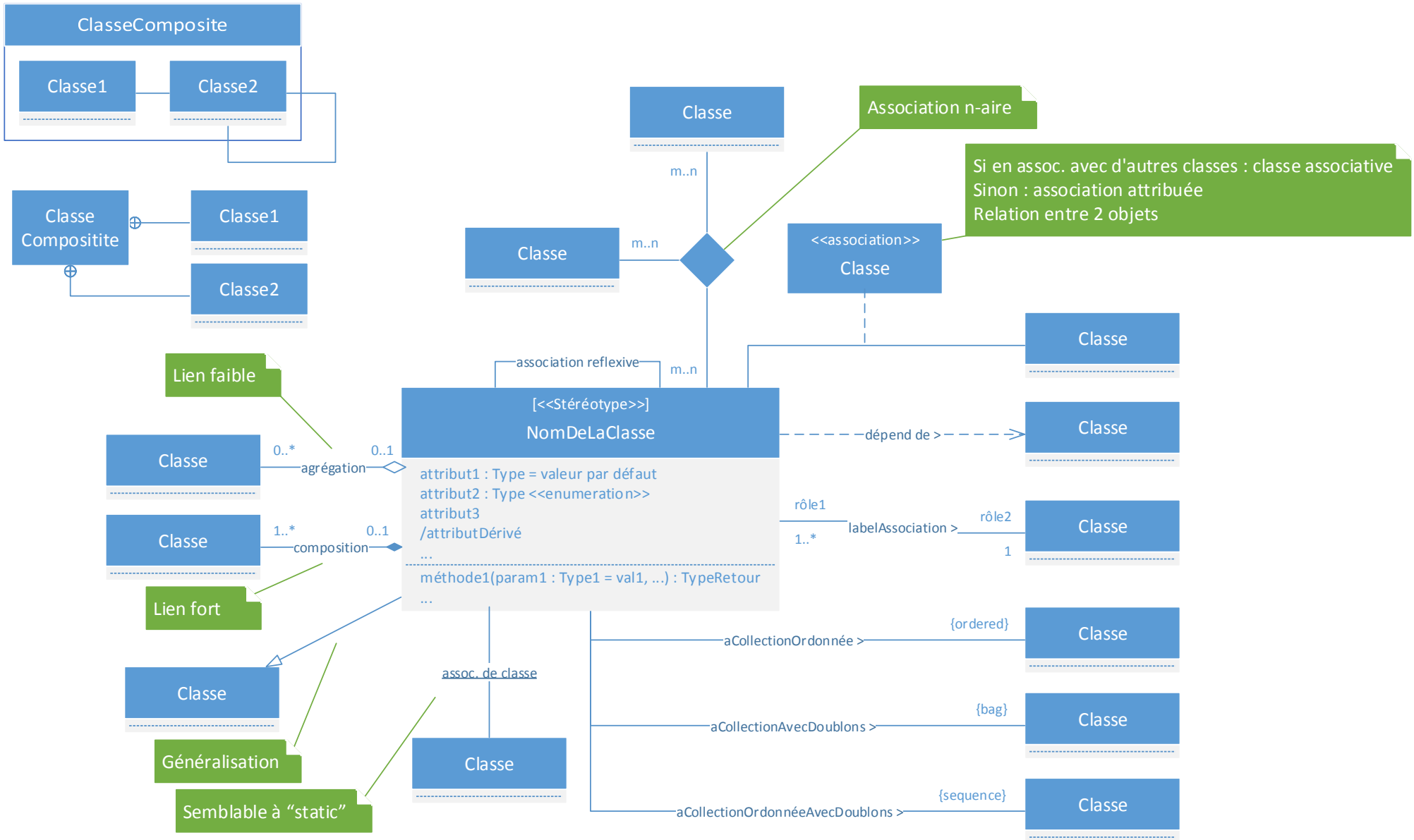
Pour un cas d'utilisation, on fait un diagramme de séquence système.
Il n'y a que l'acteur et un objet « système ».

VI. Diagramme Objet

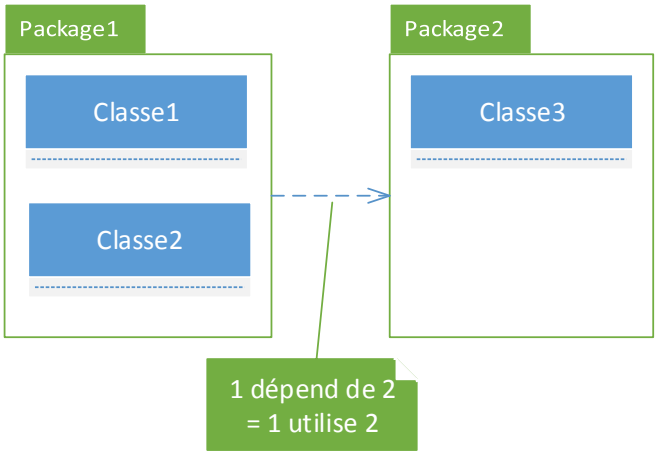


(Proche du diagramme de classes)

VII. Diagramme de classes



VIII. Diagramme de packages



On peut stéréotyper les packages <<category>> si on veut désigner une catégorie

IX. Dépendances

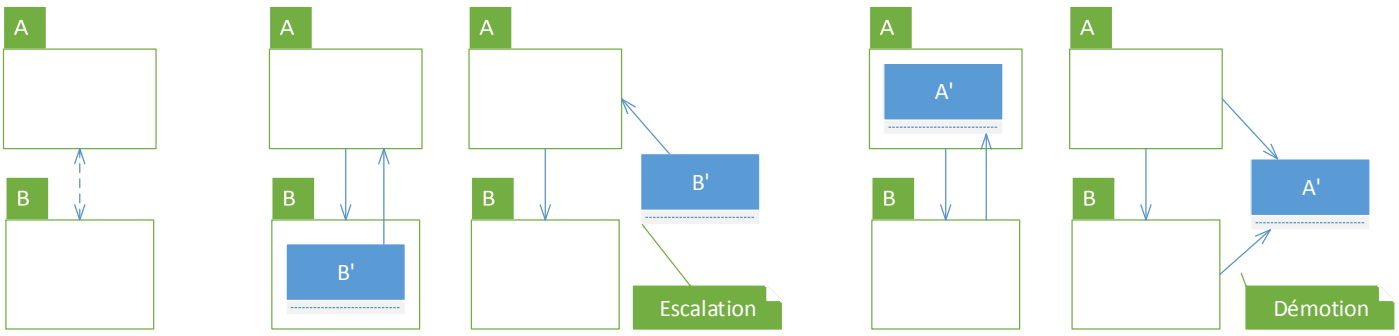
Association entre classes ⇔ Dépendance entre packages des classes, même navigabilité

On veut des dépendances unidirectionnelles non-cycliques.

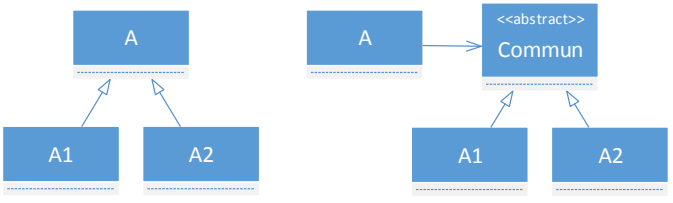
- **Dépendance conceptuelle :** association, généralisation
- **Dépendance opérationnelle (publique) :** classe utilisée en paramètre dans une méthode
- **Dépendance contextuelle (privée) :** classe utilisée dans le corps de la méthode

En créant des dépendances unidirectionnelles, on choisit qu'une classe ait connaissance de l'autre.

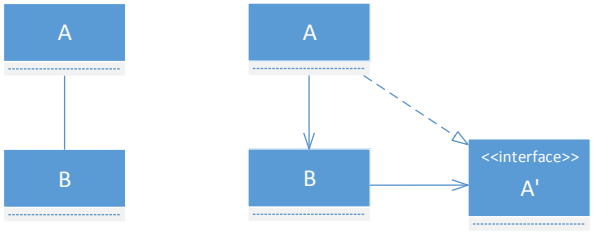
Méthodes pour casser une dépendance bidirectionnelle :



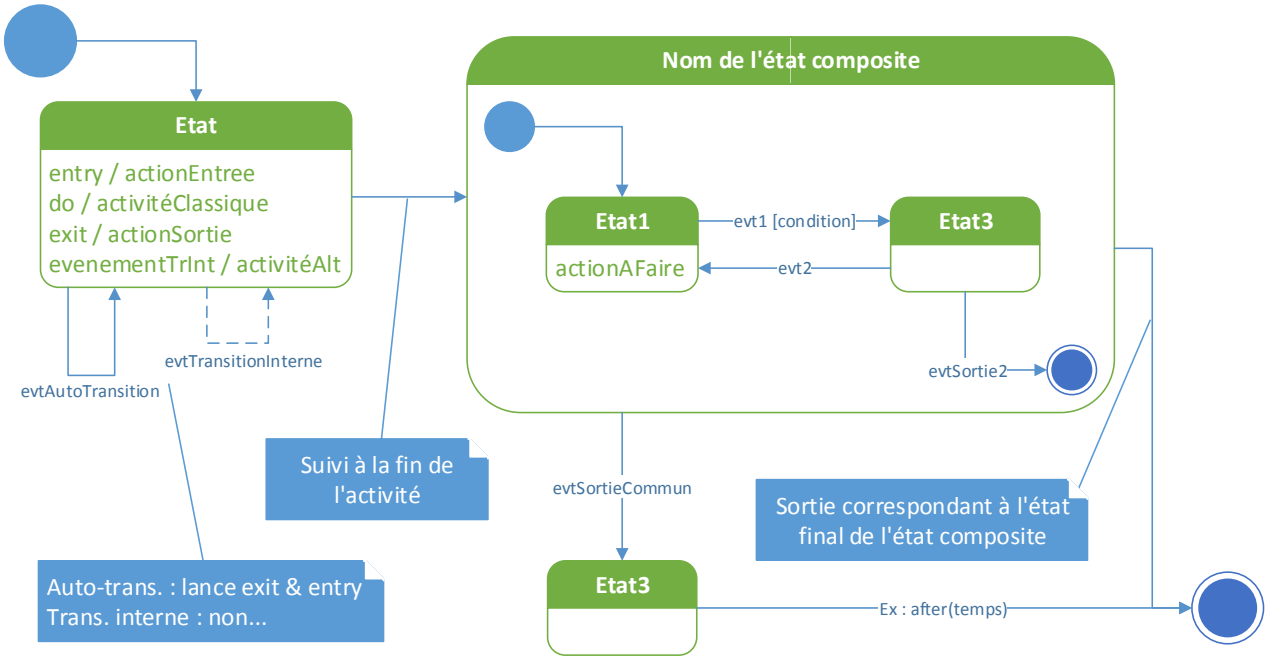
Héritage → Composition :



Association abstraite :



X. Diagramme d'états



- Action : Non interruptible
- Activité : Interruptible
- Evènements :
  - **Signaux :** Asynchrone
  - **Message :** Appel d'opération, synchrone
  - **Temporel :** after(duree)
  - **Modifiant :** when(condition booléenne)
-

XI. Patterns d'analyse

1. Pattern <<history>> : à un instant

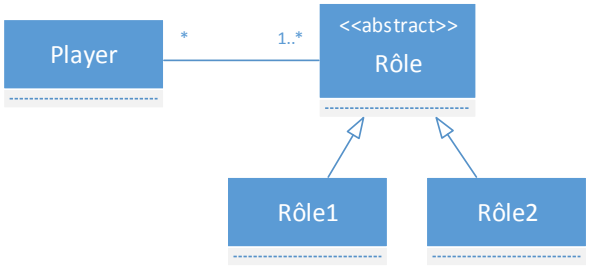
2. Anti-pattern : Actions vs association

Une association ne peut être une action. Pour des actions, créer des opérations et si besoin de classe pour qualifier l'action. (Ex : qualification d'un prêt dans une bilio)

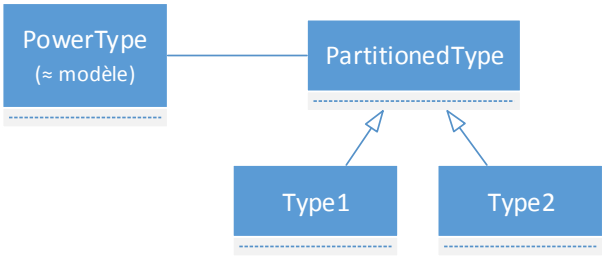
3. Anti-pattern : Non transmutabilité de l'objet

Un objet ne peut changer de classe au cours de sa vie, même pas se spécialiser. Solution : pattern player-rôle.

4. Pattern Player-Role



5. Pattern Powertype

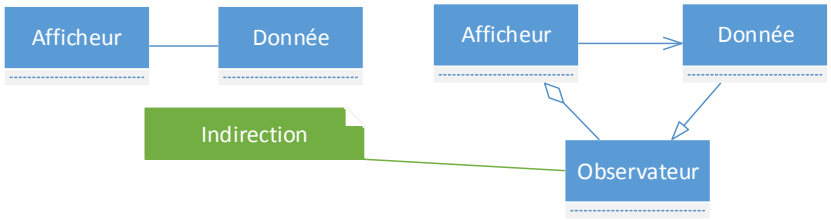


6. Pattern Abstraction-Occurrence

XII. Patterns GRASP

1. Patterns de conception

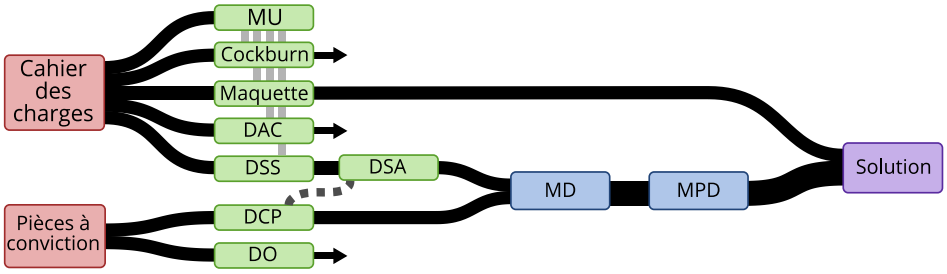
- **Expert :** Détient l'information
- **Créateur :** Construction d'objets *(B crée A : A ∈ B / B utilise A / B a les données d'init de A)*
- **Contrôleur :** Gestion des évènements d'entrées *(Classe façade)*
- **Polymorphisme :** Comportement avec variantes
- **Fabrication pure :** Classe artificielle pour respecter les patterns
- **Indirection :** Eviter le couplage de classes *(Objet intermédiaire)*
- **Protection des variations :** Limiter l'impact des variations *(Interfaces stables, limiter les dépendances)*



2. Patterns d'évaluation

- **Faible couplage :** Limiter les dépendances *(couplage = nombre de dep. d'autres classes)*
- **Forte cohésion :** Limiter la complexité *(opérations cohérentes)*

XIII. Méthodes d'analyse et de conception



1. Grille de Levesque

Permet de trouver le besoin

2. MU : Modèle d'usage (diagramme de CU)

Liste les cas d'utilisation

3. DO & DCP : Analyse textuelle (méthode Abbot)

Partie du texte	Elément de modélisation	Exemple
Nom propre	objet	Jim
Nom commun	classe	Jouet, Poupée
Verbe actif	méthode	Achète, recommande
Verbe être	héritage	Est un, un genre de
Verbe avoir	agrégation	A, possède
Verbe modal	contraintes	Doit trouver
Adjectif	attribut	Agé de 2 ans
Verbe transitif	méthode	Entre, fournir
Verbe intransitif	exception ou evt	Dépend de

On peut réaliser un DO (diagramme objet) ou un DCP (diagramme de classes participantes).

4. DSA : Transformée de Jacobson d'analyse

DSS + DCP → DSA (Diagramme de séquence d'analyse)

On remplace le système par les objets qui sont derrière.

5. MD : Fusion et croisement de Jacobson

- **Fusion :** DCP1 + … + DCPn → MD (Modèle du domaine *(diag. de classes)*)
- **Croisement de Jacobson :** DCPx + DSAy → MD_{init}

6. DSC : Transformée de Jacobson de conception

 **<<Bondary>>** Visualise ou transmet des données à l'acteur

 **<<Control>>** Cas d'utilisation / Session / Profil utilisateur / Système / Classes du domaine

 **<<Entity>>** Classes du modèle du domaine